# 9. Serial and Parallel Demonstration :

## Pattern Finder Example -

```java
package com.example.utils;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.List;

/*
* Below utility searches the given pattern in the file.
*/
public class PatternFinder {

   /*
    * Looks for the given pattern in the file,
    * and returns the list of line numbers
    * in which the pattern is found.
    */
   public List<Integer> find(File file, String pattern) {

       List<Integer> lineNumbers = new ArrayList<Integer>();

       // Open the file for reading.
       try (BufferedReader br = new BufferedReader(new FileReader(file))) {

           int lineNo = 1;
           String line;

           // for each line in the file.
           while ( (line = br.readLine()) != null) {

               if (line.contains(pattern)) {
                   // capture the lineNo where the pattern is found.
                   lineNumbers.add(lineNo);
               }

               lineNo++;
           }

       } catch(Exception e) {
           e.printStackTrace();
       }
       // Just introduced the delay for demo.
       try { Thread.sleep(1000); } catch(Exception e) {}

       return lineNumbers;
   }

}
```

*Note - For this program to work, create a folder named "sample" under "src" folder and create few files with content with in "sample" folder.*

## Serial approach -

Here we are not using threads, instead we are searching each file in sequential order.

```
import java.io.File;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

import com.example.utils.PatternFinder;

public class Main {

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

       // pattern to search
       String pattern = "public";

       // Directory or folder to search
       File dir = new File("./src/sample");

       // list all the files present in the folder.
       File [] files = dir.listFiles();

       PatternFinder finder = new PatternFinder();

       long startTime = System.currentTimeMillis();

       // for each file in the list of files

       for (File file : files) {

           List<Integer> lineNumbers = finder.find(file, pattern);
           if (! lineNumbers.isEmpty()) {
              System.out.println(
                   pattern + "; found at " + lineNumbers +
                   " in the file - " + file.getName());
           }

       }

       System.out.println(
           " Time taken for search - " + (System.currentTimeMillis() - startTime));

   }
}
```

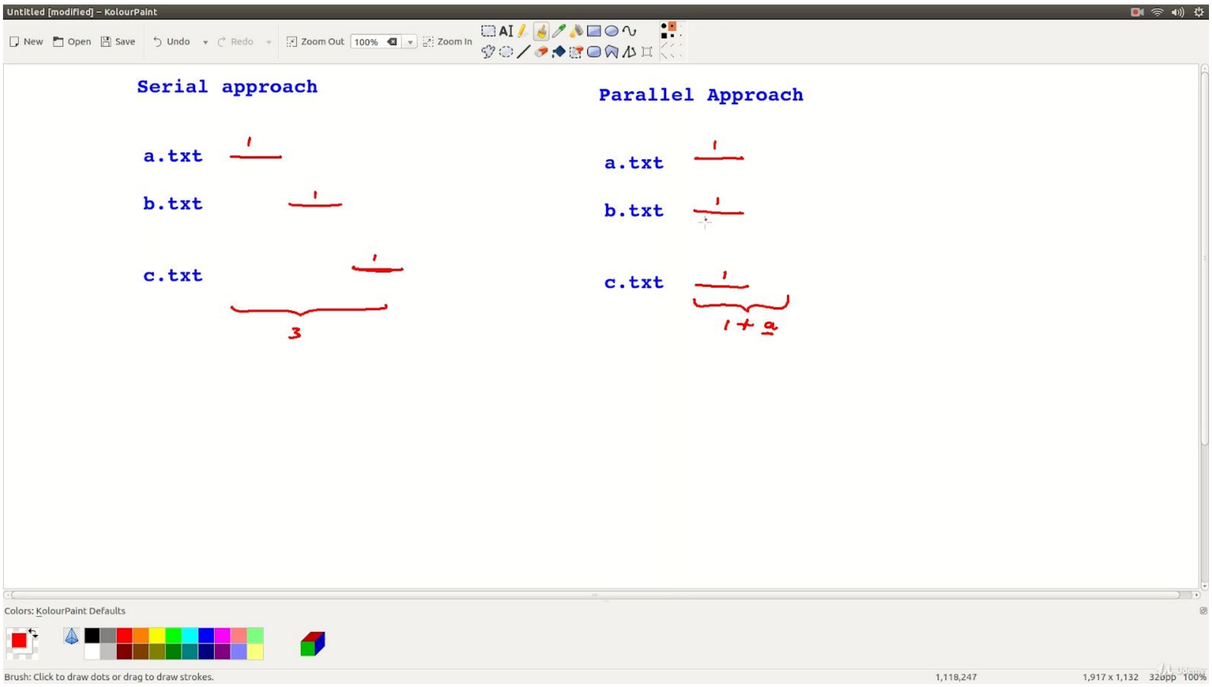
## Parallel approach -

Here we are creating a fixed thread pool of size 3 and using it to search the files, so that we can scan 3 files in parallel. And taking the help of the Future object to return the search result.

```
import java.io.File;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

import com.example.utils.PatternFinder;

public class Main {

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

       String pattern = "public";

       File dir = new File("./src/sample");
       File [] files = dir.listFiles();

       PatternFinder finder = new PatternFinder();

       // Fixed thread pool of size 3.
       ExecutorService executor = Executors.newFixedThreadPool(3);

       // Map to store the Future object against each
```

```
    // file search request, later once the result is obtained
    // the Future object will be
    // replaced with the search result.

    Map<String, Object> resultMap = new HashMap<String,Object>();

    long startTime = System.currentTimeMillis();

    for (File file : files) {

        // Submit a Callable task for the file.
        Future<List<Integer>> future =
           executor.submit(
               new Callable<List<Integer>>() {
                  public List<Integer> call() {
                      List<Integer> lineNumbers = finder.find(file, pattern);
                      return lineNumbers;
                  }
               });

        // Save the future object in the map for
        // fetching the result.

        resultMap.put(file.getName(), future);
    }

    // Wait for the requests to complete.
    waitForAll( resultMap );

    // Display the result.
    for (Map.Entry<String, Object> entry : resultMap.entrySet()) {
           System.out.println(
                pattern + " found at - " + entry.getValue() +
                " in file " + entry.getKey());
    }


    System.out.println(
        " Time taken for search - "
        + (System.currentTimeMillis() - startTime));

}

private static void waitForAll(Map<String, Object> resultMap)
                        throws Exception {

    Set<String> keys =  resultMap.keySet();

    for (String key : keys) {
        Future<List<Integer>> future =
                (Future<List<Integer>>) resultMap.get(key);

        while (! future.isDone()) {
               // Passing the CPU to other
               // threads so that they can
               // complete the operation.
               // With out this we are simply
               // keeping the CPU in loop and
               // wasting its time.

              Thread.yield();
           }

           // Replace the future object with the obtained result.
           resultMap.put(key, future.get());
       }

   }
}
```

**IMPORTANT NOTE - If a thread doesn't need CPU, it is always a good idea to pass the control to the other threads so that CPU time is effectively utilized.**



Brush: Click to draw dots or drag to draw strokes.