8. Concurrency

1. Threads

Concurrency

- A single application is often expected to do more than one thing at a time
- Software that can do such things is known as *concurrent* software
- Since version 5.0, the Java platform has also included high-level concurrency APIs

Processes

- A process has a self-contained execution environment
- A process generally has a complete, private set of basic run-time resources (e.g own memory space)
- A Java application can create additional processes using a ProcessBuilder object.
- **Multiprocess applications are beyond the scope of this lesson**

Threads I

- Threads are sometimes called *lightweight processes*
- Both processes and threads provide an execution environment, but creating a new thread requires fewer resources than creating a new process.
- Threads exist within a process — every process has at least one thread

Threads II

- Threads share the process's resources, including memory and open files
- From the application programmer's point of view, you start with just one thread, called the *main thread*
- This thread has the ability to create additional threads

Defining a Thread

- An application that creates an instance of Thread must provide the code that will run in that thread:
  - *Provide a Runnable object.*
  - *Create Thread Subclass.*

Runnable Object

- The Runnable interface defines a single method, run, meant to contain the code executed in the thread
- The Runnable object is passed to the Thread constructor
- Thread's start method is called

Runnable Object Example

```
public class HelloRunnable implements
    Runnable {
     public void run() {
        System.out.println("Hello from a thread!");
     }
    public static void main(String args[]) {
      (new Thread(new HelloRunnable())).start();
    }
}
```

Runnable Object in Java 8

```
public static void main(String args[]) {
    Runnable r =
     () -> System.out.println("Hello world!");
     new Thread(r).start();
}
```

Thread Subclass

- The Thread class itself implements Runnable, though its run method does nothing
- An application can subclass Thread, providing its own implementation of run

Thread Subclass Example

```
public class HelloThread extends Thread {
   public void run() {
   System.out.println("Hello from a thread!");
   }
   public static void main(String args[]) {
    (new HelloThread()).start();
   }
}
```

Runnable vs Thread Subclass

- A Runnable object employment is more general, because the Runnable object can subclass a class other than Thread
- Thread subclassing is easier to use in simple applications, but is limited by the fact that your task class must be a descendant of Thread
- A Runnable object is applicable to the high-level thread management APIs

Pausing Execution with Sleep

- Thread.sleep causes the current thread to suspend execution for a specified period
- *This is an efficient means of making processor time available to the other threads of an application or other applications that might be running on a computer system*
- The sleep period can be terminated by interrupts

Sleep Example

```
public class SleepMessages {
    public static void main(String args[]) throws
    InterruptedException {
     String importantInfo[] = { "Mares eat oats",
          "Does eat oats", "Little lambs eat ivy",
          "A kid will eat ivy too"};
     for (int i = 0; i < importantInfo.length; i++) {
          Thread.sleep(4000);
          System.out.println(importantInfo[i]);
          }
    }
}
```

Thread Race Example

- Create two classes: first implements Runnable interface, and second extends Thread class. Method run() in both classes prints thread and iteration numbers and sleeps in some seconds.

Thread Race Example

- See 811ThreadRace project for the full text.

Thread Terminations

- A thread terminates when:
  - its run method returns, by executing a return statement
  - after executing the last statement in the method body
  - if an exception occurs that is not caught in the method
- The interrupt method can be used to request termination of a thread

Interrupted Status

- When the interrupt method is called on a thread, the interrupted status of the thread is set
- This is a boolean flag that is present in every thread
- Each thread should occasionally check whether it has been interrupted

How to Check Interrupted Status

- To find out whether the interrupted status was set, first call the static Thread.currentThread method to get the current thread and then call the isInterrupted method:

```
while (!Thread.currentThread().isInterrupted())
{
   do more work
}
```

InterruptedException

- If a thread is blocked, it cannot check the interrupted status
- This is where the InterruptedException comes in
- When the interrupt method is called on a thread that blocks on a call such as sleep or wait, the blocking call is terminated by an InterruptedException

InterruptedException Example

```
for (int i = 0; i < importantInfo.length; i++) {
    // Pause for 4 seconds
   try { Thread.sleep(4000); }
   catch (InterruptedException e) {
    return;
    }
    System.out.println(importantInfo[i]);
}
```

Joins

- The join method allows one thread to wait for the completion of another
- If t is a Thread object whose thread is currently executing, t.join() causes the current thread to pause execution until t's thread terminates
- Overloads of join allow the programmer to specify a waiting period
- join responds to an interrupt by exiting with an InterruptedException

Join Exercise

- Modify 811ThreadRace project so that first thread should wait for second thread finishing

ThreadRace Class

```
public static void main(String[] args) throws
    InterruptedException{
     ThreadRunnab r = new ThreadRunnab();
     Thread t1 = new Thread(r);
     Thread t2 = new ThreadThread();
     r.setThread(t2);
     t1.start();
     t2.start();
}
```

Join Exercise

- See 812ThreadJoin project for the full text.

Thread Priority

- public final void setPriority(int newPriority) - changes the priority of this thread
- public final int getPriority() - returns this thread's priority

Sharing Resources Example

- Try to generate Fibonacci series in one thread and print its values in another thread

Sharing Resources Example

- See 813Resources project for the full text.

Manuals

- http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html