Introduction to Java
CS9053 Section I
Thursday 6 PM – 8:30 PM
Prof. Dean Christakos
Assignment 7
March 16th, 2024
Due: March 28th, 2024 11:59 PM

Part I: Working with stacks and queues

1. A stack parameterized with the type E has the following methods:

   boolean empty() – tests if the stack is empty

   E peek() – Looks at the object at the top of the stack without removing it from the stack

   E pop() – Removes the object at the top of the stack and returns that object as the value of this function

   E push(E item) – Pushes an item onto the top of this stack

   int search(Object o) – Returns the 1-based position where an object is on this stack. If it is not there, then it returns -1

   Implement MyStack<E> by providing these methods using an ArrayList.

   Consult the Stack interface https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html#search-java.lang.Object- to see how these methods should work.

   Implementing the Queue interface parameterized with the type E has a lot of methods, but I want to concentrate on the following:

   int size() – returns the number of elements in the queue

   boolean isEmpty() – returns a boolean of whether the queue is empty

   boolean offer(E e) - inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions.

   public E remove() - retrieves and removes the head of this queue. Throws **NoSuchElementException** if this queue is empty

public E poll() - retrieves and removes the head of this queue, or returns null if this queue is empty.

public E element() – retrieves, but does not remove, the head of the queue. Throws **NoSuchElementException** if this queue is empty

public E peek() - retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.

Implement MyQueue<E> by providing these methods using an ArrayList as a field

2. We are going to implement a function called `isBalanced(String inString)`, which you can find in the `BalancedParentheses` class, which sees if a String of parentheses are balanced. You are going to use a Stack (you can use `MyStack` or Java's own `Stack` implementation) to do it.

   Balanced parentheses means that each opening symbol has a corresponding closing symbol and the pairs of parentheses are properly nested.

   For example, these are balanced parentheses:

   ```
   (()()()())
   ```

   ```
   ((((()))))
   ```

   These are unbalanced:

   ```
   (((((((())
   ```

   ```
   ()))
   ```

   Implement `isBalanced(String inString)` which returns a boolean **true** if the parentheses string is balanced and **false** if it is not, using a stack

Part II: Sets

The reason we like Sets in Java is because they help us think about Sets in a mathematical sense and we can easily implement the functions of Sets that exist in Math—eg, Set intersections and

unions. In Python, these set functions are explicit. In Java, they are not, something which I briefly forgot during lecture.

Create a class `MathSet` which extends `HashSet`. It should have three methods:

`public Set intersection(Set s2)`: Takes a Set, s2, and returns the intersection of the Set and s2—the elements that are in both sets.

`public Set union(Set s2)`: Takes a Set, s2, and returns the union of the Set and s2—the combination of all elements.

```
public Set<Pair<T,S>> cartesianProduct(Set s2)
```

I have provided a Pair class for this. Return the Cartesian Product of the base set, s and s2: s × s2:

A **Cartesian product** of two sets A and B, written as A×B, is the set containing **ordered** pairs from A and B. That is, if C=A×B, then each element of C is of the form (x,y)
where x∈A and y∈B:

A×B={(x,y)|x∈A and y∈B}.

For example, if A={1,2,3} and B={H,T}, then

A×B={(1,H),(1,T),(2,H),(2,T),(3,H),(3,T)}

Note that here the pairs are ordered, so for example, (1,H)≠(H,1). Thus A×B is **not** the same as B×A.

Part III: Maps

I've created an ArrayList of 100 random integers from 0 to 9. Using maps, have the method sortByFrequency sort the ArrayList according to the ascending frequency of the occurrence of the value in the array list. For example, if the Array list contains:

[1, 2, 2, 1, 1, 1, 5]

The sorted result would be:
[5, 2, 2, 1, 1, 1, 1]

Because "5" occurs 1 time, "2" occurs 2 times, and "1" occurs 4 times.

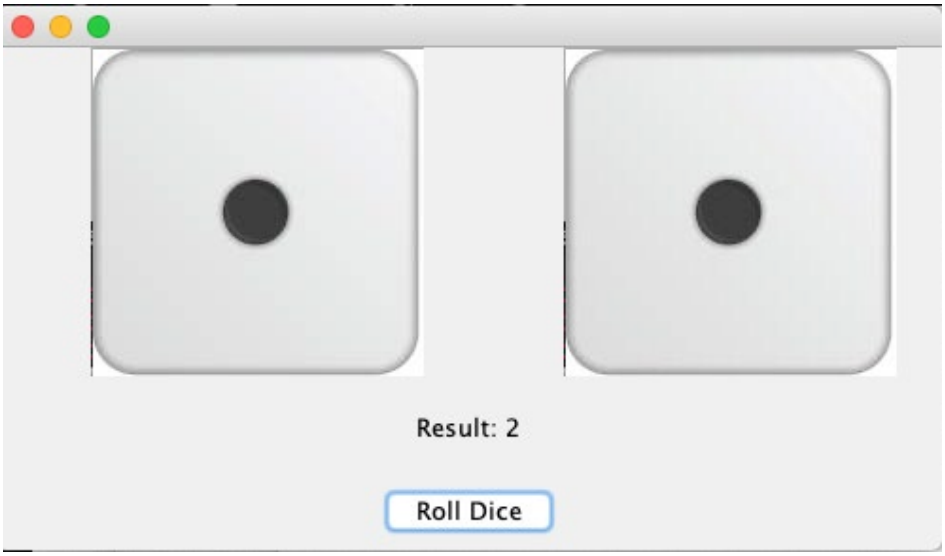
Use a map to solve this problem.

You may get some numbers that aren't "sorted". Explain why.

1 point of extra credit if you come up with a way to fix this.

Part IV: Graphics

You are going to create a “roll the dice” application. It will look like this:



The way it works is that when you click on the “Roll Dice” button, it will cycle through the dice images a random number of times until landing on the final one, which will have the total of the two dice. At that point, it will show the sum of the dice. If you click on ONE die, it will “roll” that one and then update the total.

Hints: Since there’s little “skeleton” code available, modify the “ImagePanel” code. The ActionListeners/MouseListeners will update the image in the ImagePanel and repaint() a random number of times before settling on the final value.