Exception Handling

Agenda

- What is Exception and Error
- Types of Exceptions
- Handling Exceptions

Exception & Error

TRAINING CENTER

What is an Exception

An unwanted, unexpected event that disturbs normal flow of the program.

Exception Handling

Exception handling doesn't mean repairing an exception. We must define alternative way to continue rest of the program normally this way of "defining alternative is nothing but exception handling".

Error

Most of the cases errors are not caused by our program these are due to lack of system resources and these are non recoverable.

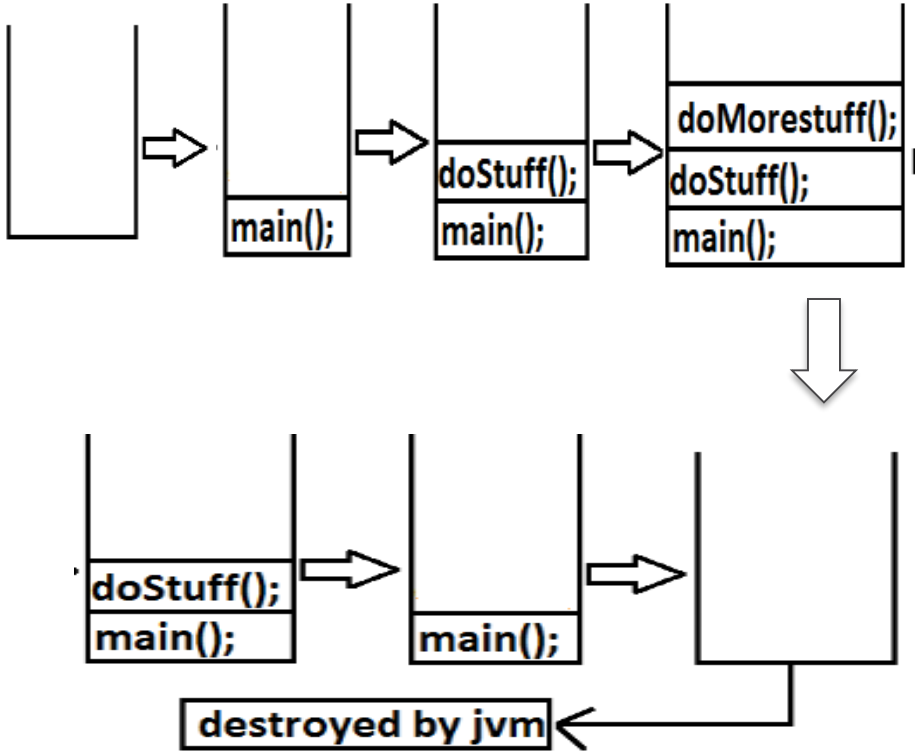
Example

```
try {
read data from London file
 }
catch(FileNotFoundException e) {
 use local file and continue rest of
the program normally
}
}
.
.
.
```

Runtime Stack Mechanism

TRAINING CENTER

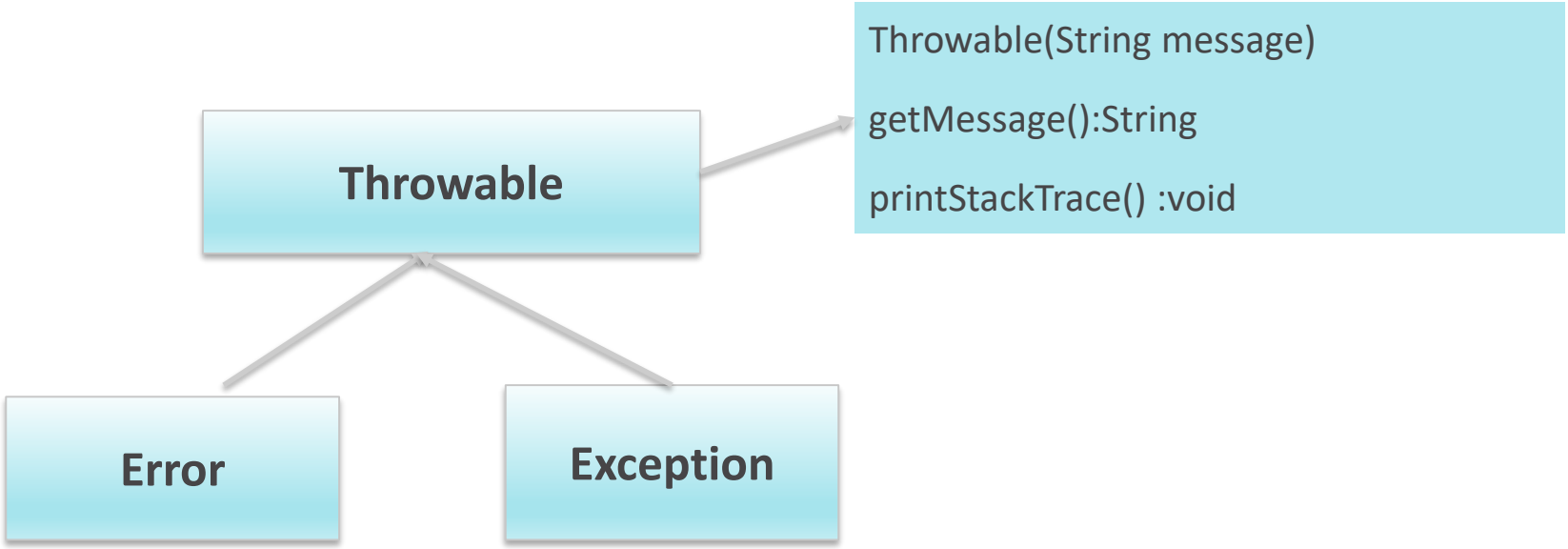
```
Example:
class SampleTest
{
public static void main(String[]
args)
{
 doStuff();
}
public static void doStuff()
{ doMoreStuff();
 }
 public static void doMoreStuff()
{ System.out.println("Hello");
 }
}

Output: Hello
```



Exception Hierarchy

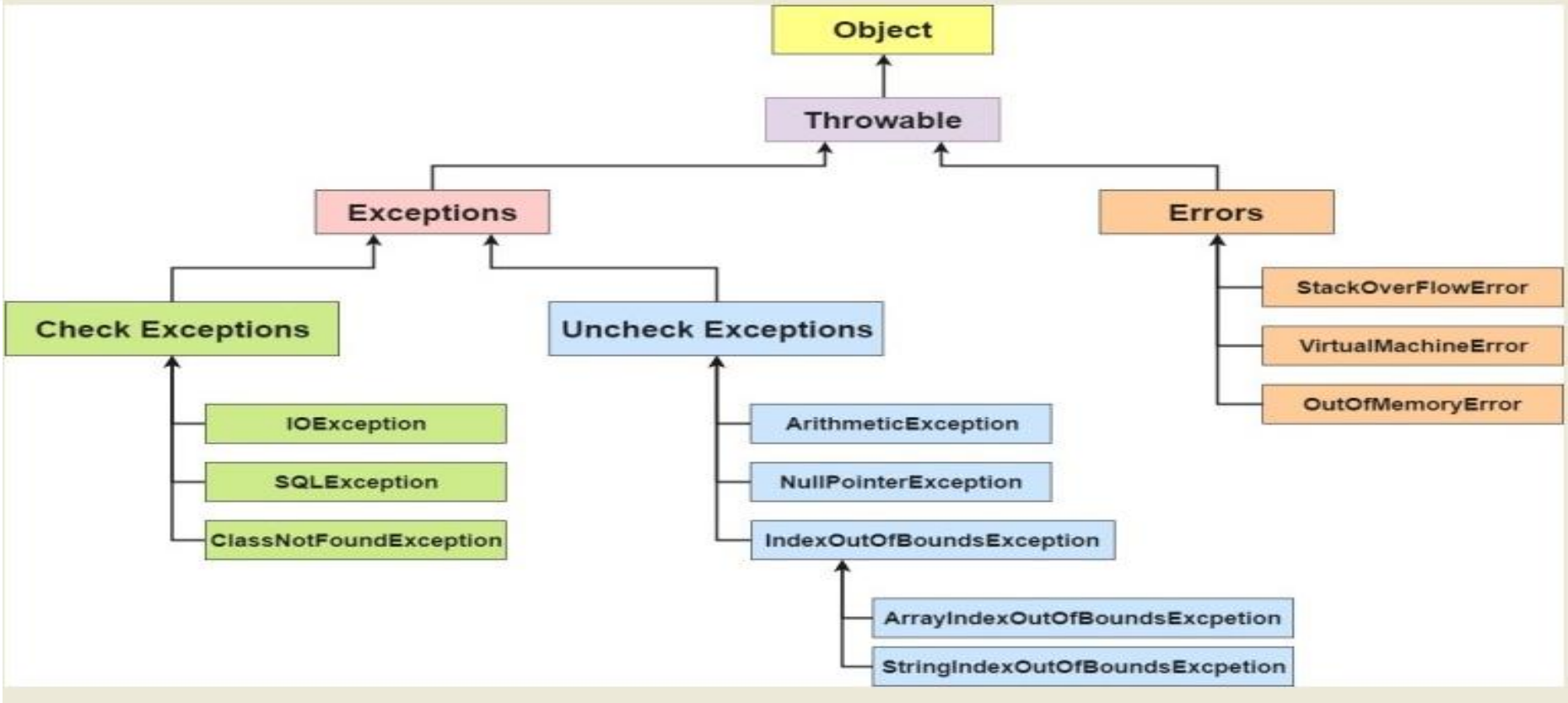
TRAINING CENTER

- The following is the class hierarchy for Java exceptions:



Detailed Exception Hierarchy

TRAINING CENTER



Advantage of Exception Handling

TRAINING CENTER

Without try catch	With try catch

```
class Test {
 public static void main(String[] args)
{
  System.out.println("statement1");
  System.out.println(10/0);
  System.out.println("statement3");
 }
}
output: statement1 RE:AE:/by zero at
Test.main()
Abnormal termination.
```

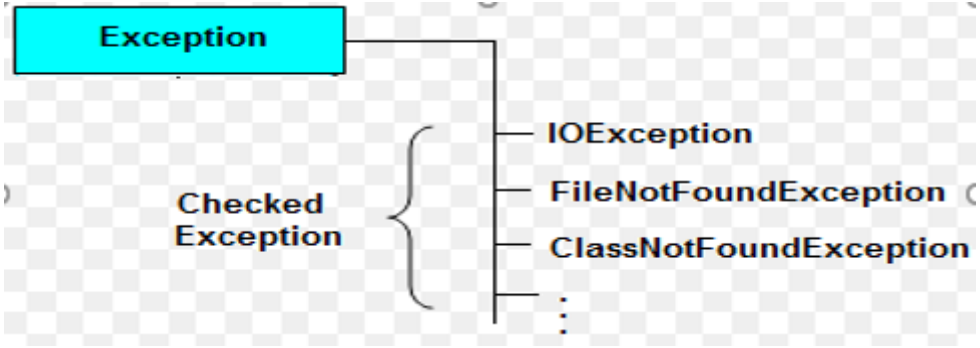
```
class Test{
public static void main(String[] args)
{
  System.out.println("statement1");
try {
  System.out.println(10/0);
} catch(ArithmeticException e)
{
  System.out.println(10/2);
 }
  System.out.println("statement3");
 }
} Output: statement1
  5
  statement3
Normal termination.
```

Exception Handling Keywords

Exception handling keywords:
1.try: To maintain risky code.
2.catch: To maintain handling code.
3.finally: To maintain cleanup code.
4.throw: To handover our created exception object to the JVM manually.
5.throws: To delegate responsibility of exception handling to the caller method.

Checked Exception

Checked: are the exceptions that are checked at compile time. 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.



Example:

The following Java program that opens file at location “C:\test\.txt” and prints the first three lines of it.

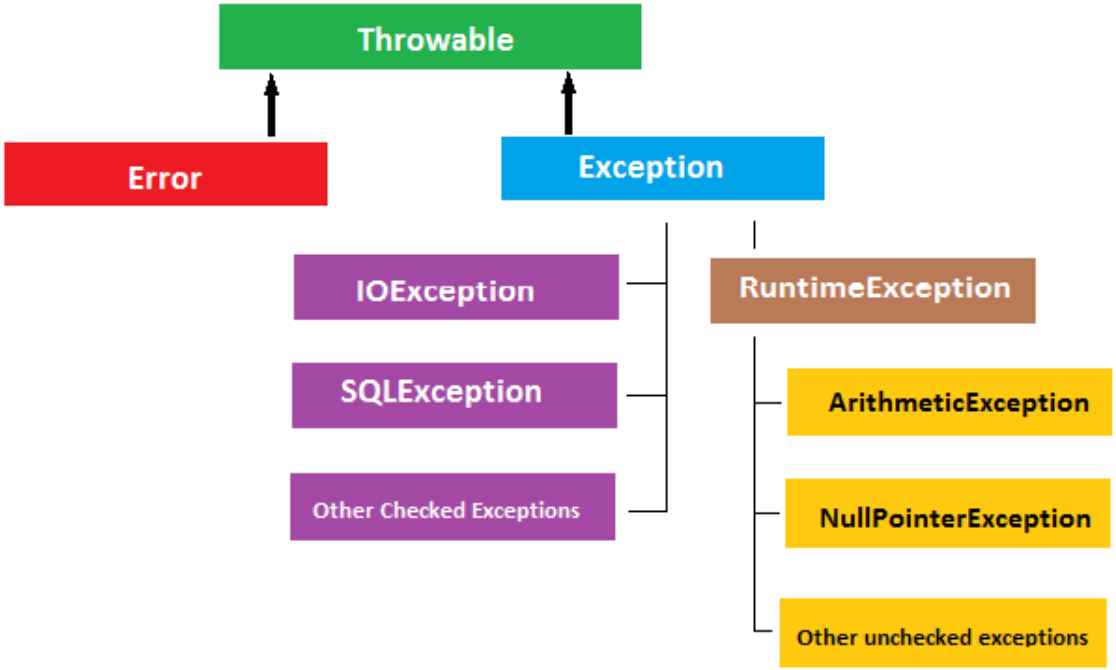
Checked Exception Example

TRAINING CENTER

```
import java.io.BufferedReader;
import java.io.FileReader;
public class TestReadFile {
	public static void main(String[] args) {
		FileReader file=new FileReader("C:\\test\\testData.txt");
		BufferedReader fileReader=new BufferedReader(file);
		for(int counter=0;counter<3;counter++){
			System.out.println(fileReader.readLine());
		}
		fileReader.close();
	}
}
```

Unchecked Exception

Unchecked are the exceptions that are not checked at compiled time.
The classes which inherit RuntimeException are known as unchecked exceptions



UnChecked Exception

```
public class TestUnChecked {
    public static void main(String[] args) {
        try {
            int result=10/0;
        } catch (ArithmeticException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
            System.out.println("Rest of the code");
        }
    }
```

Output: / by zero
Rest of the code
java.lang.ArithmeticException: / by zero
at TestUnChecked.main(TestUnChecked.java:6)

finally block

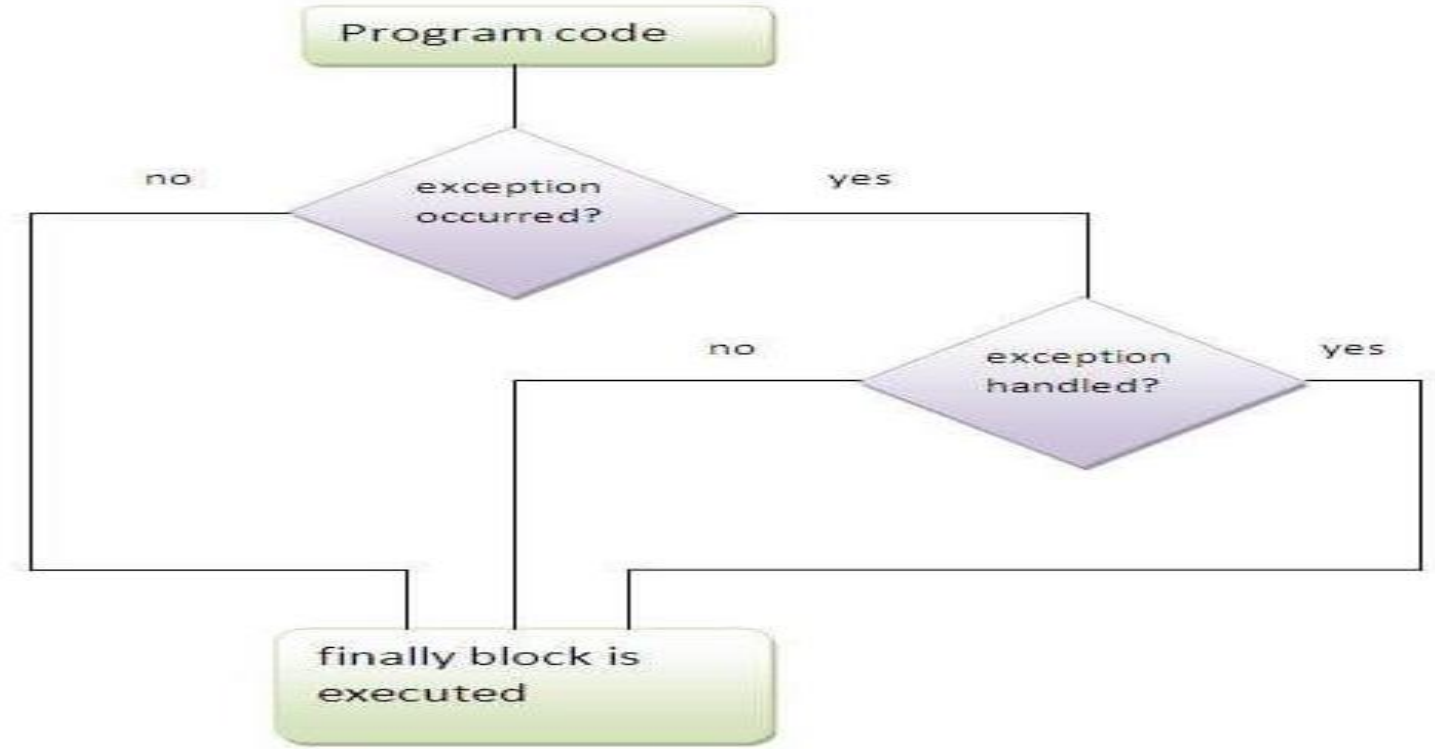
TRAINING CENTER

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.
Java finally block follows try or catch block.

Syntax:

```
finally{

    }
```

finally block



finally block Example

TRAINING CENTER

```
class TestFinallyBlock{
  public static void main(String args[]){
        try{
              int data=25/5;
              System.out.println(data);
              }
        catch(NullPointerException e){System.out.println(e);}
        finally{System.out.println("finally block is always executed");}
        System.out.println("rest of the code...");
  }
}
```

throw block

TRAINING CENTER

Java throw keyword is used to explicitly throw an exception. We can throw either checked or unchecked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception.

Example:

throw exception;

throw new IOException("sorry device error);

```
public class TestThrow1{
    static void validate(int age){
        if(age<18)
            throw new ArithmeticException("not valid");
        else
            System.out.println("welcome to vote");
        }
     public static void main(String args[]){
     validate(13);
    System.out.println("rest of the code...");
  }
}
```

Version Enhancements – 1.7

TRAINING CENTER

Try with resources

```
BufferedReader br=null;
 try{
 br=new BufferedReader(new
FileReader("abc.txt")); //use br
based on our requirements
}
catch(IOException e)
 {
 // handling code
}
finally {
if(br != null
)
br.close();
}
```

Multi Catch Block

```
try{
---------------- ---------------
-
}
 catch(ArithmeticException |
NullPointerException e)
{
e.printStackTrace();
}
catch(ClassCastException |
IOException e)
{
System.out.println(e.getMessage())
;
}
```

Best Practices

TRAINING CENTER

1. Never swallow the exception in catch block

```
catch (NoSuchMethodException e) {
  return null;
}
```

2. Declare the specific checked exceptions that your method can throw

```
public void foo() throws Exception { //Incorrect way
}
```

3. Do not catch the Exception class rather catch specific sub classes

```
try {
  someMethod();
} catch (Exception e) {
  LOGGER.error("method has failed", e);
}
```

4. Always correctly wrap the exceptions in custom exceptions so that stack trace is not lost

```
catch (NoSuchMethodException e) {
  throw new MyServiceException("Some information: " + e.getMessage()); //Incorrect way
}
```

This destroys the stack trace of the original exception and is always wrong. The correct way of doing this is:

```
catch (NoSuchMethodException e) {
  throw new MyServiceException("Some information: " , e); //Correct way
}
```

5. Either log the exception or throw it but never do the both

```
catch (NoSuchMethodException e) {
  LOGGER.error("Some information", e);
  throw e;
}
```

6. Never throw any exception from finally block

```
try {
 someMethod();  //Throws exceptionOne
} finally {
 cleanUp();   //If finally also threw any exception the exceptionOne will be lost forever
}
```

7. Always include all information about an exception in single log message

Don’t do this:

```
LOGGER.debug("Using cache sector A");
LOGGER.debug("Using retry sector B");
```

Do it like this:

```
LOGGER.debug("Using cache sector A, using retry sector B");
```

References

TRAINING CENTER

https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html

Thank You