Threads in Java

Threads are part of the Java language. There are two ways to create a new thread of execution.

- Declare a class to be a subclass of `Thread`. This subclass should override the run method of class `Thread`. An instance of the subclass can then be allocated and started.
- The other way to create a thread is to declare a class that implements the `Runnable` interface. That class then implements the `run` method. An instance of the class can then be allocated, passed as an argument when creating `Thread`, and started.

Thread examples in Java

- Create threads by extending the `Thread` class:
  threads/java/ThreadsExample.java
- Create threads by implementing the `Runnable` interface:
  threads/java/RunnableExample.java
- Create as many threads as the system would allow:
  threads/java/MaxThreads.java

In-class Exercise (1)

Given the following classes:

```
class Worker extends Thread {...}
class BusyBee implements Runnable {...}
```

Which of the following statements (one or more) correctly creates and starts running a thread?

1. `new Worker();`
2. `new Worker().start();`
3. `new Thread(new BusyBee()).start();`
4. `new Thread(new BusyBee());`

In-class Exercise (2)

Let's get together and compute? Consider the following code and choose the statement that best explains how the code runs.

```
public class ComputeALot implements Runnable  {
    public void run() {
        /* ... */
    }
    public static void main (String args[]) {
        ComputeALot playground = new ComputeALot();
        Thread [] tid = new Thread[4];
        for (int i=0; i < tid.length; i++)
            tid[i] = new Thread(playground);
    }

}
```

1. The code creates and runs four threads that all run the method `run`
2. The code creates four threads but they don't do anything.
3. The code creates three threads but they don't do anything.
4. The code creates and runs three threads that all run the method named `run`

Relevant Java Classes/Interfaces

- See documentation for basic classes: `java.lang.Thread`, `java.lang.ThreadGroup` and `java.lang.Runnable` interface.
- See the `Object` class for synchronization methods.
- A collection of threads that work together is known as a thread pool. For automatic management of thread pools, see: `Executor` interface from `java.util.concurrent` package.

Controlling Threads

- `start()`
- `stop()`, `suspend()` and `resume()` *Note: These have been deprecated in the current version of java*
- `sleep()`.
- `interrupt()`: wake up a thread that is sleeping or blocked on a long I/O operation
  - Thread Interrupt example
- `join()`: causes the caller to block until the thread dies or with an argument (in millisecs) causes a caller to wait to see if a thread has died

A Thread's Life

A thread continues to execute until one of the following thing happens.

- it returns from its target `run()` method.
- it's interrupted by an uncaught exception.
- it's `stop()` method is called.

What happens if the `run()` method never terminates, and the application that started the thread never calls the `stop()` method?

- *The thread remains alive even after the application has finished!* (so the Java interpreter has to keep on running...)

Daemon Threads

- Useful for simple, background periodic tasks in an application.
- The `setDaemon()` method marks a thread as a daemon thread that should be killed and discarded when no other application threads remain.

```
class BackgroundTask extends Thread {
    BackgroundTask() {
        setDaemon( true);
        start();
    }
    public void run() {
        //perform background tasks
        ...
    }
}
```

Race Conditions with Java Threads

- Multiple threads in Java will have race conditions (read/write conflicts based on time of access) when they run. We have to resolve these conflicts.
- Example of a race condition: `Account.java`, `TestAccount.java`
- Another example of a race condition: `PingPong.java`

Thread Synchronization in Java (1)

- Java has `synchronized` keyword for guaranteeing mutually exclusive access to a method or a block of code. Only one thread can be active among all synchronized methods and synchronized blocks of code in a class.

```
//Only one thread can execute the update method at a time.
synchronized void update() { //... }

// Access to individual datum can also be synchronized.
// The object buffer can be used in several classes, implying
// synchronization among methods from different classes.

synchronized(buffer) {
    this.value = buffer.getValue();
    this.count = buffer.length();
}
```

- Every Java object has an implicit monitor associated with it to implement the synchronized keyword. Inner class has a separate monitor than the containing outer class.
- Java allows Rentrant Synchronization, that is, a thread can reacquire a lock it already owns. For example, a synchronized method can call another synchronized method.

Synchronization Example

- Race conditions: `Account.java`, `TestAccount.java`
- Thread safe version using `synchronized` keyword: `RentrantAccount.java`

Thread Synchronization in Java (2)

- The `wait()` and `notify()` methods (of the `Object` class) allow a thread to give up its hold on a lock at an arbitrary point, and then wait for another thread to give it back before continuing.
- Another thread must call `notify()` for the waiting thread to wakeup. If there are other threads around, then there is no guarantee that the waiting thread gets the lock next. *Starvation* is a possibility. We can use an overloaded version of `wait()` that has a timeout.
- The method `notifyAll()` wakes up all waiting threads instead of just one waiting thread.

Example with wait()/notify()

```
class MyThing {
  synchronized void waiterMethod() {
    // do something
    // now we need to wait for the notifier to do something
    // this gives up the lock and puts the calling thread to sleep
    wait();
    // continue where we left off
  }

  synchronized void notifierMethod() {
    // do something
    // notifier the waiter that we've done it
    notify();
    //do more things
  }

  synchronized void relatedMethod() {
    // do some related stuff
  }
}
```

Synchronized Ping Pong using `wait()`/`notify()`

See example `threads/SynchronizedPingPong.java`

MS Windows API for Threads

In MS Windows, the system call interface is not documented. Instead the MS Windows API is documented, which helps with being able to run programs portably across multiple versions of the MS Windows operating systems.
Creating a thread gives you a *handle* that is used to refer to the actual object that represents a thread.

- CreateThread(...). Create a new thread and start running the start function specified in the new thread.
- ExitThread(...), GetExitCodeThread(...), TerminateThread(...), GetCurrentThreadId(), GetCurrentThread().
- WaitForSingleObject(...), WaitForMultipleObjects(...). These can be used to wait for either a process or a thread.

Get detailed information from
`http://msdn.microsoft.com/library/`

MS Windows API for Threads


```
HANDLE WINAPI CreateThread(
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
  SIZE_T dwStackSize,
  LPTHREAD_START_ROUTINE lpStartAddress,
  LPVOID lpParameter,
  DWORD dwCreationFlags,
  LPDWORD lpThreadId
);
//prototype for a thread start method
DWORD WINAPI ThreadProc(
  LPVOID lpParameter
);

DWORD WINAPI GetCurrentThreadId(void);
HANDLE WINAPI GetCurrentThread(void);

VOID WINAPI ExitThread(
  DWORD dwExitCode
);
BOOL WINAPI TerminateThread(
  HANDLE hThread,
  DWORD dwExitCode
);
```

Compiling Multithreaded Programs in Visual Studio

- The `/MT` or `/MTd` flags to the compiler in Visual Studio enable multi-threaded behavior. These are turned on by default in Visual Studio 2005 onward.
- Go to the project properties. In the *Property Pages* dialog box, click the *C/C++* folder. Select the *Code Generation* page. From the *Runtime Library* drop-down box, select the appropriate Multi-threaded option (it should already be the default). Click *OK*.
- See the following page for details on the various multi-threading and related flags for the C/C++ compiler: `http://msdn.microsoft.com/en-us/library/2kzt1wy3(v=vs.110).aspx`

MS Windows API Examples

- `ms-windows/threads/thread-hello-world.c`
- `ms-windows/threads/thread-scheduling.c`
- `ms-windows/threads/thread-test.c`
- and others in the ms-windows/files-processes examples folder....