Problem Set 02 Worksheet

Question 1

```
public class Rectangle {
  private int width;
  private int height;

  public Rectangle(int width, int height) {
    this.width = width;
    this.height = height;
  }

  public int getArea() {
    return this.width * this.height;
  }

  @Override
  public String toString() {
    return "Height: " + this.height + " Width: " + this.width;
  }
}
```

Rectangle Specification

We will consider the following specification for `Rectangle` class.

> *If the most recent call to* `setWidth` *is* `r.setWidth(w)` *and the most recent call to* `setHeight` *is* `r.setHeight(h)`, *then* `r.getArea()` *must return* `w * h`.

1. Translate the specification into *test cases*. You may omit the call to `setWidth` and `setHeight` as they are not yet created. Write the test using the following template (*or simply make it a* `.jsh` *file*)

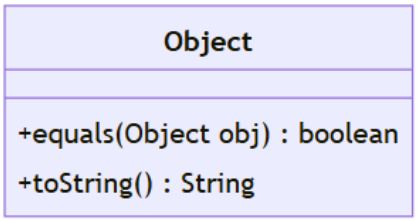
   ```
   System.out.println("<expr>");
   System.out.println(<expr> == <expected value>);
   ```

   This way, when you run the code, you will see a sequence of string following by either `true` or `false` and may quickly identify the problem.

   Test Cases

   ```
   Rectangle r = new Rectangle(     ,      );
   ```

2. Draw the class diagram for `Rectangle` using only the information you have so far.



Square Specification

We will consider the following specification for `Square` class.

> *A square must have all four sides to always be of the same length*[1].

Question 1A

1. Understand the JShell example and answer the questions below.

```
jshell> new Square(5)
$.. ==> Height: 5 Width: 5
jshell> new Square(5).getArea()
$.. ==> 25
```

   a. What is the method signature for the constructor of `Square`?

   b. What methods do we need to *add* to `Square`?

2. Add the `Square` class to the class diagram above following a *good design* after answering the questions above[2].

3. Try the following codes. Can they compile? If not, what is/are the error messages you get?

```
public class Square
    extends Rectangle {



}
```

```
public class Square
    extends Rectangle {
  public Square(int length) {

  }
}
```

4. Can you deduce what are the *implicit* rules for constructors and `super(..)`?

5. Add new test cases to test for `Square`.

   Test Cases

```
Rectangle r = new
```

6. Does any test cases fail?

7. Does it satisfy Liskov's substitution principle?

---

[1] Implicitly we assume that the angle must also be right-angle otherwise we have a parallelogram.

[2] Note that at this point, you should not write any code yet. This is only the design phase.

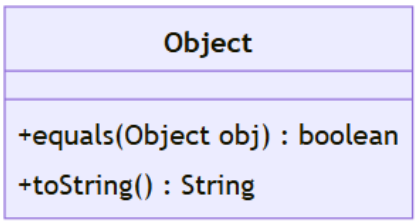
Question 1B

Add the following two methods into `Rectangle` class.

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

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

1. Draw the class diagram for `Rectangle` and `Square` using only the information you have so far.



2. Consider the following test case.

   Test Cases

```
Rectangle r = new Rectangle(5, 5);
System.out.println("Rectangle r = new Rectangle(5, 5);");
r.setHeight(5);
r.setWidth(9);
System.out.println("r.setHeight(5);");
System.out.println("r.setWidth(9);");
System.out.println("r.getArea();");
System.out.println(r.getArea() == 45);
```

   Modify the test case to use `Square`.

   a. Does any test cases fail?

   b. Does it satisfy Liskov's substitution principle? Any specification violated? If so, add new test cases to show this.

      Test Cases

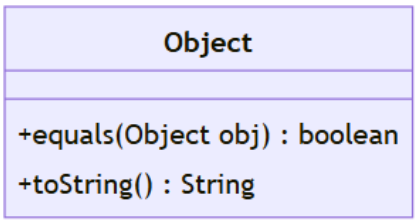
Question 1C

Implement two overriding methods in the `Square` class.

```
@Override
public void setWidth(int width) {
  super.setHeight(width);
  super.setWidth(width);
}
```

```
@Override
public void setHeight(int height) {
  super.setHeight(height);
  super.setWidth(height);
}
```

1. Draw the class diagram for `Rectangle` and `Square` using only the information you have so far.

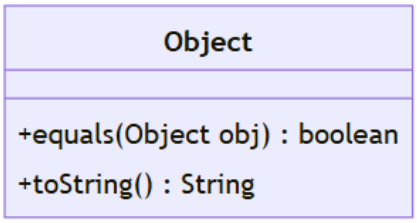


2. Consider the test cases we have so far. Run them.
    a. Does any test cases fail?

    b. Does it satisfy Liskov's substitution principle? Any specification violated?

Question 1D

1. Design a possible class diagram where `Rectangle` inherits from `Square`.



2. Can you design test cases that violates either the specification for
    a. Square

        A square must have all four sides to always be of the same length

    b. Rectangle

        If the most recent call to `setWidth` *is* `r.setWidth(w)` *and the most recent call to* `setHeight` *is* `r.setHeight(h)`*, then* `r.getArea()` *must return* `w * h`.

Question 2

Prepare the following before the class.

1. Print `R02_TypeCast.pdf` single-sided (***important, cannot be double-sided***)
2. For each compile-time type (*indicated on the top-right corner*), cut along the dashed line
    - You should have holes with half a smiley face on every compile-time type
3. Bring all to class.

Question 2A

Draw the class diagram for `Circle`, `Shape`, and `Printable`. We may ignore `Object` in this case. When extending (or implementing) an interface, use dashed line with arrow to connect to interface.

Question 2B

Try compiling the following implementation of `Circle`. Compile together with the two interfaces `Shape` and `Printable`.

```
abstract class Shape {
  public abstract double getArea();
}
abstract class Printable {
  public abstract void print();
}
class Circle extends Shape, Printable {
}
```

Does it compile? If not, what's the problem?

Question 2C

Try compiling the following implementation of `Circle`. Compile together with the two interfaces `Shape` and `Printable`.

```
interface PrintableShape extends Shape, Printable {
}
class Circle implements PrintableShape {
}
```

Does it compile? If not, what's the problem?

Question 2D

Consider the following imaginary class diagram on the left. What is/are the potential problem? What about using interface on the right? Is there any problem?