NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
MIDTERM ASSESSMENT FOR
Semester 2 AY2019/2020

CS2030 Programming Methodology II

March 2020 Time Allowed 1 Hour

INSTRUCTIONS TO CANDIDATES

1. This assessment paper contains 3 questions and comprises 6 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 24. 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.

This midterm took place in the beginning of the pandemic and replaced the intended practical assessment. Thus, the questions and format are not of a typical written midterm test.

1. (10 points) **Test-Driven Development**

   Test-Driven Development (TDD) is the process of writing programs to pass given test cases that are presented one at a time. Your task for this question is to write a series of short programs so as to pass the test cases.

   - Include the necessary import statements as your program will be compiled before testing against jshell.
   - Ensure that OOP principles are adhered to whenever applicable.

   The parts below are independent of each other.

   (a) (1 point) Write two classes `X` and `Y` that passes the following tests.

   ```
   jshell> new X(1)
   $3 ==> X:1
   jshell> new Y(new X(1))
   $4 ==> Y->X:1
   jshell> /exit
   ```

   (b) (3 points) Write a class `A` that passes the following tests.

   ```
   jshell> new A(1)
   $.. ==> [A:1]
   jshell> new A(1).next(2)
   $.. ==> [A:1][A:2]
   jshell> new A(1).next(2).next(3)
   $.. ==> [A:1][A:2][A:3]
   jshell> A a1 = new A(1)
   jshell> A a2 = a1.next(2)
   jshell> A a3 = a1.next(2).next(3)
   jshell> a2
   a2 ==> [A:1][A:2]
   jshell> a1
   a1 ==> [A:1]
   jshell> /exit
   ```

(c) (3 points) Write two classes `B` and `C` that passes the following tests. Moreover, ensure that `B` and `C` **are not cyclic-dependent**.

Note: Cyclic dependency is not emphasised in CS2030S AY21/22 Sem 2.

```
jshell> B b
jshell> C c
jshell> c = b
|  Error:
|  incompatible types: B cannot be converted to C
|  c = b
|      ^
jshell> b = c
|  Error:
|  incompatible types: C cannot be converted to B
|  b = c
|      ^
jshell> new B()
$.. ==> B
jshell> new C()
$.. ==> C
jshell> new B().add(new B())
$.. ==> BB
jshell> new B().add(new C())
$.. ==> BC
jshell> new B().add(new C().add(new B()))
$.. ==> BCB
jshell> new B().add(new C().add(new C()))
$.. ==> BCC
jshell> new C().add(new C().add(new B())).add(new C())
$.. ==> CCBC
jshell> /exit
```

(d) (3 points) Write a class `D` that passes the following test.

```
jshell> class E { public String toString() { return "E"; }}
jshell> class F extends E { public String toString() { return "F"; }}
jshell> List<E> p = D.add(new ArrayList<E>(), new E())
jshell> List<F> q = D.add(new LinkedList<F>(), new F())
jshell> List<E> r = D.add(D.add(new LinkedList<E>(), new E()), new F())
jshell> List<F> s = D.add(D.add(new ArrayList<F>(), new F()), new F())
jshell> /var
|    List<E> p = [E]
|    List<F> q = [F]
|    List<E> r = [E, F]
|    List<F> s = [F, F]
jshell> List<E> x = D.join(p, q) // join q to the end of p
jshell> List<E> y = D.join(p, p)
jshell> List<F> z = D.join(q, q)
jshell> /var
|    List<E> p = [E]
|    List<F> q = [F]
|    List<E> r = [E, F]
|    List<F> s = [F, F]
|    List<E> x = [E, F]
|    List<E> y = [E]
|    List<F> z = [F]
jshell> D.join(q, p) // results in a compilation error
|    :
|    :
jshell> /exit
```

2. (10 points) **Type Conversion**

Consider two types S and T. If $S <: T$, then the Java compiler can convert type S into T and T into S (with explicit type casting). If two types S and T are not related, then the compiler cannot convert between them since their types are incompatible. This type mismatch occurs since Java does not know what is the process to apply to convert from one type to another when they are not a subtype of each other.

If one, however, can supply the code to perform the conversion, such type casting can still be done. You will develop the code to perform such customized type conversion in this question.

Write all your classes and interface into the space provided using valid Java 11 syntax (including `import` statement(s)).

- Write a generic interface `TypeCaster` with two type parameters `S` and `T`. `S` is the type we want to cast from; `T` is the type we want to cast into. There may not be any subtype-supertype relationship between `S` and `T`. `TypeCaster` has an abstract method `cast`. The method `cast` should take in an object of type `S` and return an object of type `T`.
- Write a generic class `ToString` with a single type parameter `S` that implements the `TypeCaster` interface. The `ToString` class is used to cast an object of type `S` to a `String`. The `cast` method should simply return the result of calling the `toString` method of the given object.
- Write a class `Round` that implements the `TypeCaster` interface. The `Round` class is used to cast a value of type `double` to an `int`. The `cast` method should simply return the result of calling the `Math.round` method of the given `double` value.
- Write a generic class `ToList` with a single type parameter `T` that implements the `TypeCaster` interface. The `ToList` class is used to cast an array of type `T[]` to a list of type `List<T>`. The `cast` method should return the list containing the same elements in the given array in the same order.
- Write a class `ListCaster` with a static generic method `castList` with two type parameters `S` (the type we want to cast from) and `T` (the type we want to cast into). `castList` takes in a list containing objects of type `S` and a `TypeCaster`, and casts every element in the list into type `T` using the given `TypeCaster` object. `castList` should return the resulting list. Make sure that any `TypeCaster` object that casts a supertype of `S` into a subtype of `T` can be used as a parameter of method `castList`.

3. (4 points) **PANDEMIC!!!**

Read the following problem description carefully and propose an object-oriented design for the problem described below. Your design should abide by the object-oriented principles you have learned. Avoid cyclic dependency.

> The COVID-19 Task Force is developing a system to keep track of confirmed COVID-19 cases in Singapore. Each confirmed case has an integer case id. There are two types of confirmed cases: imported and local. For each imported case (and only imported case), the system keeps track of the country the case is imported from. For each confirmed case, the system keeps track of the contacts of the case. A contact is another confirmed case (can be either local or imported). A case can have zero or more contacts. Each contact is labelled with the nature of the contact, which can be either one of the three: casual contact, close contact, family member.
>
> The cases can be grouped into clusters. Each cluster has a name (a `String`). A cluster can contain one or more cases. But a case might not necessarily belong to a cluster. A case can belong to multiple clusters.

Please note that:

- Declare the Java classes and/or interfaces, showing the relationships (composition, subtype, supertype) among them. You need to show the fields of each class, but you do not need to include any methods in your design.
- Write down your declarations in either top-down or bottom-up fashion.
- There is no need to show the external driver/client class (such as `Main`)
- Remember to include `import` statements as necessary.
- Write all the classes and/or interfaces in the box provided using valid Java 11 syntax.

END OF PAPER