ENSAM RABAT
جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat
المدرسة الوطنية العليا للفنون والمهن بالرباط
École Nationale Supérieure d'Arts et Métiers de Rabat

Threads in Java

1- What are Threads?

Threads are a fundamental concept in **computer science** and are widely used in various applications to achieve concurrency, improve responsiveness, and utilize resources efficiently.

2- Applications of Threads:

- **Multithreaded Servers:** where threads handle multiple client requests simultaneously, allowing the server to serve multiple clients concurrently.
- **Parallel Processing:** to execute tasks concurrently, enhancing performance in tasks such as scientific simulations and data processing.
- **Background Tasks:** Threads handle background tasks like periodic maintenance, logging, and monitoring, ensuring they run independently of the main program flow.
- **Thread Pools:** Thread pools are employed to manage reusable threads efficiently, enhancing performance by minimizing thread creation and destruction overhead.

3- Types of Threads:

User-level Threads (ULTs): Managed entirely by the application and not visible to the operating system kernel. They are lightweight but can't benefit from multiple processor cores simultaneously.
Kernel-level Threads (KLTs): Managed by the operating system kernel. They are more heavyweight but can take advantage of multiple processor cores.

4- Threads in Java:

➢ How to **create** a Thread in Java?

In Java, you can create threads using either of two methods:
Subclassing the **Thread** class and overriding its run() method:

Main.java | MyThread.java | Run_interface.java

```java
package Threads_Test;

public class MyThread extends Thread {

    public MyThread() {
    }
    @Override
    public void run(){
        System.out.println("Hello, world!");
    }
}

```

or **Implementing** the **Runnable** interface and implements its run() method:

Main.java | MyThread.java | Run_interface.java

```java
package Threads_Test;

public class Run_interface  implements Runnable {

    public Run_interface() {
    }
    @Override
    public void run(){
        System.out.println("Hello, world!");
    }
}

```

When extending the Thread class, your class cannot extend any other class due to **Java's single inheritance limitation**. Conversely, implementing the Runnable interface allows your class to extend other classes while still enabling multithreading.

- **Running Threads in Java:**

You can run a thread either by creating an instance of **Run_interface** class and pass it to Thread constructor, or by creating an instance of **MyThread** class then start the thread by using the method **start()**.

```
Main.java ×   MyThread.java   Run_interface.java
1  package Threads_Test;
2  public class Main {
3      public static void main(String[] args) {
4
5          Thread t0 = new Thread(new Run_interface());
6  // Or   MyThread t0 = new Mythread();
7          t0.start();
8      }
9  }
10
```

```
Problems   Javadoc   Declaration   Console ×
<terminated> Main [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (24 mars 2024,
Hello, world!
```

- Java Thread Common Methods:

- **sleep(long** millis**)**: Pauses the execution of the current thread for the specified amount of time.
- **currentThread()**: Returns a reference to the currently executing thread object.
- **join()**: Waits for the thread on which it's called to die.
- **getPriority()**: Returns the priority of the thread.
- **setPriority(int** priority**)**: Changes the priority of the thread.
- **getName()**: Returns the name of the thread.

- **setName(String** name**)**: Changes the name of the thread.
- **getId()**: Returns the id of the thread.
- **toString()**: Returns a string representation of this thread, including the thread's name, priority, and thread group.
- **isAlive()**: Tests if the thread is alive.
- **getState()**: Returns the state of the thread.
- **suspend()**: Suspends the thread.
- **interrupt()**: Interrupts the thread.
- **isInterrupted()**: Tests whether the thread has been interrupted.
- **resume()**: Resumes the suspended thread.
- **stop()**: Stops the thread (deprecated and not recommended for general use).

Example:

Main.java × MyThread.java Run_interface.java

```java
package Threads_Test;
public class Main {
    @SuppressWarnings("static-access")
    public static void main(String[] args) throws Exception {

        Thread t0 = new Thread(new Run_interface());
        Thread t1 = new Thread(new Run_interface());

        t0.setName("Thread num_0");
        t1.setName("Thread num_1");

        System.out.println(t0.toString());
        System.out.println(t1.toString());

        System.out.println(t0.getName() +" : "+ t0.getState());
        System.out.println(t1.getName() +" : "+ t1.getState());

        t0.setPriority(Thread.MAX_PRIORITY);
        t1.setPriority(Thread.MIN_PRIORITY);

        t0.start();
        t1.start();

        System.out.println(t0.getName() +" : "+ t0.getState());
        System.out.println(t1.getName() +" : "+ t1.getState());

        t0.sleep(1000);
        t1.sleep(1000);

        System.out.println(t0.getName() +" : "+ t0.getState());
        System.out.println(t1.getName() +" : "+ t1.getState());

        t0.join();
        t1.join();

        System.out.println(t0.getName() +" : "+ t0.getState());
        System.out.println(t1.getName() +" : "+ t1.getState());
    }
}
```

Main.java × MyThread.java Run_interface.java ×

```java
package Threads_Test;
public class Run_interface  implements Runnable {
    public Run_interface() {
    }
    @Override
    public void run(){
        for (int i = 0; i < 3; i++) {
            try {
                System.out.print(Thread.currentThread().getName()+ " --> " + i + " \n");
                Thread.sleep(2000);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

Output:

```
Console ×
<terminated> Main [Java Application] C:\Program Files\Ja
Thread[Thread num_0,5,main]
Thread[Thread num_1,5,main]
Thread num_0 : NEW
Thread num_1 : NEW
Thread num_0 : RUNNABLE
Thread num_1 : RUNNABLE
Thread num_0 --> 0
Thread num_1 --> 0
Thread num_0 --> 1
Thread num_1 --> 1
Thread num_0 : TIMED_WAITING
Thread num_1 : TIMED_WAITING
Thread num_0 --> 2
Thread num_1 --> 2
Thread num_0 : TERMINATED
Thread num_1 : TERMINATED
```