Operating systems and process oriented programming (1DT096)

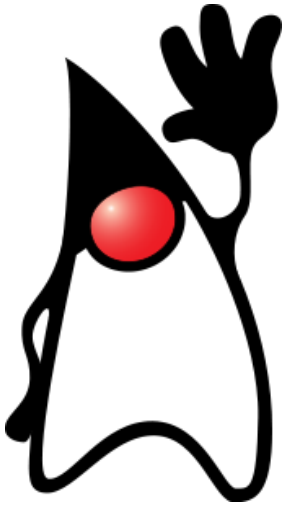
Grupp 4:

Andreas Rubensson
Carl Wingårdh
Erik Österberg
Lucas Arnström
Oskar Ahlberg
Jin Wen Ting

30-3-2015

Concurrency / Synchronization

History



History

-Java was called the Project Oak, while it was developed in the 1990’s, in the care of Sun Microsystems. Due to the wast different architectures of computers the problem was cross compatibility.

-Java was released to the general public in 1995.

History

-The goal of project oak was to develop an Virtual Machine and a language for that vm.

-The language was similar to c / c++

-Write once Run Anywhere

History

The leading design elements was:

There were five primary goals in the creation of the Java language:

1. It should use the object-oriented programming methodology.

2. It should allow the same program to be executed on multiple operating systems.

3. It should contain built-in support for using computer networks.

4. It should be designed to execute code from remote sources securely.

5. It should be easy to use by selecting what was considered the good parts of other object-oriented languages.

Concurrency

- One process (JVM)
- Threads
- At least one thread (main thread), system threads
- Share resources

Concurrency

Subclass of Thread

```
public class HelloThread extends Thread { … }

HelloThread t = new HelloThread();
t.start();
```

Implement Runnable

```
public class HelloThread implements Runnable{ … }

HelloThread t = new HelloThread();
new Thread(t).start();
```

Concurrency

- Interrupt
  - A signal sent to stop another thread
  - t.interrupt()
  - t.interrupted()
  - InterruptedException
- Join
  - Wait for another thread to terminate
  - t.join()

t : thread

Concurrency

- Executors
  - Thread managers
  - Sepeare thread management from rest of application
  - e.execute(t);
- Thread Pools
  - Improvements:
    - Minimized overhead and manage threads
  - Memory allocation streamlined
  - Pool sizes
  - Keep-alive times, queueing and queue maintenance, rejected tasks, hook methods

e : executor
t : runnable thread

Synchronization

- Synchronized
  - Execution control
  - Creates “happens-before” memory barrier
    - Possibly large overhead due to CPU-cache flushing
      - Synchronizes its variables with memory before execution. Then re-synchronizes them after execution.
  - Method declaration
  - Block

Synchronization

- Volatile
  - Memory control
  - No thread-local caching
    - No CPU caching is permitted
    - All reads and writes are done in memory
    - Possibly small overhead compared to “synchronized”
  - Only use atomic reads and writes to secure thread-safety
  - Used on references

Example

http://www.tutorialspoint.com/java/java_thread_synchronization.htm

```
class PrintDemo {
  public void printCount(){
   try {
       for(int i = 5; i > 0; i--) {
         System.out.println("Counter   ---   "  + i
);}
    } catch (Exception e) {
       System.out.println("Thread  interrupted.");}
  }
}
```

```
class ThreadDemo extends Thread {
  private Thread t;
  private String threadName;
  PrintDemo  PD;

  ThreadDemo( String name,  PrintDemo pd){
     threadName = name;
      PD = pd;}
  public void run() {
    PD.printCount();
    System.out.println("Thread " +  threadName + "
exiting.");}

  public void start ()
  {
     System.out.println("Starting " +  threadName );
     if (t == null){
        t = new Thread (this, threadName);
        t.start ();
     }
  }
}
```

```
public class TestThread {
  public static void main(String args[]) {

     PrintDemo PD = new PrintDemo();

     ThreadDemo T1 = new ThreadDemo( "Thread - 1
", PD );
     ThreadDemo T2 = new ThreadDemo( "Thread - 2
", PD );

     T1.start();
     T2.start();

     // wait for threads to end
     try {
        T1.join();
        T2.join();
     } catch( Exception e) {
        System.out.println("Interrupted");
     }
  }
}
```

Example

http://www.tutorialspoint.com/java/java_thread_synchronization.htm

This produces different result every time running this program:

```
Starting Thread - 1
Starting Thread - 2
Counter   ---   5
Counter   ---   4
Counter   ---   3
Counter   ---   5
Counter   ---   2
Counter   ---   1
Counter   ---   4
Thread Thread - 1  exiting.
Counter   ---   3
Counter   ---   2
Counter   ---   1
Thread Thread - 2  exiting.
```

Example

http://www.tutorialspoint.com/java/java_thread_synchronization.htm

```
class PrintDemo {
  public void printCount(){
   try {
       for(int i = 5; i > 0; i--) {
          System.out.println("Counter   ---   "  + i );
       }
    } catch (Exception e) {
       System.out.println("Thread  interrupted.");
    }
  }
}
```

```
class ThreadDemo extends Thread {
  private Thread t;
  private String threadName;
  PrintDemo  PD;

  ThreadDemo( String name,  PrintDemo pd){
     threadName = name;
     PD = pd;
  }
  public void run() {
    synchronized(PD) {
      PD.printCount();
    }
    System.out.println("Thread " +  threadName + "
exiting.");
  }
  public void start () {
    System.out.println("Starting " +  threadName );
    if (t == null){
      t = new Thread (this, threadName);
      t.start ();
    }
  }
}
```

```
public class TestThread {
  public static void main(String args[]) {

    PrintDemo PD = new PrintDemo();

    ThreadDemo T1 = new ThreadDemo( "Thread - 1
", PD );
    ThreadDemo T2 = new ThreadDemo( "Thread - 2
", PD );

    T1.start();
    T2.start();

    // wait for threads to end
    try {
      T1.join();
      T2.join();
    } catch( Exception e) {
      System.out.println("Interrupted");
    }
  }
}
```

Example

http://www.tutorialspoint.com/java/java_thread_synchronization.htm

```
Starting Thread - 1
Starting Thread - 2
Counter   ---   5
Counter   ---   4
Counter   ---   3
Counter   ---   2
Counter   ---   1
Thread Thread - 1  exiting.
Counter   ---   5
Counter   ---   4
Counter   ---   3
Counter   ---   2
Counter   ---   1
Thread Thread - 2  exiting.
```

Pros and Cons

- Pros
  - Good portability
  - Platform independent
  - Automatic garbage collection
  - Intelligent IDE’s
  - Java API
  - Easy to create threads

Pros and Cons

- Cons
  - Outperformed by for example c and c++
  - Large memory footprints
  - Bugs in the JVM
  - Lacks unsigned int types
  - Limited array size

Real World Examples

http://javarevisited.blogspot.se/2014/12/where-does-java-used-in-real-world.html

Android Apps

Server Apps

Java Web Apps

Software Tools

Embedded Space

Reflections

Code intensive

Good API

Powerful but slow

Interesting and useful

References

- History
  - http://en.wikipedia.org/wiki/Java_(programming_languag) 26-03-15
  - http://www.java.com/en/ 26-03-15
- Concurrency
  - http://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html 26-03-15
  - http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html 23-03-15
  - http://docs.oracle.com/javase/tutorial/essential/concurrency/interrupt.html 26-03-15
  - http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html#interrupt() 25-03-15
  - https://docs.oracle.com/javase/tutorial/essential/concurrency/join.html 25-03-15
  - http://en.wikipedia.org/wiki/Java_concurrency 23-03-15
  - http://docs.oracle.com/javase/tutorial/essential/concurrency/executors.html 26-03-15
  - http://docs.oracle.com/javase/tutorial/essential/concurrency/exinter.html 26-03-15
  - https://docs.oracle.com/javase/tutorial/essential/concurrency/pools.html 26-03-15
- Pros and cons
  - http://anynowhere.com/bb/posts.php?t=3891 26-03-15
  - http://c2.com/cgi/wiki?JavaProsAndCons 26-03-15
  - http://c2.com/cgi/wiki?JavaDiscussion 26-03-15
  - http://c2.com/cgi/wiki?WhyJavaIsGreat 26-03-15
  - http://zeroturnaround.com/rebellabs/flavors-of-concurrency-in-java-threads-executors-forkjoin-and-actors/ 26-03-15
  - http://java.dzone.com/articles/10-reasons-why-java-rocks-more 26-03-15
  - http://en.wikipedia.org/wiki/Criticism_of_Java 26-03-16