Exception Handling.

Object — Parent class of all Java Classes

↓

Throwable — class is the parent class of all the exception classes.

Exception	Error
→ due to programs	→ due to lack of sys. resource.
→ Recoverable	→ Not recoverable
→ Types	→ out of memory error
1) Compile time	Type : 1) Run Time Exception.
2) Run Time	2) IOException (checked exception)

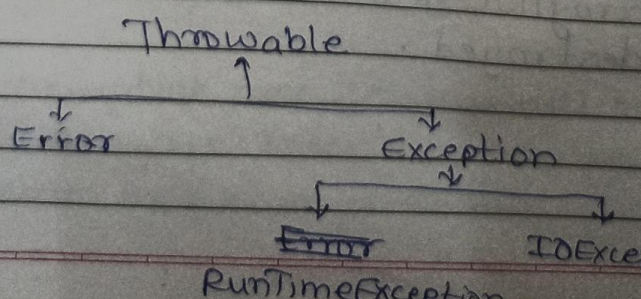
* Exception

An exception is an unexpected event that occurs during program execution. It affects the flow of the program instructions which can cause the program to terminate abnormally / end.

error and exceptions:

→ Can occur like these reasons

- Invalid user input.
- Device failure.
- Loss of network connection.
- By Physical limitations (out of disk memory)
- Code errors.
- Opening an unavalible file.



* Errors :

Errors represent irrecoverable conditions such as JVM running out of memory, memory leaks, stack overflow errors, library incompatibility, infinite recursion, etc.

* Exceptions
→ Exceptions can be caught and handled by the program.
→ When exception occurs within a method it creates an object called exception object.
→ It contains the information about the exception such as the name and description of the exception and state of the program when the exception occured.

* Types of exception :-

1) Runtime Exception.
→ A runtime exception happens due to programming error. They are also known as unchecked exceptions.

→ These exceptions are not checked at ~~runtin~~ compile time but run time.
* Some of common runtime exceptions are
- Improper use of an API - IllegalArgumentException
- Null pointer access exception - NullPointerException.
- Out-of-bounds array access - ArrayIndexOutOfBoundsException

multiple ~~try~~ catch is possible but there should be respective try

- Dividing a number by 0 - ArithmeticException

"If it is a runtime exception, it is your fault."

- ArrayIndexOutOfBondsException would not have occured if you tested the array index against the array bounds.

2. IOException.

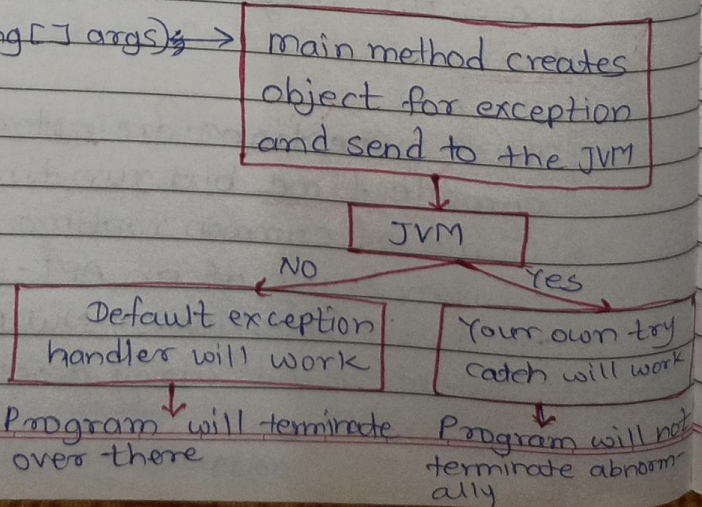
→ An IOException is also known as checked exception.

→ They are checked by the compiler at the compile-time and the programmer is prompted to handle these exceptions.

Like

- Trying to open file that dosen't exist results in FileNotFoundException.

```
class A
{
    p s v main (String [] args) ~~{~~
    ~~}~~
```



final keyword

final class → can't inherited
final method → can't be overridden
final variable → value can't changes

DATE
PAGE NO.

* List of different approaches to handle exceptions in Java
→ throw and throws keyword
→ try .... catch block.
→ finally block.

** try .... catch block.

→ used to handle exceptions and termination of program.

```
try {
    //code
}
catch(excetion){
    //code
}
```

Generally written as Exception e

! we can use perticular exception name also like ArithmeticException

△ try :- code that might generate an exception.

△ catch : code that is executed when there occurs an exception inside the try block.

```
class Main {
    psvm(String[] args){

        try {
            int divideByZero = 5/0;
            Sout("Rest of code in try block");
        }
```

```
catch(Exception e){
    sout(e.getMessage());
}
}
}
```

O/P : / by zero.

Important

only try or only catch block can't be used we can use try block along with finally block but we can't use catch block without try block.

** try.....finally block.

→ we can also use try block along with finally block.
→ Only finally can't be used.
→ finally block is always executed whether there is an exception in try block or not.

try finally
→ If there is exception then first finally will execute then exception shown.

try{ }
catch{ }
finally{ }
→ If exception is handled or there is no exception it will execute after the given try catch.

Note: Cases when finally block does not execute
- use of System.Exit() method.
- An exception occurs in the finally block.
- The death of a thread.

Ex.
```
class Main{
    psvm(String[] args){
        try{
            int a = 5/0;
        }
        finally{
            sout("Always executed");
        }
    }
}
```

O/P → Always executed
Exception info will printed here.

Catching multiple exceptions.

→ From Java SE7 and later, we can now catch more than one type of exception with one catch block.

→ reduces code duplicity and increases code simplicity and effeciency.

→ Each exception type that can be handled by the catch block is separated using a verticle bar |

Ex.
```
try{
    // code
}
catch(ExceptionType1 | ExceptionType2 ex){
    // catch block.
}
```

Multiple catch blocks.

```
try{
   //code
}
catch(Exception1 e1){
 // code
}
catch(Exception2 e2){
  //code
}
```

these must be different.

** Throw and throws keywords.

i> throws

We use the **throws** keyword in method declaration to declare the type of exceptions that might occur within it.

Syntax:

```
accessModifier returnType methodName() throws ExceptionType1, ...{
    // code
}
```

we can declare multiple exceptions at one time.

Ex:

```
public static void findFile() throws IOException{
    //code that may produce IOException
    File newFile = new File("test.txt");
    FileInputStream stream = new FileInputStream(newFile);
}
```

O/P

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

> If the file test.txt does not exist, FileInputStream throws a FileNotFoundException which extends the IOException class.

> If a method does not handle exceptions, the type of exceptions that may occur within it must be specified in the throws clause so that method further up in the call stack can handle them or specify them using throws keyword themselves.

> The findFile() method specifies that an IOException can be thrown. The main() method calls this method and handles the exception if it is thrown.

ii) throw keyword.

> The throw keyword is used to explicitly throw a single exception.

> When an exception is thrown, the flow of program execution transfers from the try block to catch block. We use the throw keyword within a method.

Syntax:

```
throw throwableObject;
```

↳ It is an instance of class Throwable or subclass of the Throwable class.

Ex.
```
public static void divideByZero(){
throw new ArithmeticException("Trying to divide by 0");
}
```