NATIONAL UNIVERSITY OF SINGAPORE

CS2030(S) - PROGRAMMING METHODOLOGY II
(Semester 1: AY2020/21)

Final Assessment

Due: 19:00hrs on Monday, 30 November 2020

INSTRUCTIONS (Read carefully!)

1. This assessment paper contains **SIX (6)** questions and comprises **SEVEN (7)** printed pages, including this page.
2. Answer **ALL** questions. The maximum marks is **40**.
3. You may refer to your lecture notes, recitation solutions, and lab codes.
4. You may use the Luminus Forum (under the Heading "Final Exam Queries") to ask clarification questions. The instructors will respond on a best-effort basis (ie. responses may not be immediate). Do not use email, or another Forum Heading, to ask questions; they will be ignored.
5. By taking this assessment, you are agreeing to abide by the following Honor Code:
   (i) You will not discuss with, or receive help from, anyone.
   (ii) You will not search for solutions or help, whether online or offline.
   (iii) You will not share your answers with, or give help to, anyone.
   (iv) You will act with integrity.
6. **Breaching the Honor Code will result in severe penalties!**
7. Zip all your files (except the video) into a single zip file, without password. Name your zip file: `XXX.zip` where `XXX` is your student number starting with A0. Upload your file by the deadline above to the folder `Enn submission folder` in Luminus Files, where `Enn` is your Exam Group. You may upload as many times as you wish by the deadline, but only the latest file will be graded. As a backup, also email your submission to `cs2030nus@gmail.com`.
8. Separately upload the video of your screen capture, as stated in the exam protocol.

Question 1: Object-Oriented Design (8 marks)

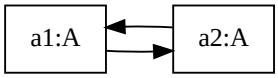
(a) (2 marks) Study the following class `A`.

```
class A {
    private A other;

    void set(A other) {
        this.other = other;
    }

    A get() {
        return this.other;
    }
}
```

Without modifying class `A`, write a series of jshell instructions to generate two instances `a1` and `a2` both of type `A`, where the `other` property `a1` refers to `a2`, and vice versa.



Write further jshell instructions to test that the assignment of the `other` property is valid and correct. Write your answer in the file: `oop-a.jsh`

(b) (2 marks) Suppose we would like to include more classes `B`, `C`, etc. that are similar to class `A`, i.e. having an `other` property that can refer to any object of type `A`, `B`, `C`, etc. Come up with an appropriate design and write the complete implementation. For simplicity, you may assume instances of class `A` and `B` currently, but your design should be readily extensible for other classes. There is no need to write jshell tests; the tests in part (a) must work here also.
Write your answer in the file: `oop-b.java`

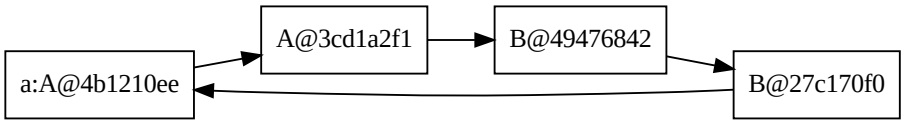
(c) (3 marks) Notice that the implementation of part(a) is mutable. Specifically the `other` property can be mutated after the object is created. Re-implement the solution of part (b) such that objects of classes `A`, `B`, `C`, etc. are now immutable. You may assume that the `other` property will always refer to some other object (i.e. not itself).
Write your answer in the file: `oop-c.java`

(d) (1 mark) Write a `Main` class with a `main` method to demonstrate how a chain of objects is created such that the following

```
System.out.println(a); // a references an instance of type A
System.out.println(a.get());
System.out.println(a.get().get());
System.out.println(a.get().get().get());
System.out.println(a.get().get().get().get());
System.out.println(a.get().get().get().get().get());
```

would result in the output (addresses of the objects might differ),

```
A@4b1210ee
A@3cd1a2f1
B@49476842
B@27c170f0
A@4b1210ee
A@3cd1a2f1
```



Write your answer in the file: `oop-d.java`. There is no need to reproduce the output statements.

Question 2: Generics (6 marks)

Study the following `replace` method.

```
void replace(List<Integer> src, List<Integer> dst) {
    if (src.size() == dst.size()) {
        for (int i = 0; i < src.size(); i++) {
            if (src.get(i) > dst.get(i)) {
                dst.set(i, src.get(i));
            }
        }
    }
}
```

```
jshell> List<Integer> destination = new ArrayList<>(List.of(1,2,3))
destination ==> [1, 2, 3]

jshell> replace(new ArrayList<>(List.of(9,9,9)), destination)

jshell> destination
destination ==> [9, 9, 9]
```

Notice that the `destination` list is replaced by elements of the `source` list since the corresponding element in the `source` list is larger.

(a) (4 marks) Rewrite the `replace` method so that it works on lists of any type `T`. The `replace` method should also take as input the criteria for replacement.

Write your answer in the file: `generics-a.jsh`

(b) (2 marks) Construct a test such that the types of the source list, destination list and type of criteria are different, but still related by some super-class or sub-class relationship. Express an appropriate criteria such that all elements of the destination are always replaced by the source.

Write your answer in the file: `generics-b.jsh`

Question 3: Lambda (2 marks)

Refer to the class `Foo` below.

```
class Foo {
    static int y = 1;

    Runnable bar() {
        int x = 1;
        Runnable r1 = () -> System.out.println(x);
        x = x + 1;
        return r1;
    }

    Runnable baz() {
        Runnable r2 = () -> System.out.println(y);
        y = y + 1;
        return r2;
    }
}
```

Explain why one of the lambdas compiles without error, while the other does not. Write your answer in the file: `lambda.txt`

Question 4: Functor (6 marks)

Your classmate I.M. Smart has created a `Box` class, and claims that it is a functor.

```
public class Box {
    private String value = "";

    private Box(String s) {
        value = s.toUpperCase();
    }

    public static Box of(String input) {
        return new Box(Objects.requireNonNull(input));
    }

    public Box map(Function<String, String> f) {
        return new Box(f.apply(this.value));
    }

    @Override
    public String toString() {
        return value;
    }
}
```

(a) (1 mark) What is the output of `Box.of("the Quick bROwn fox").toString()` ? Write your answer in the file: `functor-a.txt`

(b) (2 marks) What could `x` be if
`Box.of(x).map(s-> s.replace('P', 'E')).toString()`
produces `AEELE`
Write your answer in the file: `functor-b.txt`

(c) (3 marks) Does `Box` obey the functor laws? Give a concrete example.
Write your answer in the file: `functor-c.txt`

Question 5: Lazy Evaluation (10 marks)

Refer to the code in `LazyList.java`, provided for you in this exam folder.

(a) (2 marks) After the following code is executed, how many times is `filter` eagerly called? (Just state the number; no explanation needed.)

```
LazyList.intRange(101, 200).filter(n-> n%2==0);
```

Write your answer in the file: `lazy-a.txt`

(b) (2 marks) After the following code is executed, how many times is `map` eagerly called? (Just state the number; no explanation needed.)

```
LazyList.intRange(2, 100)
    .map(n-> 2*n)
    .filter(n-> n > 11);
```

Write your answer in the file: `lazy-b.txt`

(c) (2 marks) Counting the number of items in a `LazyList` is a reduction operation. Write an instance method `count()` that returns the number of items in `this` list by calling `reduce` with suitable arguments. Do not use explicit loops: eg. `for`, `while`, or recursion; and do not use Java `stream`. Instead, use `map`, `flatMap` or `filter`, as appropriate. You may assume that the list is finite.
Write your answer in the file: `lazy-c.java`

(d) (4 marks) Let's rewrite the `permute` function of Recitation 8, so that it can permute objects of type `T`, and handle the case $r = 0$:

```
<T> LazyList<LazyList<T>> permute(LazyList<T> LL, int r) {
    if (r == 0)
        return LLmake(LazyList.makeEmpty(), LazyList.makeEmpty());
    else if (LL.isEmpty())
        return LazyList.makeEmpty();
    else
        return LL.flatmap(x ->
                          permute(remove(LL, x), r - 1)
                          .map(y -> LLmake(x,y)));
}

<T> LazyList<T> remove(LazyList<T> LL, T n) {
    return LL.filter(x-> !x.equals(n));
}
```

You would have learned from mathematics that an r-**Combination** is an r-Permutation where order does not matter. That is, `(1, 2, 3)` is the same 3-Combination as `(1, 3, 2)`, even though they are different 3-Permutations.

Complete the code below to implement the `choose` function that returns a LazyList of $r-$Combinations (each of which is a length-r LazyList), when given a LazyList of n objects as input. You may assume that the input is finite.

Hint: Observe that you can group all the r-Combinations into two groups: those that use the first list item, and those that do not. Recursively generate these two groups, then `concat` them.

```
<T> LazyList<LazyList<T>> choose(LazyList<T> LL, int r) {
    if (r == 0)
        return LLmake(LazyList.makeEmpty(), LazyList.makeEmpty());
    else if (LL.isEmpty())
        return LazyList.makeEmpty();
    else
        //insert your code here
}
```

Write your answer in the file: `lazy-d.java`

Question 6: Streams (8 marks)

(a) (1 mark) The following program fragment is run in jshell.

```
int s = 0;
IntStream.range(1, 100).boxed().forEach(x -> s = s + x);
s;
```

What is the purpose of the above program fragment? State **what it does**, and not how it is done.
Write your answer in the file: `stream-a.txt`

(b) (2 marks) Comment on the way the stream was constructed to achieve the purpose of part (a). Rewrite the solution if necessary.
Write your answer in the file: `stream-b.jsh`

(c) (1 mark) Suppose we parallelize the stream construct in part (a) by including `parallel()` just before the `forEach` operator. Would it be correct? Explain.
Write your answer in the file: `stream-c.txt`

(d) (2 marks) Complete the following stream pipeline to count the total number of letters in a given list of words. If there are no words in the list, `count` will return 0.

```
int count(List<String> words) {
    return words.stream()
        .map(
        .reduce(
}
```

Write your answer in the file: `stream-d.jsh`

(e) (2 marks) Do the same as in part (d), but this time with only one `reduce` operation.

```
int count(List<String> words) {
    return words.stream()
        .reduce(
}
```

Write your answer in the file: `stream-e.jsh`

— END OF PAPER —