Runnable interface in Java

Difficulty Level : Easy Last Updated : 07 Jun, 2018

`java.lang.Runnable` is an interface that is to be implemented by a class whose instances are intended to be executed by a thread. There are two ways to start a new Thread – Subclass `Thread` and implement `Runnable`. There is no need of subclassing `Thread` when a task can be done by overriding only `run()` method of `Runnable`.

Steps to create a new `Thread` using `Runnable` :
1. Create a `Runnable` implementer and implement `run()` method.
2. Instantiate `Thread` class and pass the implementer to the `Thread`, `Thread` has a constructor which accepts `Runnable` instance.
3. Invoke `start()` of `Thread` instance, start internally calls `run()` of the implementer. Invoking `start()`, creates a new `Thread` which executes the code written in `run()`.
Calling `run()` directly doesn't create and start a new `Thread`, it will run in the same thread. To start a new line of execution, call `start()` on the thread.
Example,

```java
public class RunnableDemo {

    public static void main(String[] args)
    {
        System.out.println("Main thread is- "
                        + Thread.currentThread().getName());
        Thread t1 = new Thread(new RunnableDemo().new RunnableImpl());
        t1.start();
    }

    private class RunnableImpl implements Runnable {

        public void run()
        {
            System.out.println(Thread.currentThread().getName()
                             + ", executing run() method!");
        }
    }
}
```

Output:

```
Main thread is- main
Thread-0, executing run() method!
```

Output shows two active threads in the program – main thread and Thread-0, main method is executed by the Main thread but invoking start on `RunnableImpl` creates and starts a new thread – Thread-0.

What happens when Runnable encounters an exception ?

`Runnable` can't throw checked exception but `RuntimeException` can be thrown from `run()`. Uncaught exceptions are handled by exception handler of the thread, if JVM can't handle or catch exceptions, it prints the stack trace and terminates the flow.

Example,

```
import java.io.FileNotFoundException;

public class RunnableDemo {

    public static void main(String[] args)
    {
        System.out.println("Main thread is- " +
                           Thread.currentThread().getName());
        Thread t1 = new Thread(new RunnableDemo().new RunnableImpl());
        t1.start();
    }

    private class RunnableImpl implements Runnable {

        public void run()
        {
            System.out.println(Thread.currentThread().getName()
                               + ", executing run() method!");
            /**
             * Checked exception can't be thrown, Runnable must
             * handle checked exception itself.
             */
            try {
                throw new FileNotFoundException();
            }
            catch (FileNotFoundException e) {
                System.out.println("Must catch here!");
                e.printStackTrace();
            }

            int r = 1 / 0;
            /*
```

```
                 * Below commented line is an example
                 * of thrown RuntimeException.
                 */
                // throw new NullPointerException();
            }
        }
    }
```

Output:

```
java.io.FileNotFoundException
    at RunnableDemo$RunnableImpl.run(RunnableDemo.java:25)
    at java.lang.Thread.run(Thread.java:745)
Exception in thread "Thread-0" java.lang.ArithmeticException: / by zero
    at RunnableDemo$RunnableImpl.run(RunnableDemo.java:31)
    at java.lang.Thread.run(Thread.java:745)
```

Output shows that `Runnable` can’t throw checked exceptions, `FileNotFoundException` in this case, to the callers, it must handle checked exceptions in the `run()` but RuntimeExceptions (thrown or auto-generated) are handled by the JVM automatically.

References :

http://www.javargon.com/2016/11/javalangrunnable-interface.html