Threads in Java

Concurrent and parallel programming

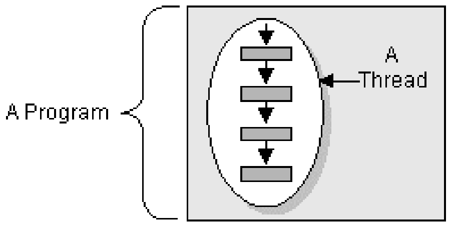
Programowanie współbieżne i równoległe

Academic year: 2018/19, Lecture 2

Paweł Lula, Cracow University of Economics, Poland
pawel.lula@uek.krakow.pl

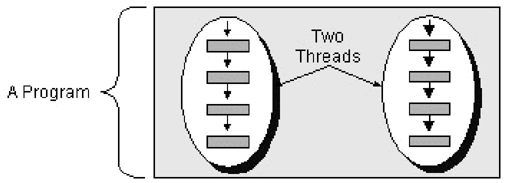
Single thread program in Java

- **Thread** – a part of a computer program which can be executed simultaneously.
- Threads share computer's resources (memory, open files).
- **Single-thread program** – a computer program in which only one thread (*main thread*) exists. In Java a main thread is defined as a method called *main*.



Multithread program

- **Multithread program** – a program with more than one thread.
- Every thread can create additional threads.



Threads representation in Java

- Every thread in Java program is represented by the object defined by the `Thread` class.
- Main methods defined in the `Thread` class:
    - `public Thread()` – class constructor
    - `public Thread(Runnable target)` – class constructor allowing to pass another object containing the definition of actions which should be performed by a given thread
    - `public Thread(String name)` – class constructor allowing to define a thread's name
    - `public void run()` – the method which defines operations which should be performed by a given thread
    - `public void start()` – the method which calls a `run()` method

Definition of the exemplary thread

```
class Star extends Thread {
  public void run() {
        for (int i = 0; i < 10;i++) {
                System.out.print("*");
                try {
                        sleep(1000);
                }
                catch(InterruptedException e) {
                }
        }
  }
}
```

.....

```
class Plus extends Thread {
  public void run() {
        for (int i = 0; i < 10;i++) {
                System.out.print("+");
                try {
                        sleep(1000);
                }
                catch(InterruptedException e) {
                }
        }
  }
}
```

…..

```
public class ThreadsCreation1 {
   public static void main(String [] args) {
            Star g = new Star();
            Plus p = new Plus();
            g.start();
            p.start();
            System.out.print(“FINISH");
   }
}
```

Program outcomes:

```
FINISH*++**++*+*+*+*+*+**+
```

The definition of the ***sleep*** method (class: Thread):

public static void sleep(long millis) throws InterruptedException

7

Runnable interface as a tool for thread’s creation

- The code executed in the thread should be defined in the class implementing ***Runnable*** interface
- The interface ***Runnable*** defines a single method:

  public void run()

  this method defines the code which should be executed in the thread.
- The use of ***Runnable*** interface is a more general solution than the use of the inheritance mechanism. It allows to defined thread’s code in classes that are not descendants of the ***Thread*** class.
- The object of the class implementing ***Runnable*** interface should be passes to the ***Thread*** object as a constructor’s parameter.
- The ***start()*** method is used for starting the code from the ***run()*** method.

8

Example

```
class Star implements Runnable {
  public void run() {
        for (int i = 0; i < 10;i++) {
                System.out.print("*");
                try {
                        Thread.sleep(1000);
                }
                catch(InterruptedException e) {
                }
        }
  }
}
```

.....

```
class Plus implements Runnable {
  public void run() {
        for (int i = 0; i < 10;i++) {
                System.out.print("+");
                try {
                        Thread.sleep(1000);
                }
                catch(InterruptedException e) {
                }
        }
  }
}
```

…..

```
public class ThreadsCreation2 {
  public static void main(String [] args)
  {
        Thread g = new Thread(new Star());
        Thread p = new Thread(new Plus());
        g.start();
        p.start();
        System.out.print(“FINISH");
  }
}
```

Program outcomes:

```
FINISH*+*+*+*+*+*+*+*+*+*+*+*+
```

Thread interruption

t ← object of the ***Thread*** class

```
t.interrupt();
//    sending a request to a thread,
//    that it should stop its execution
//    a thread can ignore this request!!!
```

Serving an interruption request

- Every thread has its *interrupt status flag*
- public static boolean interrupted()
  - static method
  - if *interrupt status flag* is set to TRUE, then it is changed to FALSE
- public boolean isInterrupted()
  - non-static method
  - does not change the value of the interrupt status flag

Checking for interruption

```
for (int i = 0; i < inputs.length; i++) {
      ...
      if (Thread.interrupted()) {
            // We've been interrupted
            return;
      }
}
```

Checking for interruption

- ***sleep*** method support *InterruptedException*

```
for (int i = 0; i < importantInfo.length; i++) {
      // Pause for 4 seconds
      try {
            Thread.sleep(4000);
      }
      catch (InterruptedException e) {
            // We've been interrupted
            return;
      }
      // Print a message
      System.out.println(importantInfo[i]);
}
```

Creation of own method supporting InterruptedException

```
method_header(......) throws InterruptedException {
      ...
      if (Thread.interrupted()) {
            throw new InterruptedException();
      }
      ...
}
```

Waiting for the completion of another thread

- t ← an object of Thread method

```
t.join();
// pausing the current thread
// until the completion of the t's thread
```

The extended version of join method

The extended version:

join (long millisecond)

- Example:

  in *t1* thread the main method is used:

  t2.join(10000)

- The t1 thread is paused until the earlier event appears:
  - the completion of t2 thread,
  - the specified period of time finished.

Thank you for your attention!

Paweł Lula, Cracow University of Economics, Poland
pawel.lula@uek.krakow.pl