COMP3021

Lab 10: multithreading and synchronization

Why we need multithreading

- We want our program do two tasks simultaneously
  - Draw GUI and respond to user events (main thread of JavaFX)
  - Background computation

- E.g. In Microsoft Word,
  - A thread to respond when you are typing words
  - A thread to do spell checking, word counting,…

- Otherwise, the GUI could be frozen
  - E.g., Chrome not responding when downloading webpage resources

How to implement multithread (General, NOT limited to GUI)

1. Extending the Thread class (not preferred),
2. **Passing the thread constructor an object which implements the Runnable interface**
    1. Regular class
    2. Anonymous class
    3. Lambda

1. Creating a regular class that implements Runnable

```java
// TaskClass.java
// Custom task class
public class TaskClass implements Runnable {
  ...
  public TaskClass(...) { ... }
  // Implement the run method in Runnable interface
  public void run() {
    ...
  }
}

// Client.java
public class Client {
  ...
  public void someMethod() {
    ...
        // Create an instance of TaskClass
        TaskClass task = new TaskClass(...);

        // Create a thread
        Thread thread = new Thread(task);

        // Start a thread
        thread.start();
        ...
  }
  ...
}
```

run() method defines what task each thread will do

Create a new thread and let it start

2. Anonymous class

```
1 // Client.java
2 public class Client {
3   ...
4   public void someMethod() {
5     ...
6     // Create an instance of TaskClass
7     TaskClass task = new TaskClass(...);
8
9     // Create a thread
10    Thread thread = new Thread(new Runnable() {
11      // Implement the run method in Runnable interface
12      public void run() {
13        ...
14      }
15    });
16
17    // Start a thread
18    thread.start();
19    ...
20  }
21  ...
22 }
```

Create a new thread using Anonymous class

3. Lambda expression

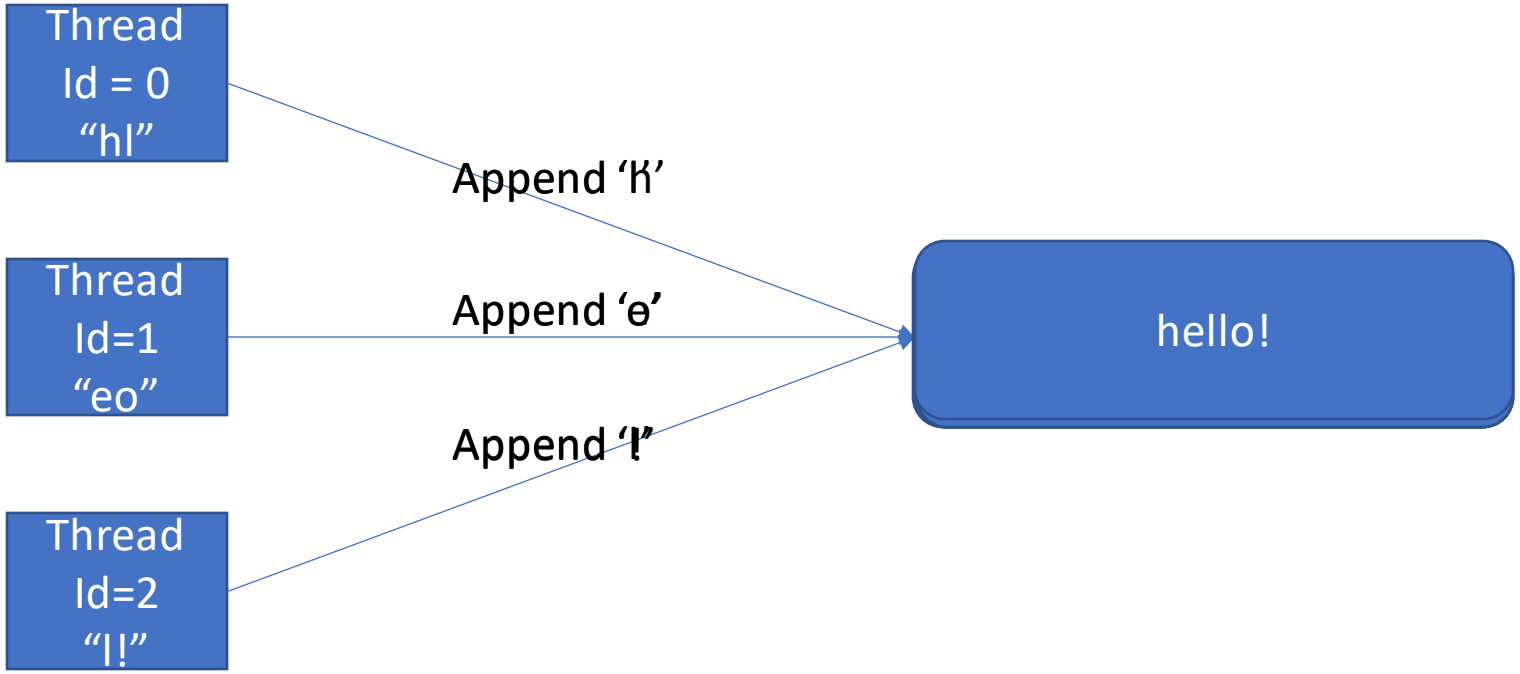
```
1 // Client.java
2 public class Client {
3   ...
4   public void someMethod() {
5     ...
6     // Create an instance of TaskClass
7     TaskClass task = new TaskClass(...);
8
9     // Create a thread
10    Thread thread = new Thread(() -> {
11      ...
12    });
13
14    // Start a thread
15    thread.start();
16    ...
17  }
18  ...
19 }
```

Create a new thread using Anonymous class

What we will do in today's lab

Round Robin String Merge

- Each thread has an ID and its own string segment.
- Threads take turns to append a character in the shared result string.
- In the end, all string segments are merged together.



Tips

- Synchronize Threads
  - You need to synchronize all threads to make sure they append char one by one in a desired order.
  - If your implementation is correct, the merged result should be deterministic in different executions.
  - You can use any technique to synchronize threads, including but not limited to:
    - synchronized keywords
    - wait()/notify()/notifyAll()
    - ReentrantLock
    - Condition
    - Semaphore
    - BlockingQueue
    - …
- Make sure all threads can **exit** after the merge is done.

Submission

- Grading:
  - 1 point: pass all given tests
- Deadline: **Nov 18 23:59.**
- Submit to CASS **lab10**