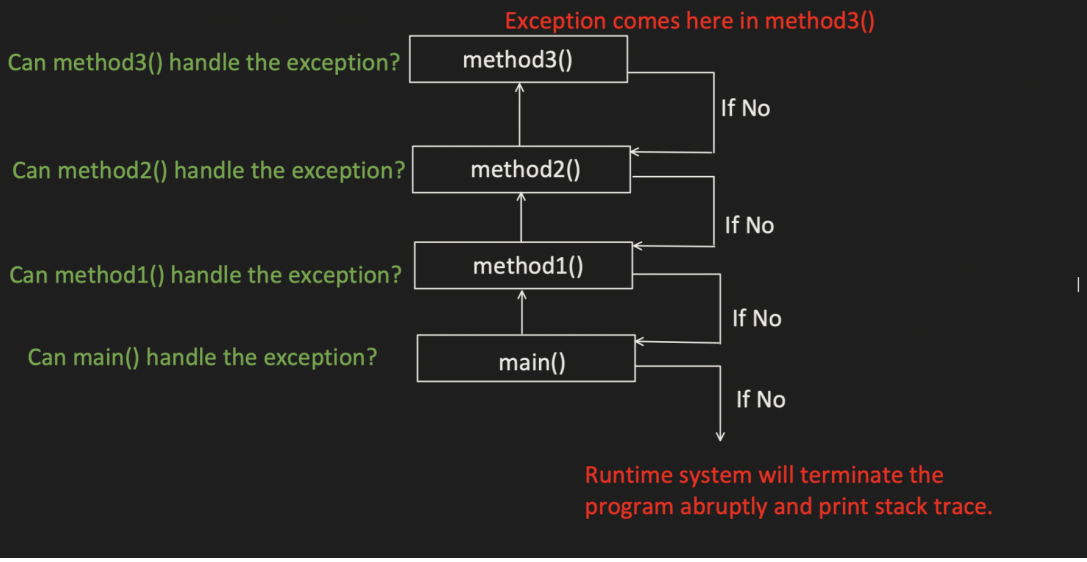
Exception Handling

What is Exception?

- It's an event, that occurs during the execution of the program.
- It will disrupt your program normal flow.
- It Creates the Exception Object, which contain information about the Error like
  - Its Type of Exception and Message
  - Stack trace etc.
- Runtime system use this Exception Object and find the class which can handle it.

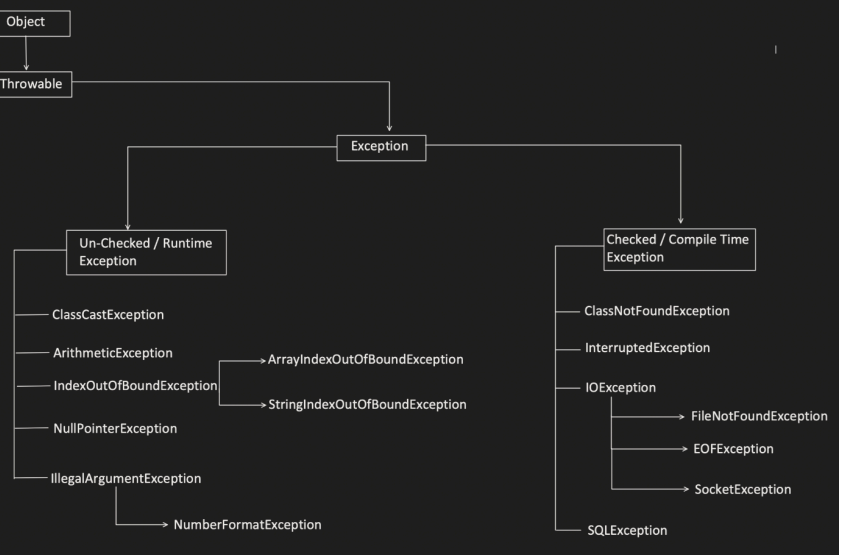



```java
public class Main {

    public static void main(String[] args) {
        Main sampleObj = new Main();
        sampleObj.method1();
    }

    private void method1(){
        method2();
    }

    private void method2(){
        method3();
    }

    private void method3(){
        int b = 5/0;
    }
}
```

Output:

```
Exception in thread "main" java.lang.ArithmeticException Create breakpoint : / by zero
    at Main.method3(Main.java:18)
    at Main.method2(Main.java:14)
    at Main.method1(Main.java:10)
    at Main.main(Main.java:6)
```

Lets Understand the Exception Hierarchy:




```
public class Main {

    public static void main(String[] args) {

        String[] arr = new String[900000000*900000000*900000000];
    }
}
```

Exception in thread "main" java.lang.OutOfMemoryError: Java heap space

Un-Checked / Runtime Exception:

These are the exceptions which occurs during runtime and compiler not forcing us to handle them.

```
public class Main {

    public static void main(String[] args){
            method1();
    }

    public static void method1() {
        throw new ArithmeticException();
    }
}
```

ClassCastException:

```
public class Main {

    public static void main(String[] args) {
        Object val = 0;
        System.out.println((String)val);
    }
}
```

```
Exception in thread "main" java.lang.ClassCastException Create breakpoint : java.lang.Integer cannot be cast to java.lang.String
    at Main.main(Main.java:8)
```

ArithmeticException:

```java
public class Main {

    public static void main(String[] args) {
        int val = 5 / 0;
    }
}
```

```
Exception in thread "main" java.lang.ArithmeticException Create breakpoint : / by zero
    at Main.main(Main.java:5)
```

IndexOutOfBoundException:

```java
public class Main {

    public static void main(String[] args) {
        int[] val = new int[2];
        System.out.println(val[3]);
    }
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException Create breakpoint : 3
    at Main.main(Main.java:6)
```

```java
public class Main {

    public static void main(String[] args) {
        String val = "hello";
        System.out.println(val.charAt(5));
    }
}
```

```
Exception in thread "main" java.lang.StringIndexOutOfBoundsException Create breakpoint : String index out of range: 5
    at java.lang.String.charAt(String.java:658)
    at Main.main(Main.java:6)
```

NullPointerException:

```java
public class Main {

    public static void main(String[] args) {
        String val = null;
        System.out.println(val.charAt(0));
    }
}
```

```
Exception in thread "main" java.lang.NullPointerException Create breakpoint
    at Main.main(Main.java:6)
```

IllegalArgumentException:

```java
public class Main {

    public static void main(String[] args) {
        int val = Integer.parseInt( s: "abc");
    }
}
```

```
Exception in thread "main" java.lang.NumberFormatException Create breakpoint : For input string: "abc"
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
    at java.lang.Integer.parseInt(Integer.java:580)
    at java.lang.Integer.parseInt(Integer.java:615)
    at Main.main(Main.java:5)
```

Checked / Compile time Exception:

Compiler verifies them during the compile time of the code and if not handled properly, code compilation will fail.

```java
public class Main {

    public static void main(String[] args) {
        method1();
    }

    public static void method1(){
        throw new ClassNotFoundException();
    }
}
```

```
Main.java:9: error: unreported exception ClassNotFoundException; must be caught or declared to be thrown
        throw new ClassNotFoundException();
        ^
1 error
```

Lets try to Handle the Exception using "**throws**":

```java
public class Main {

    public static void main(String[] args) {
        method1();
    }

    public static void method1() throws ClassNotFoundException{
        throw new ClassNotFoundException();
    }
}
```

"throws" tells hat, this method MIGHT throw this exception (or might not), so pls caller you handle it appropriately.

Caller class need to then take care

```java
public class Main {

    public static void main(String[] args) throws ClassNotFoundException{
        method1();
    }

    public static void method1() throws ClassNotFoundException{
        throw new ClassNotFoundException();
    }
}
```

Lets try to Handle the Exception using "**try/catch**" block:

```java
public class Main {

    public static void main(String[] args){
        method1();
    }

    public static void method1(){
        try {
            throw new ClassNotFoundException();
        }
        catch (ClassNotFoundException exceptionObject){
            //handle this exception scenario like logging
            exceptionObject.printStackTrace();
        }
    }
}
```

OR

```java
public class Main {

    public static void main(String[] args){
        try {
            method1();
        }
        catch (ClassNotFoundException exceptionObj){
            //handle it
        }
    }

    public static void method1() throws ClassNotFoundException{
        throw new ClassNotFoundException();
    }
}
```

Let's talk about now, how to handle the exception

Using: try, catch, finally, throw, throws

1. Try/Catch:
   - Try block specify the code which can throw exception.
   - Try block is followed either by Catch block or finally block.
   - Catch block is used to catch all the exception which can be thrown in the try block.
   - Multiple catch block can be used.

```java
public class Main {

    public static void main(String[] args){
        try {
            method1( name: "dummy");
        }
        catch (ClassNotFoundException exceptionObject){
            //handle it
        }
        catch (InterruptedException exceptionObject){
            //handle it
        }
        catch (FileNotFoundException exceptionObject){
            //handle this exception
        }
    }

    public static void method1(String name) throws ClassNotFoundException, InterruptedException {
       if(name.equals("dummy")) {
           throw new ClassNotFoundException();
       }
       else if(name.equals("interrupted")) {
            throw new InterruptedException();
        }
    }
}
```

Catch block, can only catch exception which can be thrown by try block.

Catch all Exception Object

```java
public class Main {

    public static void main(String[] args){
        try {
            method1( name: "dummy");
        }
        catch (ClassNotFoundException exp){

        }
        catch (Exception exp){

        }
    }

    public static void method1(String name) throws ClassNotFoundException, InterruptedException {
        if(name.equals("dummy")) {
            throw new ClassNotFoundException();
        }
        else if(name.equals("interrupted")) {
            throw new InterruptedException();
        }
    }
}
```

```java
public class Main {

    public static void main(String[] args){
        try {
            method1( name: "dummy");
        }
        catch (Exception exp){

        }
        catch (ClassNotFoundException exp){

        }
    }

    public static void method1(String name) throws ClassNotFoundException, InterruptedException {
        if(name.equals("dummy")) {
            throw new ClassNotFoundException();
        }
        else if(name.equals("interrupted")) {
            throw new InterruptedException();
        }
    }
}
```

Already Caught above

Catch Multiple Exceptions in One Catch Block

```java
public class Main {

    public static void main(String[] args){
        try {
            method1( name: "dummy");
        }
        catch (ClassNotFoundException | InterruptedException exp){

        }
        catch (Exception exp){

        }
    }

    public static void method1(String name) throws ClassNotFoundException, InterruptedException {
        if(name.equals("dummy")) {
            throw new ClassNotFoundException();
        }
        else if(name.equals("interrupted")) {
            throw new InterruptedException();
        }
    }
}
```

2. Try/catch/finally Or try/finally block

- Finally block can be use after try or after catch block.
- Finally block will always get executed, either if you just return from try block or from catch block.
- At most, we can add only 1 finally block.
- Mostly used for closing the object, adding logs etc.
- If JVM related issues like out of memory, system shut down or our process is forcefully killed. Then finally block do not get executed.

```java
public class Main {

    public static void main(String[] args){
        try {
            method1( name: "dummy");
        }
        catch (ClassNotFoundException exp){
            //handle the exception here.
        }
        finally {
            //do something there
        }

    }

    public static void method1(String name) throws ClassNotFoundException {
        if(name.equals("dummy")) {
            throw new ClassNotFoundException();
        }
    }
}
```

or

```java
public class Main {

    public static void main(String[] args) throws ClassNotFoundException{
        try {
            method1( name: "dummy");
        }
        finally {
            //do something there
        }

    }

    public static void method1(String name) throws ClassNotFoundException {
        if(name.equals("dummy")) {
            throw new ClassNotFoundException();
        }
    }
}
```

or

```java
public class Main {

    public static void main(String[] args){
        try {
            method1( name: "dummy");
        }
        finally {
            //do something there
        }

    }

    public static void method1(String name) throws ArithmeticException {
        if(name.equals("dummy")) {
            throw new ArithmeticException();
        }
    }
}
```

```java
public class Main {

    public static void main(String[] args){
        try {
            method1( name: "dummy1");
            return;
        }
        finally {
            System.out.println("inside finally");
        }

    }

    public static void method1(String name)  {

    }
}
```

Output:

```
inside finally

Process finished with exit code 0
```

3. **Throw:**
   - **It is used to throw a new exception or**
   - **To re-throw the exception.**

```java
public class Main {

    public static void main(String[] args) throws ClassNotFoundException {

        try {
            method1();
        }
        catch (ClassNotFoundException e){
            throw e;
        }
    }

    public static void method1() throws ClassNotFoundException{

        throw new ClassNotFoundException();
    }
}
```

Creating custom/ user-defined Exception class

```java
public class MyCustomException extends Exception {

    MyCustomException(String message) {
        super(message);
    }
}
```

```java
public class Main {

    public static void main(String[] args) {

        try {
            method1();
        }
        catch (MyCustomException e){
            //handle it
        }
    }

    public static void method1() throws MyCustomException{

        throw new MyCustomException("some issue arise");
    }
}
```

At last, lets ask ourself, why we need to handle the Exception:

- It makes our code clean by separating the error handling code from regular code.
- It allows program to recover from the error.
- It allow us to add more information, which support debugging
- Improves security, by hiding the sensitive information

```
public void myMethod(int schoolClassNumber) {

    int noOfStudents = getStudentCapacityofClass(schoolClassNumber);
    String[] names = new String[noOfStudents];
    names[0] = "new value";
}
```

Without Exception handling, we have to do this:
notice 2 things : readability and error code need to be returned

```java
public  int myMethod(int schoolClassNumber) {

    int errorCode = 0; // 0 means no success

    if (schoolClassNumber > 0 && schoolClassNumber <= 12) {
        int noOfStudents = getStudentCapacityofClass(schoolClassNumber);
        if (noOfStudents != 0) {
            String[] names = new String[noOfStudents];
            if (names != null && names.length > 0) {
                names[0] = "new value";
            } else {
                return -3;
            }
        } else {
            return -2;
        }
    } else {
        errorCode = -1;
    }
    return errorCode;
}
```

With Exception handling:

```
public void myMethod(int schoolClassNumber) {

    try {
        int noOfStudents = getStudentCapacityofClass(schoolClassNumber);
        String[] names = new String[noOfStudents];
        names[0] = "new value";
    }
    catch (IndexOutOfBoundsException expObj){
        //do something
    }
    catch (Exception expObj){
        //do something
    }
}
```

But remember, exception handling is little expensive, if stack trance is huge and it is not handled or handled at parent class.

```
public int myMethod(int a, int b) {
    int val;
    try{
         val = a/b;
    } catch (ArithmeticException exp){
        val = -1;
    }
    return val;
}

public int myMethod2(int a, int b) {

        if(b == 0){
            return -1;
        }

        int val = a/b;
        return val;
}
```

Try to avoid using exception handling, if you can. Like in this example, its not required