I Love Being Right

Dr. Andrew Rosen

1 Let's play a game

For this lab, we will be making a small text based game to teach you and your friends a little bit about human psychology and get you back into the habit of programming. If you're in lab, we'll start with a small demo of the game. You can find the game here for reference. When the game is over and we all know how to play it, flip to the next page.

This game plays off of a simple cognitive bias called the confirmation bias. People are predisposed to look for information that confirms their beliefs. If you believe the rule is something like "each number is twice as large as the last number" you are going to make guesses that confirm your belief. Each positive answer will further reinforce your belief.

In science, we need to check our hypotheses by attempting to *falsify* them, or prove them wrong. We need to attempt to disprove the rule, by checking sequences that don't follow our hypothesized rule.

2 Let's make a game

Your task is to recreate this game in Java, albeit in a simpler, text-based format. The player enters sequences of three numbers to see if they follow the hidden rule. When the player has entered their sequence, the program should reply whether or not the sequence followed the rule.

You can use the same rule or create your own. Regardless, the game is played the same way.

2.1 How

Create a loop which gives the player instructions on what to input. Then read input from the player. The input will be either one of three options:

- If the user enters the word "answer" or some other string you choose to indicate the player is ready to end the game and guess.
  - In this case, prompt the user to guess our game's rule, then output the answer.
  - You don't have to tell the user if he guessed the rule correctly; it would be beyond the scope of this Lab.
- Three numbers separated by spaces.
  - Let's call a trio of numbers and the corresponding output a `Guess`. Once a user makes a `Guess`, we will store it (how? see below).
  - If the user enters a sequence that follows the rules, output "Yes!"
  - Otherwise output "No."
- If the user enters the word "previous", display all the previous `Guess`es the user made (the numbers and whether they were right or wrong)
- Treat any other entry as an exception.

2.2 Error Handling

Your program should not crash, regardless of what I input. Possible inputs I might enter:

- Too many numbers for a guess
- Too few numbers for a guess
- An invalid option
- Complete gibberish

2.3 Storing your answers

You will be storing your answers in an `ArrayList` of `Guess` objects. A `Guess` is a very simple. It should hold three numbers[1] and have a `toString()` method to make printing out its contents easier.

An `ArrayList` object is much like an array. In fact, for now, think of it as a smart array which can grow to whatever size you need. It doesn't need a predefined size. All the methods for the `ArrayList` can be found by clicking on this sentence; there are a lot of them, but we will see that `ArrayList` is powerful and simple.

Once you import `java.util.ArrayList`, you can create an ArrayList like so:

```
ArrayList list =  new ArrayList();
```

and add anything you want by doing this:

```
list.add(thingYouWantToAdd);
```

However, you will be seeing warning such as "`ArrayList` is a raw type. References to generic type `ArrayList<E>` should be parameterized" This is because `ArrayList` uses generics, which we will cover in more detail during lecture. Briefly, a generic type in Java, such as `ArrayList`, lets you define what you use it with. With collections of objects, such as `ArrayList`, we can specify that we want an ArrayList that holds just one kind of object. For example, if I wanted to create an `ArrayList` of `Guess` objects, we do the following[2]:

```
ArrayList<Guess> list =  new ArrayList<Guess>();
```

and then you can add guesses to your `ArrayList`.

[1] int or double is your choice, but if you choose ints, you need to handle rejecting inputted doubles

[2] This assumes you have already made your guess object

2.4 Useful Methods

There are a many ways to parse your input, but only a few that are easy. Here are a few different suggestions.

- **Only** use `Scanner.nextLine()` to read in input. Using `Scanner.nextInt()` and `Scanner.nextLine()` together will lead to confusion.
- To turn a string into a number, use `Integer.parseInt( theString)` or `Double.parseInt( theString)`.
  - `String.split()` is useful for breaking apart strings seperated by *delimiters*, such as commas, or, in this program, spaces.
  - `String.split()` returns an array of Strings.
- How do I tell what the string the user input is? First, check either if it's the word "answer", then check if it's the word "previous" first. Otherwise assume it's sequence of numbers three numbers and try to parse them. If the user enters anything else, this will cause an error which you can catch with the `try catch` block.
- Regular expressions are also useful here, if you know how to use them. If you do not know how to use regular expressions and feel like you already know a good amount of programming **learn regular expressions.**[3]
- Break up individual tasks into methods. For example you can have one method to grab user input, another method to detect whether a guess follows a rule, and to print all the `Guess` objects stored in an `ArrayList`.

3 Your Grade

Turn in your assignment on Canvas and come see the Professor or a TA to demo your work for credit.

The breakdown is as follows.

40 points The user inputs three numbers and the program tells whether or not they follow the hidden rule.

10 points The user input "answer" ends the game.

20 points The `try catch` block ensures the program does not crash.

20 points Guesses are stored in an `ArrayList` and can be recalled by entering the word "previous."

10 points The code is relatively organized and does not resemble a mass of spaghetti.

[3] Regular Expressions or regex are not often formally taught as part of a CS curriculum. Messing around with them for two hours might be the most two useful hours of the semester.