Java Threads: An Introduction

Ed Armstrong
sharcnet.ca
2014
(1) What are threads?
(2) How to create and control a thread.
(3) Synchronization and signals.
(4) Example
What is a Thread?

- Light-weight process that can run concurrently with other threads.
- All Threads share a node, sharing memory.
- Main() is a thread.
- Java garbage collection is a thread.
Single Core
Multi-Core (4 Core)
How to create and control threads.

- Thread class and runnable interface
- thread.start()
- thread.join()
- thread.sleep()
- Keyword : 'volatile'
Creating a thread

There are 2 ways to create a thread in Java

(1) Extend the 'Thread' class.

(2) Implement the 'Runnable' interface.
Method #1
Extend the 'Thread' class

```java
public class Example1 extends Thread{

    public static void main(String[] args){
        Example1 thread = new Example1();
        thread.start();
    }

    public void run(){
        ...
    }
}
```
public class Example1 implements Runnable{

    public static void main(String[] args) { 
        Example1 example = new Example1();
        Thread thread = new Thread(example);
        thread.start();
    }

    public void run() { 
        .... do something ....
    }
}
Common Methods

**start()** : Causes a thread to begin execution; the Java Virtual Machine calls the 'run' method of the interface.

**join()** : Blocks while waiting for a particular thread to terminate.

**sleep(long msec)** : Causes the currently executing thread to temporarily cease execution.
public class Example1 extends Thread{

    public static void main(String[] args){
        new Example1().start();
    }

    public void run() {
        System.out.println("I am an independent thread!");
        System.out.println("My Thread ID is " + this.getId());
        System.out.println("My Name is " + this.getName());
    }
}
I am an independent thread!
My Thread ID is 9
My Name is Thread-0
> _
public class Example2 extends Thread{

    public Example2() { super(); }
    public Example2(String s) { super(s); }

    public static void main(String[] args){
        new Example2().start();
        new Example2("Albert Threadington").start();
    }

    @Override
    public void run() {
        System.out.println("I am an independent thread!");
        System.out.println("My Thread ID is " + this.getId());
        System.out.println("My Name is " + this.getName());
    }
}
I am an independent thread!
I am an independent thread!
My Thread ID is 11
My Thread ID is 10
My Name is Albert Threadington
My Name is Thread-0
>_
Race Condition

A race condition occurs when where some event (in this case the output) is dependent on the sequence or timing of other uncontrollable events; the order in which threads access I/O.
void join()

Waits for this thread to terminate.

void join(long millis)

Waits at most millis milliseconds for this thread to terminate.

void join(long millis, int nanos)

Waits at most millis milliseconds plus nanos nanoseconds for this thread to terminate.

These definitions from the java api
http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html
start - join

Thread

A (main)

B (thread)

Running Thread

Blocked Thread

Time
public class Example3 extends Thread{

    public Example3() { super(); }
    public Example3(String s){ super(s); }

    public static void main(String[] args) throws InterruptedException{
        Thread t1 = new Thread(new Example3());
        Thread t2 = new Thread(new Example3("Albert Threadington"));
        t1.start();
        t1.join();
        t2.start();
    }

    @Override
    public void run() {
        System.out.println("I am an independent thread!");
        System.out.println("My Thread ID is " + this.getId());
        System.out.println("My Name is " + this.getName());
    }
}
I am an independent thread!
I am an independent thread!
My Thread ID is 9
My Thread ID is 10
My Name is Albert Threadington
My Name is Thread-0
>_
public class Example3 extends Thread{

    private volatile boolean isRunning = true;

    public static void main(String[] args) throws InterruptedException{
        ... stuff ...
    }

    public void run() {
        while(isRunning){
            ... do something ...
        }
    }

    public void flagStop(){
        isRunning = false;
    }
}
public class Example3 extends Thread{

    private volatile boolean isRunning = true;

    public static void main(String[] args) throws InterruptedException{
        ... stuff ...
    }

    @Override
    public void run() {
        while(isRunning){
            ... do something ...
        }
    }

    public void flagStop(){
        isRunning = false;
    }
}

- 'isRunning' is actually accessed by 2 (or more) threads, and may be kept locally in machine registers.
- Loading and storing of some variables are atomic.
- Their value may be in cache memory or registers.
- Volatile ensures the variable is kept in main memory.
- More elegant than sync'd getters/setters.
Synchronization locks.
Keyword: Synchronized

- object's method
- class's static method
- locking an object
  - instance level code block
  - class level code block
Keyword: Synchronized

- instance method
- static method
- instance code block
- static code block

```java
public class Example {
    public synchronized void doSomething(){
        ....
        /* code goes here */
        ....
    }
    ....
}
```
Keyword: Synchronized

- instance method
- static method
- instance code block
- static code block

```java
public class Example {
    public synchronized static void doSomething(){
        ....
        /* code goes here */
        ....
    }
}
```
Keyword: Synchronized

- instance method
- static method
- instance code block
- static code block

```java
public class Example {
    public void doSomething(Object obj){
        synchronized(obj){
            ...
        }
    }
}

public class Example {
    public void doSomething(){
        -- not synch'd --
        synchronized(this){
            ....
        }
        -- not synch'd --
    }
}
```
Keyword: Synchronized

- instance method
- static method
- instance code block
- static code block

```java
public class Example {
    public void doSomething() {
        -- not synch'd --
        synchronized (SomeClass.class) {
            ....
        }  // not synch'd
    }
}
```
wait - notify
public class ThreadA extends Thread{
    public void run(){
        synchronized(WaitNotify.class){
            try {
                System.out.println("Thread A - Before");
                WaitNotify.class.wait();
                System.out.println("Thread A - After");
            } catch (InterruptedException ex) {}
        }
    }
}

public class ThreadB extends Thread{
    public void run(){
        synchronized(WaitNotify.class){
            System.out.println("Thread B - Before");
            WaitNotify.someCondition = true;
            WaitNotify.class.notify();
            System.out.println("Thread B - After");
        }
    }
}
```java
public class ThreadA extends Thread{
    public void run(){
        synchronized(WaitNotify.class){
            try {
                System.out.println("Thread A - Before");
                while (WaitNotify.someCondition == false) WaitNotify.class.wait();
                System.out.println("Thread A - After");
            } catch (InterruptedException ex) {}  
        }
    }
}

public class ThreadB extends Thread{
    public void run(){
        synchronized(WaitNotify.class){
            System.out.println("Thread B - Before");
            WaitNotify.someCondition = true;
            WaitNotify.class.notify();
            System.out.println("Thread B - After");
        }
    }
}
```
Master-Slave Example
Final Example

/work/edward/Examples/Farmer-Worker/
Thank You,
Questions?