Lab 0: Introduction to Java Programming

Problem 1: Fizz Razz Buzz

Write a program called Fizz.java that loops over each number from 1 to 100, and does the following:

- If the number is a multiple of 3, print “Fizz”
- If the number is a multiple of 4, print “Razz”
- If the number is a multiple of 5, print “Buzz”
- If the number is a multiple of 3 and 4, print “FizzRazz”
- If the number is a multiple of 4 and 5, print “RazzBuzz”
- If the number is a multiple of 3 and 5, print “FizzBuzz”
- If the number is a multiple of 3, 4 and 5, print “FizzRazzBuzz”
- Otherwise just print the number

Problem 2: Rock Paper Scissors

This is another problem inspired by children’s games. Both players choose one of “Rock”, “Paper”, and “Scissors” to play against each other. Rock beats Scissors, Scissors beats Paper, and Paper beats Rock. Write a Java program called RPS.java that plays Rock Paper Scissors against the user in a while loop

Problem 3: Guessing Game

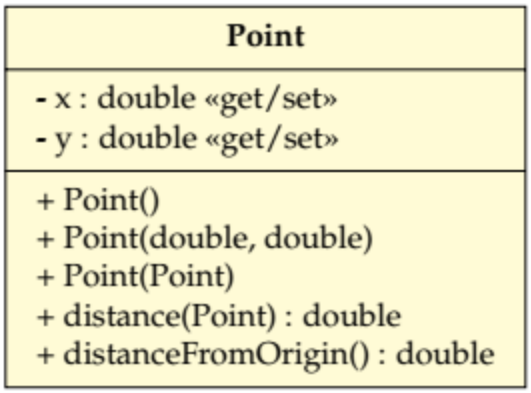
Write a program called Guess.java, that implements the guessing game, “I am thinking of a number between x and y.”

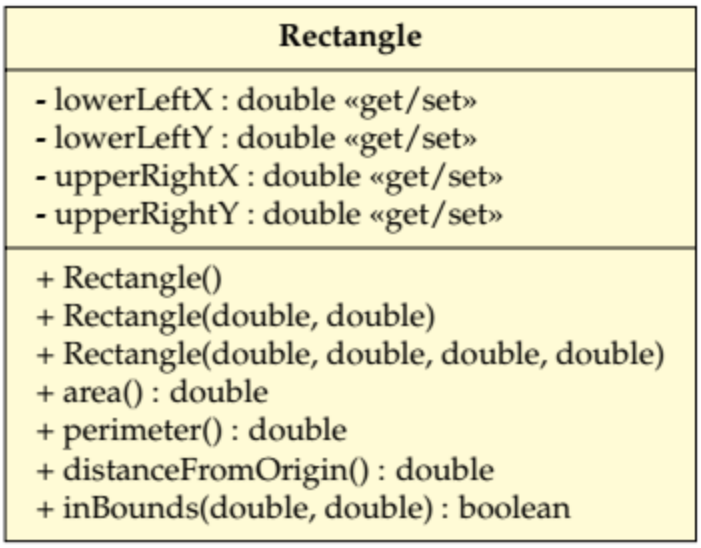
Use a Scanner to prompt the user for the lower and upper bounds of the number they’re thinking of, and correct the error if y < x (i.e. swap the numbers). Use a binary search algorithm to iteratively guess what number they are thinking of. On each guess, prompt the user to see whether your guess is less than, greater than, or equal to their number. When the user responds that the guess was correct, print the number of guesses the program made and then exit.

Lab 1: Working with Objects

Problem 1: Geometry

Define classes: Point and Rectangle based on the UML diagrams:





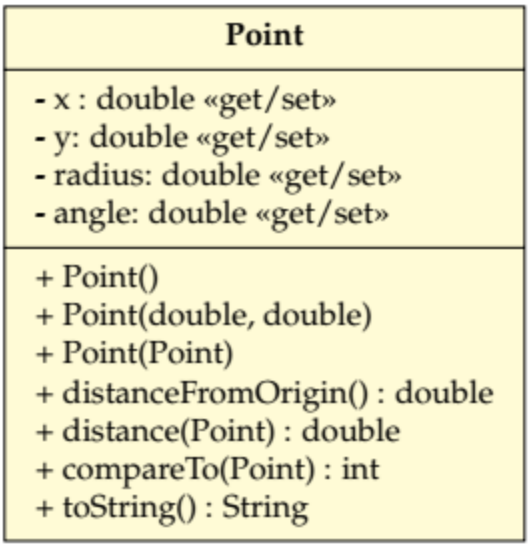
Create the Circle class based on the following skeleton code and draw the UML diagram

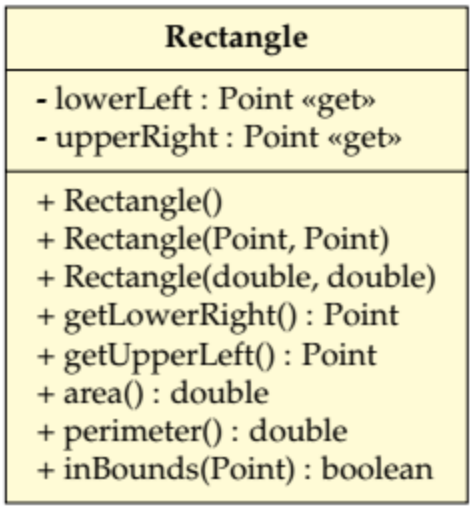
```
1  public class Circle {
2      private double centerX;
3      private double centerY;
4      private double radius;
5
6      public Circle() { ... }
7      public Circle(double x, double y, double radius) { ... }
8
9      // getters and setters go here
10
11     public double area() { ... }
12     public double perimeter() { ... }
13     public double distanceFromOrigin() { ... }
14     public double inBounds() { ... }
15 }
```

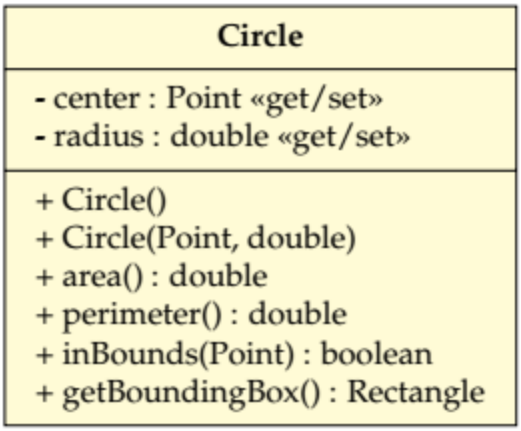
Lab 2: Encapsulation and Reusing Code

Problem 1: Geometry

Implement the Point, Rectangle, and Circle based on the UML diagrams







Problem 2: Temperature

Implement the Temperature class based on the UML diagram:

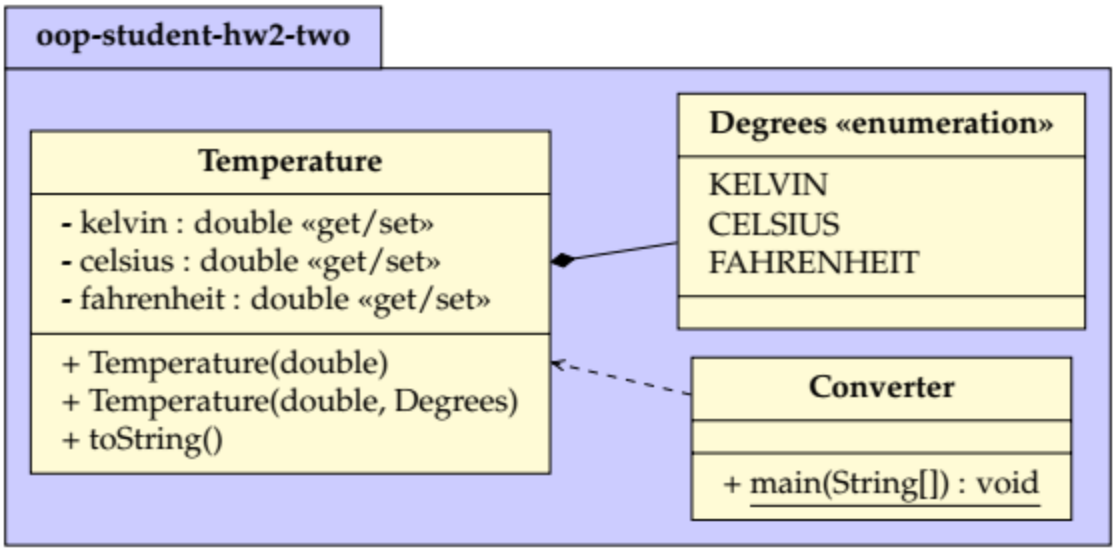
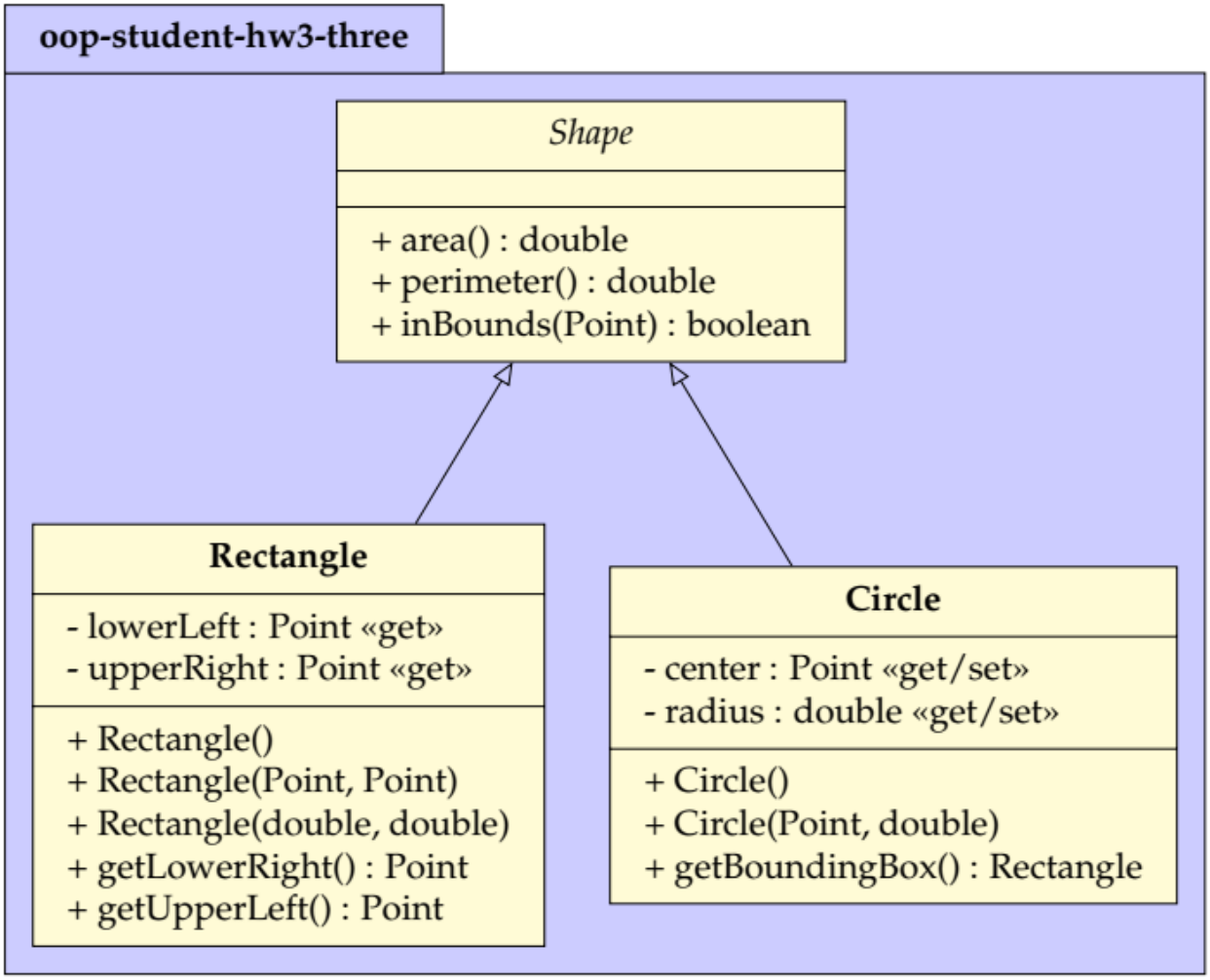


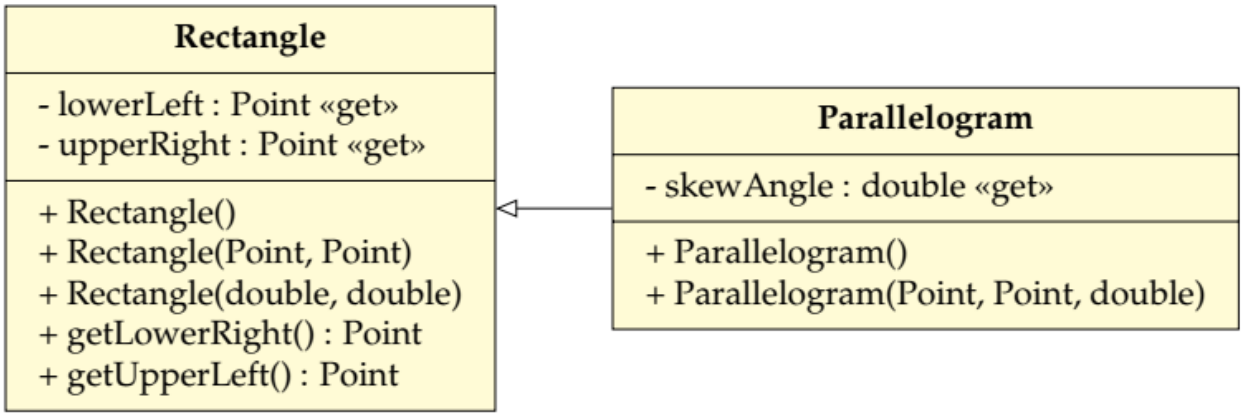
Figure 8: Package diagram – the underlining on main means it is declared `static`

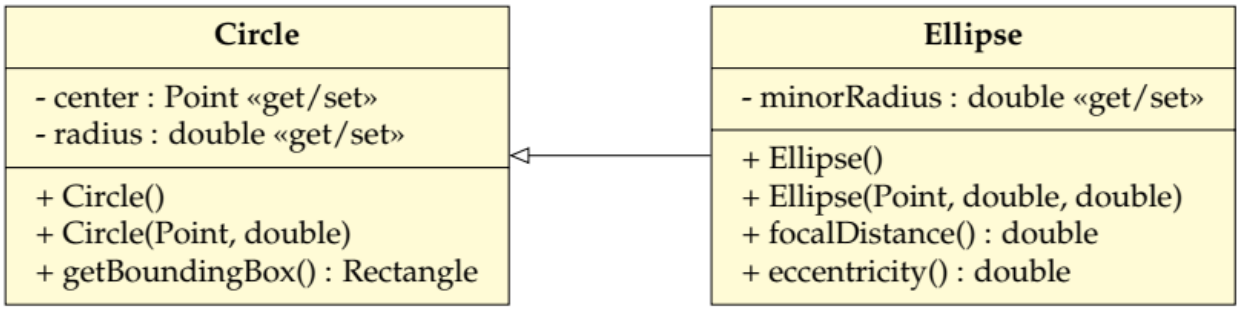
Lab 3: Inheritance

Problem 3: More Geometry

Implement the following classes:


oop-student-hw3-three
Shape
+ area() : double
+ perimeter() : double
+ inBounds(Point) : boolean
Rectangle
- lowerLeft : Point «get»
- upperRight : Point «get»
+ Rectangle()
+ Rectangle(Point, Point)
+ Rectangle(double, double)
+ getLowerRight() : Point
+ getUpperLeft() : Point
Circle
- center : Point «get/set»
- radius : double «get/set»
+ Circle()
+ Circle(Point, double)
+ getBoundingBox() : Rectangle


Rectangle
- lowerLeft : Point «get»
- upperRight : Point «get»
+ Rectangle()
+ Rectangle(Point, Point)
+ Rectangle(double, double)
+ getLowerRight() : Point
+ getUpperLeft() : Point
Parallelogram
- skewAngle : double «get»
+ Parallelogram()
+ Parallelogram(Point, Point, double)

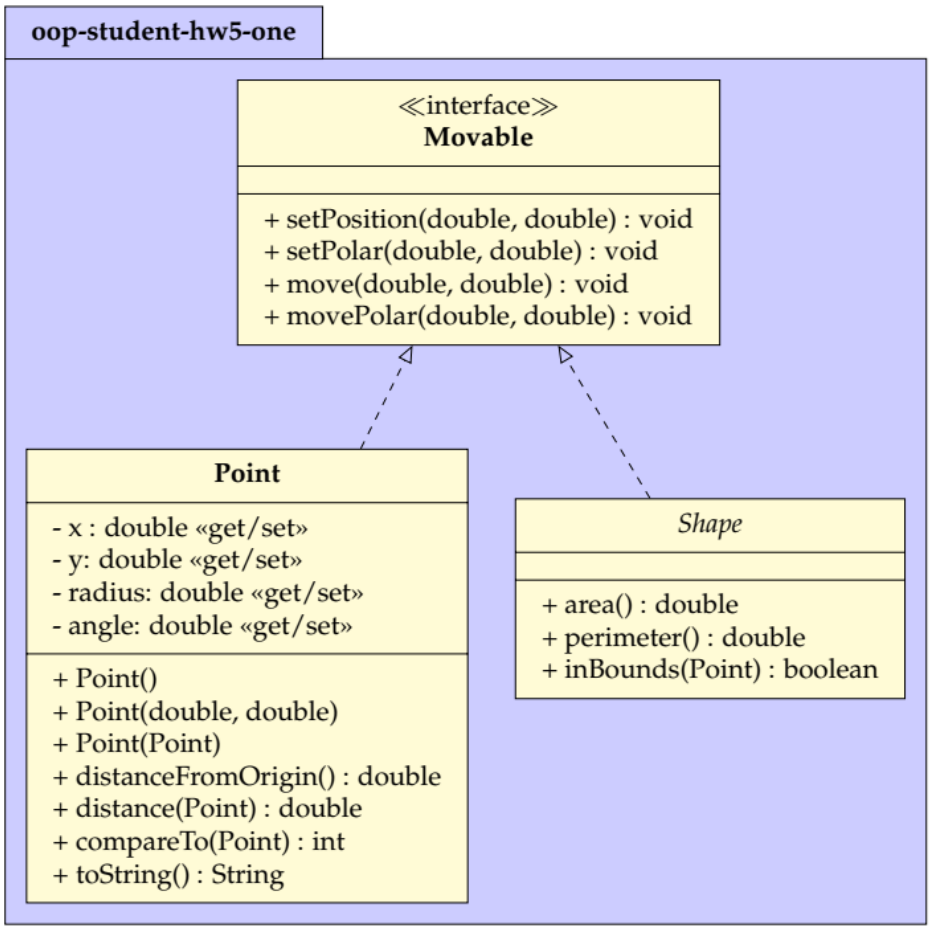


Lab 5: Interfaces

Problem 1: Even More Geometry!

Implement the following interfaces

Movable



Cloning

Also add the interface Cloneable to both Point and Shape. Since Point contains only primitive types, it does not need a custom implementation of clone().

For both Rectangle.java and Circle.java, write the clone() method so that it does a deep copy. When you clone a shape object, it should also clone new copies of any Points within that object. This way moving a point in the new copy will not change the position of the original shape.

Comparator

Add two new classes, ShapeAreaComparator.java and ShapePerimeterComparator.java, both of which implement the interface Comparator<Shape>.

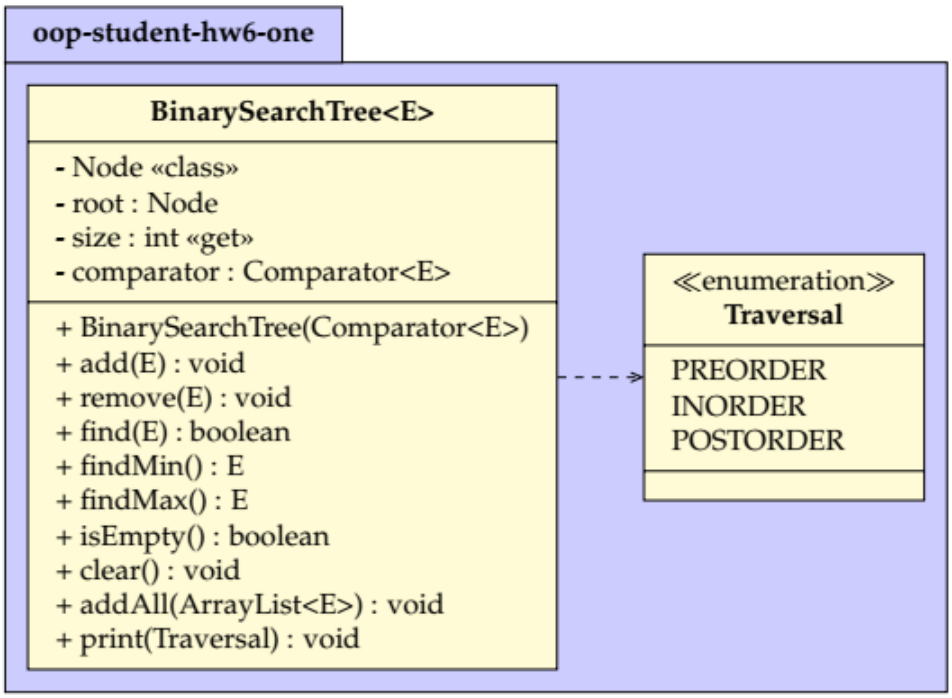
The first class implements a compare() method that takes two Shapes, and returns -1, 0, or 1 if the area() of the first shape is less than, equal to, or greater than that of the second shape, (respectively).

The second class does the same thing, but compares the shapes based on their perimeter() instead of their area().

Lab 6: Generic Classes and Data Structures

Problem 1: Binary Search Tree

For this problem, you will write a Binary Search Tree to store a sorted data. Your class should follow this template:



Here, Node is an inner class which is managed by the outer BinarySearchTree class. It can contain its own and methods to help the outer class insert, search for, and remove elements from the tree:

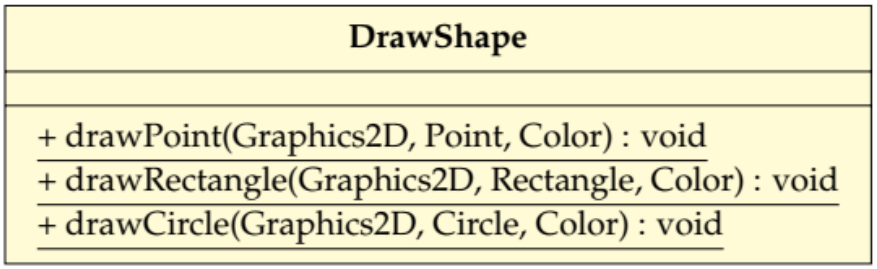
```
private class Node {
    private E data;
    private Node left;
    private Node right;

    private Node(E data, Node left, Node right);
    private Node add(E data);
    private Node find(E data);
    private Node remove(E data, Node parent);
    ...
}
```

Lab 7: Graphics Programming

Problem 1: Draw Shape

Create a small helper class, DrawShape.java, to draw shapes onto a Graphics2D surface. You will use this code in the next problems. You will have to import classes from your geometry package, last seen in Homework 5. The class contains three static methods:



drawPoint() takes a Point and draws a point at that position on the Graphics2D object. Draw the point using the given color, but set the color back to its original value afterward (save a copy of the original color before you start drawing).

Note: The builtin Graphics2D object doesn't have a drawPoint() method. Instead, you might use drawLine() to draw a line of length 1, or drawOval() to draw a circle with a radius of 1 – the implementation is up to you.

drawRectangle() takes a Rectangle and uses the Graphics2D's drawRect() method to draw its outline to the surface. Draw the rectangle using the given order, but set the color back to its original value after drawing it.

Note: The arguments to drawRect() are (x, y, width, height), not (x1, y1, x2, y2). drawCircle() takes a Circle and uses the Graphics2D's fillOval() method to draw the circle in the given color. Draw the rectangle using the given order, but set the color back to its original value after drawing it.

Note: The fillOval() method takes arguments (x, y, width, height), but x and y represent the lower left corner of the oval's bounding box, not the center!

Lab 8: Interaction and Event Handling

Problem 1: Clock Timer

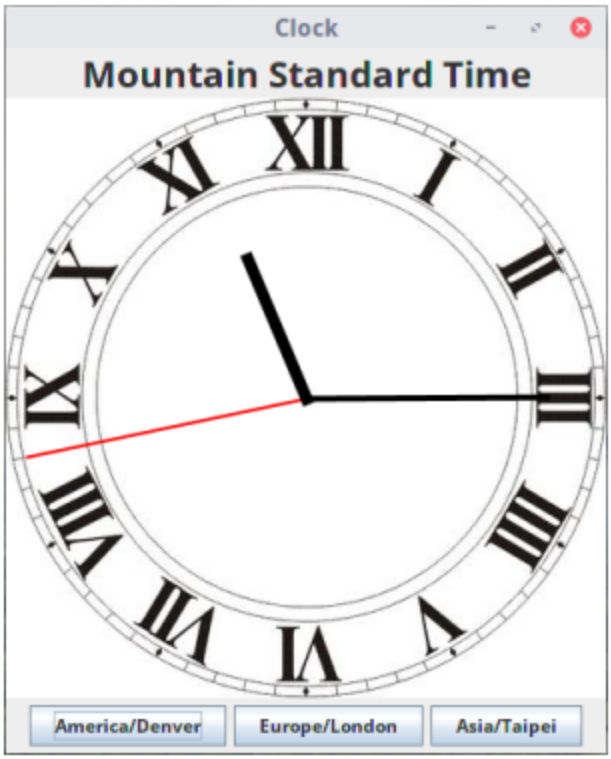


Figure 2: The finished product should look similar to this

The clock timer is composed with ClockFace and ClockFrame classes:

ClockFace
- hour : int «get/set» - minute : int «get/set» - second : int «get/set»
+ ClockFace() + ClockFace(int, int, int) + tick() : void + setTimeZone(TimeZone) : void + paintComponent(Graphics) : void

ClockFrame
- clock : ClockFace - tzLabel : JLabel
+ ClockFrame() + main(String[]) : void

ClockFace

ClockFace.java is the same as in the previous assignment, except for the addition of two new methods.

tick() moves the time forward by one second, then calls repaint(). Increment second, but if the result is 60 set it to 0 and increment minute. If the minute is then 60, set it to 0 and increment hour. And when hour gets to 24, wrap it around to 0.

setTimeZone() takes a timezone object, (from java.util.TimeZone), and sets the hour, minute, and second to the current local time in that time zone. To do that, you can pass the timezone to TimeZone.setDefault(), then get the time again from LocalTime.now(). Use TimeZone.getDefault() to save the system timezone before overwriting it, and set it back to its original value before exiting the method. Call repaint() after setting the time.

ClockFrame

ClockFrame inherits from JFrame. As in the previous version, it adds a ClockFace to the main window, and a JLabel containing the default timezone to the top of the frame. The private attributes clock and tzLabel should store these two components.

The next step is to create a new Timer object, to call clock.tick() once every second (1000 ms). The first argument to the Timer constructor is the delay, in milliseconds, to wait between events. The second argument is an instance of ActionListener – the code in its actionPerformed() method will run every time the timer goes off.

You can set up the ActionListener by adding a new inner class to the project. Alternatively, you can use an anonymous class 1 to instantiate an instance of the interface on the fly.

Add a new JPanel to the SOUTH of the ClockFrame, and add three JButtons to the panel. Each button will correspond to a different timezone ID (like "America/Denver" or "Europe/London"). You can choose any three timezone IDs you want, as long as they’re valid. The buttons should each have an ActionListener, to do two things when the button is clicked:

1. Set the timezone of the clock to the timezone of the button
2. Set the text of the tzLabel to the display name of the button’s time zone