Segurança Informática

2022/2023

Introdução

1

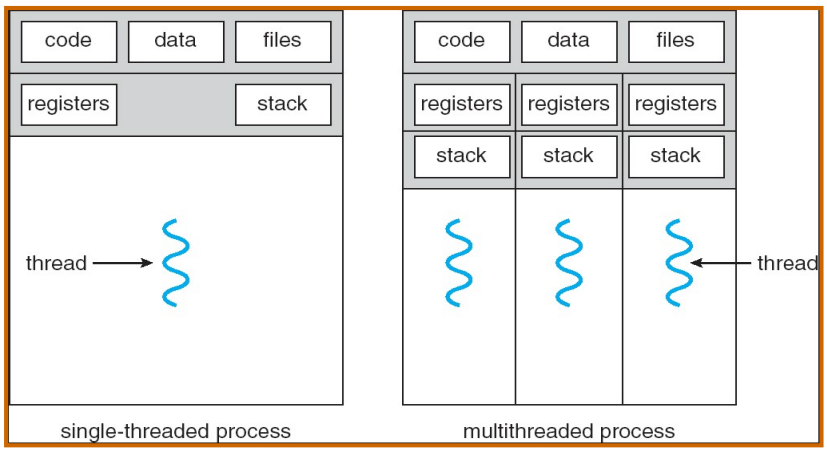
Sumário

- Tópicos úteis para a realização do projeto
  - *Threads*
  - *Sockets*
  - *Streams*
  - *Ficheiros*

3

Thread (fio de execução): conceito

- Threads vs Processos



Threads: concretização em Java

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

- Criação de Threads
  - Criar uma subclasse da classe `Thread` ou
  - Implementar a Interface `Runnable`
- Em ambos os casos:
  - Implementar o método `run`
- Iniciar a execução de uma Thread
  - O método `start` cria os recursos do sistema necessários à execução da Thread (por exemplo, memória), escalona a Thread e invoca o método run.
  - O método `run` nunca é invocado directamente pelo programador
- Terminar a execução de threads
  - O método `stop` da class Thread está "deprecated".
  - A thread deve implementar uma forma segura de terminar (existem exemplos em **http://download.oracle.com/javase/tutorial/essential/concurrency/threads.html**)
- Sincronização de Thread – Cuidado !!
  - ver **http://download.oracle.com/javase/tutorial/essential/concurrency/threads.html**

Threads: exemplos

```
class ThreadsExample extends Thread {
        private String msg = null;

        ThreadsExample(String m) {
           msg = m;
        }

        public void run(){
           for (int i=0; i < msg.length(); i++) {
              System.out.println(msg.charAt(i));
              try {
                  sleep((int)(Math.random()*100));
              } catch (InterruptedException e) {
                  System.err.println(e);
              }
           }
           System.out.println();
        }
}

public class CallThreadsExample {
   public static void main(String[] args) {
        for (int i=0; i < args.length; i++){
           ThreadsExample newThread =
                        new ThreadsExample(args[i]);
           newThread.start();
           System.out.println("fim no main");
        }
   }
}
```

```
class ThreadsExample implements Runnable {
        private String msg = null;

        ThreadsExample(String m) {
           msg = m;
        }

        public void run(){
           for (int i=0; i < msg.length(); i++) {
              System.out.println(msg.charAt(i));
              try {
                  Thread.sleep((int)(Math.random()*100));
              } catch (InterruptedException e) {
                  System.err.println(e);
              }
           }
           System.out.println();
        }
}

public class CallThreadsExample {
   public static void main(String[] args) {
        for (int i=0; i < args.length; i++){
           ThreadsExample newThread =
                 new ThreadsExample(args[i]);
           new Thread(newThread).start();
           System.out.println("fim no main");
        }
   }
}
```


Threads: classes *Timer* e *TimerTask*

- Úteis para a concretização de tarefas periódicas ou para escalonar tarefas futuras de forma mais simples que usando threads
- A classe `java.util.TimerTask` representa uma tarefa
  - Implementa `Runnable` e tem um método abstrato `run`, logo é muito parecida com a `Thread`.
  - Existe um método `cancel()` que serve para cancelar a execução da tarefa, se escalonada.
- A classe `java.util.Timer` permite o escalonamento de tarefas periódicas ou não através dos métodos:
  - `schedule(TimerTask task, Date time)`: define que *task* deve ser executada uma única vez em *time*.
  - `schedule(TimerTask task, long delay, long period)`: define que *task* deve ser executada após *delay* ms e repetida a cada period ms **após seu término**.
  - `scheduleAtFixedRate(TimerTask task, long delay, long period)`: define que *task* deve ser iniciada a cada *period* ms **após *delay* ms**.

Sockets (TCP)

http://docs.oracle.com/javase/tutorial/networking/

- Definição:
  - *A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program.*

- Operações a concretizar num cliente
  - Open a socket.
    ```
    Socket echoSocket = new Socket("taranis.di.fc.ul.pt", 7); //"127.0.0.1"
    ```
  - Open an (object) input stream and (object) output stream to the socket.
    ```
    ObjectInputStream in = new ObjectInputStream(echoSocket.getInputStream());
    ObjectOutputStream out = new ObjectOutputStream(echoSocket.getOutputStream());
    ```
  - Write to and read from the stream according to the protocol.
    ```
    out.writeObject(userInput);
    String fromServer = (String) in.readObject();
    ```
  - Close the streams.
    ```
    out.close();
    in.close();
    ```
    A ordem é relevante
  - Close the socket.
    ```
    echoSocket.close();
    ```


Sockets (TCP)

- Operações a concretizar num servidor
  - Create a socket to listen on a specific port
    ```
    Server serverSocket = new ServerSocket(4444);
    ```
  - Accepting a connection from a client
    ```
    Socket clientSocket = serverSocket.accept();
    ```
  - Open an input stream and output stream to the socket.
    ```
    ObjectInputStream in = new ObjectInputStream(clientSocket.getInputStream());
    ObjectOutputStream out = new ObjectOutputStream(clientSocket.getOutputStream());
    ```
  - Read from and write to the stream according to the protocol.
    ```
    String fromClient = (String) in.readObject();
    out.writeObject(answer);
    ```
  - Close the streams.
    ```
    out.close();
    in.close();
    ```
  - Close all sockets.
    ```
    clientSocket.close();
    serverSocket.close();
    ```

Para atender vários clientes:
while (true) {
  accept a connection ;
  create a thread to deal with the client ;
  (ou usa uma **threadpool**)
}

Streams

http://docs.oracle.com/javase/tutorial/essential/io/bytestreams.html

- ByteStreams
  - programs use *byte streams* to perform input and output of 8-bit bytes
  - byte stream classes are descended from a particular InputStream and OutputStream
  - Exemplo: file I/O byte streams, FileInputStream and FileOutputStream
- CharacterStreams
  - automatically translates the Unicode internal format to and from the local character set
  - all character stream classes are descended from a particular Reader and Writer
  - Exemplo: file I/O: FileReader and FileWriter
- BufferedStreams
  - Vantagens – ver **http://docs.oracle.com/javase/tutorial/essential/io/buffers.html**
  - Classes to wrap unbuffered streams:
    Buffered **byte** streams: BufferedInputStream and BufferedOutputStream
    Buffered **character** streams: BufferedReader and BufferedWriter
  - to force the flushing of an output stream manually, invoke its flush method

Streams – cont.

- DataInputStream, DataOutputStream
  - Transferência apenas de dados primitivos (boolean, char, short …) num formato independente do hardware (cp. "formato da rede" nas aulas de sistemas distribuídos)

```
DataOutputStream out = new DataOutputStream(clSocket.getOutputStream());
out.writeInt(17);
out.writeFloat(3.1415);
out.writeByte('x');

DataInputStream in = new DataInputStream(clSocket.getInputStream());
int x = in.readInt();
float f = in.readFloat();
byte b = in.readByte();
```

- ObjectInputStream, ObjectOutputStream
  - Transferência de dados complexos (Objects) num formato complexo (inclui tipo do objecto e todos os seus atributos); os dados primitivos também são bem tratados

```
ObjectOutputStream out = new ObjectOutputStream(clientSocket.getOutputStream());
String userInput = ...;
out.writeObject(userInput);

ObjectInputStream in = new ObjectInputStream(clientSocket.getInputStream());
String fromClient = (String) in.readObject());
```


Ficheiros

- Provavelmente já conhecem de disciplinas anteriores… as classes fundamentais:
- `File`: representam um nome dentro do sistema de ficheiros, servindo portanto para representar ficheiros, diretorias, links, etc.
  - Método `list()` permite obter a lista de ficheiros numa diretoria
  - Método `lastModified()` permite saber quando o ficheiro foi modificado pela última vez
- `FileInputStream`: input stream (binário) básico para leitura de dados em ficheiros. Assim como nos sockets, pode ser composto com outros (e.g., ObjectInputStream para ler objetos serializados em ficheiros).
- `FileOutputStream`: output stream (binário) básico para escrita de dados em ficheiros. Também pode ser composto.