Department of Computer Science
COS226 - Concurrent Systems

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Practical Assignment 6

- ☐ **Date Issued:** Tuesday, 20 October 2020
- ☐ **Date Due:** Monday, 2 November 2020, 18:00
- ☐ **Submission Procedure:** Upload your tasks on the CS Assignments Portal website
- ☐ **Assessment:** The practical will be marked online through the Assignments Portal and slots will open in the second week of the practical.
- ☐ This assignment consists of **3 tasks**

---

1 Introduction

1.1 Objectives and Outcomes

You must complete this assignment individually.

You may ask the Teaching Assistants for help but they will not be able to give you the solutions. They will be able to help you with coding, debugging and understanding the concepts explained both in the textbook and during lectures.

1.2 Source Code

Before you begin, you should download the source code from the ClickUP website. You are allowed to make changes to all the files in the source code. Ensure that your **student number** and **name** appears in the comments section at the top of each file that you submit.

1.3 Mark Allocation

This assignment is divided into three tasks. For each task:

(a) your program must produce the expected output;

(b) your program must not throw any exceptions; and

(c) your program must implement the correct mechanisms or algorithms, as specified by the assignment.

Your code will be assessed online through a script. Please note that the same marking rubric is used for auto-assessment and manual marking. If you do wish to have the structure of you code marked offline you can query after you receive your marks.

The marks for this practical are allocated as follows:

Activity	Mark
Task 1	8
Task 2	8
Task 3	8
Total	**24**

2 Program Logger

The program output will be displayed using a Logger object from the java.util.logging package. Each class that needs to display an output needs to declare a global Logger variable. For example:

```
private static final Logger LOGGER = Logger.getLogger("global");
```

The display configurations are specified in the logger.properties file.

This practical assignment will make use of two logging levels to display messages, `INFO` and `FINE`.

When you set a level, all messages belonging to that level and higher levels will be displayed. For example, if the level is set to `FINER`, messages logged as `FINER, FINE`, and `INFO` will be displayed. Please make sure to implement your output messages using the **correct logging method**. Failure to do so may result in you not being awarded marks for certain items, as the marking script will use the logging level to examine different types of thread behaviours.

The keywords enclosed in square brackets (**[keyword]**) should be replaced with the associated data.

3 Implementation

This practical will require you to implement different techniques for concurrent access of shared objects.

- Task 1 requires you to implement a Linked List that uses **optimistic synchronisation.** This concept is explained in chapter 9.
- Task 2 requires you to implement a Linked List that uses **lazy synchronisation**. This concept is explained in chapter 9.
- Task 3 requires you to implement a Bounded Queue that uses **optimistic synchronisation**. This concept is explained in chapter 10.

4 Scenario

Imagine working in a group project that consists of many different tasks that need to be completed in order to complete the project. Each person can add tasks or remove tasks from the group's to-do list. A person can also see if a certain task is already on the to-do list. In this scenario, a person is represented by a thread, a task is represent by a node, and the to-do list is represented by the data structure used by the task.

Just like normal to-do lists should not have multiple tasks with the same name, the list should not have nodes that have the same name (no duplicates). When adding a task, you add it to the end of the list.

NOTE: You will be simulating the same scenario in all tasks, just using different implementations. Also note the code in the book is a guideline for the concepts. **Do not** copy the book exactly and expect full marks. You need to use the book as a guideline in order to implement the scenario given to you.

4.1 Task 1 - Optimistic Synchronisation

You will need to implement all of these functions. Something to remember about this list is you have two sentinel nodes that are never removed or searched. These are the "HEAD" and "TAIL" nodes of the list. You can give them these task names in your implementation, or you could leave the names as empty strings.

4.1.1 What to Implement

- `GroupMember`
  - `GroupMember(OptimisticToDoList list, Boolean adder, String taskName)`
  - `void run()`
- `Task`
  - `Task(String taskName)`
  - `void lock()`
  - `void unlock()`
  - `String getTaskName()`
  - `Task getNextTask()`
  - `void setNextTask(Task nextTask)`
- `OptimisticToDoList`
  - `OptimisticToDoList(String[] taskNames)`
  - `void addTask(String name)`
  - `void removeTask(String name)`
  - `void printList()`
  - `boolean validate(Task predecessor, Task current)`

4.1.2 Implementation Notes

- Use the values in the string array passed to the `OptimisticToDoList` constructor to create a list. That is, create a new list that contains all the values in the given `taskNames` array.
- There will be a group members that will add tasks to the list and r group members that will remove tasks from the list. A Boolean value passed to the group member's constructor will be used to determine whether the group member is an adder or a remover, where:
  - A value of `true` represents an adder
  - A value of `false` represents a remover
- Each thread will need to do the following:
  - If the group member is an adder, it will call the `addTask()` function with the `taskName` it received in its constructor.
  - If the group member is a remover, it will call the `removeTask()` function with the `taskName` it received in its constructor.
- This sequence of events needs to happen x times. **The Main class is responsible for simulating several runs of this scenario.**
- The configurable scenario values can be specified through the following `MemberOptions` static variables:
  - a: `MemberOptions.adders`
  - r: `MemberOptions.removers`
  - x: `MemberOptions.runs`
- Please do not hardcode any values as our marking scripts tests with our own values. All values used need to be done in a dynamic nature.

4.1.3 Expected Output

A full example of output is shown at the end of this section. The `[thread-id]` keyword refers to the thread-local ID generated by the `ThreadID` class.

- When a group member's `run()` method is called, you need to output the following:

  ```
  Group member [thread-id] is going to try to [add/remove] a task
  ```

  Note: This must be logged using the INFO level.

- When a group member successfully adds a task, you need to output the following:

  ```
  Group member [thread-id] added task [name]
  ```

Note: This must be logged using the INFO level.

- When a group member cannot add a task because the task is already in the list, output:

  `Group member [thread-id] could not add task [name] as it is already in the list`

  Note: This must be logged using the INFO level.

- When a group member successfully removes a task, you need to output the following:

  `Group member [thread-id] removed task [name]`

  Note: This must be logged using the INFO level.

- When a group member cannot remove a task because the task is not in the list, output:

  `Group member [thread-id] could not remove task [name] as it was not in the list`

  Note: This must be logged using the INFO level.

- When `printList()` is called, you need to output the following:

  ```
  Todo List Size:  [size]
  [name], [name], [name], [name], [name], [name]
  ```

  Note: This must be logged using the FINE level.

The following is the output for 1 run with an initial task list of `[A,C,E,G,I,K,M,O,Q,S]`, and 3 adding and 3 removing group members:

```
Group member 0 is going to try to add a task
Group member 3 is going to try to remove a task
Group member 5 is going to try to remove a task
Group member 4 is going to try to add a task
Group member 2 is going to try to remove a task
Group member 1 is going to try to add a task
Group member 0 added task B
Group member 3 removed task E
Group member 2 removed task G
Group member 4 could not add task O as it is already in the list
Group member 1 added task F
Group member 5 could not remove task P as it was not in the list
Todo List Size:  10
A, C, I, K, M, O, Q, S, B, F
```

4.1.4 Upload Instructions

You must archive your files and upload them to the Task 1 submission slot on the CS Assignments Portal. Some points to take note of:

- You need to upload all of your files, except `Main.java`.
- Ensure there are no binary files in your archive (such as `.class` files).
- You can archive your files using `zip`, `tar`, or `tar.gz`.
- Make sure your files are at the root of your archive. Do not archive a folder.

4.2 Task 2 - Lazy Synchronisation

You will need to implement all of these functions. Something to remember about this list is you have two sentinel nodes that are never removed or searched. These are the "HEAD" and "TAIL" nodes of the list. You can give them these task names in your implementation, or you could leave the names as empty strings.

4.2.1 What to Implement

- `GroupMember`
  - `GroupMember(LazyToDoList list, boolean adder, String taskName)`
  - `void run()`
- `Task`
  - `Task(String taskName)`
  - `void lock()`
  - `void unlock()`
  - `String getTaskName()`
  - `Task getNextTask()`
  - `void setNextTask(Task nextTask)`
  - `void setMarkedField(boolean marked)`
  - `void getMarkedField()`
- `LazyToDoList`
  - `LazyToDoList(String[] taskNames)`
  - `void addTask(String name)`
  - `void removeTask(String name)`
  - `void printList()`

4.2.2 Implementation Notes

- Use the values in the string array passed to the `LazyToDoList` constructor to create a list. That is, create a new list that contains all the values in the given `taskNames` array.
- There will be a group members that will add tasks to the list and r group members that will remove tasks from the list. A Boolean value passed to the group member's constructor will be used to determine whether the group member is an adder or a remover, where:
  - A value of `true` represents an adder
  - A value of `false` represents a remover

- Each thread will need to do the following:
  - If the group member is an adder, it will call the `addTask()` function with the `taskName` it received in its constructor.
  - If the group member is a remover, it will call the `removeTask()` function with the `taskName` it received in its constructor.
- This sequence of events needs to happen x times. **The Main class is responsible for simulating several runs of this scenario.**
- The configurable scenario values can be specified through the following `MemberOptions` static variables:
  - a: `MemberOptions.adders`
  - r: `MemberOptions.removers`
  - x: `MemberOptions.runs`
- Please do not hardcode any values as our marking scripts tests with our own values. All values used need to be done in a dynamic nature.

4.2.3 Expected Output

A full example of output is shown at the end of this section. The `[thread-id]` keyword refers to the thread-local ID generated by the `ThreadID` class.

- When a group member's `run()` method is called, you need to output the following:

  ```
  Group member [thread-id] is going to try to [add/remove] a task
  ```

  Note: This must be logged using the INFO level.

- When a group member successfully adds a task, you need to output the following:

  ```
  Group member [thread-id] added task [name]
  ```

  Note: This must be logged using the INFO level.

- When a group member cannot add a task because the task is already in the list, output:

  ```
  Group member [thread-id] could not add task [name] as it is already in
  the list
  ```

  Note: This must be logged using the INFO level.

- When a group member successfully logically removes a task, you need to output the following:

  `Group member [thread-id] logically removed task [name]`

  Note: This must be logged using the INFO level.

- When a group member successfully physically removes a task, you need to output the following:

  `Group member [thread-id] physically removed task [name]`

  Note: This must be logged using the INFO level.

- When a group member cannot remove a task because the task is not in the list, output:

  `Group member [thread-id] could not remove task [name] as it was not in the list`

  Note: This must be logged using the INFO level.

- When `printList()` is called, you need to output the following:

  ```
  Todo List Size:  [size]
  [name], [name], [name], [name], [name], [name]
  ```

  Note: This must be logged using the FINE level.

The following is the output for 1 run with an initial task list of `[B,D,E,Z,I]`, and 3 adding and 2 removing group members:

```
Group member 2 is going to try add a task
Group member 0 is going to try add a task
Group member 3 is going to try remove a task
Group member 1 is going to try add a task
Group member 4 is going to try remove a task
Group member 0 could not add task B as it is already in the list
Group member 3 could not remove task P as it was not in the list
Group member 1 added task K
Group member 2 added task S
Group member 4 logically removed task B
Group member 4 physically removed task B
Todo List Size:  6
D, E, Z, I, K, S
```

4.2.4 Upload Instructions

You must archive your files and upload them to the Task 2 submission slot on the CS Assignments Portal. Some points to take note of:

- You need to upload all of your files, except `Main.java`.
- Ensure there are no binary files in your archive (such as `.class` files).
- You can archive your files using `zip`, `tar`, or `tar.gz`.
- Make sure your files are at the root of your archive. Do not archive a folder.

4.3 Task 3 - Bounded Queue

You will need to implement all of the functions mentioned below. Something to remember about this queue is you have one sentinel node that is never removed or searched. This is the "HEAD" node of the queue. You can give it this as the task name in your implementation, or you could leave the name as an empty string. Please note, since this is a queue it needs to follow FIFO principles.

4.3.1 What to Implement

You are given files for this practical, implement all functions that are defined in these files. Please note you can add functions if you need them.

- `GroupMember`
  - `GroupMember(BoundedQueueToDoList queue, boolean enqueue, String taskName)`
  - `void run()`
- `Task`
  - `Task(String taskName)`
  - `String getTaskName()`
  - `Task getNextTask()`
  - `void setNextTask(Task nextTask)`
- `BoundedQueueToDoList`
  - `BoundedQueueToDoList(int capacity, String[] tasks)`
  - `void enq(String taskName)`
  - `void deq()`
  - `void printList()`

4.3.2 Implementation Notes

- Use the values in the string array passed to the `BoundedQueueToDoList` constructor to create a queue. That is, create a new queue that contains all the values in the given `tasks` array. The `capacity` is the maximum amount of tasks the queue can have.
- There will be n group members that will add tasks to the list and m group members that will remove tasks from the list. A boolean value passed to the group member's constructor will be used to determine whether the group member is an adder or a remover, where:
  - A value of `true` represents an enqueuer
  - A value of `false` represents a dequeuer

- Each thread will need to do the following:
  - If the group member is an enqueuer. It will sleep for a random amount. This amount must be between 0 and 10. It will then call the `enq()` function with the taskName it received in its constructor.
  - If the group member is a dequeuer. It will sleep for a random amount. This amount must be between 0 and 15. It will then call the `deq()`.
- This sequence of events needs to happen x times. **The Main class is responsible for simulating several runs of this scenario.**
- The configurable scenario values can be specified through the following `MemberOptions` static variables:
  - n: `MemberOptions.enqueuers`
  - m: `MemberOptions.dequeuers`
  - x: `MemberOptions.runs`
- Please do not hardcode any values as our marking scripts tests with our own values. All values used need to be done in a dynamic nature.

4.3.3 Expected Output

- When a group member's `run()` method is called, you need to output the following:

  ```
  Group member [thread-id] is going to try [add/remove] a task
  ```

  Note: This must be logged using the INFO level.

- When a group member successfully adds a task, you need to output the following:

  ```
  Group member [thread-id] added task [name]
  ```

  Note: This must be logged using the INFO level.

- When a group member cannot add a task because the task is already in the list, output:

  ```
  Group member [thread-id] could not add task [name] as it is already in
  the list
  ```

  Note: This must be logged using the INFO level.

- When a group member successfully removes a task, you need to output the following:

  ```
  Group member [thread-id] removed task [name]
  ```

Note: This must be logged using the INFO level.

- When `printList()` is called, you need to output the following:

  ```
  Todo List Size:  [size]
  [name], [name], [name], [name], [name], [name]
  ```

  Note: This must be logged using the FINE level.

A full example of the program output. This was a run with 2 dequeuers and 4 enqueuers, and an initial task list of `[A,Z]`:

```
Group member 1 is going to try remove a task
Group member 0 is going to try add a task
Group member 3 is going to try add a task
Group member 4 is going to try add a task
Group member 2 is going to try remove a task
Group member 5 is going to try add a task
Group member 0 could not add task A as it is already in the list
Group member 4 added task O
Group member 5 added task L
Group member 1 removed task A
Group member 3 added task D
Group member 2 removed task Z
Todo List Size:  3
O, L, D
```

4.3.4 Upload Instructions

You must archive your files and upload them to the Task 3 submission slot on the CS Assignments Portal. Some points to take note of:

- You need to upload all of your files, except `Main.java`.
- Ensure there are no binary files in your archive (such as `.class` files).
- You can archive your files using `zip`, `tar`, or `tar.gz`.
- Make sure your files are at the root of your archive. Do not archive a folder.