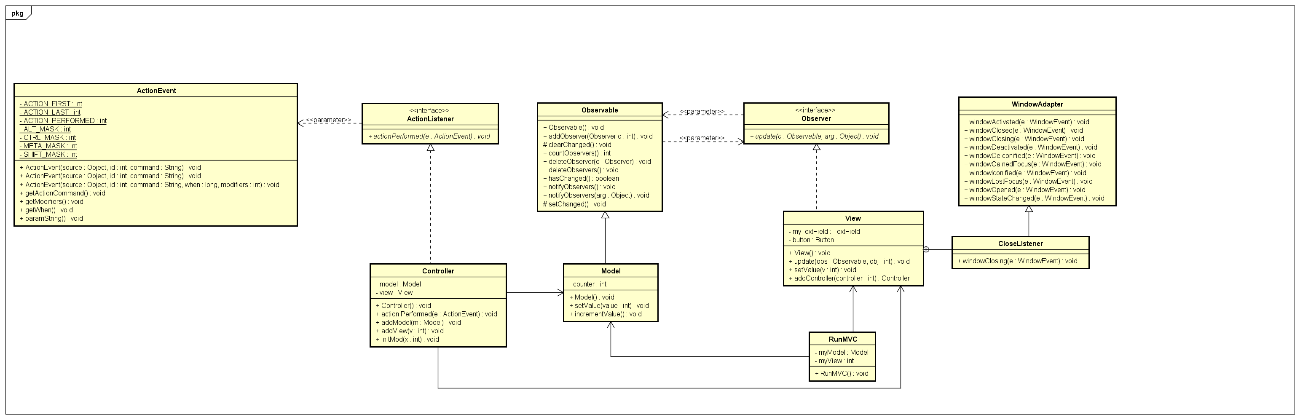


**Laboratório de Engenharia de Software**

**Lista 1**

|  |  |
| --- | --- |
| **Nome** | **RA** |
| Andressa Almeida | 1680481522041 |
| Daniel Augusto | 1680481521030 |
| Diego de Melo | 1680481521036 |
| Elias Sanai | 1680481521016 |

1. O padrão MVC (Model-View-Controller) é um padrão de projeto mais complexo que é empregado principalmente em interfaces gráficas. Em anexo há um exemplo em Java de sua aplicação no arquivo MVC.zip (de http://www.austintek.com/mvc/). Pede-se: (a) Desenhar um diagrama de classes UML correspondente ao exemplo fornecido. (b) A partir deste diagrama, identificar padrões de software conhecidos que foram empregados em sua construção.



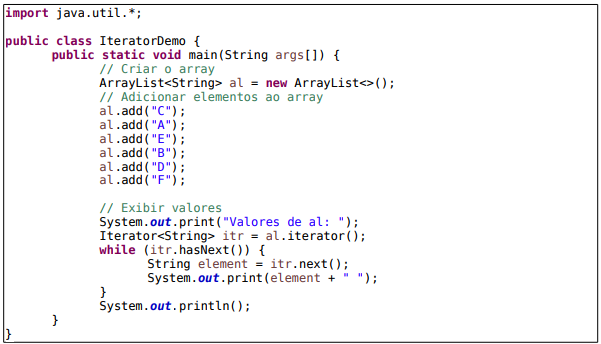
1. Elaborar, com o auxílio do padrão Visitor, um programa Java que, a partir de um nome de pasta fornecida, exiba na tela os nomes de arquivos e pastas que casem com uma expressão regular fornecida pelo usuário. Depois, elaborar um diagrama de classes UML equivalente ao programa desenvolvido.

Dica: para manipular arquivos ou diretórios do sistema operacional, utilize a classe File de Java e processe a árvore recursivamente.

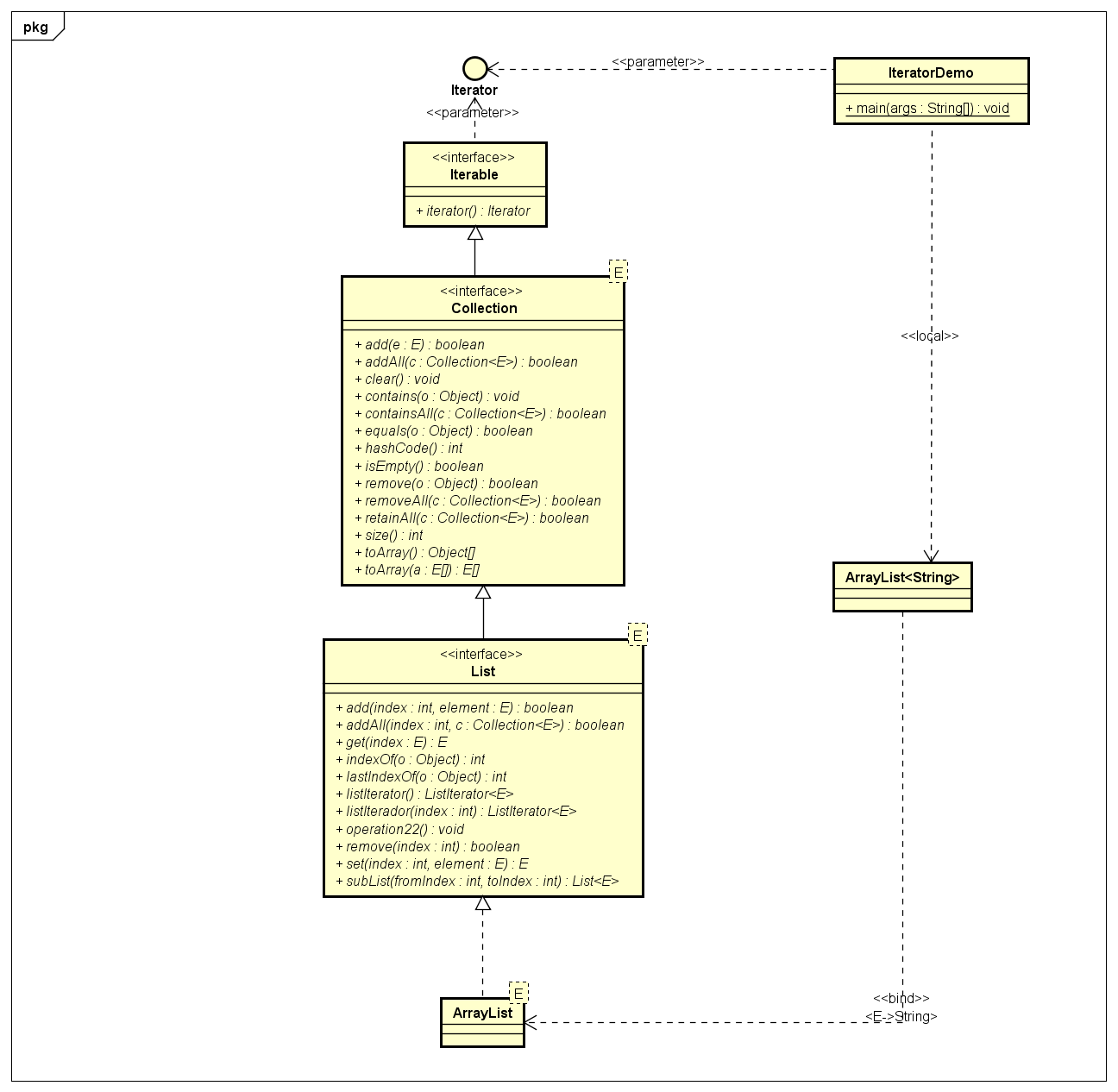
1. Elabore um programa em Java que utilize o padrão Strategy para definir uma coleção parametrizada (por exemplo, um vetor) de modo que se possa utilizar diversos métodos de ordenação sobre essa mesma coleção (o usuário define). Exemplo:

|  |
| --- |
| MyVector v = new MyVector<>(10);  v.sort(new QuickSort()); //Ordena com quicksort  //...  v.sort(new BubbleSort()); //Ordena com bubblesort |

1. Considere o programa Java a seguir:



Pede-se: pesquise o padrão Factory Method e então tente identificar seu uso nas classes de coleções e de iteradores da biblioteca do Java, justificando sua resposta com um diagrama UML do padrão e como ele foi usado para resolver o problema de iteração sobre uma coleção.



O Factory Method é um padrão de projeto que que encapsula a criação dos objetos.

O Factory Method é um padrão de projeto que que encapsula a criação dos objetos.

Isso acontece com as Coleções, pois todas implememtam a classe Iterable que contém um Iterator.

Cada coleção guarda um objeto dentro dela que é responesável por fazer a interação dos elementos da coleção independendtemente do tipo da coleção.

Igualmente ao Factory Method, essa criação acaba não sendo feita diretamente pelo programador, a criação foi encapsulada dentro da própria coleção.

1. Assuma que se tem um arquivo-texto contendo números de cartões de crédito. Cada linha deste arquivo contem os seguintes dados, separados por ponto e vírgula: o número do cartão, a data de expiração e o nome do titular do cartão. Elaborar um programa Java que utilize os seguintes nomes de classes a seguir:

◦ CreditCard;

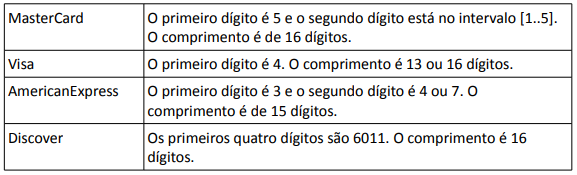
◦ VisaCC, MasterCC, AmExCC, todas subclasses de CreditCard.

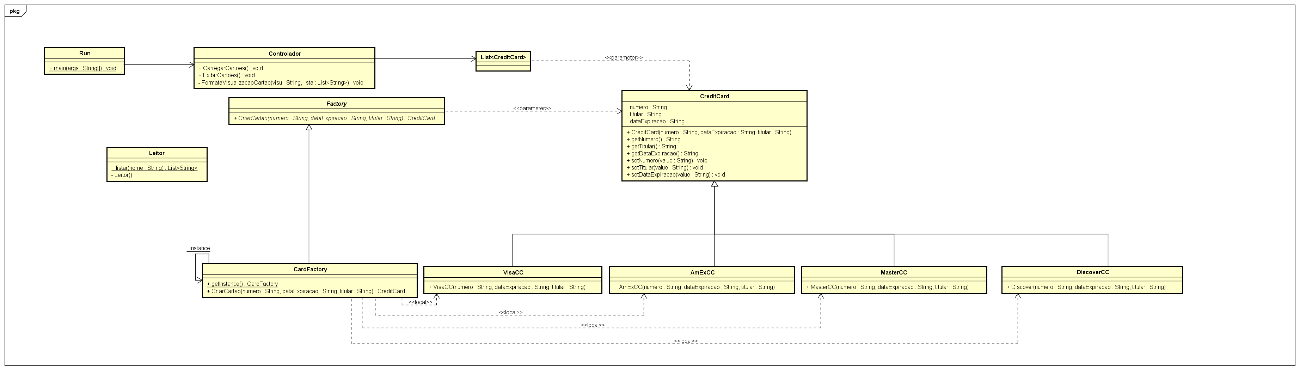
Futuramente, novas classes de cartão poderão ser adicionadas. Assim, pede-se

(a) Implementar um programa Java que leia um arquivo-texto com diversos números de cartões. O programa deverá ter as classes descritas anteriormente, além de outras que se julgar necessárias para implementar o programa.

(b) O programa deverá ler cada informação de cartão e então identificar que cartão é este, criando um objeto da classe correspondente na memória e o armazenando em uma estrutura de dados conveniente na memória. Depois, o programa deverá listar na tela os cartões identificados, agrupados por tipo (Amex, Visa etc). Tentar identificar e aplicar um ou mais padrões de projeto para serem adaptados a este projeto – não esquecer de escrever e justificar os nomes dos padrões na documentação a ser enviada.

NOTA: não é necessário utilizar o algoritmo exato para identificar os cartões (ISO 2894/ANSI 4.13). Utilize a tabela a seguir:





public class CardFactory extends Factory {

static private CardFactory instance;

//Singleton Method

static CardFactory getInstance()

{

if(instance==null)

{

instance= new CardFactory();

}

return instance;

}

/\*\*\*

\* Retorna um CreditCard com uma instância de uma de suas filhas

\* @param numero

\* @param dataExpiracao

\* @param titular

\* @return

\*/

@Override

CreditCard CriarCartao(String numero,String dataExpiracao,String titular)

{

CreditCard cartao=null;

char n[]=numero.toCharArray();

/\*\*\*

\* A partir do primeiro digito do cartão sabe-se à qual bandeira pertece

\* A seguir como é difinido a qual bandeira pertence determinado cartão

\* AmericanExpress - O primeiro dígito é 5 e o segundo dígito está no intervalo [1..5].O comprimento é de 16 dígitos.

\* Visa - O primeiro dígito é 4. O comprimento é 13 ou 16 dígitos.

\* MasterCard - O primeiro dígito é 5 e o segundo dígito está no intervalo [1..5].O comprimento é de 16 dígitos.

\* Discover - Os primeiros quatro dígitos são 6011. O comprimento é 16 dígitos.

\*/

int digito2;

if((numero.substring(0,4).equals("6014")) &&(numero.length()==16))// Discover

{

cartao = new DiscoverCC(numero, dataExpiracao, titular);

}

if(n[0]=='3' &&((n[1]=='4') || (n[1]=='7')) && numero.length()==15) //American Express

{

cartao=new AmExCC(numero, dataExpiracao, titular);

}

if(n[0]=='4' &&(numero.length()==13 || numero.length()==16))//Visa

{

cartao=new VisaCC(numero, dataExpiracao, titular);

}

digito2=Integer.parseInt(n[1]+"");

if(numero.length()==16 &&(n[0]=='5' &&(digito2>0 && digito2<6))) //MasterCard

{

cartao=new MasterCC(numero, dataExpiracao, titular);

}

return cartao;

}

}

public class AmExCC extends CreditCard{

AmExCC(String numero,String dataExpiracao,String titular)

{

super(numero,dataExpiracao,titular);

}

}

public class MasterCC extends CreditCard{

MasterCC(String numero,String dataExpiracao,String titular)

{

super(numero,dataExpiracao,titular);

}

}

public class Run {

public static void main(String args[])

{

Controlador c= new Controlador();

c.CarregarCartoes();

//c.Test();

c.ExibirCartoes();

}

}

public class Leitor {

/\*\*\*

\* Retorna um lista de Strings. Cada String corresponde a uma linha.

\* @param nome

\* @return

\* @throws FileNotFoundException

\* @throws IOException

\*/

static public List<String> listar(String nome)

{

List<String> lista= new ArrayList();

try

{

FileReader arq = new FileReader(nome);

BufferedReader lerArq = new BufferedReader(arq);

String texto= lerArq.readLine();

while(texto!=null)

{

lista.add(texto);

texto= lerArq.readLine();

}

arq.close();

}catch(FileNotFoundException ex)

{

System.out.println(ex.getMessage());

}

catch(IOException ex)

{

System.out.println(ex.getMessage());

}

return lista;

}

private Leitor(){}

}

public abstract class Factory {

abstract CreditCard CriarCartao(String numero,String dataExpiracao,String titular);

}

public class Controlador {

private List<CreditCard> cartoes= new ArrayList<>();

/\*\*\*

\* Carrega todos os cartoes dentro do arquivo na variavel 'cartoes' desta classe

\*/

void CarregarCartoes()

{

List<String> lista=Leitor.listar("cartoes.txt");// Cada linha no arquivo será adicionado na 'lista'

String[] dados;

CardFactory f= CardFactory.getInstance();

CreditCard cartao;

for(String texto:lista)

{

dados=texto.split(";");

cartao=f.CriarCartao(dados[0], dados[1], dados[2]);

if (cartao!=null)

cartoes.add(cartao);

}

}

/\*\*\*

\* Exibe todos os cartões em ordem na tela

\*/

void ExibirCartoes()

{

List<String> amex= new ArrayList<>();

List<String> visa= new ArrayList<>();

List<String> master= new ArrayList<>();

List<String> disc= new ArrayList<>();

String c;

for(CreditCard cartao:cartoes) //Utilizado para carregar cada cartão em sua lista específica

{

c="Número: "+ cartao.getNumero()+"\n";

c+="Data de Expiração: "+ cartao.getDataExpiracao()+"\n";

c+="Titular: "+ cartao.getTitular()+"\n";

if(cartao instanceof AmExCC)

{

amex.add(c);

}else if(cartao instanceof VisaCC)

{

visa.add(c);

}else if(cartao instanceof MasterCC)

{

master.add(c);

}else if(cartao instanceof DiscoverCC)

{

disc.add(c);

}

}

FormataVisualizacaoCartao("America Express", amex);

FormataVisualizacaoCartao("Visa", visa);

FormataVisualizacaoCartao("MasterCard", master);

FormataVisualizacaoCartao("Discover", disc);

}

private void FormataVisualizacaoCartao(String visu,List<String> lista)

{

System.out.println(visu);

System.out.println("---------------");

for(String l:lista)

{

System.out.println(l);

}

}

}

public class DiscoverCC extends CreditCard{

public DiscoverCC(String numero, String dataExpiracao, String titular) {

super(numero, dataExpiracao, titular);

}

}

public class VisaCC extends CreditCard{

VisaCC(String numero,String dataExpiracao,String titular)

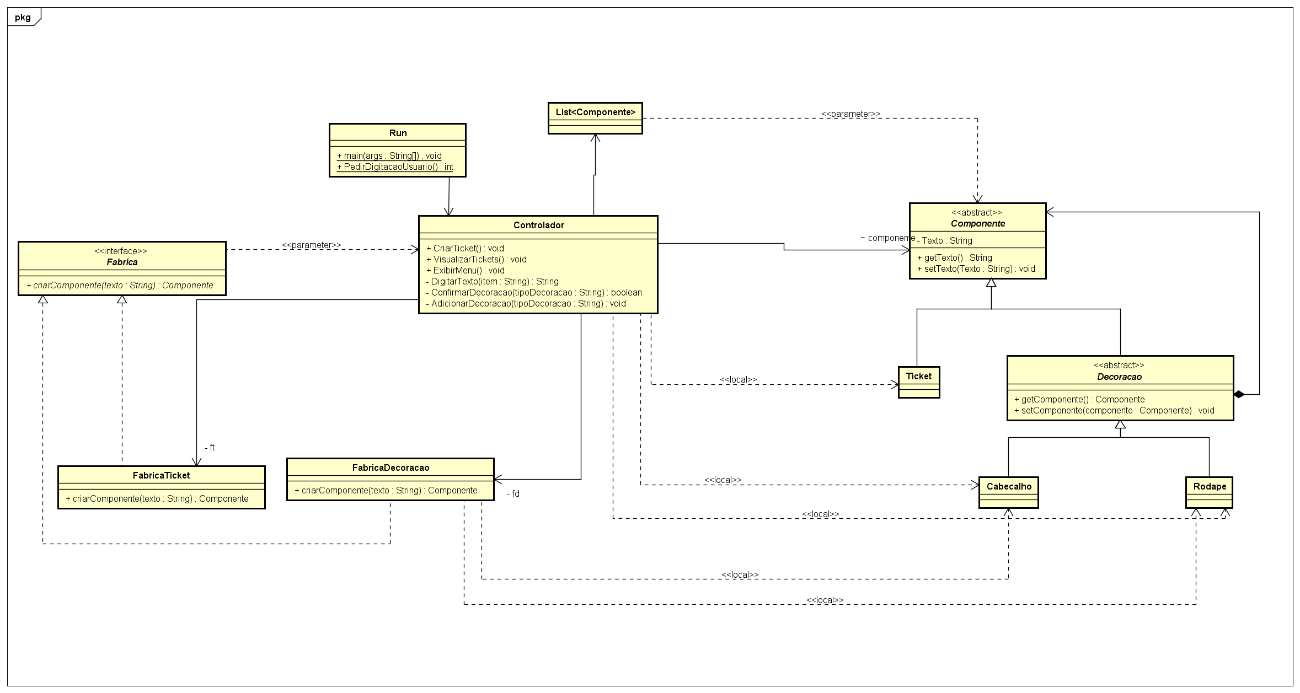
{

super(numero,dataExpiracao,titular);

}

}

1. Pesquisar como o padrão Decorator poderia ser utilizado para resolver o problema da impressão de tickets comercializados por uma empresa de promoção, sabendo que: ◦ O ticket sempre possui um texto básico que deverá aparecer quando for impresso; ◦ A empresa precisa adicionar, de modo flexível, diferentes cabeçalhos e/ou rodapés aos tickets comercializados, por imposição de seus clientes. Propor um programa Java que simule esta situação e implemente o padrão Decorator para este caso.



public class Rodape extends Decoracao{

}

public class Cabecalho extends Decoracao {

}

public abstract class Componente {

private String Texto;

/\*\*

\* @return the Texto

\*/

public String getTexto() {

return Texto;

}

/\*\*

\* @param Texto the Texto to set

\*/

public void setTexto(String Texto) {

this.Texto = Texto;

}

}

public class Run {

public static void main(String args[])

{

System.out.println("Programa decorador de Tickets");

System.out.println("-----------------------------\n");

Controlador c= new Controlador();

int opcao=0;// Valor digitado pelo usuário

boolean continuar=true;

do{

c.ExibirMenu();//Mostra opções disponíveis pro usuário

System.out.print("Digite o opção desejada: ");//Pede pro usuário digitar o que deseja

opcao=PedirDigitacaoUsuario();

switch(opcao)

{

case 1:

c.CriarTicket();

break;

case 2:

c.VisualizarTickets();

break;

case 0:

continuar=false;

break;

}

}while(continuar);

}

/\*\*\*

\* Pede para que o usuário digite algum número entre os valores permitidos

\* @return

\*/

public static int PedirDigitacaoUsuario()

{

int opcao;

Scanner scanner = new Scanner(System.in);

String entrada;

opcao = -1;

boolean opcaoInvalida=true;

while(opcaoInvalida)

{

entrada=scanner.next();

try

{

opcao=Integer.parseInt(entrada);

}

catch(NumberFormatException ex)

{

opcao=-1;

}

if(opcao>=0 && opcao<=2)// Verifica se o que foi digitado é válido

{

opcaoInvalida=false;

}else

{

opcaoInvalida=true;

System.out.print("Opção inválida,por favor digite novamente: ");

}

}

return opcao;

}

}

public interface Fabrica {

Componente criarComponente(String texto);

}

public class Controlador {

List<Componente> componentes= new ArrayList<>();

FabricaDecoracao fd=new FabricaDecoracao();

FabricaTicket ft= new FabricaTicket();

Componente componente;

/\*\*\*

\* Cria um novo ticket e o adiciona em 'componentes'

\*/

public void CriarTicket()

{

componente= ft.criarComponente("ticket");//Cria um Ticket

componente.setTexto(DigitarTexto("ticket"));//Define o texto do ticket

String tipoDecoracao;

//Adiciona um cabecalho caso o usuário deseje

tipoDecoracao="cabecalho";

if(ConfirmarDecoracao(tipoDecoracao))

AdicionarDecoracao( tipoDecoracao);

//Adiciona um rodape caso o cliente deseje

tipoDecoracao="rodape";

if(ConfirmarDecoracao(tipoDecoracao))

AdicionarDecoracao( tipoDecoracao);

componentes.add(componente);//Adiciona o novo ticket na lista

}

void VisualizarTickets()

{

String separador="---------------------------------------";

for(Componente c: componentes)

{

System.out.println(separador);

Componente n=c;

Rodape rodape;

Cabecalho cabecalho;

Ticket ticket;

if(n instanceof Rodape)

{

rodape= (Rodape) n;

n=rodape.getComponente();

}

else

rodape=null;

if (n instanceof Cabecalho)

{

cabecalho=(Cabecalho)n;

n=cabecalho.getComponente();

}

else

{

cabecalho=null;

}

ticket=(Ticket)n;

System.out.println("Ticket: "+ticket.getTexto());

if(cabecalho!=null)

{

System.out.println("Cabecalho: "+cabecalho.getTexto());

}

if(rodape!=null)

{

System.out.println("Rodape: "+rodape.getTexto());

}

}

System.out.println(separador);

}

/\*\*\*

\* Mostra o menu das opcoes disponíveis

\*/

public void ExibirMenu()

{

System.out.println("1 - Adicionar Ticket");

System.out.println("2 - Ver Tickets");

System.out.println("0 - Sair");

}

/\*\*\*

\*

\* @param item É necessário saber de onde esse texto é para que sej informado ao usuário

\* @return

\*/

public String DigitarTexto(String item)

{

String texto;

Scanner scanner = new Scanner(System.in);

System.out.println("Digite o texto do "+item+" : ");

texto=scanner.next();

return texto;

}

/\*\*\*

\* Pergunta ao usuário se deseja receber uma Decoração do tipoDecoracao

\* @param tipoDecoracao Informa o tipo da Decoracao

\* @return Retorna true caso a resposta seja afirmativa

\*/

boolean ConfirmarDecoracao(String tipoDecoracao)

{

String resposta;

Scanner scanner= new Scanner(System.in);

System.out.print("Deseja "+tipoDecoracao+"?[S/N]: ");

resposta=scanner.next();

if(resposta.trim().toUpperCase().equals("S"))

return true;

return false;

}

/\*\*\*

\*

\* @param c Componente que será decorado

\* @param tipoDecoracao Tipo de decoracao que será adicionada

\*/

private void AdicionarDecoracao(String tipoDecoracao)

{

Decoracao decoracao=(Decoracao) fd.criarComponente(tipoDecoracao);// Cria uma nova Decoracao do tipoDecoracao

decoracao.setComponente(componente); //Faz com que o componente seja decorado

componente=decoracao;// É alterado a referencia do Componente para a Decoracao porque ela encapsula o COmponente dentro dela, portanto ele não é perdido

componente.setTexto(DigitarTexto(tipoDecoracao)); //Define o texto que está na nova Decoracao

}

}

public abstract class Decoracao extends Componente{

private Componente componente;

/\*\*

\* @return the componente

\*/

public Componente getComponente() {

return componente;

}

/\*\*

\* @param componente the componente to set

\*/

public void setComponente(Componente componente) {

this.componente = componente;

}

}

public class FabricaTicket implements Fabrica{

@Override

public Componente criarComponente(String texto) {

Componente c=null;

if("TICKET".equals(texto.toUpperCase()))

c= new Ticket();

return c;

}

}

public class FabricaDecoracao implements Fabrica{

@Override

public Componente criarComponente(String texto) {

Componente c=null;

if("CABECALHO".equals(texto.toUpperCase()))

{

c= new Cabecalho();

}

else if("RODAPE".equals(texto.toUpperCase()))

{

c= new Rodape();

}

return c;

}

}

public class Ticket extends Componente {

}