Java Tutorial

MutliThreading

Introduction

Every java application runs with the concept of threads or multithreading

Thread is a part of a program which can execute individually

Executing multiple threads simultaneously is

Multithreading

Introduction

To understand Multithreading concept we should know the following…

What is Multitasking?

What is Multithreading?

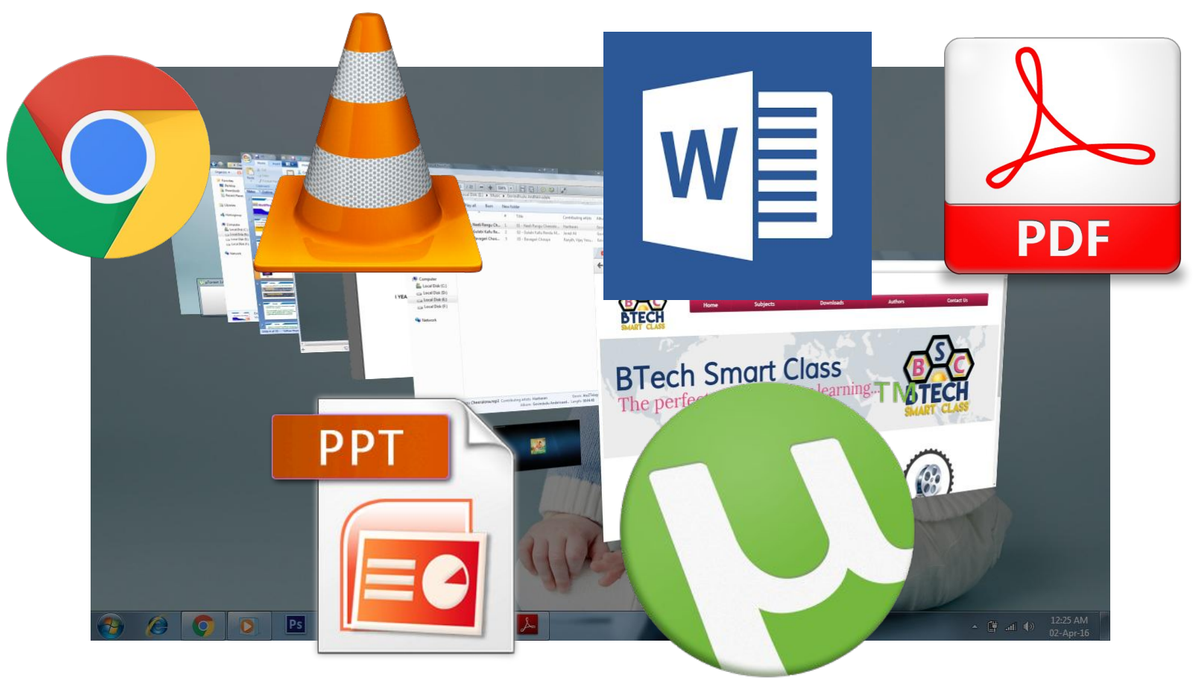
What is Concurrency?

What is Parallelism?

Introduction

What is Multitasking?

It is the ability to perform multiple jobs concurrently. That is running multiple programs concurrently



MS Word Spell Checking

Word suggestion in mobile

Playlist in MediaPlayer

Video Games

Introduction

What is Concurrency?

It is the process of executing multiple processes simultaneously on a single CPU

What is Parallelism?

It is the process of executing multiple processes simultaneously on individual CPUs

Thread

Definition: A part of the program which can run individually

```
public File Downloader(String url)
{
      Code for downloading a file
      -
      -
      return  File;
}
```

What happens if I want to download 4 File?

- Execute the program for 4 times
- Create 4 objects and call the method one after the other

The threads concept makes it easy

Thread

When a java program is executed automatically a thread is created known as **main thread** by Java-Runtime system

In java, every thread is created with the help of built-in class

Thread

which is in the

java.lang

package

```
class CurrentThreadDemo {
    public static void main(String args[]) {
        Thread t = Thread.currentThread();
        System.out.println("Current thread: " + t);
        // change the name of the thread
        t.setName("My Thread");
        System.out.println("After name change: " + t);
        try {
                for(int n = 5; n > 0; n--) {
                        System.out.println(n);
                        Thread.sleep(1000);
                }
        }
        catch (InterruptedException e) {
                System.out.println("Main thread interrupted");
        }
    }
}
```

Creating Thread

Create an object of Thread class

```
Thread  t  = new  Thread( ) ;
```

Override run() method with the code to be run by that thread

```
public void run( ){
    Code to be run by the
    thread
    .....
}
```

Call the run( ) method using start( ) method

```
t . Start( );
```

Concept

Exception handling will not corrects the runtime error, but just informs the user by providing some information about that error.

Reasons

An exception can occur for many different reasons, including the following….

- A user has entered invalid data
- A file that needs to be opened cannot be found
- A network connection has been lost in the middle of communications or the JVM has run out of memory
- Physical problem like device not working, cable related problem

Handling
Exceptions

To handle an exception in java we use the following keywords…

1. try
2. catch
3. finally
4. throw
5. throws

Handling
Exceptions

To handle an exception in java we use the following keywords…

1. try
2. catch
3. finally
4. throw
5. throws

The try keyword is used to define a block of statements which may generate exception

Handling Exceptions

To handle an exception in java we use the following keywords…

2 catch

1 try

3 finally

The catch keyword is used to define a block of statements which can handle the exception occurred in try block

Every try block must have atleast one catch block

The try block may have multiple catch blocks

Handling
Exceptions

To handle an exception in java we use the following keywords…

3 finally

1 try

The finally keyword is used to a block of state[illegible]ch must be execute irres[illegible]Exception occurance

2 catch

4 throw

5 throws

Handling Exceptions

To handle an exception in java we use the following keywords…

4. throw
1. try
2. catch
3. finally
5. throws

The throw k[illegible] used to throw an e[illegible]xplicitly.

throw Keyword

Example

```
import java.io.*;
class ThrowDemo{
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter any two Integer values");
        int a = input.nextInt();
        int b = input.nextInt();
        try{
            if(b == 0)
                throw new Exception("Can not divide!!!");
            int result = a / b;
            System.out.println("Result of a / b = "+result);
        }
        catch(Exception e){
             System.out.println(e);
        }
    }
}
```

Handling Exceptions

To handle an exception in java we use the following keywords…

5. throws
1. try
2. catch
3. finally
4. throw

The throws keyword is used to list the types of Exceptions that a method might throw.

throws Keyword

Example

```
class ThrowsDemo {
    void myMethod(int n) throws IOException,ClassNotFoundException {
        if(n ==1)
       throw new IOException(“Message 1 !!!”);
        else
       throw new ClassNotFoundException(“Message 2 !!!”);
    }
}
class ThrowsDemoTest  {
    public static void main(String args[]) {
        ThrowsDemo obj = new ThrowsDemo();
        try{
            obj.myMethod(1);
        }
        catch(Exception e){
             System.out.println(e);
        }
    }
}
```

Categories

In java there are TWO types of exceptions

Checked Exceptions

The checked exceptions are checked at compile-time

Unchecked Exceptions

The unchecked exceptions are checked at runtime

Checked Exceptions

Checked exceptions should handle the exception using **try - catch** block or it should declare the exception using throws keyword, otherwise the program will give a compilation error.

IOException
SQLException
DataAccessException
ClassNotFoundException

Checked Exceptions

Example

```
import java.io.*;
class Example {
    public static void main(String args[]) {
        FileInputStream fis = new FileInputStream("B:/myfile.txt");
        /*This constructor FileInputStream(File filename) throws FileNotFoundException */
        int k;
        while(( k = fis.read() ) != -1) {
            System.out.print((char)k);
        }
        fis.close();
        /*The method close() closes the file input stream * It throws IOException*/
    }
}
```

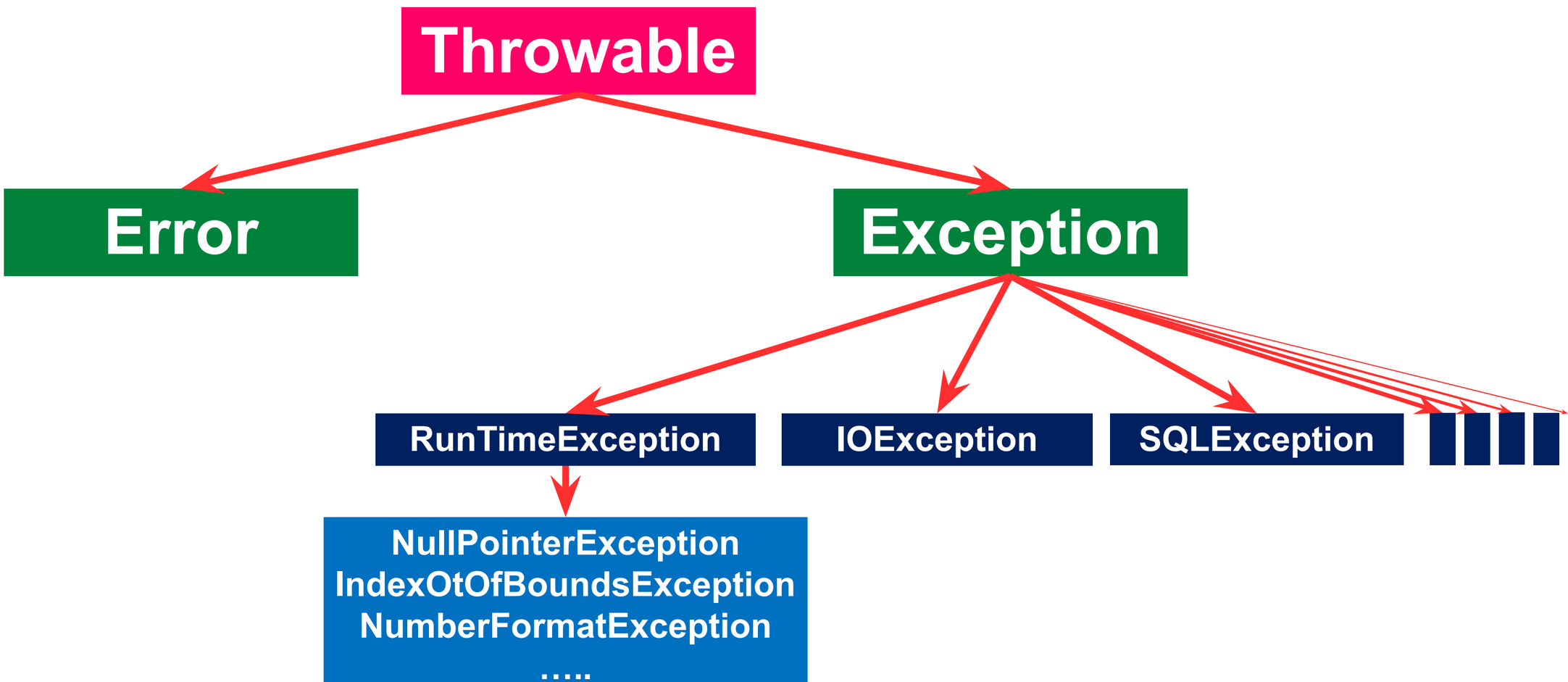
Unchecked Exceptions

Unchecked Exceptions mostly arise due to programming errors like accessing method of a null object, accessing element outside an array bonding or invoking method with illegal arguments etc,.

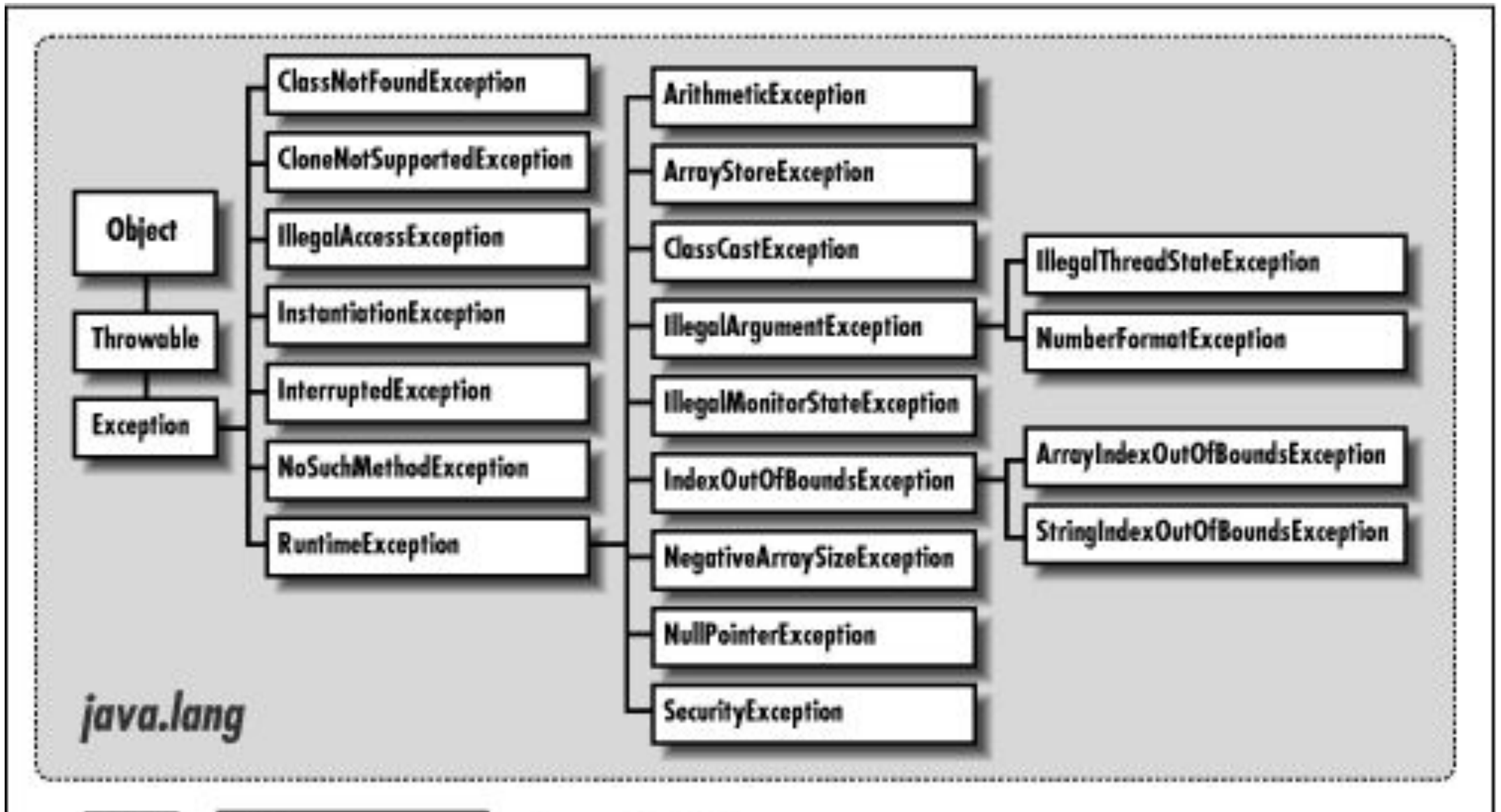
NullPointerException
ArrayIndexOutOfBound
IllegalArgumentException
IllegalStateException

Exception Hierarchy

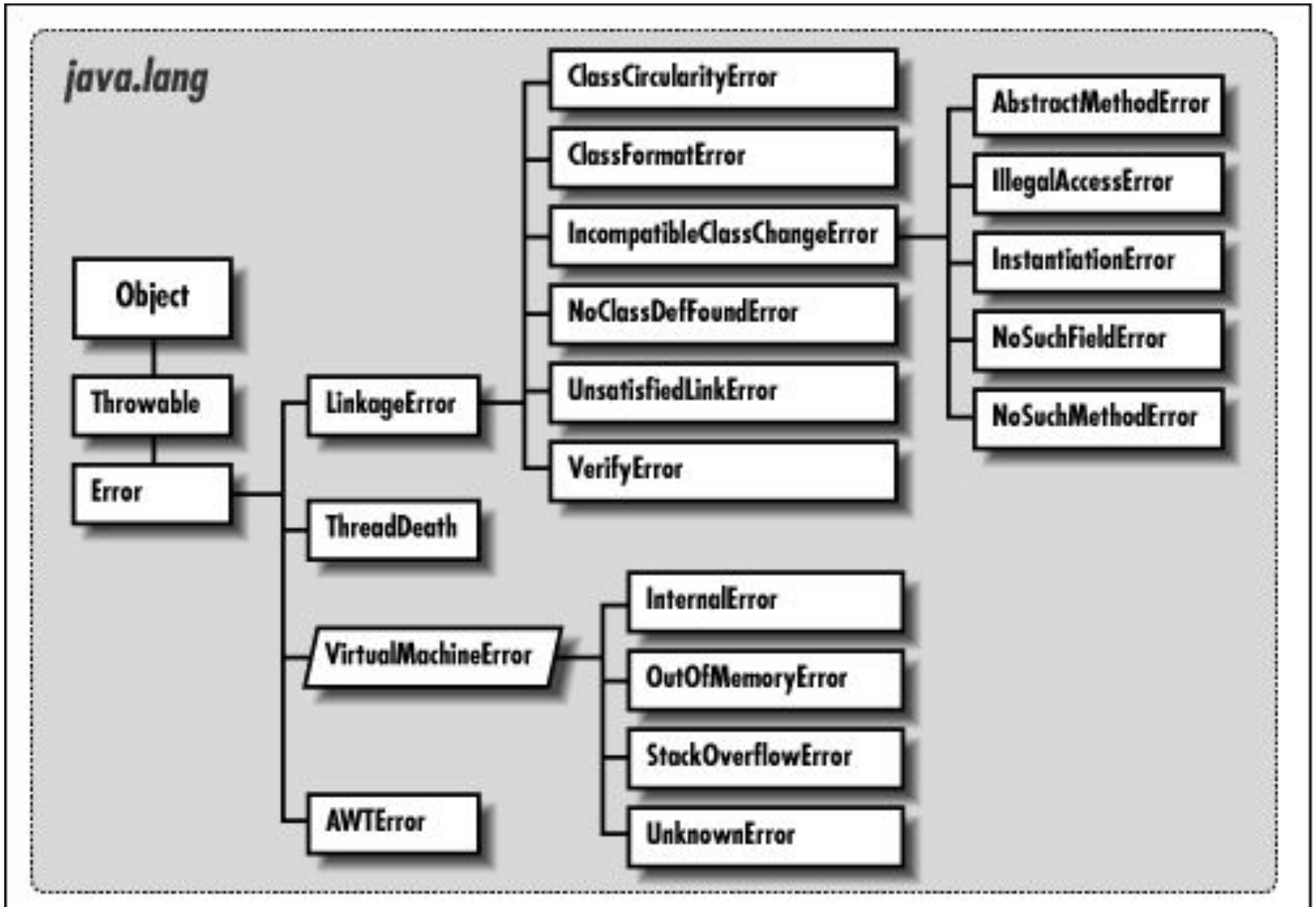
In java all the Exceptions are defined as Classes to handle them. All those classes are in following hierarchy…



Exception Hierarchy



Exception Hierarchy



Creating Exceptions

To create your own exception types to handle situations just define a subclass of Exception

```
class MyOwnException extends Exception {
    public MyOwnException(String msg){
        super(msg);
    }
}
class EmployeeTest {
    static void employeeAge(int age) throws MyOwnException{
        if(age < 0)
            throw new MyOwnException("Age can't be less than zero");
        else
            System.out.println("Input is valid!!");
    }
    public static void main(String[] args) {
        try { employeeAge(-2); }
        catch (MyOwnException e){ e.printStackTrace(); }
    }

}
```