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 `TaskList.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_checks.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 `TaskList.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 `TaskList` that implements a list of to-do tasks. The class reads a list of tasks from a file and provides several APIs to print the list, remind users about the tasks, mark a task as completed, and calculate the reward points for completing tasks.

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

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

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

```
TaskList list1 = new TaskList(filename); // read from file
TaskList list2 = new TaskList(); // read from standard input
```

Types of Tasks. The class can handle three different types of tasks.

- Type 0: A task that is without a deadline and can be completed any time.
- Type 1: A task that comes with a deadline.
- Type 2: A task that comes with a deadline and is delegated to an assignee to complete.

Note that we cannot assign a task without a deadline to an assignee.

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

- The first line of the file contains a positive integer n, which is the number of tasks.
- The next n lines contain information about the tasks. 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 tasks (0, 1, 2)
  - The second field is a string that describes the task.
  - The third field applies only if the task has a deadline. This field is a non-negative integer that indicates the number of days before the task is due.
  - The fourth field applies only if the task has a deadline and is assigned to someone else to complete. This field is a string that contains the name of the assignee.

You can assume that the given test data follows the input format correctly, with the exception that the first field might indicate an invalid task type (neither 0, 1, nor 2). An example input file `Sample.txt` is given to the right.

```
4
0,Finish Quiz
2,Setup Server,5,Foo
1,Email Ah Keong,2
1,Revise CS2030S,0
```

Listing Tasks. There are two methods provided to list the task, `printTaskDescriptions` and `printTaskDetails`.

The method `printTaskDescriptions` takes in no arguments and returns nothing. It prints the description of each task enumerated in the same order as they appear in the input files.

For example `new TaskList("Sample.txt").printTaskDescriptions()` would print

```
0 Finish Quiz
1 Setup Server
2 Email Ah Keong
3 Revise CS2030S
```

The method `printTaskDetails` takes in no arguments and returns nothing. It prints the detailed information of each task enumerated in the same order as they appear in the input files, including the due date (if any), assignee (if any), and whether the task is completed or not.

For example `new TaskList("Sample.txt").printTaskDetails()` would print

```
0 [ ] Finish Quiz
1 [ ] Setup Server | Due in 5 days | Assigned to Foo
2 [ ] Email Ah Keong | Due in 2 days
3 [ ] Revise CS2030S | Due in 0 days
```

Completing Tasks. To complete a task, we can call `completeTask` with the task index as an argument. For instance, to complete Task 2 "Email Ah Keong", we call `completeTask(2)`. After we execute:

```
TaskList list = new TaskList("Sample.txt");
list.complete(2);
list.printTaskDetails();
```

the following will be printed

```
0 [ ] Finish Quiz
1 [ ] Setup Server | Due in 5 days | Assigned to Foo
2 [X] Email Ah Keong | Due in 2 days
3 [ ] Revise CS2030S | Due in 0 days
```

Completing a task that has been completed has no effect.

Tasks Due Today. We can also call the method `printDueToday` to print the details of all tasks that are due today, i.e., in 0 days.

Executing this snippet

```
TaskList list = new TaskList("Sample.txt");
list.printDueToday();
```

would cause the following to be printed:

```
3 [ ] Revise CS2030S | Due in 0 days
```

Reminders. The class `TaskList` provides a method `remindAll` that can go through all incomplete tasks with deadlines and print a reminder.

If a task has an assignee, it prints a string with the format "Sending a reminder to complete DESCRIPTION to ASSIGNEE", replacing DESCRIPTION and ASSIGNEE with the actual description and assignee of the task.

If the task has no assignee, it reminds the owner of the task by printing "The task DESCRIPTION is due in DUE_IN days", replacing DESCRIPTION and DUE_IN with the actual description and the number of days the task is due in.

For example, executing this snippet

```
TaskList list = new TaskList("Sample.txt");
list.complete(2);
list.remindAll();
```

causes the following to be printed:

```
Sending a reminder to complete "Setup Server" to Foo
The task "Revise CS2030S" is due in 0 days
```

Reward Points. Users can receive reward points for completing tasks before their deadlines. If a task is completed k days before its due, k points would be rewarded. This reward point is given even for tasks that are delegated to assignees. The method `getRewardPoints` returns the number of reward points accumulated after completing the tasks.

For example, executing this snippet:

```
TaskList list = new TaskList("Sample.txt");
list.completeTask(2);
list.completeTask(0);
list.completeTask(1);
list.getRewardPoints(); // return 7
```

would return 7 since completing "Setup Server" yields 5 points and completing "Email Ah Keong" yields 2 points. The task "Finish Quiz" has no deadline and so completing it does not contribute to the reward points.

Handling Errors. You can assume that the input file format is valid. However, the type of tasks in the input might be an integer that is out of range, i.e., neither 0, 1, nor 2.

When the class `TaskList` loads an input file with such an error, it would print an error message with the invalid task type. For example, if the task type is given as 4, it would print: `Invalid task type in input: 4` and abort the loading of the tasks.

Your Tasks

Task 1: Rewrite this program using OOP principles

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

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

In the current implementation, the methods `createTask` and `loadTasks` return a boolean flag to indicate if the method is successful or not, and stored the invalid task type in an attribute `errorMsg`. When this error is encountered in `createTask`, it updates the `errorMsg`. The error is handled in the constructor of `TaskList`.

With exception handling, we can remove the need to return a boolean flag to indicate if the method is successful or not, and remove the need to maintain the attribute `errorMsg`.

To achieve this, create a new checked exception called `WrongTaskTypeException` and throws this exception from `createTask` when an error is encountered. Catch and handle this exception in the constructor of `TaskList`.

In your revised code, `createTask` and `loadTasks` must return `void` instead. The error message must no longer be stored in `TaskList` as an attribute.

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