

```

import edu.univasf.poo.Pacote;
import edu.univasf.poo.PacoteDoisDias;
import edu.univasf.poo.PacoteNoite;
import java.util.Scanner;
public class ManipulaPacotes
{
    public static void main(String args[])
    {
        Pacote vetor[] = new Pacote[100];
        int numeroDeElementos = 0, opcao;
        Scanner entrada = new Scanner(System.in);
        do
        {
            System.out.println ("Digite:");
            System.out.println ("1 - Inserir um pacote para entrega em dois dias;");
            System.out.println ("2 - Inserir um pacote para entrega a noite;");
            System.out.println ("3 - Imprimir enderecos para postagem e custos da postagem; ");
            System.out.println ("4 - Imprimir custos total das postagens.");
            System.out.println ("5 - Finaliza o programa.\nOpcao? ");
            opcao = entrada.nextInt();
            switch(opcao)
            {

```

```

case 1:
    if (numeroDeElementos<100)
    {
        vetor[numeroDeElementos] = new PacoteDoisDias(0.0,
PacoteDoisDias.getCustoPorQuilo(), PacoteDoisDias.getTaxaExtra());
        System.out.println ("Dados do remetente");
        System.out.print ("Nome: ");
        entrada.nextLine();
        vetor[numeroDeElementos].setNomeRemetente
(entrada.nextLine());
        System.out.print ("Endereco: ");
        vetor[numeroDeElementos].setEnderecoRemetente
(entrada.nextLine());
        System.out.print ("Cidade: ");
        vetor[numeroDeElementos].setCidadeRemetente
(entrada.nextLine());
        System.out.print ("CEP: ");
        vetor[numeroDeElementos].setCEPRemetente
(entrada.nextLong());
        System.out.println ("Dados do destinatario");
        System.out.print ("Nome: ");
        entrada.nextLine();
        vetor[numeroDeElementos].setNomeDestinatario
(entrada.nextLine());
    }

```

```

        System.out.print ("Endereco: ");
        vetor[numeroDeElementos].setEnderecoDestinatario
(entrada.nextLine());
        System.out.print ("Cidade: ");
        vetor[numeroDeElementos].setCidadeDestinatario
(entrada.nextLine());
        System.out.print ("CEP: ");
        vetor[numeroDeElementos].setCEPDestinatario
(entrada.nextLong());
        System.out.print ("\nPeso do pacote: ");
        vetor[numeroDeElementos].setPeso (entrada.nextDouble());
        numeroDeElementos++;
    }
    else
        System.out.println ("Nao ha espaco para mais pacotes.");
    break;
case 2:
    if (numeroDeElementos<100)
    {
        vetor[numeroDeElementos] = new PacoteNoite(0.0,
PacoteNoite.getCustoPorQuilo(), PacoteNoite.getTaxaExtraPorQuilo());
        System.out.println ("Dados do remetente");
        System.out.print ("Nome: ");
        entrada.nextLine();

```

```
        vetor[numeroDeElementos].setNomeRemetente
(entrada.nextLine());
        System.out.print ("Endereco: ");
        vetor[numeroDeElementos].setEnderecoRemetente
(entrada.nextLine());
        System.out.print ("Cidade: ");
        vetor[numeroDeElementos].setCidadeRemetente
(entrada.nextLine());
        System.out.print ("CEP: ");
        vetor[numeroDeElementos].setCEPRemetente
(entrada.nextLong());
        System.out.println ("Dados do destinatario");
        System.out.print ("Nome: ");
        entrada.nextLine();
        vetor[numeroDeElementos].setNomeDestinatario
(entrada.nextLine());
        System.out.print ("Endereco: ");
        vetor[numeroDeElementos].setEnderecoDestinatario
(entrada.nextLine());
        System.out.print ("Cidade: ");
        vetor[numeroDeElementos].setCidadeDestinatario
(entrada.nextLine());
        System.out.print ("CEP: ");
        vetor[numeroDeElementos].setCEPDestinatario
(entrada.nextLong());
```

```

        System.out.print ("\nPeso do pacote: ");
        vetor[numeroDeElementos].setPeso (entrada.nextDouble());
        numeroDeElementos++;
    }
    else
        System.out.println ("Nao ha espaco para mais pacotes.");
        break;
case 3:
    for (int i=0; i<numeroDeElementos; i++)
    {
        System.out.println (vetor[i]);
        System.out.printf ("Custo da postagem: %f",
vetor[i].calculaCusto());
    }
    break;
case 4:
    {
        double auxiliar=0;
        for (int i=0; i<numeroDeElementos; i++)
            auxiliar += vetor[i].calculaCusto();
        System.out.printf ("Custo total das postagens: %f\n",
auxiliar);
    }
    break;

```

```
case 5:  
    System.out.println ("\nObrigado por utilizar nosso software.");  
    break;  
default:  
    System.out.println ("\nOpcao invalida!");  
}  
}while(opcao!=5);  
}  
}
```

Linguagem de Programação Java

Herança/Polimorfismo

Pode-se especificar uma classe abstrata em Java. Para tal, utiliza-se a palavra reservada ***abstract***. Desta forma, a classe abstrata não pode ser instanciada e serve como base para classes derivadas.

Vejam os um exemplo:

```

abstract class Base {
    public void func(){
        System.out.println("Esta eh func() de Base");
    }
}
class Derivada extends Base {
    public void func(){
        System.out.println("Esta eh func() de Derivada");
    }
}
public class TestePolimorfismo2 {
    public static void main(String args[]){
        Base b;
        Derivada d = new Derivada();
        d.func();
        b = d;
        b.func();
        Base b2 = new Base(); /* ERRO! Pois Base é uma classe abstrata */
    }
}

```


Linguagem de Programação Java

Herança/Polimorfismo

Assim como podemos especificar uma classe abstrata em Java, também podemos especificar métodos abstratos em um classe abstrata. Para tal, também utiliza-se a palavra reservada ***abstract***. Um método abstrato é um método que não tem implementação na classe que o contém e que será implementado nas classes derivadas da mesma.

Vejamos um exemplo:

```
abstract class Base {  
    public abstract void func();  
}  
class Derivada extends Base {  
    public void func(){  
        System.out.println("Esta eh func() de Derivada");  
    }  
}  
public class TestePolimorfismo2 {  
    public static void main(String args[]){  
        Base b;  
        Derivada d = new Derivada();  
        d.func();  
        b = d;  
        b.func();  
        Base b2 = new Base(); /* ERRO! Pois Base é uma classe abstrata */  
    }  
}
```

Linguagem de Programação Java

Herança/Polimorfismo

Java apresenta um recurso que possibilita a definição de ações que devem ser obrigatoriamente executadas por uma classe, mas de forma particular.

Este recurso é denominado Interface. Uma interface contém assinaturas de métodos que devem ser implementados dentro de uma classe.

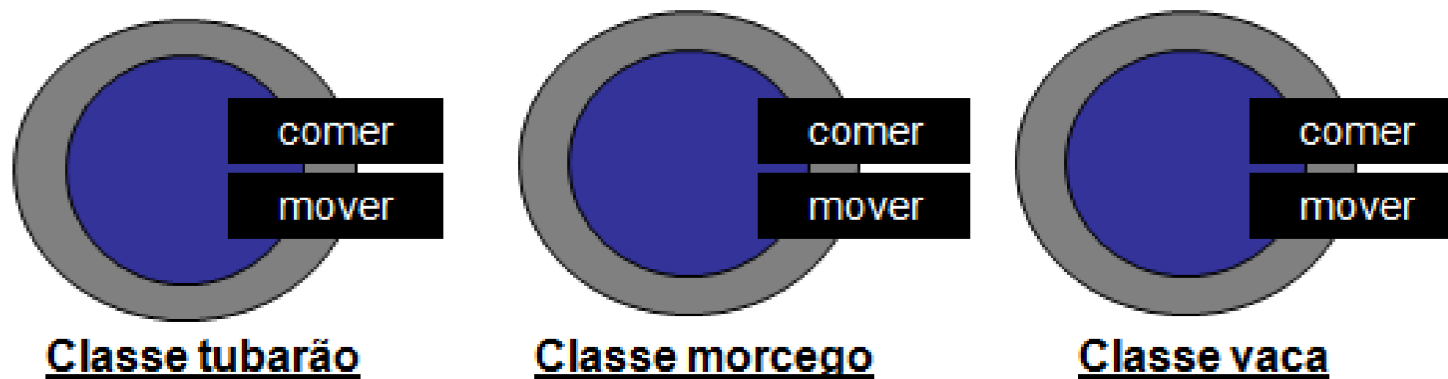
Um aluno atento deve ter se perguntado qual a funcionalidade deste recurso.

Pense em uma situação na qual é necessário se determinar um conjunto de comportamentos desejáveis por um grupo de classes, porém executados de forma particular por cada classe.

Linguagem de Programação Java

Herança/Polimorfismo

Neste caso, podemos nos lembrar do exemplo mencionado anteriormente,



Vejam os um exemplo:

```

abstract class Animal {
    private float peso;
    public Animal(float peso) {
        setPeso((float)0.1);
    }
    public void setPeso(float peso) {
        this.peso = peso;
    }
    public float getPeso() {
        return (peso);
    }
}
interface ComportamentoAnimal {
    public void comer(float entrada);
}
class Morcego extends Animal implements ComportamentoAnimal {
    public Morcego(float peso) {
        super(peso);
    }
    public void comer(float entrada) {
        setPeso(getPeso()+entrada*(float)0.75);
    }
}
class Vaca extends Animal implements ComportamentoAnimal {
    public Vaca(float peso) {
        super(peso);
    }
    public void comer(float entrada) {
        setPeso(getPeso()+entrada*(float)0.50);
    }
}

```

Linguagem de Programação Java

Herança/Polimorfismo

É interessante frisar que uma classes abstratas fornece uma base para codificação de uma classe por completo, diferentemente das interfaces que definem apenas assinaturas de métodos.

A combinação adequada de heranças e interfaces proporciona uma forma alternativa para resolver problemas que necessitam de heranças múltiplas disponíveis em C++.

Linguagem de Programação Java

Herança/Polimorfismo

Um aluno atento deve ter se perguntado:

Existe, em Java, uma maneira de determinar em tempo de execução para qual das classes derivadas um ponteiro para a classe base está apontando?

A resposta é sim.

Assim como na linguagem C++, Java disponibiliza para esta finalidade um comando, no caso o ***instanceof***.

O exemplo a seguir o utiliza.

```
abstract class Base {
    public void func(){
        System.out.println("Esta eh func() de Base");
    }
}
class Derivada1 extends Base {
    public void func(){
        System.out.println("Esta eh func() de Derivada1");
    }
}
class Derivada2 extends Base{
    public int teste;
    public void func(){
        System.out.println("Esta eh func() de Derivada2");
    }
}
public class TestePolimorfismo3 {
    public static void main(String args[]){
        Base b;
        Derivada1 d1 = new Derivada1();
        Derivada2 d2 = new Derivada2();
    }
}
```



```
b = d1;
if (b instanceof Derivada1)
{
    b.func();
    System.out.println("Entrou no primeiro if");
}
b = d2;
if (b instanceof Derivada1)
{
    b.func();
    System.out.println("Entrou no segundo if");
}
}
```

Linguagem de Programação Java

Exercício:

Utilizando este conceito implemente uma nova opção no menu do exercício do slide 533 a qual possibilita determinar o custo total apenas dos pacotes que serão entregues à noite.

//conteúdo do arquivo ManipulaPacotes.java

```
...
    System.out.println ("5 - Imprimir o custo total
apenas dos pacotes que serao entregues a noite;");
...
    case 5:
        {
            double auxiliar=0;
            for (int i=0; i<numeroDeElementos; i++)
                if (vetor[i] instanceof PacoteNoite)
                    auxiliar += vetor[i].calculaCusto();
            System.out.printf ("Custo total das %s %f\n",
                "postagens para entrega a noite: ", auxiliar);
        }
        break;
...
    }while(opcao!=6);
}
}
```

Linguagem de Programação Java

Classe Object

Um detalhe, porém muito relevante, é que todas as classe de Java são subclasses da classe denominada **Object**.

Ou seja, quando declaramos uma classe em Java e não especificamos que esta é derivada de nenhuma outra, mesmo assim implicitamente esta classe é uma subclasse da classe **Object**.

A classe **Object** possui onze métodos e já estudamos alguns destes métodos que são herdados pelas subclasses da classe **Object**, ou seja, pelas demais classe declaradas em Java.

Linguagem de Programação Java

Classe Object

Dentre os métodos da classe *Object* já estudamos:

➤ *toString* -> retorna uma representação String de um objeto;

➤ *finalize* -> é utilizado pelo coletor de lixo para realizar a limpeza de termino de um objeto antes da memória ocupada por este ser reinvidicada.

Estudaremos mais alguns métodos desta classe, para isto vamos analisaremos o seguinte código:

```
class Teste { // implicitamente tem-se "class Teste extends Object"
```

```
    private int v1;
```

```
    public void setV1(int novoV1) {
```

```
        v1 = novoV1;
```

```
    }
```

```
    public int getV1() {
```

```
        return v1;
```

```
    }
```

```
}
```

```
public class TesteObject {
```

```
    public static void main(String args[]) {
```

```
        Teste t1 = new Teste();
```

```
        t1.setV1(7);
```

```
        System.out.printf("\n%d\n", t1.getV1());
```

```
        Teste t2 = t1;
```

```
        if (t1 == t2)
```

```
            System.out.println("As referencias contidas em t1 e t2 sao iguais.");
```

```
            t2 = t1.clone(); /*Este método é protected e, além disto, problemas podem ocorrer quando os objetos clonados possuírem variáveis com referências, para tal devemos sobrescrevê-lo. */
```

```
    if (t1 == t2)
        System.out.println("As referencias contidas em t1 e t2 sao iguais.");
    /* Para comparar objetos devemos sobrescrever o método equals herdado da classe Object */
    if (t1.equals(t2))
        System.out.println("Os objetos t1 e t2 possuem o mesmo estado.");
    }
}
```

```

class Teste {
    private int v1;
    public void setV1(int novoV1) {
        v1 = novoV1;
    }
    public int getV1() {
        return v1;
    }
    public boolean equals (Teste entrada) { //sobrescrevendo equals
        return (getV1()==entrada.getV1());
        //return (v1==entrada.v1);
    }
    public Teste clone () { //sobrepondo clone
        Teste aux = new Teste();
        aux.setV1(getV1());
        return aux;
    }
}

```


Fim