**CS102 – Algorithms and Programming II**

**Lab Programming Assignment 3**

**Fall 2021**

|  |
| --- |
| **ATTENTION:**   * Compress all of the Java program source files (.java) files into a single zip file. * The name of the zip file should follow the below convention:   **CS102\_Sec1\_Asgn3\_YourSurname\_YourName.zip**   * Replace the variables “YourSurname” and “YourName” with your actual surname and name. * You may ask questions on Moodle. * Complete most of the assignment before the lab session and upload the above zip file to Moodle by the deadline (otherwise, significant points will be taken off). You will get a chance to update and improve your solution by consulting to the TA during the lab. You will resubmit your code once you demo your work to the TA. * Deadline is at **02 November 2021, Tuesday 23:55**. * You have to come to lab session on 03 November 2021, Wednesday to demonstrate your work. |

**[Part 1: 20 pt.]**

In this part, you are going to create a class hierarchy for shapes using your inheritance, abstract class and interface knowledge. The underlined parts are optional and just there for you to test your program. Still, read those parts to learn more about what you need to do. You can reuse some of the code that you prepared in Lab 01. You will use the classes you have completed here in the next part.

1. Design a class hierarchy to include classes *Shape*, *Rectangle* (with *int* sides, *width* and *height*), *Circle* (with *int radius* ) and *Square* (with *int side*). *Shape* should be abstract with method *double getArea()* and Square should inherit *getArea()* from Rectangle.
2. Create another class, *ShapeContainer*, to hold a set of shapes. It should have methods *void add( Shape s)* and *double getArea()* and *String toString()*. Write a *ShapeTester* class with a menu that allows the user to create an empty set of shapes (*ShapeContainer*), add as many circle and rectangle shapes to it as they wish, compute & print out the total surface area of the entire set of shapes, and print out information about all of the shapes in the container by calling the toString() method for each shape. Experiment. Try to predict what would happen when you (i) comment out the getArea() method of the Circle class, and (ii) also make the Circle class abstract, before finally (iii) creating an instance of the (now abstract) Circle class to add to the shapes collection. Test your predictions.
3. The customer is impressed with your work so far, and so asks you to extend the program. They want shapes to be locatable (i.e. to have an *x, y* location, and *getX()*, *getY()* and *setLocation( x, y)* methods). As a good designer you decide to first create a *Locatable* interface with these methods, then have the *Shape* class implement it. In this way all shapes automatically become locatable.
4. Impressed, the customer wants even more! This time they ask for shapes to be *Selectable*, so you again start by creating a Java interface, having *boolean getSelected()* and *setSelected( boolean)* and *Shape contains( int x, int y)*. Unfortunately, this time you are ***not allowed***to change the *Shape* class. Modify your other classes so that each shape added to the *ShapeContainer* is *Selectable*. Change the *toString()* methods of each shape class so they show whether the shape is selected or not. Add another option to your ShapeTester menu that allows the user to find the first Shape that contains a given x, y point and, afterwards, toggle its selected state. Provide another menu option that removes all selected shapes from the set of shapes. Good design practice suggests you should ask the *ShapeContainer* object to do the work of finding the first Shape containing the given point and of removing all selected shapes (rather than trying to do the work yourself in the ShapeTester class, which might/would require knowledge of the insides of the *ShapeContainer* class).

**[Part 2: 80 pt.]**

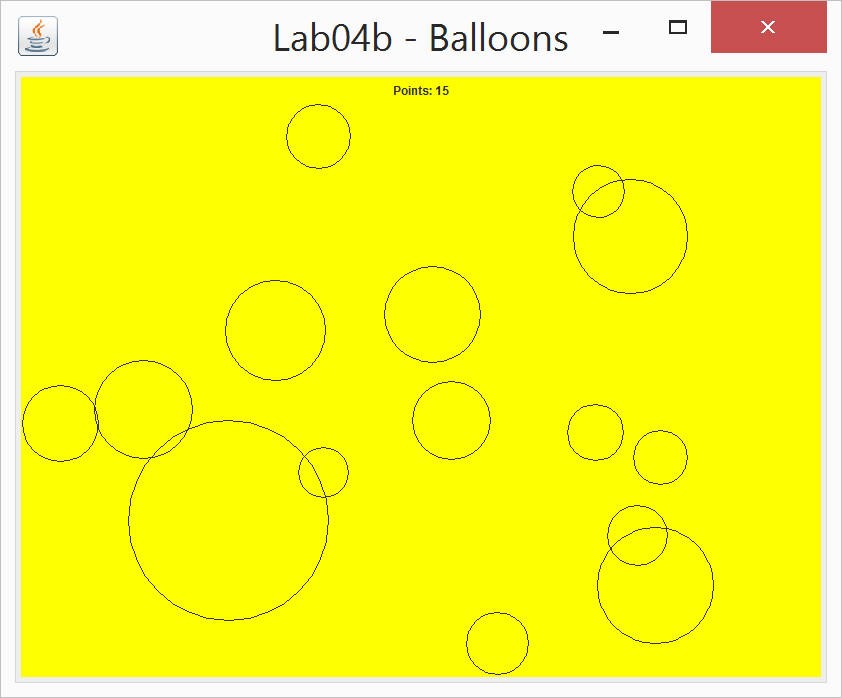
The Balloons game challenges players to burst as many balloons as they can in the time allowed. It is based on the Shape hierarchy you created in **Part 1**. You may be able to [download the jar file and play](https://web.archive.org/web/20180328095038/http:/www.cs.bilkent.edu.tr/~david/cs102/assignments/lab04/balloons_v2.jar) the basic game (assuming you have the right version of Java installed on your computer, the planets are appropriately aligned, etc., etc.), but if not, don't worry, the real fun is in programming it yourself.

1. From **Part 1**, you should already have a Shape class that implements Locatable, a Circle class that implements Selectable, and a ShapeContainer class that holds a collection of such selectable shape objects.
2. Before you start, create a new project folder for Lab04\_Part2, and copy your Shape hierarchy into it, making sure to put it in a package called shapes. To the package, add an interface Drawable, which has a single method, "void draw( Graphics g);". Add two new methods to ShapeContainer: int size() ~that returns the number of shapes in the container~ and int selectAllAt( int x, int y). ~that returns the number of shapes found at the point x,y and sets the selected property of those shapes to true. Note: your class should already have a method void removeSelected() ~that removes all selected shapes from the container~ and Shape contains( int x, int y) ~that returns the first shape that includes the point x, y, or null if none do~, however, it ***should not*** have a method, such as Shape get( int index), that returns the Shape at a specified index location in the collection.

From now on everything should be done in another package (possible the default one).

1. Create a normal application, called Lab04\_Part2, with a main method. Have it create a JFrame into which you can place the balloons game panel. Create a BalloonsGamePanel class and add an instance of it to your JFrame. In the BalloonsGamePanel class, have a ShapeContainer property, called balloons. Override the paintComponent method of the BalloonsGamePanel so that it calls the draw method of each shape in the container.
2. Next create a Balloon class that extends your Circle class and implements Drawable. Balloons should have a default radius of 25. Try adding an instance or two to your balloons ShapeContainer to ensure everything is in order! Add a grow method to your Balloon class, that increases the radius of the balloon (Circle) by a fixed number of units, up to a maximum size, say 100, at which point the circle should be selected and the radius set to zero.
3. Add a javax.swing.Timer object to your BalloonsGamePanel, and have it call the grow() method of each Balloon in balloons, every 250 milliseconds or so. Have the game constructor create say 25 balloons at random locations on the panel.
4. To simulate bursting a balloon with a pin, add a MouseListener to your BalloonsGamePanel and override the mousePressed method so that it calls the selectAllAt( x, y) method of balloons using the location of mouse pressed event. You will need to call removeSelected() to actually remove the balloons from the screen/collection and while this could be done in the mouse listener, it may be more conveniently placed in the timer object's listener.
5. Once all this is working, you simply need to add a few bells 'n whistles to make it into a reasonable game.
6. Add a points property to your class. Add points only if more than two balloons are burst with a single mouse press. Whenever there are less than say 15 balloons left, add another balloon at a random location ~do this in the timer's listener. Add a JLabel to the game panel and use it to display the points scored so far.
7. Add yet another property, elapsedTime, to keep track of how long the game has been running. Increment it in the timer's listener method, and then stop the game after a fixed period. Use a JOptionPane to inform the user the game is over. Perhaps allow the user to "Play again?" if they wish.
8. There are lots of ways you might improve this basic game... use your imagination and show us what you can do.







Have fun!

**IMPORTANT NOTES:**

1. Please comment your code according to the documentation and commenting conventions used in the textbook.