LUIS DIAS PEREIRA

VISUALIZAÇÃO DE CAMADAS DE INFORMAÇÃO EM AMBIENTE INTERNET

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática, Curso de Mestrado em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Carlos A. P. de Carvalho

CURITIBA
2001

UFPR
Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática do aluno ***Luis Dias Pereira***, avaliamos o trabalho intitulado "***Visualização de Camadas de Informação em Ambiente Internet***", cuja defesa foi realizada no dia 05 de março de 2001. Após a avaliação, decidimos pela aprovação do candidato.

Curitiba, 05 de março de 2001.

Prof. Dr. Carlos Alberto Ricanço de Carvalho
Presidente - Orientador

Prof. Dr. Luiz Alberto Vieira Dias
Membro Externo - UNIVAP

Prof. Dr. Hélio Pedrini
DINF/UFPR

"Um teólogo perguntou ao supercomputador mais poderoso se Deus existe. O computador respondeu que não era suficientemente potente para saber e pediu para ser conectado a todos os demais supercomputadores do mundo. Não obstante, a potência atingida foi insuficiente. Assim, o computador foi ligado a todos os mainframes do mundo, a todos os minicomputadores e a todos os computadores pessoais. Finalmente, acabou sendo conectado a todos os computadores de bordo, a todas as microondas, aos videocassetes, aos relógios digitais, e assim por diante. O teólogo perguntou pela última vez se Deus existe. E o computador respondeu: ***agora existe!"***

Randall L. Tobias (ex-vice-presidente da AT&T).

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

1 INTRODUÇÃO

1.1 O AMBIENTE INTERNET

Segundo pesquisa da Nua Internet Surveys de novembro de 2000, aproximadamente 407 milhões de usuários no mundo estão conectados à Internet. No Brasil a estimativa chega a 9,84 milhões de usuários, o que corresponde a aproximadamente 5,7% da população (NUA Internet Surveys, 2000).

O IBOPE estimou, baseado em uma pesquisa realizada entre 24 de agosto e 6 de setembro de 2000, que aproximadamente 4,7 milhões de consumidores teriam interesse em acessar a Internet, tornando-se usuários dentro de um período de seis meses (IBOPE, 2000). Conforme dados apresentados anteriormente, esta expectativa de crescimento corresponde a aproximadamente 50% da quantidade estimada de usuários existentes até novembro de 2000.

Considerando a velocidade fenomenal com que novas tecnologias e funcionalidades ficam disponíveis e a crescente taxa de usuários, a Internet promete ser a revolução mais importante desde a introdução do computador pessoal (BEN-NATAN, 1997, p. xiii). Hoje, não é mais uma promessa e sim uma realidade.

Dado ao grande crescimento no volume de acessos, informações disponibilizadas e a melhoria na qualidade de transmissão de dados no ambiente Internet, torna-se necessário disponibilizar mecanismos e conhecimentos que busquem uma melhor integração e distribuição da informação.

A Internet é um meio que integra diferentes plataformas, tanto de *software* como de *hardware*. Neste contexto, abre-se a possibilidade de criar sistemas que executam dentro desse novo meio, obtendo-se proveito da portabilidade oferecida e da possibilidade de acessar informações, independente da posição geográfica em que elas se encontrem.

1.2 SIG E AS CAMADAS DE INFORMAÇÃO

Sistemas de informação geográfica (SIG) representam um papel importante com relação à eficiência no acesso às informações, na tomada de decisões e na avaliação de custos. São utilizados para analisar e mostrar grandes conjuntos de

dados geograficamente referenciados e facilitam o desenvolvimento de modelos mais fidedignos (DARBAR, POWERS, NJUGUNA, 1997).

1.2.1 SIG como uma Tecnologia Integradora

O SIG vem emergindo como uma poderosa tecnologia porque permite integrar dados e métodos, visando apoiar as formas tradicionais de análise geográfica, tais como análise por sobreposição de mapas, possibilitando buscar e analisar uma grande quantidade de dados (FOOTE, LYNCH, 1997). O desenvolvimento do SIG tem se baseado em inovações que ocorreram em disciplinas distintas: Geografia, Cartografia, Fotogrametria, Sensoriamento Remoto, Topografia, Geodésia, Engenharia Civil, Estatística, Ciência da Computação, Pesquisa Operacional, Inteligência Artificial, Demografia, e muitos outros ramos das Ciências Sociais, Naturais e Engenharias.

1.2.2 Considerações sobre SIG

FOOTE e LYNCH (FOOTE, LYNCH, 1997) fazem três importantes observações com relação aos Sistemas de Informação Geográfica:

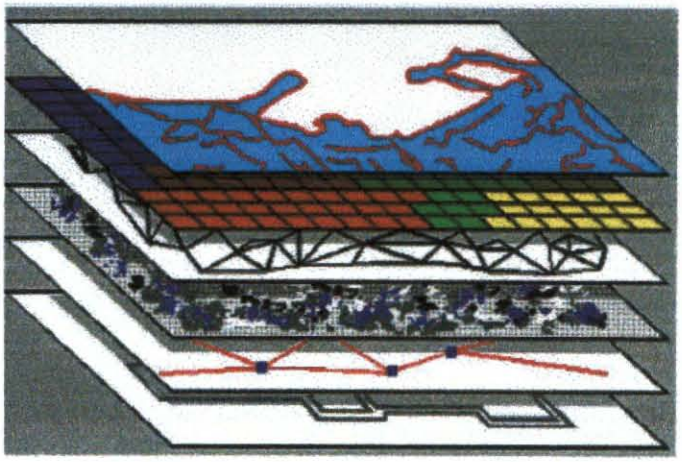
1) toda a informação em um SIG é vinculada a um sistema de referência espacial, usando geo-referência como o meio primário de armazenar e acessar a informação;
2) SIG é um integrador de tecnologias;
3) SIG's servem para tomada de decisão.

Existem vários tipos de SIG, cada um apresentando propósitos distintos e servindo a diferentes tipos de tomada de decisão, entre muitos se pode citar: AGIS (Sistema de Informação Geográfica Automatizada), Sistemas de Informação Ambiental, Sistemas de Informação Referenciada Geograficamente, Sistemas de Informação Baseada em Imagem, Sistemas de Informação para Planejamento, Sistema de Manuseio de Dados Espaciais, etc.

1.2.3 A Fonte das Informações

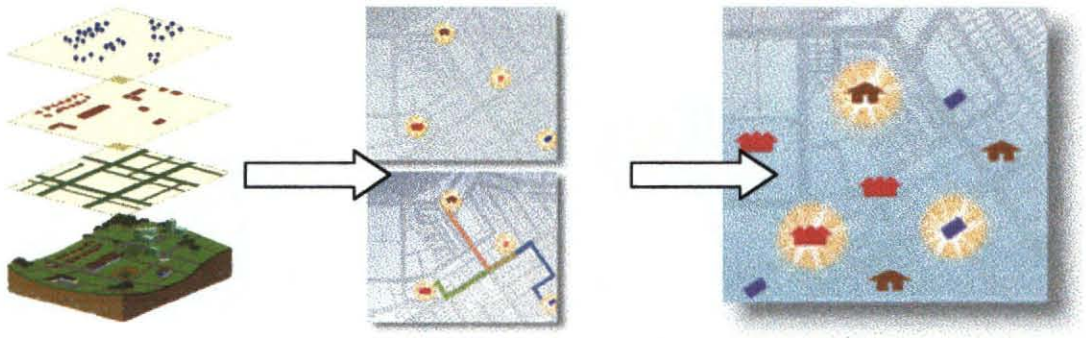
As informações em um SIG são geralmente estruturadas em temas, que fornecem informações geográficas relacionadas a um determinado assunto. Uma coleção temática de camadas de informação (também conhecidas por *layers*) pode fornecer as informações geográficas através de elementos gráficos (pontos, linhas e polígonos), criados a partir de coordenadas geográficas. As camadas de informação podem ser sobrepostas obedecendo às referências geográficas relacionadas. Na FIGURA 01 é ilustrada uma sobreposição de camadas de informação.

FIGURA 01 – CAMADAS DE INFORMAÇÃO



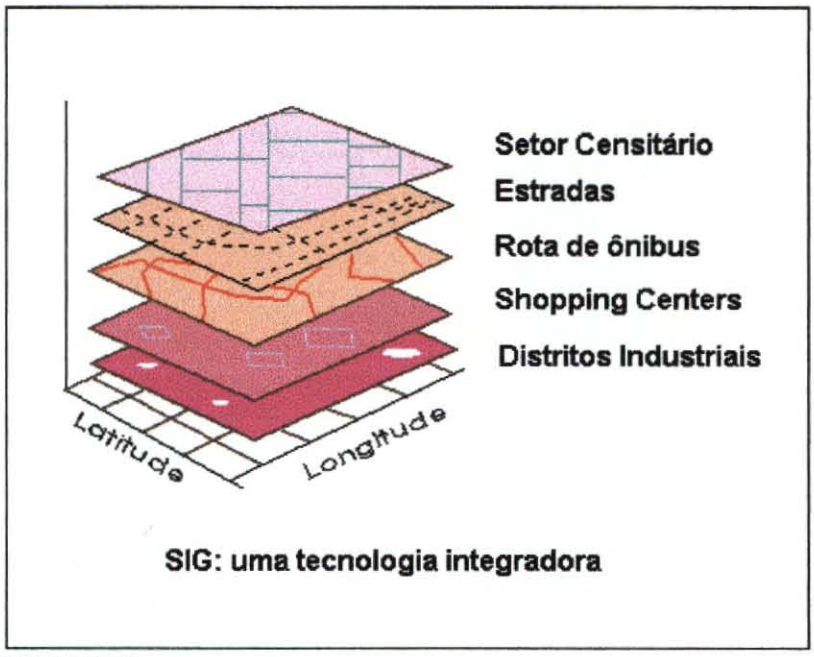
A visualização de camadas de informação é um recurso que pode ser utilizado por um SIG, com o intuito de fornecer informações geográficas a partir de elementos visualizados. As informações geográficas contêm uma referência geográfica explícita, ou seja, uma latitude e longitude ou coordenada; ou implícita, tal qual um endereço, código postal ou nome de rua. Essas referências geográficas permitem localizar características e eventos relacionados à superfície da terra, com o fim de análise e tomada de decisão. Na FIGURA 02 pode-se visualizar uma ilustração de utilização de referências geográficas.

FIGURA 02 – REFERÊNCIAS GEOGRÁFICAS



Na FIGURA 03 é ilustrado um conjunto de mapas obtidos para auxiliar o planejamento de transporte urbano. Cada um destes mapas temáticos é referenciado como uma camada. Cada camada foi cuidadosamente sobreposta de forma que toda localização é precisamente ajustada, a partir das coordenadas geográficas indicadas pela latitude e longitude, às localizações correspondentes em todos os mapas.

FIGURA 03 – PLANEJAMENTO DE TRANSPORTE



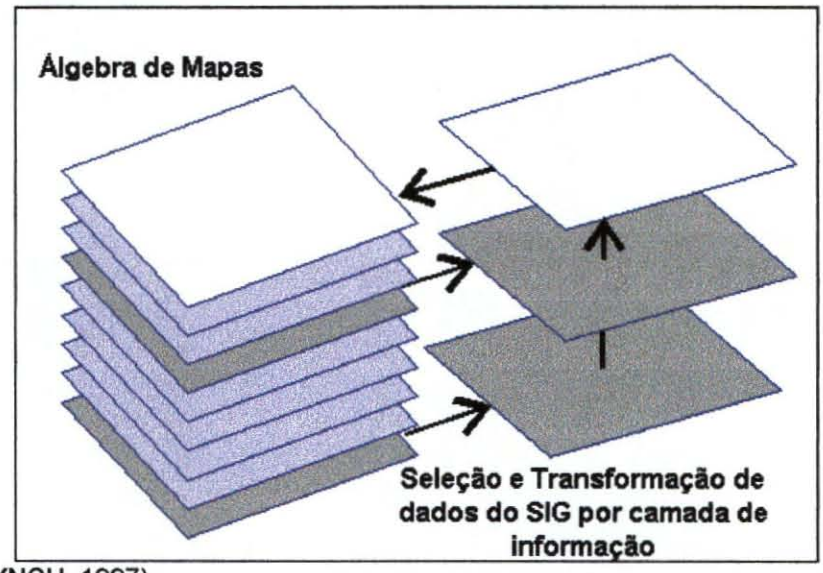
FONTE: (FOOTE, LYNCH, 1997)

1.2.4 Álgebra de Mapas

O processo de combinar e transformar informações de camadas diferentes pode ser chamado de álgebra de mapas, pois envolve soma e subtração de informação.

Nem toda análise de informação requer o uso de todas as camadas simultaneamente. Em alguns casos, um investigador usará seletivamente a informação para considerar relações entre camadas específicas (FOOTE, LYNCH, 1997).

FIGURA 04 – ÁLGEBRA DE MAPAS



FONTE: (FOOTE, LYNCH, 1997)

1.3 O PROJETO

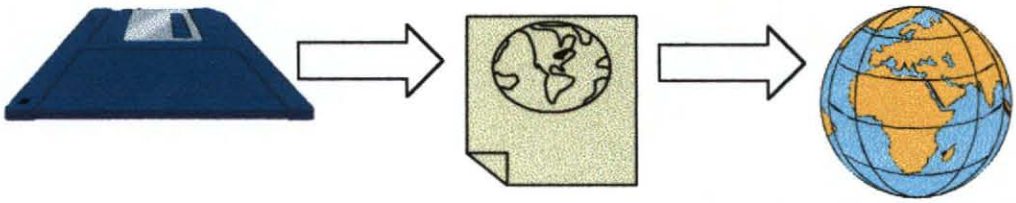
1.3.1 Objetivo Principal

O projeto tem como objetivo principal à implementação de um sistema para visualização de camadas de informação, representadas por meio de classes implementadas em linguagem de programação orientada a objetos, possibilitando o desenvolvimento de recursos e rotinas que possam manipular este tipo de objeto, e facilitando assim a sua distribuição no meio Internet.

1.3.1.1 O Objeto de Origem

As camadas de informação serão originadas de um objeto *.shp (*Shapefile*), padrão de arquivo vetorial definido pela ESRI (*Environmental Systems Research Institute*). Esse objeto será convertido para uma classe que corresponde à sua representação, e através dos recursos gráficos do sistema ele será visualizado. Na FIGURA 05 é ilustrada a seqüência das ações básicas sofridas pelo objeto.

FIGURA 05 – CONVERTENDO OBJETOS



Objeto *Shapefile* **Classe** **Visualização do objeto**

1.3.2 Objetivos Secundários

São três os objetivos secundários:

a) propor uma solução de custo reduzido e de fácil utilização para distribuição de informações no ambiente Internet;
b) criação de um ambiente que possibilite a transformação de arquivos em objetos;
c) criar base para estudo da distribuição de objetos no meio Internet.

1.3.3 Não é o Objetivo do Trabalho

O trabalho em questão não visa produzir as camadas de informação e nem os dados relacionados a elas. O objetivo principal, como já foi dito, é o de desenvolver um conjunto de ferramentas que possibilite a manipulação de objetos, representados por camadas de informação, e criar mecanismos para visualização desses objetos no meio Internet.

1.3.4 Motivação para Estudo do Caso

Os usuários de sistemas de SIG estão crescendo rapidamente da academia para a indústria e agências de governo federal, estadual e a níveis locais. Isso não significa um conjunto exaustivo de aplicações. A maioria das informações utilizada pelo governo e pela indústria tem um contexto de espaço e as aplicações são ilimitadas (DARBAR, POWERS, NJUGUNA, 1997).

Atualmente, o desenvolvimento de sistemas para SIG está reservado a um determinado número de indivíduos (corporações ou não) que detêm um relativo monopólio de mercado. Muito desse conhecimento está restrito ao meio comercial, não facilitando a popularização da tecnologia.

Este trabalho visa socializar o conhecimento de uma área específica de SIG, produzir conhecimento com relação à manipulação computacional de arquivos vetoriais, apresentar solução para visualização dinâmica de mapas no ambiente Internet, e produzir material referencial para desenvolvimento de sistemas para esse ambiente.

1.4 METODOLOGIA DE DESENVOLVIMENTO

1.4.1 Elementos Utilizados

Basicamente serão utilizados cinco elementos para o desenvolvimento:

a) UML (*Unified Modeling Language*) como ferramenta de modelagem;
b) Arquivos vetoriais do tipo *ShapeFile* como objeto para distribuição;
c) Java 1.2 como linguagem de programação orientada a objetos;
d) Navegador Internet (*Browser*) como interface do sistema;
e) Servidor Internet como meio de distribuição.

1.4.2 Processos Relacionados

A partir dos elementos citados no item 1.4.1, descrevem-se, na seqüência, os processos relacionados:

a) utilizando-se da UML, modelar as fases de análise e projeto, produzindo o conjunto necessário de classes para implementação do sistema;
b) através de organizações governamentais ou não (universidades, empresas de cartografia e/ou geoprocessamento, etc.), adquirir um conjunto de camadas de informação, representadas por arquivos do tipo *ShapeFile*;
c) identificar a especificação técnica do objeto *Shapefile*, estabelecer os padrões de formação, mecanismos de manipulação, complexidade e limitações de utilização desse tipo de arquivo no ambiente Internet;
d) utilizando-se da linguagem de programação Java 1.2, implementar o sistema para visualização e manipulação de camadas de informação;
e) verificar compatibilidade do sistema com os navegadores;
f) disponibilizar o sistema através de um servidor Internet.

2 REVISÃO DE LITERATURA

Na revisão de literatura, relacionada à distribuição de informações geográficas no meio Internet, verificou-se a existência de três cenários, que justificam a execução deste trabalho:

CENÁRIO A — A dependência de SIG por demanda de processamento, pelo servidor responsável pela distribuição da informação, e apresentando uma estrutura pouco interativa com o usuário do sistema.

CENÁRIO B — Uma grande dependência por ferramentas de desenvolvimento de SIG que, muitas vezes, possuem um elevado valor comercial.

CENÁRIO C — O interesse por desenvolvimento de sistemas de distribuição de informação mais interativos e dinâmicos, favorecendo usuários com pequena e específica demanda de distribuição e consulta de informações geográficas.

2.1 CENÁRIO A

No documento: **Interactive Information Service Using World Wide Web Hypertext** (PUTZ, 1994), é proposta uma solução de SIG utilizando-se de conversão de Hiper Texto e transferência de imagem do tipo *.gif. A solução é criativa, porém, limita os recursos da visualização de imagens, com a utilização de arquivos do tipo raster, e demanda uma grande quantidade de processamento pelo servidor Internet.

2.2 CENÁRIO B

Na descrição dos documentos listados abaixo, podemos verificar uma grande dependência de ferramentas[1], que possibilitam disponibilizar informações geográficas no meio Internet, implicando em um relativo custo financeiro e inviabilizando, em muitos casos, a distribuição desse tipo de informação:

a) **Gis on the Internet and Environmental Information and Planning** (SCHLLER, 1998), apresenta uma solução de SIG para acesso de informações via Internet, utilizando-se a tecnologia MapObject® e MapObject Internet Map Server®;

---

[1] As ferramentas: MapObjects®, MapObjects Internet Map Server® e ArcView IMS®, são marcas registradas da ESRI (*Environmental Systems Research Institute*).

b) **An Internet GIS Application for the Economic Development Depart of the City of Pittsburg** (MONZON, 1999), propõe uma solução para um sistema, desenvolvido com a utilização da ferramenta ArcView IMS®;

c) **Advanced Internet Map Server Techniques for Building Intuitive Web Sites for a Nontechnical User Base** (LOWE, 1999), propõe uma solução de SIG utilizando a implementação em linguagem C++ e funcionalidades do MapObject®;

d) **Web-Based GIS Migration: How to Get Your GIS Data and Application to Your Users and of All Those Computers** (LU, 2000), apresenta uma solução de migração de SIG para o meio Internet, utilizando as ferramentas ArcView IMS® e MapObjects IMS®;

e) **User Focus for a Multimap Web Site: Improving the Usability of an Online Atlas** (SVENSSON, 2000), propõe uma solução de visualização de mapas em ambiente Internet, utilizando recursos do ArcViewIMS®;

f) **Developing Internet-Based GIS Applications** (MARSHALL, 2000), Examina várias técnicas que podem ser utilizadas para desenvolver aplicações SIG, baseadas em Internet, utilizando as ferramentas MapObjects® e MapObjects Internet Map Server®.

2.3 CENÁRIO C

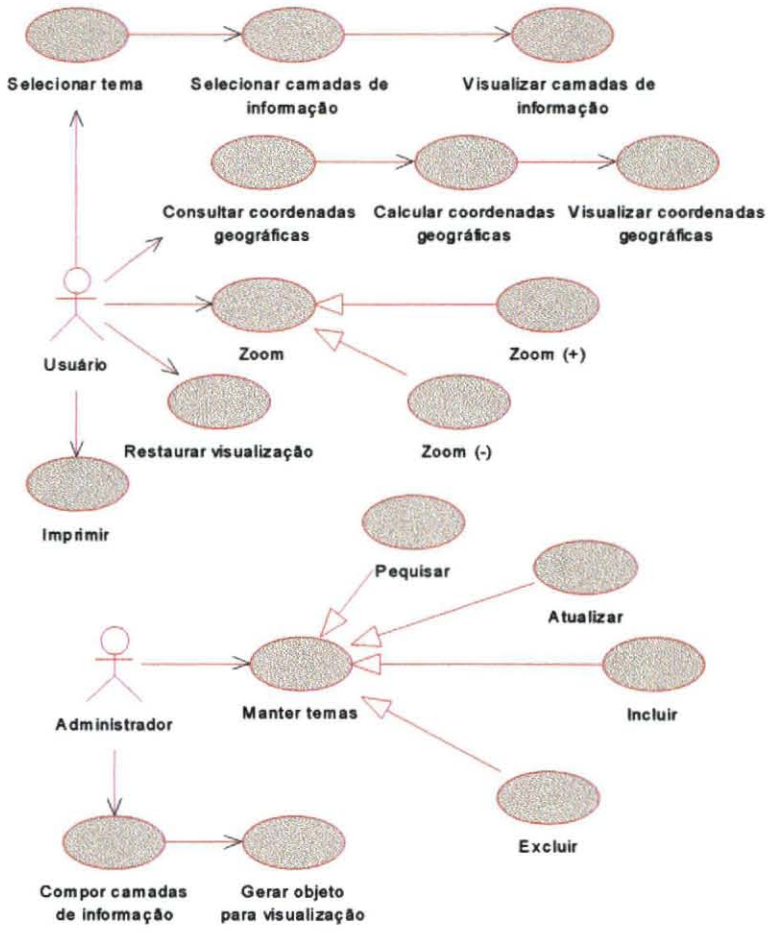
Segundo FONSECA e DAVIS JUNIOR (1998) no artigo **Geoprocessamento e Internet: Cenário Atual e Perspectivas**: *"O atual estágio de desenvolvimento da tecnologia WWW permite uma distribuição de informações muito mais interativa e dinâmica do que há alguns anos atrás. Com relação às informações geográficas, a Internet se apresenta como uma forma de acesso interessante, principalmente para usuários com pequenas e específicas demandas. Demonstra-se que isto é possível utilizando programas (applets) desenvolvidos na linguagem JAVA, ..."* (FONSECA, DAVIS JUNIOR, 1998). Justifica-se, com esta observação, que existe uma necessidade de desenvolvimento de ferramentas que possibilitem a distribuição de informações geográficas, para diversas camadas da população, utilizando-se do meio Internet.

3 ANÁLISE

A análise orientada a objetos introduziu uma série de novos conceitos de modelagem e, apesar da inexistência de uma concordância universal entre eles, um número limitado de idéias aparece repetidamente (PRESSMAN, 1995, p. 317-318). Em função disto, será adotada, no decorrer do trabalho, uma linguagem de modelagem orientada a objetos, nomeada *Unified Modeling Language* (UML), que propõe uma unificação de conceitos. Entretanto, não será adotada uma metodologia padrão para modelagem. Para melhor compreensão das notações sugeridas pela UML e utilizadas pelo trabalho, sugere-se consultar o apêndice 2.

3.1 DIAGRAMA DE CASO DE USO

FIGURA 06 – DIAGRAMA DE CASO DE USO



3.1.1 Definindo o Cenário do Sistema

No diagrama de caso de uso, apresentado na FIGURA 06, foi definido o cenário do projeto. O diagrama representa os limites do sistema e apresenta suas atividades básicas. Possui dois atores (usuário e administrador), sendo que o ator ***Usuário*** representa a pessoa que utiliza o sistema (para acesso às informações) e o ***Administrador*** o responsável pela manutenção dos dados do sistema. Neste caso, o sistema é considerado como sendo o conjunto de ferramentas utilizadas com o fim específico de **visualização de camadas de informação** no ambiente Internet, com o objetivo de consulta e visualização das informações disponibilizadas ou de criação e manutenção de mecanismo para acesso a essas informações.

3.1.2 Rotinas Relacionadas ao Ator Usuário

TABELA 01 – ROTINAS DO USUÁRIO

Rotina	**Descrição da rotina**
Selecionar tema	Selecionar um tema para visualização.
Selecionar camadas de informação	Selecionar camadas relacionadas ao tema selecionado.
Visualizar camadas de informação	Visualizar camadas relacionadas ao tema selecionado.
Consultar coordenadas geográficas	Consultar coordenada geográfica de um ponto em específico.
Calcular coordenadas geográficas	Calcular coordenada geográfica do ponto solicitado pelo usuário.
Visualizar coordenadas geográficas	Visualizar a coordenada geográfica calculada.
Zoom	Aumentar (Zoom (+)) ou diminuir (Zoom (-)) a imagem visualizada.
Restaurar visualização	Restaurar visualização das camadas de informação para um formato pré-determinado pelo sistema.
Imprimir	Imprimir a imagem visualizada das camadas de informação.

3.1.3 Rotinas Relacionadas ao Ator Administrador

TABELA 02 – ROTINAS DO ADMINISTRADOR

Rotina	**Descrição da rotina**
Manter temas	Pesquisar, atualizar, incluir e excluir temas para consulta e visualização.
Compor camadas de informação	A partir do objeto *Shapefile,* compor conjunto de camadas de informação que possam ser relacionadas a um tema.
Gerar objeto para visualização	Gerar os objetos correspondentes ao conjunto de camadas de informação, relacionadas a um tema, para visualização no meio Internet.

3.2 CARACTERÍSTICAS DO SISTEMA

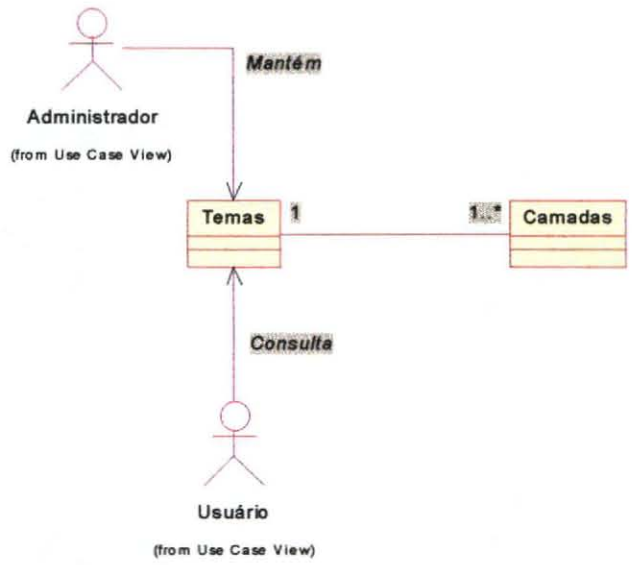
O sistema deve possuir as seguintes características:

a) ferramentas para entrada e manipulação de informações geográficas;
g) rotinas de gerenciamento de consulta a informação;
b) suportar pesquisa, análise e visualização de informações geográficas;
c) interface gráfica com o usuário (GUI) para fácil acesso às ferramentas do sistema;
d) decomposição, composição e visualização do objeto *.shp, possibilitando a distribuição do mesmo no meio Internet/Intranet.

3.3 DEFININDO O DIAGRAMA DE CLASSES PARA UM MODELO CONCEITUAL

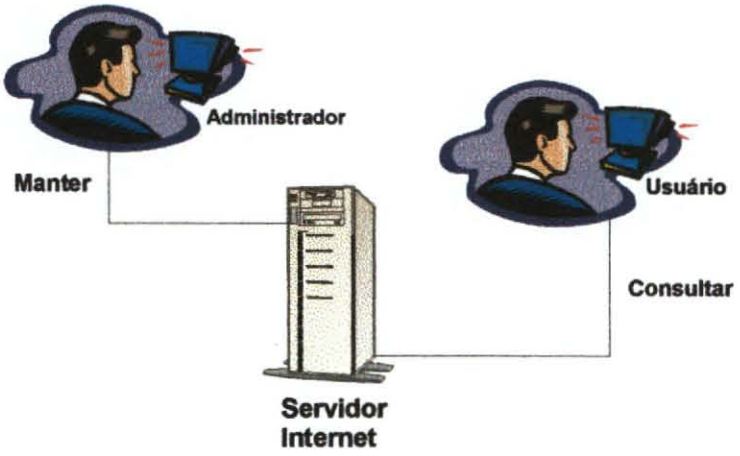
Representa-se neste momento o diagrama de classes, contendo uma definição conceitual dos objetos do sistema. O objetivo nesta fase é apenas identificar os objetos, a partir do diagrama de caso de uso, e inseri-los dentro do contexto do sistema. A representação será feita partindo de uma estrutura compacta, não se preocupando com os atributos e as operações das classes.

FIGURA 07 – MODELO CONCEITUAL DO SISTEMA



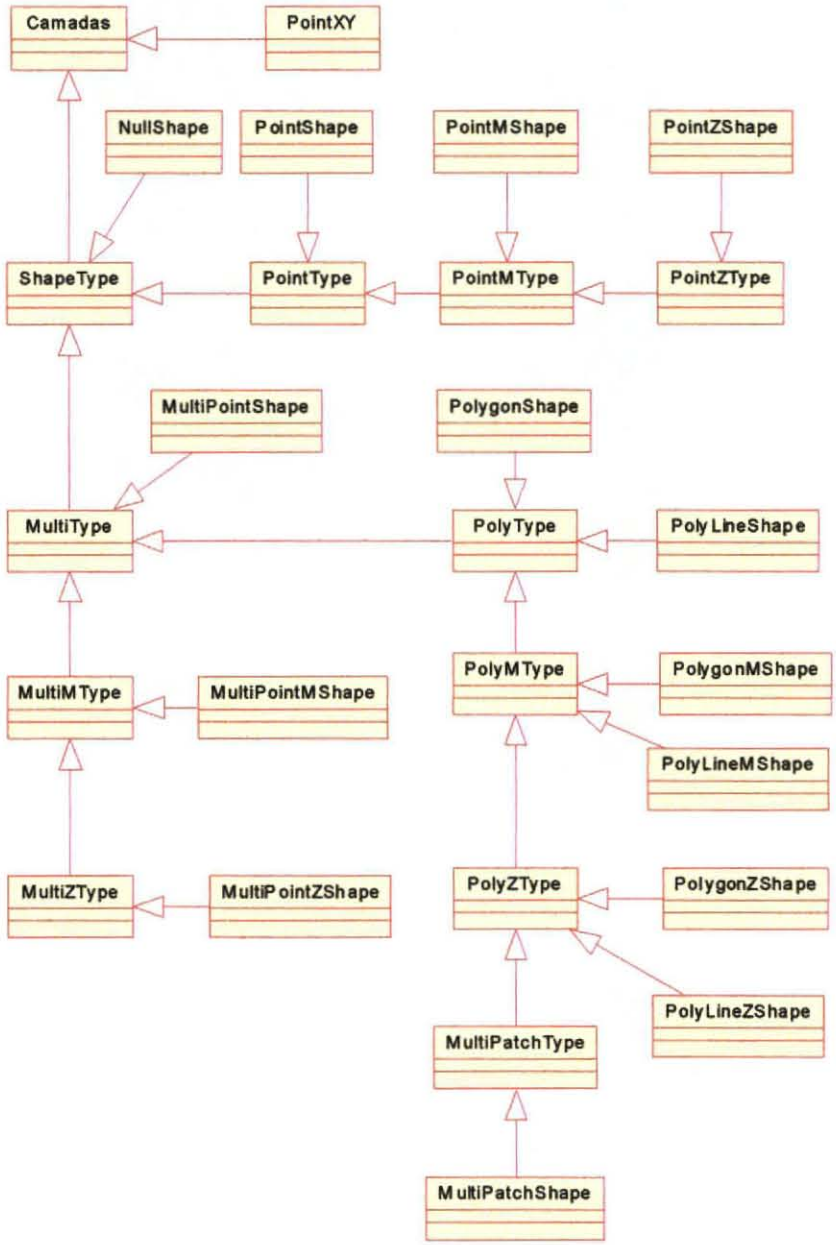
No diagrama de classes apresentado na FIGURA 07, identificam-se as funções básicas, representadas pelos relacionamentos (Consulta e Mantém) dos atores (Usuário e Administrador) com o sistema. Para cada tema (consultado ou mantido) poderá existir uma ou mais camadas de informação. Na FIGURA 08 é apresentada uma ilustração, que representa as ações desempenhadas pelos atores em relação ao sistema.

FIGURA 08 – AÇÕES DOS ATORES



3.3.1 Expandindo o Objeto Camadas

FIGURA 09 – MODELO CONCEITUAL DO OBJETO CAMADAS



No diagrama de classe apresentado na FIGURA 09, identifica-se o objeto *camadas* com suas possíveis abstrações. Para uma melhor compreensão, aconselha-se consultar o apêndice 1 deste trabalho, que contém a especificação técnica dos possíveis tipos assumidos pelo objeto em questão.

3.3.2 Possíveis Instâncias Assumidas pelo Objeto Camadas

Na TABELA 03, identificam-se os possíveis tipos assumidos pelo objeto *camadas em uma determina instância do objeto. Conforme diagrama de classes*, apresentado na FIGURA 09, os tipos são identificados como sendo as classes finais de um conjunto finito de generalizações. As generalizações do objeto visam absorver possíveis redundâncias de atributos e de operações.

TABELA 03 – INSTÂNCIAS DO OBJETO

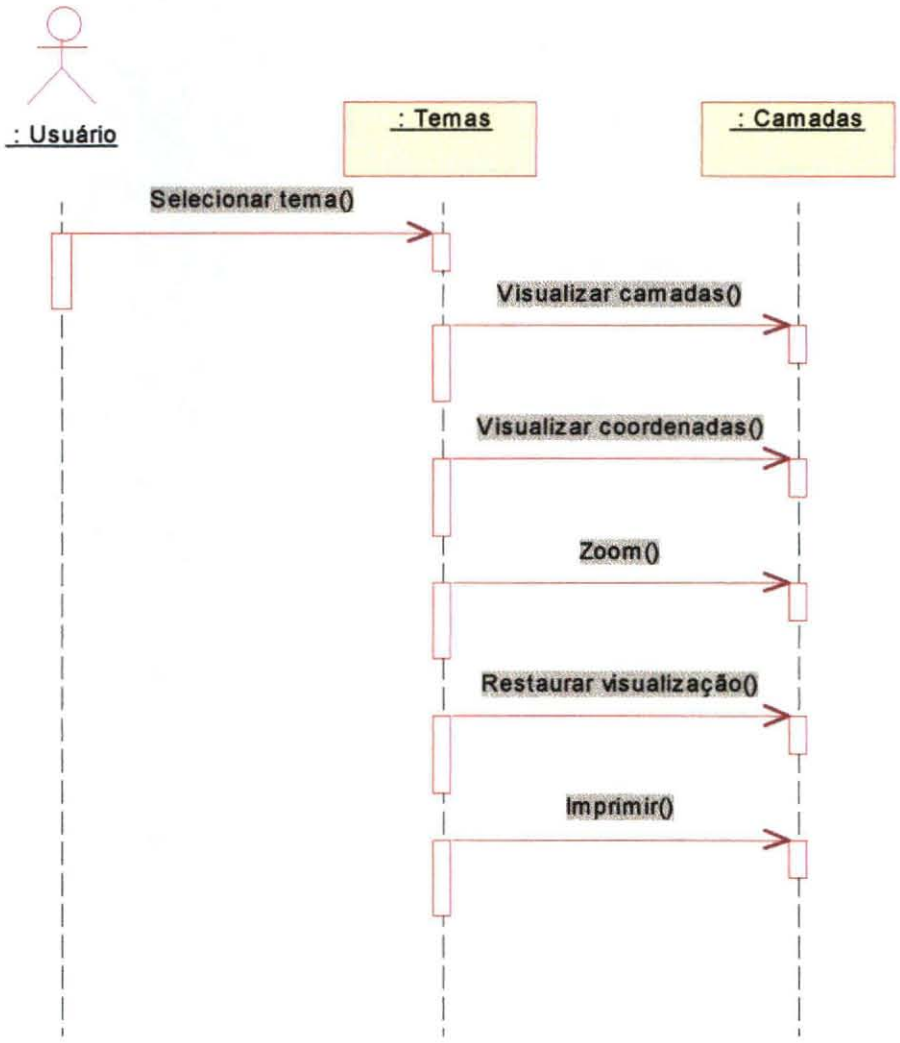
Instância
NullShape
PointShape
PolyLineShape
PolygonShape
MultiPointShape
PointZShape
PolyLineZShape
PolygonZShape
MultiPointZShape
PointMShape
PolyLineMShape
PolygonMShape
MultiPointMShape
MultiPatchShape

3.4 IDENTIFICANDO AS SEQÜÊNCIAS DAS AÇÕES

Identifica-se neste momento, através de diagramas de seqüências, as possíveis ações assumidas pelos atores na interação com o sistema.

3.4.1 Definindo o Diagrama de Seqüências para o Ator Usuário

FIGURA 10 – DIAGRAMA DE SEQÜÊNCIAS PARA O USUÁRIO



No diagrama de seqüências da FIGURA 10, descrevem-se as funções que representam a interação do ator ***Usuário*** com o sistema. A partir de um tema selecionado, o ***Usuário*** poderá ter acesso a uma ou mais das seguintes ações: visualização de camadas de informação, visualização de coordenadas geográficas, *zoom* do objeto visualizado, restaurar padrões de visualização e impressão do objeto visualizado.

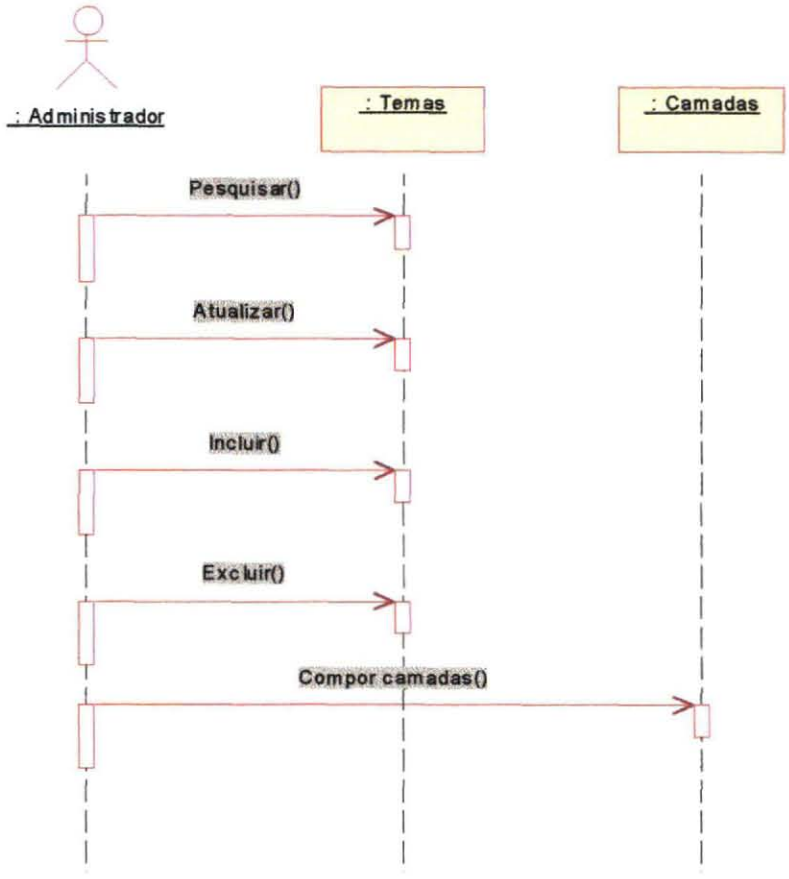
3.4.1.1 Descrevendo ações assumidas pelo Ator Usuário

TABELA 04 – AÇÕES DO USUÁRIO

Ações	**Descrição da ação**
Selecionar tema	Responsável pelo início do processo de visualização, obrigatória para ocorrência das demais ações descritas.
Visualizar camadas	Iniciada a partir de um tema selecionado, que tem como fim a visualização do objeto correspondente ao tema selecionado.
Visualizar coordenadas	Ação que retorna o valor das coordenadas geográficas de um ponto específico, contido dentro da área visualizada.
Zoom	Relacionada ao processo de visualização que permite ampliar ou diminuir a área visualizada.
Restaurar visualização	Possibilita restaurar os padrões básicos de visualização, caso este tenha sido alterado.
Imprimir	Inicia processo e impressão de camadas de informação.

3.4.2 Definindo o Diagrama de Seqüências para o Ator Administrador

FIGURA 11 - DIAGRAMA DE SEQÜÊNCIAS PARA O ADMINISTRADOR



No diagrama de seqüências apresentado na FIGURA 11, são apresentadas as funções que representam a interação do ator ***Administrador*** com o sistema. Lembramos que, apesar de um ator poder assumir diversos papéis dentro dos limites de um sistema, o ator ***Usuário*** poderá não ter permissão de *atuar* no papel de ***Administrador***; entretanto, o ator ***Administrador*** sempre poderá *atuar* como ***Usuário***.

3.4.2.1 Descrevendo ações assumidas pelo Ator Administrador

TABELA 05 – AÇÕES DO ADMINISTRADOR

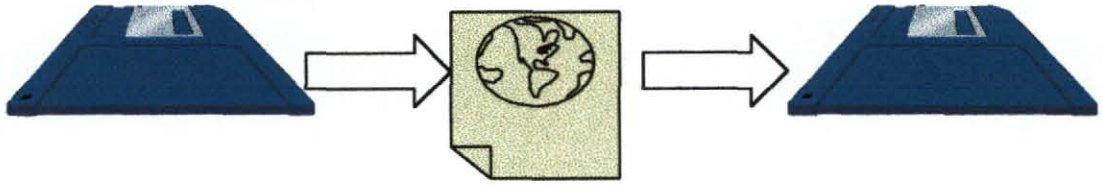
Ações	**Descrição da ação**
Pesquisar	Pesquisar temas disponíveis para visualização.
Atualizar	Atualizar temas disponibilizados.
Incluir	Incluir novos temas para visualização.
Excluir	Excluir temas disponibilizados.
Compor camadas	Compor objeto contendo as camadas de informação relacionadas a um tema em específico.

3.5 O OBJETO *.SER

Para garantir a portabilidade de distribuição, será utilizado um arquivo de intercâmbio (**.ser*), que corresponde à representação direta da classe que contém o objeto *Shapefile*.

O objeto *Shapefile* é decomposto, obedecendo a sua especificação técnica, a qual é definida no apêndice 1, composto em uma classe que corresponde à sua representação, a qual é transformada em um arquivo de intercâmbio. Conforme ilustrado na FIGURA 12.

FIGURA 12 – DECOMPOSIÇÃO E COMPOSIÇÃO DE OBJETOS



Objeto *Shapefile (*.shp)* **Classe** **Arquivo de intercâmbio (*.ser)**

3.6 A REPRESENTAÇÃO DO OBJETO SHAPEFILE PELA CLASSE

Conforme a especificação técnica do objeto, o arquivo *.shp é constituído por um cabeçalho de arquivo, seguido por um determinado número de registros, sendo cada registro dividido em duas partes: cabeçalho do registro e conteúdo do registro.

Cabeçalho do arquivo	
Cabeçalho do registro	Conteúdo do registro
...	...
Cabeçalho do registro	Conteúdo do registro

3.6.1 Modelo Conceitual do Arquivo Shapefile

FIGURA 13 – MODELO CONCEITUAL DO ARQUIVO SHAPEFILE

ShapeSHP é a representação do arquivo *Shapefile* como objeto, e *BBox* (*"Bounding Box"*) é o campo do cabeçalho do arquivo que representa o menor retângulo ortogonal, que contém toda a imagem representada pelo arquivo, e que será representado como uma extensão do objeto *ShapeSHP*.

3.6.2 Modelo Conceitual do Registro do Arquivo Shapefile

FIGURA 14 – MODELO CONCEITUAL DO REGISTRO DO ARQUIVO

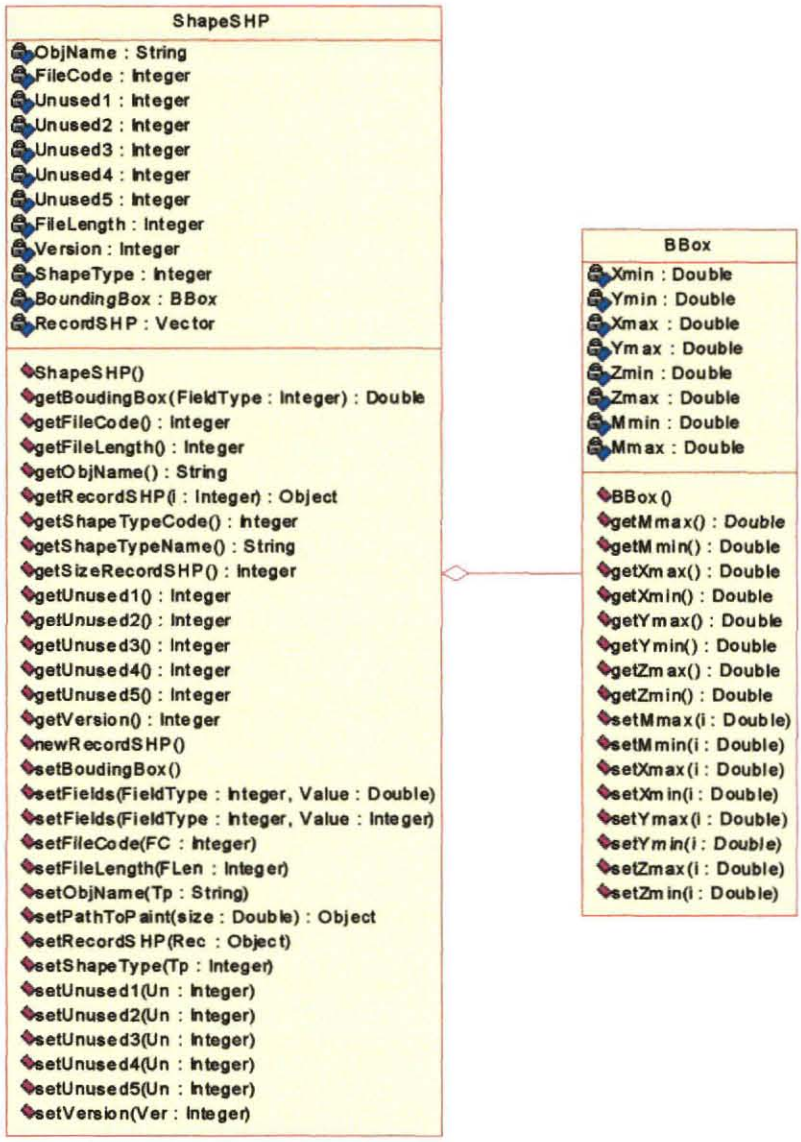
Record é a representação de um registro do arquivo *Shapefile* como objeto, e *Header* é a representação do cabeçalho do registro como extensão do objeto *Record*. O objeto *ShapeSHP* deverá conter um vetor de objetos do tipo *Record*, que corresponderá aos registros do arquivo.

4 PROJETO

Para atender aos requisitos da fase de projeto, faz-se necessário especificar de maneira refinada as características dos objetos descritos na fase de análise, identificando e incluindo os atributos e operações correspondentes, buscando uma representação próxima do código fonte (PRESSMAN, 1995, p. 418).

4.1 ESPECIFICANDO AS OPERAÇÕES DA CLASSE SHAPESHP

FIGURA 15 – DIAGRAMA DE CLASSES PARA O OBJETO SHAPESHP



4.1.1 Definindo as Operações da Classe ShapeSHP

TABELA 06 – OPERAÇÕES DA CLASSE SHAPESHP

Método	**Definição**
getBoudingBox	Retorna os valores das coordenadas mínimas e máximas que contêm a imagem a ser visualizada. O parâmetro FieldType pode receber os seguintes valores: 36, 44, 52, 60, 68, 76, 84, 92 e retornando respectivamente os valores: X mínimo, Y mínimo, X máximo, Y máximo, Z mínimo, Z máximo, M mínimo, M máximo ou 0.0 se um parâmetro diferente for passado.
getFileCode	Retorna o valor do atributo FileCode.
getFileLength	Retorna o valor do atributo FileLength.
getObjName	Retorna o valor do atributo ObjName.
getRecordSHP	Retorna o objeto contido no vetor RecordSHP indexado pelo parâmetro i.
getShapeTypeCode.	Retorna o valor do atributo ShapeType.
getShapeTypeName	Retorna o tipo correspondente ao atributo ShapeType, conforme tabela contida no item 2.2.1 do apêndice 1.
getSizeRecordSHP	Retorna a quantidade de elementos contidos no vetor RecordSHP.
getUnused1	Retorna o valor do atributo Unused1.
getUnused2	Retorna o valor do atributo Unused2.
getUnused3	Retorna o valor do atributo Unused3.
getUnused4	Retorna o valor do atributo Unused4.
getUnused5	Retorna o valor do atributo Unused5.
getVersion	Retorna o valor do atributo Version.
newRecordSHP	Atribui um novo vetor de objetos do tipo Record ao atributo RecordSHP.
setBoudingBox	Atribui uma nova instância do objeto BBox, descrito no item 3.1.2, ao atributo BoudingBox.
setFields com parâmetro Value = Double	Atribui os valores das coordenadas mínimas e máximas que contêm a imagem a ser visualizada, ao objeto BoudingBox. O parâmetro FieldType pode receber os seguintes valores: 36, 44, 52, 60, 68, 76, 84, 92 atribuindo respectivamente os valores: X mínimo, Y mínimo, X máximo, Y máximo, Z mínimo, Z máximo, M mínimo, M máximo.
setFields com parâmetro Value = Integer	Atribui os valores aos atributos: FileCode, Unused1, Unused2, Unused3, Unused4, Unused5, FileLength, Version e ShapeType. O parâmetro FieldType pode receber os seguintes valores: 0, 4, 8, 12, 16, 20, 24, 28, 32 correspondendo respectivamente aos atributos citados.
setFileCode	Atribui valor ao atributo FileCode.
setFileLength	Atribui valor ao atributo FileLength
setObjName	Atribui valor ao atributo ObjName.
setPathToPaint	Retorna o objeto que contém as coordenadas dos pontos das camadas de informação convertidas pelo parâmetro size.
setRecordSHP	Adiciona um elemento no vetor do atributo RecordSHP.
setShapeType	Atribui valor ao atributo ShapeType.
setUnused1	Atribui valor ao atributo Unused1.
setUnused2	Atribui valor ao atributo Unused2.
setUnused3	Atribui valor ao atributo Unused3.
setUnused4	Atribui valor ao atributo Unused4.
setUnused5	Atribui valor ao atributo Unused5.
setVersion	Atribui valor ao atributo Version.

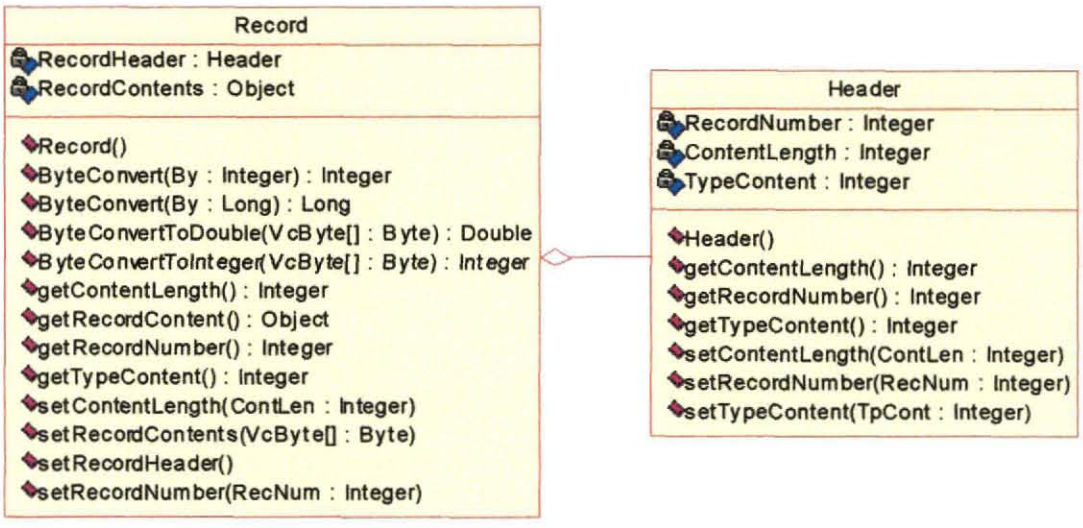
4.1.2 Definindo as Operações da Classe BBox

TABELA 07 – OPERAÇÕES DA CLASSE BBOX

Método	**Definição**
getMmax	Retorna o valor do atributo Mmax.
getMmin	Retorna o valor do atributo Mmin.
getXmax	Retorna o valor do atributo Xmax.
getXmin	Retorna o valor do atributo Xmin.
getYmax	Retorna o valor do atributo Ymax.
getYmin	Retorna o valor do atributo Ymin.
getZmax	Retorna o valor do atributo Zmax.
getZmin	Retorna o valor do atributo Zmin.
setMmax	Atribui valor ao atributo Mmax.
setMmin	Atribui valor ao atributo Mmin.
setXmax	Atribui valor ao atributo Xmax.
setXmin	Atribui valor ao atributo Xmin.
setYmax	Atribui valor ao atributo Ymax.
setYmin	Atribui valor ao atributo Ymin.
setZmax	Atribui valor ao atributo Zmax.
setZmin	Atribui valor ao atributo Zmin.

4.2 ESPECIFICANDO AS OPERAÇÕES DA CLASSE RECORD

FIGURA 16 – DIAGRAMA DE CLASSES PARA O OBJETO RECORD



4.2.1 Definindo as Operações da Classe Record

TABELA 08 – OPERAÇÕES DA CLASSE RECORD

Método	Definição
ByteConvert com parâmetro Integer	Verifica se o Byte é negativo; se for, converte o valor.
ByteConvert com parâmetro Long	Verifica se o Byte é negativo; se for, converte o valor.
ByteConvertToDouble	Recebe um vetor de oito Bytes e retorna um número do tipo Double.
ByteConvertToInteger	Recebe um vetor de quatro Bytes e retorna um número do tipo Integer.
getContentLength	Retorna o valor do atributo ContentLength.
getRecordContent	Retorna o objeto correspondente ao atributo RecordContent.
getRecordNumber	Retorna o valor do atributo RecordNumber.
getTypeContent	Retorna o valor do atributo TypeContent.
setContentLength	Atribui valor ao atributo ContentLength.
setRecordContents	Atribui valor ao atributo RecordContents.
setRecordHeader	Cria uma nova instância do objeto RecordHeader.
setRecordNumber	Atribui valor ao atributo RecordNumber.

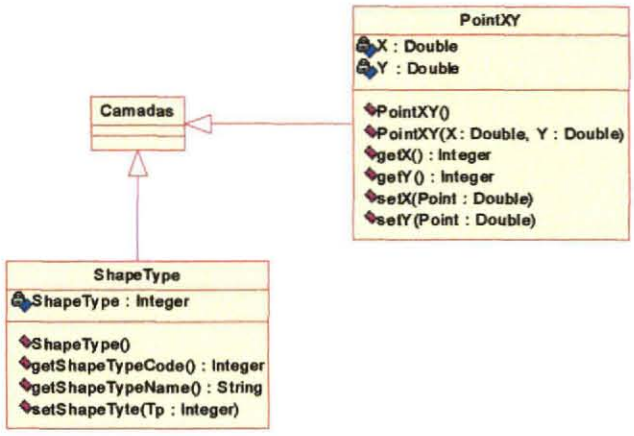
4.2.2 Definindo as Operações da Classe Header

TABELA 09 – OPERAÇÕES DA CLASSE HEADER

Método	Definição
getContentLength	Retorna o valor do atributo ContentLength.
getRecordNumber	Retorna o valor do atributo RecordNumber.
getTypeContent	Retorna o valor do atributo TypeContent.
setContentLength	Atribui valor ao atributo ContentLength.
setRecordNumber	Atribui valor ao atributo RecordNumber.
setTypeContent	Atribui valor ao atributo TypeContent.

4.3 GENERALIZANDO A CLASSE CAMADAS

FIGURA 17 – DIAGRAMA DE CLASSES PARA O OBJETO CAMADAS



4.3.1 O Objeto PointXY

PointXY é a representação de um ponto de coordenada (X, Y) por uma classe.

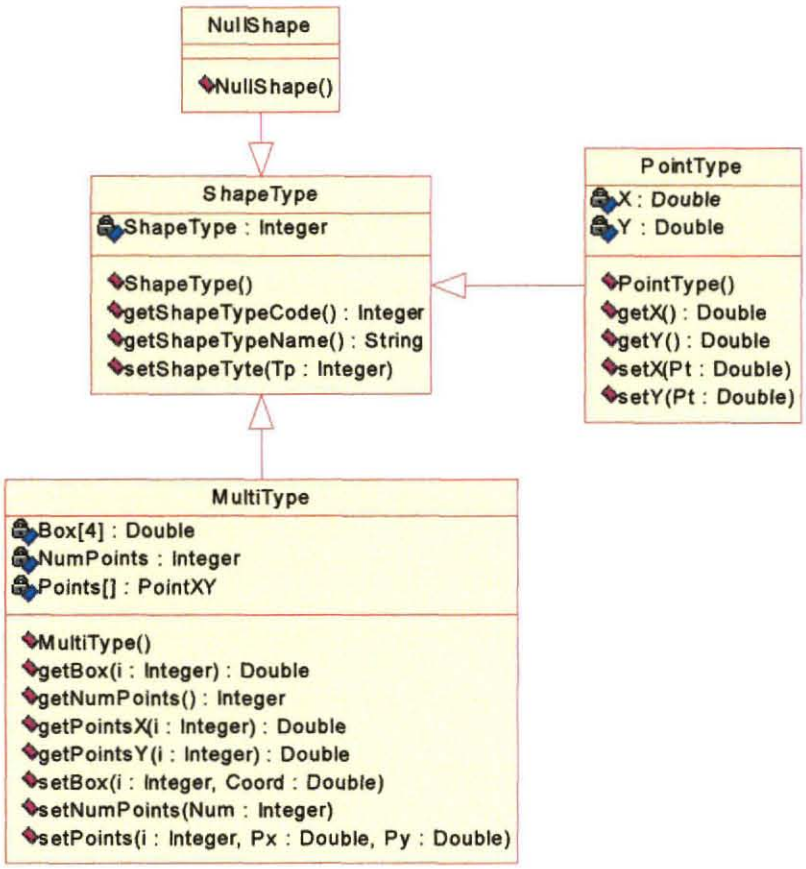
4.3.1.1 Definindo as operações da classe PointXY

TABELA 10 – OPERAÇÕES DA CLASSE POINTXY

Método	**Definição**
PointXY com parâmetro X e Y	Cria uma nova instância de PointXY atribuindo valores para os atributos X e Y.
getX	Retorna o valor do atributo X.
getY	Retorna o valor do atributo Y.
setX	Atribui valor ao atributo X.
setY	Atribui valor ao atributo Y.

4.4 GENERALIZANDO A CLASSE SHAPETYPE

FIGURA 18 – DIAGRAMA DE CLASSES PARA O OBJETO SHAPETYPE



4.4.1 Definindo as Operações da Classe ShapeType

TABELA 11 – OPERAÇÕES DA CLASSE SHAPETYPE

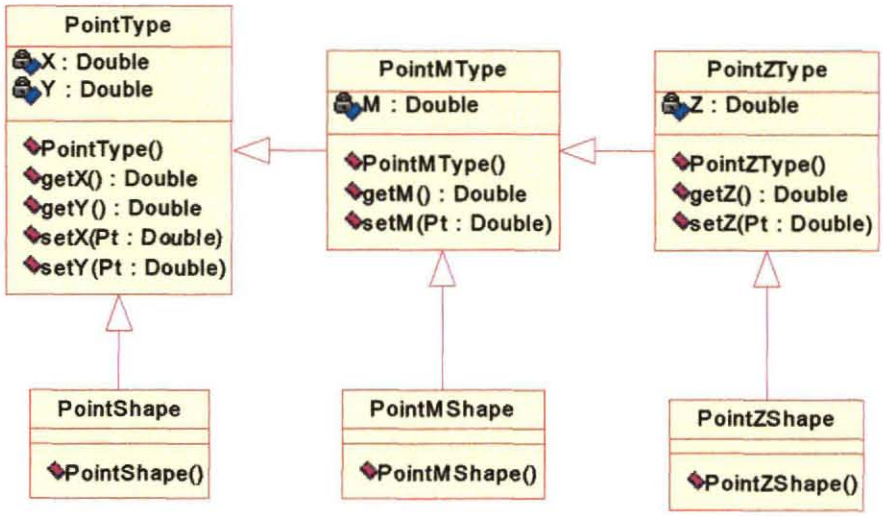
Método	Definição
getShapeTypeCode	Retorna o valor do atributo ShapeType.
getShapeTypeName	Retorna o tipo do atributo ShapeType conforme tabela contida no item 2.2.1 do apêndice 1.
setShapeType	Atribui valor ao atributo ShapeType.

4.4.2 O Objeto NullShape

NullShape é a representação do conteúdo do registro, de tipo *Null Shape*, do arquivo *.shp, conforme especificação contida no item 2.4.1 do apêndice 1.

4.5 GENERALIZANDO A CLASSE POINTTYPE

FIGURA 19 – DIAGRAMA DE CLASSES PARA O OBJETO POINTTYPE



4.5.1 Definindo as Operações da Classe PointType

TABELA 12 – OPERAÇÕES DA CLASSE POINTTYPE

Método	Definição
getX	Retorna o valor do atributo X.
getY	Retorna o valor do atributo Y.
setX	Atribui valor ao atributo X.
setY	Atribui valor ao atributo Y.

4.5.1.1 O objeto PointShape

PointShape é a representação do conteúdo do registro, de tipo *Point*, do arquivo *.shp, conforme especificação contida no item 2.4.2 do apêndice 1.

4.5.2 Definindo as Operações da Classe PointMType

TABELA 13 – OPERAÇÕES DA CLASSE POINTMTYPE

Método	**Definição**
getM	Retorna o valor do atributo M.
setM	Atribui valor ao atributo M.

4.5.2.1 O objeto PointMShape

PointMShape é a representação do conteúdo do registro, de tipo *PointM*, do arquivo *.shp, conforme especificação contida no item 2.4.6 do apêndice 1.

4.5.3 Definindo as Operações da Classe PointZType

TABELA 14 – OPERAÇÕES DA CLASSE POINTZTYPE

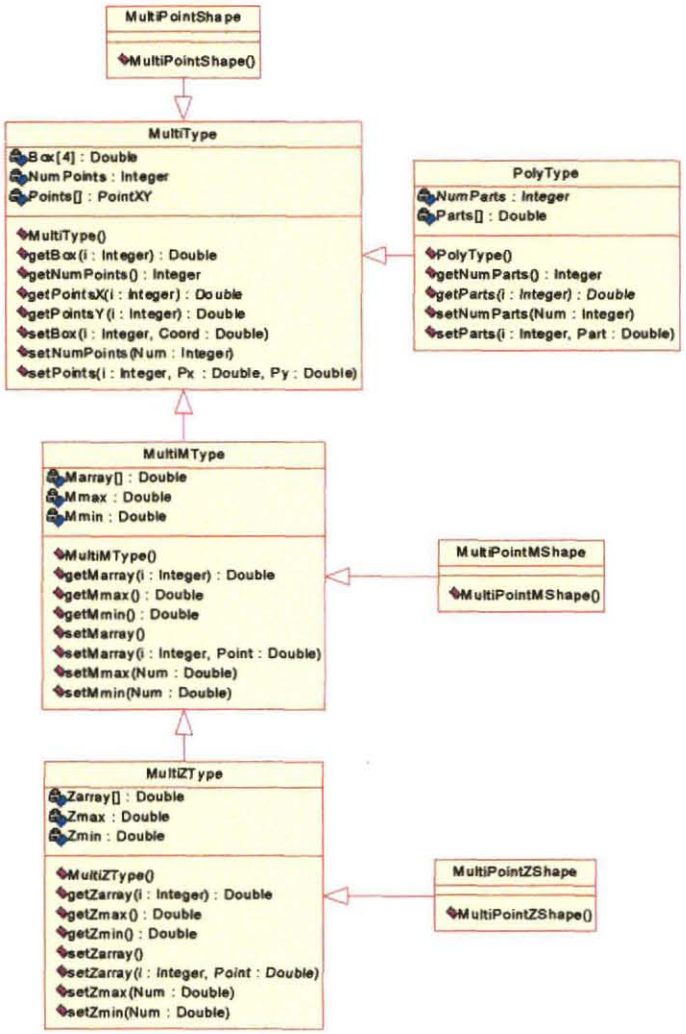
Método	**Definição**
getZ	Retorna o valor do atributo Z.
setZ	Atribui valor ao atributo Z.

4.5.3.1 O objeto PointZShape

PointZShape é a representação do conteúdo do registro, de tipo *PointZ*, do arquivo *.shp, conforme especificação contida no item 2.4.10 do apêndice 1.

4.6 GENERALIZANDO A CLASSE MULTITYPE

FIGURA 20 – DIAGRAMA DE CLASSES PARA O OBJETO MULTITYPE



4.6.1 Definindo as Operações da Classe MultiType

TABELA 15 – OPERAÇÕES DA CLASSE MULTITYPE

Método	**Definição**
getBox	Retorna o valor do atributo Box indexado pelo parâmetro *i*.
getNumPoints	Retorna o valor do atributo NumPoints.
getPointsX	Retorna o valor de X do atributo Points indexado por *i*.
getPointsY	Retorna o valor de Y do atributo Points indexado por *i*.
setBox	Atribui valor ao atributo Box indexado por *i*.
setNumPoints	Atribui valor ao atributo NumPoints.
setPoints	Atribui valor a atributo Points indexado por *i*.

4.6.1.1 O objeto MultiPointShape

MultiPointShape é a representação do conteúdo do registro, de tipo *MultiPoint*, do arquivo *.shp, conforme especificação contida no item 2.4.3 do apêndice 1.

4.6.2 Definindo as Operações da Classe MultiMType

TABELA 16 – OPERAÇÕES DA CLASSE MULTIMTYPE

Método	**Definição**
getMarray	Retorna o valor do atributo Marray indexado por i.
getMmax	Retorna o valor do atributo Mmax.
getMmin	Retorna o valor do atributo Mmin.
setMarray	Cria uma nova instância de Marray.
setMarray com parâmetros i e point	Atribui o valor point ao atributo Marray indexado por i.
setMmax	Atribui valor ao atributo Mmax.
setMmin	Atribui valor ao atributo Mmin.

4.6.2.1 O objeto MultiPointMShape

MultiPointMShape é a representação do conteúdo do registro, de tipo *MultiPointM*, do arquivo *.shp, conforme especificação contida no item 2.4.7 do apêndice 1.

4.6.3 Definindo as Operações da Classe MultiZType

TABELA 17 – OPERAÇÕES DA CLASSE MULTIZTYPE

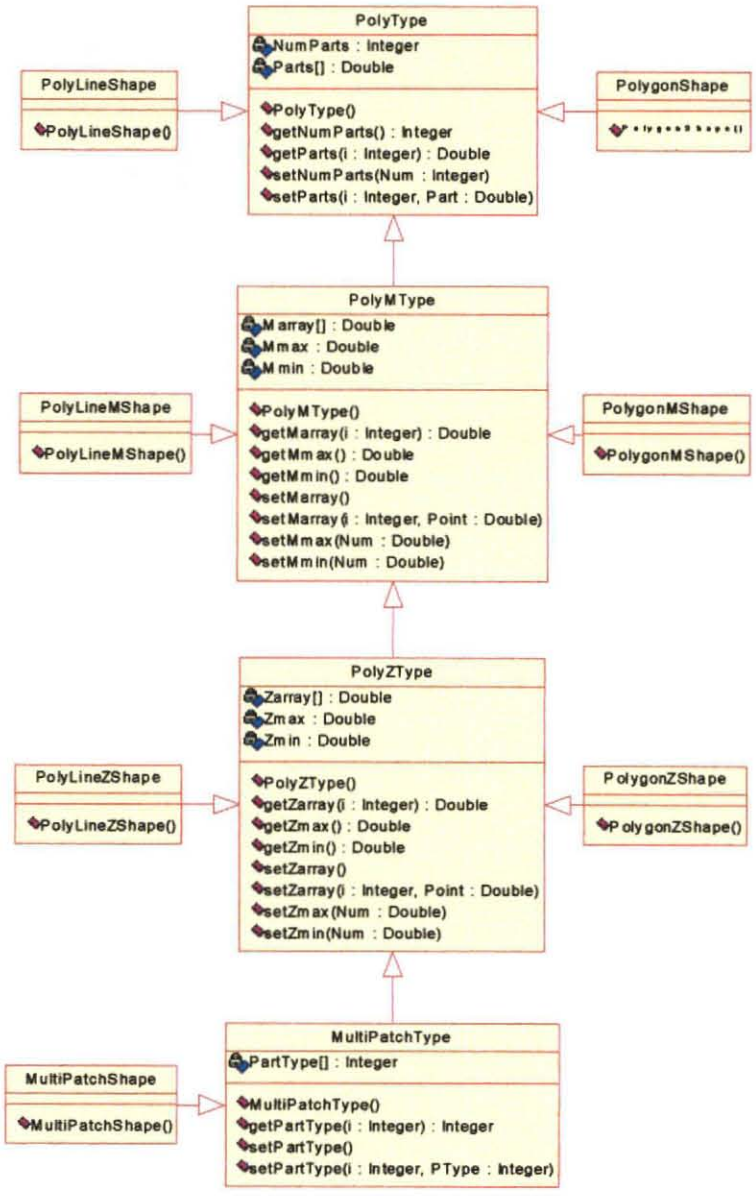
Método	**Definição**
getZarray	Retorna o valor do atributo Zarray indexado por i.
getZmax	Retorna o valor do atributo Zmax.
getZmin	Retorna o valor do atributo Zmin.
setZarray	Cria uma nova instância de Zarray.
setZarray com parâmetro i e point	Atribui o valor point ao atributo Zarray indexado por i.
setZmax	Atribui valor ao atributo Zmax.
setZmin	Atribui valor ao atributo Zmin.

4.6.3.1 O objeto MultiPointZShape

MultiPointZShape é a representação do conteúdo do registro, de tipo *MultiPointZ*, do arquivo *.shp, conforme especificação contida no item 2.4.11 do apêndice 1.

4.7 GENERALIZANDO A CLASSE POLYTYPE

FIGURA 21 – DIAGRAMA DE CLASSES PARA O OBJETO POLYTYPE



4.7.1 Definindo as Operações da Classe PolyType

TABELA 18 – OPERAÇÕES DA CLASSE POLYTYPE

Método	**Definição**
getNumParts	Retorna o valor do atributo NumParts.
getParts	Retorna o valor do atributo Parts indexado por i.
setNumParts	Atribui valor ao atributo NumParts.
setParts	Atribui valor ao atributo Parts indexado por i.

4.7.1.1 O objeto PolyLineShape

PolyLineShape é a representação do conteúdo do registro, de tipo *PolyLine*, do arquivo *.shp, conforme especificação contida no item 2.4.4 do apêndice 1.

4.7.1.2 O objeto PolygonShape

PolygonShape é a representação do conteúdo do registro, de tipo *Polygon*, do arquivo *.shp, conforme especificação contida no item 2.4.5 do apêndice 1.

4.7.2 Definindo as Operações da Classe PolyMType

TABELA 19 – OPERAÇÕES DA CLASSE POLYMTYPE

Método	**Definição**
getMarray	Retorna o valor do atributo Marray indexado por i.
getMmax	Retorna o valor do atributo Mmax.
getMmin	Retorna o valor do atributo Mmin.
setMarray	Cria uma nova instância de Marray.
setMarray com parâmetros i e point	Atribui o valor point ao atributo Marray indexado por i.
setMmax	Atribui valor ao atributo Mmax.
setMmin	Atribui valor ao atributo Mmin.

4.7.2.1 O objeto PolyLineMShape

PolyLineMShape é a representação do conteúdo do registro, de tipo *PolyLineM*, do arquivo *.shp, conforme especificação contida no item 2.4.8 do apêndice 1.

4.7.2.2 O objeto PolygonMShape

PolygonMShape é a representação do conteúdo do registro, de tipo *PolygonM*, do arquivo *.shp, conforme especificação contida no item 2.4.9 do apêndice 1.

4.7.3 Definindo as Operações da Classe PolyZType

TABELA 20 – OPERAÇÕES DA CLASSE POLYZTYPE

Método	**Definição**
getZarray	Retorna o valor do atributo Zarray indexado por i.
getZmax	Retorna o valor do atributo Zmax.
getZmin	Retorna o valor do atributo Zmin.
setZarray	Cria uma nova instância de Zarray.
setZarray com parâmetro i e point	Atribui o valor point ao atributo Zarray indexado por i.
setZmax	Atribui valor ao atributo Zmax.
setZmin	Atribui valor ao atributo Zmin.

4.7.3.1 O objeto PolyLineZShape

PolyLineZShape é a representação do conteúdo do registro, de tipo *PolyLineZ*, do arquivo *.shp, conforme especificação contida no item 2.4.12 do apêndice 1.

4.7.3.2 O objeto PolygonZShape

PolygonZShape é a representação do conteúdo do registro, de tipo *PolygonZ*, do arquivo *.shp, conforme especificação contida no item 2.4.13 do apêndice 1.

4.7.4 Definindo as Operações da Classe MultiPatchType

TABELA 21 – OPERAÇÕES DA CLASSE MULTIPATCHTYPE

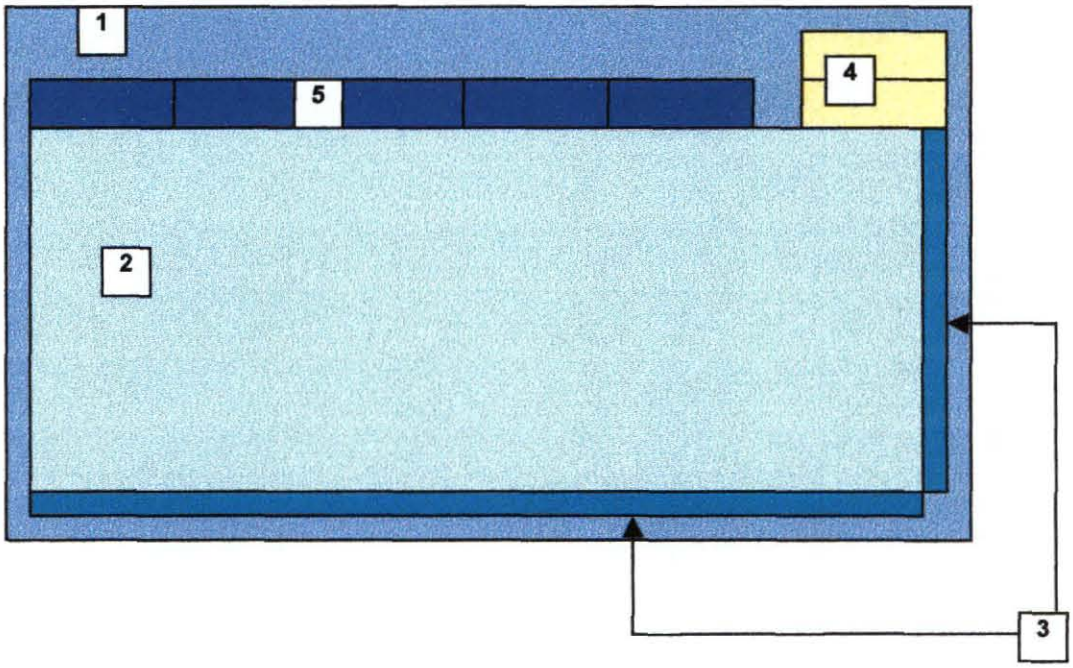
Método	**Definição**
getPartType	Retorna o valor do atributo PartType indexado por i.
setPartType	Cria uma nova instância de PartType.
setPartType com parâmetro i e PType	Atribui valor PType ao atributo PartType indexado por i.

4.7.4.1 O objeto MultiPatchShape

MultiPatchShape é a representação do conteúdo do registro, de tipo *MultiPatch*, do arquivo *.shp, conforme especificação contida no item 2.4.14 do apêndice 1.

4.8 DEFININDO A INTERFACE COM O USUÁRIO

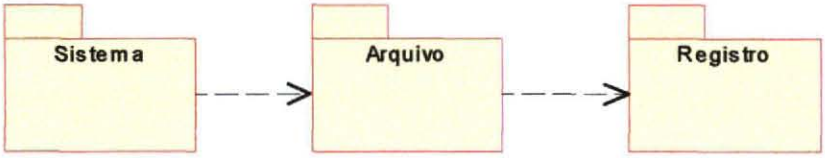
FIGURA 22 – INTERFACE DA APPLET DE VISUALIZAÇÃO



1 Applet 1 – funções de controle de visualização de imagem.
2 Applet 2 – visualização da imagem.
3 Barras de navegação – movimentação da imagem na Applet2.
4 Campos para visualização de coordenadas geográficas.
5 Botões (Zoom(+), Zoom(-), Restaurar, Imprimir e Selecionar cor).

4.9 EMPACOTANDO O SISTEMA

FIGURA 23 – DIAGRAMA DE PACOTES



4.9.1 Definindo as Classes Contidas no Pacote Sistema

TABELA 22 – PACOTE SISTEMA

Applet para visualização de camadas de informação
Frame para decomposição e composição do objeto ShapeFile
Classe para impressão de objeto visualizado

4.9.2 Definindo as Classes Contidas no Pacote Arquivo

TABELA 23 – PACOTE ARQUIVO

Bbox
Header
Record
ShapeSHP

4.9.3 Definindo as Classes Contidas no Pacote Registro

TABELA 24 – PACOTE REGISTRO

MultiMType
MultiPatchShape
MultiPatchType
MultiPointMShape
MultiPointShape
MultiPointZShape
MultiType
MultiZType
NullShape
PointMShape
PointMType
PointShape
PointType
PointXY
PointZShape
PointZType
PolygonMShape
PolygonShape
PolygonZShape
PolyLineMShape
PolyLineShape
PolyLineZShape
PolyMType
PolyType
PolyZType
ShapeType

4.10 DEFININDO REGRAS BÁSICAS PARA DISTRIBUIÇÃO DO SISTEMA

Para possibilitar uma grande mobilidade na distribuição de informações geográficas, constituídas por camadas de informação, pelo usuário que vier a utilizar o sistema, foram adotados os seguintes critérios:

a) o pacote de distribuição do sistema deverá ser constituído basicamente por duas partes: uma correspondendo ao conjunto de ferramentas necessárias para decomposição do objeto *ShapeFile* e composição do objeto *.ser (arquivo de intercâmbio), outra correspondendo ao conjunto de ferramenta que será utilizado para distribuição das camadas de informação no meio Internet (*Applet* de visualização);
b) a composição dos temas relacionados à visualização das camadas de informação ficará aberta ao usuário. Esses temas deverão ser constituídos através de *links* em HTML;
c) os *links* em HTML deverão fornecer recursos para passagem de parâmetros, possibilitando a visualização de camadas de informação;
d) deverá fazer parte do pacote de distribuição do sistema um modelo de código em HTML, compatível com a *Applet* Java, para ser utilizado pelo *link* criado pelo usuário.

5 IMPLEMENTAÇÃO

"*A codificação é uma extensão do processo de projetar*" (RUMBAUGH, BLAHA, PREMERLANI, 1994, p. 371). Com este objetivo, serão apresentados na seqüência alguns trechos de códigos implementados em linguagem de programação Java 1.2.

5.1 IMPLEMENTANDO SOLUÇÕES

Apresenta-se neste capítulo somente a implementação relevante para solução do problema, o restante do código fonte está contido no apêndice 3.

5.1.1 Decompondo o Objeto *shp

A decomposição do objeto é caracterizada pela leitura *Byte* a *Byte* do arquivo de origem (*.shp), a converção dos *Bytes* e conseqüentemente a atribuição dos valores ao objeto *ShapeSHP*, que é a representação virtual do objeto *ShapeFile*. Esta seqüência de ações foi implementada na operação **decomporObjeto (String filePath)**, listada no QUADRO 01, onde o objeto de origem (*.shp) é passado como parâmetro (filePath). Esse processo é necessário para obter os elementos necessários para visualização do objeto e para implementação de recursos de distribuição no meio Internet.

QUADRO 01 – CÓDIGO FONTE – OPERAÇÃO DECOMPOROBJETO

```
public void decomporObjeto(String filePath) {
        try{
                FileInputStream fis = new FileInputStream(filePath);
                int i = 0;
                double b = 0.00;
                //Instância do objeto ShapeSHP, que representa *.shp
                ObjetoSHP = new ShapeSHP();
                ObjetoSHP.setObjName("");
                //Cabeçalho do arquivo
                while (i < 28){ // Byte Big Order - Integer
                        ObjetoSHP.setFields(i, ((fis.read() << 24) |
                                (fis.read() << 16) + (fis.read() << 8) + fis.read()));
                        i = i + 4;
                }
                while (i < 36){ // Byte Little Order - Integer
```

```
                    ObjetoSHP.setFields(i, (fis.read() + (fis.read() << 8) +
                            (fis.read() << 16) | (fis.read() << 24)));
                    i = i + 4;
            }
            // BoudingBox
            ObjetoSHP.setBoudingBox();
            while (i < 100){ // Byte Little Order - Double
                            ObjetoSHP.setFields(i, new Double(b).longBitsToDouble((
                            (ByteConvert(fis.read())) +
                            (ByteConvert(fis.read()) << 8) +
                            (ByteConvert(fis.read()) << 16) +
                            (ByteConvert((long)fis.read()) << 24) +
                            (ByteConvert((long)fis.read()) << 32) +
                            (ByteConvert((long)fis.read()) << 40) +
                            (ByteConvert((long)fis.read()) << 48) +
                            ((long)fis.read() << 56))));
                    i = i + 8;
            }
            int FileEnd = (ObjetoSHP.getFileLength() * 2);
            //Records do arquivo de origem
            ObjetoSHP.newRecordSHP();
            while (i < FileEnd){ // Records
                    Record RecAux = new Record();
                    RecAux.setRecordHeader();
                    RecAux.setRecordNumber(((fis.read() << 24) |
                            (fis.read() << 16) + (fis.read() << 8) + fis.read()));
                    RecAux.setContentLength(((fis.read() << 24) |
                            (fis.read() << 16) + (fis.read() << 8) + fis.read()));
                    i = i + 8;
                    byte VecContents[] = new byte[RecAux.getContentLength() * 2];
                    fis.read(VecContents); // lê (RecAux.getContentLength() * 2) bytes
                    RecAux.setRecordContents(VecContents);
                    i = i + (RecAux.getContentLength() * 2);
                    //Incluir o Record no vetor de registros (Records)
                    ObjetoSHP.setRecordSHP(RecAux);
            }
            fis.close();
    } catch (Exception e) {}
}
```

5.1.2 Compondo o Conteúdo do Registro

A composição do conteúdo do registro do arquivo de origem é obtida a partir das características do objeto, que estão representadas no apêndice 1 deste trabalho. O registro é um elemento do objeto de origem (*.shp), correspondendo a uma parte da camada de informação. Esse objeto pode possuir um ou mais registros. No QUADRO 02 apresenta-se o trecho de código onde é apresentada a composição do registro de tipo *Polygon*. A operação *setRecordContents*, com a implementação de código para os demais tipos, está listada no apêndice 3.

QUADRO 02 – CÓDIGO FONTE – OPERAÇÃO SETRECORDCONTENTS

```
public void setRecordContents(byte VcByte[]){
        //ShapeType
        int ShpType = (ByteConvert((int)VcByte[0]) + (ByteConvert((int)VcByte[1]) << 8) +
                        (ByteConvert((int)VcByte[2]) << 16) | ((int)VcByte[3] << 24));
        this.RecordHeader.setTypeContent(ShpType);
        int i = 4;
        double b = 0.00;
        int j = 0;
        int X, Y, W;
        byte ByteDouble[] = new byte[8];
        byte ByteInteger[] = new byte[4];
        switch (ShpType) {
                case 0:
                        //"Null Shape";
                        RecordContents = new NullShape();
                        //RecordContents <-- NullShape
                        break;
                        // case 1 ... case 4//
                case 5:
                        //"Polygon"
                        PolygonShape RecContTypePolygon = new PolygonShape();
                        while (i < 36){ //Box
                                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                                RecContTypePolygon.setBox(j, ByteConvertToDouble(ByteDouble));
                                i = i + 8;
                                j++;
                        }
                        //NumParts
                        System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                        RecContTypePolygon.setNumParts(ByteConvertToInteger(ByteInteger));
                        i = i + 4;
                        //NumPoints
                        System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                        RecContTypePolygon.setNumPoints(ByteConvertToInteger(ByteInteger));
                        i = i + 4;
                        j = 0;
                        //Parts
                        //X = 44 + 4 * NumParts
                        while (i < (44 + (4 * RecContTypePolygon.getNumParts()))){
                                System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                                RecContTypePolygon.setParts(j, ByteConvertToInteger(ByteInteger));
                                i = i + 4;
                                j++;
                        }
                        i = (44 + (4 * RecContTypePolygon.getNumParts()));
                        j = 0;
                        //Points
                        while (j < (RecContTypePolygon.getNumPoints())){
                                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                                double X1 =  ByteConvertToDouble(ByteDouble);
                                i = i + 8;
                                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                                double Y1 =  ByteConvertToDouble(ByteDouble);
                                i = i + 8;
                                RecContTypePolygon.setPoints(j, X1, Y1);
                                j++;
```

```
                    }
                    //RecordContents <-- PolygonShape
                    //RecordContents é um atributo, do tipo Object, da classe Record
                    RecordContents = new PolygonShape();
                    RecordContents = RecContTypePolygon;
                    break;

                    // case 6 ... case 31//
            default:
                    //return "Tipo não definido!";
                    RecordContents = new NullShape();
                    //RecordContents <-- NullShape
                    break;
            }
}
```

5.1.3 Compondo Objeto *.ser

Após o objeto de origem ter sido totalmente convertido para uma classe que o representa, a classe será convertida em um arquivo (*.ser), dito de intercâmbio, utilizado para distribuição no meio Internet. No QUADRO 03 é listado o código que corresponde à conversão do objeto *ObjetoSHP* em um arquivo. *ObjetoSHP* é a instância da classe que representa o objeto de origem (*.shp).

QUADRO 03 – CÓDIGO FONTE – OPERAÇÃO COMPONDOOBJETO

```
public void compondoObjeto() {
        try{
                FileOutputStream fos = new FileOutputStream(ivjTextField1.getText() + ".ser");
                ObjectOutputStream oo = new ObjectOutputStream(fos);
                oo.writeObject(ObjetoSHP);
                oo.flush();
                fos.close();
        } catch (Exception e){}
}
```

5.1.4 Distribuindo o Objeto *.ser

A distribuição do objeto *.ser é feita através de uma página em HTML, que comporta a execução de uma *Applet* Java por um navegador padrão (Netscape ou Internet Explorer). O texto padrão, da página em HTML, está descrito no QUADRO 04. A distribuição do objeto é representada pelas linhas em negrito, onde o parâmetro **QUANTARQ** identifica a quantidade de arquivos distribuídos (cada

arquivo representa uma camada de informação, e neste caso específico, o objeto visualizado terá duas camadas), e os parâmetros **ARQNOME1** e **ARQNOME2** identificam os nomes dos arquivos distribuídos. Não existe limite de quantidade de arquivos passados como parâmetro, deve-se apenas observar que o nome do parâmetro que representa o nome do arquivo é finalizado por um número que representa a ordem de leitura dos arquivos; sendo assim, para **QUANTARQ** = n teremos **ARQNOME1** até **ARQNOMEn** parâmetros de arquivos.

QUADRO 04 – CÓDIGO FONTE – HTML PADRÃO

```
<HTML>
<HEAD><TITLE></TITLE></HEAD>
<BODY>
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93" WIDTH = 620 HEIGHT = 370
codebase="http://java.sun.com/products/plugin/1.3/jinstall-13-win32.cab#Version=1,3,0,0">
<PARAM NAME = CODE VALUE = shapeobject.MapViewApplet.class >
<PARAM NAME = ARCHIVE VALUE = ShpObj.jar >
<PARAM NAME="QUANTARQ" VALUE= "2">
<PARAM NAME="ARQNOME1" VALUE= "/prcont.ser">
<PARAM NAME="ARQNOME2" VALUE= "/prindus.ser">
<PARAM NAME="type" VALUE="application/x-java-applet;version=1.3">
<PARAM NAME="scriptable" VALUE="false">
</OBJECT>
</BODY>
</HTML>
```

5.1.5 Recebendo Nome do Arquivo como Parâmetro

Quando a *Applet* Java é executada, os parâmetros passados pela página HTML serão obtidos através da rotina identificada no QUADRO 05, onde o texto em negrito representa o trecho de código responsável pela captura dos parâmetros enviados pelo navegador no ato da leitura do texto em HTML. Após ter capturado os parâmetros, o sistema poderá efetuar as operações necessárias para transferência e visualização do objeto no meio. Os recursos de processamento na manipulação do objeto serão efetuados pelo equipamento do usuário (que estiver efetuando a consulta). O servidor Internet atua como elemento de distribuição, não executando rotinas de processamento relacionadas à visualização do tema consultado, facilitando assim a interação do usuário com o objeto da consulta.

QUADRO 05 – CÓDIGO FONTE – OPERAÇÃO GETTESTAPPLET1

```
private TestApplet getTestApplet1() {
        if (ivjTestApplet1 == null) {
                try {
                        Class iiCls = Class.forName("shapeobject.TestApplet");
                        ClassLoader iiClsLoader = iiCls.getClassLoader();
                        ivjTestApplet1 = (shapeobject.TestApplet)
                                java.beans.Beans.instantiate(iiClsLoader,"shapeobject.TestApplet");
                        Integer StrToInt = new Integer(getParameter("QUANTARQ"));
                        int QtArq = StrToInt.intValue();
                        ivjTestApplet1.setVecObjSHP(QtArq);
                        for (int i = 0; i < QtArq; i++){
                                Integer IntToStr = new Integer(i+1);
                                ivjTestApplet1.setFileName(
                                        getParameter("ARQNOME" + IntToStr.toString()));
                        }
                        ivjTestApplet1.setName("TestApplet1");
                        ivjTestApplet1.setBounds(2, 47, 595, 300);
                } catch (java.lang.Throwable ivjExc) {handleException(ivjExc);}
        }
        return ivjTestApplet1;
}
```

5.1.6 Definindo Parâmetros para Visualização do Objeto

As definições de algumas características serão necessárias para visualização das camadas de informação. Estas características serão obtidas a partir do primeiro objeto distribuído, representado pelo parâmetro de nome **ARQNOME1** (do código em HTML). **CenterX** e **CenterY** representam a coordenada do canto superior esquerdo da área retangular que contém o objeto a ser visualizado (canto superior esquerdo da *Bouding Box*), **X1** representa o comprimento do lado X da área retangular que contém o objeto visualizado, **Y1** representa o comprimento do lado Y e **Diagonal** representa a diagonal da área retângula que contém o objeto. Esses elementos serão utilizados por algumas rotinas do sistema, entre elas, definição do tamanho da visualização, deslocamento da imagem visualizada, entre outras.

QUADRO 06 – CÓDIGO FONTE – OPERAÇÃO SETFILENAME

```
public void setFileName(String nome){
                //nome = NomeDoArquivo.ser
        try{
                InputStream s1 = TestApplet.class.getResourceAsStream(nome);
                ObjectInputStream oo1 = new ObjectInputStream(s1);
                //Incluir objeto no vetor de camadas
                VecObjSHP[IndexCamadas] = (ShapeSHP) oo1.readObject();
                //Se for o primeiro objeto
                if (IndexCamadas == 0) {
                        ObjetoSHP1 = new ShapeSHP();
                        ObjetoSHP1 = VecObjSHP[IndexCamadas];
                        CenterX = ObjetoSHP1.getBoudingBox(Xmin);
                        CenterY = ObjetoSHP1.getBoudingBox(Ymax);
                        X1 = ObjetoSHP1.getBoudingBox(Xmax) –
                                ObjetoSHP1.getBoudingBox(Xmin);
                        Y1 = ObjetoSHP1.getBoudingBox(Ymax) –
                                ObjetoSHP1.getBoudingBox(Ymin);
                        Diagonal = (java.lang.Math.pow(
                                java.lang.Math.pow(X1, 2.0) + java.lang.Math.pow(Y1, 2.0), 0.5));
                        //Definir o tamanho do objeto para visualização
                        setSize();
                        //Definir o centro de origem da tela para visualização do objeto
                        setCenterToPaint();
                }
                IndexCamadas = IndexCamadas + 1;
                s1.close();
                ver = 2;
        } catch (Exception e){ver = 0;}
}
```

5.1.7 Criando o Objeto para Visualização

A função **setPathToPaint** cria um objeto geométrico, obtido a partir das coordenadas geográficas da camada de informação a ser visualizada. No QUADRO 07 é representada uma parte do código dessa função, onde se pode verificar a criação de um objeto do tipo polígono, formado a partir de linhas traçadas entre as coordenadas dos pontos, obtidos através da camada de informação. A criação do objeto geométrico dependerá diretamente do tipo do objeto geográfico a ser visualizado, e poderá assumir um dos tipos descritos no apêndice 1 deste trabalho.

Supondo que o objeto a ser "desenhado" fosse um quadrado de vértices A, B, C e D, a seguinte seqüência de eventos seria executada: mover para o vértice A, traçar uma linha até o vértice B, traçar uma linha até o vértice C, traçar uma linha até o vértice D, traçar uma linha até o vértice de origem (A), e o elemento estaria geometricamente constituído.

QUADRO 07 – CÓDIGO FONTE – OPERAÇÃO SETPATHTOPAINT

```
public GeneralPath setPathToPaint(double size){
        GeneralPath PathObj = new GeneralPath();
 for (int i = 0; i < this.getSizeRecordSHP(); i++){
        int j = 0;
        int PartEnd = 0;
        Record Rec1 = new Record();
        Rec1 = (Record)this.getRecordSHP(i);
        switch (Rec1.getTypeContent()) {
                case 0:

                        // ... 0 até 5 //

                case 5:
                        //"Polygon"
                        PolygonShape ObjCont5 = new PolygonShape();
                        ObjCont5 = (PolygonShape)Rec1.getRecordContent();
                        //Número de partes do polígono
                        while (j < ObjCont5.getNumParts()){
                                if ((j + 1) < ObjCont5.getNumParts()){
                                        PartEnd = ObjCont5.getParts(j + 1);
                                }
                                else{
                                        PartEnd = ObjCont5.getNumPoints();
                                }
                                //Mover para a primeira coordenada geográfica
                                PathObj.moveTo(
                                        (float)(ObjCont5.getPointsX(ObjCont5.getParts(j)) * size),
                                        (float)(-(ObjCont5.getPointsY(ObjCont5.getParts(j)) * size)));
                                for (int j1 = (ObjCont5.getParts(j) + 1); j1 < PartEnd; j1++){
                                        //Traçar linhas a partir das coordenadas fornecidas
                                        PathObj.lineTo(((float)(ObjCont5.getPointsX(j1) * size)),
                                                        ((float)(-(ObjCont5.getPointsY(j1) * size))));
                                }
                                //Traçar linha até o ponto de origem
                                PathObj.closePath();
                                j++;
                        }
                        break;

                        // ... 6 até 31//

                default:
                        //"Tipo não definido!";
                        break;
        }
 }
 //Retornar o elemento geometricamente constituído
 return(PathObj);
}
```

5.1.8 Visualizando o Objeto

A rotina do QUADRO 08 representa a criação da interface gráfica responsável pela visualização das camadas de informação. ***GeneralPath*** é a classe Java responsável por compor um objeto a partir de elementos geométricos (pontos, linha e polígonos), será esse o objeto efetivamente "desenhado" na tela da *Applet*.

O deslocamento do centro da origem da tela de visualização deve-se ao fato de que na tela de visualização os eixos X e Y são positivos (representando um único quadrante), enquanto em um plano de coordenada geográfica os eixos X e Y podem receber valores negativos e positivos (possuindo quatro quadrantes).

QUADRO 08 – CÓDIGO FONTE – OPERAÇÃO PAINT

```
public void paint(Graphics g) {
        //Criar a área gráfica 2D
        Graphics2D g2 = (Graphics2D)g;
        GeneralPath PathObj1 = new GeneralPath();
        if (ver != 0){
                //Deslocar o centro da origem da tela de visualização
                g2.translate(CenterToPaintX, CenterToPaintY);
                for (int i = 0; i < QtdeCamadas; i++){
                        //Criar o objeto a ser desenhado
                        PathObj1 = VecObjSHP[i].setPathToPaint(Size);
                        //Se não for um objeto constituído por linhas
                        if (VecObjSHP[i].getShapeTypeCode() != 3 &&
                                VecObjSHP[i].getShapeTypeCode() != 13 &&
                                VecObjSHP[i].getShapeTypeCode() != 23){
                                //Preencher o objeto com uma cor
                                g2.setPaint(Cor);
                                g2.fill(PathObj1);
                        }
                        //Desenhar o objeto
                        g2.setColor(Color.black);
                        g2.draw(PathObj1);
                }
        }
}
```

6 RESULTADO DO TRABALHO

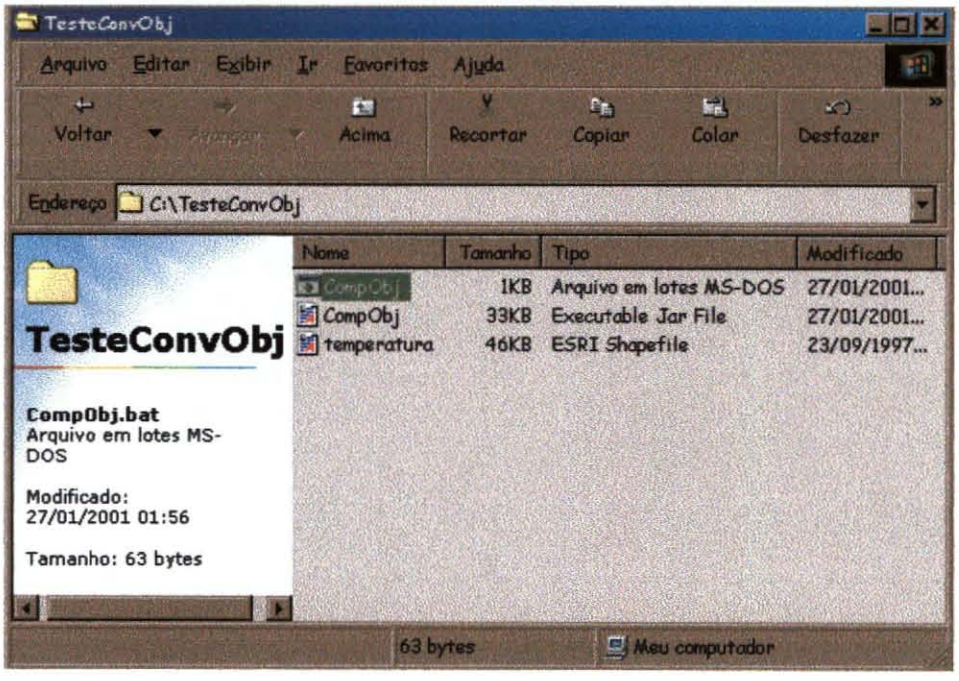
6.1 CRIANDO UM SISTEMA BÁSICO DE CONSULTA

Apresentam-se na seqüência os processos necessários para criação de um sistema básico de consulta e visualização de camadas de informação, desde a decomposição do objeto de origem até a sua efetiva visualização.

6.1.1 Decomposição e Composição das Camadas de Informação

Para decomposição e composição das camadas de informação será necessária a utilização dos arquivos *CompObj.jar*, *CompObj.bat* e os arquivos do tipo *.shp (que neste caso é o arquivo *temperatura.shp*), que serão distribuídos pelo usuário. Para o exemplo de implementação de um sistema básico de consulta, foi criado o subdiretório *TesteConvObj* (podendo possuir qualquer outro nome), apresentado pela FIGURA 24, que contém os arquivos necessários para a primeira parte do processo de criação do sistema de consulta.

FIGURA 24 – SEPARANDO ELEMENTOS NECESSÁRIOS



6.1.2 Decompondo o Objeto *temperatura.shp* e Compondo o Objeto *prtemp.ser*

O próximo passo será a execução do arquivo *CompObj.bat*, que possibilitará a interpretação do código Java do aplicativo para conversão do objeto *.shp. O aplicativo será interpretado através da seguinte linha de comando:

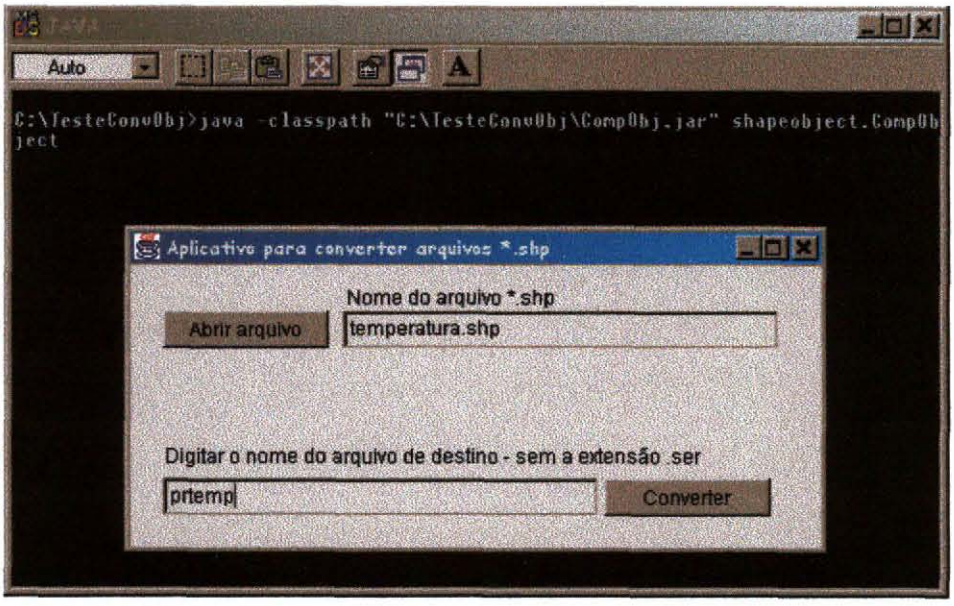
C:\TesteConvObj> java –classpath "C:\TesteConvObj\CompObj.jar" shapeobject.CompObject

O usuário deverá atentar para o caminho "*C:\TesteConvObj\CompObj.jar*", que corresponde à localização do arquivo *CompObj.jar*. Caso seja diferente, o arquivo *CompObj.bat* poderá ser alterado através de qualquer editor de textos do tipo *.txt. Também é necessária a existência do interpretador Java no equipamento utilizado para conversão dos objetos.

Aconselha-se utilizar um padrão de oito caracteres para o nome do arquivo de intercâmbio (*.ser), possibilitando, assim, compatibilidade com diferentes tipos de sistemas operacionais utilizados por servidores de acessos à Internet.

O aplicativo para converter arquivos *.shp possui dois campos: um para selecionar o objeto a ser decomposto (*.shp) e outro onde será digitado no nome do objeto de destino (*.ser). Na FIGURA 25 é apresentada interface do sistema, no momento da decomposição do objeto *temperatura.shp* e composição do objeto *prtemp.ser*.

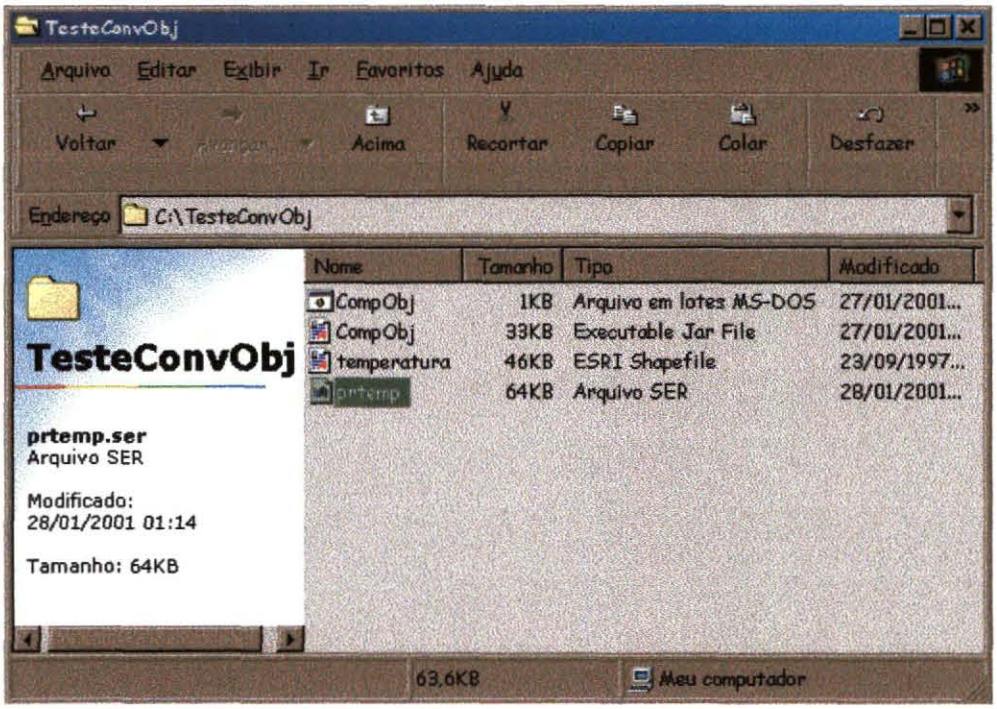
FIGURA 25 – DECOMPOSIÇÃO E COMPOSIÇÃO DE OBJETOS



6.1.3 O Arquivo de Intercâmbio Criado

Para este exemplo, o arquivo de intercâmbio criado a partir do objeto *temperatura.shp*, foi o arquivo nomeado *prtemp.ser*, conforme apresentado na FIGURA 26.

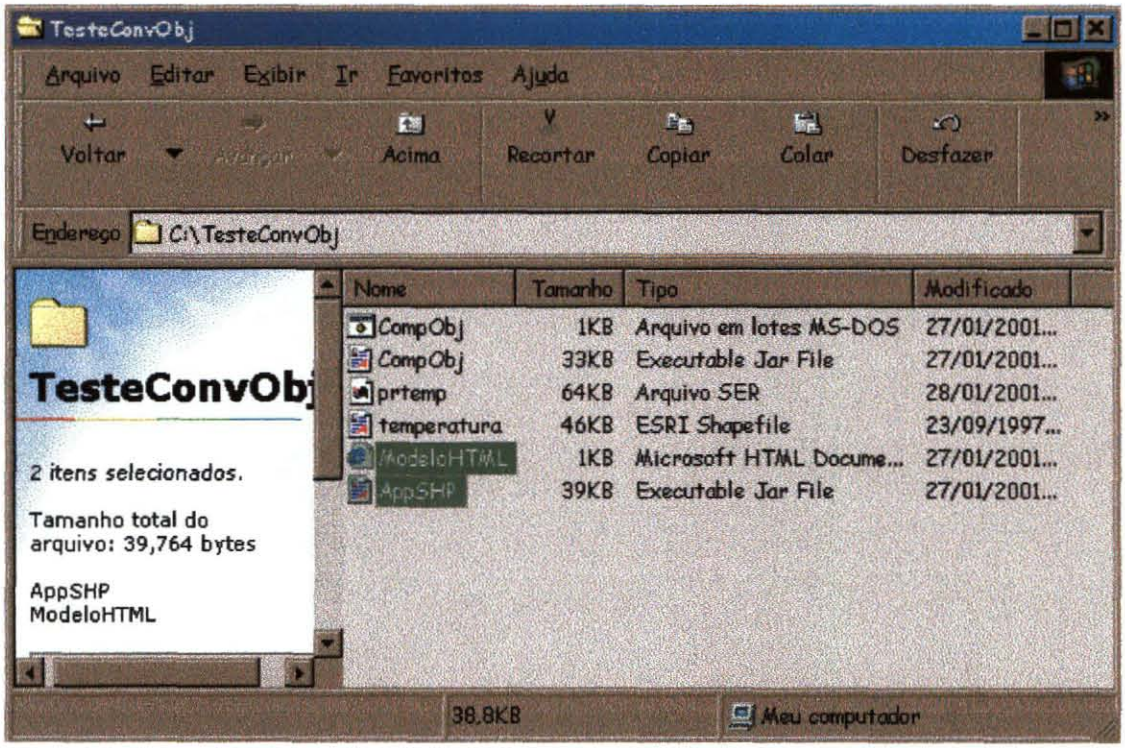
FIGURA 26 – A CRIAÇÃO DO ARQUIVO *.SER



6.1.4 Distribuindo o Objeto Criado

Para distribuição do objeto criado será necessária a utilização dos arquivos *AppSHP.jar* (que contém a *Applet* de visualização) e *ModeloHTML.html* (que corresponde ao modelo de página em HTML, a qual deverá ser utilizada para ativar a *Applet* de visualização), que foram incluídos no diretório de trabalho, conforme mostra a FIGURA 27.

FIGURA 27 – ATIVANDO RECURSOS DE DISTRIBUIÇÃO DE OBJETOS



6.1.5 O Código em HTML

Será necessária a inclusão dos parâmetros no código em HTML, contido no arquivo *ModeloHTML.html*, conforme mostrado no QUADRO 09.

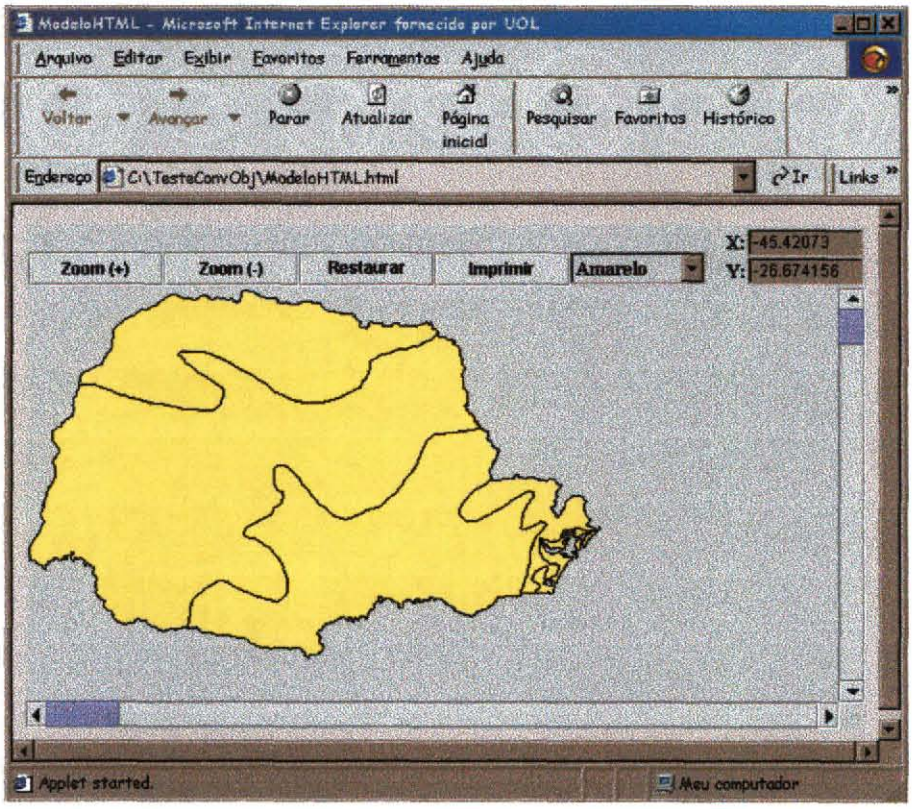
QUADRO 09 – CÓDIGO FONTE – PÁGINA PADRÃO EM HTML

```
<HTML>
<HEAD>
<TITLE></TITLE>
</HEAD>
<BODY>
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
WIDTH = 620 HEIGHT = 370
codebase="http://java.sun.com/products/plugin/1.3/jinstall-13-win32.cab#Version=1,3,0,0">
<PARAM NAME = CODE VALUE = shapeobject.MapViewApplet.class >
<PARAM NAME = ARCHIVE VALUE = AppSHP.jar >
<PARAM NAME="QUANTARQ" VALUE= "1">
<PARAM NAME="ARQNOME1" VALUE= "/prtemp.ser">
<PARAM NAME="type" VALUE="application/x-java-applet;version=1.3">
<PARAM NAME="scriptable" VALUE="false">
</OBJECT>
</BODY>
</HTML>
```

6.1.6 Testando a Visualização do Objeto

Para testar a visualização do objeto, basta acessar o arquivo ModeloHTML.html (com o código em HTML devidamente alterado, conforme item 6.1.5) através de um navegador (Internet Explorer ou Netscape), que procederá a execução da *Applet* de visualização, conforme ilustra a FIGURA 28.

FIGURA 28 – TESTE DE VISUALIZAÇÃO



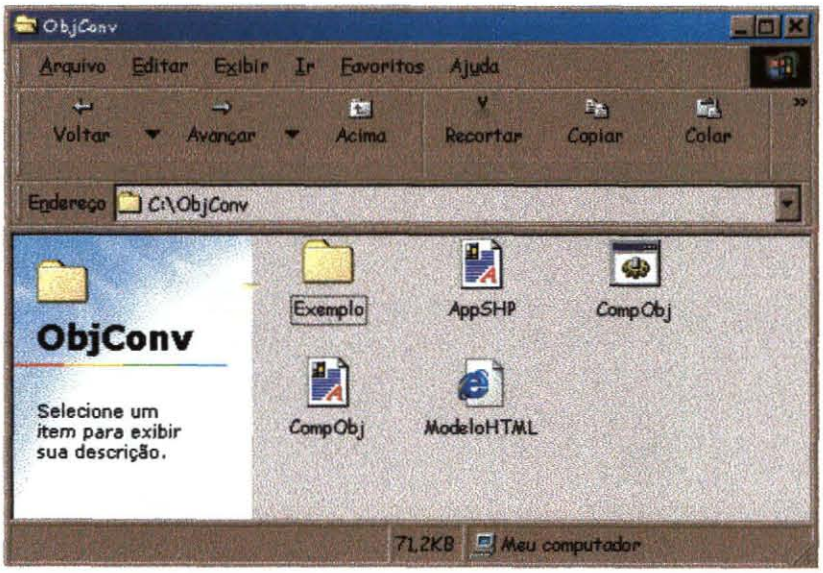
6.2 DISTRIBUIÇÃO DO PACOTE DE APLICATIVOS

Foi criado um pacote para distribuição do aplicativo, que contém as classes de decomposição e composição do objeto, a *Applet* Java responsável pela visualização das camadas de informação, um modelo de página em HTML utilizado para distribuição das camadas de informação e um exemplo da utilização dos recursos disponibilizados. O pacote é estruturado conforme TABELA 25.

TABELA 25 – CONTEÚDO DO PACOTE DE DISTRIBUIÇÃO

Arquivo	Tipo	Descrição
AppSHP.jar	Arquivo compactado *.jar	Contém as classes Java responsáveis pela visualização das camadas de informação.
CompObj.bat	Arquivo .bat	Contém a linha de comando que executará o aplicativo para decomposição e composição dos objetos a serem distribuídos.
CompObj.jar	Arquivo compactado *.jar	Contém as classes Java responsáveis pela decomposição e composição dos objetos a serem distribuídos.
ModeloHTML.html	Arquivo de código HTML.	Contém o código necessário para execução da Applet Java usada para visualização de camadas de informação.
Exemplo	Subdiretório	Contém um conjunto de arquivos criados para servir de exemplo de distribuição e visualização de camadas de informação.

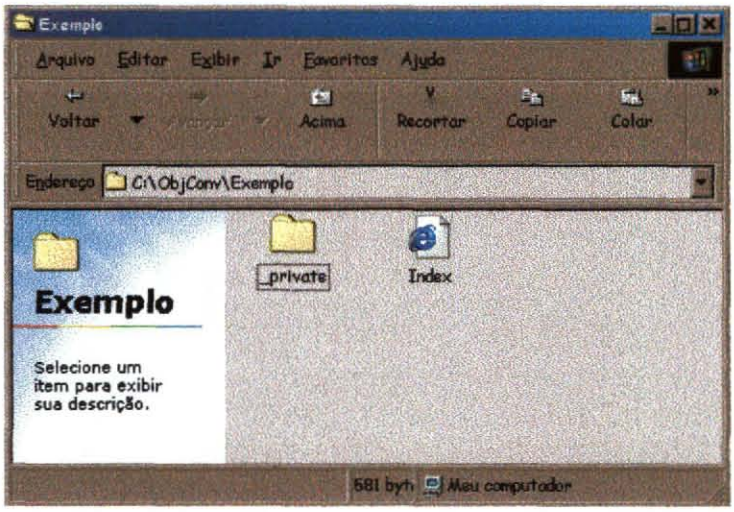
FIGURA 29 – PACOTE PARA DISTRIBUIÇÃO



6.2.1 Subdiretório Exemplo

O subdiretório Exemplo, contido no pacote de distribuição, contém o conjunto básico de recursos para distribuição e visualização de camadas de informação para um exemplo em específico. Este subdiretório é a representação exata do diretório que deve ser criado no servidor de serviços Internet/Intranet para distribuição dos objetos a serem visualizados, contendo o subdiretório *_private* e a página em HTML, nomeada *Index.html*, conforme ilustrado na FIGURA 30.

FIGURA 30 – SUBDIRETÓRIO EXEMPLO



6.2.1.1 Objetos distribuídos pelo exemplo

O subdiretório *_private* contido no subdiretório Exemplo contém o conjunto de páginas HTML, a *Applet* de visualização e os objetos a serem distribuídos (*.ser), conforme ilustra a FIGURA 31, correspondendo ao conjunto de elementos necessários para acesso ao sistema de consulta, criado para este exemplo.

FIGURA 31 – SUBDIRETÓRIO _PRIVATE

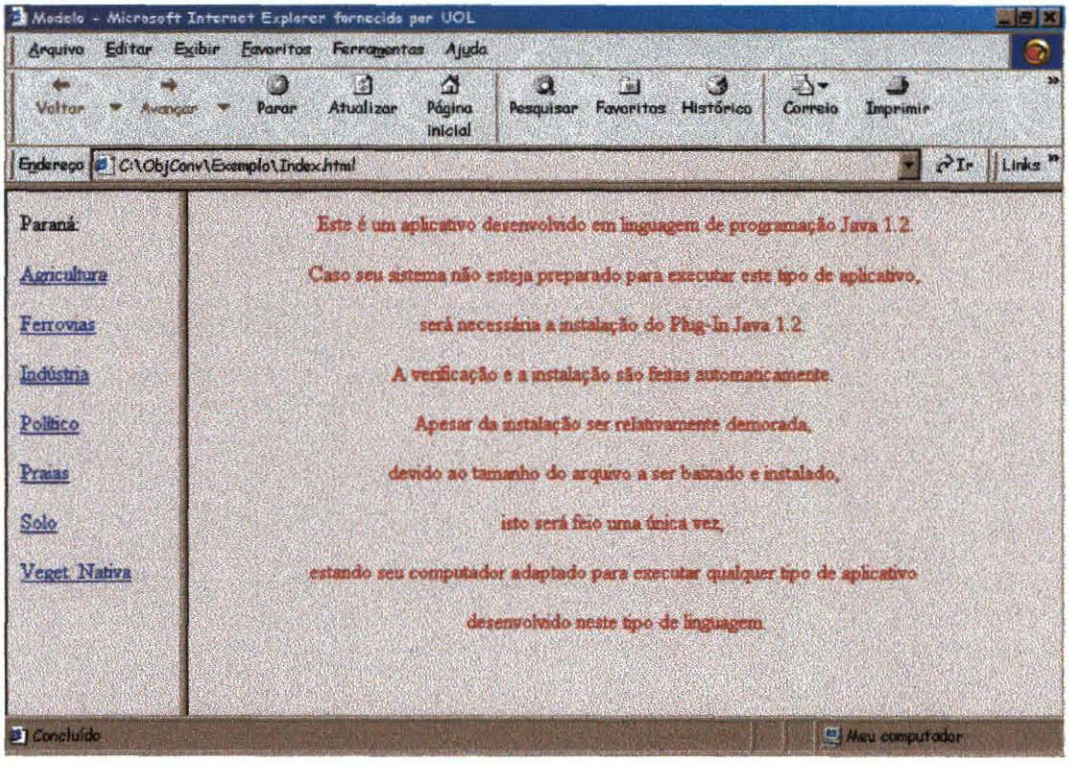
6.3 APRESENTANDO O EXEMPLO

Foi criado um modelo de distribuição e visualização de camadas de informação para um tema em específico. O exemplo adotou um conjunto de camadas de informação, relacionado ao tema Paraná, que representam alguns assuntos relacionados a esse tema. Na seqüência será apresentada a constituição do modelo de consulta criado para este exemplo.

6.3.1 Página Inicial

Na FIGURA 32, mostra-se a página inicial do exemplo. Ela serve de abertura para o sistema de visualização de camadas de informação, e fornece uma breve explicação referente à necessidade de instalação do *Plug-In* Java, caso o sistema do usuário não esteja adaptado para utilização de recursos Java 1.2. Alguns navegadores não estão ainda habilitados para execução de sistemas desenvolvidos na versão 1.2 do Java. A parte esquerda da página, ilustrada na FIGURA 32, contém os assuntos relacionados ao tema, que poderão ser visualizados.

FIGURA 32 – MODELO – PÁGINA INICIAL



6.3.1.1 A Instalação do *Plug-In* Java

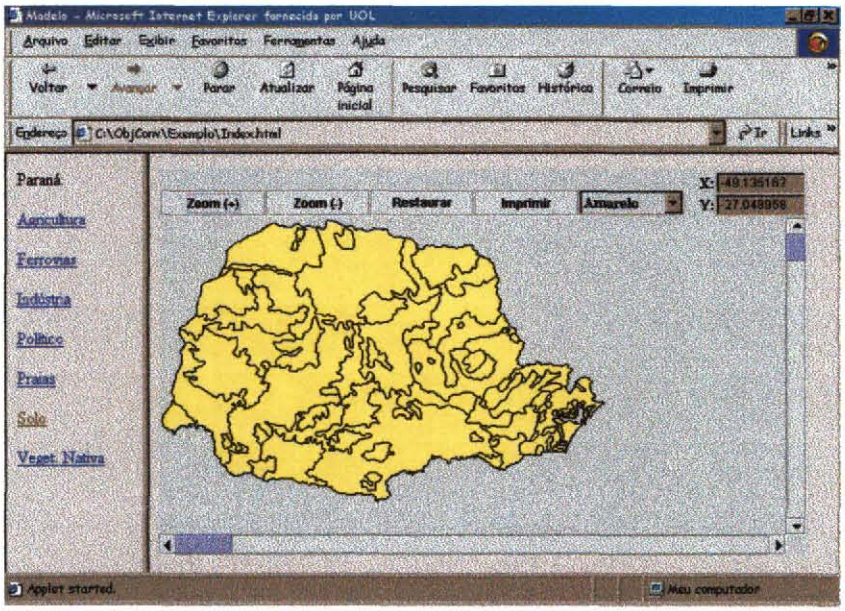
No item 6.3.1 foi mencionada a falta de habilitação, por alguns navegadores, de recursos da versão 1.2 do Java. Para solucionar este problema, o sistema verificará automaticamente a necessidade de instalação do referido *Plug-In*. Sendo necessário, o usuário poderá ou não optar pela instalação, porém, existem dois critérios a considerar com relação a essa opção do usuário:

a) o usuário, optando pela instalação do *Plug-In*, o processo será um pouco demorado, em função do tamanho do arquivo de instalação, porém o processo é totalmente automatizado, e, após a instalação, o navegador do usuário estará totalmente compatível com a versão 1.2 do Java, podendo executar qualquer recurso desta versão. Este processo é feito uma única vez;
b) o usuário, não optando pela instalação do *Plug-In*, o navegador não estará compatível com a versão 1.2 do Java, impossibilitando desta maneira a execução da *Applet* de visualização, ou de qualquer outro recurso desenvolvido em Java 1.2.

6.3.2 Selecionando o Assunto Solo do Tema Paraná

Quando é selecionado o item **Solo**, a *Applet* Java será executada, fornecendo a visualização das camadas de informação relacionadas ao assunto selecionado. A visualização é apresentada na FIGURA 33.

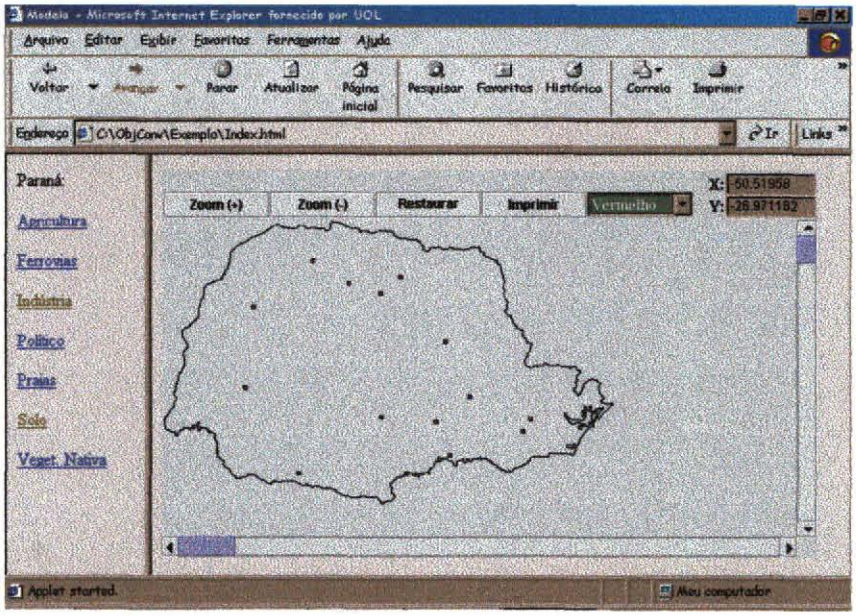
FIGURA 33 – MODELO – SELECIONANDO O ASSUNTO SOLO



6.3.3 Selecionando o Assunto Indústria do Tema Paraná

Quando é selecionado o item **Indústria**, a *Applet* Java será executada, fornecendo a visualização das camadas de informação relacionadas ao assunto selecionado, conforme ilustra a FIGURA 34. E assim ocorre, sucessivamente, para os demais assuntos disponibilizados.

FIGURA 34 – MODELO – SELECIONANDO O ASSUNTO INDÚSTRIA



7 CONCLUSÃO

7.1 RESULTADO FINAL

O projeto provou que é possível a distribuição de objetos dentro do meio Internet, sem grandes investimentos de recursos financeiros ou vinculações comerciais. Atendeu ao seu objetivo principal, produzindo uma ferramenta de fácil utilização para a visualização de camadas de informação no ambiente Internet. Criou fonte de estudos com relação à transformação de arquivos em objetos e a distribuição desses objetos no meio Internet.

7.2 EVOLUÇÃO DA FERRAMENTA

O projeto foi apresentado em sua fase inicial, e sugere-se a implementação de novos recursos, possibilitando a evolução da ferramenta e agregando novos mecanismos para consulta de informações geográficas.

7.2.1 Sugestão de Implementação de Recursos

Sugeri-se a implementação de novos recursos com o fim de intercâmbio entre o usuário e o sistema, visando à obtenção de informações fornecidas pelo objeto distribuído, entre eles pode-se citar:

- navegação entre camadas de um mesmo tema;
- navegação entre partes de uma camada;
- calculo e visualização de distância (em metros) entre pontos.

Sugeri-se também a implementação de ferramenta de visualização para o sistema de decomposição de composição do objeto de distribuição, facilitando assim o trabalho do administrador na composição dos temas a serem visualizados.

7.3 IMPORTÂNCIA CIENTÍFICA

Pode-se citar dois aspectos de relevância científica apresentados neste trabalho, sendo eles:

1) proposta de distribuição de objetos no meio Internet, através da representação por classes implementadas em linguagem de programação orientadas a objetos;
2) desenvolvimento de soluções com o intuito de popularização das tecnologias de distribuição da informação, criando recursos de fácil utilização e contribuindo para inovação tecnológica na produção de mecanismo de disseminação da informação no meio Internet.

7.3.1 Mostra de Talentos Científicos

Este trabalho foi submetido e aprovado para Mostra de Talentos Científicos Gis Brasil 2001, concorrendo ao PRÊMIO GIS BRASIL DE INCENTIVO: DA PESQUISA PARA A PRÁTICA.

REFERÊNCIAS BIBLIOGRÁFICAS

BEN-NATAN, R. **Objects on the Web – Design, Building, and Deploying Object-Oriented Applications for the Web**. New York : McGraw-Hill, 1997.

DARBAR, P. M.;POWERS, M. E.; NJUGUNA, Kamau. **Application of Graphical User Interface as a Tool in the Study of Pipeline Safety**. ESRI Internacional User Conference, 1997. Disponível em: <http://www.esri.com/library/userconf/proc97/proc97/to300/pap259/p259.htm> Acesso em: 23 set. 2000.

ESRI White Paper. **ESRI Shapefile Technical Description**. United States of America : Environmental Systems Research Institute, 1998

FONSECA, F. T.; DAVIS JUNIOR, C. A. **Geoprocessamento e Internet: Cenário Atual e Perspectivas**. In: GISBRASIL 1998. Disponível em:< http://www.fatorgis.com.br/artigos/gis/geo_int/geo_int.htm> Acessado em: 27 dez. 2000.

FOOTE, K. E.; LYNCH, M. **The Geographer's Craft Project.** Departamento de Geografia da Universidade do Texas. Disponível em: <http://www. http://www.prudente.unesp.br/dcartog/arlete/gis/intro_t.htm> Acessado em: 08 mar. 2001.

IBOPE. **8ª edição da Pesquisa Internet POP**. Disponível em: <http://www.ibope.com.br/digital/produtos/internetpop/8pop.html> Acesso em: 25 nov. 2000.

LOWE, J. W. **Advanced Internet Map Server Techniques for Building Intuitive Web Sites for a Nontechnical User Base**. Disponível em: <http://www.esri.com/library/userconf/proc99/proceed/abstracts/a598.html> Acessado em: 20 dez. 2000.

LU, X. **Web-Based GIS Migration: How to Get Your GIS Data and Applications to Your Users and off All Those Computers**. Disponível em: <http://www.esri.com/library/userconf/proc00/professional/abstracts/a708.html> Acessado em: 20 dez. 2000.

MARSHALL, J. **Developing Internet-Based GIS Applications**. Disponível em: <http://www.esri.com/library/userconf/proc00/professional/abstracts/a405.html> Acessado em: 20 dez. 2000.

MONZON, P. **An Internet GIS Application for the Economic Development Department of the City of Pittsburg**. Disponível em: <http://www.esri.com/library/userconf/proc00/professional/abstracts/a453.html> Acessado em: 20 dez. 2000.

NUA Internet Surveys, **How Many Online?**. Disponível em: <http://www.nua.ie/surveys/how_many_online/>. Acesso em: 25 nov. 2000.

PRESSMAN, R. S. **Engenharia de Software**. São Paulo : Makron Books, 1995.

PUTZ, S. **Interactive Information Services Using World-Wide Web Hypertext**. Disponível em: <http://www.parc.xerox.com/istl/projects/www94/iissuwwwh.html> Acessado em: 18 dez. 2000.

RUMBAUGH, J.; BLAHA, M.; PREMERLANI, W.; EDDY, F.; LORENSEN, W. ***Modelagem e Projetos Baseados em Objetos***. 1ª Edição. Rio de Janeiro : Campus, p: 64, 1994. ISBN 85-7001-841-X.

SCHALLER, J. **Gis on the Internet and Environmental Information and Planning**. Disponível em: <http://www.esri.com/library/userconf/europroc98/proc/idp27.html> Acessado em: 20 dez. 2000.

SVENSSON, B. **User Focus for a Multimap Web Site: Improving the Usability of an Online Atlas**. Disponível em: <http://www.esri.com/library/userconf/proc00/professional/abstracts/a761.html> Acessado em: 20 dez. 2000.

DOCUMENTOS CONSULTADOS

BARBOSA, R. W. **Gerando Java com VisualAge versão 2.0**. Rio de Janeiro : Brasport, 1999. ISBN 85-7452-001-2.

BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. ***The UML Specification Documents***. Santa Clara, CA.: Rational Software Corp, 1997.

CARR, D. F. **Developing Web Apps: Not the Right Job for IT**. Internet World, 4 de maio, 1998.

COAD, P.; YOURDON, E.. ***Análise Baseada em Objetos***. Campus, 1996. ISBN 85-352-0042-8.

DAVIS, D. **GIS for Everyone. Redlands**, CA, United States of America: ESRI Press, 1999.

ERIKSSON, H.; PENKER, M. **UML Toolkit**. United States of America : Wiley Computer Publishing, 1998.

FOWLER, M.; SCOTT, K. **UML Distilled : Applying the Standard Object Modeling Language**. United States of America : Addison-Wesley, 1998.

JUBIN, H.; Jalapeño Team. **JAVA BEANS by Example**. Prentice Hall. 1997. ISBN 0-13-790338-3.

KAWAMOTO, Wayne. **A Paper Trail to the Web**. PC Computing, 12 de julho, 1999.

LARMAN, Craig. **Applying UML and Patterns – An Introduction to Object-Oriented *Analysis and Design***. Prentice Hall. 1999. ISBN 0-13-748880-7.

NAISBITT, J. **Paradoxo Global : Quanto maior a economia mundial, mais poderosos são os seus protagonistas menores: nações, empresas e indivíduos**. Rio de Janeiro: Campus, 1994. p. 53-113.

OLIVEIRA, S. L. de. **Tratado de Metodologia Científica : Projetos de Pesquisa, TGI, TCC, Monografias, Dissertações e Teses**. São Paulo : Pioneira, 1997. ISBN 85-221-0070-5.

PATTERSON, D. A.; HENNESSY, J. L. **Computer Organization & Design – The Hardware/Software Interface**. United States of America : Morgan Kaufmann Publishe, 1997.

SHELL, M. **Using Defined Layers to Display and Query**. ESRI ArcUser Magazine. United States of America: Julho-Setembro, 1998.

SUN Microsystems. **Java 2D API WHITE PAPER**. Disponível em: <http://java.sun.com/products/java-media/2D/whitepaper.html > Acesso em: 02 nov. 2000.

SUN Microsystems. **Accessing Resources in a Location-Independent Manner**. Disponível em: <http://www.javasoft.com/products/jdk/1.1/docs/guide/misc/resources.html> Acesso em: 06 nov. 2000.

SUN Microsystems. **Java Plug-in Software HTML Converter Features**. Disponível em: <http://java.sun.com/products/plugin/1.3/features.html > Acesso em: 03 nov. 2000.

SUN Microsystems. **JAVA PLUG-IN 1.3 SOFTWARE TROUBLESHOOTING FAQ**. Disponível em: <http://java.sun.com/products/plugin/1.3/troubleshooting.faq.html > Acesso em: 03 nov. 2000.

SUN Microsystems. **The Java 2 Runtime Environment Notes for Developers**. Disponível em: <http://java.sun.com/j2se/1.3/runtime.html > Acesso em: 03 nov. 2000.

SUN Microsystems. **JAVA PLUG-IN 1.3 Demonstration Applets**. Disponível em: <http://192.9.48.9/products/plugin/1.3/demos/applets.html > Acesso em: 03 nov. 2000.

SUN Microsystems. **Java Development Kit (JDK) 1.1.x – Signed Applet Example**. Disponível em: <http://java.sun.com/security/signExample/index.html > Acesso em: 05 nov. 2000.

SUN Microsystems. **JAVA PLUG-IN 1.3 DEVELOPER INFORMATION FAQ**. Disponível em: <http://java.sun.com/products/plugin/1.3/devinfo.faq.html > Acesso em: 03 nov. 2000.

SUN Microsystems. **JAVA PLUG-IN 1.3 BASIC INFORMATION FAQ**. Disponível em: <http://java.sun.com/products/plugin/1.3/basics.faq.html > Acesso em: 03 nov. 2000.

SUN Microsystems. **JAVA PLUG-IN PRODUCT**. Disponível em: <http://java.sun.com/products/plugin/ > Acesso em: 03 nov. 2000.

SUN Microsystems. **Using javakey**. Disponível em: <http://java.sun.com/security/usingJavakey.html > Acesso em: 05 nov. 2000.

SUN Microsystems. **Frequently Asked Questions Java 2D**. Disponível em: <http://java.sun.com/products/java-media/2D/forDevelopers/java2dfaq.html > Acesso em: 02 nov. 2000.

SUN Microsystems. **Java 2D API DATASHEET**. Disponível em: <http://java.sun.com/products/java-media/2D/datasheet.html > Acesso em: 02 nov. 2000.

SUN Microsystems. **Java 2D API Sample Page**. Disponível em: <http://java.sun.com/products/java-media/2D/samples/index.html> Acesso em: 02 nov. 2000.

SUN Microsystems. **Java Plug-in 1.3 Documentation**. Disponível em: <http://java.sun.com/products/plugin/1.3/docs/index.html> Acesso em: 04 nov. 2000.

SUN Microsystems. **Java Plug-in 1.3 Software Overview**. Disponível em: <http://java.sun.com/products/plugin/1.3/overview.html> Acesso em: 04 nov. 2000.

SUN Microsystems. **USING THE HTML CONVERTER**. Disponível em: <http://java.sun.com/products/plugin/1.3/docs/htmlconv.html> Acesso em: 04 nov. 2000.

SUN Microsystems. **Java 2 Runtime Environment Version 1.3.0 Installation Notes**. Disponível em: <http://java.sun.com/j2se/1.3/jre/install-windows.html> Acesso em: 02 nov. 2000.

SUN Microsystems. **JAVA PLUG-IN HTML SPECIFICATION**. Disponível em: < http://java.sun.com/products/plugin/1.3/docs/tags.html> Acesso em: 04 nov. 2000.

SUN Microsystems. **Programmer's Guide to the Java 2d API – Enhanced Graphics and Imaging for Java**. United States of America : SUN Microsystems, Inc. Business, 03 de maio,1999.

UNIVERSIDADE FEDERAL DO PARANÁ. Sistemas de Bibliotecas. **Normas para Apresentação de Documentos Científicos**, 2000, pt. 2, 6, 7 e 8.

Vincent Hardy. **Java 2D API Graphics**. Disponível em: <http://developer.java.sun.com/developer/Books/2dgraphics/> Acesso em: 02 nov. 2000.

WANG, C. B. **Techno Vision II**. Tradução Maria Nolf e Miguel Cabrera; revisão técnica José Davi Furlan. São Paulo: Makron Books, 1998. p. 85 - 136.

WU, Y. **Automated Map Digitizing: Developments in Raster to Vector Conversion Technology**. Proceedings of the Seventeenth Annual ESRI User Conference,1997.

APÊNDICES

Apêndice 1 - Especificação Técnica do Objeto *.shp

Sumário

Lista de Ilustrações

1 ESPECIFICAÇÃO TÉCNICA DO OBJETO

A descrição técnica do objeto tem como objetivo fornecer seus elementos essências, e com isto promover subsídios para representa-lo por uma estrutura lógica constituída de classes de objetos.

A especificação técnica do objeto foi obtida utilizando-se como fonte de referência à publicação: *"ESRI Shapefile Techinical Description"* , descrita no capítulo Referências Bibliográficas.

2 O OBJETO *.shp

Trata-se de um arquivo binário vetorial, que contém coordenadas geográficas de um determinado elemento contido dentro dos limites do espaço geográfico terrestre. É representado pela extensão de arquivo *.shp.

2.1 CONSTITUIÇÃO FÍSICA

Um arquivo *.shp é constituído por um cabeçalho de arquivo seguido por um determinado número de registros, sendo cada registro dividido em duas partes: cabeçalho do registro e conteúdo do registro.

FIGURA 01 – CONSTITUIÇÃO FÍSICA DO ARQUIVO *.SHP

Cabeçalho do arquivo	
Cabeçalho do registro	Conteúdo do registro
Cabeçalho do registro	Conteúdo do registro
...	
Cabeçalho do registro	Conteúdo do registro

2.2 ESPECIFICAÇÃO DO CABEÇALHO DO ARQUIVO

O cabeçalho do arquivo possui tamanho fixo de 100 *bytes*, e é constituído conforme tabela abaixo:

TABELA 01 – CABEÇALHO DO ARQUIVO

Posição	Campo	Valor	Tipo de dado	Ordem do Byte
Byte 0	File Code	9994	*Integer*	Alta
Byte 4	Não usado	0	*Integer*	Alta
Byte 8	Não usado	0	*Integer*	Alta
Byte 12	Não usado	0	*Integer*	Alta
Byte 16	Não usado	0	*Integer*	Alta
Byte 20	Não usado	0	*Integer*	Alta
Byte 24	Tamanho do arquivo	Tamanho	*Integer*	Alta
Byte 28	Versão	1000	*Integer*	Baixa
Byte 32	Tipo	Shape Type	*Integer*	Baixa
Byte 36	Bounding Box	Xmin	*Double*	Baixa
Byte 44	Bounding Box	Ymin	*Double*	Baixa
Byte 52	Bounding Box	Xmax	*Double*	Baixa
Byte 60	Bounding Box	Ymax	*Double*	Baixa
Byte 68*	Bounding Box	Zmin	*Double*	Baixa
Byte 76*	Bounding Box	Zmax	*Double*	Baixa
Byte 84*	Bounding Box	Mmin	*Double*	Baixa
Byte 92*	Bounding Box	Mmin	*Double*	Baixa

Notas: (*) Valor 0.0 se não mensurado.
O valor do campo "Tamanho do arquivo" é o tamanho total do arquivo em palavras de 16 bits.

O campo "*Bounding Box*" representa o menor retângulo ortogonal para as coordenadas *X* e *Y*, e potencialmente para *M* e *Z*, que contém toda a imagem representada pelo arquivo. Se o arquivo não possui registros, os valores de *Xmin*, *Ymin*, *Xmax*, *Ymax* será 0.0.

2.2.1 Valores para Shape Type

O campo "Tipo" (*Shape Type*) pode assumir um dos valores constantes na TABELA 02, que é apresentada na seqüência:

TABELA 02 – VALORES ASSUMIDOS PELO TIPO SHAPETYPE

Valor	Tipo
0	*Null Shape*
1	*Point*
3	*PolyLine*
5	*Polygon*
8	*MultiPoint*
11	*PointZ*
13	*PolyLineZ*
15	*PolygonZ*
18	*MultiPointZ*
21	*PointM*
23	*PolyLineM*
25	*PolygonM*
28	*MultiPointM*
31	*MultiPatch*

2.3 ESPECIFICAÇÃO DO CABEÇALHO DO REGISTRO

O cabeçalho de cada registro possui tamanho fixo de 8 *bytes* e é responsável por informar os valores para: Número do registro e Tamanho do conteúdo do registro. Na tabela, apresentada na seqüência, será descrita a constituição do cabeçalho do registro:

TABELA 03 – CABEÇALHO DO REGISTRO

Posição	Campo	Valor	Tipo de dado	Ordem do Byte
Byte 0	Número do registro	Número	*Integer*	Alta
Byte 4	Tamanho do conteúdo	Tamanho	*Integer*	Alta

Notas: O campo "Número do registro" possui o valor 1 para o primeiro registro.
Para cada registro, o campo "Tamanho do conteúdo" informa o tamanho do conteúdo do registro em palavras de 16 *bits*, deste modo cada registro contribui com 4 + "Tamanho do conteúdo" (em palavras de 16 *bits*), para o tamanho total do arquivo gravado no *Byte 24* no cabeçalho do arquivo, sendo 4 o tamanho do cabeçalho do registro.

2.4 CONTEÚDO DO REGISTRO

O conteúdo do registro é constituído pelo tipo (*Shape Type*), representado por um dos valores descritos na TABELA 02, seguido por um conjunto de dados geométricos. O tamanho do conteúdo do registro depende do número de partes e vértices da camada de informação especificada pelo registro. O formato do conjunto

de dados geométricos dependerá diretamente do tipo, podendo assim, assumir 14 formatos diferentes.

2.4.1 Especificação do Conteúdo do Registro de Tipo Null Shape

TABELA 04 – TIPO NULL SHAPE

Posição	Campo	Valor	Tipo do dado	Tamanho	Ordem do Byte
Byte 0	Tipo	0	*Integer*	1	Baixa

2.4.2 Especificação do Conteúdo do Registro de Tipo Point

TABELA 05 – TIPO POINT

Posição	Campo	Valor	Tipo do dado	Tamanho	Ordem do Byte
Byte 0	Tipo	1	*Integer*	1	Baixa
Byte 4	X	X	*Double*	1	Baixa
Byte 12	Y	Y	*Double*	1	Baixa

2.4.3 Especificação do Conteúdo do Registro de Tipo MultiPoint

TABELA 06 – TIPO MULTIPOINT

Posição	Campo	Valor	Tipo do dado	Tamanho	Ordem do Byte
Byte 0	Tipo	8	*Integer*	1	Baixa
Byte 4	Box	Box	*Double*	4	Baixa
Byte 36	NumPoints	NumPoints	*Integer*	1	Baixa
Byte 40	Points	Points	*Points*	NumPoints	Baixa

2.4.4 Especificação do Conteúdo do Registro de Tipo PolyLine

TABELA 07 – TIPO POLYLINE

Posição	Campo	Valor	Tipo do dado	Tamanho	Ordem do Byte
Byte 0	Tipo	3	*Integer*	1	Baixa
Byte 4	Box	Box	*Double*	4	Baixa
Byte 36	NumParts	NumParts	*Integer*	1	Baixa
Byte 40	NumPoints	NumPoints	*Integer*	1	Baixa
Byte 44	Parts	Parts	*Integer*	NumParts	Baixa
Byte X	Points	Points	*Point*	NumPoints	Baixa

Nota: X = 44 + (4 * NumParts).

2.4.5 Especificação do Conteúdo do Registro de Tipo Polygon

TABELA 08 – TIPO POLYGON

Posição	Campo	Valor	Tipo do dado	Tamanho	Ordem do Byte
Byte 0	Tipo	5	*Integer*	1	Baixa
Byte 4	Box	Box	*Double*	4	Baixa
Byte 36	NumParts	NumParts	*Integer*	1	Baixa
Byte 40	NumPoints	NumPoints	*Integer*	1	Baixa
Byte 44	Parts	Parts	*Integer*	NumParts	Baixa
Byte X	Points	Points	*Point*	NumPoints	Baixa

Nota: X = 44 + (4 * NumParts).

2.4.6 Especificação do Conteúdo do Registro de Tipo PointM

TABELA 09 – TIPO POINTM

Posição	Campo	Valor	Tipo do dado	Tamanho	Ordem do Byte
Byte 0	Tipo	21	*Integer*	1	Baixa
Byte 4	X	X	*Double*	1	Baixa
Byte 12	Y	Y	*Double*	1	Baixa
Byte 20	M	M	*Double*	1	Baixa

2.4.7 Especificação do Conteúdo do Registro de Tipo MultiPointM

TABELA 10 – TIPO MULTIPOINTM

Posição	Campo	Valor	Tipo do dado	Tamanho	Ordem do Byte
Byte 0	Tipo	28	*Integer*	1	Baixa
Byte 4	Box	Box	*Double*	4	Baixa
Byte 36	NumPoints	NumPoints	*Integer*	1	Baixa
Byte 40	Points	Points	*Points*	NumPoints	Baixa
Byte X*	Mmin	Mmin	*Double*	1	Baixa
Byte X+8*	Mmax	Mmax	*Double*	1	Baixa
Byte X+16	Marray	Marray	*Double*	NumPoints	Baixa

Notas: X = 40 + (16 * NumPoints).
*** Opicional.**

2.4.8 Especificação do Conteúdo do Registro de Tipo PolyLineM

TABELA 11 – TIPO POLYLINEM

Posição	Campo	Valor	Tipo do dado	Tamanho	Ordem do Byte
Byte 0	Tipo	23	*Integer*	1	Baixa
Byte 4	Box	Box	*Double*	4	Baixa
Byte 36	NumParts	NumParts	*Integer*	1	Baixa
Byte 40	NumPoints	NumPoints	*Integer*	1	Baixa
Byte 44	Parts	Parts	*Integer*	NumParts	Baixa
Byte X	Points	Points	*Point*	NumPoints	Baixa
Byte Y*	Mmim	Mmim	*Double*	1	Baixa
Byte Y+8*	Mmax	Mmax	*Double*	1	Baixa
Byte Y+16*	Marray	Marray	*Double*	NumPoints	Baixa

Notas: X = 44 + (4 * NumParts); Y = X + (16 * NumPoints).
*** Opicional.**

2.4.9 Especificação do Conteúdo do Registro de Tipo PolygonM

TABELA 12 – TIPO POLYGONM

Posição	Campo	Valor	Tipo do dado	Tamanho	Ordem do Byte
Byte 0	Tipo	25	*Integer*	1	Baixa
Byte 4	Box	Box	*Double*	4	Baixa
Byte 36	NumParts	NumParts	*Integer*	1	Baixa
Byte 40	NumPoints	NumPoints	*Integer*	1	Baixa
Byte 44	Parts	Parts	*Integer*	NumParts	Baixa
Byte X	Points	Points	*Point*	NumPoints	Baixa
Byte Y*	Mmim	Mmim	*Double*	1	Baixa
Byte Y+8*	Mmax	Mmax	*Double*	1	Baixa
Byte Y+16*	Marray	Marray	*Double*	NumPoints	Baixa

Notas: X = 44 + (4 * NumParts); Y = X + (16 * NumPoints).
*** Opicional.**

2.4.10 Especificação do Conteúdo do Registro de Tipo PointZ

TABELA 13 – TIPO POINTZ

Posição	Campo	Valor	Tipo do dado	Tamanho	Ordem do Byte
Byte 0	Tipo	11	*Integer*	1	Baixa
Byte 4	X	X	*Double*	1	Baixa
Byte 12	Y	Y	*Double*	1	Baixa
Byte 20	Z	Z	*Double*	1	Baixa
Byte 28	Measure	M	*Double*	1	Baixa

2.4.11 Especificação do Conteúdo do Registro de Tipo MultiPointZ

TABELA 14 – TIPO MULTIPOINTZ

Posição	Campo	Valor	Tipo do dado	Tamanho	Ordem do Byte
Byte 0	Tipo	18	*Integer*	1	Baixa
Byte 4	Box	Box	*Double*	4	Baixa
Byte 36	NumPoints	NumPoints	*Integer*	1	Baixa
Byte 40	Points	Points	*Points*	NumPoints	Baixa
Byte X	Zmin	Zmin	*Double*	1	Baixa
Byte X+8	Zmax	Zmax	*Double*	1	Baixa
Byte X+16	Zarray	Zarray	*Double*	NumPoints	Baixa
Byte Y*	Mmin	Mmin	*Double*	1	Baixa
Byte Y+8*	Mmax	Mmax	*Double*	1	Baixa
Byte Y+16*	Marray	Marray	*Double*	NumPoints	Baixa

Notas: X = 40 + (16 * NumPoints); Y = X + 16 + (8 * NumPoints).
*** Opicional.**

2.4.12 Especificação do Conteúdo do Registro de Tipo PolyLineZ

TABELA 15 – TIPO POLYLINEZ

Posição	Campo	Valor	Tipo do dado	Tamanho	Ordem do Byte
Byte 0	Tipo	13	*Integer*	1	Baixa
Byte 4	Box	Box	*Double*	4	Baixa
Byte 36	NumParts	NumParts	*Integer*	1	Baixa
Byte 40	NumPoints	NumPoints	*Integer*	1	Baixa
Byte 44	Parts	Parts	*Integer*	NumParts	Baixa
Byte X	Points	Points	*Points*	NumPoints	Baixa
Byte Y	Zmin	Zmin	*Double*	1	Baixa
Byte Y+8	Zmax	Zmax	*Double*	1	Baixa
Byte Y+16	Zarray	Zarray	*Double*	NumPoints	Baixa
Byte Z*	Mmin	Mmin	*Double*	1	Baixa
Byte Z+8*	Mmax	Mmax	*Double*	1	Baixa
Byte Z+16*	Marray	Marray	*Double*	NumPoints	Baixa

Notas: X = 44 + (4 * NumParts); Y = X + (16 * NumPoints) ; Z = Y + 16 + (8 * NumPoints).
*** Opicional.**

2.4.13 Especificação do Conteúdo do Registro de Tipo PolygonZ

TABELA 16 – TIPO POLYGONZ

Posição	Campo	Valor	Tipo do dado	Tamanho	Ordem do Byte
Byte 0	Tipo	15	*Integer*	1	Baixa
Byte 4	Box	Box	*Double*	4	Baixa
Byte 36	NumParts	NumParts	*Integer*	1	Baixa
Byte 40	NumPoints	NumPoints	*Integer*	1	Baixa
Byte 44	Parts	Parts	*Integer*	NumParts	Baixa
Byte X	Points	Points	*Points*	NumPoints	Baixa
Byte Y	Zmin	Zmin	*Double*	1	Baixa
Byte Y+8	Zmax	Zmax	*Double*	1	Baixa
Byte Y+16	Zarray	Zarray	*Double*	NumPoints	Baixa
Byte Z*	Mmin	Mmin	*Double*	1	Baixa
Byte Z+8*	Mmax	Mmax	*Double*	1	Baixa
Byte Z+16*	Marray	Marray	*Double*	NumPoints	Baixa

Notas: X = 44 + (4 * NumParts); Y = X + (16 * NumPoints) ; Z = Y + 16 + (8 * NumPoints).
*** Opicional.**

2.4.14 Especificação do Conteúdo do Registro de Tipo MultiPatch

TABELA 17 – TIPO MULTIPATCH

Posição	Campo	Valor	Tipo do dado	Tamanho	Ordem do Byte
Byte 0	Tipo	31	*Integer*	1	Baixa
Byte 4	Box	Box	*Double*	4	Baixa
Byte 36	NumParts	NumParts	*Integer*	1	Baixa
Byte 40	NumPoints	NumPoints	*Integer*	1	Baixa
Byte 44	Parts	Parts	*Integer*	NumParts	Baixa
Byte W	PartTypes	PartTypes	*Integer*	NumParts	Baixa
Byte X	Points	Points	*Points*	NumPoints	Baixa
Byte Y	Zmin	Zmin	*Double*	1	Baixa
Byte Y+8	Zmax	Zmax	*Double*	1	Baixa
Byte Y+16	Zarray	Zarray	*Double*	NumPoints	Baixa
Byte Z*	Mmin	Mmin	*Double*	1	Baixa
Byte Z+8*	Mmax	Mmax	*Double*	1	Baixa
Byte Z+16*	Marray	Marray	*Double*	NumPoints	Baixa

Notas: W = 44 + (4 * NumParts); X = W + (4 * NumParts); Y = X + (16 * NumPoints);
Z = Y + 16 + (8 * NumPoints).
*** Opicional.**

Apêndice 2 - UML (*Unified Modeling Language*).

SUMÁRIO

LISTA DE FIGURAS

1 UML - Unified Modeling Language

Uma linguagem de modelagem consiste das notações (símbolos usados em um modelo) e o conjunto de regras que dizem como devem ser usadas estas notações. *Unified Modeling Language* (UML) é uma linguagem de modelagem que procura, através de diagramas, definir as principais fases dentro das diversas metodologias baseadas em objetos - análise, projeto e implementação. Na seqüência será apresentado o conjunto mínimo de elementos de modelagem, que foi adotado pelo trabalho para modelar o sistema de visualização de camadas de informação.

2 DIAGRAMAS

2.1 DIAGRAMA DE CASO DE USO

Um modelo de caso de uso é descrito pela UML através de um **diagrama de caso de uso**. Um diagrama de caso de uso contém elementos modelados para um sistema: os atores e os casos de uso, e mostra os diferentes relacionamentos, como também generalizações, associações e dependências entre estes elementos.

O **ator** é alguém ou alguma coisa que interage com o sistema, envia ou recebe mensagem, troca informações. Uma pessoa pode ser diferentes atores em um sistema, ele possui um nome, e este nome reflete o papel do ator no sistema. O nome não deve refletir uma instância específica do ator nem a funcionalidade. Atores são **classes** com o estereótipo "ator" e o nome da classe reflete o papel do ator.

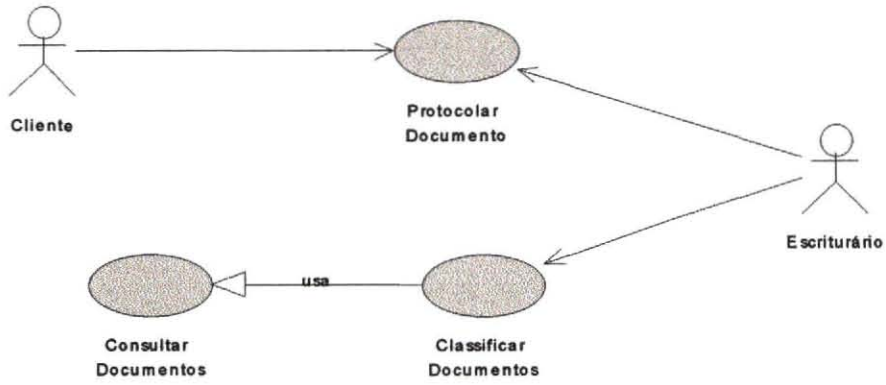
Um caso de uso é sempre iniciado por um ator que envia ou recebe uma mensagem **(estímulo)**. Um ator pode receber ou enviar mensagens de um sistema ou de outro ator. Os atores podem ser também classificados como sendo **ativos** ou **passivos**, o ativo inicia o caso de uso e o passivo nunca inicia, mas pode participar de um ou mais casos de uso.

Caso de Uso em UML é definido com sendo um conjunto de seqüências de ações que produz um resultado para um ator em particular. Um caso de uso é representado por uma elipse contendo o nome do caso de uso que geralmente está contida dentro dos limites do sistema, e pode estar conectada com um ator,

associação ou uma associação de comunicação. As ações podem envolver comunicação entre uma certa quantidade de atores, como também executando trabalhos e cálculos dentro do sistema. Um caso de uso é sempre iniciado por um ator.

Uma instância de um caso de uso é dita **cenário,** que representa um uso atual do sistema.

FIGURA 01 – DIAGRAMA DE CASO DE USO

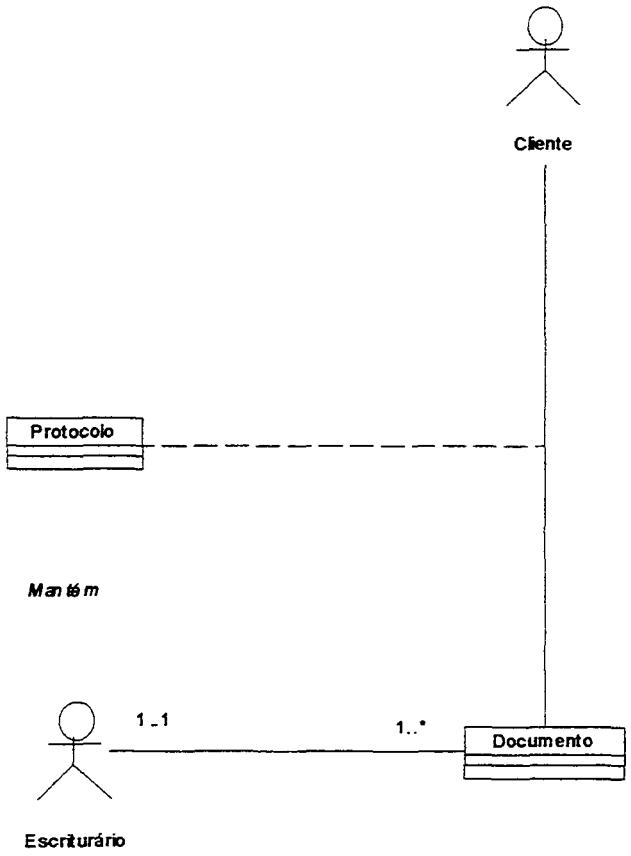


2.2 DIAGRAMA DE CLASSES PARA UM MODELO CONCEITUAL

O diagrama de classes para um modelo conceitual apresenta-se com atributos e operações ainda não completamente definidos. Uma classe possui três partes distintas: o nome que define a classe, os atributos e as operações pertencentes a esta classe. Uma classe pode estar relacionada à outra classe e pode possuir generalizações, sendo que a multiplicidade pode estar indicada nas associações. O diagrama de classes para um modelo conceitual não precisaria possuir atributos ou operações, pois em uma primeira implementação poderia conter somente as classes e suas respectivas associações, sendo que poderia sofrer acréscimos de implementação durante a fase de análise, principalmente quando da modelagem do diagrama de seqüências. Durante esta modelagem, é comum surgir detalhamentos em forma de operações e atributos, que podem enriquecer o diagrama de classes.

Na finalização da fase de projeto o diagrama de classes deve conter totalmente ou quase totalmente todos os atributos e operações, para que não cause discordância com a fase de implementação.

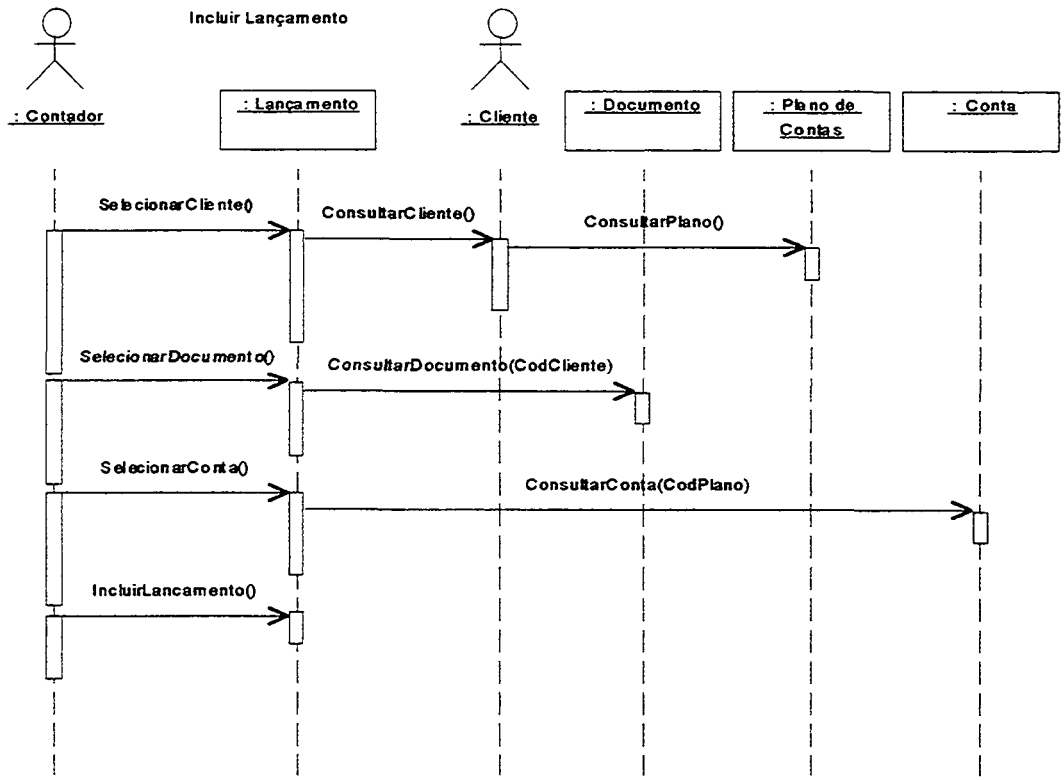
FIGURA 02 – DIAGRAMA DE CLASSES PARA UM MODELO CONCEITUAL



2.3 DIAGRAMA DE SEQÜÊNCIAS

Um diagrama de seqüências mostra para um cenário em particular os eventos externos gerados por um ator, seus pedidos e os eventos internos ao sistema. Na fase de análise este diagrama trata o sistema como uma caixa preta, não importando ainda os detalhes do sistema, que somente serão conhecidos na fase de projeto.

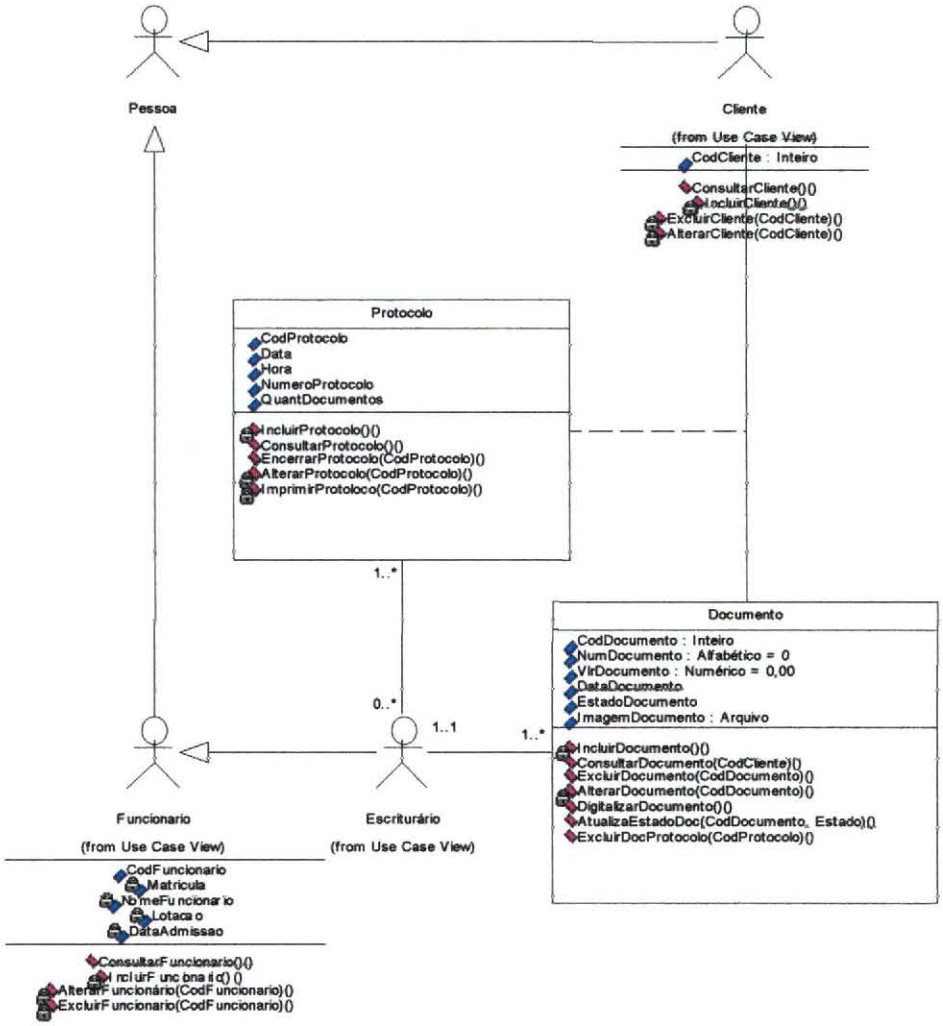
FIGURA 03 – DIAGRAMA DE SEQÜÊNCIAS



2.4 DIAGRAMA DE CLASSES

A descrição do diagrama de classes é equivalente ao diagrama de classes para um modelo conceitual, porém, ele deve receber a especificação dos atributos e operações. O diagrama de classes normalmente é utilizado na fase de projeto e é a complementação do diagrama de classes para um modelo conceitual definido na fase de análise.

FIGURA 04 – DIAGRAMA DE CLASSES



2.4.1 Atributos

Os atributos são dados guardados pelos objetos de uma classe, são informações a respeito de uma determinada instância de um objeto. Diferentes instâncias de um objeto podem ter valores iguais ou diferentes para um dado atributo. Cada atributo possui um nome distinto dentro de uma classe. Um atributo pode ser nomeado de valor de dado, e não tem identidade, pois a ocorrência de um determinado valor por um atributo não pode ser distinguível; sendo assim, um atributo não pode ser um objeto.

2.4.2 Operações

Pode-se dizer que operação é um serviço executado por um objeto em resposta a um estímulo, é uma função ou transformação aplicada a objetos ou por estes a uma classe.

Enquanto os substantivos, dentro da descrição de um caso de uso, são fortes candidatos a classes, os verbos são fortes candidatos a operações. Utilizando-se do curso típico dos eventos de um caso de uso, podemos facilmente identificar os substantivos e verbos correspondentes.

Todos os objetos de uma classe compartilham as mesmas operações. Cada nome de operação deve ser seguido pelos detalhes, quando estes existirem, como a lista de argumentos, que é escrita entre parênteses após o nome da operação e separada por vírgulas, e o tipo do resultado é precedido por dois pontos. O tipo de resultado, quando este existir, não deve ser omitido, visando diferenciar operações que não retornam valores.

2.4.3 Método

Código implementado em linguagem de programação correspondente ao corpo da operação, ou seja, um método é uma implementação de uma operação em uma classe.

2.4.4 Mensagem

Comunicação enviada pelo objeto remetente ao objeto destinatário. Quando o objeto destinatário recebe uma mensagem (também dito evento), este pode retornar uma resposta ao objeto remetente.

As mensagens podem ser ***assíncronas*** ou ***síncronas***. Diz-se ***assíncronas*** quando o objeto remetente envia uma mensagem e não interrompe seu processamento no aguardo de resultados, e ***síncronas*** quando o objeto remetente depende de resultados do objeto destinatário para continuar seu processamento.

Apêndice 3 – Código Fonte

SUMÁRIO

1 PACOTE REGISTRO

1.1 CLASSE MultiMType

```
public class MultiMType extends MultiType {
        private double Mmin;
        private double Mmax;
        private double Marray[];

public MultiMType() {
        super();
}
        public double getMarray(int i){
                return Marray[i];
        }
        public double getMmax(){
                return Mmax;
        }
        public double getMmin(){
                return Mmin;
        }
        public void setMarray(int i, double Point){
                Marray[i] = Point;
        }
        public void setMmax(double Num){
                Mmax = Num;
        }
        public void setMmin(double Num){
                Mmin = Num;
        }
}
```

1.2 CLASSE MultiPatchShape

```
public class MultiPatchShape extends MultiPatchType {
public MultiPatchShape() {
        super();
        this.setShapeType(31); //MultiPatch Shape
}
}
```

1.3 CLASSE MultiPatchType

```
public class MultiPatchType extends PolyZType {
        private int PartType[];

public MultiPatchType() {
        super();
}
        public int getPartType(int i){
```

```
                return PartType[i];
        }
        public void setPartType(){
                PartType = new int[this.getNumParts()];
        }
        public void setPartType(int i, int PType){
                PartType[i] = PType;
        }
}
```

1.4 CLASSE MultiPointMShape

```
public class MultiPointMShape extends MultiMType {
public MultiPointMShape() {
        super();
        this.setShapeType(28); //MultiPointM Shape
}
}
```

1.5 CLASSE MultiPointShape

```
public class MultiPointShape extends MultiType {
public MultiPointShape() {
        super();
        this.setShapeType(8); //MultiPoint Shape
}
}
```

1.6 CLASSE MultiPointZShape

```
public class MultiPointZShape extends MultiZType {
public MultiPointZShape() {
        super();
        this.setShapeType(18); //MultiPointZ Shape
}
}
```

1.7 CLASSE MultiType

```
public class MultiType extends ShapeType {
        private double Box[] = new double[4];
        private int NumPoints;
        private PointXY Points[];

public MultiType() {
        super();
}
        public double getBox(int i){
                return this.Box[i];
```

```
        }
        public int getNumPoints(){
                return this.NumPoints;
        }
        public double getPointsX(int i){
                return this.Points[i].getX();
        }
        public double getPointsY(int i){
                return this.Points[i].getY();
        }
        public void setBox(int i, double Coord){
                this.Box[i] = Coord;
        }
        public void setNumPoints(int Num){
                if (Num < 0){
                        Num = (Num ^ -256);
                }
                this.NumPoints = Num;
                this.Points = new PointXY[this.NumPoints];
        }
        public void setPoints(int i, double Px, double Py){
                this.Points[i] = new PointXY(Px, Py);
        }
}
```

1.8 CLASSE MultiZType

```
public class MultiZType extends MultiMType {
        private double Zmin;
        private double Zmax;
        private double Zarray[];

public MultiZType() {
        super();
}
        public double getZarray(int i){
                return Zarray[i];
        }
        public double getZmax(){
                return Zmax;
        }
        public double getZmin(){
                return Zmin;
        }
        public void setZarray(int i, double Point){
                Zarray[i] = Point;
        }
        public void setZmax(double Num){
                Zmax = Num;
        }
        public void setZmin(double Num){
                Zmin = Num;
        }
}
```

1.9 CLASSE NullShape

```
public class NullShape extends ShapeType {
public NullShape() {
        super();
        this.setShapeType(0); //Null Shape
}
}
```

1.10 CLASSE PointMShape

```
public class PointMShape extends PointMType {
public PointMShape() {
        super();
        this.setShapeType(21); //PointM Shape
}
}
```

1.11 CLASSE PointMType

```
public class PointMType extends PointType {
        private double M;

public PointMType() {
        super();
}
        public double getM(){
                return M;
        }
        public void setM(double Pt){
                M = Pt;
        }
}
```

1.12 CLASSE PointShape

```
public class PointShape extends PointType {
public PointShape() {
        super();
        this.setShapeType(1); //Point Shape
}
}
```

1.13 CLASSE PointType

```
public class PointType extends ShapeType {
        private double X;
        private double Y;

        public PointType() {
        super();
}
        public double getX(){
                return X;
        }
        public double getY(){
                return Y;
        }
        public void setX(double Pt){
                X = Pt;
        }
        public void setY(double Pt){
                Y = Pt;
        }
}
```

1.14 CLASSE PointXY

```
public class PointXY implements java.io.Serializable {
        static final long serialVersionUID = 7835202140011595704L;
        private double X;
        private double Y;
public PointXY() {
                super();
                setX(0);
                setY(0);
}
        public PointXY(double X, double Y) {
                super();
                setX(X);
                setY(Y);
        }
        public double getX(){
                return X;
        }
        public double getY(){
                return Y;
        }
        public void setX(double Point){
                X = Point;
        }
        public void setY(double Point){
                Y = Point;
        }
}
```

1.15 CLASSE PointZShape

```
public class PointZShape extends PointZType {
public PointZShape() {
        super();
        this.setShapeType(11); //PointZ Shape
}
}
```

1.16 CLASSE PointZType

```
public class PointZType extends PointMType {
        private double Z;
public PointZType() {
        super();
}
        public double getZ(){
                return Z;
        }
        public void setZ(double Pt){
                Z = Pt;
        }
}
```

1.17 CLASSE PolygonMShape

```
public class PolygonMShape extends PolyMType {
public PolygonMShape() {
        super();
        this.setShapeType(25); //PolygonM Shape
}
}
```

1.18 CLASSE PolygonShape

```
public class PolygonShape extends PolyType {
public PolygonShape() {
        super();
        this.setShapeType(5); //Polygon Shape
}
}
```

1.19 CLASSE PolygonZShape

```
public class PolygonZShape extends PolyZType {
public PolygonZShape() {
        super();
        this.setShapeType(15); //PolygonZ Shape
}
}
```

1.20 CLASSE PolyLineMShape

```
public class PolyLineMShape extends PolyMType {
public PolyLineMShape() {
        super();
        this.setShapeType(23); //PolyLineM Shape
}
}
```

1.21 CLASSE PolyLineShape

```
public class PolyLineShape extends PolyType {
public PolyLineShape() {
        super();
        this.setShapeType(3); //PolyLine Shape
}
}
```

1.22 CLASSE PolyLineZShape

```
public class PolyLineZShape extends PolyZType {
public PolyLineZShape() {
        super();
        this.setShapeType(13); //PolyLineZ Shape
}
}
```

1.23 CLASSE PolyMType

```
public class PolyMType extends PolyType {
        private double Mmin;
        private double Mmax;
        private double Marray[];

public PolyMType() {
        super();
}
        public double getMarray(int i){
```

```
                return Marray[i];
        }
        public double getMmax(){
                return Mmax;
        }
        public double getMmin(){
                return Mmin;
        }
        public void setMarray(int i, double Point){
                Marray[i] = Point;
        }
        public void setMmax(double Num){
                Mmax = Num;
        }
        public void setMmin(double Num){
                Mmin = Num;
        }
}
```

1.24 CLASSE PolyType

```
public class PolyType extends MultiType {
        private int NumParts;
        private int Parts[];
public PolyType() {
        super();
}
        public int getNumParts(){
                return NumParts;
        }
        public int getParts(int i){
                return Parts[i];
        }
        public void setNumParts(int Num){
                if (Num < 0){
                        Num = (Num ^ -256);
                }
                NumParts = Num;
                Parts = new int[NumParts];
        }
        public void setParts(int i, int Part){
                Parts[i] = Part;
        }
}
```

1.25 CLASSE PolyZType

```
public class PolyZType extends PolyMType {
        private double Zmin;
        private double Zmax;
        private double Zarray[];

public PolyZType() {
        super();
```

```
}
        public double getZarray(int i){
                return Zarray[i];
        }
        public double getZmax(){
                return Zmax;
        }
        public double getZmin(){
                return Zmin;
        }
        public void setZarray(int i, double Point){
                Zarray[i] = Point;
        }
        public void setZmax(double Num){
                Zmax = Num;
        }
        public void setZmin(double Num){
                Zmin = Num;
        }
}
```

1.26 CLASSE ShapeType

```
import java.awt.geom.*;


public class ShapeType implements java.io.Serializable {
        static final long serialVersionUID = 7835202140011595704L;
        private int ShapeType;

public ShapeType() {
        super();
}
        public int getShapeTypeCode(){
                return ShapeType;
        }
        public String getShapeTypeName(){
                switch (ShapeType) {
                        case 0:
                                return "Null Shape";

                        case 1:
                                return "Point";

                        case 3:
                                return "PolyLine";

                        case 5:
                                return "Polygon";

                        case 8:
                                return "MultiPoint";

                        case 11:
                                return "PointZ";
```

```
                        case 13:
                                return "PolyLineZ";

                        case 15:
                                return "PolygonZ";

                        case 18:
                                return "MultiPointZ";

                        case 21:
                                return "PointM";

                        case 23:
                                return "PolyLineM";

                        case 25:
                                return "PolygonM";

                        case 28:
                                return "MultiPointM";

                        case 31:
                                return "MultiPatch";

                        default:
                                return "Tipo não definido!";
                }
        }
        public void setShapeType(int Tp){
                ShapeType = Tp;
        }
}
```

2 PACOTE ARQUIVO

2.1 CLASSE BBox

```
public class BBox extends ShapeSHP {
        private double Xmin;
        private double Ymin;
        private double Xmax;
        private double Ymax;
        private double Zmin;
        private double Zmax;
        private double Mmin;
        private double Mmax;
public BBox() {
        super();
}
        public double getMmax(){
                return Mmax;
        }
        public double getMmin(){
                return Mmin;
```

```
        }
        public double getXmax(){
                return Xmax;
        }
        public double getXmin(){
                return Xmin;
        }
        public double getYmax(){
                return Ymax;
        }
        public double getYmin(){
                return Ymin;
        }
        public double getZmax(){
                return Zmax;
        }
        public double getZmin(){
                return Zmin;
        }
        public void setMmax(double i){
                Mmax = i;
        }
        public void setMmin(double i){
                Mmin = i;
        }
        public void setXmax(double i){
                Xmax = i;
        }
        public void setXmin(double i){
                Xmin = i;
        }
        public void setYmax(double i){
                Ymax = i;
        }
        public void setYmin(double i){
                Ymin = i;
        }
        public void setZmax(double i) {
                Zmax = i;
        }
        public void setZmin(double i){
                Zmin = i;
        }
}
```

2.2 CLASSE Header

```
public class Header extends Record{
        private int RecordNumber;
        private int ContentLength;
        private int TypeContent;
public Header() {
        super();
}
        public int getContentLength(){
                return ContentLength;
```

```
        }
        public int getRecordNumber(){
                return RecordNumber;
        }
        public int getTypeContent(){
                return TypeContent;
        }
        public void setContentLength(int ContLen){
                ContentLength = ContLen;
        }
        public void setRecordNumber(int RecNum){
                RecordNumber = RecNum;
        }
        public void setTypeContent(int TpCont){
                TypeContent = TpCont;
        }
}
```

2.3 CLASSE Record

```
import mestrado.shape.objeto.*;
import java.awt.Shape;
import java.awt.geom.*;

public class Record implements java.io.Serializable {
        static final long serialVersionUID = 7835202140011595704L;
        private Header RecordHeader;
        private Object RecordContents;

public Record() {
        super();
}
        public int ByteConvert(int By){
                if (By < 0){
                  By = (By ^ -256);
                }
                return(By);
        }
        public long ByteConvert(long By){
                if (By < 0){
                  By = (By ^ -256);
                }
                return(By);
        }
        public double ByteConvertToDouble(byte VcByte[]){
                // Converte Byte (baixa ordem) para Double
                double b = 0.00;
                return (new Double(b).longBitsToDouble((
                        (ByteConvert((int)VcByte[0])) +
                        (ByteConvert((int)VcByte[1]) << 8) +
                        (ByteConvert((int)VcByte[2]) << 16)  +
                        (ByteConvert((long)VcByte[3]) << 24) +
                        (ByteConvert((long)VcByte[4]) << 32) +
                        (ByteConvert((long)VcByte[5]) << 40) +
                        (ByteConvert((long)VcByte[6]) << 48) +
                        ((long)VcByte[7] << 56))));
```

```
        }
        public int ByteConvertToInteger(byte VcByte[]){
                return ((
                        (ByteConvert((int)VcByte[0])) +
                        (ByteConvert((int)VcByte[1]) << 8) +
                        (ByteConvert((int)VcByte[2]) << 16) |
                        ((int)VcByte[3] << 24)));
        }
        public int getContentLength(){
                return RecordHeader.getContentLength();
        }
        public Shape getPathShape(){
                PolygonShape ObjCont5 = (PolygonShape)this.getRecordContent();
                GeneralPath ShpObj =
                        new GeneralPath(PathIterator.WIND_NON_ZERO,
                                        ObjCont5.getNumPoints());
                int PartEnd;
                String s = " / ";
                for (int i = 0; i < 1/*ObjCont5.getNumParts()*/; i++){
                        ShpObj.moveTo((float)ObjCont5.getPointsX(ObjCont5.getParts(i))
                                * 100,(float)ObjCont5.getPointsY(ObjCont5.getParts(i)) * 100);
                        for (int j = (ObjCont5.getParts(i) + 1); j < 6; j++){
                                ShpObj.lineTo((float)ObjCont5.getPointsX(j) * 100,
                                        (float)ObjCont5.getPointsY(j) * 100);
                        }
                }
                AffineTransform Af = new AffineTransform();
                return (ShpObj);
        }
        public Object getRecordContent(){
                return this.RecordContents;
        }
        public int getRecordNumber(){
                return RecordHeader.getRecordNumber();
        }
        public int getTypeContent(){
                return RecordHeader.getTypeContent();
        }
        public void setContentLength(int ContLen){
                RecordHeader.setContentLength(ContLen);
        }
public void setRecordContents(byte VcByte[]){

        //ShapeType
        int ShpType = (ByteConvert((int)VcByte[0]) + (ByteConvert((int)VcByte[1]) << 8) +
                        (ByteConvert((int)VcByte[2]) << 16) | ((int)VcByte[3] << 24));
        this.RecordHeader.setTypeContent(ShpType);
        int i = 4;
        double b = 0.00;
        int j = 0;
        int X, Y, W;
        byte ByteDouble[] = new byte[8];
        byte ByteInteger[] = new byte[4];
        switch (ShpType) {
                case 0:
                        //"Null Shape";
                        RecordContents = new NullShape();
                        //RecordContents <-- NullShape
                        break;
```

```
case 1:
        //"Point";
        PointShape RecContTypePoint = new PointShape();
        //X
        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
        RecContTypePoint.setX(ByteConvertToDouble(ByteDouble));
        i = i + 8;
        //Y
        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
        RecContTypePoint.setY(ByteConvertToDouble(ByteDouble));
        i = i + 8;
        //RecordContents <-- PointShape
        RecordContents = new PointShape();
        RecordContents = RecContTypePoint;
        break;

case 3:
        //"PolyLine";
        PolyLineShape RecContTypePolyLine = new PolyLineShape();
        while (i < 36){ //Box
                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                RecContTypePolyLine.setBox(j, ByteConvertToDouble(ByteDouble));
                i = i + 8;
                j++;
        }
        //NumParts
        System.arraycopy(VcByte, i, ByteInteger, 0, 4);
        RecContTypePolyLine.setNumParts(ByteConvertToInteger(ByteInteger));
        i = i + 4;
        //NumPoints
        System.arraycopy(VcByte, i, ByteInteger, 0, 4);
        RecContTypePolyLine.setNumPoints(ByteConvertToInteger(ByteInteger));
        i = i + 4;
        j = 0;
        //Parts
        // X = 44 + 4 * NumParts
        while (i < (44 + (4 * RecContTypePolyLine.getNumParts()))){
                System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                RecContTypePolyLine.setParts(j, ByteConvertToInteger(ByteInteger));
                i = i + 4;
                j++;
        }
        j = 0;
        //Points
        while (j < (RecContTypePolyLine.getNumPoints())){
                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                double X1 = ByteConvertToDouble(ByteDouble);
                i = i + 8;
                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                double Y1 = ByteConvertToDouble(ByteDouble);
                i = i + 8;
                RecContTypePolyLine.setPoints(j, X1, Y1);
                j++;
        }
        //RecordContents <-- PolyLineShape
        RecordContents = new PolyLineShape();
        RecordContents = RecContTypePolyLine;
        break;
```

```
case 5:
        //"Polygon"
        PolygonShape RecContTypePolygon = new PolygonShape();
        while (i < 36){ //Box
                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                RecContTypePolygon.setBox(j, ByteConvertToDouble(ByteDouble));
                i = i + 8;
                j++;
        }
        //NumParts
        System.arraycopy(VcByte, i, ByteInteger, 0, 4);
        RecContTypePolygon.setNumParts(ByteConvertToInteger(ByteInteger));
        i = i + 4;
        //NumPoints
        System.arraycopy(VcByte, i, ByteInteger, 0, 4);
        RecContTypePolygon.setNumPoints(ByteConvertToInteger(ByteInteger));
        i = i + 4;
        j = 0;
        //Parts
        //X = 44 + 4 * NumParts
        while (i < (44 + (4 * RecContTypePolygon.getNumParts()))){
                System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                RecContTypePolygon.setParts(j, ByteConvertToInteger(ByteInteger));
                i = i + 4;
                j++;
        }
        i = (44 + (4 * RecContTypePolygon.getNumParts()));
        j = 0;
        //Points
        while (j < (RecContTypePolygon.getNumPoints())){
                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                double X1 =  ByteConvertToDouble(ByteDouble);
                i = i + 8;
                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                double Y1 =  ByteConvertToDouble(ByteDouble);
                i = i + 8;
                RecContTypePolygon.setPoints(j, X1, Y1);
                j++;
        }
        //RecordContents <-- PolygonShape
        RecordContents = new PolygonShape();
        RecordContents = RecContTypePolygon;
        break;

case 8:
        //"MultiPoint";
        MultiPointShape RecContTypeMultiPoint = new MultiPointShape();
        while (i < 36){ //Box
                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                RecContTypeMultiPoint.setBox(j,
                                ByteConvertToDouble(ByteDouble));
                i = i + 8;
                j++;
        }
        //NumPoints
        System.arraycopy(VcByte, i, ByteInteger, 0, 4);
        RecContTypeMultiPoint.setNumPoints(ByteConvertToInteger(ByteInteger));
        i = i + 4;
```

```
                j = 0;
                //Points
                while (j < (RecContTypeMultiPoint.getNumPoints())){
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        double X1 = ByteConvertToDouble(ByteDouble);
                        i = i + 8;
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        double Y1 = ByteConvertToDouble(ByteDouble);
                        i = i + 8;
                        RecContTypeMultiPoint.setPoints(j, X1, Y1);
                        j++;
                }
                //RecordContents <-- MultiPointShape
                RecordContents = new MultiPointShape();
                RecordContents = RecContTypeMultiPoint;
                break;

        case 11:
                //"PointZ";
                PointZShape RecContTypePointZ = new PointZShape();
                //X
                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                RecContTypePointZ.setX(ByteConvertToDouble(ByteDouble));
                i = i + 8;
                //Y
                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                RecContTypePointZ.setY(ByteConvertToDouble(ByteDouble));
                i = i + 8;
                //Z
                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                RecContTypePointZ.setZ(ByteConvertToDouble(ByteDouble));
                i = i + 8;
                //M
                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                RecContTypePointZ.setM(ByteConvertToDouble(ByteDouble));
                i = i + 8;
                //RecordContents <-- PointZShape
                RecordContents = new PointZShape();
                RecordContents = RecContTypePointZ;
                break;

        case 13:
                //"PolyLineZ";
                PolyLineZShape RecContTypePolyLineZ = new PolyLineZShape();
                while (i < 36){ //Box
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        RecContTypePolyLineZ.setBox(j,
                                        ByteConvertToDouble(ByteDouble));
                        i = i + 8;
                        j++;
                }
                //NumParts
                System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                RecContTypePolyLineZ.setNumParts(ByteConvertToInteger(ByteInteger));
                i = i + 4;
                //NumPoints
                System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                RecContTypePolyLineZ.setNumPoints(ByteConvertToInteger(ByteInteger));
                i = i + 4;
```

```
j = 0;
//Parts
// X = 44 + (4 * NumParts)
while (i < (44 + (4 * RecContTypePolyLineZ.getNumParts()))){
        System.arraycopy(VcByte, i, ByteInteger, 0, 4);
        RecContTypePolyLineZ.setParts(j,
                                ByteConvertToInteger(ByteInteger));
        i = i + 4;
        j++;
}
j = 0;
//Points
X = (i + (16 * RecContTypePolyLineZ.getNumPoints()));
while (i < X){ // Y = X + (16 * NumPoints)
        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
        double X1 = ByteConvertToDouble(ByteDouble);
        i = i + 8;
        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
        double Y1 = ByteConvertToDouble(ByteDouble);
        i = i + 8;
        RecContTypePolyLineZ.setPoints(j, X1, Y1);
        j++;
}
//Zmin
System.arraycopy(VcByte, i, ByteDouble, 0, 8);
RecContTypePolyLineZ.setZmin(ByteConvertToDouble(ByteDouble));
i = i + 8;
//Zmax
System.arraycopy(VcByte, i, ByteDouble, 0, 8);
RecContTypePolyLineZ.setZmax(ByteConvertToDouble(ByteDouble));
i = i + 8;
j = 0;
//Zarray
Y = (i + (8 * RecContTypePolyLineZ.getNumPoints()));
while (i < Y){ // Z = Y + (8 * NumPoints)
        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
        RecContTypePolyLineZ.setZarray(j,
                                ByteConvertToDouble(ByteDouble));
        i = i + 8;
        j++;
}
if (i < (VcByte.length)) { // Mmin, Mmax, Marray Opcional
        //Mmin
        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
        RecContTypePolyLineZ.setMmin(
                        ByteConvertToDouble(ByteDouble));
        i = i + 8;
        //Mmax
        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
        RecContTypePolyLineZ.setMmax(
                ByteConvertToDouble(ByteDouble));
        i = i + 8;
        j = 0;
        //Marray
        while (j < (RecContTypePolyLineZ.getNumPoints())){
                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                RecContTypePolyLineZ.setMarray(j,
                                ByteConvertToDouble(ByteDouble));
                i = i + 8;
```

```
                        j++;
                    }
                }
                //RecordContents <-- PolyLineZShape
                RecordContents = new PolyLineZShape();
                RecordContents = RecContTypePolyLineZ;
                break;

            case 15:
                //"PolygonZ";
                PolygonZShape RecContTypePolygonZ = new PolygonZShape();
                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                while (i < 36){ //Box
                    RecContTypePolygonZ.setBox(j, ByteConvertToDouble(ByteDouble));
                    i = i + 8;
                    j++;
                }
                //NumParts
                System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                RecContTypePolygonZ.setNumParts(ByteConvertToInteger(ByteInteger));
                i = i + 4;
                //NumPoints
                System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                RecContTypePolygonZ.setNumPoints(ByteConvertToInteger(ByteInteger));
                i = i + 4;
                j = 0;
                //Parts
                // X = 44 + (4 * NumParts)
                while (i < (44 + (4 * RecContTypePolygonZ.getNumParts()))){
                    System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                    RecContTypePolygonZ.setParts(j,
                                        ByteConvertToInteger(ByteInteger));
                    i = i + 4;
                    j++;
                }
                j = 0;
                //Points
                X = (i + (16 * RecContTypePolygonZ.getNumPoints()));
                while (i < X){ // Y = X + (16 * NumPoints)
                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                    double X1 = ByteConvertToDouble(ByteDouble);
                    i = i + 8;
                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                    double Y1 = ByteConvertToDouble(ByteDouble);
                    i = i + 8;
                    RecContTypePolygonZ.setPoints(j, X1, Y1);
                    j++;
                }
                //Zmin
                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                RecContTypePolygonZ.setZmin(ByteConvertToDouble(ByteDouble));
                i = i + 8;
                //Zmax
                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                RecContTypePolygonZ.setZmax(ByteConvertToDouble(ByteDouble));
                i = i + 8;
                j = 0;
                //Zarray
                Y = (i + (8 * RecContTypePolygonZ.getNumPoints()));
```

```
                while (i < Y){ // Z = Y + (8 * NumPoints)
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        RecContTypePolygonZ.setZarray(j,
                                                ByteConvertToDouble(ByteDouble));
                        i = i + 8;
                        j++;
                }
                if (i < (VcByte.length)) { // Mmin, Mmax, Marray Opcional
                        //Mmin
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        RecContTypePolygonZ.setMmin(ByteConvertToDouble(ByteDouble));
                        i = i + 8;
                        //Mmax
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        RecContTypePolygonZ.setMmax(
                                        ByteConvertToDouble(ByteDouble));
                        i = i + 8;
                        j = 0;
                        //Marray
                        while (j < (RecContTypePolygonZ.getNumPoints())){
                                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                                RecContTypePolygonZ.setMarray(j,
                                                        ByteConvertToDouble(ByteDouble));
                                i = i + 8;
                                j++;
                        }
                }
                //RecordContents <-- PolygonZSShape
                RecordContents = new PolygonZShape();
                RecordContents = RecContTypePolygonZ;
                break;

        case 18:
                //"MultiPointZ";
                MultiPointZShape RecContTypeMultiPointZ = new MultiPointZShape();
                while (i < 36){ //Box
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        RecContTypeMultiPointZ.setBox(j,
                                                ByteConvertToDouble(ByteDouble));
                        i = i + 8;
                        j++;
                }
                //NumPoints
                System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                RecContTypeMultiPointZ.setNumPoints(ByteConvertToInteger(ByteInteger));
                i = i + 4;
                j = 0;
                //Points
                X = (i + (16 * RecContTypeMultiPointZ.getNumPoints()));
                while (i < X){ // Y = X + (16 * NumPoints)
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        double X1 = ByteConvertToDouble(ByteDouble);
                        i = i + 8;
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        double Y1 = ByteConvertToDouble(ByteDouble);
                        i = i + 8;
                        RecContTypeMultiPointZ.setPoints(j, X1, Y1);
                        j++;
                }
```

```
                    //Zmin
                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                    RecContTypeMultiPointZ.setZmin(ByteConvertToDouble(ByteDouble));
                    i = i + 8;
                    //Zmax
                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                    RecContTypeMultiPointZ.setZmax(ByteConvertToDouble(ByteDouble));
                    i = i + 8;
                    j = 0;
                    //Zarray
                    Y = (i + (8 * RecContTypeMultiPointZ.getNumPoints()));
                    while (i < Y){ // Z = Y + (8 * NumPoints)
                            System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                            RecContTypeMultiPointZ.setZarray(j,
                                    ByteConvertToDouble(ByteDouble));
                            i = i + 8;
                            j++;
                    }
                    if (i < (VcByte.length)) { // Mmin, Mmax, Marray Opcional
                            //Mmin
                            System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                            RecContTypeMultiPointZ.setMmin(
                                    ByteConvertToDouble(ByteDouble));
                            i = i + 8;
                            //Mmax
                            System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                            RecContTypeMultiPointZ.setMmax(
                                    ByteConvertToDouble(ByteDouble));
                            i = i + 8;
                            j = 0;
                            //Marray
                            while (j < (RecContTypeMultiPointZ.getNumPoints())){
                                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                                    RecContTypeMultiPointZ.setMarray(j,
                                            ByteConvertToDouble(ByteDouble));
                                    i = i + 8;
                                    j++;
                            }
                    }
                    //RecordContents <-- MultiPointZShape
                    RecordContents = new MultiPointZShape();
                    RecordContents = RecContTypeMultiPointZ;
                    break;

            case 21:
                    //"PointM";
                    PointMShape RecContTypePointM = new PointMShape();
                    //X
                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                    RecContTypePointM.setX(ByteConvertToDouble(ByteDouble));
                    i = i + 8;
                    //Y
                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                    RecContTypePointM.setY(ByteConvertToDouble(ByteDouble));
                    i = i + 8;
                    //M
                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                    RecContTypePointM.setM(ByteConvertToDouble(ByteDouble));
                    i = i + 8;
```

```
                //RecordContents <-- PointMShape
                RecordContents = new PointMShape();
                RecordContents = RecContTypePointM;
                break;

        case 23:
                //"PolyLineM";
                PolyLineMShape RecContTypePolyLineM = new PolyLineMShape();
                while (i < 36){ //Box
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        RecContTypePolyLineM.setBox(j,
                                        ByteConvertToDouble(ByteDouble));
                        i = i + 8;
                        j++;
                }
                //NumParts
                System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                RecContTypePolyLineM.setNumParts(ByteConvertToInteger(ByteInteger));
                i = i + 4;
                //NumPoints
                System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                RecContTypePolyLineM.setNumPoints(ByteConvertToInteger(ByteInteger));
                i = i + 4;
                j = 0;
                //Parts
                // X = 44 + (4 * NumParts)
                while (i < (44 + (4 * RecContTypePolyLineM.getNumParts()))){
                        System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                        RecContTypePolyLineM.setParts(j,
                                        ByteConvertToInteger(ByteInteger));
                        i = i + 4;
                        j++;
                }
                j = 0;
                //Points
                X = (i + (16 * RecContTypePolyLineM.getNumPoints()));
                while (i < X){ // Y = X + (16 * NumPoints)
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        double X1 = ByteConvertToDouble(ByteDouble);
                        i = i + 8;
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        double Y1 = ByteConvertToDouble(ByteDouble);
                        i = i + 8;
                        RecContTypePolyLineM.setPoints(j, X1, Y1);
                        j++;
                }
                if (i < (VcByte.length)) { // Mmin, Mmax, Marray Opcional
                        //Mmin
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        RecContTypePolyLineM.setMmin(
                                        ByteConvertToDouble(ByteDouble));
                        i = i + 8;
                        //Mmax
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        RecContTypePolyLineM.setMmax(
                                ByteConvertToDouble(ByteDouble));
                        i = i + 8;
                        j = 0;
                        //Marray
```

```
                    while (j < (RecContTypePolyLineM.getNumPoints())){
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        RecContTypePolyLineM.setMarray(j,
                                ByteConvertToDouble(ByteDouble));
                        i = i + 8;
                        j++;
                    }
                }
                //RecordContents <-- PolyLineMShape
                RecordContents = new PolyLineMShape();
                RecordContents = RecContTypePolyLineM;
                break;

            case 25:
                //"PolygonM";
                PolygonMShape RecContTypePolygonM = new PolygonMShape();
                while (i < 36){ //Box
                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                    RecContTypePolygonM.setBox(j,
                            ByteConvertToDouble(ByteDouble));
                    i = i + 8;
                    j++;
                }
                //NumParts
                System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                RecContTypePolygonM.setNumParts(ByteConvertToInteger(ByteInteger));
                i = i + 4;
                //NumPoints
                System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                RecContTypePolygonM.setNumPoints(ByteConvertToInteger(ByteInteger));
                i = i + 4;
                j = 0;
                //Parts
                // X = 44 + (4 * NumParts)
                while (i < (44 + (4 * RecContTypePolygonM.getNumParts()))){
                    System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                    RecContTypePolygonM.setParts(j,
                            ByteConvertToInteger(ByteInteger));
                    i = i + 4;
                    j++;
                }
                j = 0;
                //Points
                X = (i + (16 * RecContTypePolygonM.getNumPoints()));
                while (i < X){ // Y = X + (16 * NumPoints)
                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                    double X1 = ByteConvertToDouble(ByteDouble);
                    i = i + 8;
                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                    double Y1 = ByteConvertToDouble(ByteDouble);
                    i = i + 8;
                    RecContTypePolygonM.setPoints(j, X1, Y1);
                    j++;
                }
                if (i < (VcByte.length)) { // Mmin, Mmax, Marray Opcional
                    //Mmin
                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                    RecContTypePolygonM.setMmin(
                            ByteConvertToDouble(ByteDouble));
```

```
                    i = i + 8;
                    //Mmax
                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                    RecContTypePolygonM.setMmax(
                                ByteConvertToDouble(ByteDouble));
                    i = i + 8;
                    j = 0;
                    //Marray
                    while (j < (RecContTypePolygonM.getNumPoints())){
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        RecContTypePolygonM.setMarray(j,
                            ByteConvertToDouble(ByteDouble));
                        i = i + 8;
                        j++;
                    }
                }
                //RecordContents <-- PolygonMShape
                RecordContents = new PolygonMShape();
                RecordContents = RecContTypePolygonM;
                break;

            case 28:
                //"MultiPointM";
                MultiPointMShape RecContTypeMultiPointM = new MultiPointMShape();
                while (i < 36){ //Box
                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                    RecContTypeMultiPointM.setBox(j,
                                        ByteConvertToDouble(ByteDouble));
                    i = i + 8;
                    j++;
                }
                //NumPoints
                System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                RecContTypeMultiPointM.setNumPoints(ByteConvertToInteger(ByteInteger));
                i = i + 4;
                j = 0;
                //Points
                X = (i + (16 * RecContTypeMultiPointM.getNumPoints()));
                while (i < X){ // Y = X + (16 * NumPoints)
                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                    double X1 = ByteConvertToDouble(ByteDouble);
                    i = i + 8;
                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                    double Y1 = ByteConvertToDouble(ByteDouble);
                    i = i + 8;
                    RecContTypeMultiPointM.setPoints(j, X1, Y1);
                    j++;
                }
                if (i < (VcByte.length)) { // Mmin, Mmax, Marray Opcional
                    //Mmin
                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                    RecContTypeMultiPointM.setMmin(
                                ByteConvertToDouble(ByteDouble));
                    i = i + 8;
                    //Mmax
                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                    RecContTypeMultiPointM.setMmax(
                                ByteConvertToDouble(ByteDouble));
                    i = i + 8;
```

```
                    j = 0;
                    //Marray
                    while (j < (RecContTypeMultiPointM.getNumPoints())){
                            System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                            RecContTypeMultiPointM.setMarray(j,
                                    ByteConvertToDouble(ByteDouble));
                            i = i + 8;
                            j++;
                    }
            }
            //RecordContents <-- MultiPointMShape
            RecordContents = new MultiPointMShape();
            RecordContents = RecContTypeMultiPointM;
            break;

    case 31:
            //"MultiPatch";
            MultiPatchShape RecContTypeMultiPatch = new MultiPatchShape();
            while (i < 36){ //Box
                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                    RecContTypeMultiPatch.setBox(j,
                            ByteConvertToDouble(ByteDouble));
                    i = i + 8;
                    j++;
            }
            //NumParts
            System.arraycopy(VcByte, i, ByteInteger, 0, 4);
            RecContTypeMultiPatch.setNumParts(ByteConvertToInteger(ByteInteger));
            i = i + 4;
            //NumPoints
            System.arraycopy(VcByte, i, ByteInteger, 0, 4);
            RecContTypeMultiPatch.setNumPoints(ByteConvertToInteger(ByteInteger));
            i = i + 4;
            j = 0;
            //Parts
            // W = 44 + (4 * NumParts)
            while (i < (44 + (4 * RecContTypeMultiPatch.getNumParts()))){
                    System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                    RecContTypeMultiPatch.setParts(j,
                                    ByteConvertToInteger(ByteInteger));
                    i = i + 4;
                    j++;
            }
            j = 0;
            //PartType
            W = (i + (4 * RecContTypeMultiPatch.getNumParts()));
            while (i < W){ // X = W + (4 * NumParts)
                    System.arraycopy(VcByte, i, ByteInteger, 0, 4);
                    RecContTypeMultiPatch.setPartType(j,
                                    ByteConvertToInteger(ByteInteger));
                    i = i + 4;
                    j++;
            }
            j = 0;
            //Points
            X = (i + (16 * RecContTypeMultiPatch.getNumPoints()));
            while (i < X){ // Y = X + (16 * NumPoints)
                    System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                    double X1 = ByteConvertToDouble(ByteDouble);
```

```
                        i = i + 8;
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        double Y1 = ByteConvertToDouble(ByteDouble);
                        i = i + 8;
                        RecContTypeMultiPatch.setPoints(j, X1, Y1);
                        j++;
                }
                //Zmin
                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                RecContTypeMultiPatch.setZmin(ByteConvertToDouble(ByteDouble));
                i = i + 8;
                //Zmax
                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                RecContTypeMultiPatch.setZmax(ByteConvertToDouble(ByteDouble));
                i = i + 8;
                j = 0;
                //Zarray
                Y = (i + (8 * RecContTypeMultiPatch.getNumPoints()));
                while (i < Y){ // Z = Y + (8 * NumPoints)
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        RecContTypeMultiPatch.setZarray(j,
                                        ByteConvertToDouble(ByteDouble));
                        i = i + 8;
                        j++;
                }
                if (i < (VcByte.length)) { // Mmin, Mmax, Marray Opcional
                        //Mmin
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        RecContTypeMultiPatch.setMmin(
                                ByteConvertToDouble(ByteDouble));
                        i = i + 8;
                        //Mmax
                        System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                        RecContTypeMultiPatch.setMmax(
                                ByteConvertToDouble(ByteDouble));
                        i = i + 8;
                        j = 0;
                        //Marray
                        while (j < (RecContTypeMultiPatch.getNumPoints())){
                                System.arraycopy(VcByte, i, ByteDouble, 0, 8);
                                RecContTypeMultiPatch.setMarray(j,
                                        ByteConvertToDouble(ByteDouble));
                                i = i + 8;
                                j++;
                        }
                }
                //RecordContents <-- MultiPatchShape
                RecordContents = new MultiPatchShape();
                RecordContents = RecContTypeMultiPatch;
                break;

        default:
                //return "Tipo não definido!";
                RecordContents = new NullShape();
                //RecordContents <-- NullShape
                break;
        }
}
public void setRecordHeader(){
```

```
                RecordHeader = new Header();
        }
        public void setRecordNumber(int RecNum){
                RecordHeader.setRecordNumber(RecNum);
        }
}
```

2.4 CLASSE ShapeSHP

```
import java.io.*;
import java.awt.*;
import java.util.*;
import java.awt.geom.*;
import mestrado.shape.objeto.*;
public class ShapeSHP implements java.io.Serializable {
        static final long serialVersionUID = 78352021400115957O4L;
        private String ObjName;
        private int FileCode;
        private int Unused1;
        private int Unused2;
        private int Unused3;
        private int Unused4;
        private int Unused5;
        private int FileLength;
        private int Version;
        private int ShapeType;
        private BBox BoundingBox;
        private Vector RecordSHP;
/**
 * Shape constructor comment.
 */
public ShapeSHP() {
        super();
}
        public double getBoudingBox(int FieldType){
                switch (FieldType) {
                        case 36:
                                return BoundingBox.getXmin();

                        case 44:
                                return BoundingBox.getYmin();

                        case 52:
                                return BoundingBox.getXmax();

                        case 60:
                                return BoundingBox.getYmax();

                        case 68:
                                return BoundingBox.getZmin();

                        case 76:
                                return BoundingBox.getZmax();

                        case 84:
                                return BoundingBox.getMmin();
```

```
                case 92:
                        return BoundingBox.getMmax();

                default:
                        return 0.00;

        }
}
public int getFileCode(){
        return FileCode;
}
public int getFileLength(){
        return FileLength;
}
public String getObjName(){
        return ObjName;
}
public Object getRecordSHP(int i){
        return (this.RecordSHP.elementAt(i));
}
public int getShapeTypeCode(){
        return ShapeType;
}
public String getShapeTypeName(){
        switch (ShapeType) {
                case 0:
                        return "Null Shape";

                case 1:
                        return "Point";

                case 3:
                        return "PolyLine";

                case 5:
                        return "Polygon";

                case 8:
                        return "MultiPoint";

                case 11:
                        return "PointZ";

                case 13:
                        return "PolyLineZ";

                case 15:
                        return "PolygonZ";

                case 18:
                        return "MultiPointZ";

                case 21:
                        return "PointM";

                case 23:
                        return "PolyLineM";
```

```
                    case 25:
                        return "PolygonM";

                    case 28:
                        return "MultiPointM";

                    case 31:
                        return "MultiPatch";

                    default:
                        return "Tipo não definido!";
            }
    }
    public int getSizeRecordSHP(){
            return RecordSHP.size();
    }
    public int getUnused1(){
            return Unused1;
    }
    public int getUnused2(){
            return Unused2;
    }
    public int getUnused3(){
            return Unused3;
    }
    public int getUnused4(){
            return Unused4;
    }
    public int getUnused5(){
            return Unused5;
    }
    public int getVersion(){
            return Version;
    }
    public void newRecordSHP(){
            RecordSHP = new Vector();
    }
    public void setBoudingBox(){
            BoundingBox = new BBox();
    }
    public void setFields(int FieldType, double Value){
            switch (FieldType) {
                    case 36:
                        BoundingBox.setXmin(Value);
                        break;

                    case 44:
                        BoundingBox.setYmin(Value);
                        break;

                    case 52:
                        BoundingBox.setXmax(Value);
                        break;

                    case 60:
                        BoundingBox.setYmax(Value);
                        break;

                    case 68:
```

```
                        BoundingBox.setZmin(Value);
                        break;

                case 76:
                        BoundingBox.setZmax(Value);
                        break;

                case 84:
                        BoundingBox.setMmin(Value);
                        break;

                case 92:
                        BoundingBox.setMmax(Value);
                        break;

                default:
                        break;

        }
}
public void setFields(int FieldType, int Value){
        switch (FieldType) {
                case 0:
                        setFileCode(Value);
                        break;

                case 4:
                        setUnused1(Value);
                        break;

                case 8:
                        setUnused2(Value);
                        break;

                case 12:
                        setUnused3(Value);
                        break;

                case 16:
                        setUnused4(Value);
                        break;

                case 20:
                        setUnused5(Value);
                        break;

                case 24:
                        setFileLength(Value);
                        break;

                case 28:
                        setVersion(Value);
                        break;

                case 32:
                        setShapeType(Value);
                        break;

                default:
```

```
                        break;

                }
        }
        public void setFileCode(int FC){
                FileCode = FC;
        }
        public void setFileLength(int FLen){
                FileLength = FLen;
        }
        public void setObjName(String Tp){
                ObjName = Tp;
        }
        public GeneralPath setPathToPaint(double size){
                GeneralPath PathObj = new GeneralPath();
                for (int i = 0; i < this.getSizeRecordSHP(); i++){
                        int j = 0;
                        int PartEnd = 0;
                        Record Rec1 = new Record();
                        Rec1 = (Record)this.getRecordSHP(i);
                        switch (Rec1.getTypeContent()) {
                                case 0:
                                        //"Null Shape";
                                        break;

                                case 1:
                                        //"Point";
                                        PointShape ObjCont1 = new PointShape();
                                        ObjCont1 = (PointShape)Rec1.getRecordContent();
                                        PathObj.moveTo((float)((ObjCont1.getX()*size)),
                                                (float)((ObjCont1.getY()*(-size))));
                                        PathObj.lineTo((float)((ObjCont1.getX()*size)+3),
                                                (float)(ObjCont1.getY()*(-size)));
                                        PathObj.lineTo((float)((ObjCont1.getX()*size)+3),
                                                (float)((ObjCont1.getY()*(-size))+3));
                                        PathObj.lineTo((float)((ObjCont1.getX()*size)),
                                                (float)((ObjCont1.getY()*(-size))+3));
                                        PathObj.closePath();
                                        break;

                                case 3:
                                        //"PolyLine";
                                        PolyLineShape ObjCont3 = new PolyLineShape();
                                        ObjCont3 = (PolyLineShape)Rec1.getRecordContent();
                                        while (j < ObjCont3.getNumParts()){
                                                if ((j + 1) < ObjCont3.getNumParts()){
                                                        PartEnd = ObjCont3.getParts(j + 1);
                                                }
                                                else{
                                                        PartEnd = ObjCont3.getNumPoints();
                                                }
                                                PathObj.moveTo((float)(ObjCont3.getPointsX(
                                                        ObjCont3.getParts(j)) * size),
                                                        (float)(-(ObjCont3.getPointsY(
                                                        ObjCont3.getParts(j)) * size)));
                                                for (int j1 = (ObjCont3.getParts(j) + 1);
                                                                        j1 < PartEnd; j1++){
                                                        PathObj.lineTo(((float)(
                                                                ObjCont3.getPointsX(j1)* size)),
```

```
                                        ((float)(-(ObjCont3.getPointsY(j1)
                                        * size))));
                        }
                        j++;
                }
                break;

        case 5:
                //"Polygon"
                PolygonShape ObjCont5 = new PolygonShape();
                ObjCont5 = (PolygonShape)Rec1.getRecordContent();
                while (j < ObjCont5.getNumParts()){
                        if ((j + 1) < ObjCont5.getNumParts()){
                                PartEnd = ObjCont5.getParts(j + 1);
                        }
                        else{
                                PartEnd = ObjCont5.getNumPoints();
                        }
                        PathObj.moveTo((float)(ObjCont5.getPointsX(
                                ObjCont5.getParts(j)) * size),
                                (float)(-(ObjCont5.getPointsY(
                                ObjCont5.getParts(j)) * size)));
                        for (int j1 = (ObjCont5.getParts(j) + 1);
                                                        j1 < PartEnd; j1++){
                                PathObj.lineTo(((float)(
                                        ObjCont5.getPointsX(j1) * size)),
                                        ((float)(-(ObjCont5.getPointsY(j1)
                                        * size))));
                        }
                        PathObj.closePath();
                        j++;
                }
                break;


        case 8:
                //"MultiPoint";
                MultiPointShape ObjCont8 = new MultiPointShape();
                ObjCont8 = (MultiPointShape)Rec1.getRecordContent();
                for (int j1 = 0; j1 < ObjCont8.getNumPoints(); j1++){
                        PathObj.moveTo(
                                (float)((ObjCont8.getPointsX(j1)*size)),
                                (float)((ObjCont8.getPointsY(j1)*(-size))));
                        PathObj.lineTo(
                                (float)((ObjCont8.getPointsX(j1)*size)+3),
                                (float)(ObjCont8.getPointsY(j1)*(-size)));
                        PathObj.lineTo(
                                (float)((ObjCont8.getPointsX(j1)*size)+3),
                                (float)((ObjCont8.getPointsY(j1)*(-size))+3));
                        PathObj.lineTo(
                                (float)((ObjCont8.getPointsX(j1)*size)),
                                (float)((ObjCont8.getPointsY(j1)*(-size))+3));
                        PathObj.closePath();
                }
                break;

        case 11:
                //"PointZ";
                PointZShape ObjCont11 = new PointZShape();
```

```
                    ObjCont11 = (PointZShape)Rec1.getRecordContent();
                    PathObj.moveTo(
                            (float)((ObjCont11.getX()*size)),
                            (float)((ObjCont11.getY()*(-size))));
                    PathObj.lineTo(
                            (float)((ObjCont11.getX()*size)+3),
                            (float)(ObjCont11.getY()*(-size)));
                    PathObj.lineTo(
                            (float)((ObjCont11.getX()*size)+3),
                            (float)((ObjCont11.getY()*(-size))+3));
                    PathObj.lineTo(
                            (float)((ObjCont11.getX()*size)),
                            (float)((ObjCont11.getY()*(-size))+3));
                    PathObj.closePath();
                    break;

            case 13:
                    //"PolyLineZ"; // ??? = PolyLine
                    PolyLineZShape ObjCont13 = new PolyLineZShape();
                    ObjCont13 = (PolyLineZShape)Rec1.getRecordContent();
                    while (j < ObjCont13.getNumParts()){
                            if ((j + 1) < ObjCont13.getNumParts()){
                                    PartEnd = ObjCont13.getParts(j + 1);
                            }
                            else{
                                    PartEnd = ObjCont13.getNumPoints();
                            }
                            PathObj.moveTo((float)(ObjCont13.getPointsX(
                                    ObjCont13.getParts(j)) * size), (float)
                                    (-ObjCont13.getPointsY(
                                    ObjCont13.getParts(j)) * size)));
                            for (int j1 = (ObjCont13.getParts(j) + 1);
                                                            j1 < PartEnd; j1++){
                                    PathObj.lineTo((
                                    (float)(ObjCont13.getPointsX(j1) * size)),
                                            ((float)(-(ObjCont13.getPointsY(j1)
                                            * size))));
                            }
                            j++;
                    }
                    break;


            case 15:
                    //"PolygonZ"; // ??? = Polygon
                    PolygonZShape ObjCont15 = new PolygonZShape();
                    ObjCont15 = (PolygonZShape)Rec1.getRecordContent();
                    while (j < ObjCont15.getNumParts()){
                            if ((j + 1) < ObjCont15.getNumParts()){
                                    PartEnd = ObjCont15.getParts(j + 1);
                            }
                            else{
                                    PartEnd = ObjCont15.getNumPoints();
                            }
                            PathObj.moveTo((float)(ObjCont15.getPointsX(
                                    ObjCont15.getParts(j)) * size),
                                    (float)(-(ObjCont15.getPointsY(
                                    ObjCont15.getParts(j)) * size)));
                            for (int j1 = (ObjCont15.getParts(j) + 1);
```

```
                                                    j1 < PartEnd; j1++){
                                        PathObj.lineTo(((float)
                                                (ObjCont15.getPointsX(j1) * size)),
                                                ((float)(-(ObjCont15.getPointsY(j1)
                                                * size))));
                                }
                                PathObj.closePath();
                                j++;
                        }
                        break;

                case 18:
                        //"MultiPointZ"; // ??? = MultiPoint
                        MultiPointZShape ObjCont18 = new MultiPointZShape();
                        ObjCont18 = (MultiPointZShape)Rec1.getRecordContent();

                        for (int j1 = 0; j1 < ObjCont18.getNumPoints(); j1++){
                                PathObj.moveTo(
                                        (float)((ObjCont18.getPointsX(j1)*size)),
                                        (float)((ObjCont18.getPointsY(j1)*(-size))));
                                PathObj.lineTo(
                                        (float)((ObjCont18.getPointsX(j1)*size)+3),
                                        (float)(ObjCont18.getPointsY(j1)*(-size)));
                                PathObj.lineTo(
                                        (float)((ObjCont18.getPointsX(j1)*size)+3),
                                        (float)((ObjCont18.getPointsY(j1)*(-size))+3));
                                PathObj.lineTo(
                                        (float)((ObjCont18.getPointsX(j1)*size)),
                                        (float)((ObjCont18.getPointsY(j1)*(-size))+3));
                                PathObj.closePath();
                        }
                        break;

                case 21:
                        //"PointM";
                        PointMShape ObjCont21 = new PointMShape();
                        ObjCont21 = (PointMShape)Rec1.getRecordContent();
                        PathObj.moveTo(
                                (float)((ObjCont21.getX()*size)),
                                (float)((ObjCont21.getY()*(-size))));
                        PathObj.lineTo(
                                (float)((ObjCont21.getX()*size)+3),
                                (float)(ObjCont21.getY()*(-size)));
                        PathObj.lineTo(
                                (float)((ObjCont21.getX()*size)+3),
                                (float)((ObjCont21.getY()*(-size))+3));
                        PathObj.lineTo(
                                (float)((ObjCont21.getX()*size)),
                                (float)((ObjCont21.getY()*(-size))+3));
                        PathObj.closePath();
                        break;

                case 23:
                        //"PolyLineM"; // ??? = PolyLine
                        PolyLineMShape ObjCont23 = new PolyLineMShape();
                        ObjCont23 = (PolyLineMShape)Rec1.getRecordContent();
                        while (j < ObjCont23.getNumParts()){
                                if ((j + 1) < ObjCont23.getNumParts()){
                                        PartEnd = ObjCont23.getParts(j + 1);
```

```
                    }
                    else{
                        PartEnd = ObjCont23.getNumPoints();
                    }
                    PathObj.moveTo(
                        (float)(ObjCont23.getPointsX(
                        ObjCont23.getParts(j))* size),
                        (float)(-(ObjCont23.getPointsY(
                        ObjCont23.getParts(j)) * size)));
                    for (int j1 = (ObjCont23.getParts(j) + 1);
                                                j1 < PartEnd; j1++){
                        PathObj.lineTo(((float)
                            (ObjCont23.getPointsX(j1) * size)),
                            ((float)(-(ObjCont23.getPointsY(j1)
                            * size))));
                    }
                    j++;
                }
                break;


            case 25:
                //"PolygonM";// ??? = Polygon
                PolygonMShape ObjCont25 = new PolygonMShape();
                ObjCont25 = (PolygonMShape)Rec1.getRecordContent();
                while (j < ObjCont25.getNumParts()){
                    if ((j + 1) < ObjCont25.getNumParts()){
                        PartEnd = ObjCont25.getParts(j + 1);
                    }
                    else{
                        PartEnd = ObjCont25.getNumPoints();
                    }
                    PathObj.moveTo((float)(ObjCont25.getPointsX(
                        ObjCont25.getParts(j)) * size),
                        (float)(-(ObjCont25.getPointsY(
                        ObjCont25.getParts(j)) * size)));
                    for (int j1 = (ObjCont25.getParts(j) + 1);
                                            j1 < PartEnd; j1++){
                        PathObj.lineTo(((float)(
                            ObjCont25.getPointsX(j1) * size)),
                            ((float)(-(ObjCont25.getPointsY(j1)
                            * size))));
                    }
                    PathObj.closePath();
                    j++;
                }
                break;

            case 28:
                //"MultiPointM"; // ??? = MultiPoint
                MultiPointMShape ObjCont28 = new MultiPointMShape();
                ObjCont28 = (MultiPointMShape)Rec1.getRecordContent();
                for (int j1 = 0; j1 < ObjCont28.getNumPoints(); j1++){
                    PathObj.moveTo(
                        (float)((ObjCont28.getPointsX(j1)*size)),
                        (float)((ObjCont28.getPointsY(j1)*(-size))));
                    PathObj.lineTo(
                        (float)((ObjCont28.getPointsX(j1)*size)+3),
                        (float)(ObjCont28.getPointsY(j1)*(-size)));
```

```
                                        PathObj.lineTo(
                                                (float)((ObjCont28.getPointsX(j1)*size)+3),
                                                (float)((ObjCont28.getPointsY(j1)*(-size))+3));
                                        PathObj.lineTo(
                                                (float)((ObjCont28.getPointsX(j1)*size)),
                                                (float)((ObjCont28.getPointsY(j1)*(-size))+3));
                                        PathObj.closePath();
                                }
                                break;


                        case 31:
                                //"MultiPatch";  // ??? = Polygon
                                MultiPatchShape ObjCont31 = new MultiPatchShape();
                                ObjCont31 = (MultiPatchShape)Rec1.getRecordContent();
                                while (j < ObjCont31.getNumParts()){
                                        if ((j + 1) < ObjCont31.getNumParts()){
                                                PartEnd = ObjCont31.getParts(j + 1);
                                        }
                                        else{
                                                PartEnd = ObjCont31.getNumPoints();
                                        }
                                        PathObj.moveTo(
                                                (float)(ObjCont31.getPointsX(
                                                ObjCont31.getParts(j)) * size),
                                                (float)(-(ObjCont31.getPointsY(
                                                ObjCont31.getParts(j)) * size)));
                                        for (int j1 = (ObjCont31.getParts(j) + 1);
                                                                        j1 < PartEnd; j1++){
                                                PathObj.lineTo(((float)
                                                        (ObjCont31.getPointsX(j1) * size)),
                                                        ((float)(-(ObjCont31.getPointsY(j1)
                                                        * size))));
                                        }
                                        PathObj.closePath();
                                        j++;
                                }
                                break;


                        default:
                                //return "Tipo não definido!";
                                break;
                        }
                }
                return(PathObj);

        }
        public void setRecordSHP(Record Rec){
                RecordSHP.addElement(Rec);

        }
        public void setShapeType(int Tp){
                ShapeType = Tp;
        }
        public void setUnused1(int Un){
                Unused1 = Un;
        }
        public void setUnused2(int Un){
```

```
                Unused2 = Un;
        }
        public void setUnused3(int Un){
                Unused3 = Un;
        }
        public void setUnused4(int Un){
                Unused4 = Un;
        }
        public void setUnused5(int Un){
                Unused5 = Un;
        }
        public void setVersion(int Ver){
                Version = Ver;
        }
}
```

3 PACOTE SISTEMA

3.1 CLASSE CompObject

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.lang.*;
import java.awt.geom.*;
import mestrado.shape.objeto.*;
import mestrado.shape.arquivo.shp.*;

public class CompObject extends Frame {
        ShapeSHP ObjetoSHP;
        private Button ivjButton1 = null;
        private Button ivjButton11 = null;
        private Panel ivjContentsPane = null;
        IvjEventHandler ivjEventHandler = new IvjEventHandler();
        private Label ivjLabel1 = null;
        private Label ivjLabel2 = null;
        private TextField ivjTextField1 = null;
        private TextField ivjTextField2 = null;

class IvjEventHandler implements java.awt.event.ActionListener,
                        java.awt.event.WindowListener {
                public void actionPerformed(java.awt.event.ActionEvent e) {
                        if (e.getSource() == CompObject.this.getButton1())
                                connEtoC2();
                        if (e.getSource() == CompObject.this.getButton11())
                                connEtoC3();
                };
                public void windowActivated(java.awt.event.WindowEvent e) {};
                public void windowClosed(java.awt.event.WindowEvent e) {};
                public void windowClosing(java.awt.event.WindowEvent e) {
                        if (e.getSource() == CompObject.this)
                                connEtoC1(e);
                };
                public void windowDeactivated(java.awt.event.WindowEvent e) {};
                public void windowDeiconified(java.awt.event.WindowEvent e) {};
                public void windowIconified(java.awt.event.WindowEvent e) {};
```

```
                public void windowOpened(java.awt.event.WindowEvent e) {};
        };
public CompObject() {
        super();
        initialize();
}
public CompObject(String title) {
        super(title);
}
public void button1_ActionEvents() {
        String fileName;
        FileDialog iFileDialog = new FileDialog(this);
        iFileDialog.setVisible(true);
        if (iFileDialog.getFile() == null){
                fileName = "";
        }
        else{
                fileName = iFileDialog.getFile();
                decomporObjeto(iFileDialog.getDirectory() + fileName);
        }
        ivjTextField1.setText(fileName);
        return;
}
public void button11_ActionEvents() {
        comporObjeto();
        return;
}
        public int ByteConvert(int By){
                if (By < 0){
                  By = (By ^ -256);
                }
                return(By);
        }
        public long ByteConvert(long By){
                if (By < 0){
                  By = (By ^ -256);
                }
                return(By);
        }
public void comporObjeto() {
        try{
                FileOutputStream fos = new FileOutputStream(ivjTextField2.getText() + ".ser");
                ObjectOutputStream oo = new ObjectOutputStream(fos);
                oo.writeObject(ObjetoSHP);
                oo.flush();
                fos.close();
        } catch (Exception e){}
        return;
}
private void connEtoC1(java.awt.event.WindowEvent arg1) {
        try {
                // user code begin {1}
                // user code end
                this.dispose();
                // user code begin {2}
                // user code end
        } catch (java.lang.Throwable ivjExc) {
                // user code begin {3}
                // user code end
```

```
                handleException(ivjExc);
        }
}
private void connEtoC2() {
        try {
                // user code begin {1}
                // user code end
                this.button1_ActionEvents();
                // user code begin {2}
                // user code end
        } catch (java.lang.Throwable ivjExc) {
                // user code begin {3}
                // user code end
                handleException(ivjExc);
        }
}
private void connEtoC3() {
        try {
                // user code begin {1}
                // user code end
                this.button11_ActionEvents();
                // user code begin {2}
                // user code end
        } catch (java.lang.Throwable ivjExc) {
                // user code begin {3}
                // user code end
                handleException(ivjExc);
        }
}
        public void decomporObjeto(String filePath) {
        try{
                FileInputStream fis = new FileInputStream(filePath);
                int i = 0;
                double b = 0.00;
                ObjetoSHP = new ShapeSHP();
                ObjetoSHP.setObjName("Teste");
                while (i < 28){ // Byte Big Order - Integer
                        ObjetoSHP.setFields(i, ((fis.read() << 24) |
                                (fis.read() << 16) + (fis.read() << 8) + fis.read()));
                        i = i + 4;
                }
                while (i < 36){ // Byte Little Order - Integer
                        ObjetoSHP.setFields(i, (fis.read() + (fis.read() << 8) +
                                (fis.read() << 16) | (fis.read() << 24)));
                        i = i + 4;
                }
                ObjetoSHP.setBoudingBox();
                while (i < 100){ // Byte Little Order - Double
                                ObjetoSHP.setFields(i, new Double(b).longBitsToDouble((
                                (ByteConvert(fis.read())) + (ByteConvert(fis.read()) << 8) +
                                (ByteConvert(fis.read()) << 16)      +
                                (ByteConvert((long)fis.read()) << 24) +
                                (ByteConvert((long)fis.read()) << 32) +
                                (ByteConvert((long)fis.read()) << 40) +
                                (ByteConvert((long)fis.read()) << 48) +
                                ((long)fis.read() << 56))));
                        i = i + 8;
                }
                int FileEnd = (ObjetoSHP.getFileLength() * 2);
```

```
                ObjetoSHP.newRecordSHP();
                while (i < FileEnd){ // Records
                        Record RecAux = new Record();
                        RecAux.setRecordHeader();
                        RecAux.setRecordNumber(((fis.read() << 24) |
                                (fis.read() << 16) + (fis.read() << 8) + fis.read()));
                        RecAux.setContentLength(((fis.read() << 24) | (fis.read() << 16) +
                                (fis.read() << 8) + fis.read()));
                        i = i + 8;
                        byte VecContents[] = new byte[RecAux.getContentLength() * 2];
                        fis.read(VecContents); // lê (RecAux.getContentLength() * 2) bytes
                        RecAux.setRecordContents(VecContents);
                        i = i + (RecAux.getContentLength() * 2);
                        ObjetoSHP.setRecordSHP(RecAux);
                }
                fis.close();
                } catch (Exception e) {}

        }
private java.awt.Button getButton1() {
        if (ivjButton1 == null) {
                try {
                        ivjButton1 = new java.awt.Button();
                        ivjButton1.setName("Button1");
                        ivjButton1.setBounds(20, 28, 100, 23);
                        ivjButton1.setLabel("Abrir arquivo");
                        // user code begin {1}
                        // user code end
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
        }
        return ivjButton1;
}
private java.awt.Button getButton11() {
        if (ivjButton11 == null) {
                try {
                        ivjButton11 = new java.awt.Button();
                        ivjButton11.setName("Button11");
                        ivjButton11.setBounds(287, 130, 100, 23);
                        ivjButton11.setLabel("Converter");
                        // user code begin {1}
                        // user code end
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
        }
        return ivjButton11;
}
private java.awt.Panel getContentsPane() {
        if (ivjContentsPane == null) {
                try {
                        ivjContentsPane = new java.awt.Panel();
                        ivjContentsPane.setName("ContentsPane");
                        ivjContentsPane.setLayout(null);
```

```
                        getContentsPane().add(getButton1(), getButton1().getName());
                        getContentsPane().add(getTextField1(), getTextField1().getName());
                        getContentsPane().add(getTextField2(), getTextField2().getName());
                        getContentsPane().add(getButton11(), getButton11().getName());
                        getContentsPane().add(getLabel1(), getLabel1().getName());
                        getContentsPane().add(getLabel2(), getLabel2().getName());
                        // user code begin {1}
                        // user code end
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
        }
        return ivjContentsPane;
}
private java.awt.Label getLabel1() {
        if (ivjLabel1 == null) {
                try {
                        ivjLabel1 = new java.awt.Label();
                        ivjLabel1.setName("Label1");
                        ivjLabel1.setText("Nome do arquivo *.shp");
                        ivjLabel1.setBounds(129, 10, 214, 19);
                        // user code begin {1}
                        // user code end
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
        }
        return ivjLabel1;
}
private java.awt.Label getLabel2() {
        if (ivjLabel2 == null) {
                try {
                        ivjLabel2 = new java.awt.Label();
                        ivjLabel2.setName("Label2");
                        ivjLabel2.setText("Digitar o nome do arquivo de destino -
                                                sem a extensão .ser");
                        ivjLabel2.setBounds(19, 105, 346, 23);
                        // user code begin {1}
                        // user code end
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
        }
        return ivjLabel2;
}
private java.awt.TextField getTextField1() {
        if (ivjTextField1 == null) {
                try {
                        ivjTextField1 = new java.awt.TextField();
                        ivjTextField1.setName("TextField1");
                        ivjTextField1.setBounds(129, 28, 263, 23);
                        // user code begin {1}
                        // user code end
```

```
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
        }
        return ivjTextField1;
}
private java.awt.TextField getTextField2() {
        if (ivjTextField2 == null) {
                try {
                        ivjTextField2 = new java.awt.TextField();
                        ivjTextField2.setName("TextField2");
                        ivjTextField2.setBounds(20, 130, 263, 23);
                        // user code begin {1}
                        // user code end
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
        }
        return ivjTextField2;
}
private void handleException(java.lang.Throwable exception) {

}
private void initConnections() throws java.lang.Exception {
        // user code begin {1}
        // user code end
        this.addWindowListener(ivjEventHandler);
        getButton1().addActionListener(ivjEventHandler);
        getButton11().addActionListener(ivjEventHandler);
}
private void initialize() {
        try {
                // user code begin {1}
                // user code end
                setName("CompObject");
                setLayout(new java.awt.BorderLayout());
                setSize(426, 200);
                setTitle("Aplicativo para converter arquivos *.shp");
                add(getContentsPane(), "Center");
                initConnections();
        } catch (java.lang.Throwable ivjExc) {
                handleException(ivjExc);
        }
        // user code begin {2}
        // user code end
}
public static void main(java.lang.String[] args) {
        try {
                CompObject aCompObject;
                aCompObject = new CompObject();
                aCompObject.addWindowListener(new java.awt.event.WindowAdapter() {
                        public void windowClosing(java.awt.event.WindowEvent e) {
                                System.exit(0);
                        };
                });
```

```
                aCompObject.setVisible(true);
        } catch (Throwable exception) {
                System.err.println("Exception occurred in main() of java.awt.Frame");
                exception.printStackTrace(System.out);
        }
}
}
```

3.2 CLASSE MapViewApplet

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class MapViewApplet extends Applet implements MouseListener,
                        MouseMotionListener, WindowListener, Runnable {
        Thread thread;
        private javax.swing.JScrollBar ivjJScrollBar1 = null;
        private javax.swing.JScrollBar ivjJScrollBar2 = null;
        private TestApplet ivjTestApplet1 = null;
        private javax.swing.JButton ivjJButton1 = null;
        private javax.swing.JButton ivjJButton2 = null;
        private javax.swing.JButton ivjJButton3 = null;
        private javax.swing.JButton ivjJButton4 = null;
        private int Valor;
        IvjEventHandler ivjEventHandler = new IvjEventHandler();
        private Choice ivjChoice1 = null;
        private TextField ivjTextField1 = null;
        private TextField ivjTextField11 = null;
        private Label ivjLabel1 = null;
        private Label ivjLabel11 = null;

class IvjEventHandler implements java.awt.event.ActionListener,
                        java.awt.event.AdjustmentListener, java.awt.event.ItemListener,
                        java.awt.event.MouseListener, java.awt.event.MouseMotionListener {
                public void actionPerformed(java.awt.event.ActionEvent e) {
                        if (e.getSource() == MapViewApplet.this.getJButton1())
                                connEtoM1(e);
                        if (e.getSource() == MapViewApplet.this.getJButton2())
                                connEtoM2(e);
                        if (e.getSource() == MapViewApplet.this.getJButton3())
                                connEtoM3(e);
                        if (e.getSource() == MapViewApplet.this.getJButton3())
                                connEtoM4(e);
                        if (e.getSource() == MapViewApplet.this.getJButton3())
                                connEtoM5(e);
                        if (e.getSource() == MapViewApplet.this.getJButton4())
                                connEtoM6(e);
                };
                public void adjustmentValueChanged(java.awt.event.AdjustmentEvent e) {
                        if (e.getSource() == MapViewApplet.this.getJScrollBar2())
                                connEtoC1(e);
                        if (e.getSource() == MapViewApplet.this.getJScrollBar1())
                                connEtoC2(e);
                };
                public void itemStateChanged(java.awt.event.ItemEvent e) {
                        if (e.getSource() == MapViewApplet.this.getChoice1())
```

```
                        connEtoC5(e);
                };
                public void mouseClicked(java.awt.event.MouseEvent e) {
                        if (e.getSource() == MapViewApplet.this.getJScrollBar1())
                                connEtoC3();
                        if (e.getSource() == MapViewApplet.this.getJScrollBar2())
                                connEtoC4();
                };
                public void mouseDragged(java.awt.event.MouseEvent e) {};
                public void mouseEntered(java.awt.event.MouseEvent e) {
                        if (e.getSource() == MapViewApplet.this.getJScrollBar1())
                                connEtoC3();
                        if (e.getSource() == MapViewApplet.this.getJScrollBar2())
                                connEtoC4();
                };
                public void mouseExited(java.awt.event.MouseEvent e) {
                        if (e.getSource() == MapViewApplet.this.getJScrollBar1())
                                connEtoC3();
                        if (e.getSource() == MapViewApplet.this.getJScrollBar2())
                                connEtoC4();
                };
                public void mouseMoved(java.awt.event.MouseEvent e) {
                        if (e.getSource() == MapViewApplet.this.getTestApplet1())
                                connEtoC6(e);
                };
                public void mousePressed(java.awt.event.MouseEvent e) {
                        if (e.getSource() == MapViewApplet.this.getJScrollBar1())
                                connEtoC3();
                        if (e.getSource() == MapViewApplet.this.getJScrollBar2())
                                connEtoC4();
                };
                public void mouseReleased(java.awt.event.MouseEvent e) {
                        if (e.getSource() == MapViewApplet.this.getJScrollBar1())
                                connEtoC3();
                        if (e.getSource() == MapViewApplet.this.getJScrollBar2())
                                connEtoC4();
                };
        };
public void choice1_ItemStateChanged(java.awt.event.ItemEvent itemEvent) {
        ivjTestApplet1.setCor(ivjChoice1.getSelectedIndex());
        return;
}
private void connEtoC1(java.awt.event.AdjustmentEvent arg1) {
        try {
                // user code begin {1}
                // user code end
                this.setDeslocX();
                // user code begin {2}
                // user code end
        } catch (java.lang.Throwable ivjExc) {
                // user code begin {3}
                // user code end
                handleException(ivjExc);
        }
}
private void connEtoC2(java.awt.event.AdjustmentEvent arg1) {
        try {
                // user code begin {1}
                // user code end
```

```
                this.setDeslocY();
                // user code begin {2}
                // user code end
        } catch (java.lang.Throwable ivjExc) {
                // user code begin {3}
                // user code end
                handleException(ivjExc);
        }
}
private void connEtoC3() {
        try {
                // user code begin {1}
                // user code end
                this.setValue1();
                // user code begin {2}
                // user code end
        } catch (java.lang.Throwable ivjExc) {
                // user code begin {3}
                // user code end
                handleException(ivjExc);
        }
}
private void connEtoC4() {
        try {
                // user code begin {1}
                // user code end
                this.setValue2();
                // user code begin {2}
                // user code end
        } catch (java.lang.Throwable ivjExc) {
                // user code begin {3}
                // user code end
                handleException(ivjExc);
        }
}
private void connEtoC5(java.awt.event.ItemEvent arg1) {
        try {
                // user code begin {1}
                // user code end
                this.choice1_ItemStateChanged(arg1);
                // user code begin {2}
                // user code end
        } catch (java.lang.Throwable ivjExc) {
                // user code begin {3}
                // user code end
                handleException(ivjExc);
        }
}
private void connEtoC6(java.awt.event.MouseEvent arg1) {
        try {
                // user code begin {1}
                // user code end
                this.testApplet1_MouseMoved2(arg1);
                // user code begin {2}
                // user code end
        } catch (java.lang.Throwable ivjExc) {
                // user code begin {3}
                // user code end
                handleException(ivjExc);
```

```
        }
}
private void connEtoM1(java.awt.event.ActionEvent arg1) {
        try {
                // user code begin {1}
                // user code end
                getTestApplet1().setZoomMais();
                // user code begin {2}
                // user code end
        } catch (java.lang.Throwable ivjExc) {
                // user code begin {3}
                // user code end
                handleException(ivjExc);
        }
}
private void connEtoM2(java.awt.event.ActionEvent arg1) {
        try {
                // user code begin {1}
                // user code end
                getTestApplet1().setZoomMenos();
                // user code begin {2}
                // user code end
        } catch (java.lang.Throwable ivjExc) {
                // user code begin {3}
                // user code end
                handleException(ivjExc);
        }
}
private void connEtoM3(java.awt.event.ActionEvent arg1) {
        try {
                // user code begin {1}
                // user code end
                getTestApplet1().reset();
                // user code begin {2}
                // user code end
        } catch (java.lang.Throwable ivjExc) {
                // user code begin {3}
                // user code end
                handleException(ivjExc);
        }
}
private void connEtoM4(java.awt.event.ActionEvent arg1) {
        try {
                // user code begin {1}
                // user code end
                getJScrollBar1().setValue(arg1.getModifiers());
                // user code begin {2}
                // user code end
        } catch (java.lang.Throwable ivjExc) {
                // user code begin {3}
                // user code end
                handleException(ivjExc);
        }
}
private void connEtoM5(java.awt.event.ActionEvent arg1) {
        try {
                // user code begin {1}
                // user code end
                getJScrollBar2().setValue(arg1.getModifiers());
```

```
                // user code begin {2}
                // user code end
        } catch (java.lang.Throwable ivjExc) {
                // user code begin {3}
                // user code end
                handleException(ivjExc);
        }
}
private void connEtoM6(java.awt.event.ActionEvent arg1) {
        try {
                // user code begin {1}
                // user code end
                getTestApplet1().setPrint();
                // user code begin {2}
                // user code end
        } catch (java.lang.Throwable ivjExc) {
                // user code begin {3}
                // user code end
                handleException(ivjExc);
        }
}
public void destroy() {
        super.destroy();

        // insert code to release resources here
}
public String getAppletInfo() {
        return "MapViewApplet\n" +
                "\n" +
                "Insert the type's description here.\n" +
                "Creation date: (07/11/2000 23:12:40)\n" +
                "@author: \n" +
                "";
}
private java.awt.Choice getChoice1() {
        if (ivjChoice1 == null) {
                try {
                        ivjChoice1 = new java.awt.Choice();
                        ivjChoice1.setName("Choice1");
                        ivjChoice1.setFont(new java.awt.Font("dialog.bold", 1, 14));
                        ivjChoice1.setBackground(java.awt.Color.lightGray);
                        ivjChoice1.setBounds(402, 20, 100, 26);
                        // user code begin {1}
                        ivjChoice1.insert("Amarelo",0);
                        ivjChoice1.insert("Azul",1);
                        ivjChoice1.insert("Cinza",2);
                        ivjChoice1.insert("Verde",3);
                        ivjChoice1.insert("Vermelho",4);
                        // user code end
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
        }
        return ivjChoice1;
}
private javax.swing.JButton getJButton1() {
        if (ivjJButton1 == null) {
```

```
                try {
                        ivjJButton1 = new javax.swing.JButton();
                        ivjJButton1.setName("JButton1");
                        ivjJButton1.setText("Zoom (+)");
                        ivjJButton1.setBounds(2, 20, 100, 25);
                        // user code begin {1}
                        // user code end
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
        }
        return ivjJButton1;
}
private javax.swing.JButton getJButton2() {
        if (ivjJButton2 == null) {
                try {
                        ivjJButton2 = new javax.swing.JButton();
                        ivjJButton2.setName("JButton2");
                        ivjJButton2.setText("Zoom (-)");
                        ivjJButton2.setBounds(102, 20, 100, 25);
                        // user code begin {1}
                        // user code end
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
        }
        return ivjJButton2;
}
private javax.swing.JButton getJButton3() {
        if (ivjJButton3 == null) {
                try {
                        ivjJButton3 = new javax.swing.JButton();
                        ivjJButton3.setName("JButton3");
                        ivjJButton3.setFont(new java.awt.Font("dialog.bold", 1, 12));
                        ivjJButton3.setText("Restaurar");
                        ivjJButton3.setBounds(202, 20, 100, 25);
                        // user code begin {1}
                        // user code end
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
        }
        return ivjJButton3;
}
private javax.swing.JButton getJButton4() {
        if (ivjJButton4 == null) {
                try {
                        ivjJButton4 = new javax.swing.JButton();
                        ivjJButton4.setName("JButton4");
                        ivjJButton4.setText("Imprimir");
                        ivjJButton4.setBounds(302, 20, 100, 25);
                        // user code begin {1}
                        // user code end
```

```
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
        }
        return ivjJButton4;
}
private javax.swing.JScrollBar getJScrollBar1() {
        if (ivjJScrollBar1 == null) {
                try {
                        ivjJScrollBar1 = new javax.swing.JScrollBar();
                        ivjJScrollBar1.setName("JScrollBar1");
                        ivjJScrollBar1.setMaximum(105);
                        ivjJScrollBar1.setBounds(599, 46, 20, 304);
                        // user code begin {1}
                        // user code end
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
        }
        return ivjJScrollBar1;
}
private javax.swing.JScrollBar getJScrollBar2() {
        if (ivjJScrollBar2 == null) {
                try {
                        ivjJScrollBar2 = new javax.swing.JScrollBar();
                        ivjJScrollBar2.setName("JScrollBar2");
                        ivjJScrollBar2.setMaximum(105);
                        ivjJScrollBar2.setBounds(0, 349, 600, 20);
                        ivjJScrollBar2.setVisibleAmount(10);
                        ivjJScrollBar2.setValue(0);
                        ivjJScrollBar2.setOrientation(javax.swing.JScrollBar.HORIZONTAL);
                        // user code begin {1}
                        // user code end
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
        }
        return ivjJScrollBar2;
}
private java.awt.Label getLabel1() {
        if (ivjLabel1 == null) {
                try {
                        ivjLabel1 = new java.awt.Label();
                        ivjLabel1.setName("Label1");
                        ivjLabel1.setFont(new java.awt.Font("dialog.bold", 1, 14));
                        ivjLabel1.setText("X: ");
                        ivjLabel1.setBounds(515, 3, 16, 20);
                        // user code begin {1}
                        // user code end
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
```

```
                }
        }
        return ivjLabel1;
}
private java.awt.Label getLabel11() {
        if (ivjLabel11 == null) {
                try {
                        ivjLabel11 = new java.awt.Label();
                        ivjLabel11.setName("Label11");
                        ivjLabel11.setFont(new java.awt.Font("dialog.bold", 1, 14));
                        ivjLabel11.setText("Y: ");
                        ivjLabel11.setBounds(515, 24, 16, 20);
                        // user code begin {1}
                        // user code end
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
        }
        return ivjLabel11;
}
private TestApplet getTestApplet1() {
        if (ivjTestApplet1 == null) {
                try {
                        Class iiCls = Class.forName("shapeobject.TestApplet");
                        ClassLoader iiClsLoader = iiCls.getClassLoader();
                        ivjTestApplet1 = (shapeobject.TestApplet)
                          java.beans.Beans.instantiate(iiClsLoader,"shapeobject.TestApplet");
                        Integer StrToInt = new Integer(getParameter("QUANTARQ"));
                        int QtArq = StrToInt.intValue();
                        ivjTestApplet1.setVecObjSHP(QtArq);
                        for (int i = 0; i < QtArq; i++){
                                Integer IntToStr = new Integer(i+1);
                                ivjTestApplet1.setFileName(
                                        getParameter("ARQNOME" + IntToStr.toString()));
                        }
                        ivjTestApplet1.setName("TestApplet1");
                        ivjTestApplet1.setBounds(2, 47, 595, 300);
                } catch (java.lang.Throwable ivjExc) {handleException(ivjExc);}
        }
        return ivjTestApplet1;
}
private java.awt.TextField getTextField1() {
        if (ivjTextField1 == null) {
                try {
                        ivjTextField1 = new java.awt.TextField();
                        ivjTextField1.setName("TextField1");
                        ivjTextField1.setBounds(533, 3, 85, 20);
                        ivjTextField1.setEditable(false);
                        // user code begin {1}
                        // user code end
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
        }
        return ivjTextField1;
```

```
}
private java.awt.TextField getTextField11() {
        if (ivjTextField11 == null) {
                try {
                        ivjTextField11 = new java.awt.TextField();
                        ivjTextField11.setName("TextField11");
                        ivjTextField11.setBounds(533, 24, 85, 20);
                        ivjTextField11.setEditable(false);
                        // user code begin {1}
                        // user code end
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
        }
        return ivjTextField11;
}
private void handleException(java.lang.Throwable exception) {

        /* Uncomment the following lines to print uncaught exceptions to stdout */
        // System.out.println("--------- UNCAUGHT EXCEPTION ---------");
        // exception.printStackTrace(System.out);
}
public void init() {
        try {
                super.init();
                setName("MapViewApplet");
                setLayout(null);
                setSize(620, 370);
                add(getJScrollBar2(), getJScrollBar2().getName());
                add(getJScrollBar1(), getJScrollBar1().getName());
                add(getTestApplet1(), getTestApplet1().getName());
                add(getJButton1(), getJButton1().getName());
                add(getJButton2(), getJButton2().getName());
                add(getJButton3(), getJButton3().getName());
                add(getJButton4(), getJButton4().getName());
                add(getChoice1(), getChoice1().getName());
                add(getTextField1(), getTextField1().getName());
                add(getTextField11(), getTextField11().getName());
                add(getLabel1(), getLabel1().getName());
                add(getLabel11(), getLabel11().getName());
                initConnections();
                // user code begin {1}
                // user code end
        } catch (java.lang.Throwable ivjExc) {
                // user code begin {2}
                // user code end
                handleException(ivjExc);
        }
}
private void initConnections() throws java.lang.Exception {
        // user code begin {1}
        // user code end
        getJButton1().addActionListener(ivjEventHandler);
        getJButton2().addActionListener(ivjEventHandler);
        getJScrollBar2().addAdjustmentListener(ivjEventHandler);
        getJScrollBar1().addAdjustmentListener(ivjEventHandler);
        getJScrollBar1().addMouseListener(ivjEventHandler);
```

```
        getJScrollBar2().addMouseListener(ivjEventHandler);
        getJButton3().addActionListener(ivjEventHandler);
        getJButton4().addActionListener(ivjEventHandler);
        getChoice1().addItemListener(ivjEventHandler);
        getTestApplet1().addMouseMotionListener(ivjEventHandler);
}
public static void main(java.lang.String[] args) {
        MapViewApplet applet = new MapViewApplet();
        java.awt.Frame frame = new java.awt.Frame("Applet");

        frame.addWindowListener(applet);
        frame.add("Center", applet);
        frame.setSize(350, 250);
        frame.show();

        applet.init();
        applet.start();
}
public void mapViewApplet_ComponentResized(java.awt.event.ComponentEvent componentEvent) {
        return;
}
public void mouseClicked(MouseEvent e) {
        System.out.println("mouseClicked");
}
public void mouseDragged(MouseEvent e) {
        System.out.println("mouseDragged");
}
public void mouseEntered(MouseEvent e) {
        System.out.println("mouseEntered");
}
public void mouseExited(MouseEvent e) {
        System.out.println("mouseExited");
}
public void mouseMoved(MouseEvent e) {
        System.out.println("mouseMoved");
}
public void mousePressed(MouseEvent e) {
        System.out.println("mousePressed");
}
public void mouseReleased(MouseEvent e) {
        System.out.println("mouseReleased");
}
public void paint(Graphics g) {
        super.paint(g);

        // insert code to paint the applet here
}
public void run() {

}
public void setDeslocX(){
        if (Valor != ivjJScrollBar2.getValue()){
                ivjTestApplet1.setDeslocX(ivjJScrollBar2.getValue());
        }
}
public void setDeslocY(){
        if (Valor != ivjJScrollBar1.getValue()){
                ivjTestApplet1.setDeslocY(ivjJScrollBar1.getValue());
        }
```

```
}
public void setValue1(){
        Valor = ivjJScrollBar1.getValue();
}
public void setValue2(){
        Valor = ivjJScrollBar2.getValue();
}
public void start() {
        if (thread == null){
                thread = new Thread(this);
                thread.start();
        }
}
public void stop() {
        thread = null;
}
public void testApplet1_MouseMoved2(java.awt.event.MouseEvent mouseEvent) {
        ivjTextField1.setText("" + (float)ivjTestApplet1.getCoordX(
                        mouseEvent.getPoint().getX()));
        ivjTextField11.setText("" + (float)ivjTestApplet1.getCoordY(
                        mouseEvent.getPoint().getY()));
        return;
}
public void windowActivated(WindowEvent e) {
        // Do nothing.
        // This method is required to comply with the WindowListener interface.
}
public void windowClosed(WindowEvent e) {
        // Do nothing.
        // This method is required to comply with the WindowListener interface.
}
public void windowClosing(WindowEvent e) {
        // The window is being closed.  Shut down the system.
        this.stop();
        System.exit(0);
}
public void windowDeactivated(WindowEvent e) {
        // Do nothing.
        // This method is required to comply with the WindowListener interface.
}
public void windowDeiconified(WindowEvent e) {
        // Do nothing.
        // This method is required to comply with the WindowListener interface.
}
public void windowIconified(WindowEvent e) {
        // Do nothing.
        // This method is required to comply with the WindowListener interface.
}
public void windowOpened(WindowEvent e) {
        // Do nothing.
        // This method is required to comply with the WindowListener interface.
}
}
```

3.3 CLASSE PrintShape

```
import java.awt.*;
import java.awt.print.*;
import java.awt.geom.*;
import mestrado.shape.arquivo.shp.*;

public class PrintShape implements Printable{

        private double CenterToPaintX;
        private double CenterToPaintY;
        private ShapeSHP VecObjSHP[];
        private GeneralPath PathObj1 = new GeneralPath();
        private int QtdeCamadas;
        private double Size;
        private Color Cor = Color.yellow;

public PrintShape() {
        super();
}
public PrintShape(double X, double Y, ShapeSHP VecSHP[],
                                                int QtShape, double Sz, Color Cr) {
        CenterToPaintX = X;
        CenterToPaintY = Y;
        VecObjSHP = new ShapeSHP[QtShape];
        VecObjSHP = VecSHP;
        QtdeCamadas = QtShape;
        Size = Sz;
        Cor = Cr;
}
public int print(Graphics g, PageFormat pf, int pageIndex) throws PrinterException{
        if (pageIndex >= 1) return Printable.NO_SUCH_PAGE;
        Graphics2D g2 = (Graphics2D) g;
        g2.translate(CenterToPaintX + 20.0, CenterToPaintY + 20.0);
        for (int i = 0; i < QtdeCamadas; i++){
                PathObj1 = VecObjSHP[i].setPathToPaint(Size);
                if (VecObjSHP[i].getShapeTypeCode() != 3 &&
                        VecObjSHP[i].getShapeTypeCode() != 13 &&
                        VecObjSHP[i].getShapeTypeCode() != 23){
                        g2.setPaint(Cor);
                        g2.fill(PathObj1);
                }
                g2.setColor(Color.black);
                g2.draw(PathObj1);
        }
        return Printable.PAGE_EXISTS;
}
}
```

3.4 CLASSE TestApplet

```
import java.applet.*;
import java.awt.*;
import java.io.*;
import java.awt.geom.*;
import java.security.*;
import mestrado.shape.objeto.*;
import mestrado.shape.arquivo.shp.*;
```

```
import java.awt.print.*;
public class TestApplet extends Applet {

        private ShapeSHP ObjetoSHP1;
        private ShapeSHP VecObjSHP[];
        private int QtdeCamadas;
        private int IndexCamadas = 0;
        private GeneralPath PathObj = new GeneralPath();
        private int ver = 0;
        private double X1;
        private double Y1;
        private double Diagonal;
        private double Size;
        private double Intervalo;
        private double CenterX = 0;
        private double CenterY = 0;
        private double CenterToPaintX = 0;
        private double CenterToPaintY = 0;
        private double DeslocX = 0;
        private double DeslocY = 0;
        private double SizeMax;
        private double SizeMin;
        private int Xmin = 36;
        private int Ymin = 44;
        private int Xmax = 52;
        private int Ymax = 60;
        private int Zmin = 68;
        private int Zmax = 76;
        private int Mmin = 84;
        private int Mmax = 92;
        private Color Cor = Color.yellow;

        public int ByteConvert(int By){
                if (By < 0){
                  By = (By ^ -256);
                }
                return(By);
        }
        public long ByteConvert(long By){
                if (By < 0){
                  By = (By ^ -256);
                }
                return(By);
        }
public String getAppletInfo() {
        return "TestApplet\n" +
                "\n" +
                "Insert the type's description here.\n" +
                "Creation date: (27/10/2000 22:47:31)\n" +
                "@author: \n" +
                "";
}
        public double getCoordX(double X){
                return (CenterX + (X/Size) + (DeslocX/Size));
        }
        public double getCoordY(double Y){
                return (CenterY - (Y/Size) - (DeslocY/Size));
        }
        public double getDeslocX(){
```

```
                return DeslocX;
        }
        public double getDeslocY(){
                return DeslocY;
        }
public void init() {
        super.init();

}
public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D)g;
        GeneralPath PathObj1 = new GeneralPath();
        if (ver != 0){
                g2.translate(CenterToPaintX, CenterToPaintY);
                for (int i = 0; i < QtdeCamadas; i++){
                        PathObj1 = VecObjSHP[i].setPathToPaint(Size);
                        if (VecObjSHP[i].getShapeTypeCode() != 3 &&
                                VecObjSHP[i].getShapeTypeCode() != 13 &&
                                VecObjSHP[i].getShapeTypeCode() != 23){
                                g2.setPaint(Cor);
                                g2.fill(PathObj1);
                        }
                        g2.setColor(Color.black);
                        g2.draw(PathObj1);
                }
        }
}
        public void reset(){
                DeslocX = 0;
                DeslocY = 0;
                setSize();
                setCenterToPaint();
                repaint();
        }
        public void setCenterToPaint(){
                CenterToPaintX = setPositivo(
                        ((CenterX * Size) + DeslocX)) + 1.0;
                CenterToPaintY = ((CenterY * Size) - DeslocY) + 1.0;
        }
        public void setCor(int i){
                switch (i) {
                        case 0: {
                                Cor = Color.yellow;
                                break;
                        }
                        case 1: {
                                Cor = Color.blue;
                                break;
                        }
                        case 2: {
                                Cor = Color.gray;
                                break;
                        }
                        case 3: {
                                Cor = Color.green;
                                break;
                        }
                        case 4: {
                                Cor = Color.red;
```

```
                                break;
                        }
                        default: {
                                Cor = Color.green;
                                break;
                        }
                };
                repaint();
        }
        public void setDeslocX(double desloc){
                DeslocX = (desloc * ((Size * X1)/100));
                setCenterToPaint();
                repaint();
        }
        public void setDeslocY(double desloc){
                DeslocY = (desloc * ((Size * Y1)/100));
                setCenterToPaint();
                repaint();
        }
public void setFileName(String nome){
        try{
                InputStream s1 = TestApplet.class.getResourceAsStream(nome);
                ObjectInputStream oo1 = new ObjectInputStream(s1);
                VecObjSHP[IndexCamadas] = (ShapeSHP) oo1.readObject();
                if (IndexCamadas == 0) {
                        ObjetoSHP1 = new ShapeSHP();
                        ObjetoSHP1 = VecObjSHP[IndexCamadas];
                        CenterX = ObjetoSHP1.getBoudingBox(Xmin);
                        CenterY = ObjetoSHP1.getBoudingBox(Ymax);
                        X1 = ObjetoSHP1.getBoudingBox(Xmax) –
                                        ObjetoSHP1.getBoudingBox(Xmin);
                        Y1 = ObjetoSHP1.getBoudingBox(Ymax) –
                                        ObjetoSHP1.getBoudingBox(Ymin);
                        Diagonal = (java.lang.Math.pow(
                                        java.lang.Math.pow(X1, 2.0) +
                                        java.lang.Math.pow(Y1, 2.0), 0.5));
                        setSize();
                        setCenterToPaint();
                }
                IndexCamadas = IndexCamadas + 1;
                s1.close();
                ver = 2;
         } catch (Exception e){ver = 0;}
}
        public double setPositivo(double Valor){
                if (Valor >= 0.00){
                        return Valor;
                }
                else{
                        return -(Valor);
                }
        }
public void setPrint(){
        PrinterJob job = PrinterJob.getPrinterJob();
        PageFormat pf = job.pageDialog(job.defaultPage());
        job.setPrintable(new PrintShape(CenterToPaintX, CenterToPaintY,
                                                VecObjSHP, QtdeCamadas, Size, Cor), pf);
        job.setCopies(1);
        if (job.printDialog()){
```

```
                try{ job.print();}
                catch (Exception e) {/* Exception */}
        }
        System.exit(0);
}
        public void setSize(){
                // (50 - (diagonal da BoudingBox)) / 2
                Intervalo = (50 - Diagonal) / 2;
                Size = (Intervalo * 2) + 1.0;
                if (Size < 0){
                        Size = 1;
                        Intervalo = Size;
                }
                SizeMin = Intervalo;
                SizeMax = Intervalo * 5;
        }
        public void setVecObjSHP(int Qt){
                QtdeCamadas = Qt;
                VecObjSHP = new ShapeSHP[Qt];
        }
        public void setZoomMais(){
                if (Size < SizeMax){
                        Size = Size + Intervalo;
                }
                setCenterToPaint();
                repaint();
        }
        public void setZoomMenos(){
                if (Size > SizeMin){
                        Size = Size - Intervalo;
                }
                setCenterToPaint();
                repaint();
        }
}
```