

Threads and Thread Synchronization

Threaded Programs

- a *thread* is a *flow of control* in a program.
- It is possible to have many *flows* at the same time!
- Consider what a browser does :
 - retrieve a bunch of documents at once (images found on a web page).
 - look for user input in the address/location textbox
 - render web page
 - run applets/flash/etc.

Java Threads

- Implementation is up to the JVM:
 - it's not defined whether a Java thread is mapped to a something else at the OS level (a native thread or process), or whether the JVM does timesharing itself.
- Every Java program is a threaded program!
 - garbage collection runs as a thread.

Creating a thread

- Create a **java.lang.Thread** object.
 - creates a new thread, but doesn't tell the thread what to do.
 - In other languages we tell a new thread what method to run - in Java we can't do this (no pointers to methods!).
 - In Java we need to tell the new thread what **Object** to run - the **Object** must implement the **Runnable** interface.

The **Runnable** interface

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

- A class that implements **Runnable** must have a **run ()** method.
 - this is the code that will be run by the thread.

Some Methods of Thread class

<code>start()</code>	Call the <code>run()</code> method
<code>yield()</code>	Give other threads some time.
<code>sleep()</code>	Sleep for specified time.
<code>interrupt()</code>	Wake thread up
<code>join()</code>	Wait for thread to finish.

Thread Lifetime

- A thread is done when any of:
 - the `run ()` method returns.
 - uncaught exception occurs.
 - someone calls the `threads stop ()` method (which is deprecated and shouldn't be used!).
- A program is done when all (non-daemon) threads have finished.

Simple Example

```
class Foo implements Runnable {  
    public void run() {  
        System.out.println("I am a Foo thread");  
    }  
}
```

```
class Simple {  
    public static void main(String [] args) {  
        Thread t = new Thread( new Foo());  
        t.start();  
        System.out.println("Main is done");  
    }  
}
```

Better Example

- ThreadPlay.java (on the web).
 - two different run() methods.
 - one calls yield() to allow other threads a chance.

Another way to create a thread

- Extend the class `Thread`
- Override the method `run ()`
- Start the thread by calling the `start ()` method.

Example of extending Thread

```
class Blah extends Thread {
    public void run() {
        System.out.println("I am alive!");
    }

    public static void main(String[] args) {
        Blah b = new Blah();
        b.start();
    }
}
```

Scheduling

- It is *not defined* whether thread scheduling is *pre-emptive*.
 - threads are interrupted at regular intervals and CPU time given to other threads.
 - Might happen, might not.
 - Don't write code that makes assumptions!

Synchronization

- In some situations, threads depend on each other:
 - one thread should wait until another is done.
 - threads share a data structure.
 - only one should mess with the d.s. at a time!
 - threads share any *resource* (including code!).
- Java supports synchronization in a number of ways:
 - Through the **synchronized** keyword.
 - **wait()** and **notify()** methods.

Synchronization Example

- Check out the Sun tutorial on Threads for a good synchronization example:
 - producer / consumer system.

Threads? Who cares?

- All Java programmers must be aware of threads!
 - GUI stuff is event-driven - lots of threads.
 - necessary to provide *good* response time.
 - Some containers are thread-safe (synchronized), some are not.
 - Don't just always use synchronized containers, as there is an overhead associated with synchronization!