Creating a bubble animation

This week's portfolio task involves simulating a screensaver application by creating a "Bubble Animation". There are two classes in your bubble animation. A bubble is represented by the class called `Bubble`. The animation is implemented in a class called `Screensaver`.
Before you start, download the starter project from Blackboard.
Before you write any code, watch the video about this task on Blackboard and read all of these instructions.

1 The `Bubble` class

1.1 Fields

The `Bubble` class has the following fields:

- A `Circle` object which is used as the visual representation of the bubble (in the same way a brick used a rectangle in the "Brick Walls" task).
- A direction of movement. The direction of movement is represented as TWO integer fields:
  - A distance in the X direction (In this document, I refer to this field with the name `xDirection`)
  - A distance in the Y direction (In this document, I refer to this field with the name `yDirection`)

  This distance may be negative, positive, or zero. For example, a negative X distance means the bubble moves left. A positive X direction means the bubble moves right. A zero X direction means that the bubble does not move in the X axis at all.
- A random number generator for generating random values for the direction of movement. This field is an instance of `java.util.Random`. Read the Appendix to see how to use the `Random` class in this task.

1.2 The constructor

The constructor for your bubble class should do the following:

- Construct a new `Circle` and use it to initialize the appropriate field.
- Randomize the position of the circle. You should use the existing method `randomizePosition` in the `Circle` class to randomize the position of a circle.
- Construct an instance of the `Random` object and use it to assign a value to the appropriate field (see the example code in the Appendix).

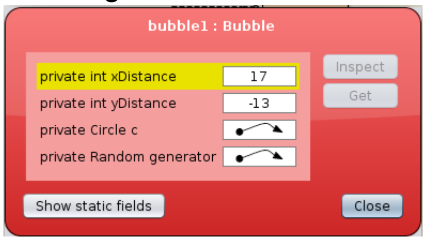
Do not make the circle visible in the constructor (we will do that later in a separate method).
Create the following method to set the direction of movement of the bubble:

```
public void setDirection()
```

In the body of this method you set the `xDirection` and the `yDirection` of your bubble. To implement this you do the following for **each** direction:

- generate a random number in the range 0 to 19 as the distance to move.
- generate a random boolean value:
  - if the boolean value is `true`, multiply your random number (the distance) by -1.

Finally call your `setDirection` method at the end of the `Bubble` constructor.
If you now construct a `Bubble` object and use the object inspector to view its state you should see something like the following:



bubble1 : Bubble

```
private int xDistance        17
private int yDistance        -13
private Circle c
private Random generator
```

1.3 The methods

1. Create a method to set the colour of the circle:

```
public void setColor(String color)
```

   This method should change the colour of your circle to the given `color` value.

2. Create a method to show the circle:

```
public void show()
```

   This method will make the circle visible.

3. Create a method to hide the circle:

```
public void hide()
```

   This method will make the circle invisible.

4. Create a method to move the circle.

```
public void move()
```

   This method will move the circle horizontally and vertically according to the values of `xDistance` and `yDistance`. A moving circle should **never** disappear off the visible canvas. You can find out the dimensions of the canvas using the following code in your `Bubble` class:

```
int height = Canvas.getCanvas().getHeight();
int width = Canvas.getCanvas().getWidth();
```

   In the X axis, the canvas ranges from 0 to `width`. In the Y axis, the canvas ranges from 0 to `height` (remember, in the Y axis, zero is at the top of the canvas and `height` is at the bottom). You should implement your `move` method so that movement on the canvas wraps around as follows:

- If the circle goes off the right hand edge of the canvas, it should reappear on the left hand side (**hint** subtract the width of the canvas from the X coordinate).
- If the circle goes off the left hand edge of the canvas, it should reappear on the right hand side (**hint** add the width of the canvas to the X coordinate).
- If the circle goes off the bottom of the canvas, it should reappear at the top (**hint** subtract the height of the canvas from the Y coordinate).
- If the circle goes off the top of the canvas, it should reappear at the bottom (**hint** add the height of the canvas to the Y coordinate).

Now test each of your methods. Construct a bubble object and call the methods to check they work as described above.

2 The `ScreenSaver` class

The `ScreenSaver` class makes use of two other classes. It makes use of the `Bubble` class to generate some bubbles. It makes use of the `ColorSelector` class to manage a list of colour names used for setting the colours of your bubbles. You will find that the `ColorSelector` class is essentially the code you used in the Brick Walls task for selecting different colours for bricks. Here we have refactored that task of managing a list of colour names into a class of its own. The class has one method, `getNextColor` which returns a `String` as the name of the next colour to use.
The following behaviour should be implemented in your `ScreenSaver` class:

- A specified number of bubbles should be created by the constructor.
- The bubbles can be shown (made visible).
- The bubbles can be hidden (made invisible).
- The number of bubbles can be changed.
- The bubbles can be animated. The animation repeatedly moves each bubble.
- The number of steps (repetitions) in the animation can be changed.

2.1 The Fields

The following fields are required:

1. An `ArrayList` of `Bubble` objects.
2. A `ColorSelector`
3. A `boolean` value which has the value `true` when the bubbles are visible and `false` when the bubbles are invisible.
4. An integer which represents the number of steps (repetitions) in the animation.

2.2 The Constructor

Your constructor has one parameter, being the number of bubbles to create.

```
public Screensaver(int numBubbles)
```

The constructor should initialize all four of the fields, constructing an `ArrayList` and a `colorSelector` where appropriate. Set the number of steps in the animation to 100.
Write a **private** method to create and configure a bubble:

```
private void createBubble()
```

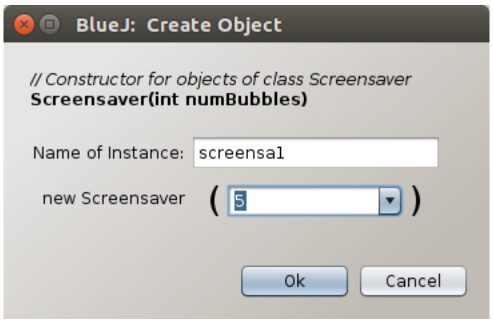
This method should:

- Construct a `Bubble` object
- Get the next colour name from your `ColorSelector`
- Change the colour of the bubble using the colour name
- Add the `Bubble` object to your `ArrayList` of bubbles.

Use a `while` or `for` loop to repeatedly call your `createBubble` method to create the required number of bubbles (according to the value of the parameter, `numBubbles`).
Write a (public) method to show each bubble:

```
public void show()
```

This method should call the `show` method in the `Bubble` class for each bubble in the list. The show method should assign the value `true` to the field that records whether or not the bubbles are visible.
Finally call your `show` method as the last step in the constructor.
Now test your constructor. You should be asked to enter the number of bubbles required:

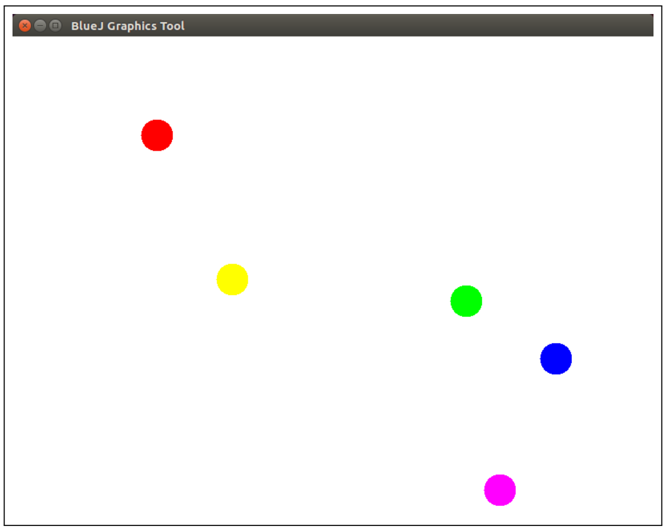


// Constructor for objects of class Screensaver
Screensaver(int numBubbles)

Name of Instance: screensa1

new Screensaver (5)

The canvas will appear and you should see the correct number of circles representing your bubbles.



2.3 The Methods

You have already written the `show` method when you wrote the constructor. Now add the rest of the methods:

1. Create a method to hide the bubbles.

   ```
   public void hide()
   ```

   This is like your show method but it should call the `hide` method of the `Bubble` class. The method should assign the value `false` to the field that records whether or not the bubbles are visible.

   Test the `hide` and `show` methods by constructing a `ScreenSaver` and calling its `hide` and then `show` method to ensure that all the bubbles are hidden and shown.

2. Write a method to return the number of steps in the animation.

   ```
   public int getAnimateLength()
   ```

   This method simply returns the value of the appropriate field.

3. Write a method to set the number of steps in the animation.

   ```
   public void setAnimateLength(int newLength)
   ```

   If the value of `newLength` is greater than zero, this method should assign `newLength` as the value of the appropriate field. If `newLength` is less than or equal to zero, the method need do nothing.

4. Write a method to implement the animation:

   ```
   public void animate()
   ```

   This method first checks if the bubbles are visible or not. If the bubbles are not visible a message should be printed out to tell the user to first show the bubbles.

   If the bubbles are visible, we run the animation. Each step in the animation involves moving every bubble in our list of bubbles.

   We need TWO repeat loops, one inside the other to implement this. We call this "nested" loops where one loop is inside another. The logic is as follows:

   ```
   repeat for the number of steps in the animation {
       repeat for each bubble in the list {
           move the bubble
       }
   }
   ```

   It is probably better to use a `for` loop for each of the "repeat" statements above. To move the bubble, we call the `move` method in the `Bubble`.

   Test your animation. It is probably safer to set the number of steps in the animation to ONE (call the method `setAnimateLength` and enter 1 as the parameter value) then check that each bubble moves once when you call `animate`. Then change the number of steps in the animation to TWO and check that each bubble moves twice when you call `animate`. Finally set the number of steps to a larger number so that the animation continues longer.

3 Advanced steps

If you finish the above, try the following steps. These steps are designed to challenge the better programmers, don't worry if you don't do these steps.

1. Implement a method to change the number of bubbles on the canvas:

   ```
   public void setNumBubbles(int numBubbles)
   ```

   If `numBubbles` is greater than zero, this method will do one of two things:

   - if `numBubbles` is less than the number of bubbles in the list, `remove` and `hide` the appropriate number of bubbles. The `ArrayList` has a method called `remove`, and you should repeatedly remove the **first** bubble in the list. The `Bubble` class has a `hide` method which will remove the bubble from the canvas.
   - if `numBubbles` is greater than the number of bubbles in the list, add the appropriate number of bubbles to the list. These bubbles should only be shown if the display is currently showing bubbles. If the other bubbles are hidden, the additional bubbles should be hidden.

2. When circles are constructed in the `Bubble` class, use the random number generator to change the size of the circle to a random size. Bubbles should be MINIMUM of 50 pixels in size and a MAXIMUM of 150 pixels in size.

3. After moving a bubble in the `animate` method of the `Screensaver`, use a random number generator to change the direction of movement of the bubble. You can do that by calling the `setDirection` method of the `Bubble`. Bubbles should have a probability of 0.1 (10%) of changing direction. Therefore if you generate a random number in the range zero to 9, the direction should change whenever a zero is generated.

Appendix: The `Random` class

To use the `Random` class in this task, you need the following:

- Import the class from the library. This line goes at the top of your `Bubble` class:

  ```
  import java.util.Random;
  ```

- Declare a field of type `Random`:

  ```
  private Random generator;
  ```

- In the constructor for your `Bubble`, initialize the `Random` object:

  ```
  generator = new Random();
  ```

- To use your `Random` object to generate random numbers, you need to use its `nextInt` method, as follows:

  ```
  int randomNumber = generator.nextInt(20);
  ```

The value 20 in the code above is an example of the use of this method. The `nextInt` method takes an upper bound value as a parameter. Here is what the official documentation says about the method:

> `public int nextInt(int bound)`
> **Parameters**:
> `bound` – the upper bound (exclusive). Must be positive.
> **Returns**:
> the next pseudorandom, uniformly distributed int value between zero (inclusive) and bound (exclusive) from this random number generator's sequence

Therefore in our example above, `nextInt(20)` will generate a random number with a value between 0 and 19 (the bound is 20 and that value is **excluded** from the values returned). If we called `nextInt(5)`, we would get a random number between 0 and 4.

- To use your `Random` object to generate random boolean values (ie `true` or `false`) you need to use its `nextBoolean` method as follows:

```
boolean bool = generator.nextBoolean();
```

You can read the full documentation for `Random` here:
`http://docs.oracle.com/javase/8/docs/api/java/util/Random.html`