**- THREAD –**

|  |
| --- |
| **Thread:** The path followed when executing a program. A **single-threaded application** has only one thread and can handle only one task at a time whereas **multithreading**, executes multiple difference paths of code at the same time (since there is only one CPU, the process runs by dividing the execution time into small pieces). |

1. **INTERFAE RUNNABLE**

* Implement the runnable interface by having your own implementation of the run method
* The **Runnable interface** is implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called run.

|  |  |
| --- | --- |
|  | **Example and explanation** |
| **Thread1 (class with constructor)** | **public class TargetEx01 implements Runnable {**  **@Override**  **public void run() {**  **for (int i = 0; i < 10; i++) {**  **System.out.println(Thread.currentThread().getName() + i);**  **try {**  **Thread.sleep(500); } catch (InterruptedException e) { }}}}** |
| **Thread2 (without constructor)** | **public class TargetEx02 implements Runnable {**  **@Override**  **public void run() {**  **for (int i = 0; i < 10; i++) {**  **System.out.println(Thread.currentThread().getName() + i);**  **try {**  **Thread.sleep(500);**  **} catch (InterruptedException e) { }}}}** |
| **Main class** | **public class TargetExTestMain {**  1. Creating a object named “target1” of the class runnable that will run the function in the class TargetEx01  2. Creating a new thread named “threadA” that will run “target1” and set its name to “A”  **public static void main(String[] args) {**  **Runnable target1 = new TargetEx01();**  **Thread threadA = new Thread(target1, "A");**  **Runnable target2 = new TargetEx02();**  **Thread threadB = new Thread(target2, "B");**  **threadA.start();**  **threadB.start();**  **for (int i = 0; i < 10; i++) {**  **System.out.println("This is "+Thread.currentThread().getName() + i);**  **try {**  main  **Thread.sleep(500);**  **} catch (InterruptedException e) {}}}}** |

1. **THREAD CLASS**

* Have a class extend the thread class and override the thread class’s **run method**.
* Disadvantage: Java does not allow multiple inheritances (only one parent class)

|  |  |
| --- | --- |
|  | **Example and explanation** |
| **Thread1 (class with constructor)** | **public class TargetEx01 extends Thread {**  Giving a constructor so that user can set name of the thread whilst making a new object of class “TargetEx01”  **public TargetEx01() { }**  **public TargetEx01(String name) {**  **super(name);}**  **@Override**  **public void run() {**  **System.out.println(Thread.currentThread().getName());**  **for (int i = 0; i < 10; i++) {**  **System.out.println("TargetEx01's run method " + i);**  **try {**  Waits 0.5 seconds for each “i” in the for command  **Thread.sleep(500);**  **} catch (InterruptedException e) { }}}}** |
| **Thread2 (without constructor)** | **public class TargetEx02 extends Thread {**  **@Override**  **public void run() {**  **for (int i = 0; i < 10; i++) {**  **System.out.println(Thread.currentThread().getName());**  **System.out.println("TargetEx02's run method " + i);**  **try {**  **Thread.sleep(500);**  **} catch (InterruptedException e) { }}}}** |
| **Main class** | **public class TargetExTestMain {**  **public static void main(String[] args) {**  Goes to constructor and sets threadA’s name to “A”  **TargetEx01 threadA = new TargetEx01("A");**  **TargetEx02 threadB = new TargetEx02();**  Sets threadB’s name to “B” using the setName() class  **threadB.setName("B”);**  **threadA.start();**  **threadB.start();**  Name = main  **System.out.println(Thread.currentThread().getName());**  **System.out.println("main function"); }}** |

1. **SYNCHRONIZATION**

* Controlling the access of multiple threads by allowing only one thread to access the resource at a given point in time using synchronized blocks.
* All synchronized blocks synchronize on the same object can only have one thread executing inside them at a time.
* All other threads attempting to enter the synchronized block are blocked until the thread inside the synchronized block exits the block.

1. **OTHER CLASSES (RELATING TO THREAD)**

* **Thread.sleep(x) :** waits for x milliseconds
* **threadName.setName()**
* **threadName.join()** : waits for thread “threadName” to finish before it starts the next one
* **threadName.isAlive() :** Returns boolean of whether the thread “threadName” is still running