

Notes - Designing Threads

Thread -

A thread is a light weight process, it is given its own context and stack etc. for preserving the state. Thread state enables the CPU to switch back and continue from where it stopped.

Creating Threads in Java -

When you launch Java application, JVM internally creates few threads, e.g. Main thread for getting started with main() and GarbageCollector for performing garbage collection and few other threads which are need for execution of a Java application.

You can create a thread and execute the tasks with in the application, this enables you to perform parallel activities with in the application.

There are two approaches,

1) Extending the Thread class and performing the task. This is not a preferred approach because you are not extending the Thread functionality, instead you are using the Thread to execute a task, hence you should prefer the second approach.

2) Implementing the Runnable interface and then submitting this task for execution. Similarly there is a Callable interface(explained later) as well.

run() method -

Once you choose your approach, you can consider the run() method as the entry point for thread execution. To simplify just think like main() for a program, run() for a thread.

start() method -

Execution of the thread should be initiated using the start() method of the Thread class. This submits the thread for execution. This takes the associated thread to ready state, this doesn't mean it is started immediately. i.e. in simple terms, when you call the start() method, it marks the thread as ready for execution and waits for the CPU turn.

```
// 1. Extending the Thread class.
class MyThread extends Thread {

        // Thread execution begins here.
        public void run() {
                        // DO THAT TASK HERE.
                        for(int i=0; i <= 1000; i++) {
                                System.out.print("T");
                        }
        }

}

// 2. Implementing the Runnable interface,
// This marks this class as Runnable and
// assures that this class contains the run()
// method. Because any class implementing the
// interface should define the abstract method
// of the interface, otherwise it becomes abstract.
class MyTask implements Runnable {

        // Thread execution begins here.
        @Override
        public void run() {
        // DO THAT TASK HERE.
        for(int i=0; i <= 1000; i++) {
                System.out.print("-");
        }
        }
}

public class Main {

        // Will run with in the main thread.
        public static void main(String[] args) {

        // Because MyThread extends the Thread
        // class, you can call the start() method
        // directly, as it is also a member of this
        // class, courtesy inheritance relation.

        MyThread thr = new MyThread();

        thr.start();


        // MyTask is a Runnable task and not a

        // Thread and hence we need to create a

        // Thread object and assign it the task

        // Note here we are calling the start

        // method over Thread object and not on

        // task object.

        MyTask task = new MyTask();

        Thread thr2 = new Thread(task);

        thr2.start();


        for(int i=0; i <= 1000; i++) {

                System.out.print("M");

        }

        }

}
```

There are three threads in the above program (system threads ignored). One the Main thread that prints "M" and thr that prints "T" and thr2 that prints "-". Because they are executed in parallel, you will see the output like MMMTTT---MMMTTT--- ...

Notes - Transform code to achieve parallelism

Transforming serial code to parallel code using Threads.

```
package com.example.io.utils;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

/* A simple utility to copy the content
 * of the given source file to the given
 * destination file. */
public class IOUtils {

        /*
        * Copies the given source stream
        * i.e. src to the given
        * destination stream i.e. dest.
        */
        public static void copy(InputStream src, OutputStream dest) throws IOException {
```

```
        int value;

        while ((value = src.read()) != -1) {

                dest.write(value);

        }

        }


        /*

        * Copies the given srcFile to the destFile.

        */

        public static void copyFile(String srcFile, String destFile) throws IOException {

        FileInputStream fin = new FileInputStream(srcFile);

        FileOutputStream fout = new FileOutputStream(destFile);


        copy(fin, fout);


        fin.close();

        fout.close();

        }

}
```

Serial Mode -

In the below example we are making a direct call to copyFile and it is executed in serial order i.e. first Copy a.txt to c.txt is executed and then once it is done, the next copy i.e. b.txt to d.txt will be initiated.

```
}
```

Note - For this program to work you need to create two files a.txt and b.txt in the src/ folder and place some content in it.

```
import java.io.IOException;

import com.example.io.utils.IOUtils;


public class Main {


   public static void main(String[] args) throws IOException {


         String sourceFile1 = "a.txt";

         String sourceFile2 = "b.txt";


         String destFile1 = "c.txt";

         String destFile2 = "d.txt";


         // 1. Copy a.txt to c.txt

         IOUtils.copyFile(sourceFile1, destFile1);


         // 2. Copy b.txt to d.txt

         IOUtils.copyFile(sourceFile2, destFile2);

         }

}
```

Parallel Mode -

The two copy operations above are initiated through two different threads, which enables us to perform the operation in parallel. For this we defined a CopyTask which is a Runnable task, you should pass the source and the destination to the constructor which are then used to perform the copy operation once the task execution begins.

```
import java.io.IOException;

import com.example.io.utils.IOUtils;


/*

 * A Runnable task to copy the given source file

 * to the given destination file.

 */

class CopyTask implements Runnable {


        String sourceFile;

        String destFile;


        public CopyTask(String sourceFile, String destFile) {

        this.sourceFile = sourceFile;

        this.destFile = destFile;

        }


        /*
        * Initiate the copy once thread execution begins.

        */

        public void run() {

        try {

                IOUtils.copyFile(sourceFile, destFile);

                System.out.println("Copied from - " + sourceFile + " to " + destFile);

        }catch(IOException e) {

                e.printStackTrace();

        }

        }

}


public class Main {


        public static void main(String[] args) throws IOException {


        String sourceFile1 = "a.txt";

        String sourceFile2 = "b.txt";


        String destFile1 = "c.txt";

        String destFile2 = "d.txt";


        // A new thread is created to initiate copy

        // from a.txt to c.txt

        // Thread-1
        new Thread(new CopyTask(sourceFile1, destFile1)).start();


      // A new thread to initiate copy from

        // b.txt to d.txt

        // Thread-2


        new Thread(new CopyTask(sourceFile2, destFile2)).start();

        }

}
```

Important Note -

Although the main thread is completed after starting the two other threads, application won't be terminated until both the threads are done with the copy.
