NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
MIDTERM ASSESSMENT FOR
Semester 2 AY2022/2023

CS2030S Programming Methodology II

February 2023 Time Allowed 70 minutes

INSTRUCTIONS TO CANDIDATES

1. This assessment paper contains 12 questions and comprises 7 printed pages, including this page.
2. Write all your answers in the answer sheet provided.
3. The total marks for this assessment is 70. Answer **ALL** questions.
4. This is a **OPEN BOOK** assessment. You are only allowed to refer to hard-copy materials.
5. All programs in this assessment paper use Java 17 and are compiled with the flags `-Xlint:unchecked` and `-Xlint:rawtypes`.

Question	Points
1 - 4	8
5	3
6	8
7	10
8	12
9	7
10	9
11	3
12	10
TOTAL	70

Questions 1-6 concern the scenario below. We are writing a program to simulate the trading of toys by children in a primary school. Sometimes, children will swap their favorite toy for something that is of equivalent monetary value, for example, swapping a toy fighter jet with a toy helicopter. However, sometimes children will swap a toy of much higher monetary value for one that is of much lower value, e.g., trading a Game Boy for a toy plane, much to the concern of their parents.

Consider the following classes `ToyAircraft`, `ToyHelicopter`, `ToyJet`, `GameBoy`, and `Swap`. Note that the interface `Swappable` is not given. The method `doSwap` from `Swap<S, T>` implements a method specified in the interface `Swappable`.

```
abstract class ToyAircraft {
}

class ToyHelicopter extends ToyAircraft {
}

class ToyJet extends ToyAircraft {
}

class GameBoy {
}

class Swap<S, T> implements Swappable<S, T> {
  private S originalToy;
  private T newToy;

  public Swap(S originalToy) {
    this.originalToy = originalToy;
  }

  @Override
  public S doSwap(T newToy) {
    this.newToy = newToy;
    return this.originalToy;
  }
}
```

Questions 1-4 are multiple-choice questions. Select the most appropriate answer and write your choice (one of A to E) in the corresponding answer box on the answer sheet.

1. (2 points) Consider the following code excerpt:

   ```
   ToyAircraft aircraft = new ToyJet();
   ```

   What is the compile-time type of `aircraft`?

   A. `Aircraft`
   B. `ToyHelicopter`
   C. `ToyAircraft`
   D. `ToyJet`
   E. None of the above.

   Solution: C. `ToyAircraft`

   This is a giveaway question. 730/744 students got this correct.

2. (2 points) Consider the same code except in Question 1. What is the run-time type of `aircraft`?

A. `Aircraft`
B. `ToyHelicopter`
C. `ToyAircraft`
D. `ToyJet`
E. None of the above.

> **Solution:** D. `ToyJet`
>
> This is another giveaway question. 739/744 students got this one correct.

3. (2 points) After type erasure, what will be the type of `originalToy` in the class `Swap`?

A. `ToyJet`
B. `ToyAirplane`
C. `S`
D. `Object`
E. None of the above.

> **Solution:** D. `Object`
>
> `S` is not bounded and so it is erased to `Object`. 692/744 students got this correct.

4. (2 points) We will now use the `Swap` class to facilitate a swap of two toys.

```
Swap<ToyHelicopter, ToyJet> swap = new Swap<>(new ToyHelicopter());
ToyHelicopter plane = swap.doSwap(new ToyJet());  // Line A
```

After type erasure, the line that is labeled Line A will become:

A. `ToyHelicopter plane = (ToyHelicopter) swap.doSwap(new ToyJet());`
B. `ToyHelicopter plane = (ToyJet) swap.doSwap(new ToyJet());`
C. `Object plane = (Object) swap.doSwap(new ToyJet());`
D. `Object plane = swap.doSwap(new ToyJet());`
E. None of the above.

> **Solution:** A.
>
> `ToyHelicopter` is the type argument for `S` so `doSwap` returns `ToyHelicopter` in the original code. After erasure, `doSwap` returns `Object` instead, and so an explicit typecast is added by the erasure.
>
> Only 589 students got this correct. Many students chose C. But, we do not erase `ToyHelicopter` to `Object` since it is not a type variable/parameter.

For the rest of the questions, write your answer in the spaces provided in the answer sheet.

5. (3 points) The code for interface `Swappable` is ommitted above. `Swappable` is a generic interface with two type parameters `S` and `T` and a single abstract method `doSwap`.

   Write the code for `Swappable`.

   Solution:

   ```
   interface Swappable<S, T>{
       S doSwap(T t);
   }
   ```

   We gave one mark each for the correct interface declaration, the correct return type of `doSwap`, and the correct method signature of `doSwap`.

   This is meant to be a giveaway but surprisingly, only 501 students scored 3 marks. A surprising number of students included fields and methods with implementation (including empty implementation) when defining the interface.

6. (8 points) The current design of `Swap` allows the swapping of a `GameBoy` for a `ToyJet`.

   ```
   Swap<GameBoy, ToyJet> swap1 = new Swap<>(new GameBoy());
   GameBoy gameBoy = swap1.doSwap(new ToyJet());
   ```

   We want to change `Swap` so that a swap is permitted only if both objects are subtypes of the same type. If the objects being swapped are of different types then the swap is not allowed (the code will not compile).

   This is done by introducing a third type parameter `R` (to indicate the type of objects being swapped) and changing the type parameters of the class `Swap`.

   The new `Swap` would prevent the following from compiling

   ```
   Swap<GameBoy, ToyJet, ToyAircraft> swap2 = new Swap<>(new GameBoy());
   Swap<GameBoy, ToyJet, GameBoy> swap3 = new Swap<>(new ToyJet());
   ```

   while still allowing the following to compile.

   ```
   Swap<ToyJet, ToyHelicopter, ToyAircraft> swap4 = new Swap<>(new ToyJet());
   ToyJet jet = swap4.doSwap(new ToyHelicopter());
   ```

   (a) (6 points) Fill in `BlankA` to `BlankC` in the declaration of `Swap` below, so that the behavior above is achieved.

   ```
   class Swap<BlankA, BlankB, BlankC> implements Swappable<S, T> {
     // code omitted
   }
   ```

   Solution:

   ```
   class Swap<S extends R, T extends R, R> implements Swappable <S, T> {
     // code omitted
   }
   ```

   There are no partial marks for this question.
   521 students scored full marks for this question.

   (b) (2 points) It is still possible to swap a `GameBoy` with a `ToyJet` if we choose an appropriate type argument for `R`.

   ```
   Swap<GameBoy, ToyJet, BlankD> swap5 = new Swap<>(new GameBoy()); // Compiles
   ```

What should we choose for `BlankD` so that the above excerpt compiles?

> **Solution:** `Object`.
> 651 students got this correct.

7. (10 points) Consider the following Java program.

```
class Point2D {
  private int x;
  private int y;

  public Point2D(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public int coordinate(boolean xCoord) {
    return (xCoord) ? this.x : this.y;
  }

  @Override
  public boolean equals(Object obj) {
    if (obj instanceof Point2D) {
      Point2D point = (Point2D) obj;
      // NOTE: This method differs from the Point class from lecture
      return this.x == point.x || this.y == point.y;
    }
    return false;
  }
}
```

(a) (5 points) A subset of the important properties of the `equals` method, as written in the Java documentation, is reproduced here:

- It is **reflexive**: for any non-`null` reference value `x`, `x.equals(x)` should return `true`.
- It is **symmetric**: for any non-`null` reference values `x` and `y`, `x.equals(y)` should return `true` if and only if `y.equals(x)` returns `true`.
- It is **transitive**: for any non-`null` reference values `x`, `y`, and `z`, if `x.equals(y)` returns `true` and `y.equals(z)` returns `true`, then `x.equals(z)` should return `true`.

Does the class `Point2D` violate LSP? Explain your reasoning in no more than 3 sentences with reference to the program above, the properties of the `equals` method above, and example code (*if any*). No marks will be awarded if no explanation is given.

> **Solution:** It violates LSP as now `Point2D::equals` is no longer transitive. Consider three possible `Point2D` instances `x = new Point2D(1,1)`, `y = new Point2D(1,2)`, and `z = new Point2D(2,2)`. Then `x.equals(y)` and `y.equals(z)` are true but `x.equals(z)` is false.
>
> - Correctly saying it violates LSP (with reasons below): 1 mark
> - Correctly identifying the transitive property is violated: 2 marks
> - Giving correct example: 2 marks
>
> If the example is not fully transitive, such as:

```
a.equals(b); // true
b.equals(c); // true
c.equals(a); // need symmetry
```

Then they will not get the full mark unless they also specify that symmetry holds.
405 students received full marks for this question.

(b) (5 points) Now we consider extending `Point2D` into `Point3D` as shown below.

```
class Point3D extends Point2D {
  private int z;

  public Point3D(int x, int y, int z) {
    super(x, y);
    this.z = z;
  }

  @Override
  public boolean equals(Object obj) {
    if (obj instanceof Point3D) {
      Point3D point = (Point3D) obj;
      return this.coordinate(true) == point.coordinate(true) &&
             this.coordinate(false) == point.coordinate(false) &&
             this.z == point.z;
    }
    return false;
  }
}
```

Does `Point3D` violate "Tell, Don't Ask"? Explain your reasoning in no more than 3 sentences with reference to the program above. You may also refer to `Point2D` codes. No marks will be awarded if no explanation is given.

Solution: It violates "Tell, Don't Ask" because of the use of `Point2D::coordinate`. `Point3D` should simply call `super.equals(point)` (or alternatively `super.equals(obj)`). This tells the superclass to check for equality as opposed to asking for the fields from the superclass.

- Correctly saying it violates tell-don't-ask (with reasons below): 1 marks
- Correctly identifying that it is asking for coordinates from the parent class: 2 marks
- Correctly stating it should tell the parent class to check for equality: 2 marks

Even if students mention that `Point2D::equals` is not doing the same thing as what is required, tell-don't-ask is still violated and they should mention that a new method in `Point2D` should be created instead that performs the correct operation.
508 students received full marks for this question.

8. (12 points) Consider the class `Array<T>` (from a file `Array.java`) and four independent programs, labelled `A`, `B`, `C`, and `D` below, taken from files `A.java`, `B.java`, `C.java` and `D.java` respectively.

```
class Array<T> {
  private T[] array;

  Array(int size) {
    @SuppressWarnings("unchecked")
    T[] a = (T[]) new Object[size];
```

```
    this.array = a;
  }

  public void set(int index, T item) {
    this.array[index] = item;
  }

  public T get(int index) {
    return this.array[index];
  }
}

// A.java
class A {
  public static void main(String[] args) {
    String s = "hello";
    Object o = s;
    o = 1;
  }
}

// B.java
class B {
  public static void main(String[] args) {
    Array<String> s = new Array<>(1);
    s.set(0, "hello");
    Array<Object> o = s;
    o.set(0, 1);
  }
}

// C.java
class C {
  public static void main(String[] args) {
    String[] s = new String[] { "hello" };
    Object[] o = s;
    o[0] = 1;
  }
}

// D.java
class D {
  public static void main(String[] args) {
    Array s = new Array(1);
    s.set(0, "hello");
    Array o = s;
    o.set(0, 1);
  }
}
```

For each of the programs `A`, `B`, `C`, and `D`, fill in the table on the answer sheet to indicate whether they would compile with an error, compile with warning(s), and run with a run-time exception. Write **YES** if there is any warning/error/exception in the corresponding column; write **NO** otherwise. If a program cannot be compiled, write **N/A** for run-time exception.

Solution:

Program	Compilation Error?	Compilation Warning?	Run-Time Exception?
`A`	NO	NO	NO
`B`	YES	NO or N/A	N/A
`C`	NO	NO	YES
`D`	NO	YES	NO

There are 12 cells to fill. We give 1 mark per correct answer.

Only 72 students (less than 10%) scored full marks. The most common mistakes happen in the last column (Run-Time Exception).

Many students think that `A` will trigger a run-time exception at `o = 1`. This statement simply reassigns the reference `o` to refer to an `Integer`, instead of trying to assign an `Integer` to a `String`. So, there is no run-time exception.

Many students also think that `D` will trigger a run-time exception at `o.set(0, 1)`. The implementation of `Array`, however, internally stores the items as `Object[]`. There is no issue storing `Integer` into an array of `Object`s.

9. (7 points) Consider the `Array<T>` class from the question above. For each of the methods below, write down the *most flexible type argument* to be put into `BlankE` to `BlankG`.

   (a) (2 points) Method `f1`

   ```
   public static <T> void f1(Array<BlankE> array, T elem) {
     elem = array.get(0);
   }
   ```

   (b) (2 points) Method `f2`

   ```
   public static <T> void f2(Array<BlankF> array, T elem) {
     array.set(0, elem);
   }
   ```

   (c) (3 points) Method `f3`

   ```
   public static <T> void f3(Array<BlankG> array, T elem) {
     elem = array.get(0);
     array.set(1, elem);
   }
   ```

Solution:

```
public static <T> void f1(Array<? extends T> array, T elem) {
  elem = array.get(0); // Producer (extends)
}
public static <T> void f2(Array<? super T> array, T elem) {
  array.set(0, elem);  // Consumer (super)
}
public static <T> void f3(Array<T> array, T elem) {
  elem = array.get(0); // Producer (extends)
  array.set(1, elem);  // Consumer (super)
}                      // -- so must only be T
```

No partial marks is given for this question. 630, 616, and 544 students got the parts correct respectively.

10. (9 points) (a) (4 points) Ah Lian learned when there are multiple overloaded methods that match a method call, Java will invoke the method that is the most specific. She was curious about what would the Java

compiler do if there are two overloading methods, but neither is more specific than the other. She decided to write the following small program to test this out:

```
class A {
  void foo(BlankH i, BlankI j) {
  }

  void foo(BlankJ i, BlankK j) {
  }
}

class Main {
  public static void main(String[] args) {
    new A().foo("hello", "world");
  }
}
```

Help Ah Lian fill out the types of the parameters (`BlankH` to `BlankK`) of the two methods named `foo` so that when called with `new A().foo("hello", "world")` , neither method is more specific than the other.

Solution: Alternative #1

```
class A {
  void foo(String i, Object j) {
  }

  void foo(Object i, String j) {
  }
}
```

Alternative #2

```
class A {
  void foo(Object i, String j) {
  }

  void foo(String i, Object j) {
  }
}
```

596 students got this question correct. No partial marks is given. Common mistakes include

```
class A {
  void foo(String i, String j) {
  }

  void foo(Object i, Object j) {
  }
}
```

(one method is more specific than the other)

```
class A {
  void foo(Object i, Object j) {
  }

  void foo(Object i, Object j) {
  }
}
```

(overloading is disallowed since the method signature is the same)

(b) (5 points) Ah Lian was also curious about what would the Java compiler do if after performing type inferencing on a generic method invocation, the compiler needs to resolve two lower-bounded constraints `U` <: `T` and `V` <: `T`, where `U` and `V` are both classes with no subtype relationship between them. She decides to write a small program to test this out:

```
import java.util.List;

class U { }
class V { }

class A {
  static <BlankL> void foo(List<BlankM> l1, List<BlankN> l2) {
  }

  public static void main(String[] args) {
    List<BlankO> l1 = List.of();
    List<BlankP> l2 = List.of();
    A.foo(l1, l2);
  }
}
```

Help Ah Lian fill in the type variable/arguments `BlankL` to `BlankP`, so that during the type inference process of the statement `A.foo(l1, l2)`, Java will encounter two (and only two) constraints `U` <: `T` and `V` <: `T`. We exclude the implicit `U` <: `Object` and `V` <: `Object` since they are always true.

Solution: Alternative #1

```
class A {
  static <T> void foo(List<? extends T> l1, List<? extends T> l2) {
  }

  public static void main(String[] args) {
    List<U> l1 = List.of();
    List<V> l2 = List.of();
    A.foo(l1, l2);
    // Argument Typing: - List<U> <: List<? extends T> => T <: U
    //                  - List<V> <: List<? extends T> => T <: V
  }
}
```

Alternative #2

```
class A {
  static <T> void foo(List<? extends T> l1, List<? extends T> l2) {
  }

  public static void main(String[] args) {
    List<V> l1 = List.of();
    List<U> l2 = List.of();
    A.foo(l1, l2);
    // Argument Typing: - List<V> <: List<? extends T> => T <: V
    //                  - List<U> <: List<? extends T> => T <: U
  }
}
```

Only slightly less than half (369) of the class got this correct. The most common mistakes include:

- Not declaring `T` as a type variable before using.

- Using undeclared type variables (`S` , `R` , `X` , `Y` , etc)
- Using invalid type parameter bound such as `T super U` .
- Using `?` as a type variable.
- Mismatch between type parameter and type argument.

We give 3 marks if the answers lead to one wrong constraint (`U = T` or `V = T` or `T` <: `U` or `T` <: `V`) but the other constraint is correct.

11. (3 points) Ah Beng wants to create a new language called *Sumatra* that is the same as Java but with one major difference. In Sumatra, for Step 1 of the dynamic binding process, if there are multiple candidates for a method to be invoked, Sumatra chooses the *least specific* method. In other words, Sumatra always chooses the *most general* method instead of the *most specific* method like in Java.

    Luckily, for step 2 of the dynamic binding process, the search for the implementation of the method still starts from the run-time type of the target of the invocation. In other words, Step 2 of dynamic binding is the same as Java.

    Describe the problem that Ah Beng will face if dynamic binding is implemented in this way. Your answer should state the problem, write a code fragment that illustrates the problem with reference to the program below, and explain why the problem occurs.

```
class A {
  void f(A a) {
  }
}

class B extends A {
  void f(B b) {
  }
}
```

Solution: The method `B::f(B)` can never be executed. Invoking either `new B().f(new B())` or `new B().f(new A())` will execute `A::f(A)` because the method signature `f(A)` is the *least specific*.

We award 1 mark for correctly stating the problem; 1 mark for explaining the problem; and 1 mark for giving a relevant example.

About 395 students received full marks for this question.

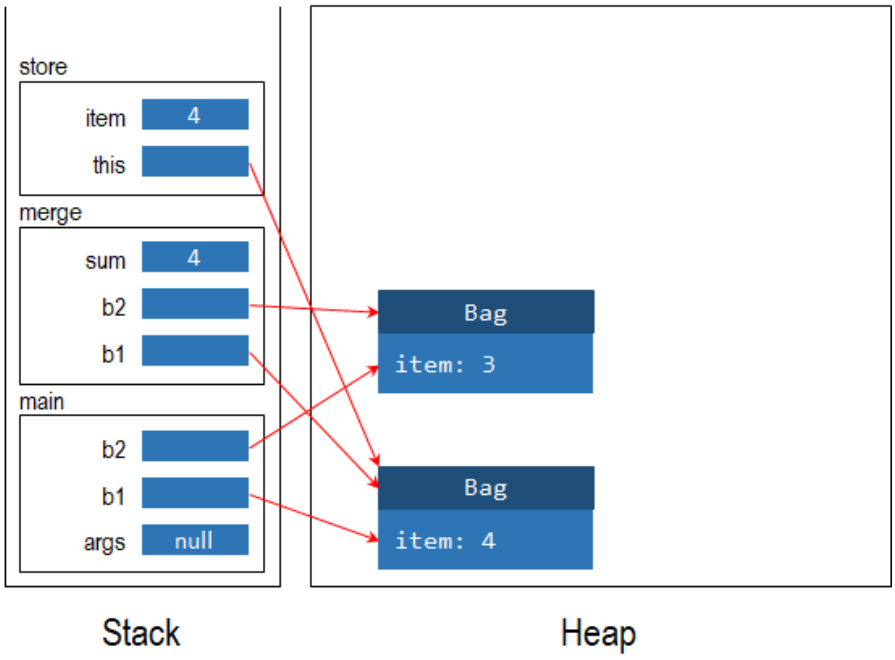
12. (10 points) Consider the program below. Draw the content of the stack and heap in the given answer sheet when the program runs and reaches Line X using the convention used in class.

    Label the stack frames with the method name, the objects on the heap with the class name, and the variables/fields with their names and their values.

```
class Bag {
  int item;

  Bag(int item) {
    this.item = item;
  }

  int retrieve() {
    return this.item;
  }
```

```
  void store(int item) {
    this.item = item;
    // --- Line X ---
  }
}

class Program {
  static Bag merge(Bag b1, Bag b2) {
    int sum = b1.retrieve() + b2.retrieve();
    b1.store(sum);
    return b1;
  }

  public static void main(String[] args) {
    args = null;
    Bag b1 = new Bag(1);
    Bag b2 = new Bag(3);
    b1 = merge(b1, b2);
  }
}
```

Solution: Stack and Heap Diagram



Only 223 students received full marks for this question.

- 1 mark for drawing `Bag b1` on the heap
- 1 mark for correct value inside the bag `b1`
- 1 mark for drawing `Bag b2` on the heap
- 1 mark for correct value inside the bag `b2`
- 1 mark for the stack frame of `main` and its variables.

- 1 mark for the correct values inside the stack frame of `main`.
- 1 mark for the stack frame of `merge` and its variables.
- 1 mark for the correct values inside the stack frame of `merge`.
- 1 mark for the stack frame of `store`, its variables, and its values.
- 1 mark for not drawing additional stack frames in the stack or objects on the heap.
- -1 mark for missing stack frame labels.
- -1 mark for missing heap labels.

If the order of stack frames is wrong, the 3 marks for the 3 stack frames above are not given.

Most students draw the objects and their fields correctly on the heap but made mistakes on the stack frames. In particular, note that `merge` is a static method so there is no `this` reference to the target. On the other hand, `store` is an instance method so there is a `this` reference to `b1`.

Some students also did not draw boxes around their stack frames/instances on the heap and got marks deducted. Students who use other ways to denote and/or label a stack frame (e.g., drawing lines between them) may not get marks deducted but are advised to stick to the conventions used in our module.

END OF PAPER