Tutorial - Week 04 5COSC019W – Object Oriented Programming – Java

Inheritance, superclass and polymorphism

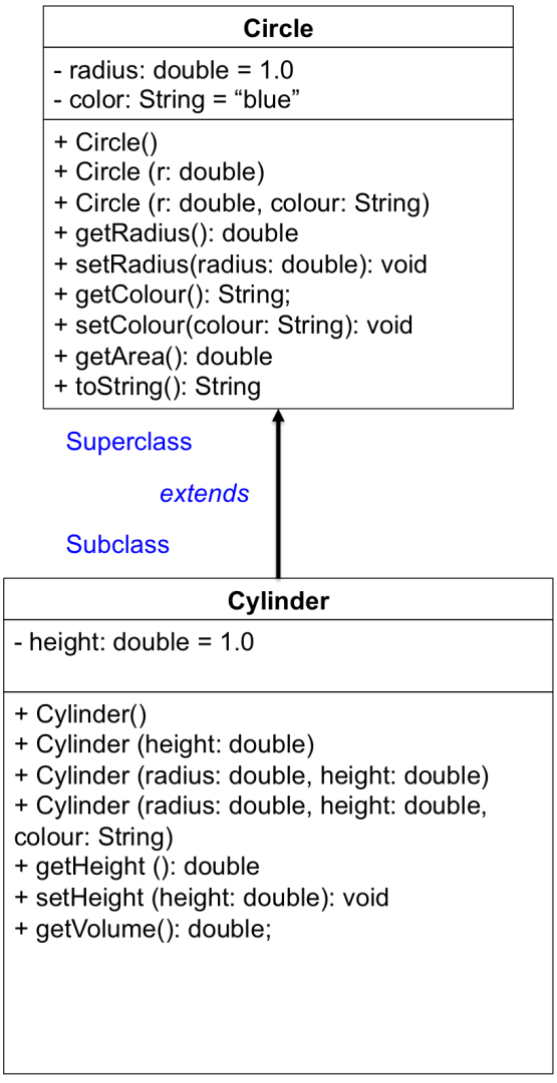
16October-20October 2023

Exercise 1

Inheritance

Inheritance allows a class to use the properties and methods of another class. In other words, the derived class inherits the states and behaviours from the base class. The derived class is also called *subclass* and the base class is also known as *super-class*. The derived class can add its own additional variables and methods.

In this first exercise, we want to create a subclass called `Cylinder`, which is derived from the superclass `Circle` as shown in the UML diagram below:



Circle Class and Code Reuse

As you can see the class Circle is the one you already implemented in your previous Tutorial. So you can reuse it, just make sure to keep the "Circle.java" file in the same directory and package where the class Cylinder will be.

Cylinder Class

A subclass `Cylinder` is derived from the superclass `Circle`. In the declaration of the class you should use the keyword `extends`:

```
public class Cylinder  extends Circle{
   //…here the code
}
```

1) Implement the class Cylinder based on the UML diagram. You can look at the following implementation and try to understand the use of the keyword `super`.

```
public class Cylinder extends Circle {
   private double height

   // Constructor with default color, radius and height
   public Cylinder() {
      super();
      height = 1.0;
   }

   // Constructor with default radius, color but given height
   public Cylinder(double height) {
      super();
      this.height = height;
   }

   // Constructor with default color, but given radius, height
   public Cylinder(double radius, double height) {
      super(radius);
      this.height = height;
   }

   // A public method for retrieving the height
   public double getHeight() {
      return height;
   }

   // A public method for computing the volume of cylinder
   //  use superclass method getArea() to get the base area
   public double getVolume() {
      double volume = getArea()*height;
      return volume;
   }
}
```

Call the superclass Circle constructor with no arguments

Call again the superclass Circle constructor with no arguments

Call the superclass constructor Circle (r), since there is one parameter

2) Write a test class to test the Cylinder class. If you are not sure how to write a test class you can refer to and study the following class:

```
public class TestCylinder {

   public static void main (String[] args) {
      // Declare and allocate a new instance of cylinder
      Cylinder c1 = new Cylinder();

      System.out.println("Cylinder:"
                       + " radius=" + c1.getRadius()
                       + " height=" + c1.getHeight()
                       + " base area=" + c1.getArea()
                       + " volume=" + c1.getVolume());

      // Declare and allocate a new instance of cylinder, specifying Height
      Cylinder c2 = new Cylinder(5.0);

      System.out.println("Cylinder:"
                       + " radius=" + c2.getRadius()
```

```
                        + " height=" + c2.getHeight()
                        + " base area=" + c2.getArea()
                        + " volume=" + c2.getVolume());

        // Declare and allocate a new instance of cylinder specifying radius and height
        Cylinder c3 = new Cylinder(5.0, 10.0);


        System.out.println("Cylinder:"
                        + " radius=" + c3.getRadius()
                        + " height=" + c3.getHeight()
                        + " base area=" + c3.getArea()
                        + " volume=" + c3.getVolume());
    }
}
```

Overriding

Overriding means to provide a specific implementation of a method that is already provided by one of its superclasses or parent classes.

3) The subclass Cylinder inherits the methods of the class Circle. If the method `getArea()` is called by the Cylinder instance it will compute the base area of a cylinder, not the surface area (because it is the one implemented in Circle).
In this exercise, you have to override the `getArea()` method in the class Cylinder so that it will calculate the surface area of the cylinder. (Remember that the surface area of a cylinder is = 2 π * radius * height + 2 * base-area).

4) In your main create an instance of the class Circle and one of the class Cylinder, and call the method `getArea()` from both instances and print the results you get on the screen.

5) After overriding the method `getArea()`, does the method `getVolume`() work correctly?

`getVolume()` uses the overriden `getArea()` found in the same class. In fact, Java will search in the superclass only if it cannot find the method in the base class.

How to fix it?

In order to use the getArea() method from the superclass you can call `super.getArea()`

6) Override in Cylinder the method toString inherited from the superclass Circle.

Example:

```
// in Cylinder class
@Override
public String toString() {
   return "Cylinder: subclass of " + super.toString()
          + " height=" + height;
    }
```

It uses the method toString() from the superclass Circle

Note: `@Override` is known as annotation (introduced in JDK 1.5). It is optional, but it is quite a handful. What it does is ask the compiler to check whether there is such a method in the superclass to be overridden. This helps greatly in case of misspelling: e.g. if `@Override` is not used and `toString()` is misspelled as for example `ToString()`, it will be treated as a new method in the subclass, instead of overriding the superclass. If `@Override` is used, the compiler will signal an error and you can recognise there was a mistake.

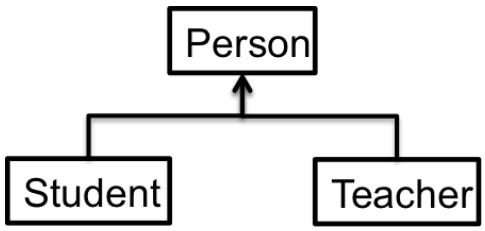
Exercise 2

According to what you learned so far let's write a SchoolApplication. This class will have a superclass Person and using inheritance you will implement two subclasses, Student and Teacher.

A `Teacher` will be like a `Person` but will have additional properties such as `salary` (the amount the teacher earns) and `subject` (e.g. "Computer Science", "Chemistry", "English", "Other").

A `Student` will be like a `Person` but will have additional properties such as `fee` (the amount the student pays every year), the grade and `IDnumber`.

The inheritance hierarchy should appear as follows:



This is the code for the class Person:

```
public class Person {

       protected String myName ;   // name of the person
       protected int myAge;        // person's age
       protected String myGender;  // "M" for male, "F" for female

       public Person(String name, int age, String gender)  {
                                  myName = name; myAge = age ; myGender = gender;   }
       }
       public String toString()  {
                                  return myName + ", age: " + myAge + ", gender: " +myGender;
       }
}
```

7) Add methods to "set" and "get" the instance variables in the Person class. These would consist of: getName, getAge, getGender, setName, setAge, and setGender.

8) Write a Student class that extends the parent class Person. The class should have:

- Three *instance variables*. They represent the IDNumber (int), the fee(double), and the grade (int).
- One *constructor* to initialise name, age, gender, idNum. Use the super reference to use the constructor in the Person superclass to initialise the inherited values.
- Write "setter" and "getter" methods for all of the class variables. For the Student class, they would be: getIDNum, getFee, setGrade, and s setIDNum, setFee, setGrade.
- Write the toString() method for the Student class. Use a super reference to do the things already done by the superclass.

9) Write a Teacher class that extends the parent class Person. The class should have:

- Two *instance variables*. They represent the salary (double), and the subject(string).
- One *constructor* to initialise name, age, subject, and salary. Use the super reference to use the constructor in the Person superclass to initialise the inherited values.
- Write "setter" and "getter" methods for all of the class variables. For the Teacher class they would be: getSalary, getSubject, setSalary, and setSubject.
- Write the toString() method for the Teacher class. Use a super reference to do the things already done by the superclass.

10) Write a testing class with a main() that constructs all of the classes (Person, Student, Teacher) and calls their toString() method. Sample usage would be:

```
Person jack = new Person("Jack Brooke", 27, "M");
System.out.println(jack);

Student beth = new Student("Elisabeth Smith", 16, "F", "122233");
System.out.println(beth);

Teacher sam = new Teacher("Sam Hamilton", 34, "M", "Computer Science",
50000);|
System.out.println(sam);
```

11) Try the following and prove which statements are correct:

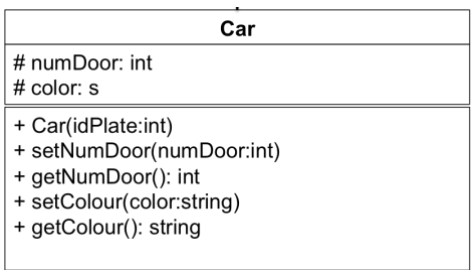
```
Person p = new Teacher ("Sam Hamilton", 34, "M", "Computer Science", 50000);

Teacher t = new Person ("Sam Hamilton", 34, "M", "Computer Science", 50000);

Person s = new Student ("Elisabeth Smith", 16, "F", "122233");
```

12) Each teacher has 3 students assigned as personal tutees. How would you model and implement this relationship?

Some theory questions in preparation for the In-class test:

1) Describe the Encapsulation principle in Object Oriented Programming.

2) Given the following UML representation of a class, write the Java code.



3) Explain with some graphical examples the Inheritance Principle.

4) Draw a use case diagram for a vending machine that sells beverages and snacks. Make use of inclusion and extension associations, and remember that a vending machine may need technical assistance from time to time.

Something more: Typical Interview Questions/Exercises

Try to solve the following exercise. It is a typical question that will likely be in an interview test. You will get the solution next week on BB!

1) Consider that you have a method “isSubstring” which checks if one word is a substring of another. Given two strings, s1 and s2, implement code to determine if s2 is a rotation of s1 using only one call to “isSubstring” (i.e., “waterbottle” is a rotation of “erbottlewat”).