

## Introduction to Computer Programming (Java A)

### Lab 10

#### [Objective]

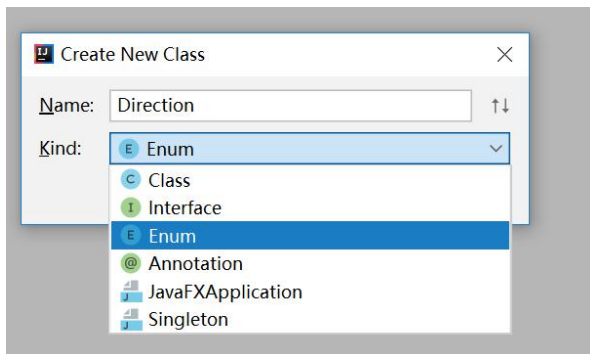
- Learn inheritance.
- Learn protected keyword.
- Learn to use enum types.

#### [Exercises]

#### Part 1: Enumerations

An enum type is a special data type that enables a variable to be a set of predefined constants. The variable must be equal to one of the values that have been predefined for it. For example, a week has seven days (MONDAY to SUNDAY).

A enum type is declared using the enum keyword, not class. Let's create a new enum type Direction with four constants named "NORTH", "SOUTH", "EAST", and "WEST", respectively. In IDEA, creating a new enum type is similar to creating a new class. The only difference is to select "Enum" in the dropdown list.



```
package sustech.cs102a.lab10;
```

```
public enum Direction {  
    NORTH, SOUTH, EAST, WEST // semicolon unnecessary  
}
```

Variables of this enum type Direction can only receive the values of the four enum constants. For example, the following code creates an object of this enum type.

```
package sustech.cs102a.lab10;
```

```
public class DirectionTest {  
    public static void main(String[] args) {  
        Direction d = Direction.EAST;  
        System.out.println(d);  
    }  
}
```

```
    }  
}
```

The above code prints “EAST”. The last statement in the main method is equivalent to `System.out.println(d.toString())`. The `toString()` method returns the name of the enum constant `EAST`.

In the code, we cannot create an object of the enum type using the “new” operator with a constructor call. If you compile the following code, you will receive the error message “Enum types cannot be instantiated”.

```
public Direction d = new Direction();
```

This is because under the hood, every enum type is internally implemented using class (the compiler will create a private constructor that cannot be called outside the enum type).

```
public final class Direction extends Enum {  
    public static final Direction NORTH = new Direction();  
    public static final Direction SOUTH = new Direction();  
    public static final Direction EAST = new Direction();  
    public static final Direction WEST = new Direction();  
} // simplified for illustration
```

From this internal view, we can see that `NORTH`, `SOUTH`, `EAST`, `WEST` are no more than four class variables pointing to four `Direction` objects. The `final` modifier makes them constants.

An enum type variable can be passed as an argument to a “switch” statement.

```
package sustech.cs102a.lab10;  
  
public class DirectionTest {  
    private Direction d;  
  
    public DirectionTest(Direction d) {  
        this.d = d;  
    }  
  
    public Direction getDirection() {  
        return d;  
    }  
  
    public static void main(String[] args) {  
        DirectionTest test = new DirectionTest(Direction.EAST);  
        switch(test.getDirection()) {  
            case EAST: // must be unqualified name of the enum constant  
                System.out.println("Countries in the east: Japan, Korea");  
                break;  
            case WEST:  
                System.out.println("Countries in the west: US, Germany");  
                break;  
            case SOUTH:  
                System.out.println("Countries in the south: Australia, New
```

```
Zealand");
    break;
case NORTH:
    System.out.println("Countries in the north: Russia, Mongolia");
    break;
}
}
```

When declaring an enum type, besides the enum constants, we can also declare other members such as **constructors**, **fields** and **methods**. A enum type constructor can specify any number of parameters and can be overloaded, but it cannot have the access modifier `public` (must be `private` or no-modifier, meaning package private).

```
package sustech.cs102a.lab10;

public enum Book {
    JHTP("Java: How to Program", "2012"),
    CHTP("C: How to Program"),
    CPPHTP("C++: How to Program", "2012"),
    VBHTP("Visual Basic: How to Program", "2011"),
    CSHARPHTP("Visual C#: How to Program");

    private final String title;
    private final String year;

    private Book(String title, String year) {
        this.title = title;
        this.year = year;
    }

    private Book(String title) {
        this.title = title;
        this.year = "no info";
    }

    public String getTitle() {
        return title;
    }

    public String getYear() {
        return year;
    }
}
```

In the enum type `Book`, there are two fields: `title` and `year`. They are declared to be constants since enum type objects only receive predefined constant values (enum constants). There are two getter methods. There are two overloaded constructors. The two constructors are used in the declarations of the enum constants. For example, when declaring the enum constant `CHTP`, the one-argument constructor is used.

We can further write the following program to test the enum type.

```
package sustech.cs102a.lab10;
import java.util.EnumSet;

public class BookTest {
    public static void main(String[] args) {
        System.out.println("All books:");

        for (Book book : Book.values()) {
            System.out.printf("%-10s", book);
            System.out.printf("%-30s", book.getTitle());
            System.out.printf("%s\n", book.getYear());
        }

        System.out.println("\nDisplaying a range of enum
constants:");

        for (Book book : EnumSet.range(Book.JHTP, Book.CPPHTP)) {
            System.out.printf("%-10s", book);
            System.out.printf("%-30s", book.getTitle());
            System.out.printf("%s\n", book.getYear());
        }
    }
}
```

The code prints:

|                                       |                              |         |
|---------------------------------------|------------------------------|---------|
| All books:                            |                              |         |
| JHTP                                  | Java: How to Program         | 2012    |
| CHTP                                  | C: How to Program            | no info |
| CPPHTP                                | C++: How to Program          | 2012    |
| VBHTP                                 | Visual Basic: How to Program | 2011    |
| CSHARPHTP                             | Visual C#: How to Program    | no info |
|                                       |                              |         |
| Displaying a range of enum constants: |                              |         |
| JHTP                                  | Java: How to Program         | 2012    |
| CHTP                                  | C: How to Program            | no info |
| CPPHTP                                | C++: How to Program          | 2012    |

In the above example, only five Book objects will be created. The constants such as Book.JHTP stores the references to the objects.

The values() method is a static method that is automatically generated by the compiler to return an array of the enum constants (an array of references to the objects of the enum type).

The generic class EnumSet's static method range() returns a collection of the enum constants in the range specified by two endpoints. In the above code, range() takes two enum constants as arguments. **The first constant should be declared before the second** (the ordinal() method of a enum constant can return the position of the constant in all declared constants). If this constraint is violated (for example, when EnumSet.range(Book.CPPHTP, Book.JHTP) is used in the code), an java.lang.IllegalArgumentException will be thrown.

### Lab exercise:

1. Create an enum type PhoneModel, which contains the following constants: IPHONE, HUAWEI, PIXEL, SAMSUNG, LG.
2. Create a field named price (int type). Write a getter method for this field.
3. Create a one-argument constructor PhoneModel(int price) that can be used to create the enum constants. The prices for the five models are: 9999, 8888, 6666, 9399, 5588.
4. Write a test program .It contains a main method that recommends possible phones for a user based on the user's budget.

Three sample runs:

|                              |             |
|------------------------------|-------------|
| Your budget: 4000            |             |
| You do have sufficient money |             |
| Your budget: 8888            |             |
| HUAWEI                       | price: 8888 |
| PIXEL                        | price: 6666 |
| LG                           | price: 5588 |
| Your budget: 10000           |             |
| IPHONE                       | price: 9999 |
| HUAWEI                       | price: 8888 |
| PIXEL                        | price: 6666 |
| SAMSUNG                      | price: 9399 |
| LG                           | price: 5588 |

## Part 2: Inheritance

### Copy the following code to Circle.java

```
public class Circle {
    private double radius;
    private double x;
    private double y;
    static final int DEFAULT_RADIUS = 5;
    private static int screenSize = 10;
    private ShapeColor color = ShapeColor.GRAY;

    public Circle(double radius, double x, double y) {
        this.radius = radius;
        this.x = x;
```

```
        this.y = y;
    }

    public Circle(double radius) {
        this.radius = radius;
        this.x = 0;
        this.y = 0;
    }

    public Circle(double x, double y) {
        this.radius = DEFAULT_RADIUS;
        this.x = x;
        this.y = y;
    }

    public static int getScreenSize() {
        return screenSize;
    }

    public static void setScreenSize(int screenSize) {
        Circle.screenSize = screenSize;
    }

    public void checkColor() {
        if (isInBoundary()) {
            color = ShapeColor.GREEN;
        } else {
            color = ShapeColor.RED;
        }
    }

    public boolean isInBoundary() {
        if (-1 * Circle.screenSize > this.x - this.radius || Circle.screenSize < this.x + this.radius) {
            return false;
        }
        if (-1 * Circle.screenSize > this.y - this.radius || Circle.screenSize < this.y + this.radius) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "Circle{" + "radius=" + "x=" + x +
            ", y=" + y + ", color=" + color + "}\n";
    }

    public double getRadius() {
        return radius;
    }

    public void setRadius(double radius) {
        this.radius = radius;
    }

    public double getX() {
        return x;
    }

    public void setX(double x) {
        this.x = x;
    }

    public double getY() {
        return y;
    }

    public void setY(double y) {
        this.y = y;
    }

    public void draw() {
        StdDraw.setPenColor(color.getColor());
        StdDraw.filledCircle(x, y, radius);
    }
}
```

```
}  
}
```

**Copy the following code to Rectangle.java**

```
public class Rectangle {  
    private double x;  
    private double y;  
    private double width;  
    private double height;  
    private static int screenSize = 10;  
    private ShapeColor color = ShapeColor.GRAY;  
  
    public Rectangle(double x, double y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public Rectangle(double x, double y, double width, double height) {  
        this.x = x;  
        this.y = y;  
        this.width = width;  
        this.height = height;  
    }  
  
    public static int getScreenSize() {  
        return screenSize;  
    }  
  
    public static void setScreenSize(int screenSize) {  
        Rectangle.screenSize = screenSize;  
    }  
  
    public void checkColor() {  
        if (isInBoundary()) {  
            color = ShapeColor.GREEN;  
        } else {  
            color = ShapeColor.RED;  
        }  
    }  
  
    public boolean isInBoundary() {  
        if (-1 * Rectangle.screenSize > this.x - this.width / 2 || Rectangle.screenSize < this.x  
+ this.width / 2) {  
            return false;  
        }  
        if (-1 * Rectangle.screenSize > this.y - this.height / 2 || Rectangle.screenSize < this.y  
+ this.height / 2) {  
            return false;  
        }  
        return true;  
    }  
  
    public double getX() {  
        return x;  
    }  
  
    public void setX(double x) {  
        this.x = x;  
    }  
  
    public double getY() {  
        return y;  
    }  
  
    public void setY(double y) {  
        this.y = y;  
    }  
  
    public double getWidth() {  
        return width;  
    }  
}
```

```
        public void setWidth(double width) {
            this.width = width;
        }

        public double getHeight() {
            return height;
        }

        public void setHeight(double height) {
            this.height = height;
        }

        public String toString() {
            return "Rectangle{" + "width=" + width + ", height=" + height + " x=" + x +
                ", y=" + y + ", color=" + color + "}\n";
        }

        public void draw() {
            StdDraw.setPenColor(color.getColor());
            StdDraw.filledRectangle(x, y, this.width / 2, this.height / 2);
        }
    }
```

### Copy the following code to ShapeColor.java

```
import java.awt.Color;

public enum ShapeColor {
    GREEN("The shape is in the Screen", Color.GREEN), RED("The shape is not in the Screen",
        Color.RED), GRAY("Haven't tested", Color.GRAY);

    private String desc; // The description of instance
    private Color color; // The color of instance

    ShapeColor (String desc, Color color) {
        this.desc = desc;
        this.color = color;
    }

    public String getDesc() {
        return this.desc;
    }

    public Color getColor() {
        return this.color;
    }
}
```

**Download StdDraw.java from the internet or from our course site ( blackboard / sakai ) . Place it at the same location as the above 3 java files.**

Now, you should have 4 java files prepared and start the following exercises.

## 1. Inheritance

From the source code, it is observed that the two classes: Circle and Rectangle have a lot of common fields, e.g., **screenSize**, **x**, **y** and **ShapeColor**, and **a lot of similar methods**. It is a good time to practice inheritance by refactoring the code.

The idea of inheritance is simple but powerful: When you want to create a new class and there is an existing class which includes some of the code that you want,



you can extend your new class from the existing class. In doing this, you can reuse the fields and methods of the existing class without having to write (and debug!) them yourself.

A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the subclass must invoke one of the constructors in its superclass. (<https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html> )

We found that, the attributes **x**, **y**, **color** and **screenSize** are both in **Circle** and **Rectangle**, then for those common attributes are more appropriated to be extracted into a super class named **Shape**, from which the subclass can use all attributes and methods.

Create a class Shape, which contains following members:

(1) Adding attributes:

```
private double x;  
private double y;  
private ShapeColor color = ShapeColor.GRAY;  
private static int screenSize = 10;
```

(2) Adding constructors with two parameters **x** and **y** in the Shape class.

```
public Shape(double x, double y) {  
    this.x = x;  
    this.y = y;  
}
```

(3) Adding getter/setter methods for the private variable;

(4) Adding toString() method (override the method of the Object class) to output the property of the Shape object;

```
@Override  
public String toString() {  
    return "x=" + x + ", y=" + y + ", color=" + color;  
}
```

Now, modify class **Circle**.

Let it inherit class **Shape** by using the keyword **extends**.

Remove those methods and attributes which can be inherited from class **Shape**.

Now class Circle only needs to define two specific attributes: radius and DEFAULT\_RADIUS.

```
private double radius;  
private static final int DEFAULT_RADIUS = 5;
```

Modify the constructor of class Circle as follows and use super ().

```
public Circle(double radius, double x, double y) {
    this.radius = radius;
    this.x = x;
    this.y = y;
}

public Circle(double radius) {
    this.radius = radius;
    this.x = 0;
    this.y = 0;
}

public Circle(double x, double y) {
    this.radius = DEFAULT_RADIUS;
    this.x = x;
    this.y = y;
}
```

Now, `x` and `y` are the attributes inherited from `Shape`. A recommend way is use the constructor of super class to initial the supper class.

For example:

```
public Circle(double radius) {
    super(0,0);
    this.radius = radius;
}
```

`this` serves as the current object, while `super` serves as the only supper class for current object.

Rewrite other constructors in the same way.

### Learn how to access the instance fields and static fields of super class in a subclass.

We will find that some errors occur in other methods, for example:

```
public boolean isInBoundary() {
    if (-1 * Circle.screenSize > this.x - this.radius || Circle.screenSize < this.x + this.radius) {
        return false;
    }
    if (-1 * Circle.screenSize > this.y - this.radius || Circle.screenSize < this.y + this.radius) {
        return false;
    }
    return true;
}
```

Change `Circle.screenSize` to `Shape.getScreenSize()` since `screenSize` is a private static field.

Change `this.x` to `super.getX()` since `x` is a private field of supper class, and so on.

```
public boolean isInBoundary() {
    if (-1 * Shape.getScreenSize() > super.getX() - this.radius
        || Shape.getScreenSize() < super.getX() + this.radius) {
        return false;
    }
    if (-1 * Shape.getScreenSize() > super.getY() - this.radius
        || Shape.getScreenSize() < super.getY() + this.radius) {
        return false;
    }
    return true;
}
```

Change other methods in the same way.

## Learn protected keyword

We can find that Circle is inconvenient to access the private attributes of superclass, so we can consider making these frequent-used attributes accessible to subclass.

**Protected** can help us.

**Step1:** Change **x**, **y** and **color** from **private** to **protected**.

```
protected double x;
protected double y;
protected ShapeColor color = ShapeColor.GRAY;
```

**Step2:** Then we change the `isInBoundary()` back to the original one except `Shape.getScreenSize()`, it can work well.

```
public boolean isInBoundary() {
    if (-1 * Shape.getScreenSize() > x - this.width / 2
        || Shape.getScreenSize() < x + this.width / 2) {
        return false;
    }
    if (-1 * Shape.getScreenSize() > y - this.height / 2
        || Shape.getScreenSize() < y + this.height / 2) {
        return false;
    }
    return true;
}
```

Change other methods in the same way.

The access right of each access modifier is shown as follows:

|   | private | default | protected                                | public |
|---|---------|---------|--|--------|
| Same package<br>same class                        | ✓       | ✓       | ✓  | ✓      |
| Same package<br>other classes                     |         | ✓       | ✓  | ✓      |
| Other packages<br>Other classes<br>Inheritance    |         |         | Inherit but<br>cannot access<br>directly | ✓      |
| Other packages<br>Other classes<br>No inheritance |         |         |  | ✓      |

## Lab exercise:

Now, modify the given class **Rectangle** to make it inherits from class **Shape**

- Make **Rectangle** extends **Shape**.
- Modify the constructors of **Rectangle**
- Modify other methods of **Rectangle**.
- Modify **toString()** method.

Run the following **ShapeTest** to test your modifications.

```
public class ShapeTest {

    public static void main(String[] args) {
        Circle c1=new Circle(0.1,1,1);
    }
}
```

```
Circle c2=new Circle(0.1,0.5,2);
Circle.setScreenSize(2);
System.out.print(c1);
c1.checkColor();
c2.checkColor();
System.out.print(c1);
System.out.print(c2);

Rectangle r1=new Rectangle(0,0,0.5,0.5);
Rectangle r2=new Rectangle(2,1,0.5,0.5);
Rectangle.setScreenSize(2);
System.out.print(r1);
r1.checkColor();
r2.checkColor();
System.out.print(r1);
System.out.print(r2);

StdDraw.setXscale(-Circle.getScreenSize(), Circle.getScreenSize());
StdDraw.setYscale(-Circle.getScreenSize(), Circle.getScreenSize());
c1.draw();
c2.draw();
r1.draw();
r2.draw();
Circle c3=new Circle(0.1,0.5,-2);
Rectangle r3=new Rectangle(-2,1,0.5,0.5);
c3.draw();
r3.draw();
}
}
```

**Output:**

Circle{radius=0.1, x=1.0, y=1.0, color=GRAY}  
Circle{radius=0.1, x=1.0, y=1.0, color=GREEN}  
Circle{radius=0.1, x=0.5, y=2.0, color=RED}  
Rectangle{width=0.5, height=0.5 x=0.0, y=0.0, color=GRAY}  
Rectangle{width=0.5, height=0.5 x=0.0, y=0.0, color=GREEN}  
Rectangle{width=0.5, height=0.5 x=2.0, y=1.0, color=RED}

