JAVA @17

Java Thread – Part II

Objective

After completing this session you will be able to understand,

- How to create thread by implementing runnable interface?
- Comparison between the two method of thread creation
- Learn about thread join method

Runnable Interface

- Threads can be created by implementing the *Runnable* interface and overriding the run method.
- *Runnable* interface contains only one method – the *run()* method which should be overridden to start a new Thread.

Implementing Runnable Interface

Step 1: Create a class ***ThreadRunnableEx*** by implementing the *Runnable* interface

Step 2: Override the ***run()*** method , the logic each thread should execute.

Step 3: Create an ***main*** method which can start the thread.

Step 4: Inside the main method create an object of the ***ThreadRunnableEx*** class.

Step 5: Inside the main method create a *Thread* object using the ***ThreadRunnableEx***. This is done by passing the ***ThreadRunnableEx*** object to the constructor of the Thread class.

Step 6: Invoke the start method on the Thread object which in turn calls the *run()* method of the ***ThreadRunnableEx*** class and starts a thread.

Example – Runnable Interface

In this demo we will learn how to

1. Create a Thread by using Runnable interface.
2. Implement the run method.
3. Create a thread object using the Runnable interface,

Components to be developed,

1. **ThreadRunnableEx** – Thread implemented using Runnable interface. Each Thread should loop and print values 0…9
2. **RunnableExMain** – The main class to start the Threads

Solution – Runnable Interface

Create a class named ***ThreadRunnableEx*** implementing the *Runnable* interface

```
public class ThreadRunneableEx implements Runnable {

    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println("Printing from  "
                    + Thread.currentThread().getName()
                    + "  , the value of i  :" + i);
            try {
                Thread.sleep(100);
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

Class ThreadRunnableEx implements *Runnable* interface override the run method to print values between 0..9.

Solution – Runnable Interface

Create class *RunnableExMain* with a main method which will start the threads.

```
public class RunnableExMain {
    public static void main(String args[]) throws InterruptedException {
        ThreadRunneableEx r1 = new ThreadRunneableEx();
        ThreadRunneableEx r2 = new ThreadRunneableEx();
        Thread t1 = new Thread(r1, "first");
        Thread t2 = new Thread(r2, "second");
        t1.start();
        t2.start();
        for (int i = 0; i < 10; i++) {
            System.out.println(Thread.currentThread().getName() + " :: " + i);
            Thread.sleep(100);
        }
    }
}
```

This creates two thread to be run

Create the Thread objects using the Runnable objects

Start the two Threads

Execute the Runnable

- Run the main class – ***ThreadExMain***
- The output will be something as shown below.

Output may vary for each execution since Thread execution depends upon the operating system.

```
main :: 0
Printing from  second  , the value of i  :0
Printing from  first  , the value of i  :0
Printing from  first  , the value of i  :1
main :: 1
Printing from  second  , the value of i  :1
Printing from  second  , the value of i  :2
main :: 2
Printing from  first  , the value of i  :2
Printing from  second  , the value of i  :3
Printing from  first  , the value of i  :3
main :: 3
Printing from  first  , the value of i  :4
Printing from  second  , the value of i  :4
main :: 4
```

Different output displayed during different run

```
main :: 0
Printing from  second  , the value of i  :0
Printing from  first  , the value of i  :0
main :: 1
Printing from  second  , the value of i  :1
Printing from  first  , the value of i  :1
main :: 2
Printing from  second  , the value of i  :2
Printing from  first  , the value of i  :2
main :: 3
Printing from  second  , the value of i  :3
Printing from  first  , the value of i  :3
main :: 4
Printing from  second  , the value of i  :4
Printing from  first  , the value of i  :4
```

Runnable vs Thread

Which Method is better? Why?

Implementing Runnable interface is recommended.

Why Runnable is better?

- If we inherit the *Thread* class for creating threads we cannot extend any other class. *Since Java does not support multiple inheritance*
- By implementing the *Runnable* interface, we can make the thread class extend other classes.

Ram faces another problem

In the previous examples we can see that the main Thread **(*Application Thread*)** exits before the other child Threads which was originated by it completes execution. This may not be desirable in some situations.

Example: Assume tax needs to be calculated for 10 employees using **calculateTax()** method, Tim the programmer decides to speedup the process by spawning 10 thread one thread per employee. Also Tim needs to ensure that the program should exit only after all the ten threads complete the tax calculation.

How can this be solved?
Tim used Thread Join Method.

What is join method?

join method in Thread class,

- ***join()*** is used to ensure that the application main thread will complete only after all the thread spawned by it inside the process completes execution.
- In other words, one thread can wait for another to complete using the join() method.
- Invoking join() guarantees that the method will not return until the threads started from it returns and join.

Example – join()

In this demo we will see how a method works with and without join .

- Step 1: We will create a Runnable class **JoinEx** and a main class **JoinExMain**
- Step 2: First we will create the main method without join() and look at the output
- Step 3: Then will inject join() to the main method and see how join changes the output.

Solution – join()

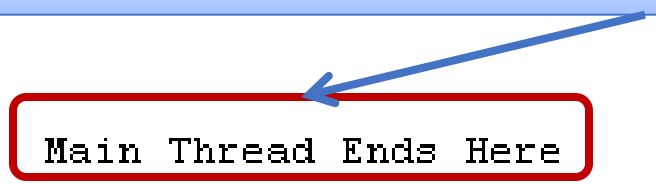
```
public class JoinEx implements Runnable {
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println("Printing from  "
                    + Thread.currentThread().getName()
                    + "  , the value of i  :" + i);
            try {
                Thread.sleep(100);
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

Main method without join, create two threads and start it.

```
public class JoinExMain {
    public static void main(String args[]) throws InterruptedException {
        JoinEx r1 = new JoinEx();
        JoinEx r2 = new JoinEx();
        Thread t1 = new Thread(r1, "first");
        Thread t2 = new Thread(r2, "second");
        t1.start();
        t2.start();
        System.out.println("Main Thread Ends Here ");
    }
}
```

Output – without join()

The main method ends before the completion of the other threads



```
Main Thread Ends Here
Printing from  second  , the value of i  :0
Printing from  first  , the value of i  :0
Printing from  second  , the value of i  :1
Printing from  first  , the value of i  :1
Printing from  second  , the value of i  :2
Printing from  first  , the value of i  :2
Printing from  first  , the value of i  :3
Printing from  second  , the value of i  :3
Printing from  first  , the value of i  :4
Printing from  second  , the value of i  :4
```

Add the join()

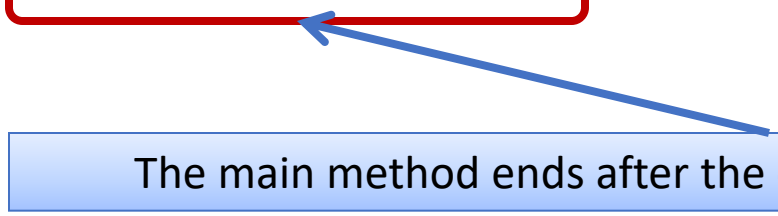
```
public class JoinExMain {
    public static void main(String args[]) throws InterruptedException {
        JoinEx r1 = new JoinEx();
        JoinEx r2 = new JoinEx();
        Thread t1 = new Thread(r1, "first");
        Thread t2 = new Thread(r2, "second");
        t1.start();
        t2.start();
        t1.join();
        t2.join();
        System.out.println("Main Thread Ends Here ");

    }
}
```

The threads join method is triggered.

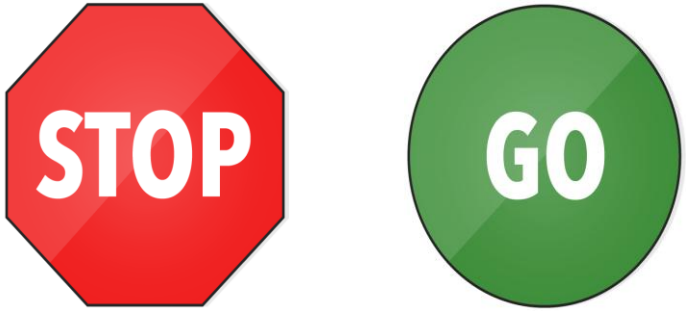
Output – with join()

```
Printing from  second  , the value of i  :0
Printing from  first  , the value of i  :0
Printing from  second  , the value of i  :1
Printing from  first  , the value of i  :1
Printing from  second  , the value of i  :2
Printing from  first  , the value of i  :2
Printing from  second  , the value of i  :3
Printing from  first  , the value of i  :3
Printing from  second  , the value of i  :4
Printing from  first  , the value of i  :4
Main Thread Ends Here
```



The main method ends after the completion of the other threads

Time To Reflect



Trainees to reflect the following topics before proceeding.

- What is Runnable interface?
- Which method is better either Thread or Runnable? Why?
- What is Thread join() method?

Thank you

You have successfully completed

Thread-II