Abstract class

2301260 Programming Techniques

ผศ. ศศิภา พันธุวดีธร ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

สรุป Polymorphism

- การที่ คำสั่งเรียกใช้เมท็อด ถูกเรียกใช้ผ่าน object reference ชนิดหนึ่ง (ระดับ superclass) แล้วตอนประมวลผลจริง คอมพิวเตอร์สามารถไปสั่งรันคำสั่งในเมท็อดของก้อนอ็อบเจกต์ต่างชนิดที่ถูกอ้างถึงโดย object reference นั้นได้
- มีข้อแม้ว่า
  - ชนิดของอ็อบเจกต์ต่าง ๆ ที่จะมาทำงานร่วมกันในแบบ polymorphism จะต้องเป็นอ็อบเจกต์ในตระกูลเดียวกัน
  - superclass และ subclass มีการใช้เมท็อดชื่อเดียวกัน ซึ่งหากมีการเปลี่ยนแปลงรายละเอียดของโค้ดภายในเมท็อด จะเกิดจากการทำ override
- จุดประสงค์คือ เพื่อให้สามารถใช้โค้ดของโปรแกรมที่สร้างขึ้นมาก่อนได้ ถึงแม้ว่าต่อ ๆ มาจะมีการสร้างคลาสลูกคลาสหลานของมันขึ้นมาเพิ่ม

สรุป polymorphism

- โค้ดในโปรแกรมเดิมที่เคยเขียนและใช้กันในระดับ superclass ก็ไม่ต้องเปลี่ยน มันยังคงใช้ชนิดของตัวแปรเดิมของมันได้ แล้วใช้วิธีการ assign reference (ตามที่เราเรียนมาว่าสามารถใช้ reference ของระดับ superclass ไปชี้ก้อนอ็อบเจกต์ของระดับ subclass ได้)
- เมื่อเรียกใช้เมท็อดที่มีชื่อเดียวกันทั้งใน superclass และ subclass ผ่าน reference ระดับ superclass ระบบจะพิจารณาเองว่า ก้อนอ็อบเจกต์จริง ๆ เป็นก้อนอ็อบเจกต์ชนิด superclass หรือ subclass ระบบก็จะไปเรียกใช้เมท็อดของชนิดนั้นให้ประมวลผล เรียกว่าเป็นการทำ late binding
- เช่น otherAcct.deposit(500); → otherAcct เป็น reference ชนิด BankAccount
- เช่น otherObject.toString(); -> otherObject เป็น reference ชนิด Object

What is an Abstract Class?

- An *abstract class* is a class that is declared **`abstract`**
- Abstract classes **cannot be instantiated (cannot new object from abstract class)**
- Abstract class can contain variables and concrete (non-abstract) methods
- Constructors and static methods cannot be declared abstract
- When an abstract class is subclassed,
  the subclass usually provides implementations
  for all of the abstract methods in its parent class
- If subclass doesn't provide implementations
  then the subclass must also be declared `abstract`.

What is an Abstract Method?

- An ***abstract method*** is a method that is declared without an implementation
- It just has a method signature (without braces, and followed by a semicolon), like this:

```
abstract void moveTo(double X, double Y);
```

ตัวอย่าง

- class A มีตัวแปร a, b เป็น int มีเมท็อด getA(), getB(), getValue() แต่ getValue() ยังไม่รู้ว่าจะทำอะไร
- class B เป็นลูกของคลาส A ให้มีตัวแปร c เป็น int มีเมท็อด getC() ส่วน getValue() ก็ยังไม่รู้ว่าจะทำอะไร
- class C เป็นลูกของคลาส B ให้เขียนรายละเอียดของ getValue() โดยนำค่าของ A, B, C มาบวกกันแล้วส่งคืนออกมา
- กำหนดคลาสทดสอบดังต่อไปนี้

Example

```
public abstract class A {
    private int a=0;
    private int b=1;
    public abstract int getValue();
    public int getA() {
        return a;
    }
    public int getB() {
        return b;
    }
}
```

```
public abstract class B extends A {
    private int c;
    public int getC() {
        return c;
    }
   //still have no implementation of getValue()
}
```

```
public class ClassC extends B {
    public int getValue() {
        return getA() + getB() + getC();
    }
}
```

```
public class Main {
    public static void main (String [] args) {
        ClassC o = new ClassC();
        System.out.println (“a = ” + o.getA());
        System.out.println (“b = ” + o.getB());
        System.out.println (“c = ” + o.getC());
        System.out.println (“value = ” + o.getValue());
    }
```

รันแล้วแสดงผลลัพธ์ที่ได้

Example

- Make abstract superclass `Shape`
  - Abstract method (must be implemented by subclass)
    - `getName`
    - Default implementation does not make sense
  - Methods that may be overridden by subclass
    - `getArea`, `getVolume`
      - Default implementations return `0.0`
    - If not overridden, uses superclass default implementation
- Make subclasses `Point`, `Circle`, `Cylinder`

- Point class
  - Instance variables : x, y
  - Methods : getName(), toString()
- Circle class
  - Instance variables : r
  - Methods : getArea(), getCircumference(), getName(), toString()
- Cylinder class
  - Instance variables : h
  - Methods : getArea(), getVolume(), getName(), toString()

Shape class

```
// Shape abstract-superclass declaration.
public abstract class Shape   {
    public double getArea()    {
            return 0.0;
    }
    public double getVolume()  {
            return 0.0;
    }
    // abstract method, overridden by subclasses
    public abstract String getName();
}
```

Point class

```
public class Point extends Shape  {
        private int x;  // x part of coordinate pair
        private int y;  // y part of coordinate pair
        public Point(int xValue, int yValue)  {
                x = xValue;
                y = yValue;
        }
        // override abstract method getName
        public String getName()      {
                return "Point";
        }
        // override toString
        public String toString()      {
                return " [" + x + ", " + y + "]";
        }
}
```

Circle class

```
public class Circle extends Point  {
        private double radius;  // Circle's radius
        public Circle(int x, int y, double radiusValue)        {
                super(x, y);  // call Point constructor
                radius = radiusValue;
        }
        public double getCircumference()          {
                return 2 * Math.PI * radius;
        }
        public double getArea()  {
                return Math.PI * radius * radius;
        }
        public String getName()     {
                return "Circle";
        }
        public String toString()    {
                return super.toString() + "; Radius = " + radius;
        }
}
```

Cylinder class

```
public class Cylinder extends Circle  {
        private double height;  // Cylinder's height
        public Cylinder(int x, int y, double radius, double heightValue)  {
                super( x, y, radius );  // call Circle constructor
                height = heightValue;
        }
        public double getArea()     {
                return 2 * super.getArea() + getCircumference() * height;
        }
        public double getVolume()         {
                return super.getArea() * height;
        }
        public String getName() {
                return "Cylinder";
        }
        public String toString()   {
                return super.toString() + "; Height = " + height;
        }
}
```

```
public class ShapeTester {
    public static void main(String[] args) {
      // create Point, Circle and Cylinder objects
      Point point = new Point(7, 11);
      Circle circle = new Circle(22, 8, 3.5);
      Cylinder cylinder = new Cylinder(20, 30, 3.3, 10.75);
      String output;
      // obtain name and string representation of each object
      output = point.getName() + " : " + point + "\n\n";
      output += circle.getName() + ": " + circle + "\n" +
          "Area = " + circle.getArea() + "\n" +
          "Circumference = " + circle.getCircumference() + "\n\n";
      output += cylinder.getName() + ": " + cylinder + "\n" +
          "Area = " + cylinder.getArea() + "\n" +
          "Volume = " + cylinder.getVolume();
      System.out.println(output);
    }
}
```

Output

Point : [7, 11]

Circle: [22, 8]; Radius = 3.5
Area = 38.48451000647496
Circumference = 21.991148575128552

Cylinder: [20, 30]; Radius = 3.3; Height = 10.75
Area = 291.3198867673815
Volume = 367.7783979741231

```
public class ShapeTester {                    // show polymorphism#1
    public static void main(String[] args) {
      Point point = new Point(7, 11);
      Circle circle = new Circle(22, 8, 3.5);
      Cylinder cylinder = new Cylinder(20, 30, 3.3, 10.75);
      String output = point.getName() + ": " + point + "\n" +
                      circle.getName() + ": " + circle + "\n" +
                      cylinder.getName() + ": " + cylinder + "\n";
      output += showDetail(point);
      output += showDetail(circle);
      output += showDetail(cylinder);
      System.out.println(output);
    }
    public static String showDetail(Shape aShape)    {
            String output = "\n\n" + aShape.getName() + ": " +
            aShape.toString() + "\nArea = " +
            aShape.getArea() + "\nVolume = " +
            aShape.getVolume();
        return output;
    }
}
```

```
public class ShapeTester { // show polymorphism#2
public static void main(String[] args) {
      Shape [] sh = new Shape[3];
      sh[0] = new Point(7, 11);
      sh[1] = new Circle(22, 8, 3.5);
      sh[2] = new Cylinder(20, 30, 3.3, 10.75);
      String output = sh[0].getName() + ": " + sh[0] + "\n" +
                  sh[1].getName() + ": " + sh[1] + "\n" +
                  sh[2].getName() + ": " + sh[2] + "\n";
      for (Shape s : sh) {
          output += "\n\n" + s.getName() + ": " +
              s.toString() + "\nArea = " +
              s.getArea() + "\nVolume = " +
              s.getVolume();
      }
      System.out.println(output);
   }
}
```

abstract class as type

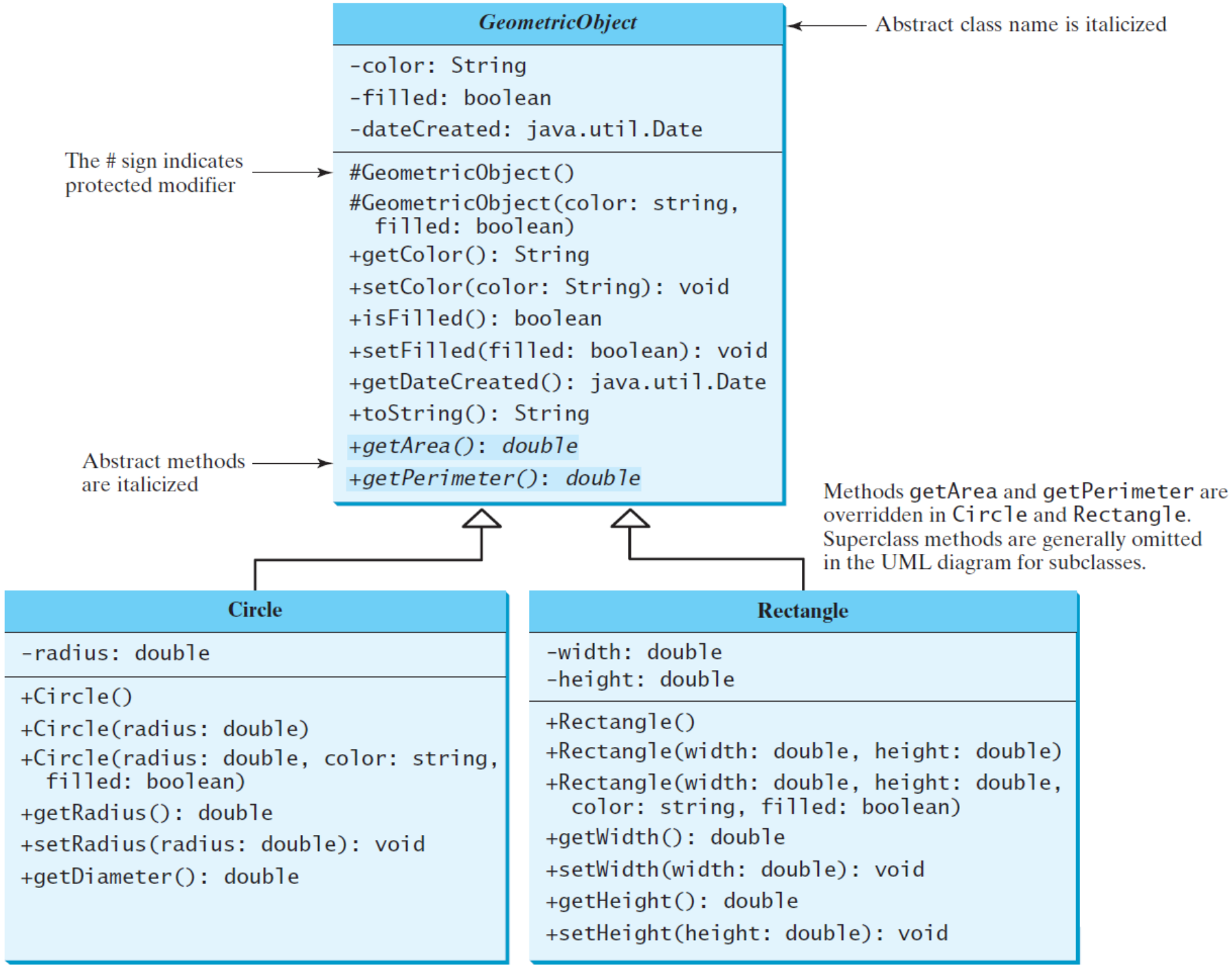
You cannot create an instance from an abstract class using the new operator, but an abstract class can be used as a data type. Therefore, the following statement, which creates an array whose elements are of

Shape type, is correct.

Shape [] sh = new Shape[3];

GraphicObject type, is correct.

GraphicObject[] graphic = new GraphicObject[10];


GeometricObject
Abstract class name is italicized
-color: String
-filled: boolean
-dateCreated: java.util.Date
The # sign indicates protected modifier
#GeometricObject()
#GeometricObject(color: string, filled: boolean)
+getColor(): String
+setColor(color: String): void
+isFilled(): boolean
+setFilled(filled: boolean): void
+getDateCreated(): java.util.Date
+toString(): String
Abstract methods are italicized
+getArea(): double
+getPerimeter(): double
Methods getArea and getPerimeter are overridden in Circle and Rectangle. Superclass methods are generally omitted in the UML diagram for subclasses.
Circle
-radius: double
+Circle()
+Circle(radius: double)
+Circle(radius: double, color: string, filled: boolean)
+getRadius(): double
+setRadius(radius: double): void
+getDiameter(): double
Rectangle
-width: double
-height: double
+Rectangle()
+Rectangle(width: double, height: double)
+Rectangle(width: double, height: double, color: string, filled: boolean)
+getWidth(): double
+setWidth(width: double): void
+getHeight(): double
+setHeight(height: double): void

สรุป Abstract class

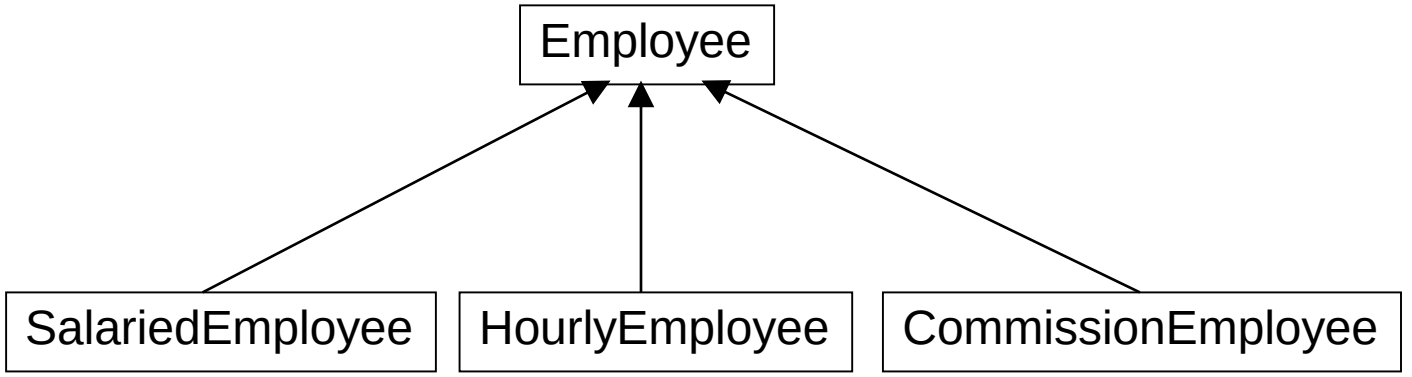
- คือคลาสที่มีเมท็อดซึ่งยังไม่ได้มีรายละเอียดของโค้ดว่าจะทำงานอย่างไร หรือคือคลาสที่ยังไม่สมบูรณ์ในตัวเอง ที่หัวคลาส จะมีคำว่า **abstract**
- เหมือนเป็นคลาสนามธรรม ที่เรารู้ว่า จะสร้างคลาสแม่แบบแบบนี้ขึ้นมา แต่เรายังไม่ได้ลงรายละเอียดให้ครบถ้วน รู้แต่ว่า หลัก ๆ ควรมีตัวแปรและชื่อเมท็อดอยู่ประมาณนี้ก่อน
- ไม่สามารถ **new object** จาก **abstract class** ไปใช้ได้ เพราะมันยังไม่ได้เป็นแม่แบบที่สมบูรณ์
- แล้วค่อยไปสร้างคลาสลูก ซึ่งเติมเต็มรายละเอียดต่าง ๆ ให้ครบถ้วน แล้วจึงสามารถ **new object** จากคลาสลูกมาใช้งานต่อไปได้

สรุป Abstract class

- ใน abstract class สามารถมีตัวแปร เมท็อดที่สมบูรณ์ และเมท็อดที่ยังประกาศเป็น abstract (constructor และ static method ประกาศเป็น abstract ไม่ได้นะ)
- ถ้า subclass ไม่ได้ override abstract method ไปเติมโค้ดให้เป็นเมท็อดที่สมบูรณ์ subclass นั้น ก็ยังต้องคงความเป็น abstract ด้วย ก็คือต้องเขียนประกาศตอนสร้าง subclass ว่า abstract ไม่งั้นคอมไพล์ไม่ผ่าน

แบบฝึกหัดในห้อง

- จงสร้างคลาสต่าง ๆ ตาม inheritance hierarchy ที่กำหนด โดยให้มี method ต่าง ๆ ตามความเหมาะสม
- ให้มี Employee เป็น abstract superclass ที่มี abstract method ชื่อ earnings
- ให้ทุกคลาส override method toString()
- ดูตารางประกอบ



	Earnings	**toString()**
Employee	Abstract	*firstName,lastName* ID:*empID*
Salaried-Employee	weeklySalary	salaried employee: *firstName, lastName* ID:*empID* weekly salary: *weeklySalary*
Hourly-Employee	If hours<=40 wage*hours If hours>40 40*wage+(hours – 40) * wage *1.5	hourly employee: *firstName, lastName* ID:*empID* hourly wage: *wage* hourly worked: *hours*
Commission-Employee	commissionRate*gross Sales	commission employee: *firstName, lastName* ID:*empID* gross sales: *grossSales* commission rate: *commissionRate*

```
public class PayrollSystemTest  {
  public static void main (String args[])    {
    //create subclass objects
     SalariedEmployee salariedEmployee =
       new SalariedEmployee (“John”, “Smith”, “111-11-1111”, 800.00);
     HourlyEmployee hourlyEmployee =
       new HourlyEmployee (“Karen”, “Price”, “222-22-2222”, 16.75, 40);
     CommissionEmployee commissionEmployee =
new CommissionEmployee (“Sue”, “Jones”, “333-33-3333”, 10000, .06);

     System.out.println (“Employees processed individually:\n”);
      System.out.println (salariedEmployee + “ earned : “ +
                                  salariedEmployee.earnings() + “\n”);
     System.out.println (hourlyEmployee + “ earned : ” +
                                  hourlyEmployee.earnings() + “\n”);
     System.out.println  (commissionEmployee + “earned : ” +
                                  commissionEmployee.earnings() + “\n””);
```

```
    Employee employees[] = new Employee[ 3 ];
    employees[ 0 ] = salariedEmployee;
    employees[ 1 ] = hourlyEmployee;
    employees[ 2 ] = commissionEmployee;

    System.out.println( "Employees processed polymorphically:\n" );

    // generically process each element in array employees
    for (i = 0; i<employees.length; i++)     {
      if ( employees[i] instanceof HourlyEmployee )        {
        HourlyEmployee employee = ( HourlyEmployee ) employees[i];
        employee.setWage(18);
      }

      System.out.println( employees[i] + "earned:" +
                                          employees[i].earnings() + "\n");
    }
  }
}
```

Employees processed individually:

SalariedEmployee:John Smith
ID : 111-11-1111
weekly salary:800.0 earned:800.0

HourlyEmployee:Karen Price
ID : 222-22-2222
hourly wage:16.75
hours worked:40.0 earned:670.0

CommissionEmployee:Sue Jones
ID : 333-33-3333
gross sales:10000.0
commission rate0.06 earned:600.0

Employees processed polymorphically:

SalariedEmployee:John Smith
ID : 111-11-1111
weekly salary:800.0earned:800.0

HourlyEmployee:Karen Price
ID : 222-22-2222
hourly wage:18.0
hours worked:40.0earned:720.0

CommissionEmployee:Sue Jones
ID : 333-33-3333
gross sales:10000.0
commission rate0.06earned:600.0

References

- Deitel, H.M., and Deitel, P.J., *Java How to Program*, nineth edition, Prentice Hall, 2012.
- Horstmann, C., *Big Java*, John Wiley & Sons, 2009.
- Liang, Y. D., Introduction to Java Programming, tenth edition, Pearson Education Inc, 2015.