CS 61B Spring 2023

Midterm 1

Thursday February 9, 2023

Intro: Welcome to CS61B Midterm 1!

Your name: Solutions

Login: sp23-s

Your SID: _____

Location: _____

[In Person] Login of person to your Left: sp23-s Right: sp23-s

Write the statement *"I have neither given nor received any assistance in the taking of this exam."* below.

_____

_____

Signature: _____

This exam is out of 4800 points. There are a few code writing questions. You may not need to use all lines provided. If you are given skeleton code to fill in, **we will not grade your answer if you alter the skeleton code or write outside the solution box**. If you are given a line limit, **we will not grade your answer if you go over the line limit or if you don't properly format your code in the attempt of using fewer lines.** For instance, the properly formatted code below takes 9 lines, counting the function signature and closing brace.

```
1  public static boolean exampleFunction() {
2      for (int i = 0; i < 10; i += 2) {
3          if (i == 2) {
4              i -= 1;
5          } else {
6              i += 1;
7          }
8      }
9  }
```

You may not use ternary operators, lambda functions, or streams. You may assume the data structures defined in the reference sheet have been imported and implement the given interfaces. You may only use methods specified in the reference sheet. Additionally, **you will not receive credit if you fail to follow any restrictions given in the problem statement**. Your code won't be checked by a compiler, but we will take off points for errors that are more than a typo.

F1Car

(300 Points)

```
1  public class F1Car {
2      private String driver;
3      private static String[] tracks;
4      private static int trackIndex;
5
6      public F1Car(String driver, String[] tracks, int trackIndex) {
7          this.driver = driver;
8          F1Car.tracks = tracks;
9          this.trackIndex = trackIndex;
10     }
11
12     public static String nextTrack() {
13         return tracks[trackIndex];
14     }
15
16     public String getDriver() {
17         return driver;
18     }
19
20     public static void main(String args[]) {
21         F1Car ferrari = new F1Car("Dylan", new String[]{"Silverstone", "Spa"}, 0);
22     }
23 }
```

Answer the questions below.

(a) Will calling `F1Car.getDriver()` error? ● **Yes** ○ No

(b) Will calling `F1Car.nextTrack()` error? ○ Yes ● **No**

(c) Will calling `ferrari.nextTrack()` error? ○ Yes ● **No**

(d) Suppose we modify the `getDriver()` method in line 16 so it reads **`public static`** `String getDriver()`. Will calling `ferrari.getDriver()` error? ● **Yes** ○ No

Login:

StudentID

(500 Points)

Suppose the university wants to write a Class that represents a student's ID card. A StudentID card has the following attributes:

- `name`: the name of the student (which is a String)
- `idNumber`: the student's ID number (which is an int)
- `buildingAccess`: A data structure containing all the **unique** building names a **specific** student has access to. You should decide the specific data structure yourself based on the supported methods, and initialize it to be empty.
- `studentDir`: a directory that maps `idNumber` to `name` that gets updated whenever a new student gets added. All students share the same directory.

On top of these attributes, we want to support a few methods, described below:

- `Constructor`: creates a new instance of a `StudentID`
- `getStudentName`: Given an `idNumber`, return the name. If the given `idNumber` doesn't exist, you should return an empty string `""`.

For example, here is the class running with the code below:

```
1  StudentID s1 = new StudentID("Eric", 1);
2  StudentID s2 = new StudentID("Eddie", 2);
3  System.out.println(StudentID.getStudentName(1)); // Should print out "Eric"
4  s1.buildingAccess.add("Soda Hall");
5  s1.buildingAccess.add("Cory Hall");
6  s1.buildingAccess.add("Soda Hall"); //s1 should have access to two buildings
7                                        (Soda Hall and Cory Hall)
```

Implement the `StudentID` class on the next page. You may use methods from the reference sheet.

```java
public class StudentID {

    public String name;
    public int idNumber;

    public Set<String> buildingAccess;
    public static Map<Integer, String> studentDir = new HashMap<>();

    public StudentID(String name, int idNumber) {
        this.name = name;
        this.idNumber = idNumber;
        studentDir.put(idNumber, name);
        buildingAccess = new HashSet<>();
    }


    public static String getStudentName(int idNumber) {
        if (studentDir.containsKey(idNumber)) {
            return studentDir.get(idNumber);
        }
        return "";
    }
}
```

Login:

Stars

(500 Points)

Write out the output of the print statements. You may assume there are no errors with this code.

```
1  public class Star {
2      public static String glow;
3      public static int luminosity = 100;
4      public String light;
5
6      public Star(String color) {
7          light = color;
8          glow = color;
9          luminosity += 100;
10     }
11
12     public void shineBright(String st, Star s) {
13         Star stella = new Star(this.light);
14         this.light = st;
15         System.out.println(s.glow);
16     }
17
18     public static void main(String[] args) {
19         Star sun = new Star("pink");
20         System.out.println(sun.light);       // blank (a)
21         System.out.println(Star.luminosity); // blank (b)
22         Star polaris = new Star("yellow");
23         System.out.println(polaris.glow);    // blank (c)
24         System.out.println(Star.glow);       // blank (d)
25         sun.shineBright("blue", polaris);    // blank (e)
26         System.out.println(sun.light);       // blank (f)
27         System.out.println(Star.luminosity); // blank (g)
28     }
29 }
```

(a) pink

(b) 200

(c) yellow

(d) yellow

(e) pink

(f) blue

(g) 400

61Bomberman

(700 Points)

61Bomberman has laid bombs inside the **`int[][] world`**, in the form of -1s inside the 2-D array! All bombs get triggered simultaneously, and destroy the terrain in their row and column, (all numbers in the same row or the same column as a bomb, including the bomb itself, get set to 0). Fill in the method `resultingWorld` that **modifies** the original world to represent the world after all bombs have been triggered. You can assume that all subarrays are the same length, and that the world has at least one row and column.

For instance, consider the following **`int[][] world`** before and after a call to `resultingWorld`:

```
[[-1, 1,-1, 8, 7],      [[ 0, 0, 0, 0, 0],
 [ 3, 6, 1, 2, 4],  →    [ 0, 6, 0, 2, 4],
 [ 2, 3, 3, 5, 9]]       [ 0, 3, 0, 5, 9]]
```

In this example, there are bombs in coordinates (0, 0) and (0, 2). As a result, row 0 (the topmost row) is populated with 0s and columns 0 (the leftmost column) and 2 are populated with 0s.

```
[[-1, 1, 4],     [[ 0, 0, 0],
 [ 3, 5, 1],  →   [ 0, 5, 0],
 [ 2, 8, 3],      [ 0, 8, 0],
 [ 6, 2,-1]]      [ 0, 0, 0]]
```

In this example, there are bombs in coordinates (0, 0) and (3, 2). As a result, rows 0 and 3 are populated with 0s and columns 0 and 2 are populated with 0s.

Implement the `resultingWorld` method on the next page. You may use methods from the reference sheet.

Login:

```java
public void resultingWorld(int[][]world) {
    Set<Integer> rows = new HashSet<>();
    Set<Integer> cols = new HashSet<>();

    //Find the bombs
    for (int x = 0; x < world.length; x += 1) {
        for (int y = 0; y < world[x].length; y += 1) {
            if (world[x][y] == -1) {
                rows.add(x);
                cols.add(y);
            }
        }
    }

    //Trigger the bombs
    for (int x = 0; x < world.length; x += 1) {
        for (int y = 0; y < world[x].length; y += 1) {
            if (rows.contains(x) || cols.contains(y)) {
                world[x][y] = 0;
            }
        }
    }
}
```

getEveryNth

(700 Points)

In Python, we can conveniently get an array of every Nth element using the "slicing" syntax `arr[::n]`, where `n` is the "step size" of the number of elements we iterate over. For example, if `arr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`, `arr[::2] = [0, 2, 4, 6, 8]`, `arr[::4] = [0, 4, 8]`, `arr[::11] = [0]`. In this question, `n` will always be a positive integer greater than 0.

We want to try to replicate this behavior in Java with our LinkedList and ArrayList classes. Fill out the code below for the two different implementations of `getEveryNth`, which is defined as follows:

- Input: A positive integer n.
- Output: A new ArrayList whose elements are the `0`th, `n`th, `2n`th, ... elements of the input List. You may NOT assume that the size of the input List is a multiple of `n`.
- Your code may NOT modify the input List.

For the purposes of this question, you may NOT use the get method of any List. You may, however, use any other functions defined as part of the List interface in the reference sheet.

Hint: the modulo operator might be useful for this question. The modulo operator returns the remainder when dividing. Some examples are attached below:

```
15 % 8 = 7 // because 15 / 8 = 1 with remainder 7
0 % 9 = 0
4 % 4 = 0
```

Part A:

```
public class ArrayList<T> implements List<T> {
    private T[] array;
    private int size;

    ...//More code that implements an Array List

    // As a reminder, you may not use the .get method in this question
    public List<T> getEveryNth(int n) {
        List<T> result = new ArrayList<>();

        for(int i = 0; i < this.size; i += n) {
            result.add(array[i]);
        }
        return result;
    }
}
```

Part B:

Login:

```
public class LinkedList<T> implements List<T> {
    private Node sentinel;
    private int size;

    private class Node {
        private T value;
        private Node next, prev;
        Node(T value) {
            this.value = value;
        }
    }
    ... //More code that implements a Linked List

    // As a reminder, you may not use the .get method in this question
    public List<T> getEveryNth(int n) {
        List<T> result = new ArrayList<>();

        Node node = sentinel.next;
        int counter = 0;

        while (node != sentinel) {
            if (counter == 0) {
                result.add(node.value);
            }

            counter = (counter + 1) % n;
            node = node.next;
        }
        return result;
    }
}
```

Part C: The type of data structure used can have a big impact on the implementation of various default methods we write. Not only does this affect the complexity of the logic, it will also impact how quickly programs run. Imagine you run the `getEveryNth` method on a List of size 9999991 getting every 99991st element (the exact values here do not matter, except that they are large).

Which implementation (LinkedList vs ArrayList) would be faster and why? Please provide a sentence of up to 20 words with the most important difference.

ArrayList is faster to iterate through because elements can be skipped.

Testception

(800 Points)

We want to write a method, `numWords(String str)` so that can count the number of words in a `String`. We define a word as a `String` of 1 or more consecutive English alphabet letters. A word does not necessarily have to be a valid word in the English language. For example,

- "Apple", "A", and "aVeryInvalidWord" are all counted as 1 word each
- "A-tree" and "cat4dog" are counted as 2 words
- "1234" and "-" are counted as 0 words.

We are given a method, `isAlpha(char c)`, that returns a **boolean** signifying if a character is an English alphabet letter.

```
public boolean isAlpha(char c) { /* implementation not shown */ }
```

We give the following examples of using `isAlpha` and the corresponding return value. You may assume `isAlpha` is free of any bugs and works as intended.

```
isAlpha('a'); // true
isAlpha('A'); // true
isAlpha('!'); // false
isAlpha(' '); // false
isAlpha('3'); // false
```

(a) We made our first attempt at writing the method, and came up with the following implementation. The following implementation is buggy:

```
1  public static int numWords(String str) {
2      int words = 0;
3      for (int i = 1; i < str.length(); i++) {
4          if (isAlpha(str.charAt(i-1)) && !isAlpha(str.charAt(i))) {
5              words += 1;
6          }
7      }
8      return words;
9  }
```

In order to test our code, we have written the following tests. Select all the tests that will fail using the above implementation. You may assume that all assertion statements are correct.

- [x] `assertThat(numWords("Count the words")).isEqualTo(3);`
- [ ] `assertThat(numWords("Count the words!")).isEqualTo(3);`
- [x] `assertThat(numWords("ThisIsOneLongWord")).isEqualTo(1);`
- [ ] `assertThat(numWords("The value of pi is 3.14")).isEqualTo(5);`
- [ ] `assertThat(numWords("6.25")).isEqualTo(0);`
- [x] `assertThat(numWords("A")).isEqualTo(1);`
- [ ] `assertThat(numWords("")).isEqualTo(0);`

Login:

(b) We could not figure out the bug, so we scrapped the code in part (a). Now, it's up to you to write the correct implementation! Select the following implementation so that it correctly counts the number of words without any bugs.

```
1  public static int numWords(String str) {
2      int words = 0;
3      boolean inWord = _______(1)_______;
4      for (char c : str.toCharArray()) {
5          if (_______(2)_______) {
6              _______(3)_______;
7              words += 1;
8          } else if (_______(4)_______) {
9              _______(5)_______;
10         }
11     }
12     return words;
13 }
```

(i) Fill in blank (1).

- ○ **true**
- ● **false**

(ii) Fill in blank (2).

- ○ `!inWord && !isAlpha(c)`
- ○ `inWord && !isAlpha(c)`
- ● **`!inWord && isAlpha(c)`**
- ○ `inWord && isAlpha(c)`

(iii) Fill in blank (3).

- ● **`inWord = true`**
- ○ `inWord = false`

(iv) Fill in blank (4).

- ○ `!inWord && !isAlpha(c)`
- ● **`inWord && !isAlpha(c)`**
- ○ `!inWord && isAlpha(c)`
- ○ `inWord && isAlpha(c)`

(v) Fill in blank (5).

- ○ `inWord = true`
- ● **`inWord = false`**

TwoWayStreet

(600 Points)

Crystal and Jedi have been hired to turn the infamously chaotic one-way portion of Telegraph Avenue into a two-way street, and it's turning out to be a logistical nightmare! Can you help them out?

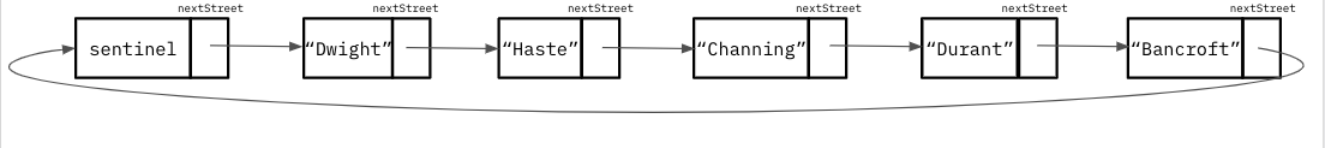
```
public class TelegraphAve {
    public CrossStreet sentinel;

    public TelegraphAve() {
        this.sentinel = new CrossStreet();
    }

    private static class CrossStreet {
        String name;
        CrossStreet nextStreet;
    }
}
```

Assume that `TelegraphAve` is represented by a singly-linked list with a circular sentinel; the sentinel's `nextStreet` represents the first cross street on Telegraph Avenue, and the `nextStreet` of the last cross street in `TelegraphAve` is the sentinel itself.

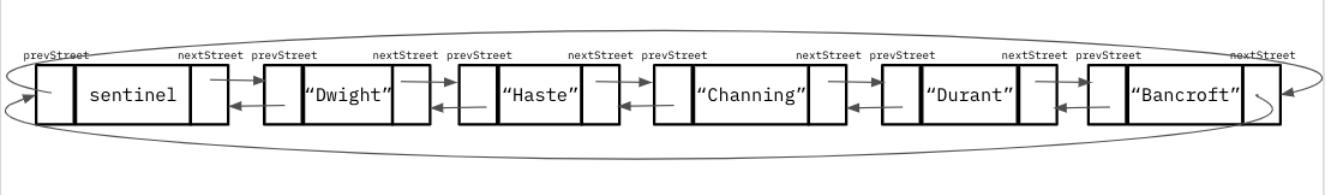
The architects' proposed change is to make every cross street on Telegraph have *both* a `nextStreet` and `prevStreet`, like so:

```
private static class CrossStreet {
    String name;
    CrossStreet nextStreet; // already exists
    CrossStreet prevStreet; // new
}
```

We already have an existing one-way street through the `nextStreet` chaining of `CrossStreet`s, so we just need to add `prevStreet` connections. Before calling `addPrevStreets`, `prevStreet` is set to **`null`** for all cross streets, as in the below example:



After calling `addPrevStreets`, `TelegraphAve` should now look like:



Login:

Implement `addPrevStreets` and `addPrevStreetsHelper` below:

```
public class TelegraphAve {
    public CrossStreet sentinel;

    // constructor and updated CrossStreet definition
    ...

    public void addPrevStreets() {
        addPrevStreetsHelper(sentinel);
    }

    private void addPrevStreetsHelper(CrossStreet curr) {
        if (curr.nextStreet.prevStreet != null) {
            return;
        }
        curr.nextStreet.prevStreet = curr;
        addPrevStreetsHelper(curr.nextStreet);
    }
}
```

Family Trees

(700 Points)

```
public interface TreeLike {
    default boolean isFriendly() {
        return true;
    }
}

public interface Needly {
    int numNeedles();
}

public class Pine implements TreeLike, Needly {
    public int numNeedles;

    public Pine(int numNeedles) {
        this.numNeedles = numNeedles;
    }

    @Override
    public int numNeedles() {
        return numNeedles;
    }
}

public class SyntaxTree implements TreeLike {
    public int branches;

    public SyntaxTree(int branches) {
        this.branches = branches;
    }

    @Override
    public boolean isFriendly() {
        return branches < 5;
    }

    public static int numNeedles() {
        return -1;
    }
}
```

Login:

Given the interfaces and classes above, answer the following questions. For each of the parts, choose all that apply.

(a) Which of the following lines would compile (i.e. no error)?

- ☐ `Needly n1 = new Needly();`
- ■ `SyntaxTree s1 = new SyntaxTree(85);`
- ☐ `Pine p1 = new Needly();`
- ■ `Needly n2 = new Pine(4);`
- ☐ `Pine p2 = new SyntaxTree(1);`
- ☐ `Needly n3 = new SyntaxTree(37);`

(b) Given the following variables, which expressions would print `-1`?

```
Pine ponderosa = new Pine(3);
TreeLike coulter = new Pine(5);
SyntaxTree chomsky = new SyntaxTree(-1);
TreeLike pinker = new SyntaxTree(8);
Needly weird = new Pine(-1);
```

- ■ `System.out.println(weird.numNeedles());`
- ☐ `System.out.println(Needly.numNeedles());`
- ■ `System.out.println(chomsky.numNeedles());`
- ■ `System.out.println(SyntaxTree.numNeedles());`

(c) Say we have the short program below:

```
Random r = new Random();
TreeLike t = ?;
boolean tIsFriendly = t.isFriendly();
```

Which expressions could go in the `?` spot such that `tIsFriendly` is *guaranteed* to be **`true`**?

- ■ `new Pine(r.nextInt(94));`
- ☐ `new SyntaxTree(100);`
- ☐ `new SyntaxTree(r.nextInt(6));`
- ■ `new SyntaxTree(r.nextInt(5));`

Feedback and Fun

(0 Points)

Congratulations for making it to the end of the exam! Feel free to leave any final thoughts, comments, feedback, or doodles here:

I hope you are having a wonderful day!