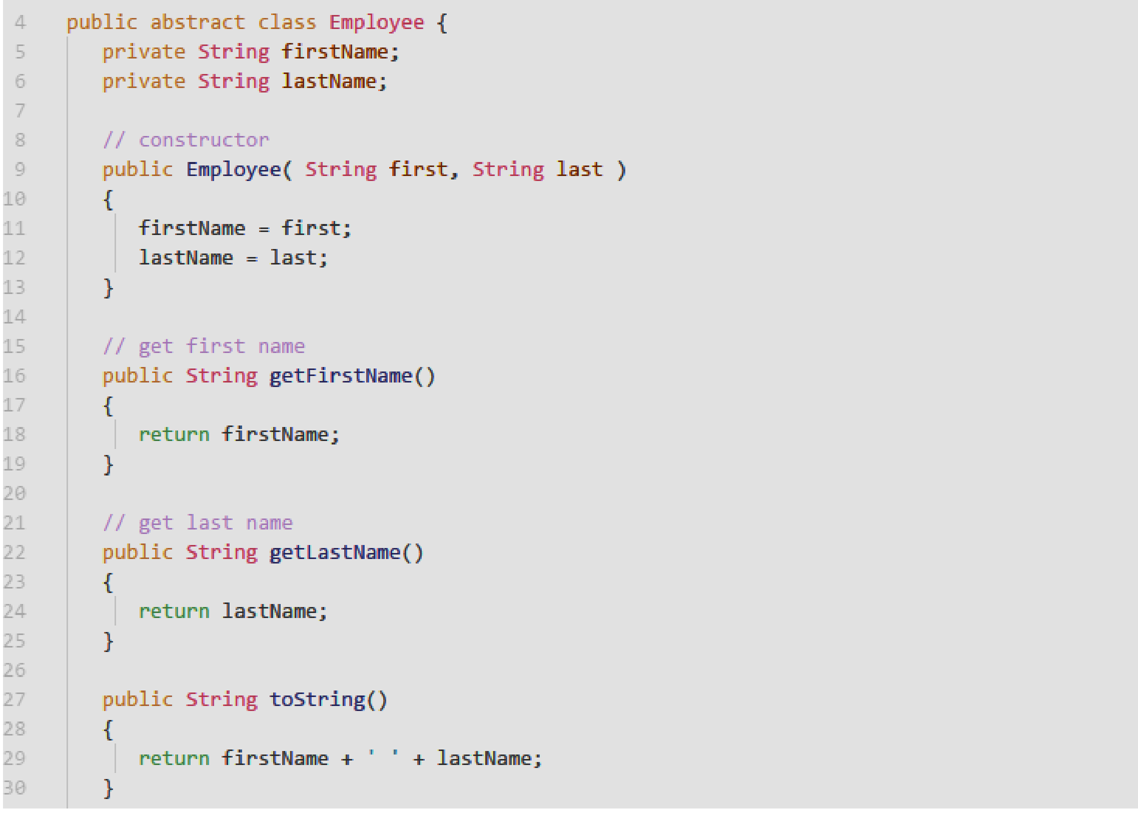
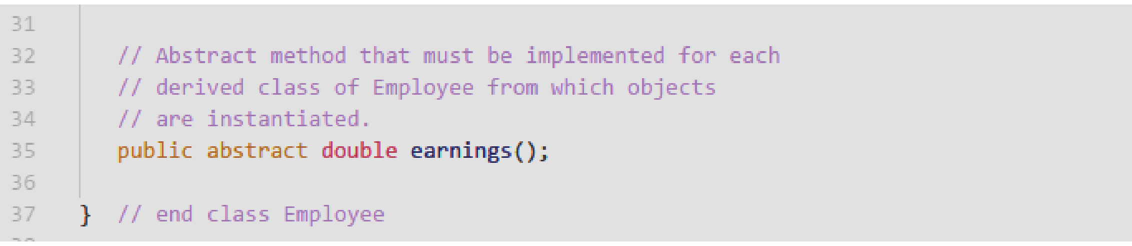
Programação Orientada a Objetos

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

Classe Employee:

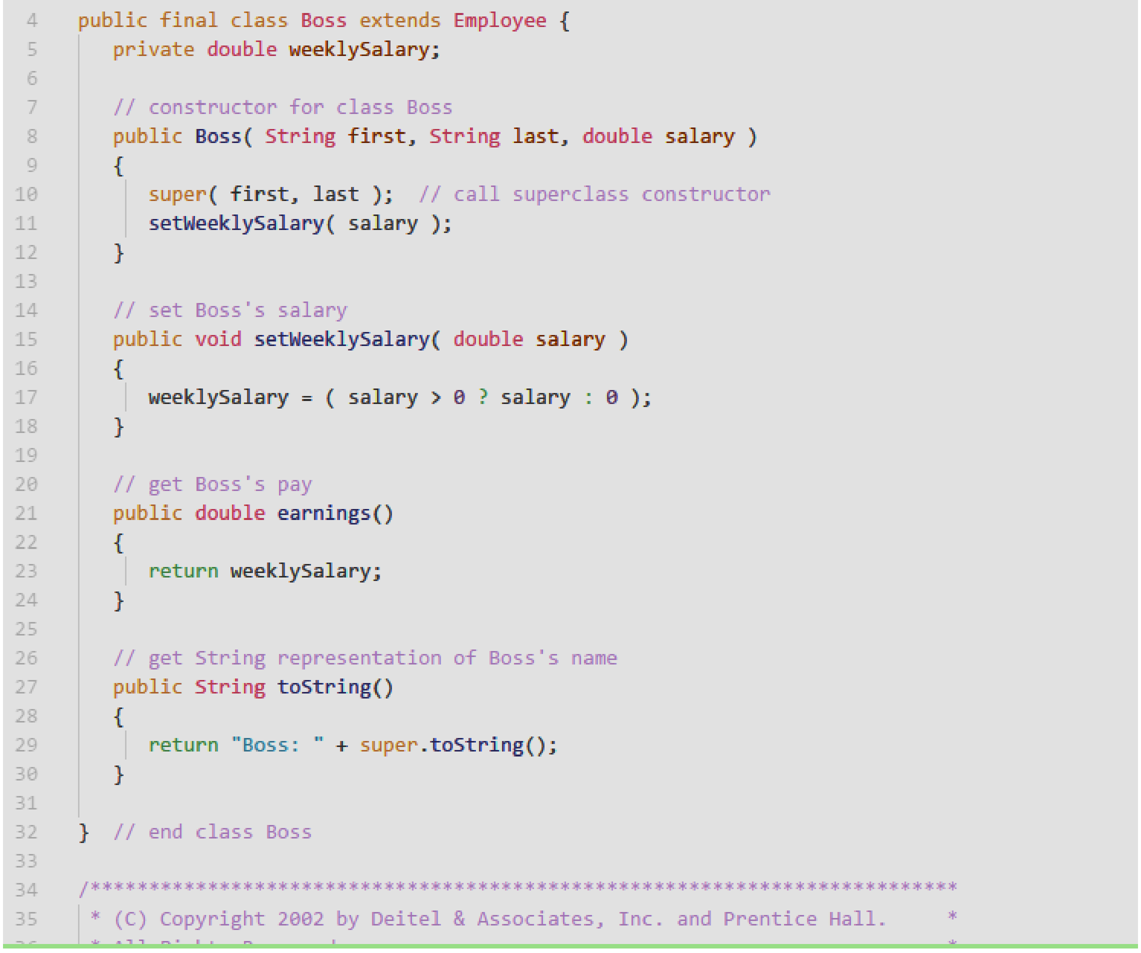


```java
public abstract class Employee {
   private String firstName;
   private String lastName;

   // constructor
   public Employee( String first, String last )
   {
      firstName = first;
      lastName = last;
   }

   // get first name
   public String getFirstName()
   {
      return firstName;
   }

   // get last name
   public String getLastName()
   {
      return lastName;
   }

   public String toString()
   {
      return firstName + ' ' + lastName;
   }
```



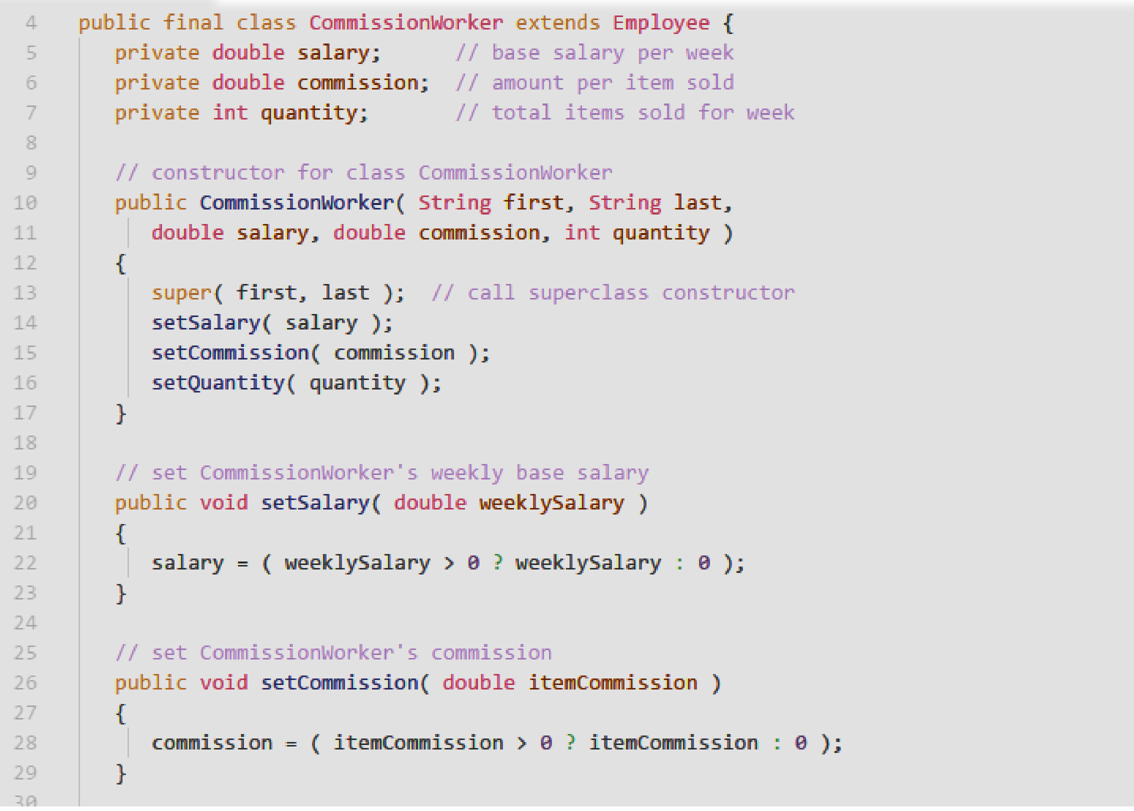
```java

   // Abstract method that must be implemented for each
   // derived class of Employee from which objects
   // are instantiated.
   public abstract double earnings();

}  // end class Employee
```

Classe Boss:

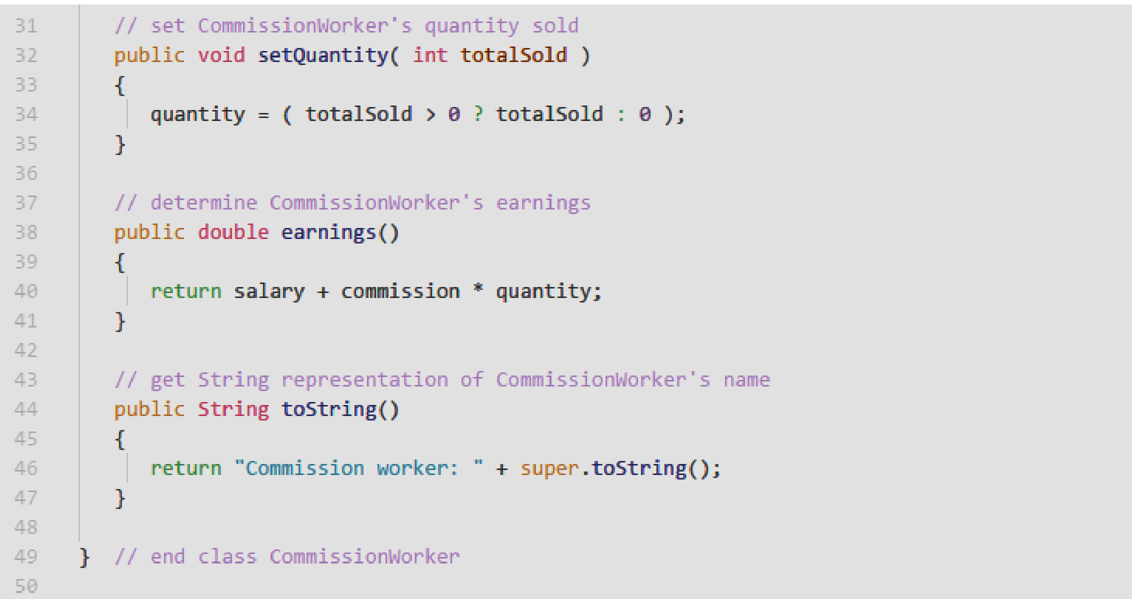


```java
public final class Boss extends Employee {
   private double weeklySalary;

   // constructor for class Boss
   public Boss( String first, String last, double salary )
   {
      super( first, last );  // call superclass constructor
      setWeeklySalary( salary );
   }

   // set Boss's salary
   public void setWeeklySalary( double salary )
   {
      weeklySalary = ( salary > 0 ? salary : 0 );
   }

   // get Boss's pay
   public double earnings()
   {
      return weeklySalary;
   }

   // get String representation of Boss's name
   public String toString()
   {
      return "Boss: " + super.toString();
   }

}  // end class Boss

/**************************************************************************
 * (C) Copyright 2002 by Deitel & Associates, Inc. and Prentice Hall.     *
```

Classe CommissionWorker:

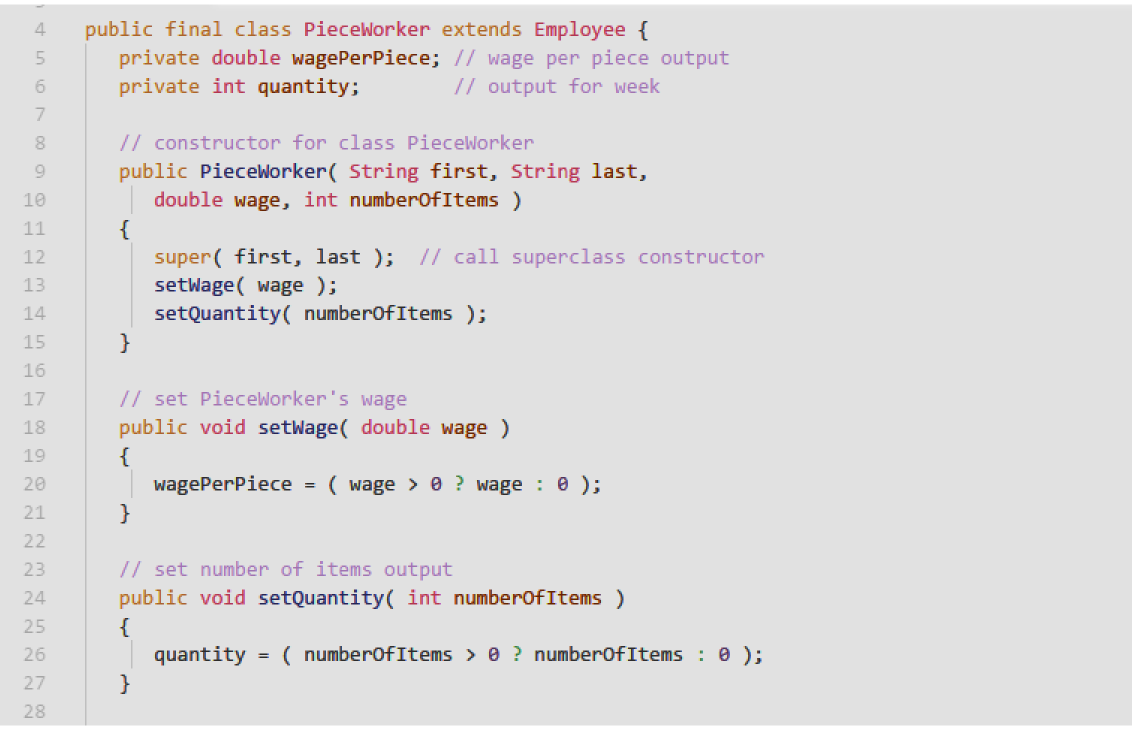


```java
public final class CommissionWorker extends Employee {
   private double salary;      // base salary per week
   private double commission;  // amount per item sold
   private int quantity;       // total items sold for week

   // constructor for class CommissionWorker
   public CommissionWorker( String first, String last,
      double salary, double commission, int quantity )
   {
      super( first, last );  // call superclass constructor
      setSalary( salary );
      setCommission( commission );
      setQuantity( quantity );
   }

   // set CommissionWorker's weekly base salary
   public void setSalary( double weeklySalary )
   {
      salary = ( weeklySalary > 0 ? weeklySalary : 0 );
   }

   // set CommissionWorker's commission
   public void setCommission( double itemCommission )
   {
      commission = ( itemCommission > 0 ? itemCommission : 0 );
   }

```

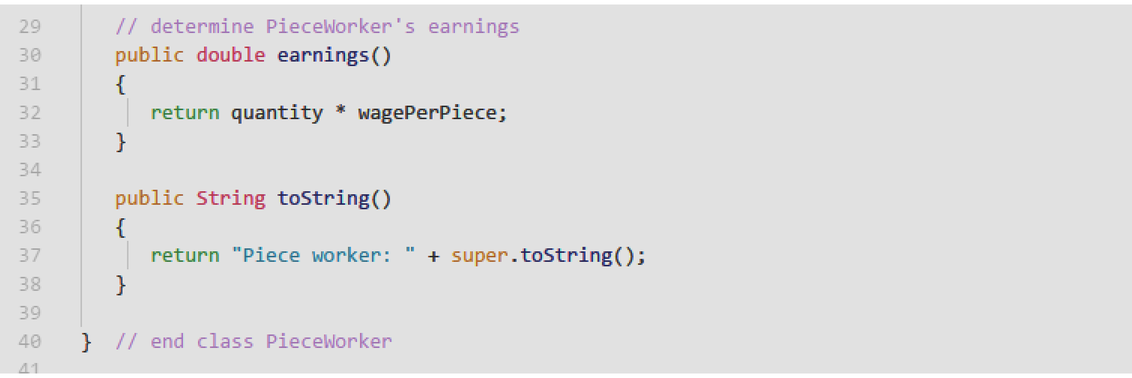


```java
   // set CommissionWorker's quantity sold
   public void setQuantity( int totalSold )
   {
      quantity = ( totalSold > 0 ? totalSold : 0 );
   }

   // determine CommissionWorker's earnings
   public double earnings()
   {
      return salary + commission * quantity;
   }

   // get String representation of CommissionWorker's name
   public String toString()
   {
      return "Commission worker: " + super.toString();
   }

}  // end class CommissionWorker

```

Classe PieceWorker:

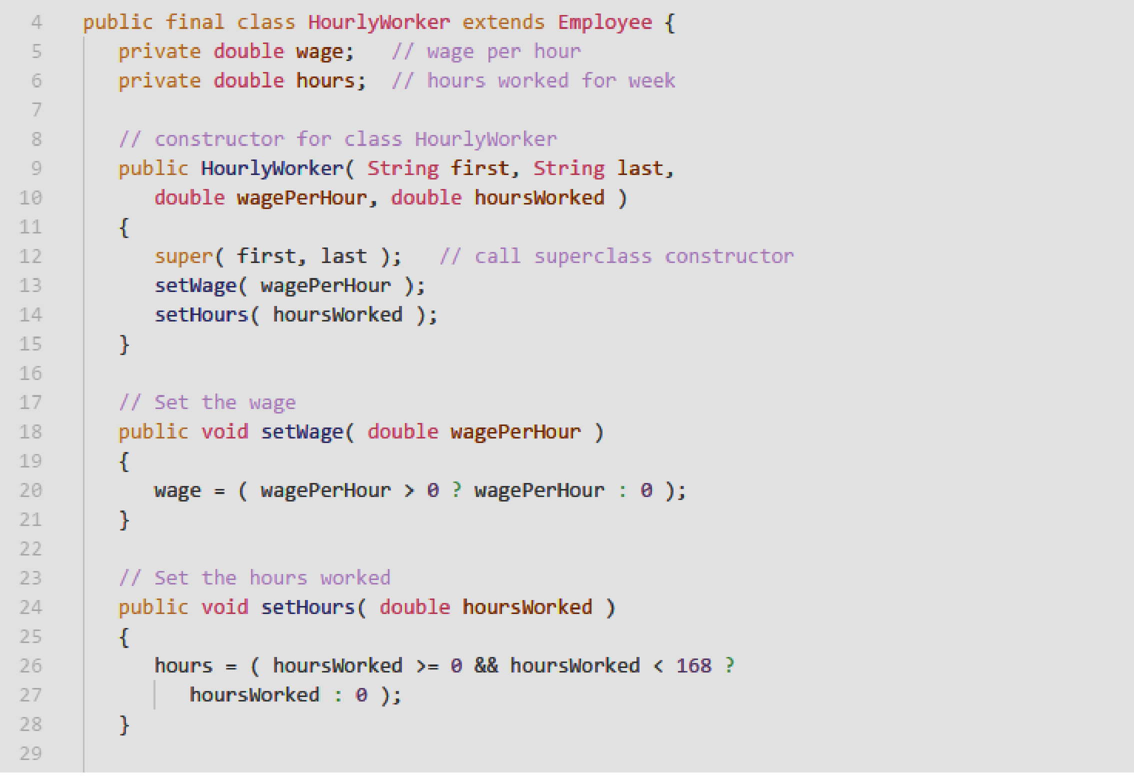


```java
public final class PieceWorker extends Employee {
    private double wagePerPiece; // wage per piece output
    private int quantity;        // output for week

    // constructor for class PieceWorker
    public PieceWorker( String first, String last,
        double wage, int numberOfItems )
    {
        super( first, last );  // call superclass constructor
        setWage( wage );
        setQuantity( numberOfItems );
    }

    // set PieceWorker's wage
    public void setWage( double wage )
    {
        wagePerPiece = ( wage > 0 ? wage : 0 );
    }

    // set number of items output
    public void setQuantity( int numberOfItems )
    {
        quantity = ( numberOfItems > 0 ? numberOfItems : 0 );
    }

```

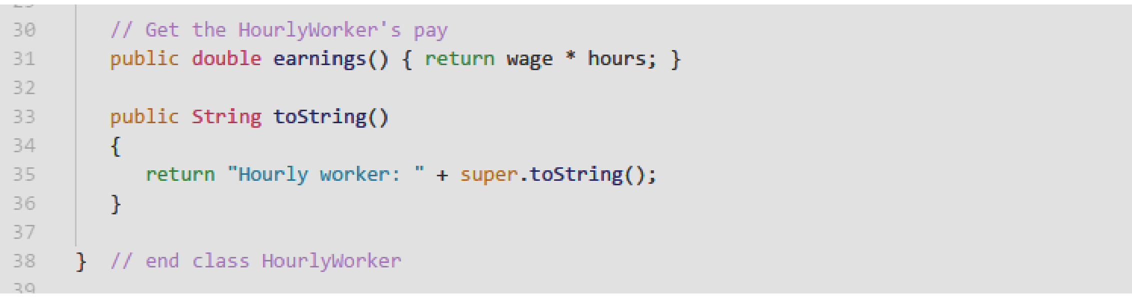


```java
    // determine PieceWorker's earnings
    public double earnings()
    {
        return quantity * wagePerPiece;
    }

    public String toString()
    {
        return "Piece worker: " + super.toString();
    }

}  // end class PieceWorker
```

Classe HourlyWorker:

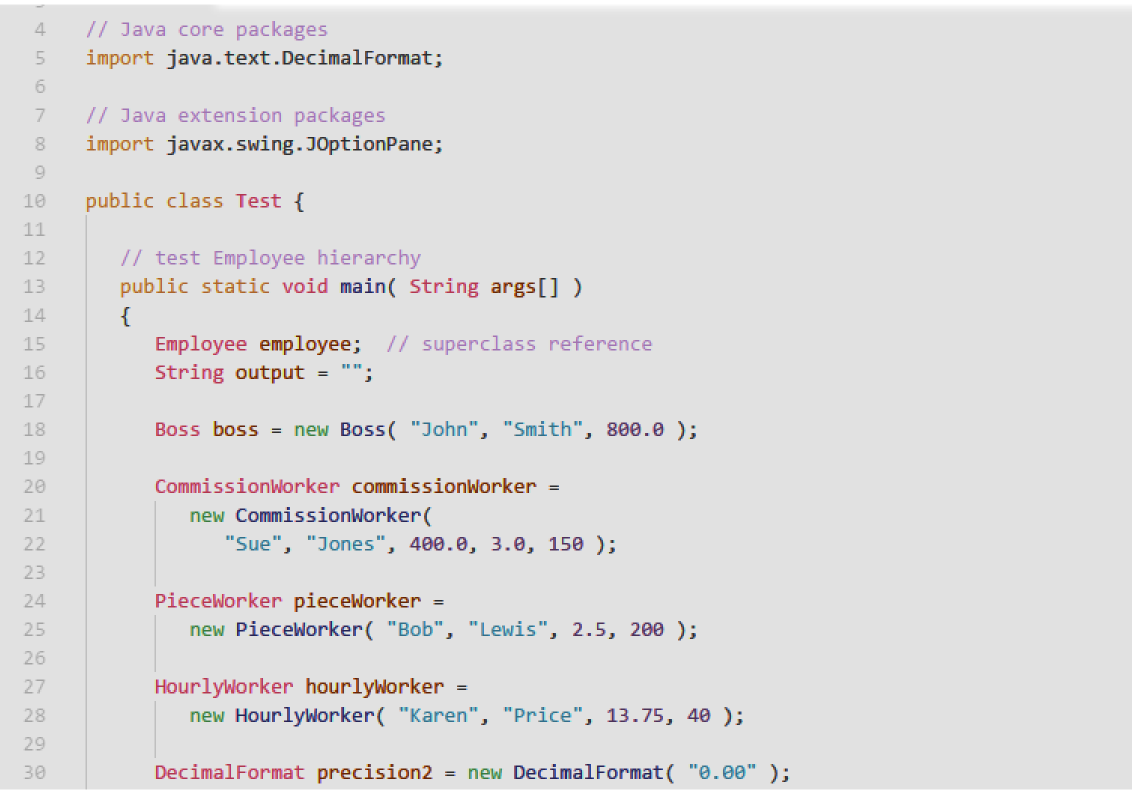


```java
public final class HourlyWorker extends Employee {
   private double wage;   // wage per hour
   private double hours;  // hours worked for week

   // constructor for class HourlyWorker
   public HourlyWorker( String first, String last,
      double wagePerHour, double hoursWorked )
   {
      super( first, last );   // call superclass constructor
      setWage( wagePerHour );
      setHours( hoursWorked );
   }

   // Set the wage
   public void setWage( double wagePerHour )
   {
      wage = ( wagePerHour > 0 ? wagePerHour : 0 );
   }

   // Set the hours worked
   public void setHours( double hoursWorked )
   {
      hours = ( hoursWorked >= 0 && hoursWorked < 168 ?
         hoursWorked : 0 );
   }

```

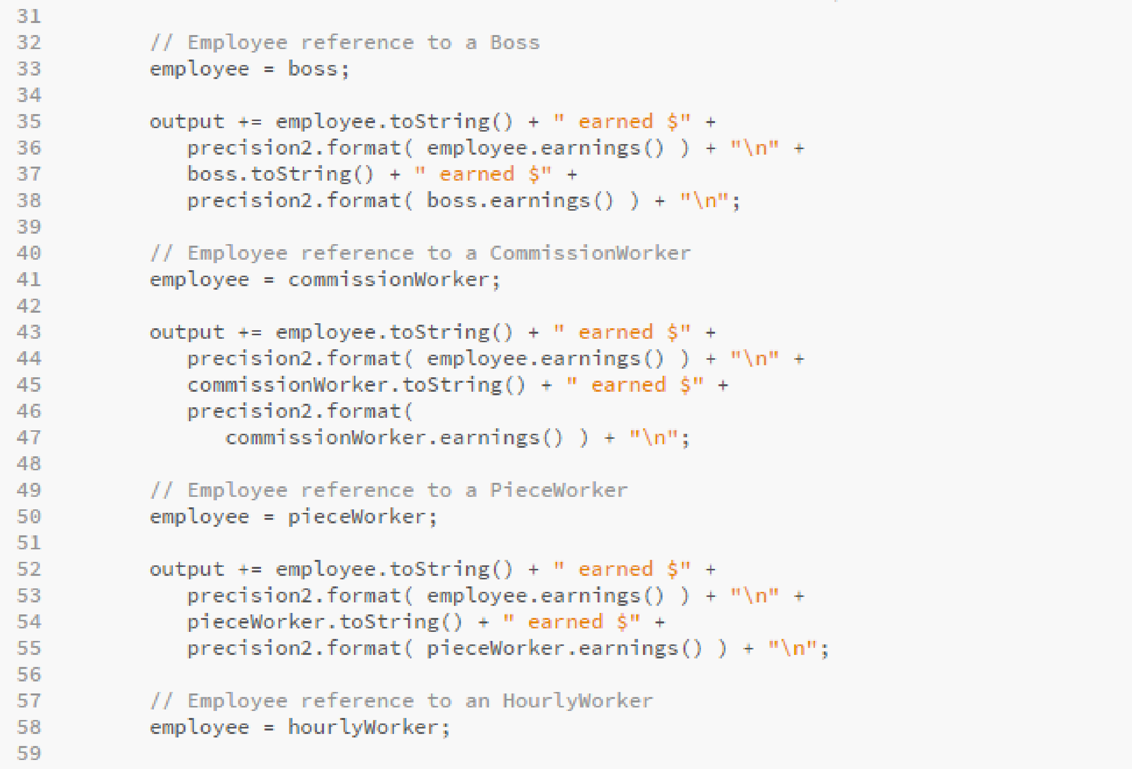


```java
   // Get the HourlyWorker's pay
   public double earnings() { return wage * hours; }

   public String toString()
   {
      return "Hourly worker: " + super.toString();
   }

}  // end class HourlyWorker
```

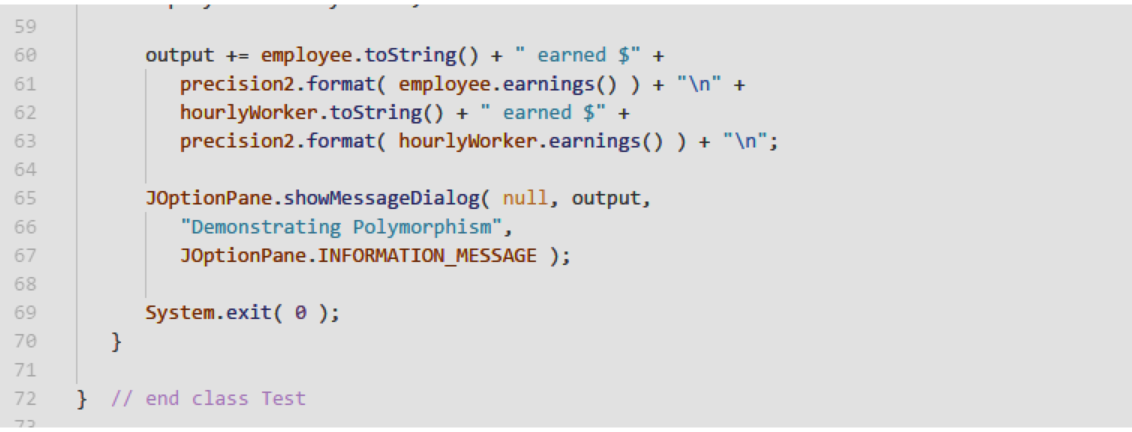
Classe Test:



```java
// Java core packages
import java.text.DecimalFormat;

// Java extension packages
import javax.swing.JOptionPane;

public class Test {

   // test Employee hierarchy
   public static void main( String args[] )
   {
      Employee employee;  // superclass reference
      String output = "";

      Boss boss = new Boss( "John", "Smith", 800.0 );

      CommissionWorker commissionWorker =
         new CommissionWorker(
            "Sue", "Jones", 400.0, 3.0, 150 );

      PieceWorker pieceWorker =
         new PieceWorker( "Bob", "Lewis", 2.5, 200 );

      HourlyWorker hourlyWorker =
         new HourlyWorker( "Karen", "Price", 13.75, 40 );

      DecimalFormat precision2 = new DecimalFormat( "0.00" );
```



```java

      // Employee reference to a Boss
      employee = boss;

      output += employee.toString() + " earned $" +
         precision2.format( employee.earnings() ) + "\n" +
         boss.toString() + " earned $" +
         precision2.format( boss.earnings() ) + "\n";

      // Employee reference to a CommissionWorker
      employee = commissionWorker;

      output += employee.toString() + " earned $" +
         precision2.format( employee.earnings() ) + "\n" +
         commissionWorker.toString() + " earned $" +
         precision2.format(
            commissionWorker.earnings() ) + "\n";

      // Employee reference to a PieceWorker
      employee = pieceWorker;

      output += employee.toString() + " earned $" +
         precision2.format( employee.earnings() ) + "\n" +
         pieceWorker.toString() + " earned $" +
         precision2.format( pieceWorker.earnings() ) + "\n";

      // Employee reference to an HourlyWorker
      employee = hourlyWorker;

```



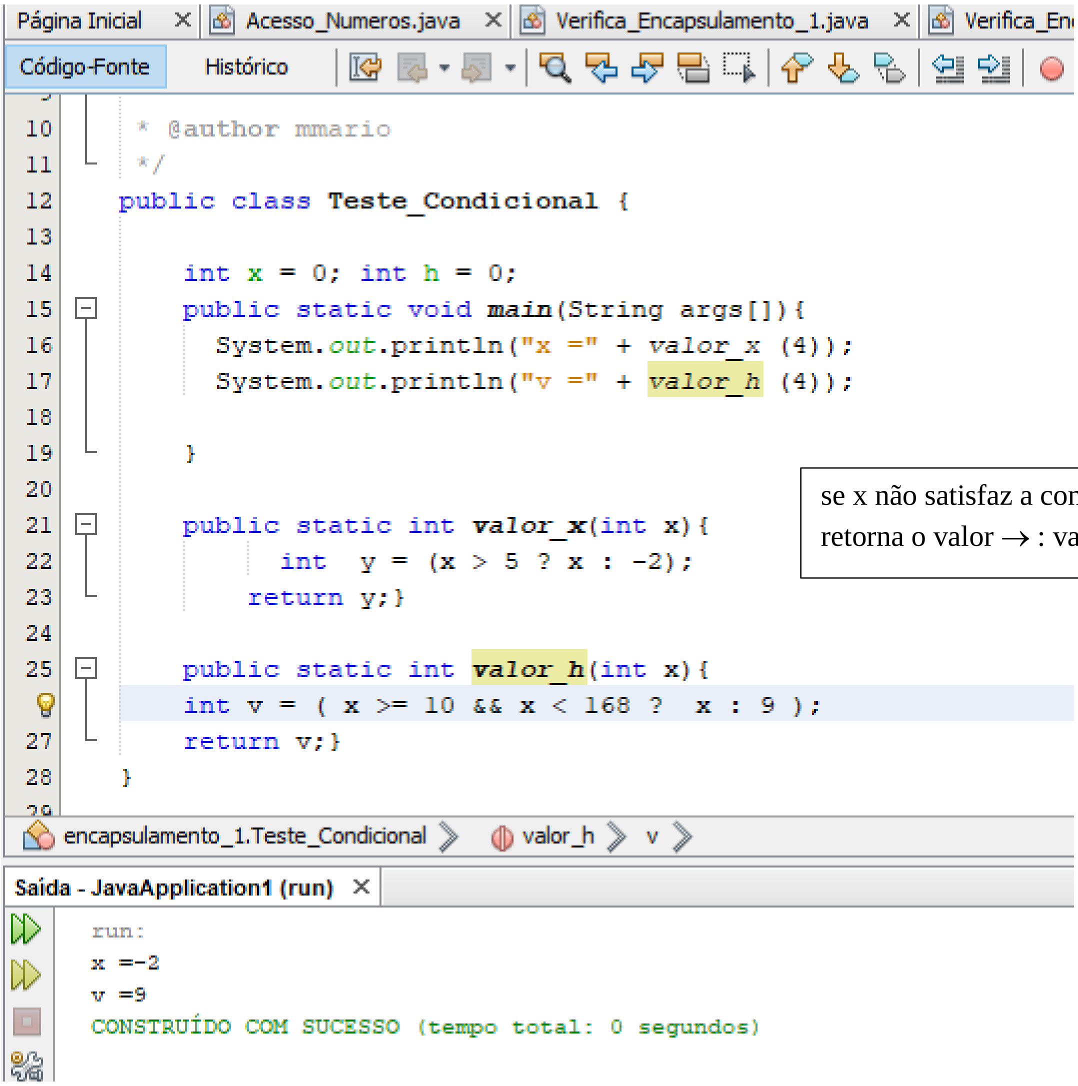
```java

      output += employee.toString() + " earned $" +
         precision2.format( employee.earnings() ) + "\n" +
         hourlyWorker.toString() + " earned $" +
         precision2.format( hourlyWorker.earnings() ) + "\n";

      JOptionPane.showMessageDialog( null, output,
         "Demonstrating Polymorphism",
         JOptionPane.INFORMATION_MESSAGE );

      System.exit( 0 );
   }

}  // end class Test
```

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.


Página Inicial
Acesso_Numeros.java
Verifica_Encapsulamento_1.java
Verifica_En
Código-Fonte
Histórico
encapsulamento_1.Teste_Condicional
valor_h
v
Saída - JavaApplication1 (run)

```java
 * @author mmario
 */
public class Teste_Condicional {

    int x = 0; int h = 0;
    public static void main(String args[]){
      System.out.println("x =" + valor_x (4));
      System.out.println("v =" + valor_h (4));

    }

    public static int valor_x(int x){
          int  y = (x > 5 ? x : -2);
        return y;}

    public static int valor_h(int x){
    int v = ( x >= 10 && x < 168 ?  x : 9 );
    return v;}
}
```

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

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