What is an exception?

An Exception is an unwanted event that interrupts the normal flow of the program. When an exception occurs program execution gets terminated. In such cases we get a system generated error message. The good thing about exceptions is that they can be handled in Java. By handling the exceptions we can provide a meaningful message to the user about the issue rather than a system generated message, which may not be understandable to a user.

Why an exception occurs?

There can be several reasons that can cause a program to throw exception. For example: Opening a non-existing file in your program, Network connection problem, bad input data provided by user etc.

Exception Handling

If an exception occurs, which has not been handled by programmer then program execution gets terminated and a system generated error message is shown to the user

Advantage of exception handling

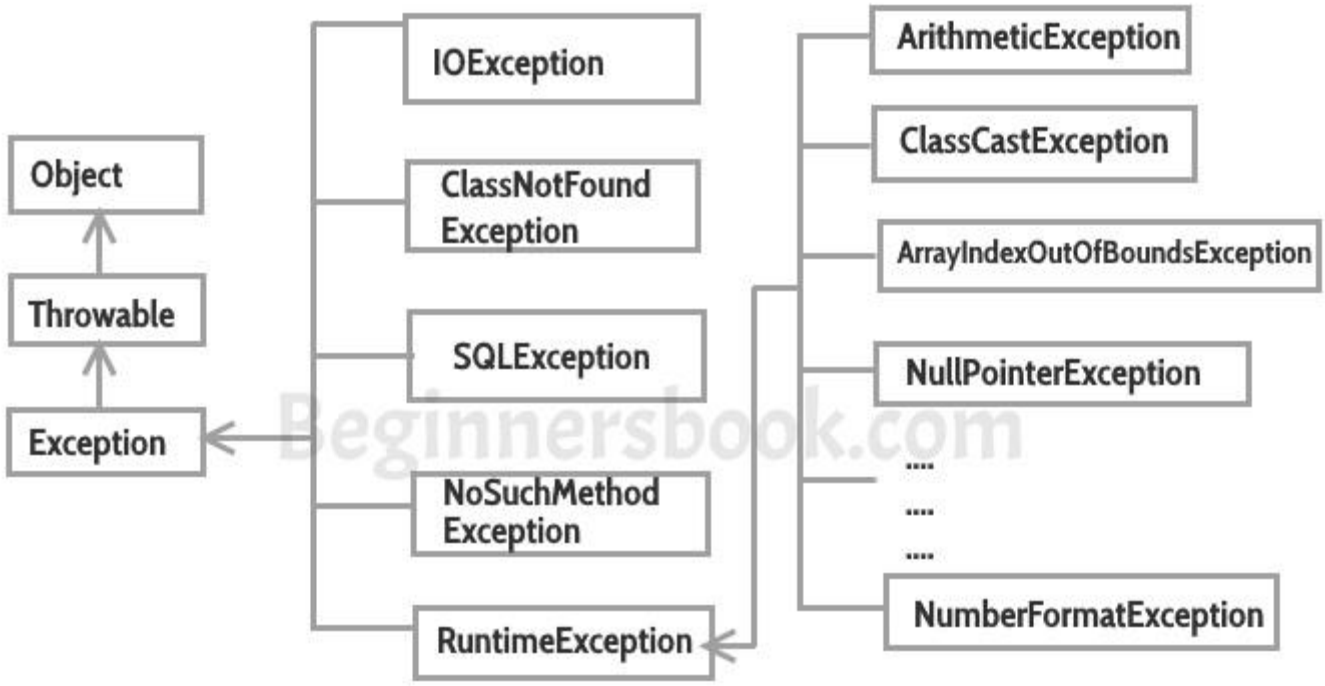
Exception handling ensures that the flow of the program doesn't break when an exception occurs. For example, if a program has bunch of statements and an exception occurs mid way after executing certain statements then the statements after the exception will not execute and the program will terminate abruptly.
By handling we make sure that all the statements execute and the flow of program doesn't break.

Difference between error and exception

Errors indicate that something severe enough has gone wrong, the application should crash rather than try to handle the error.

Exceptions are events that occurs in the code. A programmer can handle such conditions and take necessary corrective actions. Few examples: NullPointerException – When you try to use a reference that points to null. ArithmeticException – When bad data is provided by user, for example, when you try to divide a number by zero this exception occurs because dividing a number by zero is undefined. ArrayIndexOutOfBoundsException – When you try to access the elements of an array out of its

bounds, for example array size is 5 (which means it has five elements) and you are trying to access the 10th element.



Types of exceptions

There are two types of exceptions in Java:

1)Checked exceptions

2)Unchecked exceptions

Checked exceptions

All exceptions other than Runtime Exceptions are known as Checked exceptions as the compiler checks them during compilation to see whether the programmer has handled them or not. If these exceptions are not handled/declared in the program, you will get compilation error. For example, SQLException, IOException, ClassNotFoundException etc.

Unchecked Exceptions

Runtime Exceptions are also known as Unchecked Exceptions. These exceptions are not checked at compile-time so compiler does not check whether the programmer has handled them or not but it's the responsibility of the programmer to handle these exceptions and provide a safe exit. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.

Try Catch Block

Java provides an inbuilt exceptional handling.

The normal code goes into a TRY block.

The exception handling code goes into the CATCH block

Syntax for using try & catch

```
try{
    statement(s)
}
catch (exceptiontype name){
        statement(s)
}
```

```
class JavaException {
 public static void main(String args[]) {
  int d = 0;
  int n = 20;
  try {
   int fraction = n / d;
   System.out.println("This line will not be Executed");
  } catch (ArithmeticException e) {
   System.out.println("In the catch Block due to Exception = " + e);
  }
  System.out.println("End Of Main");
 }
}
```

```
C:\workspace>java JavaException
In the catch clock due to Exception = java.lang.ArithmeticException: / by zero
End Of Main
```

Java finally block

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.

```
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...");
  }
}
```

Output:5

```
finally block is always executed
rest of the code...
```

Difference between throw and throws in Java

There are many differences between throw and throws keywords. A list of differences between throw and throws are given below:

No.	throw	throws
1)	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2)	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3)	Throw is followed by an instance.	Throws is followed by class.
4)	Throw is used within the method.	Throws is used with the method signature.
5)	You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException

Java throw example

```
void m(){
throw new ArithmeticException("sorry");
}
```

Java throws example

```
void m()throws ArithmeticException{
//method code
}
```

Difference between final, finally and finalize

There are many differences between final, finally and finalize. A list of differences between final, finally and finalize are given below:

No.	final	finally	finalize
1)	Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.	Finally is used to place important code, it will be executed whether exception is handled or not.	Finalize is used to perform clean up processing just before object is garbage collected.
2)	Final is a keyword.	Finally is a block.	Finalize is a method.

Java final example

```
class FinalExample{
public static void main(String[] args){
final int x=100;
x=200;//Compile Time Error
}}
```

Java finally example

```
class FinallyExample{
public static void main(String[] args){
try{
int x=300;
}catch(Exception e){System.out.println(e);}
finally{System.out.println("finally block is executed");}
}}
```

Java finalize example

```
class FinalizeExample{
public void finalize(){System.out.println("finalize called");}
public static void main(String[] args){
FinalizeExample f1=new FinalizeExample();
FinalizeExample f2=new FinalizeExample();
f1=null;
f2=null;
System.gc();
}}
```

Java Custom Exception

If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

By the help of custom exception, you can have your own exception and message.

Let's see a simple example of java custom exception.

```
class InvalidAgeException extends Exception{
 InvalidAgeException(String s){
  super(s);
 }
}
class TestCustomException1{
   static void validate(int age)throws InvalidAgeException{
     if(age<18)
```

```
      throw new InvalidAgeException("not valid");
     else
      System.out.println("welcome to vote");
   }
   public static void main(String args[]){
      try{
      validate(13);
      }catch(Exception m){System.out.println("Exception occured: "+m);}
      System.out.println("rest of the code...");
  }
}
```

Output:Exception occured: InvalidAgeException:not valid

rest of the code...