NATIONAL UNIVERSITY OF SINGAPORE

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

CS2030S Programming Methodology II

March 2023 Time Allowed 90 minutes

INSTRUCTIONS TO CANDIDATES

1. This practical assessment consists of **one** question. The total mark is 20: 12 marks for design; 3 for style; 5 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 in your home directory.
   - The files `Test1.java`, `Test2.java`, ... to `Test7.java` for testing your solution.
   - The file `CallHistory.java` for you to improve upon.
   - The directories `inputs` and `outputs` contain the test inputs and outputs.
   - The directory `pristine` contains a copy of the original test cases and code for reference.
   - The file `Array.java` that implements the generic array `Array<T>`.
   - The files `checkstyle.sh`, `checkstyle.jar`, `cs2030_check.xml`, and `test.sh` are given to check the style of your code and to test your code.
   - You may add new classes/interfaces as needed by the design.
4. Solve the programming tasks by editing `CallHistory.java` and creating any necessary files. 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 under 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 created or 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. To run all the test cases, run `./test.sh`. You can also run the test individually. For instance, run `java Test1 < inputs/Test1.1.in` to execute `Test1` on input `inputs/Test1.1.in`.
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 or any of the new files you have added cannot be compiled, 0 marks will be given. Make sure your program compiles by running

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

before submission.

You have been given a class called `CallHistory` that keeps track of the call history of a phone. The class reads a list of phone call information from a file and provides several APIs to print the list of calls, print missed calls, return a call, and calculate the total minutes spent on the phone calls.

The class `CallHistory`, however, is written without following object-oriented principles. You are asked to re-write the class `CallHistory` to adhere to the object-oriented principles you have learned. You may create new classes as needed.

Your revised `CallHistory` must follow the same behavior as the given `CallHistory` (a copy of which can be found in `pristine/CallHistory.java` for your reference). We give an overview of the behavior of `CallHistory` here. For details, please see `CallHistory.java`.

Constructors. An instance of a `CallHistory` can be created using the constructor that takes in a `String` (representing the name of a text file) as an argument. A constructor for `CallHistory` may also take in no argument. In this case, it reads the input from the standard inputs.

```
CallHistory hist1 = new CallHistory(filename); // read from file
CallHistory hist2 = new CallHistory(); // read from standard input
```

Types of Calls. The class can handle three different types of calls.

- Type 0: A call without caller ID, i.e., the phone number of the caller is unknown.
- Type 1: A call with caller ID but cannot be found in the address book, i.e., the phone number is shown but the name of the caller is unknown.
- Type 2: A call with caller ID from a contact found in the address book.

Input File. The file to be loaded into `CallHistory` through the constructor has the following format.

- The first line of the file contains a positive integer n, which is the number of calls.
- The next n lines contain information about the calls. Each of these n lines contains two or more fields, separated by a comma.
  - The first field is always an integer and indicates the types of calls (0, 1, 2)
  - The second field is an integer that indicates the length of the call in minutes. The value of -1 indicates a missed call.
  - The third field applies only if the call comes with caller ID and indicates the phone number of the caller.
  - The fourth field applies only if the caller exists in the addressbook. This field is a string that contains the name of the caller.

You can assume that the given test data follows the input format correctly. A sample input file looks like:

```
6
0,3
1,-1,65554321
0,-1
2,5,95550001,Ahmad
1,10,65554321
2,-1,95551234,Devi
```

Listing Calls. There are two methods provided to list the calls, `printNumbers` and `printAllCalls`.

The method `printNumbers` takes in no arguments and returns nothing. It prints the phone number of each calls enumerated in the same order as they appear in the input files. The string "No Caller ID" is printed if the caller's number is not available.

For example `new CallHistory("Sample.txt").printNumbers()` would print

```
0 No Caller ID
1 65554321
2 No Caller ID
3 95550001
4 65554321
5 95551234
```

The method `printAllCalls` takes in no arguments and returns nothing. It prints the detailed information of each call enumerated in the same order as they appear in the input files, including the phone number, the length of the call (or the string "[MISSED]" if it is a missed call) and the name of the caller (if applicable).

For example `new CallHistory("Sample.txt").printAllCalls()` would print

```
0 No Caller ID | 3 minutes
1 65554321 | [MISSED]
2 No Caller ID | [MISSED]
3 95550001 | 5 minutes | Ahmad
4 65554321 | 10 minutes
5 95551234 | [MISSED] | Devi
```

Calling Back. We can make a call back to one of the listed call records, by calling the method `callback` with the call index as an argument and length of the call in minutes as the argument. If the call was a missed call, it is updated to be a non-missed call. The length of call is added to the total length of the call. For example, after we execute

```
  CallHistory hist = new CallHistory("Sample.txt");
  hist.callback(1, 24);
  hist.callback(3, 7);
  hist.printAllCalls();
```

the following will be printed

```
0 No Caller ID | 3 minutes
1 65554321 | 24 minutes
2 No Caller ID | [MISSED]
3 95550001 | 12 minutes | Ahmad
4 65554321 | 10 minutes
5 95551234 | [MISSED] | Devi
```

Note that we cannot return a phone call without caller IDs. If we try

```
  hist.callback(0, 24);
```

an error message "Unable to call back: No Caller ID" will be printed.

Print Missed Calls. We can also call the method `printMissedCalls` to print the details of all calls that are missed calls from contacts that can be found in the address book and have not been called back. Executing this snippet

```
  CallHistory hist = new CallHistory("Sample.txt");
  hist.printMissedCalls();
```

would cause the following to be printed:

```
5 95551234 | [MISSED] | Devi
```

After we execute this:

```
CallHistory hist = new CallHistory("Sample.txt");
hist.callback(5, 3);
hist.printMissedCalls();
```

then nothing would be printed as all missed call from known contact has been returned.

Minutes On Call. The class `CallHistory` provides a method `getMinutesOnCall` that can go through all calls and return the number of minutes spent on calls. For example, executing this snippet

```
CallHistory hist = new CallHistory();
hist.callback(1, 24);
hist.callback(3, 7);
System.out.println(hist.getMinutesOnCall());
```

would print the number 49, which is the sum of all minutes spent on all calls (3 + 24 + 12 + 10).

Your Tasks

Task 1: Rewrite this program using OOP principles

You should read through the file `CallHistory.java` to understand what it is doing. The given implementation applies minimal OO principles, your task in this exam is to rewrite `CallHistory.java`, including adding new classes to apply the OO principles you learned.

To achieve this, create a new class called `Call` to encapsulate the relevant attributes and methods. Create subclasses of `Call` as necessary. Use polymorphism to simplify the code in `CallHistory` and make your code extensible to possible new call types in the future. Make sure all OO principles, including LSP, tell-don't-ask, information hiding, are adhered to.

Task 2: Implement exception handling

Create a new exception class called `IllegalCallException` that is thrown when `callback` is invoked on a a call with no caller ID. The exception should be initialized with the error message "Unable to call back: No Caller ID".

This exception should be caught and handled in the method `callback` of `CallHistory`.

Reminder: all checked exceptions are a subclass of `java.lang.Exception`. The class `Exception` has the following constructor:

```
Exception(String msg)
```

that constructs a new exception with the specified detail message `msg`. The message can be retrieved by the `getMessage()` method, which returns the message as a `String`.

END OF PAPER