Concurrency:
Multi-core Programming
& Data Processing

Lab 1

-- Introduction and Java basics --

Administrative stuff

- Lab sessions: ~2 hours, homework check, hands-on exercises (bonuses?)
- 5 homework, 2 weeks for each, (hard) deadline 9 am lecture day
- 1st homework: next week
- ILIAS: homework submission, lab pdf, exercises
- Questions/comments:
  - Forum on ILIAS
  - E-mail: maria.carpen-amarie@unine.ch

Work environment

- Java – OpenJDK8
- IDE: Eclipse
- Testing: T2000 SunFire server (Sun Solaris 10 OS)
- Login:
  - Linux: SSH: `ssh iliaslogin@sunfire.unineuchatel.ch`
  - Windows: putty
- Using Java in command line:
  - Compile: `javac Program.java`
  - Execute: `java Program`

Threads in Java

- Processes vs. Threads (a sequential series of instructions executing inside a process)
- At least one thread: main thread
- Version 1: `java.lang.Thread`, overwrite `run()` method, call `start()` method
- E.g.:

```
public class MyThread extends Thread {
    public void run (){
        // do thread stuff
    }
}

MyThread newThread = new MyThread();
newThread.start();
```

Threads in Java

- Version 1.2: create task for thread, give task to thread
- Runnable interface

```
public class MyRunnableImplementation implements Runnable {
    @Override
    public void run() {
        // do task
    }
}
MyRunnableImplementation r = new MyRunnableImplementation();
Thread newThread = new Thread(r);
newThread.start();
```

Threads in Java

- Pause thread:
  - `TimeUnit.SECONDS.sleep(secs)`
  - OR
  - `Thread.sleep(millisecs)`
- Simulates long running tasks

Thread Pools

- Version 2: `java.util.concurrent,` implemented as `ExecutorService` tasks
- Tasks are distributed between the threads in the pool
- Threads can be reused
- E.g.:

```
MyRunnableImplementation r = new MyRunnableImplementation();

ExecutorService executorService = Executors.newFixedThreadPool(10);
executorService.execute(r);

executorService.shutdown();
```

Interacting with a thread

- Setting specific properties (e.g., name)
  - Inside the caller thread
  - `isAlive()` property: check if thread still running
  - `yield()`: temporary interrupt thread
  - `join()`: wait for another thread to finish
- Stopping a thread:
  - ~~`Thread.stop()`~~
  - Have a flag, check it periodically in the `run()` method: if set, continue running, otherwise stop thread execution

Basic Java synchronization

- "synchronized" keyword:
  - Methods

```
public synchronized void mySyncMethod {
    //do stuff
}
```

  - Explicitly specifying the object and the scope of the lock

```
SomeObject lockObject = new SomeObject(); //it doesn't matter what type of class we are using
public void myPartiallySyncMethod {
    synchronized(lockObject) {
        //do stuff
    }
}
```

Basic Java synchronization

- **Equivalent:**

```
synchronized static void foo() {}
```

  - … with …

```
static void foo() {
   synchronized(SomeClass.class) {}
}
```

  - **AND**

```
synchronized void foo() {}
```

  - … with …

```
void foo() {
   synchronized(this) {}
}
```

Visibility

- There is no guarantee that a reading thread will see a value written by another thread on a timely basis (or at all)
- => always use proper synchronization whenever data is shared between multiple threads
- Otherwise, stale data, infinite loops, etc.

- Volatile keyword forces read and write operations to be done in the main memory

Exercises

1. Connect to the SunFire server with the provided username and password (same as username)
   - Change password with `passwd` command
2. Download exercises/ folder for this lab from ILIAS
3. Complete, compile and run HelloWorldThreads.java
4. Complete, compile and run HelloWorldExecutor.java
5. Copy Main.java on SunFire. Compile and run.