

```

package edu.univasf.poo;
import edu.univasf.poo.Pacote;
public class PacoteDoisDias extends Pacote
{
    private static double taxaExtra = 5.5;
    public PacoteDoisDias(double novoPeso, double novoCustoPorQuilo,
double novaTaxaExtra)
    {
        super(novoPeso, novoCustoPorQuilo);
        setTaxaExtra((novaTaxaExtra>0)?novaTaxaExtra:getTaxaExtra());
    }
    public static void setTaxaExtra(double novaTaxaExtra)
    {
        taxaExtra = novaTaxaExtra;
    }
    public static double getTaxaExtra()
    {
        return taxaExtra;
    }
    public double calculaCusto()
    {
        return super.calculaCusto()+getTaxaExtra();
    }
}

```

```

package edu.univasf.poo;
import edu.univasf.poo.Pacote;
public class PacoteNoite extends Pacote {
    private static double taxaExtraPorQuilo = 1.2;
    public PacoteNoite(double novoPeso, double novoCustoPorQuilo,
double novaTaxaExtraPorQuilo)
    {
        super(novoPeso, novoCustoPorQuilo);
        setTaxaExtraPorQuilo((novaTaxaExtraPorQuilo>0)?
novaTaxaExtraPorQuilo:getTaxaExtraPorQuilo());
    }
    public static void setTaxaExtraPorQuilo(double novaTaxaExtraPorQuilo)
    {
        taxaExtraPorQuilo = novaTaxaExtraPorQuilo;
    }
    public static double getTaxaExtraPorQuilo() {
        return taxaExtraPorQuilo;
    }
    public double calculaCusto() {
        return super.calculaCusto()+getPeso()*getTaxaExtraPorQuilo();
    }
}

```

```
import edu.univasf.poo.Pacote;
import edu.univasf.poo.PacoteDoisDias;
import edu.univasf.poo.PacoteNoite;
public class TestaPacotes
{
    public static void main(String args[])
    {
        Pacote p1 = new Pacote(1, -3);
        PacoteDoisDias p2 = new PacoteDoisDias(1, 0, 0);
        PacoteNoite p3 = new PacoteNoite(1, 0, 0);
        p1.setPeso(5);
        p2.setPeso(5);
        p3.setPeso(5);
        System.out.printf ("Custo do pacote normal: %f\n", p1.calculaCusto());
        System.out.printf      ("Custo do pacote dois dias: %f\n",
p2.calculaCusto());
        System.out.printf ("Custo do pacote noite: %f\n", p3.calculaCusto());
    }
}
```

Linguagem de Programação Java

POLIMORFISMO

Assim como na linguagem C++, a linguagem Java também permite tratar objetos das classes derivadas de forma genérica. Em outras palavras, Java nos permite “programar no geral” em vez de “programar no específico” através a exploração do princípio do polimorfismo.

Já nos utilizamos da sobreposição de um método, ou seja, efetuamos a redefinição de um método definido em uma superclasse em uma de suas subclasses.

Vamos agora analisar este conceito considerando que Java também viabilizará o polimorfismo em tempo de execução, pois possibilita referenciar um objeto de uma classe derivada utilizando para tal uma referência para um objeto da classe base.

Linguagem de Programação Java

```
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");  
    }  
}
```

533 //o arquivo continua...

Linguagem de Programação Java

```
//continuação do arquivo
public class TestePolimorfismo {
    public static void main(String args[]){
        Base b = new Base();
        Derivada1 d1 = new Derivada1();
        Derivada2 d2 = new Derivada2();
        b.func(); //apresentará na tela “Esta eh func() de Base”
        b = d1;
        b.func(); //apresentará na tela “Esta eh func() de Derivada1”
        d2.teste = 7;
        b = d2;
        b.func(); //apresentará na tela “Esta eh func() de Derivada2”
        b.teste = 7; /* ERRO! */
    }
}
```

Linguagem de Programação Java

Exercício: Com base no que vimos, use a hierarquia de herança Pacote criada no exercício do slide 528 para criar um programa que exibe as informações de endereço e calcula os custos de entrega de vários pacotes. O programa deve conter um vetor de referências a objetos da classe Pacote e utilizá-las para referenciar objetos das classes PacoteDoisDias e PacoteNoite; manipulando-o através das operações disponibilizadas pelo seguinte menu.

Digite:

- 1 – Inserir um pacote para entrega em dois dias;**
- 2 – Inserir um pacote para entrega a noite;**
- 3 – Imprimir os endereços para postagem e custos da postagem;**
- 4 – Imprimir o custo total das postagens.**

Para simplificar as manipulações considere que no máximo o vetor possuirá 100 referências para pacotes.

```

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 exemplos:

```

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 535 a qual possibilita determinar o custo total apenas dos pacotes que serão entregues à noite.