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 `DayCalendar.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 `DayCalendar.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 `DayCalendar` that implements a one-day event calendar. The class reads a list of events from a file and provides several APIs to print the events, remind users about upcoming events, cancel a meeting, and calculate the busy period for the day.

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

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

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

```
DayCalendar cal1 = new DayCalendar(filename); // read from file
DayCalendar cal2 = new DayCalendar(); // read from standard input
```

Types of Events. The class can handle three different types of events.

- Type 0: A birthday, which is an all-day event without a starting time and an ending time.
- Type 1: A lesson, which is a timed event, i.e., has a starting and ending time.
- Type 2: A meeting, which is also a timed event.

A birthday event is associated with the name of a person whose birthday is today; A meeting event is associated with the name of the person to meet with.

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

- The first line of the text contains a positive integer n, which is the number of events.
- The next n lines contain information about the events. 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 events (0, 1, 2)
  - The second field is a string that describes the event. For a birthday event, this is the name of the person whose birthday is today.
  - The third and fourth fields are positive integers that represent the starting time and ending time, in hours. These two fields only apply to timed events. The ending time is always greater than or equal to the starting time.
  - The fifth field applies to meeting events. This field is a string that contains the name of the person to meet with.

You can assume that the given test data follows the input format correctly.

```
5
1,CS2030S,12,14
0,Ah Huat
2,Discuss Project,11,12,Ahmad
1,MA2101,8,10
2,Breakfast,10,11,Devi
```

Listing Events. There are two methods provided to list the events, `printEventDescriptions` and `printEventDetails`.

The method `printEventDescriptions` takes in no arguments and returns nothing. It prints the description of each non-cancelled event enumerated in the same order as they appear in the input files.

For example `new DayCalendar("Sample.txt").printEventDescriptions()` would print

```
0 CS2030S
1 Birthday (Ah Huat)
2 Discuss Project
3 MA2101
4 Breakfast
```

Note that for bithday events, the name of the person whose birthday is today is also printed in parenthesis.

The method `printEventDetails` takes in no arguments and returns nothing. It prints the non-cancelled detailed information of each event enumerated in the same order as they appear in the input files, including the starting time (if any), ending time (if any), and the person associated with the event (if any).

For example `new DayCalendar("Sample.txt").printEventDetails()` would print

```
0 CS2030S | 12 - 14
1 Birthday (Ah Huat)
2 Discuss Project | 11 - 12 | Meet with Ahmad
3 MA2101 | 8 - 10
4 Breakfast | 10 - 11 | Meet with Devi
```

Cancelling Events. To cancel an event, we can call `cancelEvent` with the event index as an argument. For instance, to cancel Event 2, the meeting with Ahmad, we call `cancelEvent(2)`. After we execute:

```
DayCalendar cal = new DayCalendar("Sample.txt");
cal.cancelEvent(2);
cal.printEventDetails();
```

The following will be printed

```
0 CS2030S | 12 - 14
1 Birthday (Ah Huat)
3 MA2101 | 8 - 10
4 Breakfast | 10 - 11 | Meet with Devi
```

Only a meeting can be cancelled. Trying to cancel a birthday event or a lesson would cause an error message to be printed. For example, if we run

```
DayCalendar cal = new DayCalendar("Sample.txt");
cal.cancelEvent(0);
cal.cancelEvent(1);
cal.printEventDetails();
```

the following will be printed

```
Unable to cancel event: CS2030S
Unable to cancel event: Birthday (Ah Huat)
0 CS2030S | 12 - 14
1 Birthday (Ah Huat)
2 Discuss Project | 11 - 12 | Meet with Ahmad
3 MA2101 | 8 - 10
4 Breakfast | 10 - 11 | Meet with Devi
```

Trying to cancel a cancelled event has no effect.

Reminders. The class `DayCalendar` provides a method `remind` that lists all events that have not started yet.

The method `remind` takes in the current time (in hour) and list all events that have not been cancelled, where the starting time is greater than or equal to the given time. For example, executing this snippet

```
DayCalendar cal = new DayCalendar("Sample.txt");
cal.remind(10);
```

causes the following to be printed:

```
0 CS2030S | 12 - 14
2 Discuss Project | 11 - 12 | Meet with Ahmad
4 Breakfast | 10 - 11 | Meet with Devi
```

Executing this snippet

```
DayCalendar cal = new DayCalendar("Sample.txt");
cal.cancelEvent(2);
cal.remind(10);
```

causes the following to be printed instead.

```
0 CS2030S | 12 - 14
4 Breakfast | 10 - 11 | Meet with Devi
```

Busy Period. Users can find out how many hours they are busy with lessons and meetings.

The method `getBusyPeriod` returns the total number of hours that the user is busy. Events that have been cancelled should not contributes towards the busy period.

For example, executing this snippet

```
DayCalendar cal = new DayCalendar("Sample.txt");
cal.cancelEvent(2);
cal.getBusyPeriod(); // return 5
```

would return 6 since taking CS2030S and MA2101 took 4 hours and the meeting with Devi took 1 hour. The meeting with Ahmad has been cancelled.

Your Tasks

Task 1: Rewrite this program using OOP principles

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

To achieve this, create a new class called `Event` to encapsulate the relevant attributes and methods. Create subclasses of `Event` as necessary. Use polymorphism to simplify the code in `DayCalendar` and make your code extensible to possible new event 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

To handle the error where user tries to cancel a birthday event or a lesson event, create a new checked exception called `IllegalCancellationException` and throws this exception from `Event` (or its subclasses, if any) when appropriate. Catch and handle this exception in the method `cancelEvent` of `DayCalendar`.

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