Object Oriented Programming

Week 10 Part 1
Threads

Lecture

- Concurrency, Multitasking, Process and Threads
- Thread Priority and State
- Java Multithreading
- Extending the Thread Class
- Defining a Class that Implements Runnable
- Adding Another Thread
- Thread Coordination

Concurrency, Multitasking, Processes and Threads

Concurrency

- *Concurrency* is doing multiple things at the same time.
  - E.g. Printing while editing a document
  - E.g. Browser loading images with accepting input
- *Multitasking* is running multiple tasks in a program, requiring
  - Starting multiple programs
  - Coordinating the programs
  - Ending the programs

Processes and Threads

- *Processes* have a self-contained execution environment
  - Separate memory and I/O
  - More computation to start
  - Less interference between processes
  - Java usually runs one process
- *Threads* is concurrency that shares an execution environment
  - Running in same memory with same I/O
  - Less computation to start
  - Treads can interfere with each other
  - Every process has at least one thread

Multitasking

- *Multitasking* is running multiple processes requiring
  - Starting multiple programs
  - Coordinating the programs
  - Ending the programs
- *Preemptive* multitasking
  - Each process is given a time slot to use the CPU
  - The process is preempted when
    - The time slot is over
    - The process needs I/O (i.e. needs another process to run to read or write data)
- *Cooperative* multitasking (now rare)
  - Programs yield to other programs

Multithreading

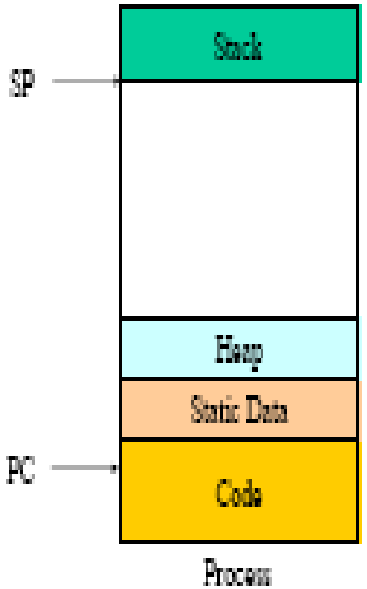
- Each process runs multiple threads of control
  - A *thread of control* is sequence of instructions that runs in a process (i.e. a program)
- A process may run multiple threads of control by trading off between them.
- A thread of execution is a program executed independently of other parts of the program.

Process VS Thread

S.No	Process	Thread
1	No Sharing of Memory	Sharing of Memory and other data structures
2	Can not Corrupt Data structures	Can Corrupt Data Structures
3	Context switching is Expensive	Context Switching is Chaeper

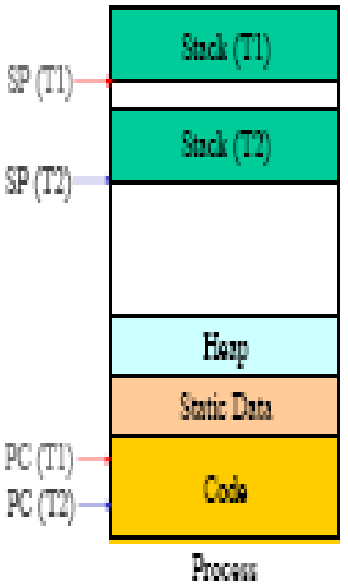
What is a Process?

- Execution context
  - Program counter (PC)
  - Stack pointer (SP)
  - Data registers
- Code
- Data
- Stack



What is a Thread?

- Execution context
  - Program counter (PC)
  - Stack pointer (SP)
  - Data registers



Thread Priority and State

Thread Priority

- Each thread has a priority
  - Priority is set by the setPriority(int newPriority) method
- Thread priorities are integers between 1 and 10
  - 1 is the minimum priority
  - 10 is the maximum priority
- The scheduler chooses the highest priority runnable thread when choosing next thread

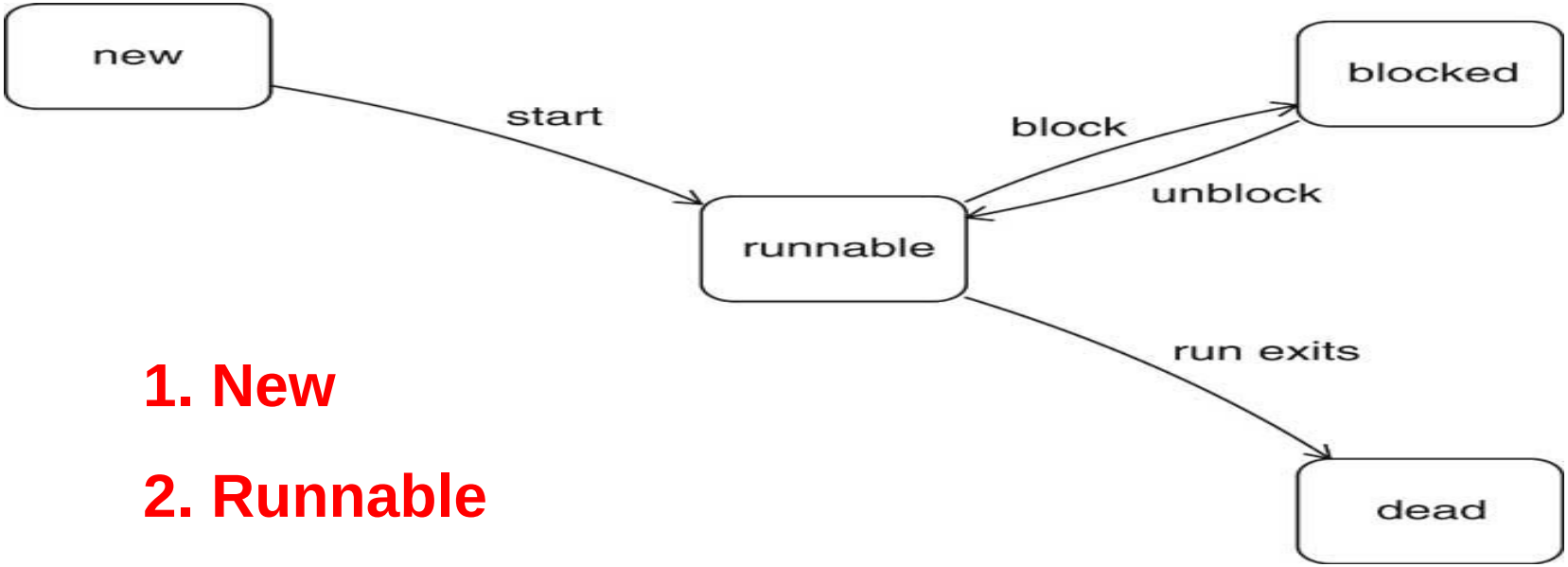
Thread State

- Thread may be in one of four states: new, runnable, blocked, and dead
- A Thread, thread, is new when it is created
- When thread.start() is called, it moves from new to runnable.
- When it the thread.run() method terminates, it moves from runnable to dead
- When a thread blocks, it moves from runnable to blocked.
- When the reason for the block goes away, it moves from blocked to runnable

Reasons from Blocking

- A thread moves from runnable to blocked if
  - It sleeps
  - It is waiting for I/O
  - It is waiting to acquire a lock
  - It is waiting for a condition

Thread State Diagram



1. New
2. Runnable
3. Blocked
4. Dead

Scheduling threads

- The scheduler starts a new thread when
  - A thread has used up its time slot
  - A thread has become blocked
  - A thread with a higher priority has become runnable
- The scheduler chooses the highest priority thread from the runnable threads.

Terminating Threads

- Threads terminate when the run() method of that thread exits
- To end a Thread, t
  1) Call t.interrupt, which sets a flag
  2) The Thread t must respond to the interrupt and exit.
- Interrupting the thread and having it exit allows the thread to clean up.

Java Multithreading

Java Multithreading

- Each thread is associated with a Thread object
  - Multitasking multithreading
  - Virtual machine executes each thread for a short time slice
  - Thread schedule activates and deactivates threads.

Thread Class

- In java.lang package
- Constructors
  - Thread()
  - Thread(String name)
  - Thread(Runnable r)
  - Thread(Runnable r, String name)

Thread methods

- getName(): returns thread name
- getPriority(): returns thread priority
- setPriority(): sets the thread's priority
- isAlive(): return true if thread is alive; false o.w.
- run(): entry point for thread (like main() for threads)
- sleep(long ms): sleep for ms milliseconds
- start(): start a thread
- interrupt(): interrupt a thread
- isInterrupted: true if interrupted; false 0.w.

Static Thread Methods

- currentThread(): the thread that is currently running

Runnable Interface

- Requires definition of the run() method

```
public interface Runnable
{
  public void run();
}
```

Defining Threads

- Two ways
  - Define a class that implements Runnable and pass it to the Thread constructor
  - Define a class that is a subclass of Thread
- In either case you need to define the run() method.

```
class RThread implements Runnable
{
    public void run() {

    }
}
```

```
class EThread extends Thread
{
    public void run(){

    }
}
```

Extending Thread Class

Example Extended Class

Constructor: sets name and sleep time

Run method: prints four messages

Sleeps sleep time milliseconds

Prints a message

```
package example.threads;

public class SubclassThread extends Thread {


    long sleepTime = 0;

    SubclassThread(long sleepTime, String name) {
        this.sleepTime = sleepTime;
        this.setName(name);
    }

    @Override
    public void run() {
        for (int i = 0; i < 4; i++) {
            try {
                Thread.sleep(sleepTime);
            } catch (InterruptedException e) {
                SimpleThreads.printMessage("Interrupted: "
                                        + e.getMessage());
            }
            SimpleThreads.printMessage("loop " + i);
        }
    }
}
```

Example Using Extended Class

Returns program start time

Prints thread name, time and note

Time elapsed is current time – start time

Define main()

Set start time

Print that main() started

Create a thread and start it

Print that main finished

```
public class SimpleThreads {

    static long startTime = 0;

    public static long getStartTime() {
        return startTime;
    }

    public static void printMessage(String message) {
        System.out.format("%s, running: %d ms, %s%n",
                Thread.currentThread().getName(),
                System.currentTimeMillis() - startTime,
                message);
    }

    public static void main(String args[])
            throws InterruptedException {
        startTime = System.currentTimeMillis();
        printMessage("Starting main()");

        SubclassThread t1 = new SubclassThread(1000, "Thread 1");
        t1.start();
        printMessage("Thread 1 started");
        printMessage("Ending main()");
    }
}
```

Example Output

Main starts

Starts thread

Main is done

Thread prints four times

```
main, running: 0 ms, Starting main()
main, running: 5 ms, Thread 1 started
main, running: 5 ms, Ending main()
Thread 1, running: 1010 ms, loop 0
Thread 1, running: 2011 ms, loop 1
Thread 1, running: 3012 ms, loop 2
Thread 1, running: 4018 ms, loop 3
```

The main thread finishes before the thread it started prints its first line. The JVM runs the thread until it completes

Defining Class that Implements Runnable

Example Runnable Class

Only difference is it implements Runnable

```
lic class RunnableThread implements Runnable {

    long sleepTime = 0;

    RunnableThread(long sleepTime) {
        this.sleepTime = sleepTime;
    }

    @Override
    public void run() {
        for (int i = 0; i < 4; i++) {
            try {
                Thread.sleep(sleepTime);
            } catch (InterruptedException e) {
                SimpleThreads.printMessage("Interrupted: "
                                            + e.getMessage());
            }
            SimpleThreads.printMessage("loop " + i);
        }
    }
}
```

Example Using Runnable Class

```java
public class SimpleThreads {

    static long startTime = 0;

    public static long getStartTime() {
        return startTime;
    }

    public static void printMessage(String message) {
        System.out.format("%s, running: %d ms, %s%n",
                Thread.currentThread().getName(),
                System.currentTimeMillis() - startTime,
                message);
    }

    public static void main(String args[])
            throws InterruptedException {
        startTime = System.currentTimeMillis();
        printMessage("Starting main()");

        Thread t1 = new Thread(new RunnableThread(1000), "Thread 1");
        t1.start();
        printMessage("Thread 1 started");

    }
}
```

Only difference is it calls the Thread constructor and passing in Runnable class

Adding Another Thread

Example: Adding Second Thread

Thread 1 prints once a second

Thread 2 prints once a half second

```
public static void main(String args[])
        throws InterruptedException {
    startTime = System.currentTimeMillis();
    printMessage("Starting main()");

    Thread t1 = new Thread(new RunnableThread(1000), "Thread 1");
    t1.start();
    printMessage("Thread 1 started");

    Thread t2 = new Thread(new RunnableThread(500), "Thread 2");
    t2.start();
    printMessage("Thread 2 started");

    printMessage("Ending main()");
}
```

Example: Second Thread Output

Main starts two thread and finishes

Thread 2 prints first time

Thread 1 prints first time

Thread 2 prints second and third time

Thread 1 prints second time

Thread 2 fourth and last time

Thread 1 third and fourth, finishing

```
main, running: 0 ms, Starting main()
main, running: 7 ms, Thread 1 started
main, running: 7 ms, Thread 2 started
main, running: 8 ms, Ending main()
Thread 2, running: 509 ms, loop 0
Thread 1, running: 1008 ms, loop 0
Thread 2, running: 1013 ms, loop 1
Thread 2, running: 1518 ms, loop 2
Thread 1, running: 2014 ms, loop 1
Thread 2, running: 2020 ms, loop 3
Thread 1, running: 3015 ms, loop 2
Thread 1, running: 4018 ms, loop 3
```

Coordinating Threads

Coordinating Threads

- Our threads do not wait for each other
- Suppose we want the main function to wait for one of the other functions
  - The join() method causes the calling thread to wait until the thread on which the method is called terminates
  - Final void join() throws InterruptedException

Join Example 1: t2.join()

```
public static void main(String args[])
        throws InterruptedException {
    startTime = System.currentTimeMillis();
    printMessage("Starting main()");

    Thread t1 = new Thread(new RunnableThread(1000), "Thread 1");
    t1.start();
    printMessage("Thread 1 started");

    Thread t2 = new Thread(new RunnableThread(500), "Thread 2");
    t2.start();
    printMessage("Thread 2 started");

    t2.join();
    printMessage("Ending main()");
}
```

Wait until t2 finishes

Example 1: Output

```
main, running: 0 ms, Starting main()
main, running: 6 ms, Thread 1 started
main, running: 6 ms, Thread 2 started
Thread 2, running: 511 ms, loop 0
Thread 1, running: 1007 ms, loop 0
Thread 2, running: 1015 ms, loop 1
Thread 2, running: 1518 ms, loop 2
Thread 1, running: 2008 ms, loop 1
Thread 2, running: 2020 ms, loop 3
main, running: 2021 ms, Ending main()
Thread 1, running: 3014 ms, loop 2
Thread 1, running: 4017 ms, loop 3
```

Main terminates when Thread 2 finishes

Example 2: Waiting for t1 to finish

```
public static void main(String args[])
        throws InterruptedException {
    startTime = System.currentTimeMillis();
    printMessage("Starting main()");

    Thread t1 = new Thread(new RunnableThread(1000), "Thread 1");
    t1.start();
    printMessage("Thread 1 started");

    Thread t2 = new Thread(new RunnableThread(500), "Thread 2");
    t2.start();
    printMessage("Thread 2 started");

    t1.join();
    printMessage("Ending main()");
}
```

Wait until t1 finishes

Example 2: Output

```
main, running: 0 ms, Starting main()
main, running: 5 ms, Thread 1 started
main, running: 6 ms, Thread 2 started
Thread 2, running: 508 ms, loop 0
Thread 1, running: 1007 ms, loop 0
Thread 2, running: 1014 ms, loop 1
Thread 2, running: 1519 ms, loop 2
Thread 1, running: 2008 ms, loop 1
Thread 2, running: 2025 ms, loop 3
Thread 1, running: 3014 ms, loop 2
Thread 1, running: 4018 ms, loop 3
main, running: 4019 ms, Ending main()
```

Main terminates when Thread 1 finishes