ISCTE IUL
Instituto Universitário de Lisboa

Java Threads

Programação Concorrente e Distribuída

Parallel and Distributed Programming

2015-2016

After this class you will be able to…

- Understand what is a Thread.
- How to start several threads.
- The main methods of the Thread class.
- Interruption of threads.
- The life cycle of a thread.

What are threads?

- The literal translation: *Linha*

- Stands for “Thread of Control”, a section of the code that runs independently from other sequential sections in the same program

Why multiple threads?

- If, in a program, we can use several threads:
  - for long calculations,
  - for text input
  - for spell checking

  the program can execute all 3 tasks concurrently so we can write the text while the calculation is being done and text is being corrected.

- It seems that all is happening at the same time.

How are threads executed?

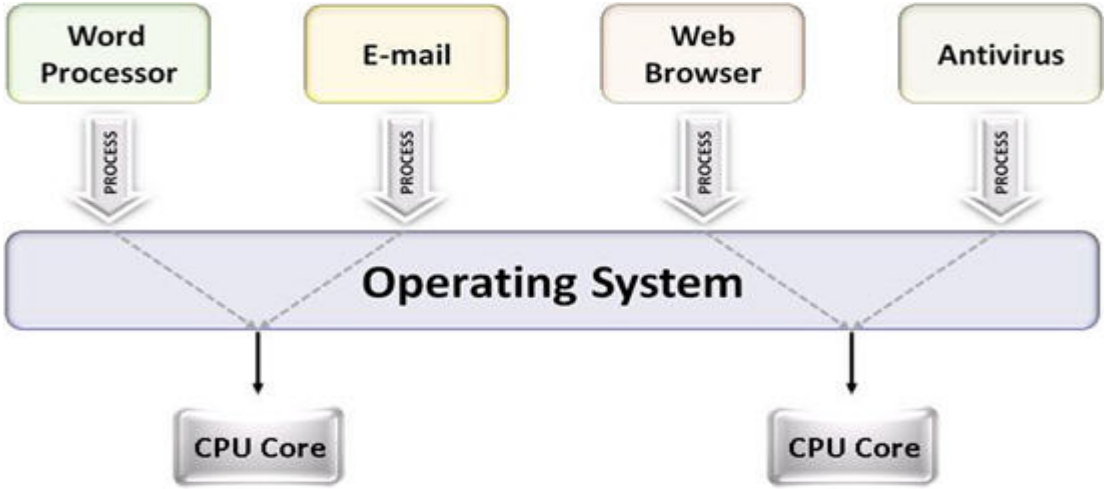
Execution of the different threads can occur in three scenarios:

- They execute in parallel, if there is more than one processor
- They can run interleaved on one processor
- They can run in a mixed parallel/ interleaved solution

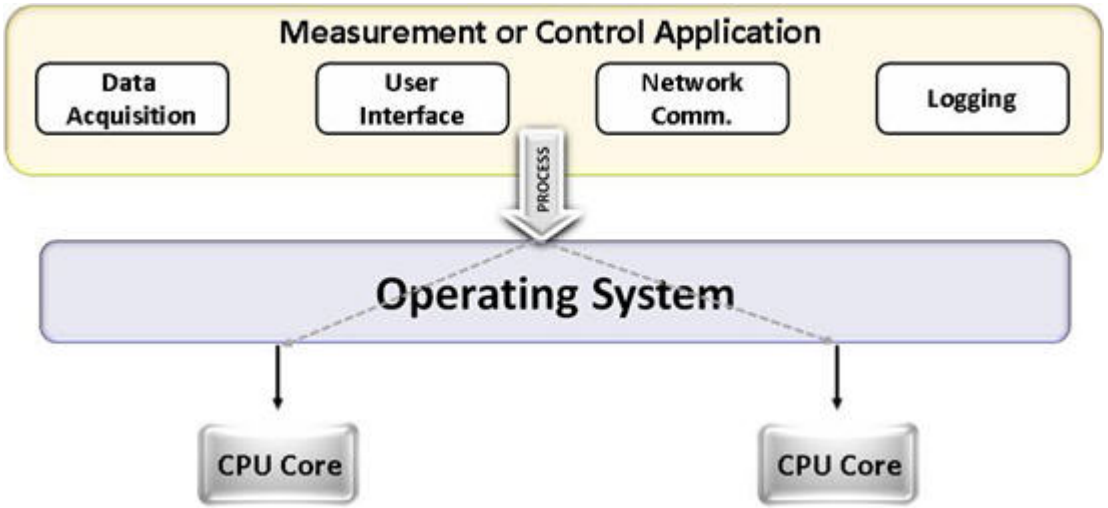
Multitasking vs. Multithreading

- **Multitasking**: A process (ex. an application) runs in a *separate* address space. Multiple processes can be running at the same time, but one process cannot manipulate the memory allocated to another process.
- **Multithreading**: A process can create multiple threads that all execute in the *same* address space. This means that all threads potentially have access to the same instances of objects, file descriptors, network connection and so on.

Multitasking



Multithreading



Important differences between processes and threads

- Creating a thread requires less resources than creating a process.
- A process usually requires the operation system to create and maintain a comprehensive low-level data structure including a virtual memory map, file descriptors, user identification, etc.
- The threads of the same process all share the data structure for the process and are therefore more high-level entities. They require little more than a their own stack.
- Threads exist inside a process — all processes have at least one thread. All threads share the resources of the process, including memory, file descriptors and so on, which makes communication between threads more efficient – but potentially problematic.

Multihreading in JAVA

1. Every thread starts execution in a specific location:

   i. For the first thread, this location is the public static method main()

   ii. Subsequent threads start in a location decided by the developer.

2. Every thread executes independently of the other threads in the program, however, there exist various mechanisms that allow threads to cooperate.

What can threads be used for?

- Graphical interfaces
- More performance
  - The processor(s) are less idle
    - When one thread is waiting, for instance reading data from a hard drive, another thread can compute.
  - Different tasks, for instance two animations running at the same time.
  - Allows for multiple processors/cores to be utilized at the same time.

Threads in JAVA

```
public class OMeuThread extends Thread {

    public void run() {
        System.out.println("o meu thread");
    }

    public static void main(String [] argv) {
        Thread t = new OMeuThread();
        t.start();
    }
}
```

The second thread is explicitly started by the programmer and execution starts in the run method

Second thread terminates when leaving the run method

The first thread ends when leaving the main method

Class Thread

- Every thread is associated to an instance of the class Thread
- An application that creates one or more threads, provides the code for those threads.
- ... there are two ways of specifying the code
  - like on the previous slide, by extending the class **Thread**
  - by implementing the **Runnable** interface

Using the Runnable interface

```
public class OMeuThread implements Runnable{

    public void run() {
        System.out.println("o meu thread");
    }

    public static void main(String [] argv) {
        Runnable omt = new OMeuThread();
        Thread t = new Thread(omt);
        t.start();
    }
}
```

Thread.sleep;

- "Sleep" is a static method that pauses the execution of the calling thread for a specific period of time
- This leaves more time for other threads and processes to execute
- Sleep times can be specified in milliseconds and in nanoseconds, but there is no guarantee that the thread is paused for exactly the time specified. It depends on the OS, on the hardware and on the resources currently available.
- sleep can throw an InterruptedException when another thread interrupts a sleeping thread

Thread.sleep;

What is the result of the following program?

Thread.sleep, Thread.currentThread and the interrupt method

```
// 1/10/2007 - Adaptado do JAVA Tutorial por Nuno David
public class SleepMessages extends Thread {
    public void run() {
        String importantInfo[] = { "Mares eat oats", "Does eat oats",
                "Little lambs eat ivy", "A kid will eat ivy too" };

        for (int i = 0; i < importantInfo.length; i++) {
            try {
                // Pause for 4 seconds
                System.out.println(currentThread() + ": sleep for 4 seconds");
                sleep(4000);
                // Print a message
                System.out.println("\t" + importantInfo[i]);
            } catch (InterruptedException e) {
                System.out.println(currentThread() + ": Ops! I was interrupted!");
            }
        }
    }
    public static void main(String args[]) throws InterruptedException {
        Thread t = new SleepMessages();
        t.start();
        int pausa = (new Random()).nextInt(16000);
        System.out.println(currentThread() + ": sleep for " + pausa / 1000 + " seconds");
        sleep(pausa);
        t.interrupt();
    }
}
```

Interruption of threads - example of console output

```
Thread[main,5,main]: sleep for 5 seconds
Thread[Thread-0,5,main]: sleep for 4 seconds
    Mares eat oats
Thread[Thread-0,5,main]: sleep for 4 seconds
Thread[Thread-0,5,main]: Ops! I was
interrupted!
Thread[Thread-0,5,main]: sleep for 4 seconds
    Little lambs eat ivy
Thread[Thread-0,5,main]: sleep for 4 seconds
    A kid will eat ivy too
Exit code: 0
No Errors
```

Interruption of threads

- **An interruption does not stop a thread or pause it!**
- When a thread is interrupted, a flag is set to indicate that the thread is now in an iterrupted state
- An interruption indicates that something happened and that the thread may need to do something different
- It is the programmer who decides how a thread should respond to the interruption. This includes deciding if a thread should terminate when interrupted or not
- If a thread spends a lot of time without calling a method that throws an interrupted exception, it can check if it was interrupted in the following way:

```
if (Thread.interrupted()){
        // DO SOMETHING
        throw new InterruptedException();
}
```

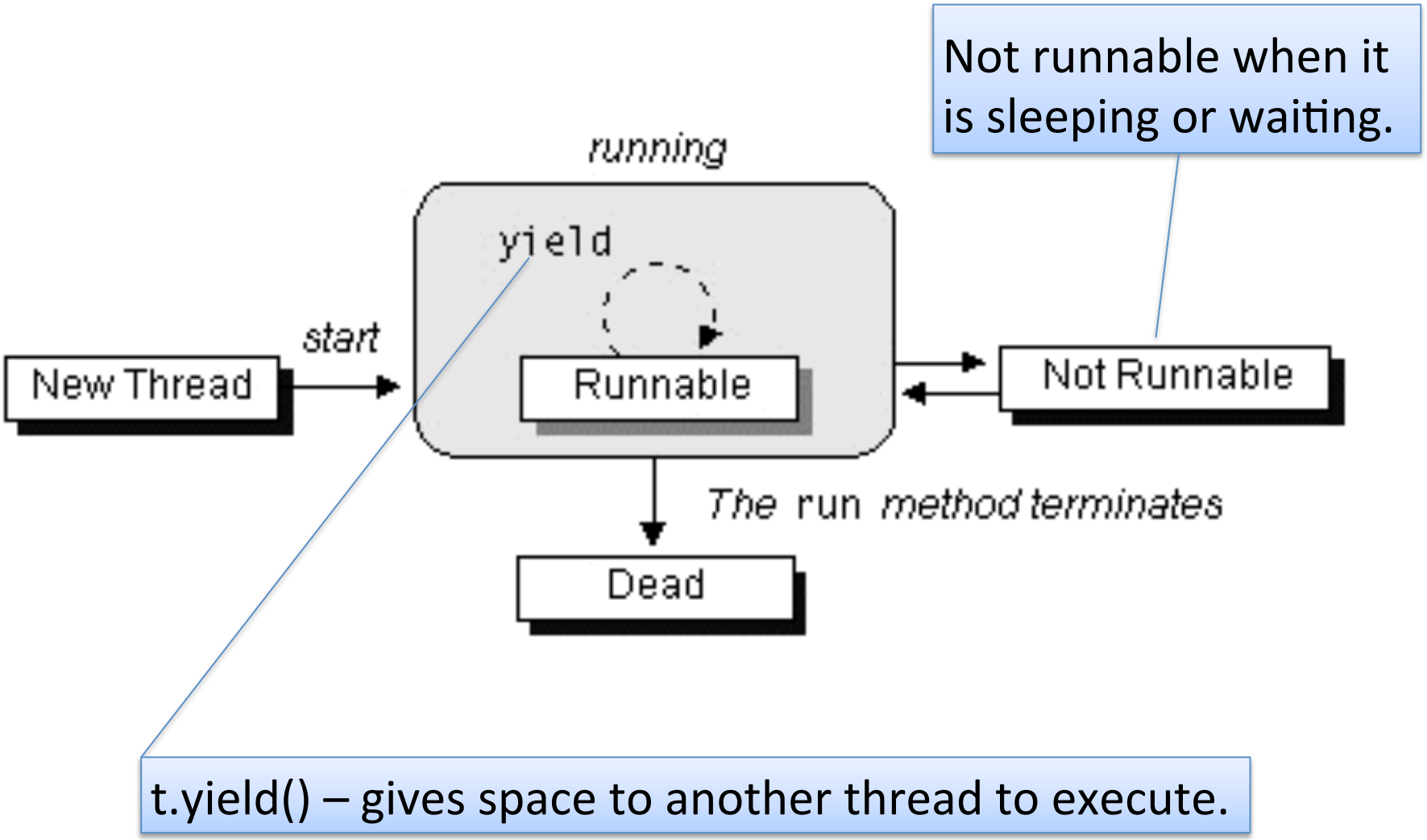
Calling interrupted() clears the interrupted flag!

Other methods on the Thread class

- `t.stop()` – terminates a thread and frees the monitors that it has control over; AVOID USING THIS METHOD. Threads should terminate in a controlled way in order to avoid potential memory and logical inconsistencies; stop() was deprecated in JDK 1.2
- `t.suspend()/t.resume()` – suspend/resume a thread t; Avoid using this as it can create *deadlocks;* these methods were deprecated in JDK 1.2
- `t.isAlive()` – checks if a thread is active. Can only be used to verify that a thread has ended.
- `t.join()` – allows for one thread to wait for another thread to end. Like with the sleep() method, an InterruptedException can be thrown

For more on the problems of stop, suspend and resume see:
"Why are Thread.stop, Thread.suspend and Thread.resume Deprecated?"

Life cycle of a Thread



To study

- Study the class Thread in the Java API
- Study the Runnable interface in the Java API
- Study the recommended bibliography
- Practical exercises

Bibliography

- JAVA Tutorial

  http://download.oracle.com/javase/tutorial/essential/concurrency/index.html

- JAVA Threads, O'Reilly, chapters 1-2
- Multithreaded, Parallel, and Distributed Programming, G.R. Andrews, chapter 1-2
- Multithreaded Programming, Lewis & Berg, chapters 1-4

Summary

Threads in JAVA

- The Thread concept – multitasking vs. multithreading
- The Thread class
- Starting threads – the Runnable interface
- Interruptions
- Other methods on the class Thread
- A threads life-cycle
- Exercises