NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
MIDTERM ASSESSMENT FOR
Semester 2 AY2020/2021

CS2030 Programming Methodology II

March 2021 Time Allowed 70 minutes

INSTRUCTIONS TO CANDIDATES

1. This assessment paper contains 30 questions and comprises 11 printed pages, including this page.
2. Write all your answers in the answersheet provided.
3. The total marks for this assessment is 40. Answer **ALL** questions.
4. This is a **OPEN BOOK** assessment. You are only allowed to refer to hardcopies materials.
5. All questions in this assessment paper use Java 11.

Section I: Multiple Choice Questions (5 points)

- For each of the questions below, **write your answer in the corresponding answer box on the answer sheet.** Each question is worth 1 marks.
- If multiple answers are equally appropriate, pick one and write the chosen answer in the answer box. Do NOT write more than one answer in the answer box.
- If none of the answers is appropriate, write `X` in the answer box.

1. (1 point) Java is described as a statically typed language. What does this mean?

   A. A class can have only one superclass.
   B. Some fields and methods to be declared as `static`.
   C. Once the compile-time type of a variable is decided, it cannot be changed.
   D. A variable of type `T` cannot be typecasted to another type `S`, unless `S` and `T` are related.

   Solution: C

2. (1 point) The code

   ```
   Array<Number> a = new Array<Integer>(3);
   ```

   does not compile because generic types are:

   A. contravariant
   B. invariant
   C. covariant
   D. supervariant

   Solution: B

3. (1 point) Consider the following generic class:

   ```
   class Wrapper<U extends Comparable<U>> {
       U value;
   }
   ```

   After type erasure, what will the type of `value` be?

   A. `Object`
   B. `Comparable<U>`
   C. `Comparable`

D. `Wrapper`

Solution: `C`

4. (1 point) A method signature consists of:

   A. The method name, the type of arguments, the number of arguments, and the order of the arguments
   B. The method name, the type of arguments, the number of arguments, the return type, and the order of the arguments
   C. The method name, the type of arguments, the return type, and the number of arguments.
   D. The method name, the number of arguments, and the order of the arguments

Solution: A

5. (1 point) What best describes what impact the `final` keyword has in this class?

```
final class Record {
  private int id;
  private String value;

  public int getID() {
    return this.id;
  }

  public String getValue() {
    return this.value;
  }
}
```

   A. All fields cannot be changed in any child class of `Record`.
   B. All methods cannot be overridden in any child class of `Record`.
   C. `Record` cannot be inherited.
   D. No interfaces can be implemented by `Record`.

Solution: C

Section II: Interface and Abstract Class (3 points)

The description below applies to Questions 6 - 8.

Consider the following:

```
interface I {
}

abstract class A<T> {
}

class C extends A<Integer> implements I {
}
```

6. (1 point) The statement below compiles without any error or warning. True or false? Please provide a rationale for your answer.

```
I i = new A<Integer>();
```

Solution: False. A does not inherit from I. A is abstract anyway.

7. (1 point) The statement below compiles without any error or warning. True or false? Please provide a rationale for your answer.

```
I i = new C();
```

Solution: True. C is a subtype of I.

8. (1 point) The statement below compiles without any error or warning. True or false? Please provide a rationale for your answer.

```
A<String> a = new C();
```

Solution: False. `C` is a subtype of `A<Integer>` and cannot be assigned to `A<String>`.

Section III: Exceptions (3 points)

The following applies to Questions 9 - 11. This question is graded by a bot. Please make sure you write your answer exactly as how Java would print its output to avoid unnecessary penalty.

Consider the following classes `Main` and `SSHClient`, where:

`PasswordIncorrectException` <: `AuthenticationException` <: `Exception`

```
class Main {
  void start() {
    try {
      SSHClient client = new SSHClient();
      client.connectPENode();
    } catch (Exception e) {
      System.out.println("Main");
    }
  }
}

class SSHClient {
  void connectPENode() throws Exception {
    try {
      // Line A (Code that could throw an exception)
    } catch (AuthenticationException e) {
      System.out.println("SSHClient");
    }
  }
}
```

After calling:

```
new Main().start()
```

9. (1 point) What would be printed if an `Exception` is thrown from Line A of `connectPENode`?

Solution:

```
Main
```

10. (1 point) What would be printed if an `AuthenticationException` is thrown from Line A of `connectPENode`?

Solution:

```
SSHClient
```

11. (1 point) What would be printed if a `PasswordIncorrectException` is thrown from Line A `connectPENode`?

Solution:

```
SSHClient
```

Section IV: Dynamic Binding (8 points)

The following description applies to Question 12 - 15.

This question is graded by the bot. Please make sure you write your answer exactly as how Java would print its output to avoid unnecessary penalty.

Consider the following four classes:

```
class A {
    void foo(A a) {
        System.out.println("class: A, parameter: A");
    }
}

class B extends A {
    @Override
    void foo(A a) {
        System.out.println("class: B, parameter: A");
    }

    void foo(B a) {
        System.out.println("class: B, parameter: B");
    }
}

class C extends B {
    void foo(C a) {
        System.out.println("class: C, parameter: C");
    }
}

class D extends C {
    @Override
    void foo(B a) {
        System.out.println("class: D, parameter: B");
    }
}
```

We initialize four variables as follows:

```
A a = new D();
B b = new D();
C c = new D();
D d = new D();
```

12. (1 point) What will be printed if we call:

```
a.foo(d);
```

Solution:

```
class: B, parameter: A
```

13. (1 point) What will be printed if we call:

```
b.foo(d);
```

Solution:

```
class: D, parameter: B
```

14. (1 point) What will be printed if we call:

```
c.foo(d);
```

Solution:

```
class: C, parameter: C
```

15. (1 point) What will be printed if we call:

```
d.foo(d);
```

Solution:

```
class: C, parameter: C
```

Solution:

	CTT of Target	All Accessible `foo` (incl. inherited)	Most Specific Callable with `d`	RTT of Target	Method Called
(a)	`A`	`void foo(A)`	`void foo(A)`	`D`	`B::foo(A)`

(b)	`B`	`void foo(A)`
		`void foo(B)`
			`void foo(B)`	`D`	`D::foo(B)`

(c)	`C`	`void foo(A)`
		`void foo(B)`
		`void foo(C)`
			`void foo(C)`	`D`	`C::foo(C)`

(d)	`D`	`void foo(A)`
		`void foo(B)`
		`void foo(C)`
			`void foo(C)`	`D`	`C::foo(C)`

Section V: Generic Typing (12 points)

The description below applies to Question 16 - 27.

Suppose we have the following classes:

```java
import java.util.List;

interface Trainable {
}

class Animal {
}

class Mammal extends Animal {
}

class Dog extends Mammal implements Trainable, Comparable<Mammal> {
    @Override
    public int compareTo(Mammal m) {
        return 1;
    }
}

class ShihTzu extends Dog {
}

class A {
  static <U, T extends U> U foo(List<? super T> list) {
    U u = null;
    return u;
  }
}
```

and the following variables:

```java
Dog dog = new Dog();
List<Dog> dogList = List.of(dog);
```

Indiciate if the following statement is true or false:

16. (1 point) `Animal` <: `Trainable`

> **Solution:** False

17. (1 point) `Dog` <: `Comparable<Animal>`

> **Solution:** False

18. (1 point) `ShihTzu` <: `Comparable<Mammal>`

Solution: True

`ShihTzu` <: `Dog`
<: `Comparable<Mammal>`.

(Updated 23 Feb 2023 10:45 AM)

19. (1 point) `ShihTzu` <: `Comparable<? extends Animal>`

Solution: True

`ShihTzu` <: `Comparable<Mammal>`
<: `Comparable<? extends Mammal>`
<: `Comparable<? extends Animal>`.

20. (1 point) `ShihTzu` <: `Comparable<? super ShihTzu>`

Solution: True

`ShihTzu` <: `Comparable<Mammal>`
<: `Comparable<? super Mammal>`
<: `Comparable<? super ShihTzu>`.

21. (1 point) `Comparable<ShihTzu>` <: `Comparable<? extends Trainable>`

Solution: True

`Comparable<ShihTzu>` <: `Comparable<? extends ShihTzu>`
<: `Comparable<? extends Trainable>`.

(Updated 20 Feb 2023 3:40 PM)

22. (1 point) The following statement would compile without warning/error. True of false?

```
dog = A.<Mammal,Dog>foo(dogList);
```

Solution: False

```
Error:
incompatible types: Mammal cannot be converted to Dog
dog = A.<Mammal,Dog>foo(dogList);
       ^-----------------------^
```

Even though the bound `Dog` <: `Mammal` is met, we can't return `Mammal` as `U` and assign it to a `Dog`. This is considered a narrowing type conversion.

23. (1 point) The following statement would compile without warning/error. True of false?

```
dog = A.<Dog,ShihTzu>foo(dogList);
```

Solution: True

The constraint `ShihTzu` <: `Dog` is met, and we can return `Dog` as `U` and assign it to another `Dog`.

24. (1 point) The following statement would compile without warning/error. True of false?

```
dog = A.<ShihTzu,ShihTzu>foo(dogList);
```

Solution: True

The constraint `ShihTzu` <: `ShihTzu` is met, and we can return `ShihTzu` as `U` and assign it to a `Dog` (widening type conversion).

25. (1 point) The following statement would compile without warning/error. True of false?

```
dog = A.<ShihTzu,Dog>foo(dogList);
```

Solution: False

```
Error:
method foo in class A cannot be applied to given types;
  required: java.util.List<? super T>
  found: java.util.List<Dog>
  reason: explicit type argument Dog does not conform to
  declared bound(s) ShihTzu
dog = A.<ShihTzu,Dog>foo(dogList);
      ^----------------^
```

The constraint `T` extends `U` is not met, since `Dog` $\not<:$ `ShihTzu`.

26. (1 point) The following statement would compile without warning/error. True of false?

```
dog = A.<Dog,Dog>foo(dogList);
```

Solution: True

The constraint `Dog` <: `Dog` is met, and we can return `Dog` as `U` and assign it to another `Dog`.

27. (1 point) If we call

```
    dog = A.foo(dogList);
```

What will `U` and `T` be inferred as?

Solution: First, consider target typing. We have `U` <: `Dog`.

Consider the argument typing. We pass `List<Dog>` into `List<? super T>`. So we have `T` <: `Dog`.

Considering the type bounds, we have `T <: U`.

Resolving these constraints, we have `T` and `U` both resolved to `Dog`.

Section VI: OO Principles (9 points)

The following applies to Questions 28 - 30.

Consider the following classes.

The class `Time` encapsulates a time measurement in units of seconds and milliseconds.

```java
class Time {
    public int second;
    public int millisecond;

    public Time(int second, int millisecond) {
        this.second = second;
        this.millisecond = millisecond;
    }
}
```

The class `Interval` encapsulates a time interval and consists of a starting time (begin) and an ending time (end).

```java
class Interval {
    private Time begin;
    private Time end;

    public Interval(Time begin, Time end) {
        this.begin = begin;
        this.end = end;
    }

    public int durationInMs() {
        return (end.second * 1000 + end.millisecond)
          - (begin.second * 1000 + begin.millisecond);
    }
}
```

A `Jiffy` is an interval with fixed duration of 10ms.

```java
class Jiffy extends Interval {
    public Jiffy() {
        super(new Time(0, 0), new Time(0, 10));
    }

    @Override
    public int durationInMs() {
        return 10;
    }
}
```

28. (3 points) The code above violates information hiding. True or false? Please provide a rationale for your answer.

Solution: True. Because `seconds` and `milliseconds` are exposed outside the abstraction barrier.

29. (3 points) The code above violates the ”tell, don’t ask” principle. True or false? Please provide a rationale for your answer.

Solution: True. Because `Interval` should tell `Time` to compute the difference instead of asking for the internals and do it itself.

30. (3 points) The code above violates the Liskov substitution principle. True or false? Please provide a rationale for your answer.

Solution: False. Any code written for `Interval` would still work if we substitute `Interval` with `Jiffy`.

END OF PAPER