Homework 1: Classes
Due: 5:00 PM, Feb 9, 2018

Contents

1 Shape-ology 2
1.1 Rectangle: Take 1 ................................................................. 3
1.2 Triangle ........................................................................ 4
1.3 Square: Take 1 ................................................................. 4
1.4 Quadrilaterals ................................................................... 5
1.5 Rectangle and Square: Take 2 ........................................... 5

2 Rational Numbers 6

3 Appendix 9

Objectives

By the end of this homework, you will know:

• how different classes can interact with one another through inheritance

By the end of this homework, you will be able to:

• write toString and equals methods
• implement a basic class heirarchy

How to Hand In

For this (and all) homework assignments, you should hand in answers for all the non-practice questions. For this homework specifically, this entails answering the Shape-ology and Rational Numbers questions.

In order to hand in your solutions to these problems, they must be stored in appropriately-named files with the appropriate package header in an appropriately-named directory. The source code files should comprise the hw01.src package, and your solution code files, the hw01.sol package.

Begin by copying the source code from the course directory to your own personal directory. That is, copy the following files from /course/cs0180/src/hw01/src/*.java to ~/course/cs018/workspace/javaproject/src/hw01/src:
• IShape.java
• Posn.java
• Line.java

Do not alter these files!

After completing this assignment, the following solution files should be in your 
˜/course/cs0180/workspace/javaproject/sol/hw01/sol directory:

• Shape-ology
  – BasicRectangle.java containing public class BasicRectangle, which implements IShape
  – BasicRectangleTest.java containing public class BasicRectangleTest
  – Triangle.java containing public class Triangle, which implements IShape
  – TriangleTest.java containing public class TriangleTest
  – BasicSquare.java containing public class BasicSquare, which extends BasicRectangle
    – BasicSquareTest.java containing public class BasicSquareTest
  – Quadrilateral.java containing public abstract class Quadrilateral, which implements IShape
    – Parallelogram.java containing public class Parallelogram, which extends Quadrilateral
      – ParallelogramTest.java containing public class ParallelogramTest
    – Trapezoid.java containing public class Trapezoid, which extends Quadrilateral
      – TrapezoidTest.java containing public class TrapezoidTest
    – Rectangle.java containing public class Rectangle, which extends Parallelogram
      – RectangleTest.java containing public class RectangleTest
    – Square.java containing public class Square, which extends Rectangle
      – SquareTest.java containing public class SquareTest
• Rational Numbers
  – Rational.java containing public class Rational
  – RationalTest.java containing public class RationalTest
  – RationalCalculator.java containing public class RationalCalculator
    – RationalCalculatorTest.java containing public class RationalCalculatorTest

To hand in your files, navigate to the ˜/course/cs0180/workspace/javaproject/ directory, and run the command ‘cs018_handin hw01’. This will automatically hand in all of the above files. Once you have handed in your homework, you should receive an email, more or less immediately, confirming that fact. If you don’t receive this email, try handing in again, or ask the TAs what went wrong.
Problems

1 Shape-ology

The USA has been peaceful lately, so Captain America has decided to catch up on the math he couldn’t learn when he was frozen for one hundred years. Unfortunately, he is extremely behind on geometry, so he wants to learn more about shapes. Thankfully, he knows Java well enough that seeing some code might help solidify his basic geometry.

The shapes you will be implementing, Rectangle, Triangle, and Square, make use of Posn and Line classes, which are very similar to the posn and line structures of CS 17. Java versions of these classes are provided for you in the /course/cs0180/src/hw01/src/ directory. Begin by copying /course/cs0180/src/hw01/src/Line.java and /course/cs0180/src/hw01/src/Posn.java to your ~/course/cs0180/workspace/javaproject/src/hw01/src directory. Any of your solution files which reference these classes must import them like this:

```java
import hw01.src.Posn;
import hw01.src.Line;
import hw01.src.IShape;
```

In order to educate Captain America on the nature of all shapes, it may be useful to demonstrate some properties that all shapes have. For example, the area and perimeter of each shape can be calculated, albeit in slightly different ways. Java allows us to abstract out this shared quality as an interface.

**Task:** We have provided an IShape interface that includes two methods: getArea and getPerimeter. Copy /course/cs0180/src/hw01/src/IShape.java to your ~/course/cs0180/workspace/javaproject/src/hw01/src directory.

As you proceed to work on this problem, make sure that all your classes implement this interface. This is a way of creating requirements for yourself, which is always good programming practice.

1.1 Rectangle: Take 1

Captain America has decided he wants to focus first on rectangles, the second best shape after stars. He would like your help in implementing a basic rectangle and the associated methods, so that he may completely grasp how rectangles work.

**Task:** A rectangle class could be characterized by its width and height, but a rectangle can also be described as four Lines. Write a BasicRectangle class, whose internal representation of rectangles uses four lines, and whose constructor looks like this:

```java
public BasicRectangle(Posn bottomLeft, double width, double height) {
    // your code here
}
```

Your BasicRectangle class should implement the IShape interface. As such, it must include the method getArea, which returns the area of the rectangle, and the method getPerimeter, which returns the perimeter of the rectangle.
Note: You may assume that the values of width and height are positive.

Hint: Your object’s area and perimeter may be requested multiple times, but these values only need to be calculated once. Be sure to cache accordingly.

Almost all classes you ever write (and certainly all the classes for this problem) should have their own toString method.

Task: Add a simple toString method to your BasicRectangle class, which returns a string identifying your shape as a rectangle, as well as the particulars that characterize each rectangle.

It would also be nice to check whether two instances of BasicRectangle are equal. However, the default equals method inherited from Object checks only for shallow equality.

Task: Override the equals method to check for deep equality.

Note: Like toString, almost all classes you ever write (and certainly all the classes for this problem) should have their own equals method.

Hint: We provide sample toString and equals methods for you in the Line and Posn classes. Follow our model in the corresponding methods you write for your classes.

1.2 Triangle

Now that Captain America has rectangles all figured out, your next job is to write a class for him that represents a triangle.

Task: Write a Triangle class that implements IShape. Be sure to include toString and equals methods with appropriate behavior.

The Triangle constructor should take as input three Posns, but it should then use three Lines to represent a triangle.

Hint: You may assume that the three Posns input to your constructor are all distinct so that they form a geometrically valid triangle.

Note: If a = new Posn(0, 0), b = new Posn(0, 1), and c = new Posn(1, 0), then new Triangle(a, b, c), new Triangle(a, c, b), new Triangle(b, a, c), new Triangle(b, c, a), new Triangle(c, a, b), and new Triangle(c, b, a) all describe the same triangles. Be sure that your Triangle class you implement accurately reflects these identities!

Another Hint: Heron’s formula can be used to compute the area $A$ of a triangle:

$$A = \sqrt{s(s-x)(s-y)(s-z)}$$

Here $x$, $y$, and $z$ are the side lengths of the triangle, and $s$ is half its perimeter. Feel free to make use of Java’s Math library, which can be called like Math.sqrt(a), without any import statement.

1.3 Square: Take 1

Whenever one concept is a special case of another, we can define the former as a subclass (i.e., subtype) of the latter. By so doing, the subclass can inherit all the data and functionality of the
superclass. This way, you can implement additional features in the subclass, but do not need to rewrite any features that it inherits from its superclass.

**Task:** Write a `BasicSquare` class (including `toString` and `equals` methods), whose constructor looks like:

```java
public BasicSquare(Posn bottomLeft, double sideLength) {
    // your code here
}
```

Write your constructor in as few lines as possible (i.e., only 1, in addition to the header).

**Hint:** A square is a special case of a rectangle: use inheritance!

**Note:** You may assume that the value of `sideLength` is positive.

### 1.4 Quadrilaterals

Your next task is to create two more classes, `Parallelogram` and `Trapezoid`, which also implement the `IShape` interface. Because Captain America’s brain is pretty frozen by now, these classes need only to represent shapes with two sides parallel to the *x*-axis. Moreover, you may assume that any inputs to your constructors create valid parallelograms and trapezoids (i.e., they always have non-negligible areas and perimeters).

When writing these two new classes, you may find yourself copying a lot of code from one to another. Whenever you find yourself reimplementing the same data and/or functionality in more than one place, it’s a good idea to step back and re-think your design. More specifically, you should consider abstracting out common data and functionality into a single place. Often, this requires changing your original design, but the end result is usually cleaner, easier to read, and most importantly, much easier to maintain.

What do parallelograms and trapezoids have in common? Like rectangles, triangles, and squares, they share area and perimeter calculators. But they share so much more, because they are both quadrilaterals! Consequently, their data representations may be similar.

**Task:** Write a `Quadrilateral` abstract class which implements the `IShape` interface.

**Hint:** Think about what attributes and behaviors all `Quadrilaterals` share. Do you need this full generality to implement parallelograms and trapezoids? Figure out exactly what you do need, and then implement an abstract `Quadrilateral` class.

**Task:** Write a `Parallelogram` class whose constructor looks like:

```java
public Parallelogram(Posn bottomLeft, Posn topLeft, double bottomLength) {
    // your code here
}
```

**Task:** Write a `Trapezoid` class whose constructor looks like:

```java
public Trapezoid(Posn bottomLeft, Posn topLeft, double bottomBaseLength, double topBaseLength) {
    // your code here
}
```
**Hint:** Can your Parallelogram and Trapezoid classes share code? If so, they should both be fairly short, because they can extend a common class—Quadrilateral.

1.5 Rectangle and Square: Take 2

Just as a square is a special case of a rectangle, a rectangle is a special case of a ... what?

**Task:** Write new Rectangle and Square classes that fit into your new, expanded class structure with the following constructors, which mimic those of your BasicRectangle and BasicSquare:

**Note:** Please do not change your BasicRectangle and BasicSquare classes - write new ones!

```java
public Rectangle(Posn bottomLeft, double width, double height) {
    // your code here
}

public Square(Posn bottomLeft, double sidelength) {
    // your code here
}
```

**Note:** Be sure not to modify your BasicRectangle and BasicSquare classes! You need to hand in both versions.

2 Rational Numbers

A rational number is a fraction in which both the numerator and denominator are integers. Among other operations, rational numbers can be added and multiplied together. In this problem, you will write a class that represents rational numbers that allows you to do simple arithmetic like this with them. After this you will use your Rational class to make a basic physics calculation very simple.

**Task:** Create a Rational class with two fields:

```java
int numerator;
int denominator;
```

**Task:** Create a Rational constructor that takes as input two integers, n and d, corresponding to this.numerator and this.denominator.

**Note:** For now, you do not have to worry about division by zero. Soon you will learn how to handle edge cases like this. In the future you will be expected to properly handle them.

This design is too simple because it allows too many representations of each rational number. For example, the rational number 2 can be represented by new Rational(2, 1), new Rational(4, 2), and new Rational(100, 50).

**Task:** Modify your code so that rational numbers are represented in lowest terms. That is, that the greatest common divisor (gcd) of the numerator and denominator must be 1.

**Hint:** To find the GCD of two numbers you can use Euclid’s method:
\[ \text{gcd}(a, b) = \begin{cases}  a & \text{if } b = 0 \\  \text{gcd}(b, a \mod b) & \text{otherwise} \end{cases} \]

**Note:** This method of finding the GCD is not the same one you implemented in CS 17, as this method works with both positive and negative (non-zero) integers.

**Task:** Write a `toString` method for the `Rational` class.

**Task:** Override the `equals` method to test for deep equality.

**Hint:** Don’t forget that `equals` takes as input an argument of type `Object`!

**Note:** Throughout, don’t forget the `Override` annotation!

**Task:** Test all of your methods in a `RationalTest` class.

**Task:** Create a new document that holds a new class called `RationalCalculator`.

**Task:** Create a method:

```
Rational forceCalculator(Rational m1, Rational m2, Rational a);
```

The `forceCalculator` method takes three arguments - two masses, \( m_1 \) and \( m_2 \), and one acceleration, \( a \). For this problem, use the formula below to calculate the total force, \( F \):

\[
F = (m_1 + m_2) \times a
\]

At first glance, you may have thought that a version of the `forceCalculator` akin to the one below was the best way to calculate the force.

```
Rational forceCalculator(Rational m1, Rational m2, Rational a){
    int massNumer = (m1.denominator * m2.numerator) + (m1.numerator * m2.denominator);
    int massDenom = (m1.denominator * m2.denominator);
    Rational totalMass = new Rational(massNumer, massDenom);
    int forceNumer = totalMass.numerator * a.numerator;
    int forceDenom = totalMass.denominator * a.denominator;
    Rational force = new Rational(forceNumer, forceDenom);
    return force;
}
```

We are happy to inform you that there is another, more succinct, way to do this by using methods contained within the `Rational` class!

**Task:** Add the following methods to your `Rational` class.

```
Rational plus(Rational toAdd);
Rational times(Rational toMult);
```

**Task:** Write the methods `plus` and `times`. The `plus` method takes as an input a `Rational`, `toAdd`, and adds it to this, returning the result as a `new` `Rational`. The `times` method behaves analogously, except that it performs multiplication.

Below is an example of `plus` being used:
Rational r = new Rational(1, 4);
Rational t = new Rational(1, 2);
return r.plus(t);  // returns a new Rational(3, 4). r and t do not change

Task: Add tests for plus and times to your RationalTest class.

Now that you have a tested, working Rational class, you can create a Rational object that can call the plus and times methods.

Task: Implement the forceCalculator method more succinctly, using the Rational plus and times methods.

Note: You do not need to account for negative masses, but you must account for negative accelerations.

Task: Test the forceCalculator method in the RationalCalculatorTest class.
3 Appendix

Here is a class diagram that depicts the structure that the source files and your solution files should have after you finish implementing Shapes:

```
public class Square
    public Square(Posn bottomLeft, double sideLength);
    public String toString();
    public boolean equals(Object o);

public class Rectangle
    public Rectangle(Posn bottomLeft, double width, double height);
    public String toString();
    public boolean equals(Object o);

public class Trapezoid
    public Trapezoid(Posn bottomLeft, Posn topLeft, double bottomBaseLength, double topBaseLength);
    public String toString();
    public boolean equals(Object o);

public class Parallelogram
    public Parallelogram(Posn bottomLeft, Posn topLeft, double bottomLength);
    public String toString();
    public boolean equals(Object o);

public class BasicSquare
    public BasicSquare(Posn bottomLeft, double sideLength);
    public String toString();
    public boolean equals(Object o);

public class BasicRectangle
    public BasicRectangle(Posn bottomLeft, double width, double height);
    public String toString();
    public boolean equals(Object o);

public interface IShape
    public double getArea();
    public double getPerimeter();
```

---

8

---

9
Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS18 document by filling out the anonymous feedback form: http://cs.brown.edu/courses/cs018/feedback