CSE 122 23au Final Exam Answer Key

1. **Conceptual:** Each of these parts should be considered independent of the others.

Part A: (Select all that apply) Which of the following are true about Sets?

- ☐ A TreeSet stores elements in sorted order
- ☐ A HashSet preserves the insertion order of its elements
- ☐ A Set does not allow for duplicate values
- ☐ A HashSet is generally more efficient than a TreeSet

Part B: Consider the following method. For each of the following commented checkpoints, fill in the table for which conditions are always true (under any circumstance), only sometimes true, or never true at each comment. You can abbreviate **A**=always, **S**=sometimes and **N**=never.

```
public Set<Integer> mystery1(List<Integer> list, int num) {
    Set<Integer> result = new TreeSet<>();
    int n = 1;
    // Checkpoint A
    if (list.isEmpty()) {
        throw new IllegalArgumentException();
    }
    // Checkpoint B
    for (int i = 0; i < list.size(); i++) {
        n = list.get(i);
        // Checkpoint C
        if (n != 0 && num % n == 0) {
            list.remove(i);
            i--;
            result.add(n);
            // Checkpoint D
        }
    }
    // Checkpoint E
    return result;
}
```

An explanation for Checkpoint A's example answers

list.isEmpty(): **S**
true for mystery([], 3) and false for mystery([1,2,3], 4). Therefore sometimes true.

num % n == 0: **A**
Any number mod 1 is 0, so this is always true.

result.isEmpty(): **A**
result was just initialized and nothing has been added yet. Therefore it is always empty here.

	Checkpoint A	Checkpoint B	Checkpoint C	Checkpoint D	Checkpoint E
list.isEmpty()	**S**	N	N	S	S
num % n == 0	**A**	A	S	A	S
result.isEmpty()	**A**	A	S	N	S

CSE 122 23au Final Exam Answer Key

Part C: (Select one option) Consider the following method. Which of the following options is the best "plain-English" explanation of what the code is doing?

```
public static Set<Integer> mystery2(Map<Integer, Set<Integer>> m) {
    Set<Integer> result = new HashSet<>();
    for (int num : m.keySet()) {
        Set<Integer> val = m.get(num);
        int sum = 0;
        for (int num2 : val) {
            sum += num2;
        }
        if (sum == num) {
            result.add(num)
        }
    }
    return result;
}
```

◯ Returns a set containing a subset of the keys of m

◯ Returns a set containing the sums of the inner sets in m

◯ Returns a set of the keys of m that equal the sum of their inner sets in m

◯ Uses a for-each loop to look at each key from m to sum the numbers in the inner sets, adding the key to a set if the sum equals the associated key

CSE 122 23au Final Exam Answer Key

2. **Code Tracing:** Consider the method below.

```
public static Set<Integer> superSecretMystery(int[][] arr) {
    Set<Integer> result = new TreeSet<>();
    for (int i = arr.length - 1; i >= 0; i--) {
        for (int j = 0; j < arr[i].length; j++) {
            result.add((j-i) * 10 + arr[i][j]);
        }
    }
    return result;
}
```

For each 2d array below, indicate the contents of the Set that superSecretMystery would return where it is passed as a parameter. Write the elements of the Set comma-separated with square brackets (for example: [1, 2, 0, 5])

Input	**Returned Set**
`[[3]]`	`[3]`

`[[7],`
` [7],`
` [7]]`
	`[-13, -3, 7]`

`[[2, 0, 4],` ` [12, 10, 0]]`	`[2, 10, 24]`
`[[5, -5, 8, 0, 6]]`	`[5, 28, 30, 46]`

CSE 122 23au Final Exam Answer Key

3. **Debugging:** Consider the following buggy implementation of **removeEvensInRange**. The intended behavior of this method is to take a list of integers, as well as integers start and end, and modify that list so that it removes the even numbers between indices start and end (exclusive). The method should also return the number of elements that were removed. For example, if a variable called list stores this sequence of values:

```
[3, -1, 0, 2, 5, -10, 8]
```

Calling removeEvensInRange(list, 1, 5); should cause list to store the following sequence of values afterwards:

```
[3, -1, 5, -10, 8]
```

Notice that the order from the original list is maintained. Assume that the given list is not empty and the inputs for start and end are valid indices of the original list.

A TA wrote a buggy implementation of this method shown below.

```
1.  public static int removeEvensInRange(List<Integer> list, int start, int end) {
2.      int count = 0;
3.      for (int i = start + 1; i < end; i++) {
4.          int num = list.get(i);
5.          if (num % 2 == 0) {
6.              count++;
7.              list.remove(i);
8.              i++;
9.              end--;
10.         }
11.     }
12.     return count;
13. }
```

Part A: There is a single bug in this program that is your task to find and fix. Identify the *1 line of code* that causes the bug. Write your answer as a line number in the box to the right. For example, if you think line 13 has a bug, write **13** in the box.

Part A Answer 8

Part B: Fix the error in the method above. Since there is only one bug, this should not take a lot of code to fix. Specifically mention which line(s) you will change and how. If you are deleting some code, make sure it's clear what parts are being removed. If you are inserting new code, make sure it is unambiguous where this new code belongs. Mention specific line number(s). Write your answer for **Part B** in the box below.

Should be: `i--;`

4. **Collections Programming**: Write a method called **mostPopularHobby** that takes in a Map with keys that are TA names and values that are sets of hobbies of that TA. For example, if a variable called m contains the following:

```
{
 Atharva = [board games, comedy shows, hiking, video gaming],
 Chaafen = [Formula 1, reading, traveling, video gaming],
 Jaylyn = [hiking, traveling, video gaming],
 Shivani = [cafes, music shows]
}
```

then the call mostPopularHobby(m) should return the String "video gaming" because three of the TAs enjoy this hobby while the other hobbies are less common.

If there is a tie between hobbies, you should break them alphabetically. For instance, if Shivani had an additional hobby of hiking, then the method should return "hiking" since it would be enjoyed by three TAs and comes alphabetically before "video gaming".

You may assume that the given map is not empty and that none of the inner sets are empty.

Your method should not construct any new data structures other than a single Map. It should also not modify the input map or any of the inner sets. You should use interface types and generics appropriately.

```
public static String mostPopularHobby(Map<String, Set<String>> m) {
    Map<String, Integer> counts = new TreeMap<>();

    for (String name : m.keySet()) {
        Set<String> hobbies = m.get(name);
        for (String h : hobbies) {
            if (!counts.containsKey(h)) {
                counts.put(h, 0);
            }
            counts.put(h, counts.get(h) + 1);
        }
    }

    int max = 0;
    String hobby = "";
    for (String h : counts.keySet()) {
        if (counts.get(h) > max) {
            max = counts.get(h);
            hobby = h;
        }
    }
    return hobby;
}
```

CSE 122 23au Final Exam Answer Key

5. **Objects Programming:** Consider the following interface Team. For this problem you are to write a class called RelayTeam which implements the Team interface and represents a team of people on a relay racing team and their times for running 400m. The RelayTeam should have a constructor which takes one String parameter, the team's mascot.

```
public interface Team {
    // Returns the mascot of this team.
    public String getMascot();

    // Adds a runner to this relay team with the given time in seconds.
    // Throws an IllegalArgumentException if the runner is already on the team
    public void addRunner(String runner, double time);

    // Substitutes newRunner for oldRunner on the relay team with newTime
    // Throws an IllegalArgumentException if oldRunner is not on the team or
    // if newRunner is already on this team, or if newTime is negative.
    public void substituteRunner(String oldRunner, String newRunner, int newTime);

    // Returns the name of the runner on this team that is the fastest (shortest time).
    // If two runners are tied for fastest, this method should return the name that
    // is alphabetically first.
    // Throws an IllegalStateException if there are no runners on this team.
    public String getFastestRunner();

    // Returns the average running time across all runners on the team,
    // or 0 if there are no runners.
    public double getAverageTime();

    // Returns true if this team has a faster average time than the given other team,
    // and false otherwise.
    public boolean hasFasterAverage(Team other);

    // Returns a string representation of this team. The format should be
    // as follows:
    //     <team mascot>'s average time: <average time> s
    // Eg, for the Cheetahs with an average time of 52 seconds,
    // the resulting toString would look like:
    //     Cheetahs's average time: 52 s
    // You do not need to round the resulting average
    public String toString();
}
```

For example, if the following lines were executed using your RelayTeam class…

```
    Team team1 = new RelayTeam("Sloths");
    team1.addRunner("Simon", 40);
    team1.addRunner("Subhash", 51);
    team1.substituteRunner("Subhash", "Samira", 40);

    Team team2 = new RelayTeam("Turtles");
    team2.addRunner("Thuy", 45);
    team2.addRunner("Tanya", 55);
```

Then the following method calls would return…

```
    team1.getMascot();                      // Sloths
    team1.getFastestRunner();               // Samira
    team2.getAverageTime();                 // 50
    team1.hasFasterAverage(team2);          // true
    team2.toString();                       // Turtles's average time: 50 s
```

Your RelayTeam class should implement the Team interface. Your RelayTeam class should have private fields and you should use interface types and generics appropriately.

Write your solution on the next page.

CSE 122 23au Final Exam Answer Key

```
import java.util.*

public class RelayTeam implements Team {
    private String mascot;
    private Map<String, Double> runners;

    public RelayTeam(String mascot) {
        this.mascot = mascot;
        runners = new TreeMap<>();
    }

    public String getMascot() {
        return mascot;
    }

    public void addRunner(String runner, double time) {
        if (runners.containsKey(runner) || time < 0) {
            throw new IllegalArgumentException();
        }
        runners.put(runner, time);
    }

    public void substituteRunner(String oldRunner, String newRunner, double newTime) {
        if (!runners.containsKey(oldRunner) ||
                                runners.containsKey(newRunner) || newTime < 0) {
            throw new IllegalArgumentException();
        }
        runners.remove(oldRunner);
        runners.put(newRunner, newTime);
    }

    public String getFastestRunner() {
        if (runners.size() == 0) {
            throw new IllegalStateException();
        }
        double fastestTime = Double.MAX_VALUE;
        String fastest = "";
        for (String runner : runners.keySet()) {
            double time = runners.get(runner);
            if (time < fastestTime) {
                fastestTime = time;
                fastest = runner;
            }
        }
        return fastest;
    }

    public double getAverageTime() {
        if (runners.size() == 0) return 0;

        double total = 0;
        for (String runner : runners.keySet()) {
            double time = runners.get(runner);
            total += time;
        }
        return total / runners.size();
    }

    public boolean hasFasterAverage(Team other) {
        return this.getAverageTime() < other.getAverageTime();
    }
```

```
    public String toString() {
        return mascot + "'s average time: " + this.getAverageTime() + " s.";
    }
}
```

6. **Stacks/Queues Programming:** Write a method called **alphabetize** that takes a queue of Strings as a parameter and modifies the queue to sort the Strings based on their first letter. We will assume that all Strings in the queue begin with either a, b, or c. For example, suppose a variable called q stores the following sequence of values:

front ["august", "cornelia street", "bejeweled", "cardigan", "afterglow"] back

and we make the following call:

```
alphabetize(q);
```

Then q should store the following values after the call:

front ["august", "afterglow", "bejeweled", "cornelia street", "cardigan"] back

Notice that the Strings that start with 'a' are at the front, followed by the Strings that start with 'b', followed by the Strings that start with 'c'. Also, notice that the queue is not fully alphabetized, rather, the ordering within a group is maintained. For example, "august" is before "afterglow" in both the original queue and the modified queue.

If the input queue is empty, then calling this method should not modify it.

For an E, your solution must obey the following restrictions. A solution that disobeys them may get an S, but it is not guaranteed.

* You may use one stack as auxiliary storage. You may not use other structures (arrays, lists, etc.), but you can have as many simple variables as you like.
* Use the Queue interface and Stack/LinkedList classes discussed in class.
* Use stacks/queues in stack/queue-like ways only. Do not use index-based methods such as get, search, or set, or for-each loops or iterators. You may call add, remove, push, pop, peek, isEmpty, and size.
* Do not use advanced material such as recursion to solve the problem.

You have access to the following two methods and may call them as needed to help you solve the problem:

```
public static void s2q(Stack<String> s, Queue<String> q) {
    while (!s.isEmpty()) {
        q.add(s.pop());
    }
}

public static void q2s(Queue<String> q, Stack<String> s) {
    while (!q.isEmpty()) {
        s.push(q.remove());
    }
}
```

You should write your solution in the box on the next page. If you need additional space, please indicate that your solution is continued on scratch paper.

CSE 122 23au Final Exam Answer Key

```
public static void alphabetize (Queue<String> q) {
    Stack<String> s = new Stack<>();
    // move strings that start w 'a' to stack
    int size = q.size();
    for (int i = 0; i < size; i++) {
       String str = q.remove();
       if (str.charAt(0) == 'a') {
           s.push(str);
       } else {
           q.add(str);
       }
    }
    // move strings that start w 'b' to stack
    size = q.size();
    for (int i = 0; i < size; i++) {
       String str = q.remove();
       if (str.charAt(0) == 'b') {
           s.push(str);
       } else {
           q.add(str);
       }
    }
    // move the rest to stack
    q2s(q,s);

    // reverse everything
    s2q(s,q);
    q2s(q,s);
    s2q(s,q);
}
```