

Programação Orientada a Objetos

Dadas as classes Employee, Boss, CommissionWorker, PieceWorker, HourlyWorker e Test:

Classe Employee:

```
4  public abstract class Employee {
5      private String firstName;
6      private String lastName;
7
8      // constructor
9      public Employee( String first, String last )
10     {
11         firstName = first;
12         lastName = last;
13     }
14
15     // get first name
16     public String getFirstName()
17     {
18         return firstName;
19     }
20
21     // get last name
22     public String getLastName()
23     {
24         return lastName;
25     }
26
27     public String toString()
28     {
29         return firstName + ' ' + lastName;
30     }
31
32     // Abstract method that must be implemented for each
33     // derived class of Employee from which objects
34     // are instantiated.
35     public abstract double earnings();
36
37 } // end class Employee
```

Programação Orientada a Objetos

Classe Boss:

```
4 public final class Boss extends Employee {
5     private double weeklySalary;
6
7     // constructor for class Boss
8     public Boss( String first, String last, double salary )
9     {
10         super( first, last ); // call superclass constructor
11         setWeeklySalary( salary );
12     }
13
14     // set Boss's salary
15     public void setWeeklySalary( double salary )
16     {
17         weeklySalary = ( salary > 0 ? salary : 0 );
18     }
19
20     // get Boss's pay
21     public double earnings()
22     {
23         return weeklySalary;
24     }
25
26     // get String representation of Boss's name
27     public String toString()
28     {
29         return "Boss: " + super.toString();
30     }
31
32 } // end class Boss
33
34 /*****
35  * (C) Copyright 2002 by Deitel & Associates, Inc. and Prentice Hall.
36  * All Rights Reserved.
37  *****/
```

Programação Orientada a Objetos

Classe CommissionWorker:

```
4 public final class CommissionWorker extends Employee {
5     private double salary;        // base salary per week
6     private double commission;    // amount per item sold
7     private int quantity;         // total items sold for week
8
9     // constructor for class CommissionWorker
10    public CommissionWorker( String first, String last,
11        double salary, double commission, int quantity )
12    {
13        super( first, last ); // call superclass constructor
14        setSalary( salary );
15        setCommission( commission );
16        setQuantity( quantity );
17    }
18
19    // set CommissionWorker's weekly base salary
20    public void setSalary( double weeklySalary )
21    {
22        salary = ( weeklySalary > 0 ? weeklySalary : 0 );
23    }
24
25    // set CommissionWorker's commission
26    public void setCommission( double itemCommission )
27    {
28        commission = ( itemCommission > 0 ? itemCommission : 0 );
29    }
30
31    // set CommissionWorker's quantity sold
32    public void setQuantity( int totalSold )
33    {
34        quantity = ( totalSold > 0 ? totalSold : 0 );
35    }
36
37    // determine CommissionWorker's earnings
38    public double earnings()
39    {
40        return salary + commission * quantity;
41    }
42
43    // get String representation of CommissionWorker's name
44    public String toString()
45    {
46        return "Commission worker: " + super.toString();
47    }
48
49 } // end class CommissionWorker
50
```

Programação Orientada a Objetos

Classe PieceWorker:

```
4 public final class PieceWorker extends Employee {
5     private double wagePerPiece; // wage per piece output
6     private int quantity;        // output for week
7
8     // constructor for class PieceWorker
9     public PieceWorker( String first, String last,
10        double wage, int numberOfItems )
11     {
12         super( first, last ); // call superclass constructor
13         setWage( wage );
14         setQuantity( numberOfItems );
15     }
16
17     // set PieceWorker's wage
18     public void setWage( double wage )
19     {
20         wagePerPiece = ( wage > 0 ? wage : 0 );
21     }
22
23     // set number of items output
24     public void setQuantity( int numberOfItems )
25     {
26         quantity = ( numberOfItems > 0 ? numberOfItems : 0 );
27     }
28
29     // determine PieceWorker's earnings
30     public double earnings()
31     {
32         return quantity * wagePerPiece;
33     }
34
35     public String toString()
36     {
37         return "Piece worker: " + super.toString();
38     }
39
40 } // end class PieceWorker
41
```

Programação Orientada a Objetos

Classe HourlyWorker:

```
4 public final class HourlyWorker extends Employee {
5     private double wage; // wage per hour
6     private double hours; // hours worked for week
7
8     // constructor for class HourlyWorker
9     public HourlyWorker( String first, String last,
10         double wagePerHour, double hoursWorked )
11     {
12         super( first, last ); // call superclass constructor
13         setWage( wagePerHour );
14         setHours( hoursWorked );
15     }
16
17     // Set the wage
18     public void setWage( double wagePerHour )
19     {
20         wage = ( wagePerHour > 0 ? wagePerHour : 0 );
21     }
22
23     // Set the hours worked
24     public void setHours( double hoursWorked )
25     {
26         hours = ( hoursWorked >= 0 && hoursWorked < 168 ?
27             hoursWorked : 0 );
28     }
29
```

```
30 // Get the HourlyWorker's pay
31 public double earnings() { return wage * hours; }
32
33 public String toString()
34 {
35     return "Hourly worker: " + super.toString();
36 }
37
38 } // end class HourlyWorker
39
```

Programação Orientada a Objetos

Classe Test:

```
4 // Java core packages
5 import java.text.DecimalFormat;
6
7 // Java extension packages
8 import javax.swing.JOptionPane;
9
10 public class Test {
11
12     // test Employee hierarchy
13     public static void main( String args[] )
14     {
15         Employee employee; // superclass reference
16         String output = "";
17
18         Boss boss = new Boss( "John", "Smith", 800.0 );
19
20         CommissionWorker commissionWorker =
21             new CommissionWorker(
22                 "Sue", "Jones", 400.0, 3.0, 150 );
23
24         PieceWorker pieceWorker =
25             new PieceWorker( "Bob", "Lewis", 2.5, 200 );
26
27         HourlyWorker hourlyWorker =
28             new HourlyWorker( "Karen", "Price", 13.75, 40 );
29
30         DecimalFormat precision2 = new DecimalFormat( "0.00" );
31
32         // Employee reference to a Boss
33         employee = boss;
34
35         output += employee.toString() + " earned $" +
36             precision2.format( employee.earnings() ) + "\n" +
37             boss.toString() + " earned $" +
38             precision2.format( boss.earnings() ) + "\n";
39
40         // Employee reference to a CommissionWorker
41         employee = commissionWorker;
42
43         output += employee.toString() + " earned $" +
44             precision2.format( employee.earnings() ) + "\n" +
45             commissionWorker.toString() + " earned $" +
46             precision2.format(
47                 commissionWorker.earnings() ) + "\n";
48
49         // Employee reference to a PieceWorker
50         employee = pieceWorker;
51
52         output += employee.toString() + " earned $" +
53             precision2.format( employee.earnings() ) + "\n" +
54             pieceWorker.toString() + " earned $" +
55             precision2.format( pieceWorker.earnings() ) + "\n";
56
57         // Employee reference to an HourlyWorker
58         employee = hourlyWorker;
59
```

Programação Orientada a Objetos

```
59
60     output += employee.toString() + " earned $" +
61         precision2.format( employee.earnings() ) + "\n" +
62         hourlyWorker.toString() + " earned $" +
63         precision2.format( hourlyWorker.earnings() ) + "\n";
64
65     JOptionPane.showMessageDialog( null, output,
66         "Demonstrating Polymorphism",
67         JOptionPane.INFORMATION_MESSAGE );
68
69     System.exit( 0 );
70 }
71
72 } // end class Test
73
```

Exercício 1: Fazer o diagrama de classes das classes que compõem a aplicação.

Exercício 2: Identificar o método abstrato e verificar como o mesmo está sendo utilizado na aplicação.

Exercício 3: Quais as características funcionais de uma classe do tipo “final”? Quais as consequências do uso das mesmas na aplicação?.

Exercício 4: Editar, compilar e executar as classes da aplicação, descrevendo como será a saída da classe executável.

Programação Orientada a Objetos

The screenshot shows an IDE with the following components:

- Top Bar:** Tabs for 'Página Inicial', 'Acesso_Numeros.java', 'Verifica_Encapsulamento_1.java', and 'Verifica_Encapsulamento_2.java'. Below the tabs is a toolbar with icons for file operations and navigation.
- Code Editor:** Displays the source code for 'encapsulamento_1.Teste_Condicional'. The code includes a class definition with a `main` method and two static methods, `valor_x` and `valor_h`, which use ternary operators for conditional logic. Line 25, `public static int valor_h(int x) {`, is highlighted. A lightbulb icon is present next to line 25.
- Annotation:** A text box on the right side of the code editor contains the text: "se x não satisfaz a condicional retorna o valor → : valor".
- Bottom Bar:** A breadcrumb navigation shows the path: `encapsulamento_1.Teste_Condicional` > `valor_h` > `v`.
- Output Console:** Titled 'Saída - JavaApplication1 (run)', it shows the execution results: `run:`, `x == -2`, `v == 9`, and a success message: `CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)`.

```
10  * @author mmario
11  */
12  public class Teste_Condicional {
13
14      int x = 0; int h = 0;
15      public static void main(String args[]) {
16          System.out.println("x =" + valor_x (4));
17          System.out.println("v =" + valor_h (4));
18      }
19
20
21      public static int valor_x(int x) {
22          int y = (x > 5 ? x : -2);
23          return y;}
24
25      public static int valor_h(int x) {
26          int v = ( x >= 10 && x < 168 ? x : 9 );
27          return v;}
28  }
```

se x não satisfaz a condicional
retorna o valor → : valor

encapsulamento_1.Teste_Condicional > valor_h > v

Saída - JavaApplication1 (run) X

run:
x == -2
v == 9
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)