BIRZEIT UNIVERSITY

**COMP231 – Advanced Programming**

# Assignment # 3

## Objectives:

1. Create hierarchy of Classes and Objects using Inheritance relationships.
2. Demonstrate the added value of using the following concepts:
   - Inheritance
   - Polymorphism
   - Generic programming
   - Dynamic biding

## Specification

**Submission**: Online **through Ritaj** as a reply to this message.

**What to submit:** Your **own** **well-structured** and **well-commented** JAVA files (**.java**) + **PDF** file (compressed into a **studentId_sec#.rar** file, e.g. **1234567_sec1.rar**).

**Deadline**: **20/11/2015** by midnight. (The online submission will be disabled after this time).

## Tasks

**Task 1:**

(*The* `Person`, `Student`, `Employee`, `Faculty`, *and* `Staff` *classes*) Design a class named `Person` and its two subclasses named `Student` and `Employee`. Make `Faculty` and `Staff` subclasses of `Employee`. A person has a name, address, phone number, and email address. A student has a class status (freshman, sophomore, junior, or senior). Define the status as a constant. An employee has an office, salary, and date hired.

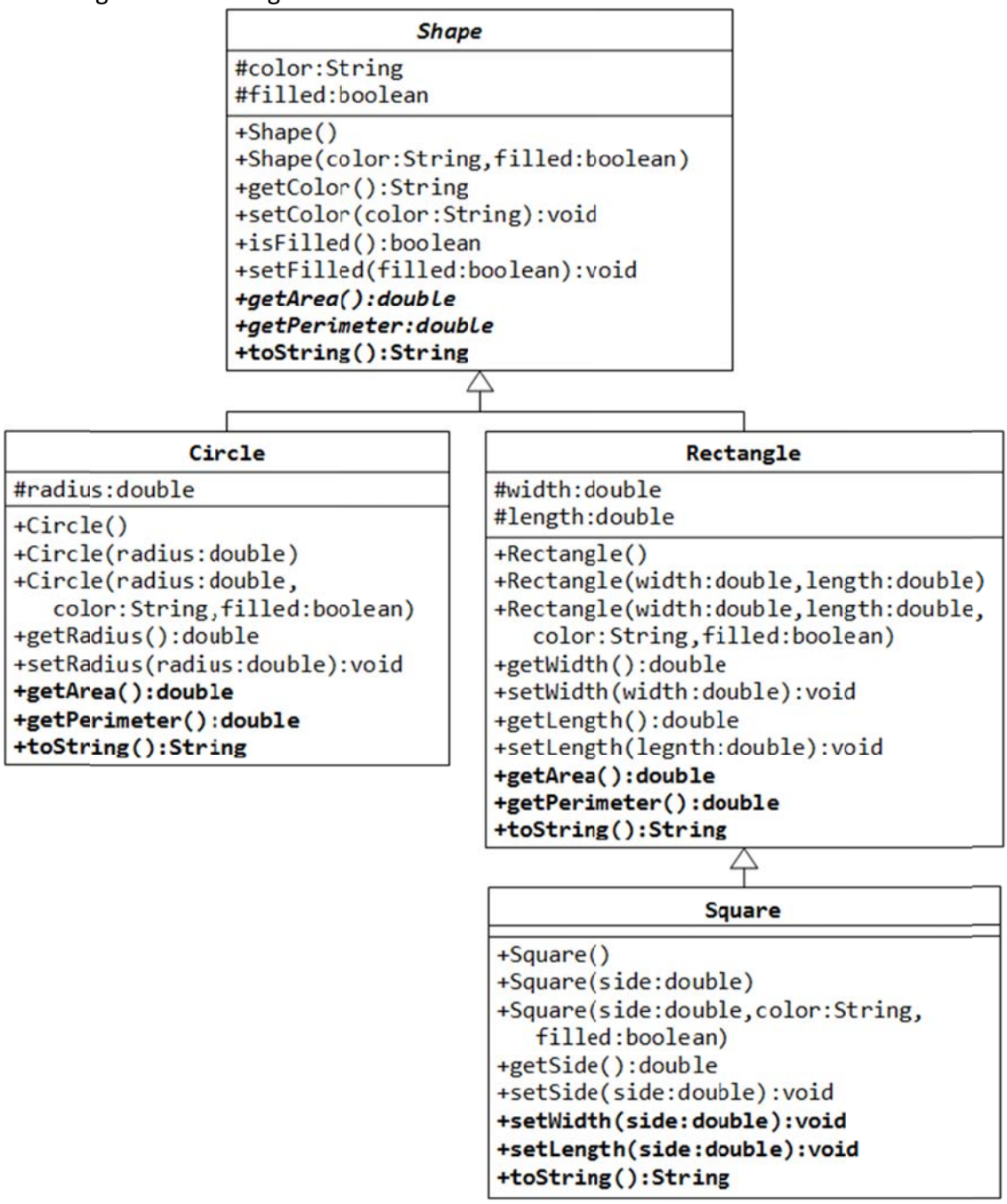
A faculty member has office hours and a rank. A staff member has a title. Override the `toString` method in each class to display the class name and the person's name.

Draw the UML diagram for the classes and implement them.

Write a test program that creates an **ArrayList** of 10 different types of **Person**s and then print the list items by invoking each **toString()** method.

Task 2:

Implement the superclass **Shape** and its subclasses **Circle**, **Rectangle** and **Square**, as shown in the following UML class diagram:



- Write a test class to test the following statements involving polymorphism and explain the outputs. Some statements may trigger compilation errors. Explain the errors (write next to each statement), if any:

```
Shape s1 = new Circle(5.5, "RED", false);  // Upcast Circle to Shape
System.out.println(s1);                    // which version?
System.out.println(s1.getArea());          // which version?
System.out.println(s1.getPerimeter());     // which version?
System.out.println(s1.getColor());
System.out.println(s1.isFilled());
System.out.println(s1.getRadius());

Circle c1 = (Circle)s1;                   // Downcast back to Circle
System.out.println(c1);
System.out.println(c1.getArea());
System.out.println(c1.getPerimeter());
System.out.println(c1.getColor());
System.out.println(c1.isFilled());
System.out.println(c1.getRadius());
```

```
Shape s2 = new Shape();

Shape s3 = new Rectangle(1.0, 2.0, "RED", false);   // Upcast
System.out.println(s3);
System.out.println(s3.getArea());
System.out.println(s3.getPerimeter());
System.out.println(s3.getColor());
System.out.println(s3.getLength());

Rectangle r1 = (Rectangle)s3;   // downcast
System.out.println(r1);
System.out.println(r1.getArea());
System.out.println(r1.getColor());
System.out.println(r1.getLength());

Shape s4 = new Square(6.6);     // Upcast
System.out.println(s4);
System.out.println(s4.getArea());
System.out.println(s4.getColor());
System.out.println(s4.getSide());
```

```
// Take note that we downcast Shape s4 to Rectangle,
//  which is a superclass of Square, instead of Square
Rectangle r2 = (Rectangle)s4;
System.out.println(r2);
System.out.println(r2.getArea());
System.out.println(r2.getColor());
System.out.println(r2.getSide());
System.out.println(r2.getLength());

// Downcast Rectangle r2 to Square
Square sq1 = (Square)r2;
System.out.println(sq1);
System.out.println(sq1.getArea());
System.out.println(sq1.getColor());
System.out.println(sq1.getSide());
System.out.println(sq1.getLength());
```

Good Luck!