CS61BL: Data Structures & Programming Methodology Summer 2014

Instructor: Edwin Liao

Midterm 1

July 9, 2014

Name:	
Student ID Number:	
Section Time:	
TA:	
Course Login: cs61bl-??	
Person on Left: Possibly "Aisle" or "Wall"	
Person on Right: Possibly "Aisle" or "Wall"	

- Fill out ALL sections on this page. (1 point)
- Do NOT turn this page until the beginning of the exam is announced.
- You should not be sitting directly next to another student.
- You may not use outside resources other than your 1 page cheat sheet.
- You have 110 minutes to complete this exam.
- Your exam should contain 6 problems over 14 pages, including the reference sheet.
- This exam comprises 15% of the points on which your final grade will be based (45 points).
- If you have a question, raise your hand and a staff member will come to help you.
- Make sure to check for corrections / clarifications that will be periodically added to the board at the front of the room.
- Best of luck. Relax – this exam is not worth having a heart failure over.

Reference Sheet: `String`

Here are some methods and descriptions from Java's `String` class API that you may find useful.

Return type and signature	Method description
`char charAt(int index)`	Returns the `char` value at the specified index.
`int indexOf(String str)`	Returns the index within this string of the first occurrence of the specified substring.
`int indexOf(String str, int fromIndex)`	Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
`int length()`	Returns the length of this string.
`String replaceAll(String regex, String replacement)`	Replaces each substring of this string that matches the given regular expression with the given replacement.
`String[] split(String regex)`	Splits this string around matches of the given regular expression.
`String subString(int beginIndex)`	Returns a new string that is a substring of this string.
`String subString(int beginIndex, int endIndex)`	Returns a new string that is a substring of this string.

1 Equality Checks (6 points)

Write a `probablyEquals` method that takes in two objects and returns `true` if one or more of the following are true:

- The two objects are equal (`.equals`) to each other.
- The two objects are equal (==) to each other.
- The two objects have the same `.toString()` representation.
- Calling `.hashCode()` on both objects returns the same `int`.

Otherwise, `probablyEquals` returns `false`. Your method should never crash given *any* input. You may assume that for any object instances `x` and `y`, `x.equals(y)` will return the same value as `y.equals(x)`.

Note: `hashCode()` is a method that returns an `int`. All objects inherit this method from the `Object` class. For the purposes of this problem, `toString()` is another method that all objects inherit from the `Object` class and that returns a `String` representation of the object.

```
public static boolean probablyEquals( Object obj1, Object obj2 ) {










}
```

2 Whose `Line` is it Anyway? (4 points)

Examine the following implementations of a `Point` class and `Line` class:

```java
public class Point {
    private int myX;
    private int myY;

    public Point(int inputX, int inputY) {
        this.myX = inputX;
        this.myY = inputY;
    }

    public int getX() {
        return this.myX;
    }

    public int getY() {
        return this.myY;
    }
}
```

```java
public class Line {
    private Point myP1;
    private Point myP2;

    public Line(Point p1, Point p2) {
        this.myP1 = p1;
        this.myP2 = p2;
    }

    public String toString() {
        String toRtn = "";
        toRtn += ("(" + myP1.getX() + "," + myP1.getY() + "),");
        toRtn += ("(" + myP2.getX() + "," + myP2.getY() + ")");
        return toRtn;
    }

    public static void main(String[] args) {
        Point p1 = new Point(1, 2);
        Line myLine = new Line(p1, p1);
        // Your code here
        System.out.println(myLine);
    }
}
```

(a) In the space below, draw the resulting box-and-arrow (a.k.a. box-and-pointer) diagram after executing lines 18 and 19 of the `Line` class (up until the `// Your code here` line).

(b) In the space below, rewrite the `// Your code here` with a single line of code so that the program prints out:

```
(1,2),(3,4)
```

3 Building a Knapsack (12 points)

The following code represents a knapsack that can carry items and keep track of the total weight of the items it contains:

```
import java.util.ArrayList;

public class Knapsack {
    protected ArrayList<String> itemNames;
    private ArrayList<Integer> itemWeights;
    private final int weightCapacity;
    private int totalWeight;

    public Knapsack(int weightCapacity) {
        itemNames = new ArrayList<String>();
        itemWeights = new ArrayList<Integer>();
        this.weightCapacity = weightCapacity;
        this.totalWeight = 0;
    }

    public void addItem(String name, int weight) {
        itemNames.add(name);
        itemWeights.add(new Integer(weight));
    }

    public void removeItem(String name) {
        int itemIndex = itemNames.indexOf(name);
        itemNames.remove(itemIndex);
        itemWeights.remove(itemIndex);
    }

    public int getTotalWeight() {
        return totalWeight;
    }
}
```

Example of usage (after you implement part a):

```
Knapsack myKnapsack = new Knapsack(5);
myKnapsack.addItem("Banana", 1);
myKnapsack.addItem("Water Bottle", 3);
myKnapsack.getTotalWeight(); // should return 4
myKnapsack.removeItem("Banana");
myKnapsack.getTotalWeight(); // should return 3
```

(a) Add code to the `Knapsack` class so that after every method call, the `totalWeight` instance variable always equals the total weight of all of the items in the `Knapsack`. You may or may not have to use all of the boxes below.

Code added immediately after line _______:

Code added immediately after line _______:

Code added immediately after line _______:

(b) Add code to the `Knapsack` class so that `Knapsack`s will never:

- contain two items with the same name
- have a total weight greater than its weight capacity
- contain an item with negative weight
- have a negative capacity

If a user tries to modify a `Knapsack` to have any of the above properties, an `IllegalStateException` (from `java.lang`) should be thrown with an informative error message. Assume that the code from part (a) has been implemented correctly. You may or may not have to use all of the boxes below.

Code added immediately after line _______:

Code added immediately after line _______:

Code added immediately after line _______:

(c) What is another case of error checking we should add to `Knapsack`?

(d) Fill out the following template for a `ValueKnapsack` class that keeps track of items' values in addition to their weights. You should use inheritance effectively. Don't worry about error-checking for this part. Assume that parts (a) and (b) have been implemented correctly.

```
public class ValueKnapsack extends Knapsack {

    private int totalValue;
    private ArrayList<Integer> itemValues;

    public ValueKnapsack(int weightCapacity) {




    }

    public void addItem(String name, int weight, int value) {








    }

    public void removeItem(String name) {








    }

    public int getTotalValue() {
        return totalValue;
    }
}
```

4 Acronym Extractor (7 points)

We want to code an `extractAcronym` method that returns the acronym of an input `String`. We will be using the `#` character in place of whitespace characters for this problem (you may assume that the input will not have any whitespace characters). An acronym consists of the first non-`#` letter of the input `String` and all non-`#` characters that immediately follow any `#` character. For example, `extractAcronym("Not#a##Number")` would return `"NaN"`. If there are `#` characters at the end of an input `String`, we ignore them.

(a) In the table below, provide test inputs for `extractAcronym` that cover at least 4 generalized input cases (not including the provided example). Include the input `String`, expected output `String`, and what your test case is testing for. Do not reuse the example. Assume that the input and return value will never be `null`.

Input `String`	Expected output	What are you testing for?
"Not#a##Number"	"NaN"	(The example) Tests that `extractAcronym` can handle cases with multiple consecutive # characters.

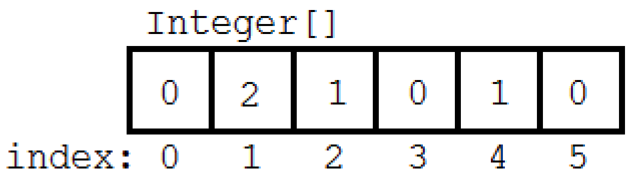
(b) Implement the `extractAcronym` method:

```
public static String extractAcronym( String input ) {

}
```

5 Vote Iterator (10 points)

Write an iterator that takes in an `Integer` array of vote counts and iterates over the votes. The input array contains the number of votes each selection received. For example, if the input array contained the following:



then calls to `next()` would eventually return 1 twice (because at index 1, the input array has value 2), 2 once, and 4 once. After that, `hasNext()` would return `false`.

Provide code for the `VoteIterator` class below. Make sure your iterator adheres to standard iterator rules.

```
import java.util.Iterator;



public class VoteIterator implements Iterator<Integer> {



    public VoteIterator(Integer[] votes) {




    }
    }

    public boolean hasNext() {








    }

    public Integer next() {








    }

    public void remove() {








    }
}
```

6 Malicious Mallory (5 points)

Eve and Mallory are lab partners in CS61BL. Unfortunately for Eve, Mallory doesn't get along with anyone. One day, when Eve isn't looking, Mallory codes the following method and adds calls to it in Eve's code:

```java
import java.util.ArrayList;

public void method() {
    ArrayList a = new ArrayList();
    String msg1 = "Code not working? Try turning your computer off and on again!";
    String msg2 = "Is your code running? Then you better go catch it!";
    String msg3 = "You might have forgotten a semicolon somewhere.";
    a.add(new IllegalArgumentException(msg1));
    a.add(new ArrayIndexOutOfBoundsException(msg2));
    a.add(new NumberFormatException(msg3));
    int index = (int) (Math.random() * 10);
    if (index >= 3) return; throw a.get(index);
}
```

This code is meant to do nothing 70% of the time, and throw one of three random Exceptions the other 30% of the time. Note: `Math.random()` returns a random `double` in the range $[0, 1)$.

(a) There is an error with this code. Explain what the error is, and whether it is a compile time error, a runtime error, or a logic error.

(b) Cross out one line of code above. Rewrite this line of code below so that the code compiles, runs, and does what Mallory intended.