NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
PRACTICAL ASSESSMENT II FOR
Semester 2 AY2022/2023

CS2030S Programming Methodology II

April 2023 Time Allowed 90 minutes

INSTRUCTIONS TO CANDIDATES

1. This practical assessment consists of **one** question. The total mark is 20: 2 marks for design; 2 for style; 16 for correctness. Style and correctness are given on the condition that reasonable efforts have been made to solve the given tasks.
2. This is an OPEN BOOK assessment. You are only allowed to refer to written/printed notes. No online resources/digital documents are allowed, except those accessible from the PE nodes (peXXX.comp.nus.edu.sg) (e.g., man pages are allowed).
3. You should see the following files/directories in your home directory.
   - `ShoppingCart.java`, `Product.java`, and `UserAccount.java`, for you to edit and solve the given task.
   - `pristine`, that contains a clean copy of the files above for your reference.
   - `Test1.java`, `Test2.java`, and `Test3.java`, for testing your code.
   - `cs2030s/fp`, that contains the cs2030s.fp package, including `Maybe.java`.
   - `checkstyle.sh`, `checkstyle.jar`, and `cs2030s_checks.xml`, for style checks.
   - `StreamAPI.md` and `MapAPI.md`, for references to Java's Stream and Map API.
4. Solve the programming tasks by editing `ShoppingCart.java`, `Product.java`, and `UserAccount.java`. You can leave the files in your home directory and log off after the assessment is over. There is no separate step to submit your code.
5. Only the files directly in your home directory will be graded. Do not put your code under a subdirectory.
6. Write your student number on top of EVERY FILE you edited as part of the `@author` tag. Do not write your name.
7. To compile your code, run `javac -Xlint:unchecked -Xlint:rawtypes *.java`. You can also compile the files individually if you wish.
8. You can run each test individually. For instance, run `java Test1` to execute `Test1`.
9. To check the style of your code, run `./checkstyle.sh`. You can also check the style of individual file with `java -jar checkstyle.jar -c cs2030_checks.xml <FILENAME>`.

IMPORTANT: If the submitted classes `ShoppingCart.java`, `Product.java`, and `UserAccount.java` cannot be compiled, 0 marks will be given for the corresponding task. Make sure your program compiles by running

```
javac -Xlint:unchecked -Xlint:rawtypes *.java
```

before submission.

You have been given three classes `UserAccount.java`, `Product.java`, and `ShoppingCart.java`.

UserAccount. Each `UserAccount` has a name and an email address. A `UserAccount` can update its name and its email address using the `setName` and `setEmail` methods respectively.

Product. A `Product` keeps track of a product ID, the product price, and the name of the product. A product can have its price changed using the `setPrice` method, its name changed using the `setName` method, and its product ID changed using the `setProductId` method. It is also possible to swap prices between two products using the `swapPrice` method. Two `Products` can be bundled together with the `bundle` method. The bundled `Product` has a price that is the sum of the two products combined with a 10% discount. The product ID of the bundled product is chosen as the product id of the first product.

ShoppingCart. In addition to these classes, there is a `ShoppingCart` class that drives many of the shopping operations. The `ShoppingCart` keeps track of products and quantities using a `Map` class, with the product id as the key and the pair (product, quantity) as its value. The method `cartContents` returns a string, listing all the products and their quantities in the shopping cart in decreasing order of product price. The method `getTotal` computes the total price of everything in the shopping cart. You can update the price or quantity of a product in the shopping cart using the methods `updateProductPrice` and `updateProductQuantity` respectively. You can remove a product from the shopping cart using the `removeProduct` method. Finally, products can be bundled with `bundleProduct` which takes in two product IDs and returns a new `Product`.

You should read through the files `UserAccount.java`, `Product.java`, and `ShoppingCart.java` to understand what methods they have, the parameters and the returned value of the methods, and their behavior in detail.

The product and quantity pairs in the shopping cart are stored using a `Map` (a collection of key-value pairs) as you have seen in Lab 5: `Maybe`. The keys in a map are unique, so you cannot have duplicate keys. To put a key-value pair into a map `m` use `m.put(K key, V value)`, and to remove a key-value pair from a map `m` use `remove(Object key)`. Although `Map` may throw exceptions, you do not have to handle exceptions and may assume that all test cases will not cause exceptions.

Task 1 (4 marks): Rewrite `Product.java` and `UserAccount.java` classes to be immutable.
All methods that mutate the class should return a new object. You are not allowed to add new methods in `Product.java` and `UserAccount.java`. We advise that you start by modifying the `UserAccount` class first, followed by the `Product` class. Make small changes at a time and make sure that your code compiles after every change.

Note that for `Product::swapPrice`, you are expected to return a `Pair` of products `(first, second)` such that `first` corresponds to the current product and `second` corresponds to the other product after the swap.

The tests `Test1.java` and `Test2.java` will test the immutability of the classes in `UserAccount.java` and `Product.java` respectively. Note that the given skeleton does not correctly pass all these tests. As in previous labs and PE1, make sure all OO principles, including LSP, tell-don't-ask, and information hiding, are adhered to, where applicable.

Task 2 (12 marks): Remove all loops, conditional statements, and `null` from `ShoppingCart`
The `ShoppingCart` class has multiple methods that have loops, conditional statements, and `null`. Remove these from the methods in `ShoppingCart.java`, except from `ShoppingCart::equals`, whilst retaining all of the functionality of `ShoppingCart.java`, You may use Java `Stream` and our `Maybe` to achieve this result.

Note that you are also not allowed to use block statements in your lambdas and you are not allowed to add new methods in `ShoppingCart.java`. For instance, the following is not allowed `x.map(x -> { .. })`.

To begin, you must change the return type of `findProduct` into `Maybe<Product>`. Make small changes at a time and make sure that your code compiles after every change.

The last test `Test3.java` will test the functionality of `ShoppingCart.java`. It correctly works on the mutable implementation you have been given.

END OF PAPER