Exception Handling

Exceptions are used in a program to signal that some error or exceptional situation has occurred, and that it doesn't make sense to continue the program flow until the exception has been handled. A method may throw an exception for many reasons, for instance if the input parameters is invalid.

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

class ExceptionHandlingBasic {
      public static void main(String []args){
            System.out.println("loading file .....");
            new ExceptionHandlingBasic().openFile();
            System.out.println("finished reading file .....");
      }
    public void openFile(){
        try {
            FileReader reader = new
FileReader("/home/neha/java.info/testFile"); // constructor can throw
FileNotFoundException
            int i=0;
            while(i != -1){
                //reader.read() may throw IOException
                i = reader.read();
                System.out.print((char) i );
            }
            reader.close();
            System.out.println("--- File End ---");
        } catch (FileNotFoundException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        } catch (IOException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
}
```

```
OUPUT:
loading file .....
https://github.com/expressjs/express-paginate
https://www.npmjs.com/package/pagination
https://programmerblog.net/nodejs-mysql-pagination-example-beginners/

□--- File End ---
finished reading file .....
```

Throwing and Catching Checked Exceptions

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

class ExceptionHandlingBasic {
        public static void main(String []args){
                System.out.println("loading file .....");
                new ExceptionHandlingBasic().openFile();
                System.out.println("finished reading file .....");
        }
   public void openFile(){
      try {
         FileReader reader = new FileReader("/home/neha/java.info/invalidFileName"); //
constructor can throw FileNotFoundException
         int i=0;
         while(i != -1){
            //reader.read() may throw IOException
            i = reader.read();
            System.out.print((char) i );
         }
         reader.close();
         System.out.println("--- File End ---");
      } catch (FileNotFoundException e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      } catch (IOException e) {
         System.out.println(e.getMessage());
```

```
            e.printStackTrace();
        }
    }
}
```

```
OUTPUT:
loading file .....
/home/neha/java.info/invalidFIle (No such file or directory)
java.io.FileNotFoundException: /home/neha/java.info/invalidFIle (No such file or directory)
            at java.io.FileInputStream.open(Native Method)
            at java.io.FileInputStream.<init>(FileInputStream.java:146)
            at java.io.FileInputStream.<init>(FileInputStream.java:101)
            at java.io.FileReader.<init>(FileReader.java:58)
            at ExceptionHandlingBasic.openFile(ExceptionHandlingBasic.java:13)
            at ExceptionHandlingBasic.main(ExceptionHandlingBasic.java:8)
finished reading file .....
```

Exception propagation in the Call Stack

main

printLogs

openFile

By the call stack is meant the sequence of method calls from the current method and back to the Main method of the program. If a method main calls printLogs, and printLogs calls openFile then the call stack looks like this. Exception are propagated up the call stack, from the method that initially throws it, until a method in the call stack catches it.

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

class ExceptionHandlingBasic {
            public static void main(String []args){
                try {
                        System.out.println("In Main method .....");
                        new ExceptionHandlingBasic().printLogs();
                        System.out.println("Existing Main Method .....");
                } catch (FileNotFoundException e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
          } catch (IOException e) {
              System.out.println(e.getMessage());
              e.printStackTrace();
          }
        }
        public void printLogs() throws FileNotFoundException, IOException{
              System.out.println("loading file .....");
              new ExceptionHandlingBasic().openFile();
              System.out.println("finished reading file .....");
        }
    public void openFile() throws FileNotFoundException, IOException {
        FileReader reader = new FileReader("/home/neha/java.info/testFile");
// constructor can throw FileNotFoundException
        int i=0;
        while(i != -1){
            //reader.read() may throw IOException
            i = reader.read();
            System.out.print((char) i );
        }
        reader.close();
        System.out.println("--- File End ---");
    }
}
```

If an exception is thrown from the reader.read() method then program execution is halted, and the exception is passed up the call stack to the method that called openFile(). If the calling method has a try-catch block, the exception will be caught there. If the calling method also just throws the method on, the calling method is also interrupted at the openFile() method call, and the exception passed on up the call stack. The exception is propagated up the call stack like this until some method catches the exception, or the Java Virtual Machine does.