UNIT-2 NOTES

OF

OOPS

What is Exception in Java?

Dictionary Meaning: Exception is an abnormal condition.

In Java, an exception is an event that disrupts the normal flow of the program. It

is an object which is thrown at runtime.

In Java, Exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program's instructions. Exceptions can be caught and handled by the program. When an exception occurs within a method, it creates an object. This object is called the exception object. It contains information about the exception, such as the name and description of the exception and the state of the program when the exception occurred.

Major reasons why an exception Occurs

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

Errors represent irrecoverable conditions such as Java virtual machine (JVM) running out of memory, memory leaks, stack overflow errors, library incompatibility, infinite recursion, etc. Errors are usually beyond the control of the programmer, and we should not try to handle errors.

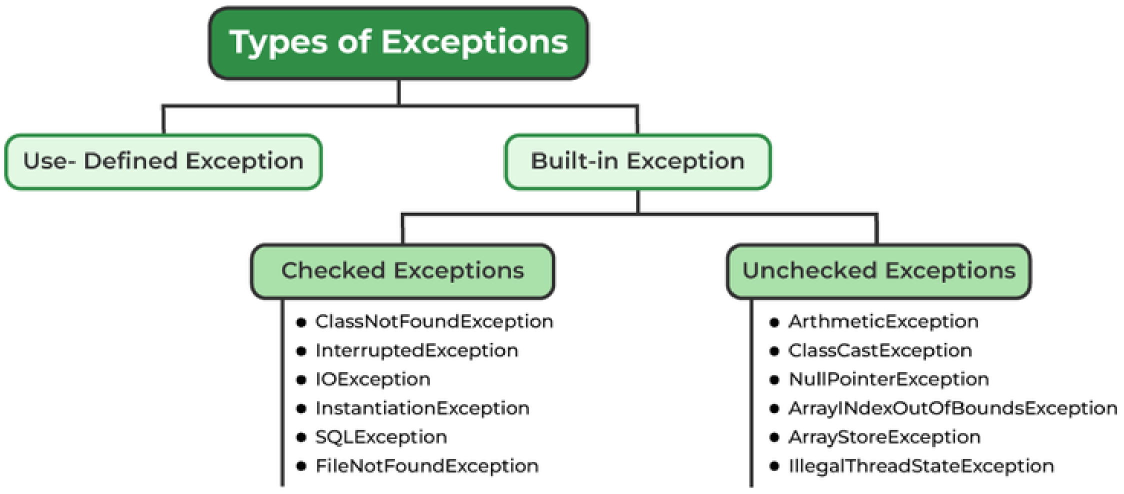
Difference between Error and Exception

Let us discuss the most important part which is the **differences between Error and Exception** that is as follows:

- **Error:** An Error indicates a serious problem that a reasonable application should not try to catch.
- **Exception:** Exception indicates conditions that a reasonable application might try to catch.

Types of Exceptions

Java defines several types of exceptions that relate to its various class libraries. Java also allows users to define their own exceptions.



Exceptions can be categorized in two ways:

1. **Built-in Exceptions**
   - Checked Exception
   - Unchecked Exception
2. **User-Defined Exceptions**

Let us discuss the above-defined listed exception that is as follows:

1. Built-in Exceptions

Built-in exceptions are the exceptions that are available in Java libraries. These exceptions are suitable to explain certain error situations.

- **Checked Exceptions:** Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler.

- **Unchecked Exceptions:** The unchecked exceptions are just opposite to the checked exceptions. The compiler will not check these exceptions at compile time. In simple words, if a program throws an unchecked exception, and even if we didn't handle or declare it, the program would not give a compilation error.

Note: *For checked vs unchecked exception, see Checked vs Unchecked Exceptions*

2. User-Defined Exceptions:

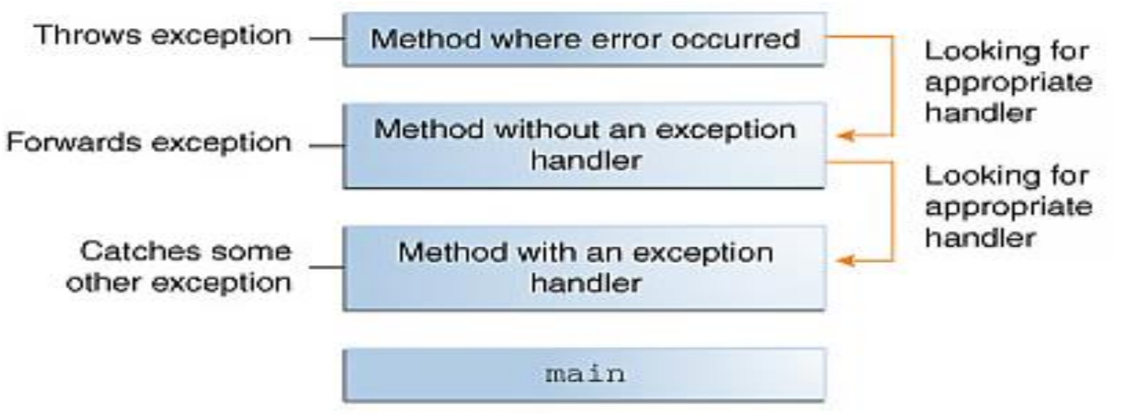
Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, users can also create exceptions, which are called 'user-defined Exceptions'.

Why does an exception occur?

The key point here is *possibility*. There are multiple things that could go wrong. Network state, user interaction, invalid inputs, etc., are some of the things which could lead to an exception. What's important is how you handle those exceptions which you think will occur sometime in some way or the other.

Exception Handling by JVM

When an exception occurs inside a java program, an object is created which describes the exception, including its type and state of the program when it occurred. This object is then passed to the Java Virtual Machine (JVM). The JVM tries to find a method or function to handle the exception. It goes through a list of methods that can potentially handle the exception.


Throws exception
Method where error occurred
Looking for appropriate handler
Forwards exception
Method without an exception handler
Looking for appropriate handler
Catches some other exception
Method with an exception handler
main

Once it finds the appropriate method to handle the exception, it hands over the exception to the method, but if it can't find an appropriate method, it terminates the program by calling the default exception-handling method.

Ways to Handle an Exception

Here are some ways in which you can handle or throw an exception:

- try… catch block
- throw/throws keyword

Java Exception Keywords

Java provides five keywords that are used to handle the exception. The following table describes each.

Keyword	Description
Try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Java Exception Handling Example

Let's see an example of Java Exception Handling in which we are using a try-catch statement to handle the exception.

JavaExceptionExample.java

```java
public class JavaExceptionExample{
  public static void main(String args[]){
   try{
      //code that may raise exception
      int data=100/0;
   }catch(ArithmeticException e){
System.out.println(e);
}
   //rest code of the program
   System.out.println("rest of the code...");
  }
}
```

OUTPUT

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

In the above example, 100/0 raises an ArithmeticException which is handled by a try-catch block.

Java Multi-catch block

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.

- At a time only one exception occurs and at a time only one catch block is executed.
- All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

Let's see a simple example of java multi-catch block.

MultipleCatchBlock1.java

```
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");
   }
```

Java finally block

Java finally block is a block used to execute important code such as closing the connection, etc.

Java finally block is always executed whether an exception is handled or not. Therefore, it contains all the necessary statements that need to be printed regardless of the exception occurs or not.

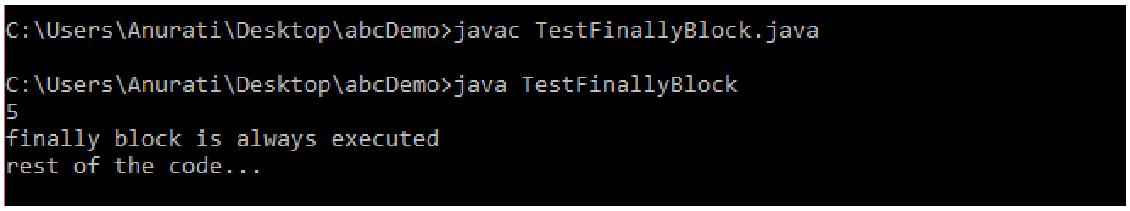
The finally block follows the try-catch block.

- finally block in Java can be used to put "**cleanup**" code such as closing a file, closing connection, etc.
- The important statements to be printed can be placed in the finally block.

TestFinallyBlock.java

```
class TestFinallyBlock {
  public static void main(String args[]){
  try{
//below code do not throw any exception
   int data=25/5;
   System.out.println(data);
  }
//catch won't be executed
  catch(NullPointerException e){
System.out.println(e);
}
//executed regardless of exception occurred or not
 finally {
System.out.println("finally block is always executed");
}

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

Output:



```
C:\Users\Anurati\Desktop\abcDemo>javac TestFinallyBlock.java

C:\Users\Anurati\Desktop\abcDemo>java TestFinallyBlock
5
finally block is always executed
rest of the code...
```

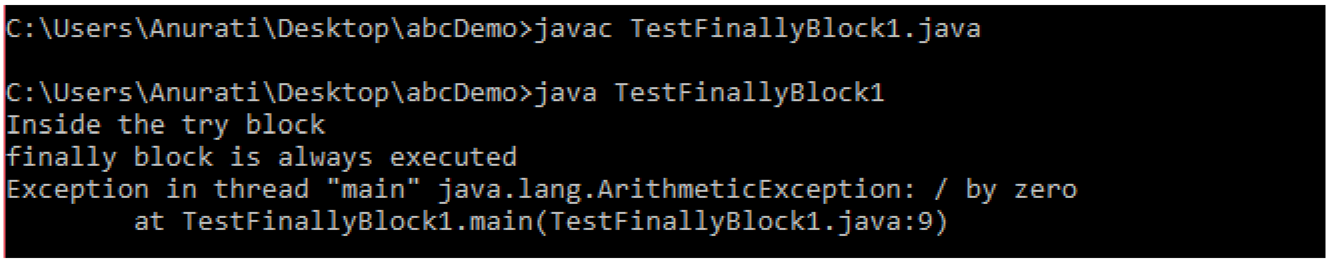
When an exception occurr but not handled by the catch block

Let's see the the fillowing example. Here, the code throws an exception however the catch block cannot handle it. Despite this, the finally block is executed after the try block and then the program terminates abnormally.

TestFinallyBlock1.java

```
public class TestFinallyBlock1{
      public static void main(String args[]){

      try {

        System.out.println("Inside the try block");

        //below code throws divide by zero exception
       int data=25/0;
       System.out.println(data);
      }
      //cannot handle Arithmetic type exception
      //can only accept Null Pointer type exception
      catch(NullPointerException e){
        System.out.println(e);
      }

      //executes regardless of exception occured or not
      finally {
        System.out.println("finally block is always executed");
      }
        System.out.println ("rest of the code...");
      }
    }
```

Output:



```
C:\Users\Anurati\Desktop\abcDemo>javac TestFinallyBlock1.java

C:\Users\Anurati\Desktop\abcDemo>java TestFinallyBlock1
Inside the try block
finally block is always executed
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at TestFinallyBlock1.main(TestFinallyBlock1.java:9)
```

Java throw keyword

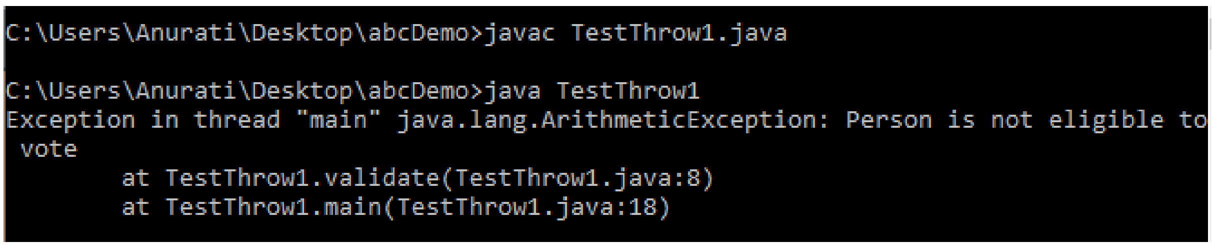
The Java throw keyword is used to throw an exception explicitly.

We specify the **exception** object which is to be thrown. The Exception has some message with it that provides the error description. These exceptions may be related to user inputs, server, etc.

We can throw either checked or unchecked exceptions in Java by throw keyword. It is mainly used to throw a custom exception. We will discuss custom exceptions later in this section.

```java
public class TestThrow1 {
    //function to check if person is eligible to vote or not
    public static void validate(int age) {
        if(age<18) {
            //throw Arithmetic exception if not eligible to vote
            throw new ArithmeticException("Person is not eligible to vote");
        }
        else {
            System.out.println("Person is eligible to vote!!");
        }
    }
    //main method
    public static void main(String args[]){
        //calling the function
        validate(13);
        System.out.println("rest of the code...");
  }
}
```

Output:



```
C:\Users\Anurati\Desktop\abcDemo>javac TestThrow1.java

C:\Users\Anurati\Desktop\abcDemo>java TestThrow1
Exception in thread "main" java.lang.ArithmeticException: Person is not eligible to
 vote
        at TestThrow1.validate(TestThrow1.java:8)
        at TestThrow1.main(TestThrow1.java:18)
```

Java throws keyword

Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code.

```
public class TestThrows {


   public static int divideNum(int m, int n) throws ArithmeticException {
      int div = m / n;
      return div;
   }
      public static void main(String[] args) {
      TestThrows obj = new TestThrows();
      try {
         System.out.println(obj.divideNum(45, 0));
      }
      catch (ArithmeticException e){
         System.out.println("\nNumber cannot be divided by 0");
      }
             System.out.println("Rest of the code..");
   }
}
```

1->This is the basic example of Exception handling,here exception is caught by catch. And other method is executed as it is.

```
class Ex

{

void add()

{

int a=32,b=4;

System.out.println(a+b);

}

public static void main(String args[])

{

try

{

int i=6;

int j=i/0;

}

catch(ArithmeticException ex)

{  System.out.println("Exception arise is"+" "+ex);

}

Ex e=new Ex();

e.add();

System.out.println("rest of application code");

}

}
```

2->This is the example of all exception classes are inside of catch block sequence is matter sub classes and at last super class is there. Exception is super class and rest of are sub classes.When exception occurs inside of try block then all line are skipped of try block after exception caught then control goes to catch block.

```
class test1
{
public static void main(String args[])
{
try
{
int i=6;
int j=i/0;
String s=null;
System.out.println(j);
 System.out.println(s.length());
}
catch(ArithmeticException ex)
{
System.out.println("Exception arises"+ex);
}
catch(NullPointerException ex)
{
System.out.println("Exception arises"+ex);
}
catch(ArrayIndexOutOfBoundsException ex)
{
System.out.println("Exception arises"+ex);
}

catch(Exception ex)

{

System.out.println("Exception arises"+ex);

}

System.out.println("rest of code");

}

}
```

4-> this is the example of nested try catch block.

```
class test5

{

public static void main(String args[])

{

try

{

int i=6;

String x=null;

System.out.println(x.length());

try

{


int j=i/0;

System.out.println(j);

}

catch(ArithmeticException ex)

{

System.out.println("Exception"+ex);

}

System.out.println("hiii");

}

catch(NullPointerException ex)

{

System.out.println("Exception"+ex);

}

System.out.println("nested try catch block");
}
}
```

3-> In this example reverse order of exception classes is not applicable, error occurs bcoz we write Exception class(Parent class firstly).

```
class test
{
public static void main(String args[])
{
int a=Integer.parseInt(args[0]);
int b=Integer.parseInt(args[1]);
try
{
int c=a/b;
String x=null;
System.out.println(x.length());
System.out.println(c);
System.out.println("no exception");
}
catch(Exception ex)

{

System.out.println("Exception arise is"+ex);

}

catch(ArithmeticException ex)

{

System.out.println("Exception arise is"+ex);

}

catch(NullPointerException ex)

{

System.out.println("Exception arise is"+ex);

}

System.out.println("hello");

}

}
```

Difference between throw and throws keyword->

Sr. no.	Basis of Differences	throw	throws
1.	Definition	Java throw keyword is used throw an exception explicitly in the code, inside the function or the block of code.	Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code.
2.	Type of exception Using throw keyword, we can only propagate unchecked exception i.e., the checked exception cannot be propagated using throw only.	Using throws keyword, we can declare both checked and unchecked exceptions. However, the throws keyword can be used to propagate checked exceptions only.	
3.	Syntax	The throw keyword is followed by an instance of Exception to be thrown.	The throws keyword is followed by class names of Exceptions to be thrown.
4.	Declaration	throw is used within the method.	throws is used with the method signature.
5.	Internal implementation	We are allowed to throw only one exception at a time i.e. we cannot throw multiple exceptions.	We can declare multiple exceptions using throws keyword that can be thrown by the method. For example, main() throws IOException, SQLException.