CO661 Class 1 - Java concurrency basics

This class covers the basics of multi-threaded shared-memory concurrency in Java. It covers some additional practical concerns specific to Java not discussed in the lectures but which you should know about.

Task 1 - Creating threads

Create a simple Java program that prints `Hello` and one second later prints `World!` where this second message is printed by a separate thread. You will need:

- A Java program `Class1.java` with public class `Class1` and some other class, e.g., `Worker`;
- for `Worker` to inherit from `Thread` and implement `public void run()`;
- a `Thread.sleep(long millis)` static method in `Worker`;
- to watch out for `InterruptedException`;
- to `start()` the worker thread from the main method.

Task 2 - Implementing `Runnable` rather than extending `Thread`

Java does not allow multiple inheritance (i.e., we cannot has a class that `extends` multiple classes, or that `extends` multiple times), therefore the general advice in Java is to try to use *interfaces* as much as possible. In Java, the `Runnable` interface[1] can be used in conjunction with the `Thread` class to avoid having to extend `Thread`. A class implementing `Runnable` must provide a public `run` method that returns `void` just as we implemented before, e.g.,

```
class Worker implements Runnable {
    public void run() { /* TODO */ }
}
```

A "runnable" thread then needs to be attached to a thread which can then be started via `start` (or waited for via `join`), e.g.

```
Thread t = new Thread(new Worker());
t.start();
```

Rewrite your solution to Task 1 using this approach.

Task 3 - Locks for fairness

The `TeaRoomInitial.java` example from Lecture 2 used a synchronized method to provide mutual exclusion for the `vend` method. Adapt this code to use a *fair* `ReentrantLock`[2] so that the waiting threads will be given fair access to this method, i.e., to avoid starvation of any thread wanting to access the tea machine. You should be able to get this working with the GUI (`TeaRoomGUI.java`) after also downloading `tearoom.zip`.

Task 4 - What to do with `InterruptedException`

[1] https://docs.oracle.com/javase/10/docs/api/java/lang/class-use/Runnable.html
[2] https://docs.oracle.com/javase/10/docs/api/java/util/concurrent/locks/ReentrantLock.html

Threads can be *blocked* which means they are waiting for some external event and are currently not making any progress e.g., due to `Thread.sleep` (waiting for a timer) or waiting for another thread to finish when using `.join()`, or blocking waiting to receive a message (which we see later). A blocked thread can be interrupted in Java by calling the `.interrupt()` method on the thread object, e.g.

```
Thread t = new Thread(new Worker());
t.start();
t.interrupt();
```

This starts a thread and immediately interrupts it. This mechanism is seen as a “polite” way to externally shut down a thread (rather than just what happens when the JVM is stopped entirely) and can be used to break deadlocks. However, we will usually focus on trying to ensure that deadlocks can never occur anyway.

By catching and handling `InterruptedException`, we can provide some “cleanup” code in the case that this task was interrupted. If we do not have any cleanup code we want to perform, (and maybe we don’t anticipate any interrupts happening our code anyway), the recommended practice is to *reinterrupt* the current thread.

What does it mean to reinterrupt a thread? Inside a thread, there is an `isInterrupted` flag.[3] If the runtime detects that this flag is true then an `InterruptedException` is thrown and the flag is immediately set back to false. Reinterrupting means setting this flag back to true by `Thread.currentThread().interrupt()`. This allows code higher up the stack to be aware of the interruption.

So rather than doing nothing in the `catch` block, the recommendation[4] is to at least do:

```
try {
    // ...
} catch (InterruptedException e) {
    Thread.currentThread().interrupt();
}
```

Edit your previous solutions to use this technique.

[3] `https://docs.oracle.com/javase/10/docs/api/java/lang/Thread.html`

[4] `https://www.ibm.com/developerworks/java/library/j-jtp05236/index.html?ca=drs-`