NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
FINAL ASSESSMENT FOR
Special Semester 1 AY2019/2020

CS2030 Programming Methodology II

June 2020 Time Allowed 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This assessment paper contains 10 questions and comprises 9 printed pages, including this page.
2. Write all your answers in the answer box provided on Examplify.
3. The total marks for this assessment is 70. Answer **ALL** questions.
4. This is a **OPEN BOOK** assessment. You are also free to refer to materials online.
5. All questions in this assessment paper use Java 11.

1. (6 points) In Java, what does the method signature consist of?

   How does the method signature relate to Overloading in Java? Give an example.

2. (2 points) Consider the following method header:

   ```
   public void doSomething(List<A> list)
   ```

   We would like this method to be able to accept any list containing elements of type A or any of its subclasses. Change this method header so that it is more permissive.

3. (5 points) Consider the following class:

   ```
   import java.util.stream.Stream;

   class Test {
     public void executeStream(Stream<Integer> stream) { }
     public void executeStream(Stream<String> stream) { }
   }
   ```

   The code excerpt will not compile. Explain what mechanism stops it from compiling, and how this mechanism works. In your answer refer to the code excerpt above.

4. (6 points) Consider the following classes `Triple` and `Client` carefully.

```
class Triple<T,R,S> {
  T t;
  R r;
  S s;
  Client c;

  Triple(T t, R r, S s, Client c) {
    this.t = t;
    this.r = r;
    this.s = s;
    this.c = c;
  }

  T getFirst() {
    return t;
  }

  R getSecond() {
    return r;
  }

  S getThird() {
    return s;
  }

  void print() {
    System.out.println(String.format(
        "%s %s %s", this.t,this.r,this.s));
  }
}
```

```
@SuppressWarnings("unchecked")
class Client {
   Triple triple;

   public Client() {
     triple = new Triple(1,2,3,this);
     triple.print();
     Client.addTriple(triple);
   }
  static void addTriple(Triple<Integer,Integer,Integer> triple) {
    System.out.println(String.format("%s", triple.t+triple.r+triple.s));
  }
}
```

As you can probably tell, these classes are not designed well, and they have not been written following the OOP principles that you learned about in CS2030! Describe at least three of the errors or problems made in the design, and note why they are bad design.

5. (6 points) Say we have four classes `A`, `B`, `C` and `D` such that `B<:A` and `C<:B`. Additionally, we have two interfaces `I1` and `I2`, and we know that `B` implements `I1`; and `C` implements `I2`. We also know that `D` implements `I1` and `I2`.

   Consider the following six variable declaration statements.

   ```
   A a;
   B b;
   C c;
   D d;
   I1 i1;
   I2 i2;
   ```

   We can create new objects of the classes above and make these variables reference these new objects. For example `a = new A();`.

   List out all legitimate and compilable assignments for all of the above variable declaration statements.

   Write each answer, one line at a time, such as:
   `x = new X();`
   `y = new Y();`
   Note that this question will be graded by a bot. So, if your answer contains any additional text, such as "The first one is x = new X();", it may lead to the answer being marked as wrong even if the intention of the answer is correct.

6. (7 points) You have been asked to put some checks on a new autopilot system for the latest drone aircraft. In the onboard software, the autopilot will make a request for a change in elevation and heading. The method `courseCorrection` in the autopilot class will execute this change. This is where you need to be sure you are making the right decision! So you have decided to introduce some new assert statements into your program for testing.

   When in flight, the aircraft is restricted to a designated fly zone, and is therefore not permitted to go above 3000m or below 100m. The change in elevation and heading can not be too drastic either, the elevation can be no more than 100m from the current elevation, and the heading can be no more than 20 degrees from the current heading. Additionally, the new status of the aircraft should not be outside the permitted fly zone. Consider the method `courseCorrection` below:

```
public class AutoPilot {
  ...
  public AircraftStatus courseCorrection(double newElevation, double newHeading) {
    AircraftStatus status = getCurrentStatus();
    // Assertion statement 1 HERE
    ...
    ... Method body where we change the heading/elevation for our aircraft
    ...
    // Assertion statement 2 HERE
    return newStatus;
  }
  ...
 }
```

   The `AircraftStatus` class can be seen below:

```
public class AircraftStatus {
  double elevation; // in metres
  double heading; // in degrees
}
```

   You are to write two `assert` statements for this method (as can be seen in the `courseCorrection` method).

   - One assertion statement that ensures that the input parameters are within the stated bounds.
   - One assertion statement that ensures the return value of the method is within the stated bounds.

   Note you do not need to write the method itself, just the assertions!

   Suppose now a subclass is created and it overrides the method above. Write an assert statement for the returned AircraftStatus `newStatus` for the overriding method, *that is different from the assert statements above*, such that the subclass adheres to the Liskov Substitution Principle.

   Your answer should consist of three Java `assert` statements, one on each line, and nothing else.

7. (10 points) Consider the following `Point` and `Circle` classes:

```
class Point {
  double x, y;

  public Point(double x, double y) {
    this.x = x;
    this.y = y;
  }

  @Override
  public String toString() {
    return "(" + x + "," + y + ")";
  }
}
```

```
class Circle {
  Point centre;

  public Circle(Point centre) {
    this.centre = centre;
  }

  @Override
  public String toString() {
    return "Circle: " + centre;
  }
}
```

By constructing a single stream pipeline, complete the following `getCircles` method to return a List as shown in the sample jshell session. Do not use any helper classes or loops.

Assume that the answer is saved in the file `getcircles.jsh`, and begin your answer with:

```
List<Circle> getCircles(int n) {
   return ...
}
```

```
jshell> import java.util.stream.*;
jshell> /open Point.java
jshell> /open Circle.java
jshell> /open getcircles.jsh
jshell> getCircles(10)
.. ==> [Circle: (1.0,10.0), Circle: (2.0,9.0), Circle: (3.0,8.0),
        Circle: (4.0,7.0), Circle: (5.0,6.0), Circle: (6.0,5.0),
        Circle: (7.0,4.0), Circle: (8.0,3.0), Circle: (9.0,2.0),
        Circle: (10.0,1.0)]
jshell> getCircles(11)
.. ==> [Circle: (1.0,11.0), Circle: (2.0,10.0), Circle: (3.0,9.0),
        Circle: (4.0,8.0), Circle: (5.0,7.0), Circle: (6.0,6.0),
        Circle: (7.0,5.0), Circle: (8.0,4.0), Circle: (9.0,3.0),
        Circle: (10.0,2.0), Circle: (11.0,1.0)]
jshell> /exit
```

8. (4 points) Consider the following four methods:

1. ```
   void add(int i, int j) {
     int k = i + j;
     a[0] = k;
   }
   ```
2. ```
   boolean compare(double x, double y) {
     return x > y;
   }
   ```
3. ```
   Optional<Double> findRadius(Circle c) {
     return Optional.ofNullable(c).map(x -> x.radius);
   }
   ```
4. ```
   <T> T getCentre(Box<T> t) {
     return t.contents;
   }
   ```

You need to determine which of these are pure functions and which are not. You should answer by writing a single character for each function. 'P' for pure functions and 'N' for those that are not. Therefore, your answer should look like four characters. For example, if you think all are pure functions you would write PPPP.

9. (10 points) You have been hired to improve an application which needs to logon to a server. The whole application becomes unresponsive when a user attempts to login in. You have isolated the main culprit of this slow down as the `userLogin` method, which takes in two strings (a username and password), and returns an object of type `LoginStatus` indicating the success of the login process. This `userLogin` method can be found below:

```
public LoginStatus userLogin(String username, String password) {
  if(this.loggedIn) {
    return new LoginStatus(true,"Already logged in!");
  } else {
    Boolean u = usernameExists(username);
    Boolean p = isValidPassword(password);
    if (u && p) {
      if(doLogin(username, password)) {
        return new LoginStatus(true,"Login by password.");
      } else {
        return new LoginStatus(false,"No Server Response");
      }
    } else {
      return new LoginStatus(false,"Incorrect Username/Password");
    }
  }
}
```

You have realised that the previous developer did not know how to use Asynchronous Programming! Rewrite this method to take advantage of Java's `CompletableFuture` class. The new method should return an object of type `CompletableFuture<LoginStatus>`.

10. (14 points) In this question you are to implement a class `CachedString` that encapsulates a lazily evaluated String value. A `CachedString` is initialised with a `Supplier`, such that when the value of the `CachedString` is needed, the `Supplier` will be evaluated to yield the value. Otherwise, the evaluation is delayed as much as possible.

    `CachedString` supports the following operations:

    - `map` takes in a `Function<? super String, String>` and returns a `CachedString` consisting of the results of applying the given function to the value of this `CachedString`.
    - `flatMap` takes in a `Function<? super String, CachedString>` and returns a `CachedString` consisting of the results of replacing the value of this `CachedString` with the value of a mapped `CachedString` produced by applying the provided mapping function to the value.
    - `forEach` takes in a `Consumer<String>` and then accepts on the value of `CachedString` and returns nothing.
    - `get` returns the value of the `CachedString`.
    - `isEmpty()` returns true if the `CachedString` is "", false otherwise.

    As an example, the expression below will return the String `"H3llo Th3r3"`.

    ```
    new CachedString(() -> "Hello")
        .map(x -> x + " ")
        .flatMap(x -> new CachedString(() -> x + "There"))
        .flatMap(x -> new CachedString(() -> x.replaceAll("e","3")))
        .get();
    ```

    Whereas, the following prints out `Answer: 2345432` and returns nothing.

    ```
    new CachedString(() -> "2345432")
        .map(x -> x + "4321")
        .flatMap(x -> new CachedString(() -> x.substring(1,8)))
        .forEach(x -> System.out.println("Answer: " + x));
    ```

    And the following returns `true`.

    ```
    new CachedString(() -> "Testing")
        .map(x -> "")
        .isEmpty();
    ```

    Given the following skeleton class, complete the constructor, and the methods `get`, `isEmpty`, `map`, `flatMap`, and `forEach`.

    ```
    import java.util.function.Supplier;
    import java.util.function.Function;

    class CachedString {
      // Write the CachedString class
    }
    ```

    Make sure that you delay any computation until the last minute!