PROGRAMMING

Lecture 18

Sushil Paudel

PREVIOUS TOPIC

- Abstraction
- Abstract Class
- Interface

TODAY'S TOPIC

- Exception Handling
- Try Catch

EXCEPTION HANDLING

- Bad things happens.
- You have to pretend that everything is going to fail.

EXCEPTION HANDLING

- An exception (or exceptional event) is a problem that arises during the execution of a program.
- When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

EXCEPTION HANDLING

- An exception can occur for many different reasons. Following are some scenarios where an exception occurs.
  - A user has entered an invalid data.
  - A file that needs to be opened cannot be found.
  - A network connection has been lost in the middle of communications, etc.

EXCPETION HANDLING

- The core advantage of exception handling is **to maintain the normal flow of the application**.
- An exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

SCENARIO

- statement 1;
- statement 2;
- statement 3;
- statement 4;
- statement 5;//exception occurs
- statement 6;
- statement 7;
- statement 8;
- statement 9;
- statement 10;

SCENARIO

- Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed.
- If we perform exception handling, the rest of the statement will be executed.
- That is why we use exception handling in Java.

TYPES OF EXCEPTION

There are two types of exceptions as below:

- Checked Exception
- Unchecked Exception

CHECKED EXCEPTION

- Checked exceptions are checked at compile time to ensure you are handling them, either by catching them or declaring the containing method throws the exception.
- If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword. Else it will throw compile time error.

Eg. FileNotFoundException

```
File file = new File("E://file.txt");
FileReader fr = new FileReader(file);
```

UNCHEKED EXCEPTION

- An unchecked exception is an exception that occurs at the time of execution.
- These are also called as **Runtime Exceptions**.
- These include programming bugs, such as logic errors or improper use of an API.
- Runtime exceptions are ignored at the time of compilation.
- For example, if you have declared an array of size 5 in your program, and trying to call the 6^{th} element of the array the an *ArrayIndexOutOfBoundsExceptionexception* occurs.

UNCHECKED EXCEPTION

- **Example**

```
int num[] = { 1, 2, 3, 4 };
System.out.println(num[5]);
```

- **Output**

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at com.exception.ExceptionTest.exceptionUnchecked(ExceptionTest.java:19)
        at com.exception.ExceptionTest.main(ExceptionTest.java:9)
```

EXCEPTION METHOD

Sr.No.	Method & Description
1	public String getMessage() Returns a detailed message about the exception that has occurred.
2	public Throwable getCause() Returns the cause of the exception as represented by a Throwable object.
3	public String toString() Returns the name of the class concatenated with the result of getMessage().
4	public void printStackTrace() Prints the result of toString() along with the stack trace to System.err, the error output stream.

COMMON SCENARIO OF EXCEPTION

Scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException.

```
int a=50/0;//ArithmeticException
```

COMMON SCENARIO OF EXCEPTION

Scenario where NullPointerException occurs

If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

```
String s=null;
System.out.println(s.length());//NullPointerException
```

SCENARIO 3

Scenario where NumberFormatException occurs

The wrong formatting of any value, may occur NumberFormatException. Suppose I have a string variable that have characters, converting this variable into digit will occur NumberFormatException.

```
String s="abc";
int i=Integer.parseInt(s);//NumberFormatException
```

SCENARIO 4

Scenario where ArrayIndexOutOfBoundsException occurs

If you are inserting any value in the wrong index, it would result

ArrayIndexOutOfBoundsException as shown below:

```
int a[]=new int[5];
a[10]=50; //ArrayIndexOutOfBoundsException
```

EXCEPTION HANDLING METHODS

- Try-catch
- Throw
- Throws

TRY-CATCH

- Java try block is used to enclose the code that might throw an exception. It must be used within the method. Java try block must be followed by either catch or finally block.

```
try {
        //code that may throw exception
} catch (Exception ex) {

}
```

EXAMPLE

```
try {
        int data = 50 / 0;
} catch (ArithmeticException e) {
        e.printStackTrace();
}
System.out.println("Hello after try-catch...");
```

TRY CATCH FINALLY

- **Java finally block** is a block that is used *to execute important code* such as closing connection, stream etc. Java finally block is always executed whether exception is handled or not. Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.

```
try{

}catch(Exception ex){

}finally {

}
```

EXAMPLE

```
try{
        System.out.println("Try 1");
}catch(Exception ex){
        System.out.println("Catch 1");
}finally {
        System.out.println("Finally 1");
}

System.out.println("\n");

try{
        System.out.println("Try 2");
        String str = null;
        int length = str.length();

}catch(Exception ex){
        System.out.println("Catch 2");
}finally {
        System.out.println("Finally 2");
}
```

OUTPUT

```
Try 1
Finally 1


Try 2
Catch 2
Finally 2
```

MULTIPLE CATCH

- A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

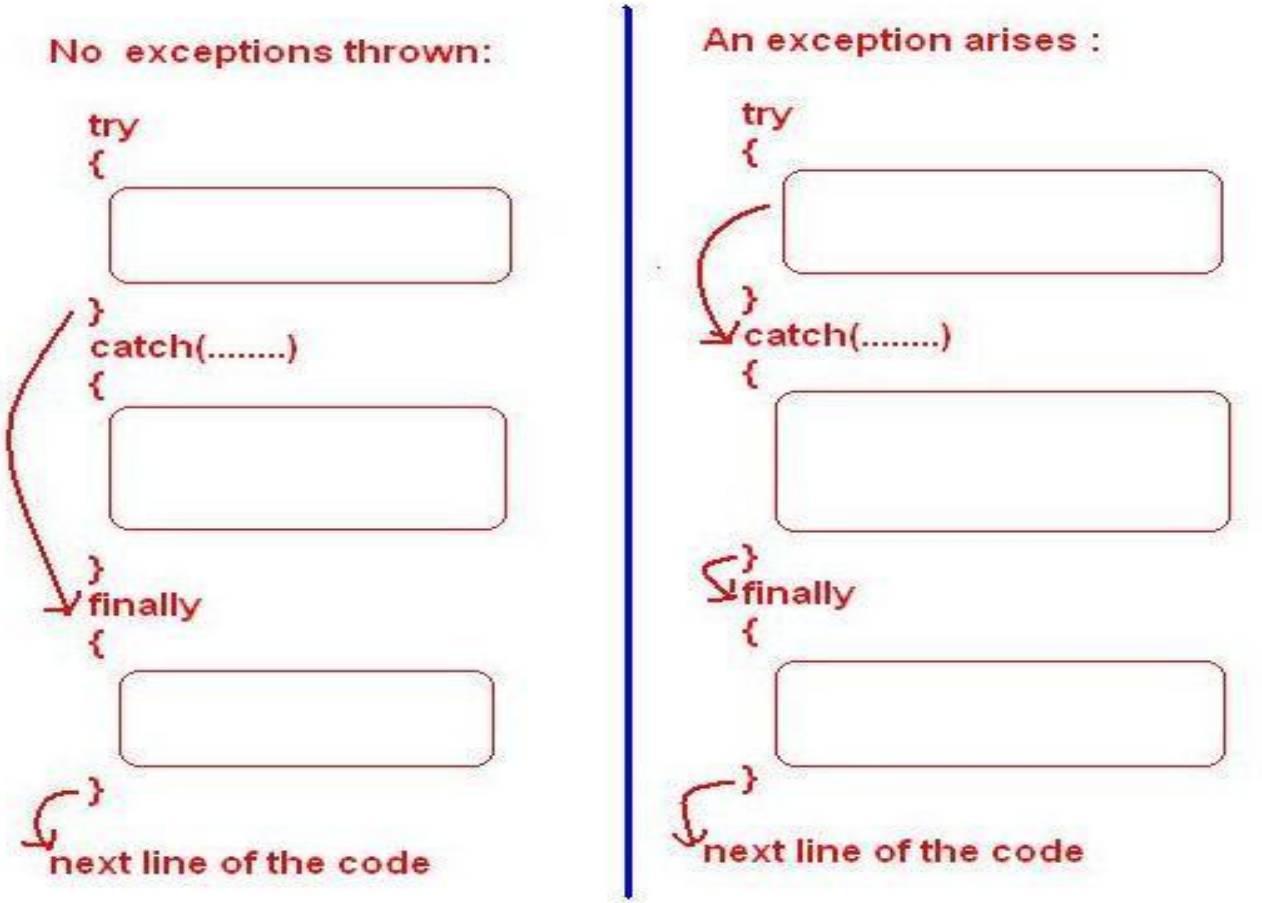
EXAMPLE

```
public class MultipleCatchBlock1 {
    public static void main(String[] args) {
            try{
                int a[]=new int[5];
                a[5]=30/0;
            }
            catch(ArithmeticException e)   {
                    System.out.println("Arithmetic Exception occurs");
             }
             catch(ArrayIndexOutOfBoundsException e)  {
                    System.out.println("ArrayIndexOutOfBounds Exception occurs");
             }
             catch(Exception e)  {
                    System.out.println("Parent Exception occurs");
             }
             System.out.println("rest of the code");
    }
}
```

OUTPUT

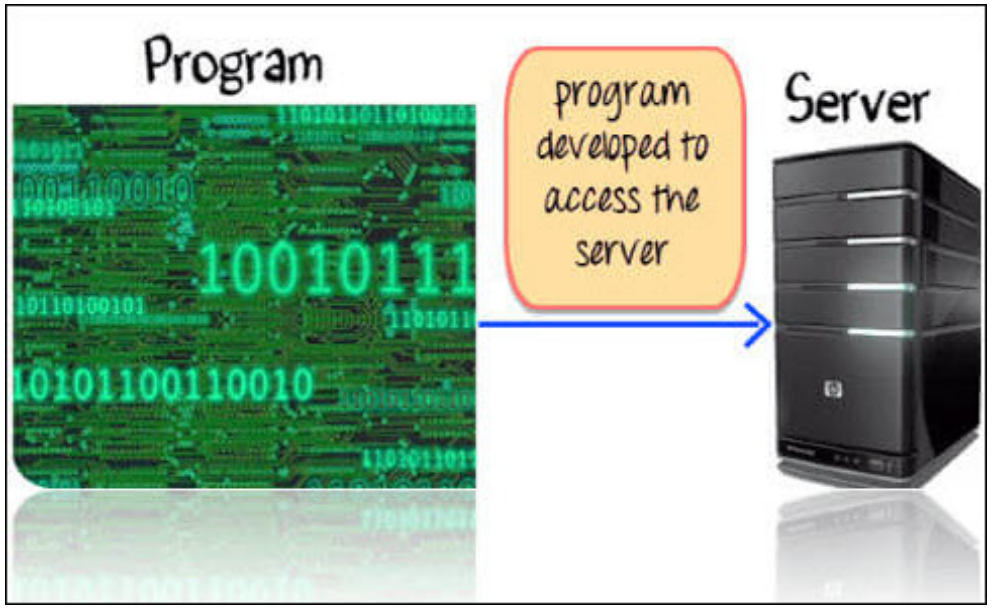
- Arithmetic Exception occurs
- rest of the code

TRY CATCH FINALLY



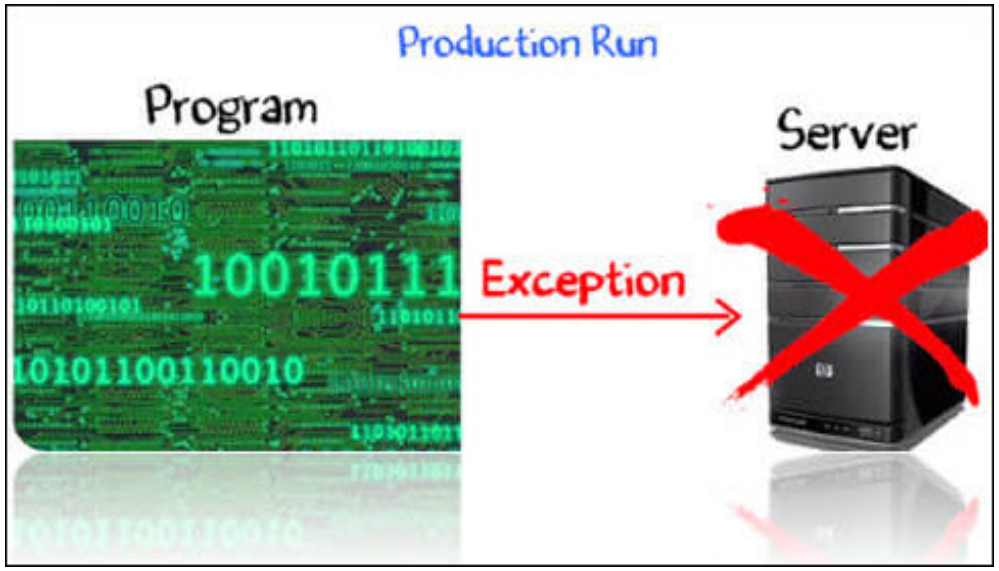
EXAMPLE

- Suppose you have coded a program to access the server. Things worked fine while you were developing the code.

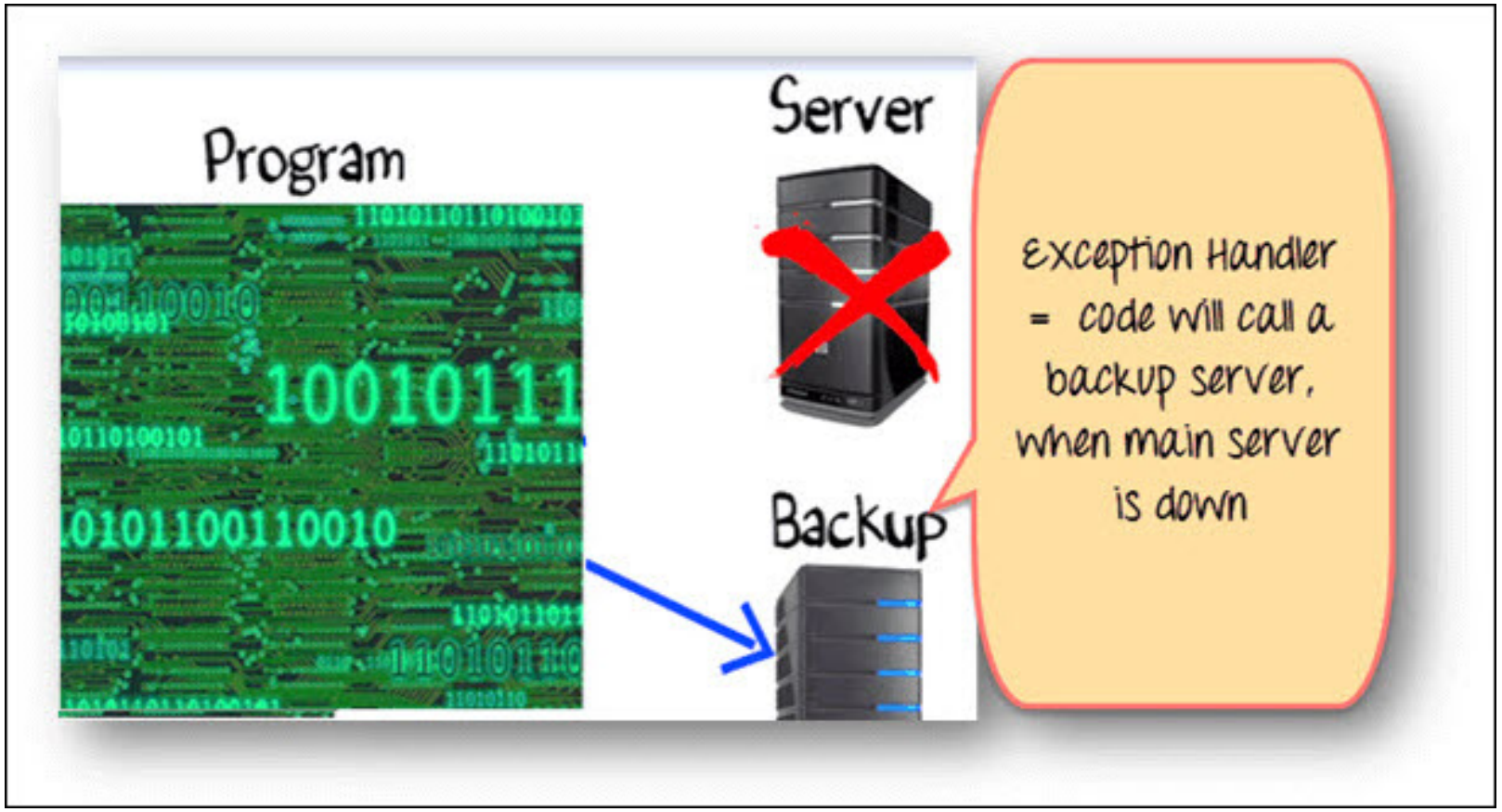


EXAMPLE

- During the actual production run, the server is down. When your program tried to access it, an exception is raised.



SOLUTION


Program
Server
Backup
Exception Handler = code will call a backup server, when main server is down

EXAMPLE

```
public class TryCatchExample1 {

        public static void main(String[] args) {

                int data= 1;
               System.out.println(”Rest of the code");
         }
}
```

OUTPUT

Rest of the code

EXAMPLE

```
public class TryCatchExample2 {

        public static void main(String[] args) {
                try {
                     int data=“f”;
                } catch(ArithmeticException e)  {
                     System.out.println(e);
                }
                System.out.println("rest of the code");
          }
}
```

OUTPUT

Compilation Error

THROW

- The throw keyword in Java is used for explicitly throwing a single exception. This can be from within a method or any block of code.
- Both checked and unchecked exceptions can be thrown using the throw keyword.
- When an exception is thrown using the throw keyword, the flow of execution of the program is stopped and the control is transferred to the nearest enclosing try-catch block that matches the type of exception thrown.
- If no such match is found, the default exception handler terminates the program.

A

```
public class VotingSystem {

	public static void validate(int age) {
		if (age < 18) {
			throw new ArithmeticException("You are not eligible to vote");
		} else {
			System.out.println("welcome to voting system");
		}
	}

	public static void main(String args[]) {
		validate(13);
		System.out.println("rest of the code...");
	}
}
```

OUTPUT

```
Exception in thread "main" java.lang.ArithmeticException: You are not eligible to vote
    at week16.tutorial.VotingSystem.validate(VotingSystem.java:8)
    at week16.tutorial.VotingSystem.main(VotingSystem.java:15)
```

THROW WITH TRY-CATCH

```
public class VotingSystem {

        public static void validate(int age) {
                if (age < 18) {
                        throw new ArithmeticException("You are not eligible to vote");
                } else {
                        System.out.println("welcome to voting system");
                }
        }

        public static void main(String args[]) {
                 try {
                        validate(15);
                 } catch (ArithmeticException ex){
                        System.out.println(ex.getMessage());
                 }

                System.out.println("rest of the code...");
        }
}
```

OUTPUT

You are not eligible to vote

rest of the code...

PRINTSTACKTRACE

```
public class VotingSystem {

        public static void validate(int age) {
                if (age < 18) {
                        throw new ArithmeticException("You are not eligible to vote");
                } else {
                        System.out.println("welcome to voting system");
                }
        }

        public static void main(String args[]) {
                 try {
                        validate(15);
                 } catch (ArithmeticException ex){
                        ex.printStackTrace();
                 }

                System.out.println("rest of the code...");
        }
}
```

OUTPUT

```
java.lang.ArithmeticException: You are not eligible to vote
    at week16.tutorial.VotingSystem.validate(VotingSystem.java:8)
    at week16.tutorial.VotingSystem.main(VotingSystem.java:16)
rest of the code...
```

THROWS

- The Java throws keyword is used to declare an exception.
- It gives an information to the programmer that there may occur an exception. So, it is better for the programmer to provide the exception handling code so that the normal flow of the program can be maintained.
- We use the throws keyword in the method declaration to declare the type of exceptions that might occur within it.

THROWS

```
package week18;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

class Main {

	public static void findFile() throws IOException {
		File newFile=new File("test.txt");
		FileInputStream stream=new FileInputStream(newFile);
	}

	public static void main(String[] args) {
		try{
				findFile();
		} catch(IOException e){
				System.out.println(e);
		}
	}
}
```

OUTPUT

```
java.io.FileNotFoundException: test.txt (No such file or directory)
```

MULTIPLE THROWS

```
package week18;

import java.io.IOException;
import java.io.InvalidClassException;

class Main {

	public static void findFile() throws NullPointerException, IOException, ArrayIndexOutOfBoundsException {
		// code that may produce NullPointerException
		// code that may produce IOException
		// code that may produce ArrayIndexOutOfBoundsException
	}

	public static void main(String[] args) {
		try{
			findFile();
		} catch (ArrayIndexOutOfBoundsException e1) {
			System.out.println("ArrayIndexOutOfBoundsException occurred");
		} catch (IOException e) {
			System.out.println("IOException occurred");
		} catch (Exception ex) {
			System.out.println("Exception occurred");
		}
	}
}
```

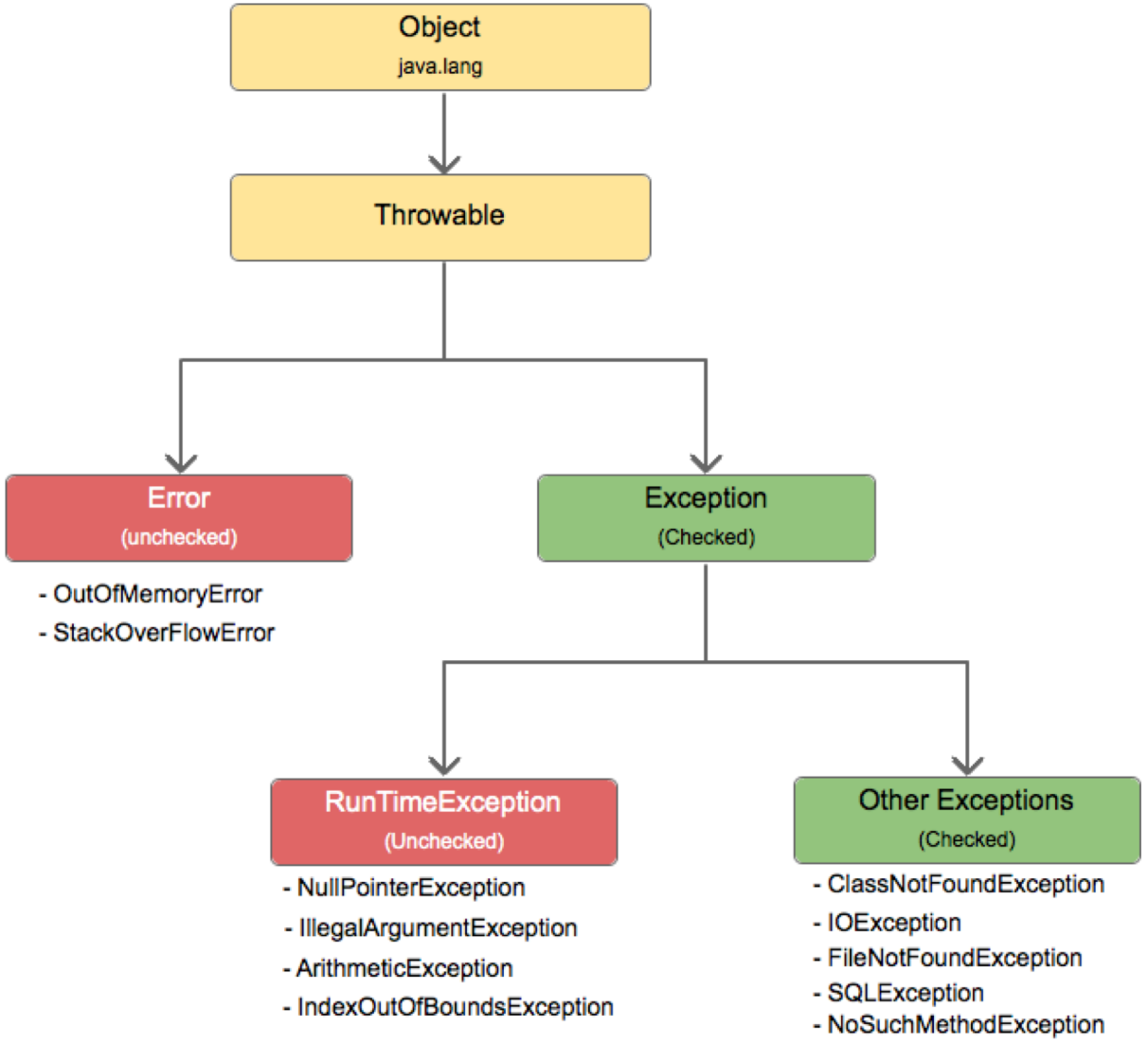
THROWS

Throw	Throws
throw keyword is used to throw an exception explicitly.	throws keyword is used to declare an exception possible during its execution. (Risky methods)
throw keyword is declared inside a method body.	throws keyword is used with method signature (method declaration).
We cannot throw multiple exceptions using throw keyword.	We can declare multiple exceptions (separated by commas) using throws keyword.

ERROR

- The error indicates a problem that mainly occurs due to the lack of system resources and our application should not catch these types of problems.
- Some of the examples of errors are system crash error and out of memory error.
- Programs can not recover from Errors once they occur. Errors will definitely cause termination of the program.

HIERARCHY


Object
java.lang
Throwable
Error
(unchecked)
- OutOfMemoryError
- StackOverFlowError
Exception
(Checked)
RunTimeException
(Unchecked)
- NullPointerException
- IllegalArgumentException
- ArithmeticException
- IndexOutOfBoundsException
Other Exceptions
(Checked)
- ClassNotFoundException
- IOException
- FileNotFoundException
- SQLException
- NoSuchMethodException

ANY QUESTIONS?

Thank you!