EXCEPTION HANDLING

Exception :

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Exception Handling :

The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

Try Catch Block :

Java provides an inbuilt exceptional handling.

1.The normal code goes into a TRY block.

2.The exception handling code goes into the CATCH block

Syntax :

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

Example :

```
public class TryCatchExample {
  public static void main(String[] args) {
      int i=50;
      int j=0;
      int data;
      try
      {
      data=i/j; //may throw exception
      }
         // handling the exception
```

```
        catch(Exception e)
        {
            // resolving the exception in catch block
           System.out.println(i/(j+2));
        }
    }
}
```

Java Finally Block :

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

  statement(s)

 } finally {

  statement(s)
 }
```

Final Block Example :

```
class JavaException {
   public static void main(String args[]){
    try{
       int d = 0;
       int n =20;
       int fraction = n/d;
    }
  catch(ArithmeticException e){
    System.out.println("In the catch clock due to Exception = "+e);
  }
  finally{
```

```
        System.out.println("Inside the finally block");
  }
}
}
```

CHECKED and UNCHECKED EXCEPTIONS

The main difference between checked and unchecked exception is that the checked exceptions are checked at compile-time while unchecked exceptions are checked at runtime.

What are **checked exceptions?**

Checked exceptions are checked at compile-time. It means if a method is throwing a checked exception then it should handle the exception using try-catch block or it should declare the exception using throws keyword, otherwise the program will give a compilation error.

Here are the few other Checked Exceptions –

SQLException

IOException

ClassNotFoundException

InvocationTargetException

Example

For example, if you use FileReader class in your program to read data from a file, if the file specified in its constructor doesn't exist, then a FileNotFoundException occurs, and the compiler prompts the programmer to handle the exception.

```
import java.io.File;
import java.io.FileReader;
```

```
public class FilenotFound_Demo {


  public static void main(String args[]) {
    File file = new File("E://file.txt");
    FileReader fr = new FileReader(file);
  }
}
```

What are **Unchecked exceptions?**

Unchecked exceptions are not checked at compile time. It means if your program is throwing an unchecked exception and even if you didn't handle/declare that exception, the program won't give a compilation error. Most of the times these exception occurs due to the bad data provided by user during the user-program interaction. It is up to the programmer to judge the conditions in advance, that can cause such exceptions and handle them appropriately. All Unchecked exceptions are direct sub classes of RuntimeException class.

Here are the few unchecked exception classes:

NullPointerException

ArrayIndexOutOfBoundsException

ArithmeticException

IllegalArgumentException

NumberFormatException

Example

```
class Example {
  public static void main(String args[]) {
        try{
          int arr[] ={1,2,3,4,5};
          System.out.println(arr[7]);
        }
     catch(ArrayIndexOutOfBoundsException e){
```

```
            System.out.println("The specified index does not exist " +
                    "in array. Please correct the error.");
        }
    }
}
```

BUILD -IN EXCEPTIONS

Built-in exceptions are the exceptions which are available in Java libraries. These exceptions are suitable to explain certain error situations. Below is the list of important built-in exceptions in Java.

Arithmetic Exception :

It is thrown when an exceptional condition has occurred in an arithmetic operation.

Ex : int a=50/0;//ArithmeticException

ArrayIndexOutOfBoundException :

It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

Ex :

int a[]=new int[5];

a[10]=50; //ArrayIndexOutOfBoundsException

ClassNotFoundException :

This Exception is raised when we try to access a class whose definition is not found

FileNotFoundException :

This Exception is raised when a file is not accessible or does not open.

IOException :

It is thrown when an input-output operation failed or interrupted

InterruptedException :

It is thrown when a thread is waiting , sleeping , or doing some processing , and it is interrupted.

NoSuchFieldException :

It is thrown when a class does not contain the field (or variable) specified

NoSuchMethodException :

It is thrown when accessing a method which is not found.

NullPointerException :

This exception is raised when referring to the members of a null object. Null represents nothing

Ex :

String s=null;

System.out.println(s.length());//NullPointerException

NumberFormatException :

This exception is raised when a method could not convert a string into a numeric format.

Ex : String s="abc";

int i=Integer.parseInt(s);//NumberFormatException

RuntimeException :

This represents any exception which occurs during runtime.

StringIndexOutOfBoundsException :

It is thrown by String class methods to indicate that an index is either negative than the size of the string

---

CUSTOM EXCEPTIONS

Custom or User Exceptions :

In java we can create our own exception class and throw that exception using throw keyword. These exceptions are known as user-defined or custom exceptions.

```
class InvalidProductException extends Exception
{
    public InvalidProductException(String s)
    {
        // Call constructor of parent Exception
        super(s);
    }
}

public class Example1
{
  void productCheck(int weight) throws InvalidProductException{
            if(weight<100){
                    throw new InvalidProductException("Product Invalid");
            }
  }

    public static void main(String args[])
    {
            Example1 obj = new Example1();
        try
        {
```

```
            obj.productCheck(60);
        }
        catch (InvalidProductException ex)
        {
            System.out.println("Caught the exception");
            System.out.println(ex.getMessage());
        }
    }
}
```

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:

Throw	Throws
Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
Throw is followed by an instance.	Throws is followed by class.
Throw is used within the method.	Throws is used with the method signature.
You cannot throw multiple exceptions.	You can declare multiple exceptions e.g.public void method()throws IOException,SQLException.

Throw Example

```
public class Example1{
   void checkAge(int age){
            if(age<18)
              throw new ArithmeticException("Not Eligible for voting");
            else
              System.out.println("Eligible for voting");
   }
   public static void main(String args[]){
            Example1 obj = new Example1();
            obj.checkAge(13);
            System.out.println("End Of Program");
   }
}
```

Throws Example

```
import java.io.*;
class ThrowExample {
  void myMethod(int num)throws IOException, ClassNotFoundException{
     if(num==1)
       throw new IOException("IOException Occurred");
     else
       throw new ClassNotFoundException("ClassNotFoundException");
  }
}

public class Example1{
```

```
  public static void main(String args[]){
   try{
     ThrowExample obj=new ThrowExample();
     obj.myMethod(1);
   }catch(Exception ex){
     System.out.println(ex);
    }
  }
}
```