ECE-480

Basic Concurrency in Java Programming Language

Name: Rituraj Behera

PID: A36602480

Due: 04/04/2009
**Executive Summary:**

Applications used by multiple interactive users or other applications always deal with concurrency. Concurrency needs a lot of hardware and software support depending on the complexity of the design. Our team has to also deal with concurrency issues in our software design using java as the Graphical User Interface (GUI) development toolkit. Even though our design does implement Multi-threading, there are only a few basic concurrency issues that has to be taken care of. Only handful applications like weather information, lighting and clock implement threads in our GUI design for the Home Automation control panel. So, it is necessary to know how to instantiate threads and handle some exceptions and prevent unexpected results during the process of software design. Only the basics of the “Thread” object is covered under java programming language and further research is recommended for handling complex concurrent issues.

**Introduction:**

Concurrency refers to the sharing of resources by multiple interactive users or application programs at the same time. Since the early 21st century, new technologies have evolved and as we head towards the future novel designs are invented and developed. We now get to use applications that are not only fast and easy to use, but can also do multi-tasking. In an “iTouch”, product developed and marketed by Apple Inc., we can listen to music, play games and also download music, all at the same time. Such level of convenience could be achieved only because of the developments done over the years by smart engineers and software developers. Only a decade earlier an application could only handle one execution a time.

Most of the applications developed and engineered these days have the ability to handle multiple users and/or applications. It is easy to realize such applications and systems that can handle various tasks the same time without any complications. But the truth of the matter is that development of such applications requires the support of multi-threading, which complicates things even further. Multithreading is a popular programming and execution model that allows multiple threads to exist within the context of a single process. These threads share the process resources but are able to execute
independently from each other. Multi-threading, therefore can cause undesirable effects and as the program gets larger in the sense that there are more active threads at the same time, synchronizing various threads becomes extremely difficult. As software developers we should always keep concurrency issues in mind when developing any application or program.

**Objective:**

For our team project, we are developing a Home automation control panel for which one of the most important task is to develop a user friendly Graphical User Interface (GUI). To build the GUI, we decided to use the Java Swing development kit for our design. As a part of our GUI design, it was necessary for our team members as software developers to consider that we would have to deal with multi-threading, which inevitably would lead to concurrency issues, even though the Java platform is designed from the ground up to support concurrent programming, with basic concurrency support in the Java programming language and the Java class libraries. I will further discuss in detail the various attributes of concurrency and techniques in java that can be used to take care of undesirable effects. Various scenarios demand for different ways of approaching a problem, but something that is not different is the basics. If we have the basics right, then breaking down a problem and working our way up towards the bigger picture becomes a lot easier. Therefore, it is important to discuss the basic elements one should keep in mind when dealing with multi-threading and concurrency.

**Body:**

- **Processes and Threads**

  Processes and threads are basic elements of an application or a program that supports concurrency. A process can have one or more active thread of execution. So, threads are basically a part of a process and there is at least one thread in each process. An application can have multiple processes or just one process with multiple
threads of execution, but concurrency is possible even on simple systems, without multiple processors or threads. Processes have their own resources and memory space and a single application can be a set of cooperating processes. In Java, most applications run as a single process on a Java Virtual Machine (JVM).

Threads, for simplicity can be also described as processes considering that the process has just one thread of execution, but creating a new thread requires fewer resources as compared to creating a process, since threads are part of a process themselves. If there are multiple threads in a process, the threads then share all the memory and resources of the process in which they are instantiated. Even though this is a very efficient way of having communication between the threads within the process, it can lead to problematic situations and sometimes hard to debug the program. This is because multiple threads within the process share the same resources and synchronizing and updating uncommitted data becomes an issue.

Java platform therefore has a feature of multi-threaded execution in which we start with a single thread called the main thread, and from the main thread additional threads are created if necessary. This leads us to the topic of thread objects and classes that we need to explore in more detail for better understanding of how threads in Java deal with concurrency.

![Diagram](image1.png)

Figure1: A user application/process implementing multi-threading
- **Class thread**

Each thread is associated with an instance of the class “thread” which uses various strategies using thread objects to deal with concurrency with multiple threads. One of the strategies that will be covered is to directly control thread creation and management if the application demands an asynchronous task by just instantiating a new thread. Below is part of a code that that I helped our implement as a part of our senior design project to update the time on the Home Automation control panel. It basically shows how to create a special class to implement and instantiate threads in java.

```java
public class Clock extends JLabel implements Runnable {
    private static final long serialVersionUID = 1L;
    Thread thread; // instantiate thread
    SimpleDateFormat dateFormatter;

    public Clock() {
        ifThread = null;
        dateFormatter = SimpleDateFormat.getTimeInstance(SimpleDateFormat.SHORT);
        this.start();
    }

    public void start() {
        ifThread = new Thread();
        ifThread.start(); // needs to be called on thread to start executing it
    }

    public void stop() {
        ifThread = null;
    }
}
```

**Figure2**: Snapshot of start() function in Clock.java running on Java Swing Eclipse JTK

As you can see from the figure, the class contains “implement Runnable” which defines a single method “start” that is needed to get the thread to start running. Also, the stop function simply stops the thread from running any further till the start function is called on it again. This is just a simple way of taking care of threads and making sure that they are not active unnecessarily as threads do take up memory and processor time slice.
Also, we could use the Subclass “Thread” to achieve the same result, but a
“Runnable” object is considered more flexible approach since it is applicable to the high-
level thread management as compared to “Thread” subclass.

In this case of the clock, the time needs to be updated every minute and so the
thread needs to call the function periodically. The most efficient way to do this is by
using the “sleep” function. This way not only does the processor have the ability to give
more time to the other threads, but also help the thread co-ordinate with various other
threads that might have more important tasks to finish and require the same resources as
that thread. Below is another simple example of how the sleep function helps the thread
activate only every ten seconds to update the time. The parameter passed to the sleep
function is in milliseconds.

```
public void refreshTime()
{
    Date now = new Date();
    String date_out = dataFormatter.format(now);
    this.text = date_out;
}
```

**Figure3:** Snapshot of run() function in Clock.java running on Java Swing
Eclipse JTK
The thread is also capable of handling exceptions, which is another great feature that the java thread class possesses. In the “run” function it declares that the main thread throws an Interrupted Exception “e”. This exception is thrown by “sleep” when other active threads try to interrupt the thread that is sleeping. We can handle such situations in any way desired. That brings us to the topic of Interrupts.

• **Interrupt handling**

An interrupt, when invoked on a thread indicates the thread to stop what it is doing and do something else depending on what we as programmers want to do with the interrupt. Mostly, we handle such situations by stopping the thread completely by making it “Null” or returning back from the function that uses the thread. In java, a thread supports its own Interruption depending on what the current situation is. As an example for our “Team 3” senior design project exceptions are handled by class “weather”, which updates time too. This interrupt is invoked on class “clock” when the weather time needs to be updated too at the same time. The “run” function just calls a return on it, which terminates the thread. This functionality was added later to the “run” function once we needed to handle multi-thread.

• **Join**

Another alternative to this situation, when there needs to be a synchronization between multiple threads, the “join” method helps one thread to wait for the completion of the other and vice versa. The function that needs to be called is simply “ThreadName.join()”.
**Conclusion:**

The issues that we covered in this assignment are just some basic ways by which we can deal with concurrencies and at least gives us a starting point for dealing with tasks and applications that require Multi-threading. As the number of active threads increases, the need to handle those threads becomes necessary since those threads might be reading and writing from the same resource. There can be three different basic kinds of situations that we have to keep in mind when dealing with basic concurrency issues. Those are:

- **Lost updates**, in which two threads, say p1 and p2, read the value from the same resource at the same time then if p1 writes a value back to the resource first and then p2, the value updated by thread p1 is lost.
- **Uncommitted data**, in which the thread p1 might have to commit a data for the thread p2 to use, but thread p2 already has consumed the value p1 had committed the results, and finally,
- **Non-repeatable reads**, in which thread p1 reads a value from the resource and then thread p2 reads from the same resource, modifies the value and either updates or deletes the value and when thread p1 tries to read the value again it is not the same or missing.

There could be many other undesirable results that can be possible with concurrency, but those mentioned above are one of the most common issues that one has to deal with.
References:

- Java tutorials >> Concurrency:

- Concurrency issues:
  http://publib.boulder.ibm.com/infocenter/iwedhelp/v6r0/index.jsp?
topic=com.ibm.db2e.doc/db2e_concurrent_tx.html

- Multi-threading in Applications: