Creación de Threads

Lecturas recomendadas:
Tutorial Java (Concurrencia):

http://docs.oracle.com/javase/tutorial/essential/concurrency/

César Sánchez

Universidad Politécnica de Madrid

Este texto se distribuye bajo los términos de la Creative Commons License
Mapa Conceptual

Concurrency = Simultaneous + Nondeterminism + Interaction

Interaction = Communication | Synchronization

Synchronization = Mutual Exclusion | Conditional Synchronization

Terminology:
- atomic
- interleaving
- mutual exclusion
- deadlock
- liveness
Short Homework 1

Homework:
- Create threads
- Basic Interaction (wait for threads to finish)
- (next week): share data
- (the week after): mutual exclusion

Fecha de cierre:
Viernes 13-Feb-2015, 11am

Entrega online:
http://lml.ls.fi.upm.es/~entrega
Today’s topics

1. Create threads
2. Sleep
3. Join
Process vs Thread

Process

- Protected resources (including memory)
- Context switch slow
- Support from OS
- High overhead

Thread

- Shared resources (including memory)
- Context switch fast
- Maybe not support from OS
- Low overhead
- Lightweight process
Process vs Thread

Process

- Protected resources (including memory)
- Context switch slow
- Support from OS
- High overhead
- Heavy process

Thread
Process vs Thread

Process
- Protected resources (including memory)
- Context switch slow
- Support from OS
- High overhead
- *Heavy* process

Thread
- Shared resources (including memory)
- Context switch fast
- Maybe not support from OS
- Low overhead
- *Lightweight* process
Thread Creation

Three steps:
1. Create a Thread object (assigns memory)
2. Give the object “life”
3. Run the object
Create a Thread Object

There are two ways to create a new thread:

1. Extend Thread:

   ```java
   class T extends Thread { ... }
   
   And instantiate A (each instance \(\rightarrow\) one thread)
   
   T t = new A()
   ```

2. Implement Runnable:

   ```java
   class R implements Runnable { ... }
   
   then instantiate the runnable and create the thread:
   
   R myrunner = new R();
   Thread t = new Thread(myrunner);
   ```
Create a Thread Object

The starting point of the thread will be the method `run()`:

```java
class T extends Thread {
    public void run() {
        // your code goes here!
    }
}
```

or

```java
class R implements Runnable {
    public void run() {
        // your code goes here!
    }
}
```
Start the Thread

The father thread invokes `start()`. The child starts with `run()` invoked.

```
R myrunner = new R();
Thread t = new Thread(myrunner);
// here we go...
t.start(); // <-- which invokes run()
// the father continues
```
Example 1. Thread Creation and Start

Implementing Runnable

```java
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();
    }
}
```
Example 2. Thread Creation and Start

Extending Thread

```java
public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new HelloThread()).start();
    }
}
```
Sleep puts a thread to sleep for *at least* the delay.

```java
public class SleepMessages {
    public static void main(String[] args) throws InterruptedException {
        String info[] = {"A", "B", "C", "D"};

        for (int i = 0; i < importantInfo.length; i++) {
            Thread.sleep(4000); //Pause for 4 secs
            System.out.println(info[i]); //Print a message
        }
    }
}
```
Join blocks thread to finish execution.

The thread that executes:

\[ t.join(); \]

Waits for \( t \) to finish.