Assignment 6 Requirement

Late homework assignments will not be accepted, unless you have a valid written excuse (medical, etc.). You must do this assignment alone. No team work or "talking with your friends" will be accepted. No copying from the Internet. Cheating means zero.

Create a new Eclipse workspace named "**`Assignment6_1234567890`**" on the desktop of your computer (replace **`1234567890`** with your student ID number). For each question below, create a new project in that workspace. Call each project by its question number: "**`Question1`**", "**`Question2`**", etc. Answer all the questions below. At the end of the assignment, create a ZIP archive of the whole workspace folder. The resulting ZIP file must be called "**`Assignment6_1234567890.zip`**" (replace **`1234567890`** with your student ID number). Upload the ZIP file on iSpace.

Here are a few extra instructions:

- Do not forget to write tests for *all* the code of *all* the classes.
- Give meaningful names to your variables so we can easily know what each variable is used for in your program.
- Put comments in your code (in English!) to explain WHAT your code is doing and also to explain HOW your program is doing it.
- Make sure all your code is properly indented (formatted). Your code should be beautiful to read.

Failure to follow these instructions will result in you losing points.

Question 1

Write an **`IShape`** interface with the following UML specification:

```
+------------------------------------+
|           <<interface>>            |
|               IShape               |
+------------------------------------+
| + getX(): int                      |
| + getY(): int                      |
| + setX(int x): void                |
| + setY(int y): void                |
| + isVisible(int w, int h): boolean |
| + isIn(int x, int y): boolean      |
| + draw(Graphics g): void           |
+------------------------------------+
```

and a **`Shape`** class that implements **`IShape`** and has the following UML specification:

```
+------------------------------------+
|                Shape               |
+------------------------------------+
| - x: int                           |
| - y: int                           |
| - color: Color                     |
+------------------------------------+
| + Shape(int x, int y)              |
| + getX(): int                      |
| + getY(): int                      |
| + setX(int x): void                |
| + setY(int y): void                |
| + isVisible(int w, int h): boolean |
| + isIn(int x, int y): boolean      |
| + draw(Graphics g): void           |
| + testShape(): void                |
+------------------------------------+
```

The **x** and **y** instance variables indicate the position of the center of the shape, and the **color** instance variable indicates the color of the shape. The color of the shape is computed randomly in the constructor of the shape class and never changes after that, like this:

```
        color = new Color((float)Math.random(),
                          (float)Math.random(),
                          (float)Math.random());
```

The **isVisible** method is abstract, and indicates whether the shape is currently visible or not inside a window of width **w** and of height **h**. The **isIn** method is abstract, and indicates whether the point at coordinates (**x**, **y**) is currently inside the shape or not. The **draw** method simply changes the color of the graphics object **g** to be the correct color for the shape.

Also add to your program a **Start** class to test your **Shape** class.

```
public class Start {
      public static void main(String[] args) {
            Shape.testShape();
      }
}
```

Question 2

Add a class **Bubble** that extends **Shape**. The **Bubble** class has an instance variable called **radius** of type **double** that represents the radius of the bubble. The constructor of the **Bubble** class takes an **x** and a **y** as arguments, which represent the position of the new bubble. The radius of a new bubble is always `10` and never changes after that.

The **isVisible** method indicates whether the bubble is currently visible inside a window of width **w** and height **h** (position (`0`, `0`) is in the upper-left corner of the window). The bubble is considered visible if at least one pixel of the bubble is visible. Therefore a bubble might be visible even when its center is outside the window, as long as the edge of the bubble is still visible inside the window.

The code of the **isVisible** method is a little bit complex, mostly because of the case where the center of the circle is just outside one of the corners of the window. So here is the code of the **isVisible** method, which you can directly copy-paste into your assignment:

```
      // Find the point (wx, wy) inside the window which is closest to the
      // center (x, y) of the circle.  In other words, find the wx in the
      // interval [0, w - 1] which is closest to x, and find the wy in the
      // interval [0, h - 1] which is closest to y.
      // If the distance between (wx, wy) and (x, y) is less than the radius
      // of the circle (using Pythagoras's theorem) then at least part of
      // the circle is visible in the window.
      // Note: if the center of the circle is inside the window, then (wx, wy)
      // is the same as (x, y), and the distance is 0.
      public boolean isVisible(int w, int h) {
            double x = getX();
            double y = getY();
            double wx = (x < 0 ? 0 : (x > w - 1 ? w - 1 : x));
            double wy = (y < 0 ? 0 : (y > h - 1 ? h - 1 : y));
            double dx = wx - x;
            double dy = wy - y;
            return dx * dx + dy * dy <= radius * radius;
```

```
    }
```

The **isIn** method indicates whether the point at coordinates (**x**, **y**) (which are the arguments of the method) is currently inside the bubble or not. The edge of the bubble counts as being inside of the bubble. HINT: use Pythagoras's theorem to compute the distance from the center of the bubble to the point (**x**, **y**).

The **draw** method uses the graphics object **g** to draw the bubble. HINT: remember that the color of the graphics object **g** is changed in the **draw** method of the superclass of **Bubble**.

Also add a **testBubble** method to test all your methods (including inherited methods, but excluding the **isVisible** method, which I provide, and excluding the **draw** method since it requires as argument a graphics object **g** that you do not currently have).

Question 3

Write a **Model** class with the following UML specification:

```
+-----------------------------------------------+
|                    Model                      |
+-----------------------------------------------+
| - score: int                                  |
| - bubbles: ArrayList<IShape>                  |
+-----------------------------------------------+
| + Model()                                     |
| + getScore(): int                             |
| + addBubble(int w, int h): void               |
| + moveAll(int dx, int dy): void               |
| + clearInvisibles(int w, int h): void         |
| + deleteBubblesAtPoint(int x, int y): void    |
| + drawAll(Graphics g): void                   |
| + testModel(): void                           |
+-----------------------------------------------+
```

When a new model object is created, the score must be zero and the arraylist must be empty.

The **getScore** method returns as result the current score for the game.

The **addBubble** method adds a new bubble to the arraylist of bubbles. The position of the center of the new bubble is random but must be inside a window of width **w** and height **h** (the arguments of the **addBubble** method), like this:

```
new Bubble((int)(w * Math.random()), (int)(h * Math.random()))
```

The **moveAll** method moves the positions of all the bubbles in the arraylist of bubbles by the amount **dx** in the **x** direction and by the amount **dy** in the **y** direction.

The **clearInvisibles** method takes as argument the width **w** and the height **h** of the window, and deletes from the arraylist of bubbles any bubble which is not visible in the window anymore. For each bubble which is deleted, the **score** decreases by **1**.

WARNING: when you use the **remove** method of Java's **ArrayList** class to remove an element of an arraylist at index **i**, the arraylist immediately shifts down by one position all the elements with higher indexes to make the arraylist one element shorter. So, for example, when removing the element at index **i**, the element at index **i+1** immediately moves to the position at index **i**, the element at index **i+2** immediately moves to the position at index **i+1**, etc. This means that on the next iteration of the loop, when **i** has become **i+1**, the element that you will be testing at index **i+1** is in fact the element that used to be at index **i+2**. Which means that the element that used to be at index **i+1** (and which is now at index **i**) will never be tested! Therefore, when removing elements from an arraylist, if your loop starts at index **0** and goes up the indexes in the arraylist, then your loop will fail to test some elements! CONCLUSION: when removing elements from an arraylist, your loop must start from the END of the arraylist and go DOWN to index **0**.

The **deleteBubblesAtPoint** method takes as argument the coordinates (**x**, **y**) of a point, and deletes from the arraylist of bubbles any bubble which contains this point (multiple bubbles might contain the point, because bubbles can overlap in the window). For each bubble which is deleted, the **score** increases by **1**.

The **drawAll** method draws all the bubbles in the arraylist of bubbles.

Make sure you test as many methods of the **Model** class as possible.

Question 4

Create a **ViewBubbles** class that extends **JPanel** and has the following UML specification:

```
+----------------------------------------------+
|                 ViewBubbles                  |
+----------------------------------------------+
| - model: Model                               |
+----------------------------------------------+
| + ViewBubbles(Model model)                   |
| - moveUp(int w, int h): void                 |
| - clickBubbles(int x, int y): void           |
| # paintComponent(Graphics g): void           |
+----------------------------------------------+
```

The constructor of the **ViewBubbles** class must create a timer (from the **javax.swing** library, not from the **java.util** library!). This timer ticks every 500 milliseconds. Every time the timer ticks, the timer's action listener must call the **moveUp** method with the width and height of the panel as arguments (remember that **ViewBubbles** extends **JPanel**, so a view is a panel). Remember to start the timer!

The constructor of the **ViewBubbles** class also adds a mouse listener to the panel (use the **MouseAdapter** class to implement the mouse listener). Every time the user clicks into the panel, the **mouseClicked** method of the mouse listener must call the **clickBubbles** method with the **x** and **y** coordinates of the mouse event as arguments.

The constructor of the **ViewBubbles** class must also set the background color of the **ViewBubbles** to white (this will help you see the bubbles).

The **moveUp** method is private and takes a width **w** and height **h** as arguments and does the following:

- it calls the **moveAll** method of the model to move all the bubbles up by **1** pixel (remember that the **y** direction is pointing down);
- it calls the **clearInvisibles** method of the model with the width **w** and height **h** as arguments to delete all the bubbles that have become invisible after moving;
- it calls the **addBubble** method of the model to add a new bubble;
- it uses the **model**'s **getScore** method to print the current score on the standard output, but only if the current score is different from the score before the call to the **clearInvisibles** method;

- it calls the **`repaint`** method (this in turn will make Java automatically call the **`paintComponent`** method of the **`ViewBubbles`**).

The **`clickBubbles`** is private takes the position (**`x`**, **`y`**) of a mouse click as arguments and does the following:

- it calls the **`deleteBubblesAtPoint`** method of the model to delete all the bubbles which contain the point where the mouse was clicked;
- it uses the **`model`**'s **`getScore`** method to print the current score on the standard output, but only if the current score is different from the score before the call to the **`deleteBubblesAtPoint`** method;
- it calls the **`repaint`** method (this in turn will make Java automatically call the **`paintComponent`** method of the **`ViewBubbles`**).

The **`paintComponent`** method calls the **`drawAll`** method of the model, which will draw all the bubbles.

Create a **`MyFrame`** class that extends **`JFrame`**. Set the size, title, default close operation of the frame and make it visible. Do not add a layout manager to the frame.

In the constructor of **`MyFrame`**, create a **`Model`** object. Then create and add a **`ViewBubbles`** object to the frame.

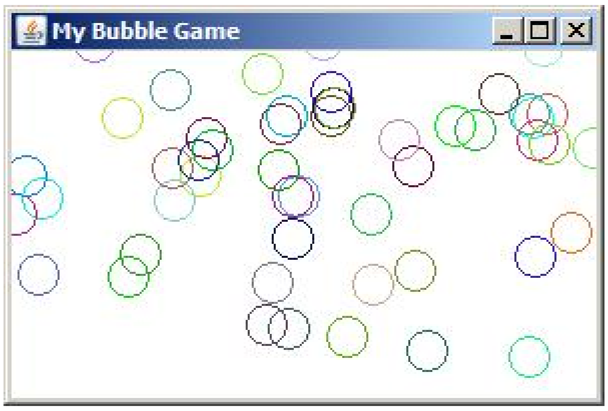
Then in the **`main`** method of the **`Start`** class, after the tests, create an anonymous class that implements the **`Runnable`** interface with a **`run`** method that creates a **`MyFrame`** object, and use the **`javax.swing.SwingUtilities.invokeLater`** method to run that code on the event dispatch thread.

Run your program and check that:

- a new bubble is correctly created every half second;
- all the bubbles move up by 1 pixel every half second;
- bubbles disappear immediately when you click on them (not the next time all the bubbles move up);
- clicking on two bubbles that overlap makes both bubbles disappear;
- your score increases when you click on bubbles;
- your score decreases when bubbles disappear at the top of the window;
- bubbles appear everywhere in the window, even when you resize the window to be bigger.

Note: when you click the mouse button, Java considers it a mouse click only if the mouse has not moved between the moment when you press the button and the moment when you release the mouse button. So make sure that you do not move the mouse at the same time you are clicking the mouse button, otherwise Java will just ignore the button click!

Your program must then look like this:



Note: once your program is working, you can decrease the timer from 500 milliseconds to 200 milliseconds, or even down to 100 milliseconds, and then see how long you can keep your score positive!