Multi-threading in Java

A Socket Example

Rhandi Martin & Angelos Stavrou
George Mason University
Processes vs. Threads

- Both are units of execution

- Process has a self-contained and independent execution environment

- Threads are light-weight processes
  - Share some resources and variables and thus communicate easily
  - No need for Inter-process communication, IPC
Threads

- A Class that *extends* Thread

- A Class that *implements* Runnable

- Must contain method:
  ```java
  public void run() {...}
  ```

- Is initialized implicitly by `run()` or explicitly by calling `start()` ...which then calls `run()`

- Member of `java.lang` package
Useful Properties of class Thread

- **Constructor**
  - Thread(ThreadGroup, parent, String name)

- **start()**
  - When sub-classing, use to call Thread.run()

- **activeCount()**
  - Returns number of active threads (independent of group)

- **enumerate(Thread[])**
  - Lists all threads in same group
Thread Properties Cont’d

- **sleep(long msec)**
  - pauses execution, throws InterruptedException

- **join()**
  - Waits for the thread to finish

- **yield()**
  - Yields the processing back to the system allowing other threads/processes to execute
Simple Example – Runnable

```java
class MyThread implements Runnable {
    Thread t;
    MyThread () {
        t = new Thread(this);
        t.start();
    }

    public void run() {
        System.out.println("Child thread started");
        System.out.println("Child thread terminated");
    }
} // end class

class Demo {
    public static void main (String[] args) {
        new MyThread();
        System.out.println("Main thread started");
        System.out.println("Main thread terminated");
    } // end main
} // end class
```

Adopted from devarticles.com
Copyright 2005 McGraw-Hill/Osborne
Simple Example – extends

```java
class MyThread extends Thread {
    MyThread()
    {
        super("My thread");
    }

    public void run() {
        System.out.println("Child thread started");
        System.out.println("Child thread terminated");
    }
}

class Demo {
    public static void main (String args[]){
        new MyThread().start();
        System.out.println("Main thread started");
        System.out.println("Main thread terminated");
    }
}
```

Adopted from devarticles.com
Copyright 2005 McGraw-Hill/Osborne
Homework 0: Problems & Requirements

- Blocking calls:
  - ServerSocket.accept()
  - BufferedReader.readLine()

- Update all other clients when one client says something
  - Requires a construct to keep track/store all client connections
Approaches

How do you keep track of a number of related threads?

Options:
- Maintain an array: Impractical
- Main a list: Doable
- ThreadGroup: The Java way!
Why Multithreading?

- Modularity
- Shared Resource Requirement
- Circumventing blocking calls
- Only choice in Java
Multithreading Steps

- Determine the methods the program must call
- Are any of them blocking?
  - Yes. Then handle some/all of these in threads
- Create ThreadGroup for similar threads
- Pass whatever structures necessary to the Thread
- Add them to the ThreadGroup
Useful Properties of class ThreadGroup

- `activeCount()`
- `enumerate(Thread[])`
Problems – Synchronization

There are detriments to Thread behavior and concurrency issues such as sharing resources.

- Inconsistencies can occur when threads access and mutate (read and write) a shared resource (variable or object)
Problems – Liveness

- Threads may be denied access to a resource (starvation),
- May wait forever for another thread to complete (live-lock),
- Threads may depend on each or multiple resources in order to progress (dead-lock)
Synchronization Solutions

- `java.util.concurrent` package
  - Semaphore

- `java.util.concurrent.*` packages
  - Atomic classes
  - Blocking classes
  - Condition interface
  - Lock classes

- ‘synchronized’ keyword
synchronized method

public synchronized int increment(int c) {
    return ++c;
} //end method

synchronized block

int c = 0;
public String toString () {
    System.out.println("Waiting to enter sync block");
    synchronized(c) {
        System.out.println(++c);
    } //end sync block
    System.out.println("Exited sync block");
} //end main
Multi-threading Socket Example

- Simple Multi-threading Server and Client

http://cs.gmu.edu/~astavrou/courses/CS_571_F09/jThreadTutorial/