National University of Singapore

SCHOOL OF COMPUTING

CS2030 — PROGRAMMING METHODOLOGY II
(Special Term Part II: AY2017/2018)

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This assessment paper consists of **FIVE(5)** questions and comprises **FIFTEEN(15)** printed pages, including this page.
2. Answer **ALL** questions. You may use pen or pencil to write your answers.
3. This is an **OPEN BOOK** assessment. The maximum mark is **50**.
4. Calculators are allowed, but not electronic dictionaries, notebooks, tablets, or other computing devices.
5. Do not look at the questions until you are told to do so.
6. Please write your **Student number** below. Do not write your name.

A	0	1						

This portion is for examiner's use only.

Question	Marks	Remarks
Q1	/16	
Q2	/6	
Q3	/8	
Q4	/12	
Q5	/8	
Total	/50	

Write your answers in the space provided.

1. [**16 marks**] Implement the following using Java streams. Any explicit looping/recursive implementations and data structures are **strictly not allowed**.

   (a) [**3 marks**] Complete the method `myCount` that takes in a `Stream<T>` of finite stream elements of generic type `T`, and returns the number of such elements. You are **not allowed** to use the terminal operation `count()`. For example,

   - `myCount(Stream.of("abc", "xyz"))` returns 2;
   - `myCount(Stream.of())` returns 0;

   ANSWER:

   ```
   public static <T> long myCount(Stream<T> stream) {
   ```

   (b) [**4 marks**] Complete the method `countRepeats` that takes in a string of lowercase letters and returns the number of occurrences of adjacent repeated letters. For example,

   - the string "mississippi" has three occurrences;
   - the string "ssss" has one occurrence

   Hint: The following `String method` might be useful:
   `char charAt(int index)` — Returns the char value at the specified index.

   ANSWER:

   ```
   public static long countRepeats(String str) {
   ```

(c) [**6 marks**] Complete the method `variance` that takes in an integer array of elements and returns the variance of the elements. The variance of an array of x_i elements is defined as

$$\sigma^2 = \frac{\sum_{k=0}^{n-1} (x_k - \mu)^2}{n-1}$$

where μ is the average of all n elements. For example,

- `variance(IntStream.rangeClosed(1,6).toArray())`
  returns `OptionalDouble[3.5]`;
- `variance(IntStream.of().toArray())`
  returns `OptionalDouble.empty`;

ANSWER:

```
public static OptionalDouble variance(int[] data) {
```

(d) [**3 marks**] Complete the method `reverse` that takes in a String and returns the reverse of the string while parallelizing the process.

```
public static String reverse(String str) {
    return str.chars() // returns an IntStream of char values
```

2. [**6 marks**] This question relates to the Discrete Event Simulator assignment.

   Before the simulation starts, customers are assigned the inter-arrival time and service time using the `RandomGenerator` object `rng` in the following manner:

```
        List<Customer> customers = new ArrayList<>();
        double now = 0;
        for (int i = 0; i < numOfCustomers; i++) {
            Customer customer = new Customer();
            customer.setArrivalTime(now);
            customer.setServiceTime(() -> rng.genServiceTime());
            customers.add(customer);
            now += rng.genInterArrivalTime();
        }
```

   Part of the `Customer` class is given below.

```
class Customer {

    double serviceTime;
    double arrivalTime;

    void setArrivalTime(double arrivalTime) {
        this.arrivalTime = arrivalTime;
    }

    void setServiceTime(double serviceTime) {
        this.serviceTime = serviceTime;
    }

    double getArrivalTime() {
        return this.arrivalTime;
    }

    double getServiceTime() {
        return this.serviceTime;
    }
    :
    :
```

   It was found that during the simulation, depending on the order in which customers were served, the service times were no longer in the order in which it was generated. Re-write the above program fragments on the following page such that it retains the order of service times generated while the customers are being served.

ANSWER:

— BLANK PAGE —

3. [**8 marks**] You are given a Java program that implements a question-answer system using two types of question formats:

   - MCQ: multiple-choice questions comprising answers: `A B C D E`
   - TFQ: true/false questions comprising answers: `T F`

   The classes for `MCQ` and `TFQ` are given below:

```
class MCQ {
    String question;
    char answer;

    public MCQ(String question) {
        this.question = question;
    }

    void getAnswer() {
        System.out.print(question + " ");
        answer = (new Scanner(System.in)).next().charAt(0);
        if (answer < 'A' || answer > 'E') {
            throw new InvalidMCQException("Invalid MCQ answer");
        }
    }
}

class TFQ {
    String question;
    char answer;

    public TFQ(String question) {
        this.question = question;
    }

    void getAnswer() {
        System.out.print(question + " ");
        answer = (new Scanner(System.in)).next().charAt(0);
        if (answer != 'T' && answer != 'F') {
            throw new InvalidTFQException("Invalid TFQ answer");
        }
    }
}
```

   In particular, an invalid answer to any of the questions will cause an exception (either `InvalidMCQException` or `InvalidTFQException`) to be thrown. These exceptions are sub-classes of the `IllegalArgumentException` class. An example is given below.

```
class InvalidMCQException extends IllegalArgumentException {
    public InvalidMCQException(String mesg) {
        super(mesg);
    }
}
```

The client class `Main` is provided to illustrate how the question-answer system works.

```
class Main {
    public static void main(String[] args) {
        try {
            MCQ mcq = new MCQ("What is the answer to this MCQ?");
            TFQ tfq = new TFQ("What is the answer to this TFQ?");

            mcq.getAnswer();
            tfq.getAnswer();
        } catch (InvalidMCQException ex) {
            System.err.println(ex);
        } catch (InvalidTFQException ex) {
            System.err.println(ex);
        }
    }
}
```

A sample run for the above is given below. User input is underlined. Notice that the program terminates once an invalid answer is given.

```
What is the answer to this MCQ? Q
InvalidMCQException:  Invalid MCQ answer
```

To better manage the different types of questions, you are to design a *more general* question-answer class `QA` that can take the place of both MCQ and TFQ types of questions (and possibly more in future, each with their own type of exceptions).

You will need to show the following:

- The entire `QA` class;
- The changes needed for the existing exception classes;
- Any other new classes that are included;
- Modifications to the `Main` driver class above for the new design. Note that the sample output should still remain the same.

ANSWER:

— **BLANK PAGE** —

— **BLANK PAGE** —

4. **[12 marks]** You own a dating agency where your clients gets to indicate which other clients they want to date with the following conditions:

- There are two types of clients `Man` and `Woman`;
- A `Man` can date another `Woman`, or vice-versa;
- A `Man` can also date another `Man`, and so does the women;
- A `Man` (or `Woman`) can only date one other `Woman` (or `Man`);
- For simplicity, all man and woman woman have unique names.

Here is an example of a possible setup of two relationships involving three men(M) and one women(W):

- Mickey(M) dates Minnie(W)
- Donald(M) dates Goofy(M)

Now suppose, there is a new relationship given by:

- Daisy(W) dates Donald(M)

This would cause Donald to "breakup" with Goofy, so that Daisy can proceed to date Donald. Hence, there are still two dating relationships, but three men (including Goofy) and two women as tracked by the system.

A sample output for the above setup is given below:

```
Number of relationships: 2
Number of men: 3
Number of women: 2

Mickey(M) and Minnie(W) are in a relationship
Daisy(W) and Donald(M) are in a relationship
Goofy(M) is not in a relationship
```

Your task is to design an OOP Java program to support the dating application. Take note of the following:

- This is a design question, so keep in mind the OOP concepts and design principles;
- There are only two import statements:

  ```
  import java.util.List;
  import java.util.ArrayList;
  ```

- You do not need to handle user input. Just write a test class to set up the above three relationships in sequence, so that it gives the desired output as shown above.

ANSWER:

— BLANK PAGE —

— **BLANK PAGE** —

5. [**8 marks**] In the lecture, we have seen the SOLID principles as a set of guiding principles for designing OO programs. How are these principles applied in the context of *streams* and *lambda expressions*? In particular, you will need to describe the applicability with respect to the following three principles:

(a) Single Responsibility Principle

(b) Liskov-Substitution Principle

(c) Open-closed Principle

Where appropriate, you should use sample program fragments to illustrate.

ANSWER:

— BLANK PAGE —
— END OF PAPER —