**Exception Handling – Part-01 – Introduction**

* **Agenda:**
  + Introduction.
  + Runtime stack mechanism.
  + Default exception handling in Java.
  + Exception Hierarchy.
  + Customized exception handling by using try catch.
  + Control flow in try catch.
  + Methods to print exception information.
  + Try with multiple catch blocks.
  + Finally block.
  + Difference between final, finally, finalize.
  + Control flow in try-catch-finally.
  + Control flow in nested try-catch-finally.
  + Various possible combinations of try catch finally.
  + throw keyword.
  + throws keyword.
  + Exception handling keywords summary.
  + Various possible compile time errors in exception handling.
  + Customized or user defined exceptions.
  + Top – 10 exceptions.
  + 1.7 version enhancements
    - try with resources.
    - Multi-catch block.
* **Introduction:**

An unexpected unwanted event that disturbs normal flow of the program is called exception.

Example:

TyrePuncturedException

SleepingException

FileNotFoundException

It is highly recommended to handle exception and main objective of exception handling is graceful termination of the program.

Exception handling doesn’t mean repairing an exception, we have to provide an alternative way to continue rest of the program normally is the concept of exception handling.

For example, our programming requirement is to read data from remote file located at London at runtime if London file is not available our program shouldn’t be terminated abnormally. We have to provide some local file to continue rest of the program normally.

Example:

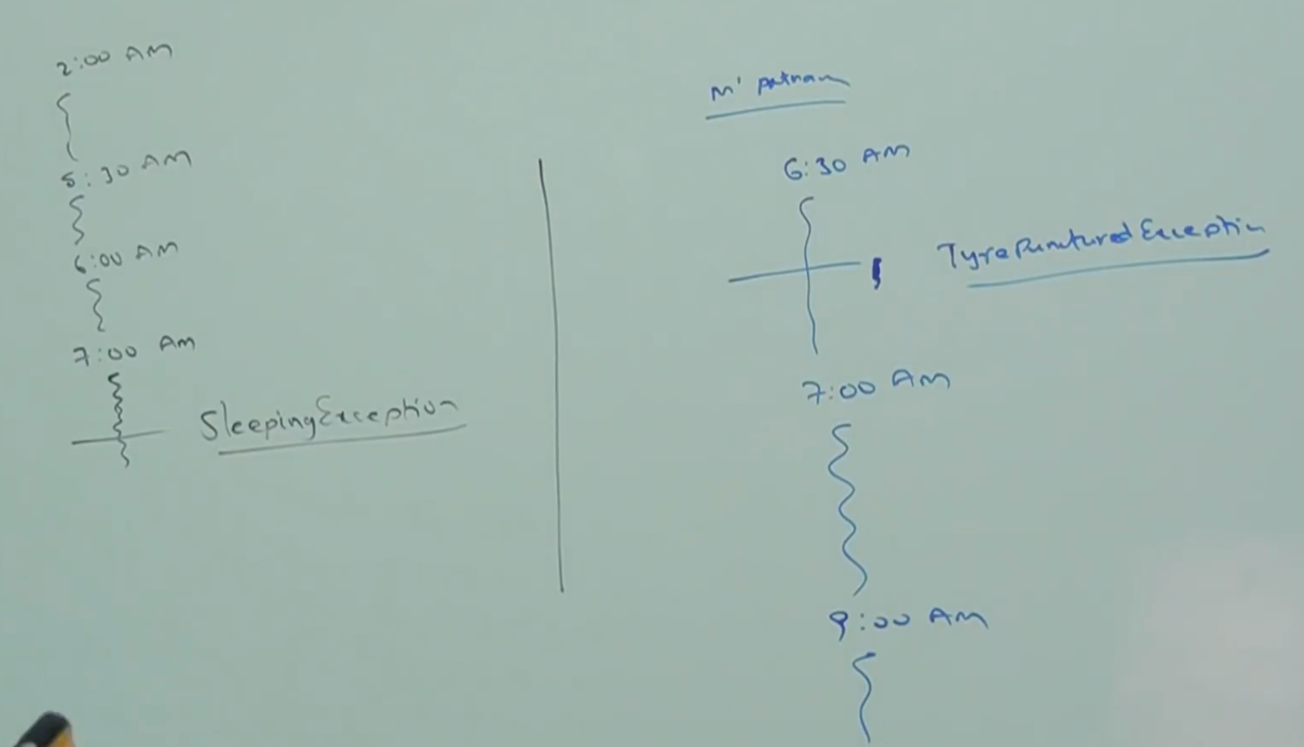
try{

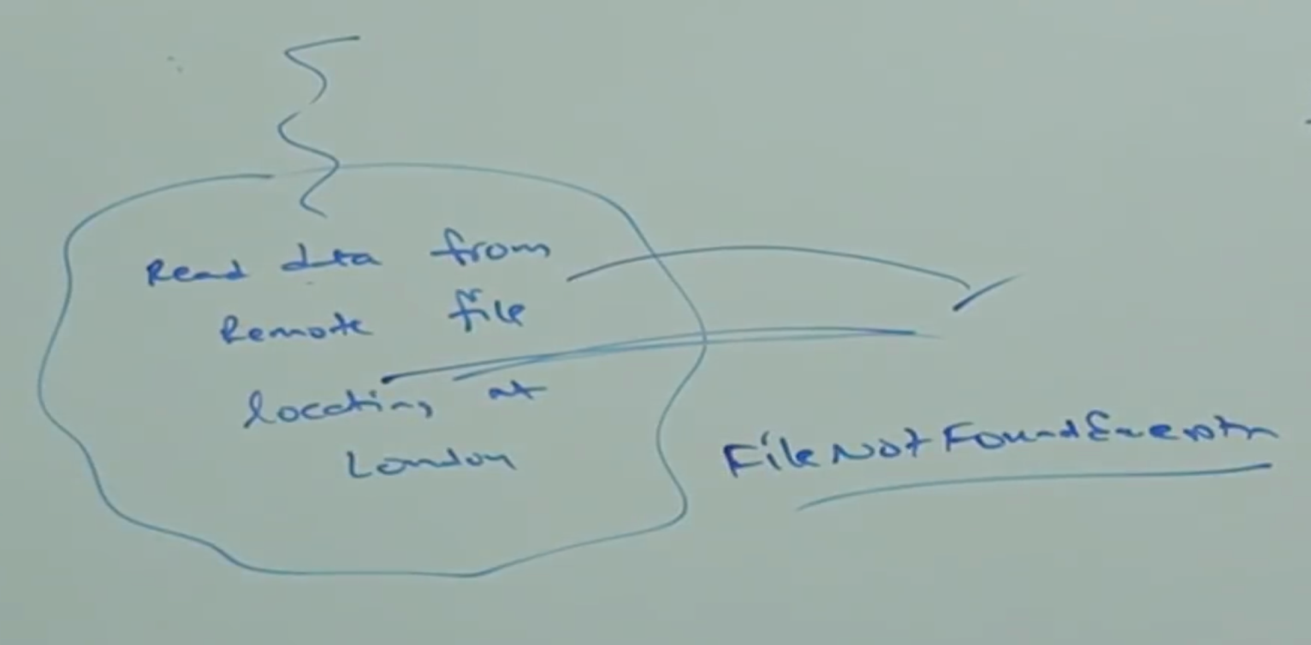
Read data from remote file located at London

} catch(FileNotFoundException e){

Use local file & continue rest of the program normally

}

****

****

* **Runtime Stack Mechanism:**

For every thread JVM will create a runtime stack, each and every method call performed by that thread will be stored in the corresponding stack. Each entry in the stack is called Stack Frame or Activation Record. After completing every method call the corresponding entry from the stack will be removed. After completing all method calls, the stack will become empty and that empty stack will be destroyed by JVM just before terminating the thread.

Example:

class Test{

public static void main(String[] args){

doStuff();

}

public static void doStuff(){

doMoreStuff();

}

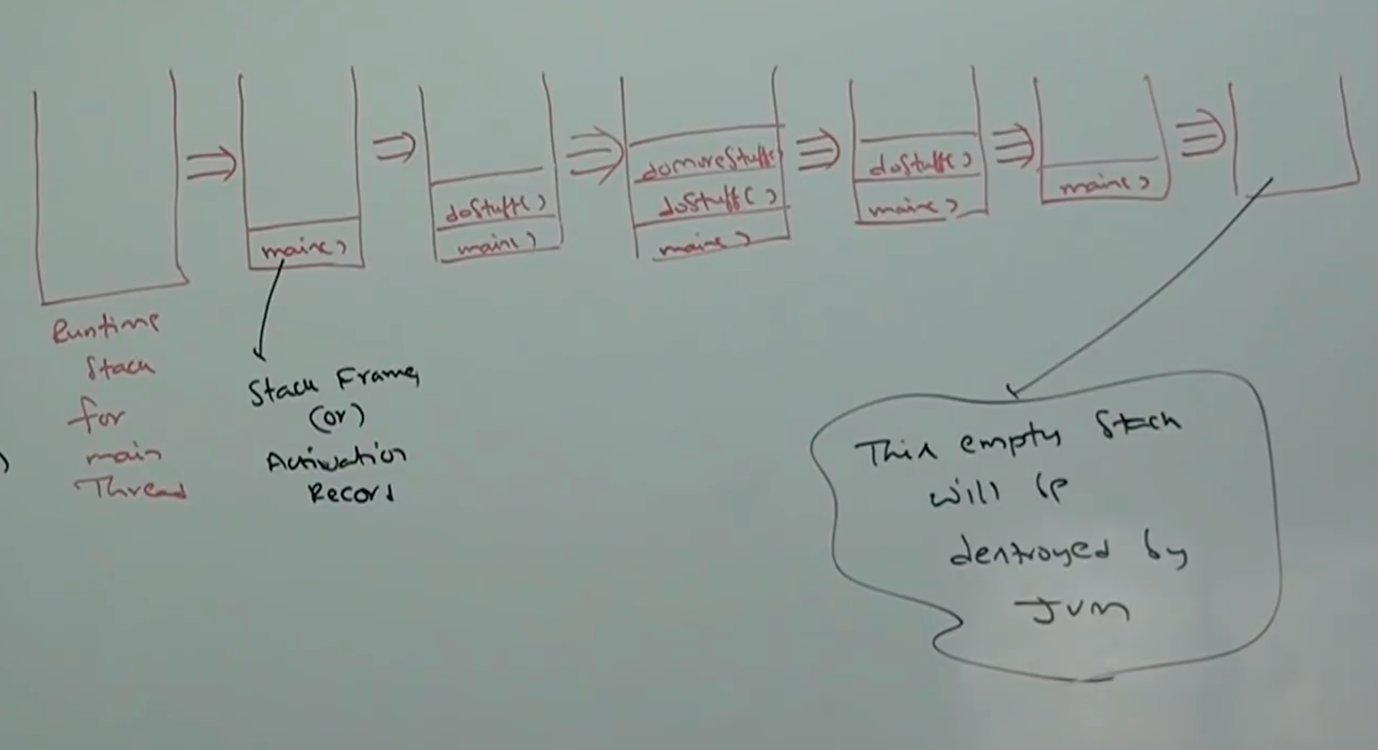
public static void doMoreStuff(){

System.out.println(“Hello”);

}

}

**Output: Hello**

****