Fundamentals of Exception Handling:

Exception Handling in Java

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.

In this page, we will learn about Java exceptions, its type and the difference between checked and unchecked exceptions.

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.

What is Exception Handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5;//exception occurs
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;
10. statement 10;

Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

Unchecked Exception
Checked Exception
Error
Exception Handling

Difference between Checked and Unchecked Exceptions

1) Checked Exception

The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

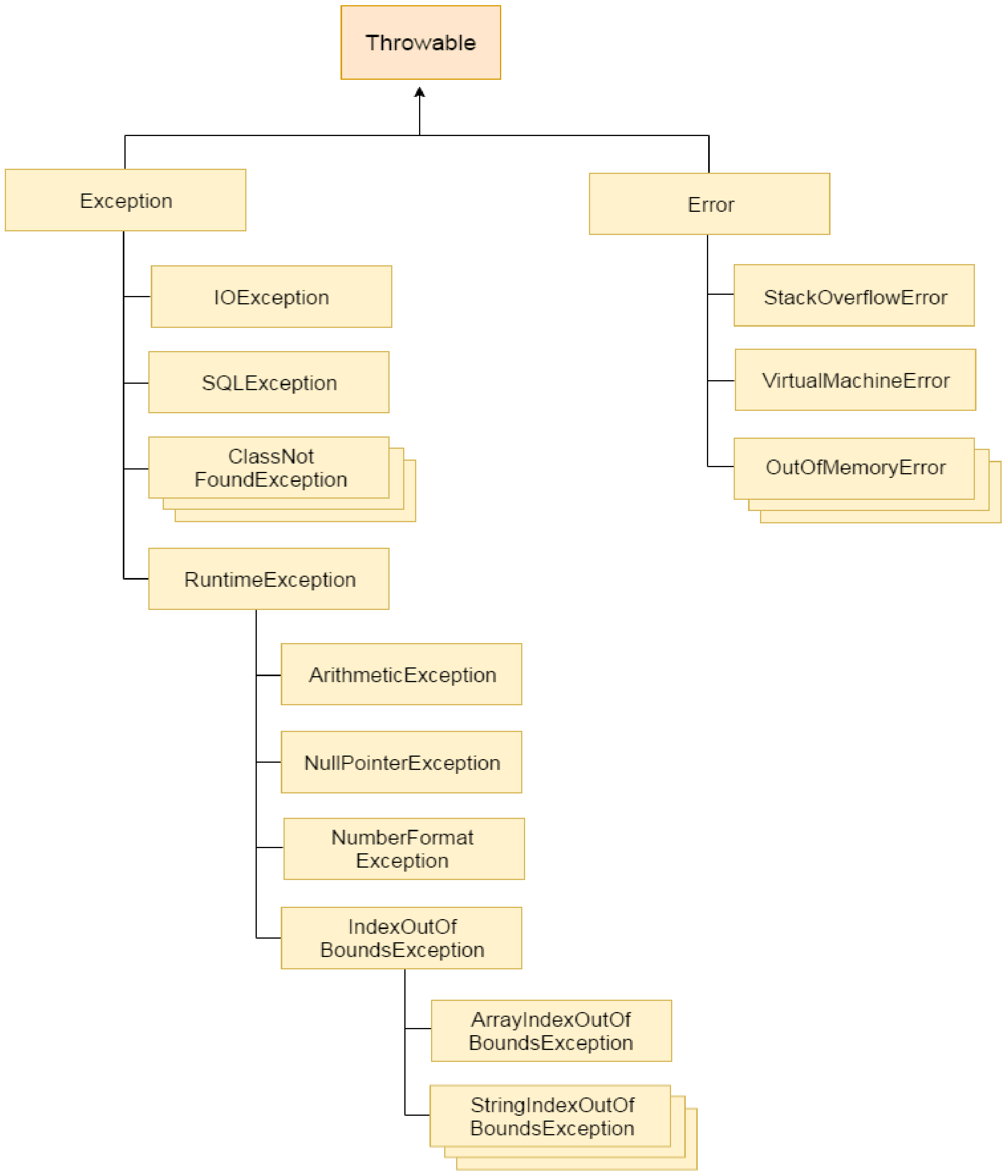
The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Hierarchy of Java Exception classes

The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:



Java Exception Keywords

There are 5 keywords which are used in handling exceptions in Java.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
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 important 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 doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

Java Exception Handling Example

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

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

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

In this page, we will learn about Java exceptions, its type and the difference between checked and unchecked exceptions.

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.

What is Exception Handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**.

An exception normally disrupts the normal flow of the application that is why we use exception handling.

Let's take a scenario:

```
1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5;//exception occurs
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;
10. statement 10;
```

Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed.

If we perform exception handling, the rest of the statement will be executed.

That is why we use exception handling in Java.

Do You Know?

- What is the difference between checked and unchecked exceptions?
- What happens behind the code int data=50/0;?
- Why use multiple catch block?
- Is there any possibility when finally block is not executed?
- What is exception propagation?
- What is the difference between throw and throws keyword?
- What are the 4 rules for using exception handling with method overriding?

Hierarchy of Java Exception classes

The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:

Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

Difference between Checked and Unchecked Exceptions

1) Checked Exception

The classes which directly inherit Throwable class except Runtime Exception and Error are known as

throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

Java Exception Handling Example

Let's see an example of Java Exception Handling where we using a try-catch statement

to handle the exception.

```
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.

Common Scenarios of Java Exceptions

There are given some scenarios where unchecked exceptions may occur. They are as follows:

1) A scenario where Arithmetic Exception occurs

If we divide any number by zero, there occurs an Arithmetic Exception.

```
int a=50/0;//Arithmetic Exception
```

2) A scenario where Null Pointer Exception occurs

If we have a null value in any variable, performing any operation on the variable throws a Null Pointer Exception.

```
String s=null;
```

```
2. System.out.println(s.length());//Null Pointer Exception
```

3) A scenario where Number Format Exception occurs

The wrong formatting of any value may occur Number Format Exception. Suppose I have a string variable that has characters, converting this variable into digit will occur Number Format Exception.

```
1. String s="abc";
2. int i=Integer.parseInt(s);//Number Format Exception
```

4) A scenario where Array Index Out Of Bounds Exception occurs

If you are inserting any value in the wrong index, it would result in Array Index Out Of Bounds Exception as shown below:

```
1. int a[]=new int[5];
2. a[10]=50; //Array Index Out Of Bounds Exception
```

Flow control in try catch finally in Java

In this article, we'll explore all the possible combinations of try-catch-finally which may happen whenever an exception is raised and how the control flow occurs in each of the given cases.

1. **Control flow in try-catch clause OR try-catch-finally clause**
    - **Case 1:** Exception occurs in try block and handled in catch block
    - **Case 2:** Exception occurs in try-block is not handled in catch block
    - **Case 3:** Exception doesn't occur in try-block
2. **try-finally clause**
    - **Case 1:** Exception occurs in try block
    - **Case 2:** Exception doesn't occur in try-block

Control flow in try-catch OR try-catch-finally

1. **Exception occurs in try block and handled in catch block:** If a statement in try block raised an exception, then the rest of the try block doesn't execute and control passes to the **corresponding** catch block. After executing the catch block, the control will be transferred to finally block(if present) and then the rest program will be executed.
    - **Control flow in try-catch:**

```
// Java program to demonstrate
// control flow of try-catch clause
// when exception occur in try block
// and handled in catch block
class GFG
{
   public static void main (String[] args)
```

```
    {

        // array of size 4.
        int[] arr = new int[4];
        try
        {
            int i = arr[4];

            // this statement will never execute
            // as exception is raised by above statement
            System.out.println("Inside try block");
        }
        catch(ArrayIndexOutOfBoundsException ex)
        {
            System.out.println("Exception caught in Catch block");
        }

        // rest program will be excuted
        System.out.println("Outside try-catch clause");
    }
}
```

Output:

```
Exception caught in Catch block

Outside try-catch clause
```

- **Control flow in try-catch-finally clause :**

```
// Java program to demonstrate
// control flow of try-catch-finally clause
// when exception occur in try block
// and handled in catch block
class GFG
{
    public static void main (String[] args)
    {

        // array of size 4.
        int[] arr = new int[4];

        try
        {
            int i = arr[4];

            // this statement will never execute
            // as exception is raised by above statement
            System.out.println("Inside try block");
        }

        catch(ArrayIndexOutOfBoundsException ex)
        {
            System.out.println("Exception caught in catch block");
        }

        finally
        {
            System.out.println("finally block executed");
        }

        // rest program will be executed
        System.out.println("Outside try-catch-finally clause");
    }
}
```

Output:

```
Exception caught in catch block

finally block executed

Outside try-catch-finally clause
```

throw and throws in Java

throw

The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. We can throw either checked or unchecked exception. The throw keyword is mainly used to throw custom exceptions.

Syntax:

throw *Instance*

Example:

throw new ArithmeticException("/ by zero");

But this exception i.e, *Instance* must be of type **Throwable** or a subclass of **Throwable**. For example Exception is a sub-class of Throwable and user defined exceptions typically extend Exception class. Unlike C++, data types such as int, char, floats or non-throwable classes cannot be used as exceptions.

The flow of execution of the program stops immediately after the throw statement is executed and the nearest enclosing **try** block is checked to see if it has a **catch** statement that matches the type of exception. If it finds a match, controlled is transferred to that statement otherwise next enclosing **try** block is checked and so on. If no matching **catch** is found then the default exception handler will halt the program.

```
// Java program that demonstrates the use of throw
class ThrowExcep
```

```
{
   static void fun()
   {
      try
      {
         throw new NullPointerException("demo");
      }
      catch(NullPointerException e)
      {
         System.out.println("Caught inside fun().");
         throw e; // rethrowing the exception
      }
   }

   public static void main(String args[])
   {
      try
      {
         fun();
      }
      catch(NullPointerException e)
      {
         System.out.println("Caught in main.");
      }
   }
}
```

Output:

```
Caught inside fun().
Caught in main.
```

Another Example:

```
// Java program that demonstrates the use of throw
class Test
{
   public static void main(String[] args)
   {
      System.out.println(1/0);
   }
}
```

Output:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

throws

throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception using a try-catch block.

Syntax:

```
type method_name(parameters) throws exception_list
exception_list is a comma separated list of all the
exceptions which a method might throw.
```

In a program, if there is a chance of rising an exception then compiler always warn us about it and compulsorily we should handle that checked exception, Otherwise we will get compile time error saying **unreported exception XXX must be caught or declared to be thrown**. To prevent this compile time error we can handle the exception in two ways:

1. By using try catch
2. By using **throws** keyword

We can use throws keyword to delegate the responsibility of exception handling to the caller (It may be a method or JVM) then caller method is responsible to handle that exception.

```
// Java program to illustrate error in case
// of unhandled exception
class tst
{
   public static void main(String[] args)
   {
      Thread.sleep(10000);
      System.out.println("Hello Geeks");
   }
}
```

Output:

```
error: unreported exception InterruptedException; must be caught or declared to be thrown
```

Explanation : In the above program, we are getting compile time error because there is a chance of exception if the main thread is going to sleep, other threads get the chance to execute main() method which will cause InterruptedException.

```
// Java program to illustrate throws
class tst
{
   public static void main(String[] args)throws InterruptedException
   {
      Thread.sleep(10000);
      System.out.println("Hello IT'ians");
   }
}
```

Output:

```
Hello IT'ians
```

Explanation : In the above program, by using throws keyword we handled the InterruptedException and we will get the output as **Hello Geeks**

Another Example:

```
// Java program to demonstrate working of throws
class ThrowsExecp
{
   static void fun() throws IllegalAccessException
   {
      System.out.println("Inside fun(). ");
      throw new IllegalAccessException("demo");
   }
   public static void main(String args[])
   {
      try
      {
         fun();
      }
      catch(IllegalAccessException e)
      {
         System.out.println("caught in main.");
      }
   }
}
```

Output:

```
Inside fun().
caught in main.
```

Important points to remember about throws keyword:

- throws keyword is required only for checked exception and usage of throws keyword for unchecked exception is meaningless.
- throws keyword is required only to convince compiler and usage of throws keyword does not prevent abnormal termination of program.
- By the help of throws keyword we can provide information to the caller of the method about the exception.

User defined exception in java

In java we have already defined, exception classes such as ArithmeticException, NullPointerException etc. These exceptions are already set to trigger on pre-defined conditions such as when you divide a number by zero it triggers ArithmeticException, In the last tutorial we learnt how to throw these exceptions explicitly based on your conditions using throw keyword.

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. In this tutorial we will see how to create your own custom exception and throw it on a particular condition.

To understand this tutorial you should have the basic knowledge of try-catch block and throw in java.

Example of User defined exception in Java

```
/* This is my Exception class, I have named it MyException
 * you can give any name, just remember that it should
 * extend Exception class
 */
class MyException extends Exception{
   String str1;
   /* Constructor of custom exception class
    * here I am copying the message that we are passing while
    * throwing the exception to a string and then displaying
    * that string along with the message.
    */
   MyException(String str2) {
        str1=str2;
   }
   public String toString(){
        return ("MyException Occurred: "+str1) ;
   }
}

class Example1{
   public static void main(String args[]){
        try{
             System.out.println("Starting of try block");
             // I'm throwing the custom exception using throw
             throw new MyException("This is My error Message");
        }
        catch(MyException exp){
             System.out.println("Catch Block") ;
             System.out.println(exp) ;
        }
   }
}
```

Output:

```
Starting of try block
Catch Block
MyException Occurred: This is My error Message
```

Explanation:

You can see that while throwing custom exception I gave a string in parenthesis (throw new MyException("This is My error Message");). That's why we have a parameterized constructor (with a String parameter) in my custom exception class.