Sarai Gould & Laura Cruz-Albrecht
CS106A

Summer 2019
August 17th

CS106A Final Exam

This is a closed book, closed note exam. You are, however, allowed 2 pages of notes. You are also allowed a Syntax Reference sheet provided to you at the exam. You will be graded on functionality -- but good style saves time and helps graders understand what you were attempting. You have 180 minutes. We hope this exam is an exciting journey.

I acknowledge and accept the letter and spirit of the honor code. I pledge to write more neatly than I have in my entire life:

Signature: _____

First Name: _____

Last Name: _____

SUNET:_____

Problem 1: Warmup Problems [15 points]

This problem has 4 parts.

Problem 1.1: Short Answer [3 points]
What is the difference between pass by value and pass by reference? Why is this important?

Problem 1.2: Short Answer [2 points]
What is the difference between a parameter and a return value?

Problem 1.3: True/False [5 points]
The following statements may be **True** or **False**. In the box below, you will write "True" to show you know a corresponding statement is true or "False" to show you know a corresponding statement is false.

1. A **HashMap** is passed by value. _____
2. An **int** is passed by value. _____
3. **ArrayList**s can store primitives and Objects. _____
4. You can alter a string using **.charAt(index)**. _____
5. A class defines a new variable type. _____

Problem 1.4: Reading Code [5 points]

You will evaluate what different data structures are equal to at different points in the code below. Those points are designated by green comments that say something like:

`// Part 1: What is deepFriedOreo equal to here?`

When asked to represent an ArrayList, please write it like:

`[a, b, c, d, e]`

where a, b, c, d, and e are different elements in the `ArrayList`.

When asked to represent a `2D array`, please write it like:

`[a, b, c, d, e],`
`[f, g, h, i, j],`
`[k, l, m, n, o],`

where `a - o` are different elements in the `2D array`. This is an example of a `2D array` with 3 rows and 5 columns.

```
public class CottonCandyStand extends ConsoleProgram{
    public void run() {
        int[][] cottonCandy = new int[3][4];

        ArrayList<Integer> deepFriedOreo = new ArrayList<>();

        for(int sprinkle = 0; sprinkle < 10; sprinkle +=2) {
            deepFriedOreo.add(sprinkle);
        }

        // Part 1: What is deepFriendOreo equal to here?

        int x = deepFriedOreo.get(0);
        int y = deepFriedOreo.get(1);
        deepFriedOreo.remove(1);

        cottonCandy[x][y] = deepFriedOreo.get(deepFriedOreo.size() - 1);

        // Part 2a: What is deepFriendOreo equal to here?
        // Part 2b: What is cottonCandy equal to here?

        int jellyBean = eatCandy(cottonCandy);
        deepFriedOreo.add(jellyBean);

        // Part 3a: What is deepFriendOreo equal to here?
        // Part 3b: What is cottonCandy equal to here?

    }

    private int eatCandy(int[][] twizzlers) {
        twizzlers[1][1] = 22;
        twizzlers[2][1] = twizzlers.length;
        return twizzlers[0][0];
    }
}
```

Answer for Part 1:

Answer for Part 2a:

Answer for Part 2b:

Answer for Part 3a:

Answer for Part 3b:

Problem 2: Raffle Tickets [20 points]

Welcome to the CS106A Carnival!

Check how closely your raffle ticket matches the winning code! We all love raffles, and we love to win prizes. This raffle is special because you can still win a prize if your raffle value is *close* to the actual raffle code. You will be determining the distance between two raffle values. The distance is defined as the number of indices where the two codes differ from each other.

Here's an example where the winning raffle value was "A1B2" and your raffle value was "AAB2":
Winning Value: "A**1**B2"
Your Value: "A**A**B2"
In this example, **the distance was 1**, because the strings differ at exactly one index.

You will write a method where you calculate the distance between the winning raffle value and your raffle value. Your method must print the distance between the two values and print the size of the prize that a user won!

- A **distance of 0** means the user won!
- A **distance of 1 or 2** means the player won a small prize!
- A **distance of greater than 2** means the player won no prizes. :(

Sample Method Call:	**Printed Output:**
`checkRaffleDistance("ABCDEF", "ABDCEF");`	`Your raffle code is a distance of 2 away!` `Congratulations, you win a small prize!`
`checkRaffleDistance("1A2B3C4D", "00000000");`	`Your raffle code is a distance of 8 away!` `Sorry, you didn't win any prizes.`
`checkRaffleDistance("1", "B");`	`Your raffle code is a distance of 1 away!` `Congratulations, you win a small prize!`
`checkRaffleDistance("1Q2W3E", "1Q2W3E");`	`Your raffle code is a distance of 0 away!` `Congratulations, you're the winner!`

You will fill in this method:

```
private void checkRaffleDistance(String winningVal, String yourVal) {
        // TODO: Fill this in

}
```

Constraints/Assumptions:

- `winningVal` and `yourVal` will always be the same length. The length will always be at least 1. There will be no values of length 0.
- `winningVal` and `yourVal` can contain letters, numbers, or other valid ASCII characters.
- `winningVal` and `yourVal` will never contain lowercase letters.

WRITE ANSWER ON NEXT PAGE

```
/* Answer for Problem 2 */

// You can assume all necessary libraries are imported for you.
// You only need to implement this method.

private void checkRaffleDistance(String winningVal, String yourVal) {









}
```

Problem 3: Designing Rides [30 points]

What's a carnival without rides? We'll be designing and using a Ride class for our Carnival.

This question has 2 parts.

Part 1: Designing the Ride Class [20 points]
Write a class that represents a Ride. A Ride has a **name** (such as "Green Python" or "Teacups"), a **type** (such as "Roller coaster" or "Spin"), an **ArrayList of guests** on the ride, and a **working status** indicating whether the ride is working (should be `true` if the ride is working, and `false` otherwise).

You create a new Ride by specifying the name and type of the ride, as illustrated below. Upon initialization, a new Ride has an empty list of guests, and *is* working.

```
Ride ride = new Ride("Teacups", "Spin");   // Name: Teacups, Type: Spin
```

A Ride should have the following six public methods:

```
/* Adds the guest to the ride. */
public void enterRide(String guestName);

/* Removes the guest from the ride. Returns true if successfully able to
remove them, returns false if they weren't on the ride. */
public boolean exitRide(String guestName);

/* Returns the number of guests on the ride. */
public int getNumGuests();

/* Returns whether the ride is working (true: working; false: not
working).*/
public boolean getWorkingStatus();

/* Sets the working status to equal the provided status. */
public void setWorkingStatus(boolean status);

/* Returns a String representation of the Ride Object. Should print out all
the instance variables associated with the Ride. */
public String toString();
```

WRITE ANSWER ON NEXT PAGE

```
/* Answer for Problem 3, Part 1 */

// You can assume that all necessary libraries are imported for you

public class Ride {
    // TODO: Create any Ride instance variables here









    // TODO: Create Ride Constructor here









    // TODO: Create Ride public methods here
```

```
// Additional space for Problem 3, Part 1

}
```

Part 2: Are the Rides Working? [10 points]

Uh oh, a surprise inspection is taking place at the carnival! Which rides are working? We need to let the inspector know which rides are working and which ones are not.

You will write a method that takes in an array of Rides as a parameter and returns a corresponding boolean array of the working status of those rides. If a ride is working, the corresponding index in the boolean array is true. If a ride is *not* working, the corresponding index in the boolean array is false.

For example, imagine we had the following array of rides:

```
Ride[] rides = [ride1, ride2, ride3, ride4];
```

Now, let's say that **ride1 and ride3 are *not* working** while **ride2 and ride4 *are* working**. Your method would return this array:

```
[false, true, false, true]
```

You will implement the following method:

```
private boolean[] workingRides(Ride[] rides) {
        // TODO: Fill this in

}
```

Constraints/Assumptions:

- You can assume all methods described in part 1 work as intended, regardless of your implementation.
- You have access to all the public methods defined in the `Ride` class from part 1.
- Your method should work regardless of the size of the `rides` array.

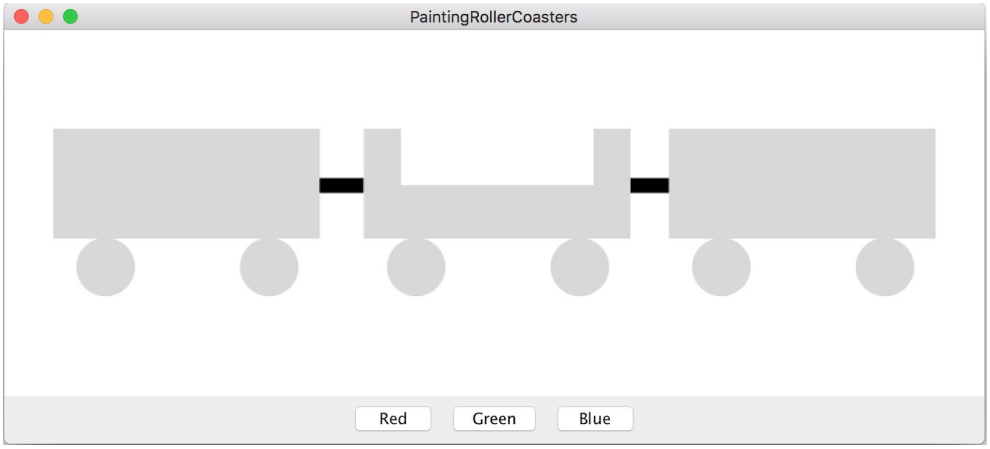
WRITE ANSWER ON NEXT PAGE

```
/* Answer for Problem 3, Part 2 */

// You can assume that all necessary libraries are imported for you
// You do not need to implement any other parts of the program
// except for the following method.

private boolean[] workingRides(Ride[] rides) {
    // TODO: Fill this in









}
```

Problem 4: Painting Roller Coasters [45 points]

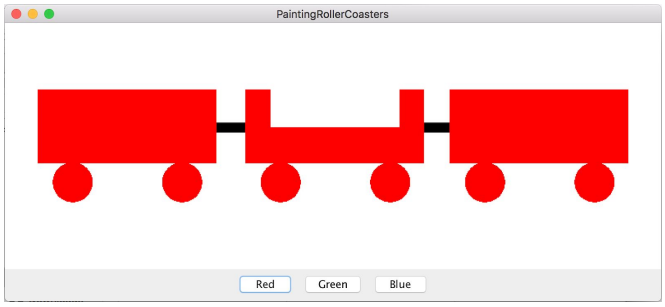
Let's paint roller coasters! We'll take images of roller coasters and customize the color we'd like them to be! You will be given an image of a roller coaster to start with, and you will write a Graphics program that can "paint" this roller coaster red, green, or blue.

When you first run your completed program, it will look like this:

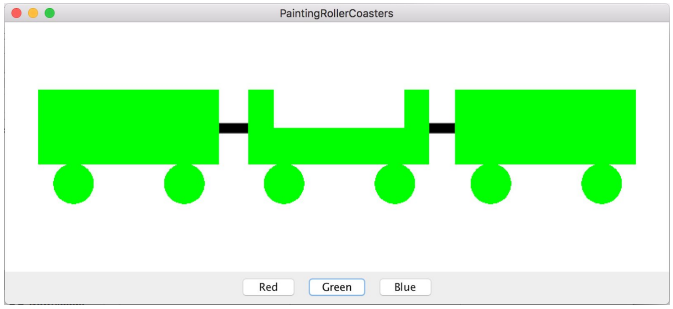


The roller coaster image is on the canvas, and there are three buttons below the canvas. These buttons are labeled "Red", "Blue", and "Green".

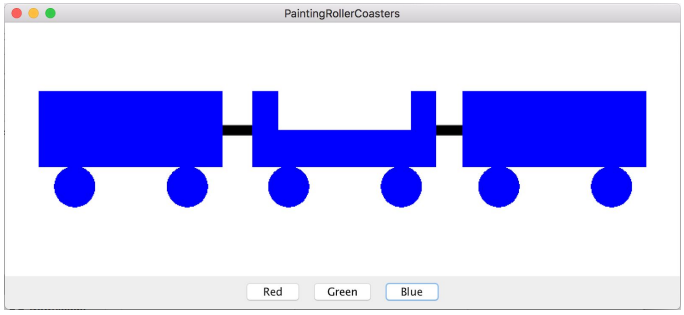
If we click on the "Red" button, this happens:



If we click on the "Green" button, this happens:



If we click on the "Blue" button, this happens:



When the "Red" button was clicked, the program used the following approach to recolor the roller coaster.

- Leave all white pixels and black pixels unmodified.
- Color all non-white and non-black pixels red.

Remember, a white pixel has an RGB value of (255, 255, 255) and a black pixel has an RGB value of (0, 0, 0). A pixel with any other combination of RGB values was colored red. Note that a red pixel has the blue and green values set to 0, and the red value set to its maximum value of 255.

A similar strategy is used for coloring the coaster green and blue. A green pixel has a green value of 255 and red and blue values of 0. Similarly, a blue pixel has a blue value of 255, and red and green values of 0.

In your program, you must:

- Add the roller coaster image to the canvas. You should use the given constant `IMAGE_FILE`.
- Add the Red, Green, and Blue buttons below the canvas.
- Process the photo when each button is pressed.
- Replace the current roller coaster image with the new processed photo on the canvas.

Constraints/Assumptions:

- Pixels can be any combination of RGB values, you cannot assume there will only be black, white, or gray pixels in the starting photo.
- You don't need to resize the image to fit the canvas or center the image in the middle of the canvas.

WRITE ANSWER ON NEXT PAGE

```
/* Answer for Problem 4 */

// You can assume that all necessary libraries are imported for you

public class PaintingRollerCoasters extends GraphicsProgram {

	// the filename of the ride image you will use
	private static final String IMAGE_FILE = "ride.png";

	// TODO: Fill this in
```

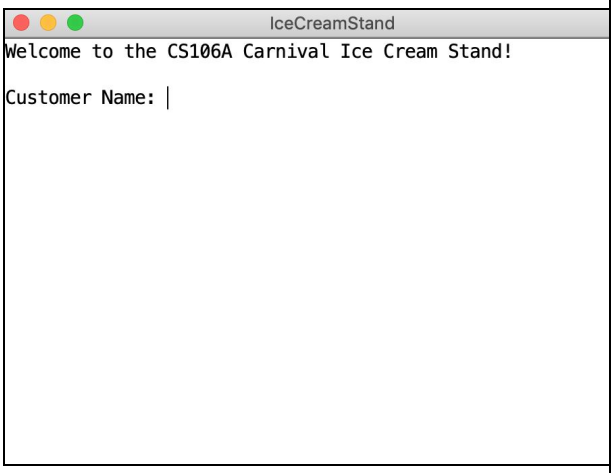
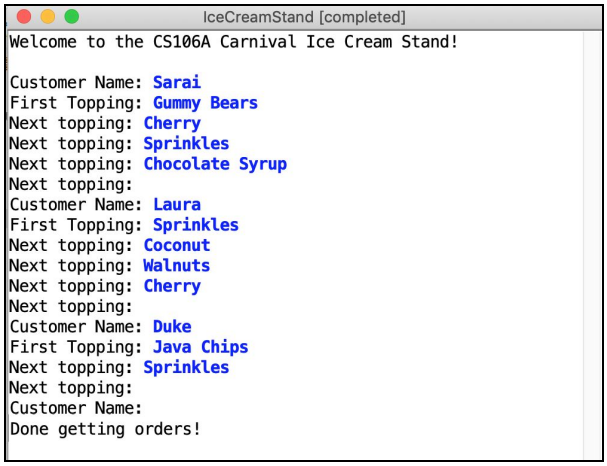
```
// Additional space for Problem 4

}
```

Problem 5: Ice Cream Stand [30 points]

Attending a Carnival is a lot of work, and we're ready for some ice cream!

This question has 2 parts.

Part 1: Collecting Toppings [15 points]
First, let's write a method that will take in customer names, record their topping choices, and return a `HashMap` containing that information for use later. You will associate a customer's name with their list of topping choices.

Before User Input:	After User Input:
	

First, we will read in a customer name. Then, we will ask for and read in the toppings they want one by one. We will keep accepting new toppings until a user inputs the empty String. Once we see the empty String, we will prompt for a new customer name. If we see an empty String instead of a new customer name, we will end the program.

After we've run the method, we would have a `HashMap` with the following mapping:

Sarai : *Gummy Bears, Cherry, Sprinkles, Chocolate Syrup*
Laura : *Sprinkles, Coconut, Walnuts, Cherry*
Duke : *Java Chips, Sprinkles*

You will define this method:

```
private /* TODO return type */ getOrders() {
        // TODO: Fill this in

}
```

Constraints/Assumptions:

- Your method must return a HashMap.
- You must fill in the TODO where you are asked to specify a return type. You should delete the TODO comment and replace it with a valid Java type.
- You don't know how many customers there will be or how many toppings they will want. You will only stop reading names or toppings when you see the corresponding empty String.

WRITE ANSWER ON NEXT PAGE

```
/* Answer for Problem 5, Part 1 */

// You can assume that all necessary libraries are imported for you
// You do not need to implement any other parts of the program
// except for the following method.

private ____________________________________ getOrders(){
      // TODO: Fill in return type above, and body of method here









}
```

Part 2: Getting Topping Counts [15 points]

We need to make sure we have enough toppings for all our customers! Second, you will write a method `getToppingCounts` that takes in the `HashMap` that you constructed in Part 1 as a parameter, and returns a new `HashMap` that instead associates a ***topping*** with a ***count*** of how many people want that topping.

If we run the `getToppingCounts` method, using the customers-to-toppings map from Part 1 as the parameter, it would return a new `HashMap` with the following mapping:

Gummy Bears : *1*
Cherry : *2*
Sprinkles : *3*
Coconut : *1*
Chocolate Syrup : *1*
Walnut : *1*
Java Chip : *1*

You will define this method:

```
private /* TODO return type */ getToppingCounts(/* TODO param type */ customerToToppings) {
        // TODO: Fill this in

}
```

Constraints/Assumptions:

- You can assume your `getOrders()` method from part 1 works as intended, regardless of your actual implementation.
- Your method must return a HashMap.
- You must fill in every TODO where you are asked to specify a return or parameter type. You should delete the TODO comment and replace it with a valid Java type.
- You cannot read in input from a user in this method. The only customer/toppings data you have has been passed in as the parameter `customerToToppings.`
- You don't know how many customers there will be or how many toppings there will be.
- You must implement the algorithm to count toppings in this method. You should not implement it in part 1.

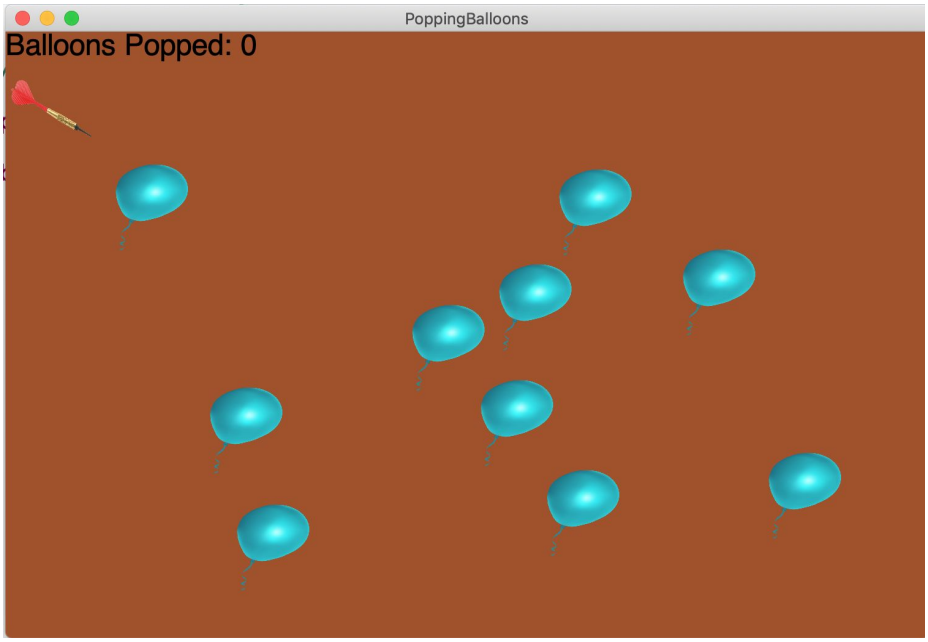
WRITE ANSWER ON NEXT PAGE

```
/* Answer for Problem 5, Part 2 */

// You can assume that all necessary libraries are imported for you
// You do not need to implement any other parts of the program
// except for the following method.


private _____ getToppingCounts(_____ customerToToppings){
      // TODO: Fill in return and parameter types above
      // and fill in body of method here









}
```

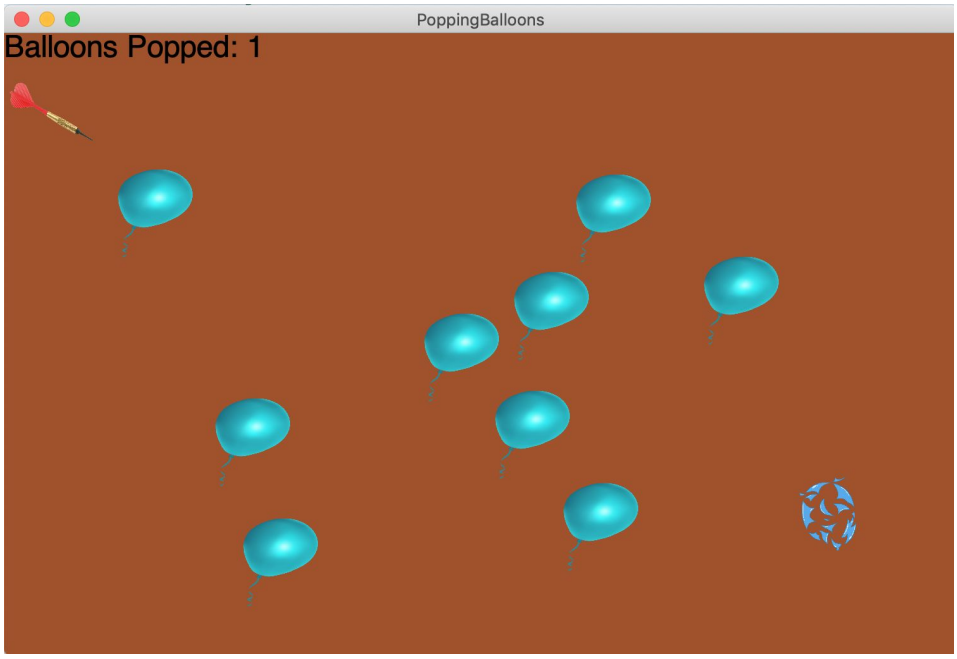
Problem 6: Popping Balloons [40 points]

You're going to try your hand at the carnival's Popping Balloons game! Your task is to write a Graphics program that adds full (unpopped) balloons to the canvas, and lets the user "throw" darts (by clicking on a dart image) to pop balloons.

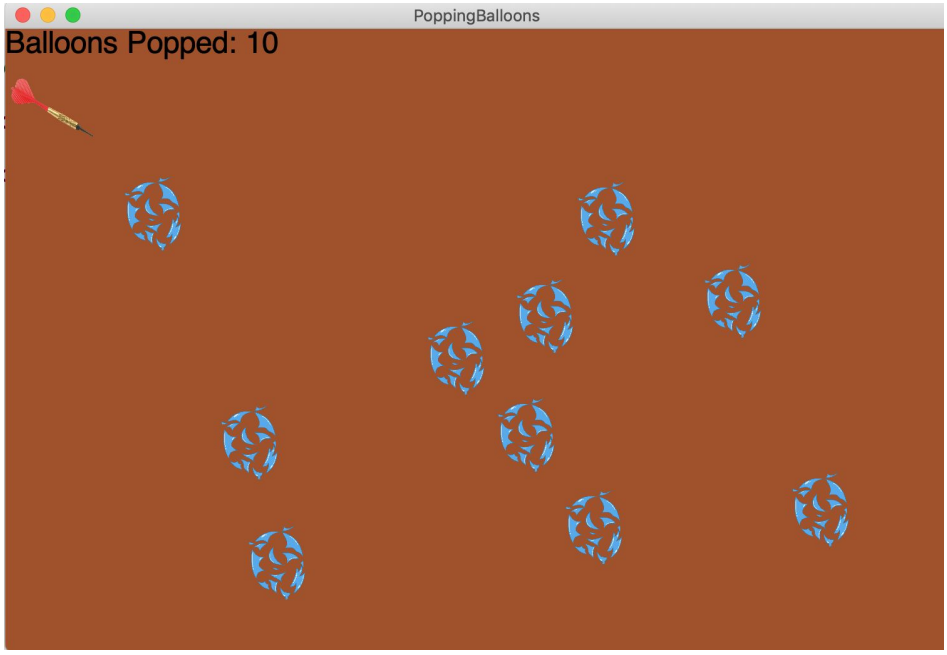
When you first run your completed program, it will look like this:



If you click on the dart, it will pop a random balloon! After one balloon is popped, the game will look something like:



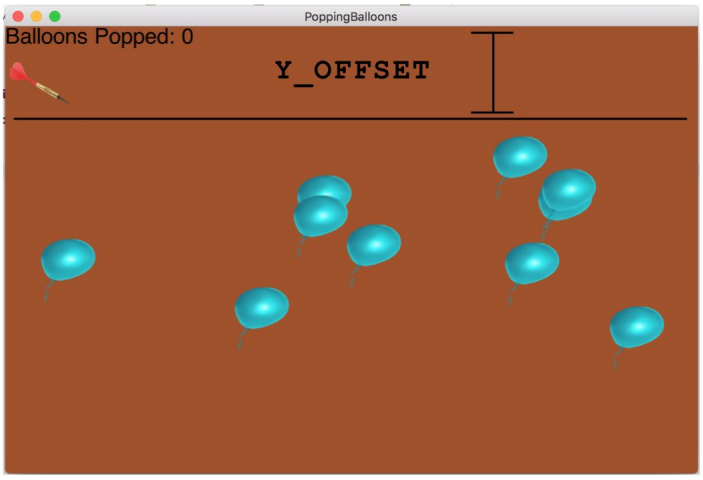
After all of the balloons have been popped, the game will look like:



We have included some setup written for you that sets the background color, adds a GLabel with the initial count of popped balloons called `poppedBalloonsLabel`, and adds a dart image called `dart`. Your program has access to instance variables for the `poppedBalloonsLabel` and `dart`.

In your program, you should:

- add images of balloons to **random** locations on the canvas. You should use the given constants `NUM_BALLOONS` and `BALLOON_FILENAME`.
- You must place balloons so that they do not overlap with the label or the dart image. You should use the given constant `Y_OFFSET` to determine the required minimum distance from the top of the canvas.



- Every time the user clicks on the dart image, you must **select a random unpopped balloon** from the canvas and pop it. To pop a balloon, you must remove its image from the canvas and replace it with a popped balloon image in the same location. You should use the constant `POPPED_FILENAME`.
- Every time the user pops a balloon, you must also update the `poppedBalloonsLabel` to display the number of balloons that have been popped.

Constraints/Assumptions:

- You **cannot use arrays** on this problem. You should use ArrayLists to solve this problem.
- You can assume both of the balloon images have a width and height of BALLOON_SIZE. You do not need to explicitly set their size.
- All balloons must be fully visible on the canvas.
- The `poppedBalloonsLabel` and `dart` have been added to the canvas for you. They are included in the starter code as instance variables.
- Even if `NUM_BALLOONS` is equal to 10 in the starter code, you cannot assume that `NUM_BALLOONS` will be equal to 10 every time your run the program. Your program must still work even if `NUM_BALLOONS` is set to a new number.
- `NUM_BALLOONS` will always be greater than 0.

WRITE ANSWER ON NEXT PAGE

```
/* Answer for Problem 6 */
// You can assume that all necessary libraries are imported for you

public class PoppingBalloons extends GraphicsProgram {
	// filenames and necessary constants
	private static final String BALLOON_FILENAME= "balloon.png";
	private static final String POPPED_FILENAME = "poppedBalloon.png";
	private static final int Y_OFFSET = 100;
	private static final int NUM_BALLOONS = 10;
	private static final int BALLOON_SIZE = 75;

	// instance variables for dart and popped balloons label
	private GImage dart = new GImage("dart.png");
	private GLabel poppedBalloonsLabel = new GLabel("Balloons Popped: 0");




	// You do not need to change the below method
	private void setup(){
		setBackground(new Color(160,82,45));
		// adds the dart picture and the popped balloons label
		add(poppedBalloonsLabel, 0, poppedBalloonsLabel.getAscent());
		add(dart, 0, poppedBalloonsLabel.getHeight());
	}
	public void run(){
		setup();
		// TODO: Put code that belongs inside of run method here.









	}
	// TODO: Put code that belongs outside of run method here.
```

```
// Additional space for Problem 6




}
```

END