National University of Singapore

SCHOOL OF COMPUTING

CS2030 — PROGRAMMING METHODOLOGY II
(Semester 1: AY2022/2023)

Nov / Dec 2022 Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This assessment paper consists of **FIVE(5)** questions and comprises **TEN(10)** printed pages, including this page.
2. Answer **ALL** questions. The maximum mark is **40**.
3. This is an **OPEN BOOK** assessment. You may refer to your lecture notes, recitation guides, lab codes, and the Java API.
4. 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 at all times.

   Breaching the Honor Code will result in severe penalties!
5. Write your answers within the individual program files and submit to the CodeCrunch course **CS2030_EX** with the task titled **CS2030 Final Assessment Submission**.
   `https://codecrunch.comp.nus.edu.sg/`
6. You are advised to submit all your files after attempting every question. No extra time will be given to submit your work at the end of the assessment. You may upload as many times as you wish, but only the latest submission will be graded.

Question	Q1	Q2	Q3	Q4	Q5	Total
Marks	8	8	8	8	8	40

1. [**8 marks**] A chatroom service allows users to join chat groups that are administered by chatbots. In this question, we only focus on the process of a new user joining the chat group managed by a particular chatbot.

   If necessary, you may use the `ImList`, `Pair` and `Lazy` classes provided.

   i. Write a `User` class such that a new user can be created and identified with a user name.

   ```
   jshell> User u1 = new User("user1")
   u1 ==> user1
   ```

   ii. Write a `Bot` class such that a new chatbot can be created with a randomly generated identifier between 0 and 999 both inclusive.

   Make use of the `java.util.Random` class to generate the identifier. Your bot identifiers might differ from the ones below.

   ```
   jshell> Bot bot = new Bot()
   bot ==> bot@186

   jshell> Bot bot = new Bot()
   bot ==> bot@693
   ```

   iii. A user can join the group administered by a bot via the `join` method.

   ```
   jshell> u1.join(bot)
   user1: bot@693 says hi to user1
   $.. ==> bot@693
   ```

   Note that the response "`bot@693 says hi to user1`" should be generated in the bot as a `String` and passed to the user to output. Since all output is isolated in the `User` class, the `Bot` class remains effect-free, i.e. the `Bot` class should not contain any output (e.g. `println`) statements.

   iv. The following sample run depicts two users joining `bot@693`.

   ```
   jshell> bot = u1.join(bot)
   user1: bot@693 says hi to user1
   bot ==> bot@693

   jshell> bot = new User("user2").join(bot)
   user1: bot@693 says hi to user2
   user2: bot@693 says hi to user2
   bot ==> bot@693
   ```

   Notice that when `user2` joins the bot, the existing `user1` will also be made known of the new join activity.

v. A bot may also initiate the creation of a chat group with a list of users.

```
jshell> bot = new Bot(List.of(u1, new User("user2")))
bot ==> bot@77
```

Since the join activity is not initiated by any user, there is no corresponding welcome message. The following depicts a new user joining the bot.

```
jshell> bot = new User("user3").join(bot)
user1: bot@77 says hi to user3
user2: bot@77 says hi to user3
user3: bot@77 says hi to user3
bot ==> bot@77
```

As expected, all three users are made known of the new member joining the group.

You are to ensure that your implementation adheres to immutability and do not contain any cyclic dependencies. For completeness, here is an example that shows an existing user `user1` joining another bot. You may assume that a user will not join the same bot multiple times.

```
jshell> Bot bot1 = u1.join(new Bot())
user1: bot@235 says hi to user1
bot1 ==> bot@235

jshell> new User("user4").join(bot)
user1: bot@77 says hi to user4
user2: bot@77 says hi to user4
user3: bot@77 says hi to user4
user4: bot@77 says hi to user4
$.. ==> bot@77

jshell> new User("user5").join(bot1)
user1: bot@235 says hi to user5
user5: bot@235 says hi to user5
$.. ==> bot@235
```

ANSWER:

2. [**8 marks**] *Submit **ALL** answers to CodeCrunch before proceeding.*

   Using classes in the `java.util.stream` package, answer the following questions by writing the appropriate generic methods.

   If necessary, you may use the `ImList`, `Pair` and `Lazy` classes provided.

   You will also need to adhere to the following constraints:

   - each method body should begin with a `return`;
   - there should be no additional helper methods;
   - do not use any form of looping or branching constructs;
   - use the two-argument `reduce` method.

   Moreover, ensure that your method implementations are *as general as possible*.

   (a) [**3 marks**] Write a method `reverse` that takes in a `List` of elements and returns the list with the element order reversed in the form of an `ImList`.

   ```
   jshell> reverse(List.<Integer>of(1, 2, 3))
   $.. ==> [3, 2, 1]
   ```

   ANSWER:

   (b) [**2 marks**] Write a method `pairing` that takes in a `List` comprising an even number of elements and outputs a `List` of `Pair`s of elements by pairing adjacent elements.

   ```
   jshell> pairing(List.<Integer>of(1, 2, 3, 4))
   $.. ==> [(1, 2), (3, 4)]
   ```

   ANSWER:

(c) **[3 marks]** Write a method `merging` that takes in a `List` comprising of `Pair`s of elements of the same type and outputs a `List` of elements by merging the pairs together.

```
jshell> merging(pairing(List.<Integer>of(1, 2, 3, 4)))
$.. ==> [1, 2, 3, 4]
```

ANSWER:

3. [**8 marks**] *Submit **ALL** answers to CodeCrunch before proceeding.*

   A student (`Student`) in CS2030 undergoes a series of practical assessments to ascertain his/her programming competency. Each practical assessment (`PA`) is made up of a series of levels, with each level (`Level`) being credited with integer marks.

   Suppose the classes `Student`, `PA` and `Level` are given. In particular, the method `getMarks()` in class `Level` has been defined that returns the marks obtained for a level.

   In this question, you will write the `getMarks()` method for the `PA` and `Student` classes, as well as the `getTotalMarks` method in the `Main` class.

   You will need to adhere to the following constraints:

   - each method body should begin with a `return`;
   - there should be no additional helper methods;
   - do not use any form of looping or branching constructs;
   - you are restricted from using the three argument `reduce` method.

   (a) [**2 marks**] Given the following `PA` class:

   ```
   class PA {
       private final List<Level> levels;

       // constructor and other methods omitted for brevity
   }
   ```

   Write the `getMarks()` method in the `PA` class to return the marks of all levels of the assessment individually as a `Stream`.

   ANSWER:

(b) **[3 marks]** You are given the following `Student` class.

```
class Student {
    private final List<Optional<PA>> listPA;

    // constructor and other methods omitted for brevity
}
```

Write the `getMarks()` method within the `Student` class to return the marks of all practical assessments (by adding up the level marks of each assessment) individually as a `Stream`.

A student may be absent for a particular practical assessment. In this case, leave the marks as `Optional.empty()`, so as to differentiate between a student missing an assessment, and getting the assessment entirely wrong.

ANSWER:

(c) **[3 marks]** Write a `Main` class with a `getTotalMarks` method that takes in a list of students (`List<Student>`) and returns the total sum of all the marks for all students across all the practical assessments they have taken. If a student misses a practical assessment, he/she would get zero marks for that assessment.

ANSWER:

4. [**8 marks**] *Submit **ALL** answers to CodeCrunch before proceeding.*

   Study the `ImList` class that has been provided to you. In this question, you are to include three additional methods in `ImList`. No existing methods in the given `ImList` should be modified unless stated otherwise.

   (a) [**4 marks**] Write the one-argument `reduce` method that takes in an appropriate `Function` (not `BiFunction`), but with no starting identity (or seed) value.

   Such a method should return an appropriate `Optional` value depending on the following situations:

   - when the list is empty;
   - when the list contains only one element;
   - when the list contains more than one element.

   The body of the `reduce` method should begin with a `return` keyword. Moreover, do not use `Optional::get()`, `Optional::isPresent()` or `Optional::isEmpty()` methods. You should also make use of existing methods in `ImList` provided as much as possible.

   At the end of the answer, show how you would make use of this one-argument `reduce` to sum up the list of elements `[1, 2, 3]`.

   ANSWER:

   (b) [**4 marks**] To prevent the creation of `ImList` via the constructors, write two overloaded `static` methods `of(..)` to create the list instead.

   - one method takes in no arguments and creates an empty `ImList`;
   - another method takes in a `List` and creates the equivalent `ImList`.

   Now suppose the following is invoked in JShell:

```
jshell> ImList.<Integer>of(List.of(1, 2, 3))
$.. ==> [1, 2, 3]

jshell> ImList.<Integer>of(List.of(1, 2, 3)).of()
$.. ==> []
```

   What happens when the last statement is invoked? Does it discard the old list, and create a new empty list? You will need to prevent any of the `of` methods from being invoked after the pipeline is created by giving out a compilation error. Do not throw exceptions.

   Write a minimally functional implementation of `ImList` to demonstrate how this can be achieved. If you write more than one class/interface, include them in your answer but make sure that they are not cyclic dependent.

   Also include your `reduce` implementation in Question 4a, as well as any other accompanying methods that `reduce` uses.

Here's a hint. We can write a static method in an inteface!

```
jshell> interface I {
   ...>    static I foo() {
   ...>       return new I() { };
   ...>    }
   ...> }
|  created interface I

jshell> I.foo()
$.. ==> I$1@4cc0edeb

jshell> I.foo().foo()
|  Error:
|  illegal static interface method call
|    the receiver expression should be replaced with the type qualifier 'I'
|  I.foo().foo()
|  ^-----------^
```

ANSWER:

5. [**8 marks**] *Submit **ALL** answers to CodeCrunch before proceeding.*

   You are given the following `processUrl` method which takes in a url as a `String`. The method fetches the webpage pointed to by the url, and returns the count of the words in the page as an integer.

```
int processUrl(String url) {
   // details omitted
}
```

   Now given a list of urls `List<String> urls`, we would like to fetch the individual webpages specified by each url in the list and obtain the number of words of each page.

   (a) [**4 marks**] Using JShell, write a program fragment to total up the number of words of all pages pointed to by the urls. Demonstrate how the fetching of pages can be done *lazily*, i.e. only on demand. Note that web pages are not static, i.e. the same url could result in different content to be fetched at different times.

   You may use assume that the following list of urls is given.

```
List<String> urls = List.of("abc.xyz", "cde.qpr", "xyz.abc")
```

   ANSWER:

   (b) [**4 marks**] Fetching webpages takes up a lot of time. Moreover, webpages are unrelated and thus can be fetched at the same time. Write a JShell program fragment to total up the number of words of all pages pointed to by the urls in a given list *within a minimum amount of time*.

   You may use assume that the following list of urls is given.

```
List<String> urls = List.of("abc.xyz", "cde.qpr", "xyz.abc")
```

   ANSWER:

— END OF PAPER —