NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
FINAL ASSESSMENT FOR
Semester 2 AY2019/2020
PART 2 of 2

CS2030 Programming Methodology II

May 2020 Time Allowed 45 Minutes

INSTRUCTIONS TO CANDIDATES

1. This assessment is divided into two parts: Part 1 and Part 2.
2. This assessment paper contains 5 questions for Part 2.
3. Write all your answers in the answer boxes provided on Examplify.
4. The total marks for Part 2 is 40. Answer **ALL** questions.
5. This is a **OPEN BOOK** assessment. You are also free to refer to materials online.
6. All questions in this assessment paper use Java 11, unless otherwise specified.

No	Question	Marks
1	Optional	10
2	Header	5
3	Subtyping	8
4	Monad	9
5	Asynchornous	8
	Total	**40**

QUESTION 1

Question 1: Optional (10 points)

Study the method below.

```
String foo(String filename) {
  MyFile f = openFile(filename);
  if (f == null) {
    f = openFile("default.txt");
  }
  if (f != null) {
    Integer i = f.readNum();
    if (i < 10 && i >= 0) {
      return "The digit is " + i;
    }
  }
  return "Unable to read a single digit";
}
```

Rewrite the method below using a `Optional`. Your answer

- must consist of only a single return statement;
- must not use additional external classes or methods beyond those already used in the given code below;
- must not use `null` or the following `Optional`'s methods: `isEmpty`, `ifPresentOrElse`, `isPresent`, and `get`;
- must not contain `if`, `switch`, the ternary `? :` operators, or other branching logic besides those internally provided by `Optional` APIs.

Note that the specification and implementation details of the external class `MyFile` used in the method are not required to answer this question.

A solution template is provided below:

```
String foo(String filename) {
  return Optional ..
    ;
}
```

You must only write the body of the method (including the keyword `return`) to obtain full marks.

Question 1: Optional (10 points)

Study the method below. The method checks if a person with the given NRIC can enter a market based on the parity of the last digit of the NRIC and the current date.

```
boolean canEnter(NRIC nric) {
  if (nric == null) {
    throw new InvalidArgumentException();
  }
  Integer lastDigit = nric.lastDigit();
  if (lastDigit == null) {
    throw new InvalidArgumentException();
  }
  if (MyCalendar.currDate() % 2 == lastDigit % 2) {
    return true;
  } else {
    return false;
  }
}
```

Rewrite the method below using a `Optional`. Your answer

- must consist of only a single return statement;
- must not use additional external classes or methods beyond those already used in the given code below;
- must not use `null` or the following `Optional`'s methods: `isEmpty`, `ifPresentOrElse`, `isPresent`, and `get`;
- must not contain `if`, `switch`, the ternary `? :` operators, or other branching logic besides those internally provided by `Optional` APIs.

Note that the specification and implementation details of the external class `NRIC` used in the method are not required to answer this question.

A solution template is provided below:

```
boolean canEnter(NRIC nric) {
  return Optional ..
    ;
}
```

You must only write the body of the method (including the keyword `return`) to obtain full marks.

Question 1: Optional (10 points)

Study the method below:

```
Optional<Internship> match(Resume r) {
  if (r == null) {
    return Optional.empty();
  }
  Optional<ArrayList<String>> optList = r.getListOfLanguages();
  List<String> list;
  if (optList.isEmpty()) {
    list = new ArrayList<String>();
  } else {
    list = optList.get();
  }
  if (list.contains("Java")) {
    return Optional.ofNullable(findInternship(list));
  } else {
    return Optional.empty();
  }
}
```

Rewrite the method using `Optional` such that

- it consists of only a single return statement;
- it does not use additional external classes or methods beyond those already used in the given code below;
- must not use `null`, `Optional`'s `isEmpty()`, `isPresent()`, `ifPresentOrElse`, and `get()` method;
- it does not contain `if`, `switch`, the ternary `? :` operators, or other branching logic besides those internally provided by `Optional` APIs.

Note that the specification and implementation details of the external classes `Resume` and `Internship` used in the method are not required to answer this question.

A solution template is provided below:

```
Optional<Internship> match(Resume r) {
  return Optional...
    ;
}
```

You must only write the body of the method (including the keyword `return`) to obtain full marks.

QUESTION 2

Question 2: Header (5 points)

Frustrated by the limitations of Java's Stream API, Ah Beng sent a proposal to the Java Executive Committee (JEC) to propose adding a new method to Java's Stream API called `merge`. The method `merge` works as follows. Suppose we call `s1.merge(s2, lambda)`, where `s1` and `s2` are streams. The method `merge` would return a new stream `s3`. The first element of `s3` is the result of applying the lambda expression `lambda` to the first element of `s1` and `s2`. The second element of `s3` is the result of applying `lambda` to the second element of `s1` and `s2`, and so on.

The example below shows how Ah Beng intended the `merge` method to be use.

```
jshell> Stream<Integer> s1 = Stream.of(1, 2, 3)
jshell> Stream<String> s2 = Stream.of("hello", "world")
jshell> BiFunction<Number, Object, String> lambda = (x, y) -> x + ":" + y
jshell> Stream<Object> s3 = s1.merge(s2, lambda)
jshell> s3.toArray()
$.. ==> ["1:hello", "2:world", "3:null"]
```

Ah Beng does not have to implement `merge`, but he has to provide the JEC with the method header for the API, specifying the type parameters (if necessary), the return type, the name, and the type of each parameter.

To convince the JEC that he knows what he is doing, Ah Beng has to come up with the most flexible method header to cater to different usage scenarios. Since you have taken CS2030, Ah Beng came to you for help.

Write down what you think the method header for the new `merge` method for `Stream<T>` should be.

Question 2: Header (5 points)

Frustrated by the limitations of Java's Stream API, Ah Lian sent a proposal to the Java Executive Committee (JEC) to propose adding a new method to Java's Stream API called `doubleReduce`. The method `doubleReduce` is an extension of the `reduce` method, and it performs a reduction on the elements of the calling stream, using the provided `identity` and `accumulator`, and is equivalent to:

```
U result = identity;
for (T i : this stream)
  for (T j : this stream)
    result = accumulator.apply(result, i, j)
return result;
```

The example below shows how Ah Lian intended the `doubleReduce` method to be use.

```
jshell> Stream.of(1,2,3).doubleReduce("", s -> (x,y) -> s + x + ":" + y + " ");
$.. ==> "1:1 1:2 1:3 2:1 2:2 2:3 3:1 3:2 3:3 "
```

Ah Lian does not have to implement `doubleReduce`, but he has to provide the JEC with the method header for the API, specifying the the type parameters (if necessary), the return type, the name, and the type of each parameter.

To convince the JEC that he knows what he is doing, Ah Lian has to come up with the most flexible method header to cater to different usage scenarios. Since you have taken CS2030, Ah Lian came to you for help.

Write down what you think the method header for the new `doubleReduce` method for `Stream<T>` should be.

Question 2: Header (5 points)

Frustrated by the limitations of Java's Stream API, Ah Kow sent a proposal to the Java Executive Committee (JEC) to propose adding a new method to Java's Stream API called `nestedMap`. The method `nestedMap` is an extension of the `map` method, and it applies a given lambda expression on the elements of the calling stream.

Ah Kow intended

```
s.nestedMap(lambda)
```

to be equivalent to:

```
s.map(i -> s.flatMap(j -> lambda.apply(i,j)));
```

Ah Kow does not have to implement `nestedMap`, but he has to provide the JEC with the method header for the API, specifying the type parameters (if necessary), the return type, the name, and the type of each parameter.

To convince the JEC that he knows what he is doing, Ah Kow has to come up with the most flexible method header to cater to different usage scenarios. Since you have taken CS2030, Ah Kow came to you for help.

Write down what you think the method header for the new `nestedMap` method for `Stream<T>` should be.

QUESTION 3

Question 3: Subtyping (8 points)

Suppose we have Java classes `A1`, `A2`, .. `A5`, with the following class hierarchy:

```
A5 <: A4 <: A3 <: A2 <: A1
```

Consider the following method call

```
scanThis( Stream.of(1).map(x -> new A3()) );
```

Ignoring what `scanThis` does to the argument, what are the possible valid types for the argument of `scanThis` so that the statement above compiles without any warning or error?

Write down, one per line, up to 10 possible valid types (and only the valid types) of the argument `scanThis`.

Note that this question will be graded by a bot – it is important to write only one type per line. Do not include any extra text.

Question 3: Subtyping (8 points)

Suppose we have Java classes `A1`, `A2`, `A3`, and interfaces `I`, with the following subtype relationships:

`A3 <: A2 <: A1`

`A2 <: I`

Consider the following method call

```
process( Optional.of(1).map(x -> new A2()) );
```

Ignoring what `process` does to the argument, what are the possible valid types for the argument of `process` so that the statement above compiles without any warning or error?

Write down, one per line, up to 10 possible valid types (and only the valid types) of the argument `process`.

Note that this question will be graded by a bot – it is important to write only one type per line. Do not include any extra text.

Question 3: Subtyping (8 points)

Suppose we have Java classes `A`, `B`, `C1`, and `C2`, with the following subtype relationships:

`C1 <: B <: A`

`C2 <: B <: A`

Consider the following method call

```
doIt( IntStream.of(1).mapToObj(x -> new B()) );
```

Ignoring what `doIt` does to the argument, what are the possible valid types for the argument of `doIt` so that the statement above compiles without any warning or error?

Write down, one per line, up to 10 possible valid types (and only the valid types) of the argument `doIt`.

Note that this question will be graded by a bot – it is important to write only one type per line. Do not include any extra text.

QUESTION 4

Question 4: Monad (9 points)

Consider the class `IntMonad` below, which encapsultes a single `int` value. The implementation of `flatMap` is incomplete.

```
class IntMonad {
  private int v;

  private IntMonad(int v) {
    this.v = v;
  }

  static IntMonad of(int v) {
    return new IntMonad(v);
  }

  IntMonad flatMap(Function<Integer, IntMonad> map) {
    ..
  }
}
```

Now, consider the following three versions of `flatMap`.

```
// (a)
return IntMonad.of(this.v);

// (b)
return map.apply(2 * this.v);

// (c)
return map.apply(map.apply(this.v).v);
```

Let's represent the three laws of Monad with letter `L`, `R`, and `A`:

- `L`: Left Identity
- `R`: Right Identity
- `A`: Associative

Which of the above implementation of `flatMap` would cause `IntMonad` to violate the Laws of Monad?

Fill in the blank with the letter (or letters) representing the laws that a given `flatMap` implementation violates. Fill in the blank with the string `none` if no law is violated. For instance, if a given implementation violates the Right Identity and the Associative law, fill in the blank with the string `RA` or `AR`.

Note that this question will be graded by a bot. So, filling in with any other text, such as "R, A", "Right Identity and Associative", "none, because ..", will lead to the answer being marked as wrong even if the intention of the answer is correct.

Question 4: Monad (9 points)

Consider the class `IntMonad` below, which encapsultes a single `int` value. The implementation of `flatMap` is incomplete.

```
class IntMonad {
  private int v;

  private IntMonad(int v) {
    this.v = v;
  }

  static IntMonad of(int v) {
    return new IntMonad(v);
  }

  IntMonad flatMap(Function<Integer, IntMonad> map) {
    ..
  }
}
```

Now, consider the following three versions of `flatMap`.

```
// (a)
return IntMonad.of(this.v);

// (b)
return map.apply(Math.max(0, this.v));

// (c)
return IntMonad.of(2 * map.apply(this.v).v);
```

Let's represents of the three laws of Monad with letter `L`, `R`, and `A`:

- `L`: Left Identity
- `R`: Right Identity
- `A`: Associative

Which of the above implementation of `flatMap` would cause `IntMonad` to violate the Laws of Monad?

Fill in the blank with the letter (or letters) representing the laws that a given `flatMap` implementation violates. Fill in the blank with the string `none` if no law is violated. For instance, if a given implementation violates the Right Identity and the Associative law, fill in the blank with the string `RA` or `AR`.

Note that this question will be graded by a bot. So, filling in with any other text, such as "R, A", "Right Identity and Associative", "none, because ..", will lead to the answer being marked as wrong even if the intention of the answer is correct.

Question 4: Monad (9 points)

Consider the class `IntMonad` below, which encapsultes a single `int` value. The implementation of `flatMap` is incomplete.

```
class IntMonad {
  private int v;

  private IntMonad(int v) {
    this.v = v;
  }

  static IntMonad of(int v) {
    return new IntMonad(v);
  }

  IntMonad flatMap(Function<Integer, IntMonad> map) {
    ..
  }
}
```

Now, consider the following three versions of `flatMap`.

```
// (a)
return IntMonad.of(this.v);

// (b)
return map.apply(this.v + 2);

// (c)
return IntMonad.of(Math.max(this.v, map.apply(this.v).v));
```

Let's represents of the three laws of Monad with letter `L`, `R`, and `A`:

- `L`: Left Identity
- `R`: Right Identity
- `A`: Associative

Which of the above implementation of `flatMap` would cause `IntMonad` to violate the Laws of Monad?

Fill in the blank with the letter (or letters) representing the laws that a given `flatMap` implementation violates. Fill in the blank with the string `none` if no law is violated. For instance, if a given implementation violates the Right Identity and the Associative law, fill in the blank with the string `RA` or `AR`.

Note that this question will be graded by a bot. So, filling in with any other text, such as "R, A", "Right Identity and Associative", "none, because ..", will lead to the answer being marked as wrong even if the intention of the answer is correct.

QUESTION 5

Question 5: Asynchronous Programming (8 points)

Consider the program below. The method `doSomething()` may run for an undeterministic amount of time.

```
import java.util.concurrent.CompletableFuture;
class CF {
  static void doSomething() { .. }

  static CompletableFuture<Void> printAsync(int i) {
    return CompletableFuture.runAsync(() -> {
        doSomething();
        System.out.print(i);
    });
  }

  public static void main(String[] args) {
    printAsync(1).join();
    CompletableFuture.allOf(printAsync(2), printAsync(3))
      .thenRun(() -> printAsync(4));
    doSomething();
  }
}
```

What are the possible outputs printed by the program if `main` runs to completion normally?

Fill in the blank with the string **yes** if a given output is possible. Fill in the blank with the string **no** if `main` will never print the given output.

Note that this question will be graded by a bot. So, filling in with any other text, such as "NO!", "yes, because ..", "never!", etc, will lead to the answer being marked as wrong even if the intention of the answer is correct.

(a) `1`

(b) `2`

(c) `3`

(d) `4`

(e) `12`

(f) `14`

(g) `23`

(h) `24`

(i) `124`

(j) `134`

(k) `243`

(l) `234`

(m) `213`

(n) `1324`

(o) `4321`

Question 5: Asynchronous Programming (8 points)

Consider the program below. The method `doSomething()` may run for an undeterministic amount of time.

```
import java.util.concurrent.CompletableFuture;
class CF {
  static void doSomething() { .. }

  static CompletableFuture<Void> printAsync(int i) {
    return CompletableFuture.runAsync(() -> {
        doSomething();
        System.out.print(i);
    });
  }

  public static void main(String[] args) {
    printAsync(1);
    CompletableFuture.anyOf(printAsync(2), printAsync(3))
      .thenRun(() -> printAsync(4))
      .join();
    doSomething();
  }
}
```

What are the possible outputs printed by the program if `main` runs to completion normally?

Fill in the blank with the string `yes` if a given output is possible. Fill in the blank with the string `no` if `main` will never print the given output.

Note that this question will be graded by a bot. So, filling in with any other text, such as "NO!", "yes, because ..", "never!", etc, will lead to the answer being marked as wrong even if the intention of the answer is correct.

(a) `1`

(b) `2`

(c) `3`

(d) `4`

(e) `12`

(f) `14`

(g) `23`

(h) `24`

(i) `124`

(j) `134`

(k) `243`

(l) `234`

(m) `213`

(n) `1324`

(o) `4321`

Question 5: Asynchronous Programming (8 points)

Consider the program below. The method `doSomething()` may run for an undeterministic amount of time.

```
import java.util.concurrent.CompletableFuture;
class CF {
  static void doSomething() { .. }

  static CompletableFuture<Void> printAsync(int i) {
    return CompletableFuture.runAsync(() -> {
        doSomething();
        System.out.print(i);
    });
  }

  public static void main(String[] args) {
    CompletableFuture.anyOf(
         printAsync(1)
           .thenRun(() -> printAsync(2)),
         printAsync(3))
      .thenRun(() -> printAsync(4));
    doSomething();
  }
}
```

What are the possible outputs printed by the program if `main` runs to completion normally?

Fill in the blank with the string `yes` if a given output is possible. Fill in the blank with the string `no` if `main` will never print the given output.

Note that this question will be graded by a bot. So, filling in with any other text, such as "NO!", "yes, because ..", "never!", etc, will lead to the answer being marked as wrong even if the intention of the answer is correct.

(a) `1`

(b) `2`

(c) `3`

(d) `4`

(e) `12`

(f) `14`

(g) `23`

(h) `24`

(i) `124`

(j) `134`

(k) `243`

(l) `234`

(m) `213`

(n) `1324`

(o) `4321`

The End