**Multithreading – Part-03 – Create Thread Using Runnable Interface**

* Defining a Thread using Runnable Interface:

We can define a thread by implementing Runnable interface

MyThread 🡪 Thread 🡪 Runnable

MyRunnable 🡪 Runnable

Runnable interface present in java.lang package and it contains only one method which is run()

public void run();

Example:

class MyRunnable implements Runnable{

public void run(){

for(int I = 0; i< 10; i++){

System.out.println(“Child Thread”);

}

}

}

Entire class is defining a thread.

Run method block is the job of the thread.

Run block is executed by the child thread.

class ThreadDemo{

public static void main(String[] args){

MyRunnable runnable = new MyRunnable();

Thread t = new Thread(runnable);

//runnable instance is “Target runnable”

t1.start();

for(int i = 0; i < 10; i ++){

System.out.println(“main thread”);

}

}

}

The block after the t1.start() will be executed by main thread.

We will get mixed output and we can’t tell exact output.

* Case Study:

MyRunnable runnable = new MyRunnable();

Thread t1 = new Thread();

Thread t2 = new Thread(runnable);

Case\_01:

t1.start();

A new thread will be created and which is responsible for the execution of thread class run method, which has empty implementation.

Case\_02:

t1.run();

No new thread will be created and thread class run method will be executed just like a normal method call.

Case\_03:

t2.run();

A new thread will be created which is responsible for the execution of MyRunnable class run method.

Case\_04:

t2.run();

A new thread won’t be created and MyRunnable run method will be executed just like a normal method call.

Case\_05:

myRunnable.start();

We will get compile time error saying, MyRunnable class doesn’t have start capability.

CE: cannot find symbol

Symbol: method start();

Location: class MyRunnable

Case\_06:

myRunnable.run();

No new thread will be created and MyRunnable run() method will be executed like normal method.

* Which approach is best to define a thread?

Among two ways of defining a thread, implements runnable approach is recommended.

In the first approach our class always extends Thread class, there is no chance of extending any other class. Hence we are missing inheritance benefit.

But in the second approach while implementing Runnable interface we can extend any other class. Hence we won’t miss any inheritance benefit.

Because of above reason, implementing Runnable interface approach is recommended than extending Thread class.

* Thread class constructors:

Thread t = new Thread();

Thread t = new Thread(Runnable r);

Thread t = new Thread(String name);

Thread t = new Thread(Runnable r, String name);

Thread t = new Thread(ThreadGroup g, String name);

Thread t = new Thread(ThreadGroup g, Runnalbe r);

Thread t = new Thread(ThreadGroup g, Runnable r, String name);

Thread t = new Thread(ThreadGroup g, Runnable r, String name, long stackSize);

* Durga’s approach to define a Thread, not recommended to use.

class MyThread extends Thread{

public void run(){

System.out.println(“Child Thread”);

}

}

class ThreadDemo{

public static void main(String[] args){

MyThread t = new MyThread();

Thread t = new Thread(t);

t1.start();

System.out.println(“main thread”);

}

}

Output:

Child thread main thread

Main thread child thread

* Getting and Setting name of a thread:

Every thread in Java has some name, it may be default name generated by JVM or customized name provided by programmer.

We can get and set name of a thread by using the following two methods of thread class.

public final String getName();

public final void setName(String name);

Example:

class MyThread extends Thread{

}

class Test{

public static void main(String[] args){

System.out.println(Thread.currentThread().getName()); //main

MyThread t = new MyThread();

System.out.println(t.getName()); // Thread-0

Thread.currentThread().setName(“Pawan Kalyan”);

System.out.println(Thread.currentThread().getName()); //Pawan Kalyan

System.out.println(10/0);

}

}

Note:

We can get current executing thread object by using Thread.currentThreadMethod();

Example:

class MyThread extends Thread{

public void run(){

System.out.println(“Run method executed by Thread:”+Thread.currentThread().getName());

}

}

class Test{

public static void main(String[] args){

MyThread t = new MyThread();

t.start();

System.out.println(“Main method executed by Thread:”+Thread.currentThread().getName());

}

}

Output:

Main method executed by Thread: main

Run method executed by Thread: Thread-0