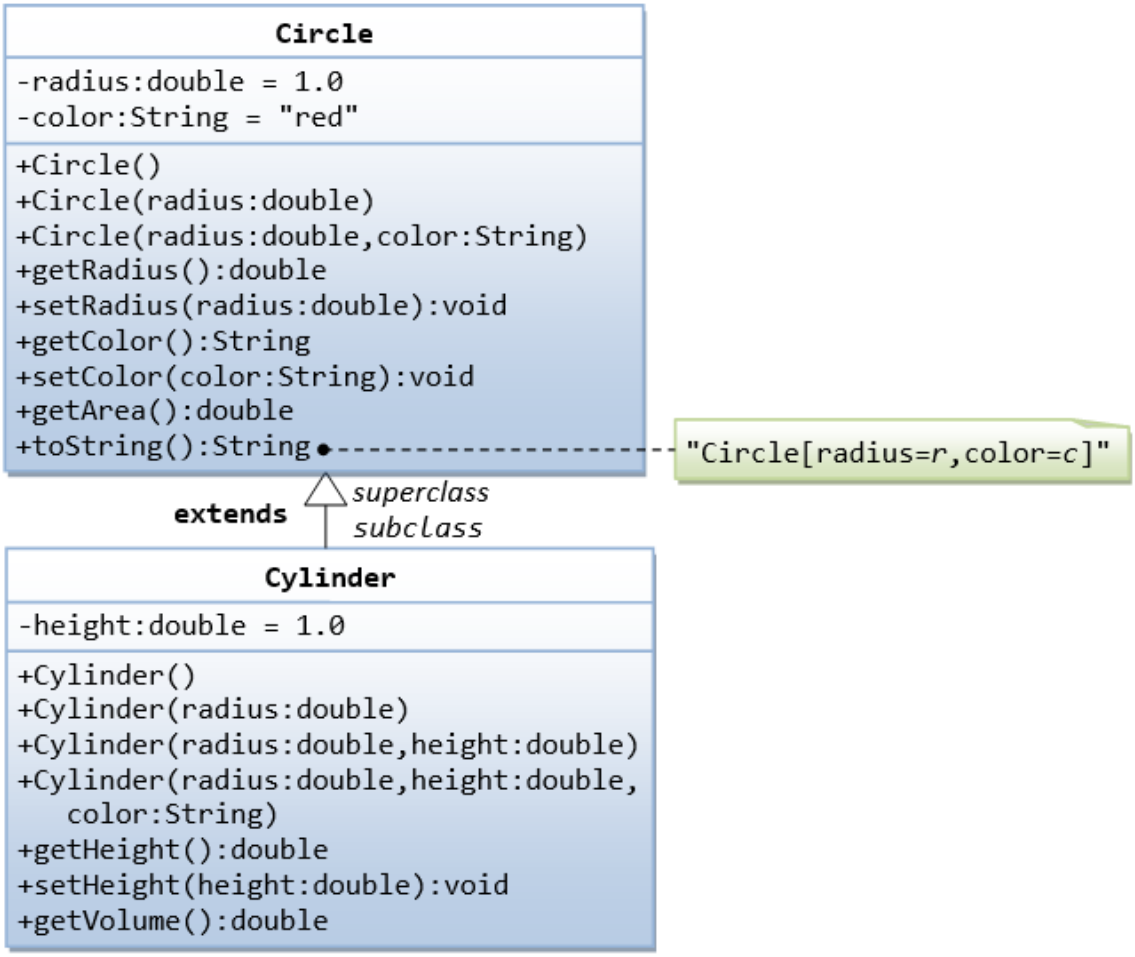
Week 6 : Inheritance, Abstract Class and Interface

Exercise 1 : The `Circle` and `Cylinder` Classes

This exercise shall guide you through the important concepts in inheritance.



In this exercise, a subclass called `Cylinder` is derived from the superclass `Circle` as shown in the class diagram (where an an arrow pointing up from the subclass to its superclass). Study how the subclass `Cylinder` invokes the superclass' constructors (via `super()` and `super(radius)`) and inherits the variables and methods from the superclass `Circle`.

The source codes for `Circle.java` is as follows:

```
/**
 * The Circle class models a circle with a radius and color.
 */
public class Circle {  // Save as "Circle.java"
   // private instance variable, not accessible from outside this class
   private double radius;
   private String color;
```

```
   // Constructors (overloaded)
   /** Constructs a Circle instance with default value for radius and color */
   public Circle() {  // 1st (default) constructor
      radius = 1.0;
      color = "red";
   }

   /** Constructs a Circle instance with the given radius and default color */
   public Circle(double r) {  // 2nd constructor
      radius = r;
      color = "red";
   }

   /** Returns the radius */
   public double getRadius() {
     return radius;
   }

   /** Returns the area of this Circle instance */
   public double getArea() {
      return radius*radius*Math.PI;
   }

   /**  Return  a  self-descriptive  string  of  this  instance  in  the  form  of
Circle[radius=?,color=?] */
   public String toString() {
      return "Circle[radius=" + radius + " color=" + color + "]";
   }
}
```

[Task 1.1] Modify class `Circle`

Modify class Circle, add :

1. variable color : string
2. Constructor Circle(radius : double, color : string)
3. Getter and setter for color

Modifikasi kelas circle dengan menambahkan atribut dan method seperti di bawah ini

You can reuse the Circle class above.

Make sure that you keep "`Circle.class`" in the same directory.

```
public class Cylinder extends Circle {  // Save as "Cylinder.java"
   private double height;  // private variable

   // Constructor with default color, radius and height
   public Cylinder() {
```

```
      super();        // call superclass no-arg constructor Circle()
      height = 1.0;
   }
   // Constructor with default radius, color but given height
   public Cylinder(double height) {
      super();        // call superclass no-arg constructor Circle()
      this.height = height;
   }
   // Constructor with default color, but given radius, height
   public Cylinder(double radius, double height) {
      super(radius);  // call superclass constructor Circle(r)
      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() {
      return getArea()*height;
   }
}
```

Write a test program (says `TestCylinder`) to test the `Cylinder` class created, as follow:

```
public class TestCylinder {  // save as "TestCylinder.java"
   public static void main (String[] args) {
      // Declare and allocate a new instance of cylinder
      //   with default color, radius, and height
      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, with default color and radius
      Cylinder c2 = new Cylinder(10.0);
      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, with default color
```

```
      Cylinder c3 = new Cylinder(2.0, 10.0);
      System.out.println("Cylinder:"
            + " radius=" + c3.getRadius()
            + " height=" + c3.getHeight()
            + " base area=" + c3.getArea()
            + " volume=" + c3.getVolume());
   }
}
```

[Task 1.2] Overriding the `getArea()` method Mempraktikan Overriding

Method Overriding and "Super": The subclass `Cylinder` inherits `getArea()` method from its superclass Circle. Try *overriding* the `getArea()` method in the subclass `Cylinder` to compute the surface area (=2π×radius×height + 2×base-area) of the cylinder instead of base area. That is, if `getArea()` is called by a `Circle` instance, it returns the area. If `getArea()` is called by a `Cylinder` instance, it returns the surface area of the cylinder.

If you override the `getArea()` in the subclass `Cylinder`, the `getVolume()` no longer works. This is because the `getVolume()` uses the *overridden* `getArea()` method found in the same class. (Java runtime will search the superclass only if it cannot locate the method in this class). Fix the `getVolume()`.

Hints: After overridding the `getArea()` in subclass `Cylinder`, you can choose to invoke the `getArea()` of the superclass `Circle` by calling `super.getArea()`.

[Task 1.3] Provide a `toString()` method Menyediakan method toString dengan merubah konten pada toString yang ada pada di class circle pada class cylinder

Provide a `toString()` method to the `Cylinder` class, which overrides the `toString()` inherited from the superclass `Circle`, e.g.,

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

Try out the `toString()` method in `TestCylinder`.

Note: `@Override` is known as *annotation* (introduced in JDK 1.5), which asks compiler to check whether there is such a method in the superclass to be overridden. This helps greatly if you misspell the name of the `toString()`. If `@Override` is not used and `toString()` is misspelled as `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. `@Override` annotation is optional, but certainly nice to have.

Exercise 2 : Superclass Shape and its Subclasses Circle, Rectangle and Square

[Task 2.1]

Write a superclass called `Shape` (as shown in the class diagram), which contains:

- Two instance variables `color` (`String`) and `filled` (`boolean`).
- Two constructors: a no-arg (no-argument) constructor that initializes the `color` to "green" and `filled` to `true`, and a constructor that initializes the `color` and `filled` to the given values.
- Getter and setter for all the instance variables. By convention, the getter for a `boolean` variable `xxx` is called `isXXX()` (instead of `getXxx()` for all the other types).
- A `toString()` method that returns "`A Shape with color of xxx and filled/Not filled`".

Write a test program to test all the methods defined in `Shape`.

Write two subclasses of `Shape` called `Circle` and `Rectangle`, as shown in the class diagram.

The `Circle` class contains:

- An instance variable `radius` (`double`).
- Three constructors as shown. The no-arg constructor initializes the radius to `1.0`.
- Getter and setter for the instance variable `radius`.
- Methods `getArea()` and `getPerimeter()`.
- Override the `toString()` method inherited, to return "`A Circle with radius=xxx, which is a subclass of yyy`", where `yyy` is the output of the `toString()` method from the superclass.

The `Rectangle` class contains:

- Two instance variables `width` (`double`) and `length` (`double`).
- Three constructors as shown. The no-arg constructor initializes the `width` and `length` to `1.0`.
- Getter and setter for all the instance variables.
- Methods `getArea()` and `getPerimeter()`.
- Override the `toString()` method inherited, to return "`A Rectangle with width=xxx and length=zzz, which is a subclass of yyy`", where `yyy` is the output of the `toString()` method from the superclass.

Write a class called `Square`, as a subclass of `Rectangle`. Convince yourself that `Square` can be modeled as a subclass of `Rectangle`. `Square` has no instance variable, but inherits the instance variables width and length from its superclass Rectangle.

- Provide the appropriate constructors (as shown in the class diagram). Hint:

```
public Square(double side) {
   super(side, side);  // Call superclass Rectangle(double, double)
}
```

- Override the `toString()` method to return "`A Square with side=xxx, which is a subclass of yyy`", where `yyy` is the output of the `toString()` method from the superclass.
- Do you need to override the `getArea()` and `getPerimeter()`? Try them out.
- Override the `setLength()` and `setWidth()` to change both the `width` and `length`, so as to maintain the square geometry.

Exercise 3 : Multiple Inheritance

The next two examples (a class describing a set of Employees employed in a factory, and a class describing a subset of Employees which are Manager in the same factory, represent an example to show the expressive power of Inheritance in Java. In particular, the class Manager is defined as subclass of Employee, in order to reuse the software written for the Employee class.

Class Employee.java

```
class Employee{
       public Employee(String n, double s, int day, int month, int year){
          name = n;
          salary = s;
          hireday = day;
          hiremonth = month;
          hireyear = year;
       }

       public void print(){
          System.out.println(name + " " + salary + " " + hireYear());
       }

       public void raiseSalary(double byPercent){
          salary *= 1 + byPercent / 100;
       }

       public int hireYear(){
               return hireyear;
       }

       private String name;
       private double salary;
       private int hireday;
       private int hiremonth;
       private int hireyear;
}
```

Class EmployeeTest.java

```
public class EmployeeTest{
       public static void main (String[] args){
           Employee[] staff = new Employee[3];
           staff[0] = new Employee("Antonio Rossi", 2000000, 1, 10, 1989);
           staff[1] = new Employee("Maria Bianchi", 2500000, 1, 12, 1991);
           staff[2] = new Employee("Isabel Vidal", 3000000, 1, 11, 1993);
           int i;
           for (i = 0; i < 3; i++) staff[i].raiseSalary(5);
           for (i = 0; i < 3; i++) staff[i].print();
       }
}
```

Class Manager.java

```
class Manager extends Employee{
	public Manager (String n, double s, int d, int m, int y){
		super(n, s, d, m, y);
		secretaryName = "";
	}

	public void raiseSalary(double byPercent){
		// add 1/2% bonus for every year of service
		GregorianCalendar todaysDate = new GregorianCalendar();
		int currentYear = todaysDate.get(Calendar.YEAR);
		double bonus = 0.5 * (currentYear - hireYear());
		super.raiseSalary(byPercent + bonus);
	}

	public String getSecretaryName(){
		return secretaryName;
	}
	private String secretaryName;
}
```

Class ManagerTest.java

```
public class ManagerTest{
	public static void main (String[] args){
		Employee[] staff = new Employee[3];
		staff[0] = new Employee("Antonio Rossi", 2000000, 1, 10, 1989);
		staff[1] = new Manager("Maria Bianchi", 2500000, 1, 12, 1991);
		staff[2] = new Employee("Isabel Vidal", 3000000, 1, 11, 1993);

		int i;
		for (i = 0; i < 3; i++) staff[i].raiseSalary(5);
		for (i = 0; i < 3; i++) staff[i].print();
	}

}
```

[Task 3.1] Extending the Sortable abstract class

Write code above, and analyzed how it work.

[Case 1]

There is an abstract class named Sortable.

```
abstract class Sortable{
   public abstract int compare(Sortable b);
   public static void shell_sort(Sortable[] a){
     //Shell sort body
   }
}
```

When Sortable extended to Employee class, the method compare will be implemented.

```
class Employee extends Sortable{
   /* another methods */

   public int compare(Sortable b){
      Employee eb = (Employee) b;
      if (salary<eb.salary) return -1;
      if (salary>eb.salary) return +1;
      return 0;
   }
}
```

[Try] Please try the codes above. Call the method compare, in EmployeeTest class

```
   Employee[] staff = new Employee[3];
   staff[0] = new Employee("Antonio Rossi", 2000000, 1, 10, 1989);
   staff[1] = new Employee("Maria Bianchi", 2500000, 1, 12, 1991);
   staff[2] = new Employee("Isabel Vidal", 3000000, 1, 11, 1993);
   Sortable.shell_sort(staff);
```

[Case 2]

Imagine that we want to order the Managers in a similar way :

```
  class Managers extends Employee extends Sortable
```

implements

It will be work?

What is your solution?