CMSC 132:
OBJECT-ORIENTED PROGRAMMING II

Threads in Java

Department of Computer Science
University of Maryland, College Park
Daemon Threads

• Java threads types
  • User
  • Daemon
    • Provide general services
    • Typically never terminate
    • Call setDaemon() before start()

• Program termination
  • All user threads finish
  • Daemon threads are terminated by JVM
Threads – Scheduling

• Scheduler
  • Determines which runnable threads to run
    • When context switching takes place
  • Can be based on thread priority
  • Part of OS or Java Virtual Machine (JVM)

• Scheduling policy
  • Non-preemptive (cooperative) scheduling
  • Preemptive scheduling
Threads – Non-preemptive Scheduling

- Threads continue execution until
  - Thread terminates
  - Executes instruction causing wait (e.g., IO)
  - Thread volunteering to stop (invoking yield or sleep)
Threads – Preemptive Scheduling

- Threads continue execution until
  - Same reasons as non-preemptive scheduling
  - Preempted by scheduler
Thread Scheduling Observations

- Order thread is selected is indeterminate
  - Depends on scheduler
- Scheduling may not be fair
  - Some threads may execute more often
- Thread can block indefinitely (starvation)
  - If other threads always execute first
- Your code should work correctly regardless the scheduling policy in place
Java Thread Example

public class ThreadNoJoin extends Thread {
    public void run() {
        for (int i = 0; i < 3; i++) {
            try {
                sleep((int)(Math.random() * 5000)); // 5 secs
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println(i);
        }
    }
    public static void main(String[] args) {
        Thread t1 = new ThreadNoJoin();
        Thread t2 = new ThreadNoJoin();
        t1.start();
        t2.start();
        System.out.println("Done");
    }
}
Java Thread Example – Output

• Possible outputs
  • 0,1,2,0,1,2,Done  // thread 1, thread 2, main()
  • 0,1,2,Done,0,1,2  // thread 1, main(), thread 2
  • Done,0,1,2,0,1,2  // main(), thread 1, thread 2
  • 0,0,1,1,2,Done,2  // main() & threads interleaved
Thread Class – join( ) Method

- Can wait for thread to terminate with join( )
- Method prototype
  - public final void join( )
    - Returns when thread is done
    - Throws InterruptedException if interrupted
public class ThreadJoin extends Thread {
    public void run() {
        for (int i = 0; i < 3; i++) {
            try {
                sleep((int)(Math.random() * 5000)); // 5 secs
            } catch (InterruptedException e) { e.printStackTrace(); }
            System.out.println(i);
        }
    }
    public static void main(String[] args) {
        Thread t1 = new ThreadJoin();
        Thread t2 = new ThreadJoin();
        t1.start();
        t2.start();
        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) { e.printStackTrace(); }
        System.out.println("Done");
    }
}
About Join

- Important: You will limit the concurrency level if you do not start/join correctly.

- Suppose you want to run many threads concurrently. **Start them all and then execute the join for each one.** Do not start one thread, then join on that thread, start the second one, join on that thread, etc.

- The following is WRONG!

  ```java
t1.start()
t1.join()
t2.start()
t2.join()
  ```

- Feel free to use arrays, sets, etc., to keep track of your threads.
About Threads

- **Common mistake** → calling the run() method. If you want to run a thread you must execute start() and not call the run() method; the run() method is called for you.

- **Thread.sleep** → Suppose you have a thread object reference (t1) and invoke t1.sleep(2000). Which thread will be sleeping for 2 seconds? It will not be t1.
Thread Example

• Swing uses a single-threaded model
• Long computations in the EDT freezes the GUI
• Example: Progress Bar Example