MIRACLE SOFTWARE SYSTEMS, INC.

Exception Handling

Tables Of Content

1.Exception Handling

The Exception Handling is one of most useful mechanism used in Java. It is used to handle the runtime errors so the flow of the progam will not be disturbed and maintained as expected.

So here we will know about Exception handling how to use it and when to use it and the difference between checked and unchecked exceptions.

1.1 Exception

- **Dictionary Meaning:** Exception is an abnormal condition.
- In java, exception is an event which disturbs the normal flow of the program. It is an object which is thrown at runtime.

Exception Handling

Exception Handling is a mechanism to handle runtime errors.So by handling the exception the flow of the program will be maintained even though the Exceptions occour.

1.2 Advantages of Exception Handling

The core advantage of exception handling is that normal flow of the application is maintained. Exception normally disturbs the normal flow of the application that is why we use exception handling.

1.3 Types of Exception:

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception.

- Checked Exception
- Unchecked Exception
- Error

1) Checked Exception:

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

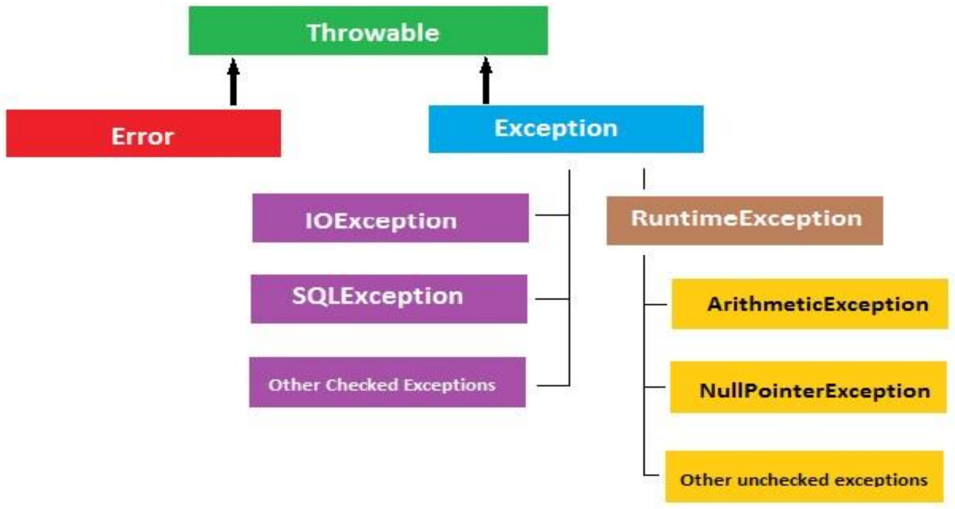
2)Unchecked Exception

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

3) Error

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

1.4 Exception Classes Hierarchy



How to use Exception Handling :
Five keywords plays major role in Exception handling:

- **try**
- **catch**
- **finally**
- **throw**
- **throws**

1.5 try block

Try block is the place where we need to store the code which is having a chance to raise an Exception. It must be used within the method and must be followed by either catch or finally block.

Example :

```
try{
          int  value/0;  // Code having the possibility to raise exception.
     }
```

1.6 catch block

Catch block is used to handle the Exception. It must be used after the try block. It is the immediate block followd after the try block.

Example :

```
catch(Exception e)// Exception is class and  e is Exception Object
       {
            System.out.println("Divider number should not be Zero");
        }
```

Output :

Divider number should not be Zero.

Without Exception Handling :

Output:

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

The JVM firstly checks whether the exception is handled or not. If exception is not handled, JVM provides a default exception handler that performs the following tasks:

1. Prints out exception description.
2. Prints the stack trace (Hierarchy of methods where the exception occurred).
3. Causes the program to terminate.

But if exception is handled by the application programmer, normal flow of the application is maintained i.e. rest of the code is executed.

Example :

1.7 Multiple catch block

If different Exceptions are going to raise in your program and if you want to handle all the rasied Exceptions then you nedd to go with **Multiple catch blocks**

Example :

```
   MyClass
       {
      public static void main(String args[]){
         try
             {
              int array[]=new int[5];
              array[5]=30/0;
             }
      catch(ArithmeticException e)
         {
                   System.out.println("task1 is completed");
             }
         catch(ArrayIndexOutOfBoundsException e)
           {
              System.out.println("task 2 completed");
           }
      catch(Exception e)
          {
                   System.out.println("common task completed");
           }

              System.out.println("restof the code...");
}
}
```

Output :

```
            task1 completed
           rest of the code...
```

Note : In this scenario only one exception will be handled at a time. Then only its prespective catch block will be executed.

Exception class shold be in last while using multiple catch blocks because it is the parent class of all Exception classes.

Nested try block:

If we write a try block inside of another try block then it is called as Nested try block

Why use nested try block?

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

Example :

```
        try
        {
                statement1;
                statement2;
                try
                {
                        statement1;
                        statement2;
                }
                catch(Exception e)
                {
                        System.out.println(e);
                }
        }
        catch(Exception e)
        {
                        System.out.println(e);
        }
```

1.8 finally block

The finally block is a block that is always executed irrespective of the Exception rasied. It is mainly used to perform some important tasks such as closing connection, stream, etc.

Note: Before terminating the program, JVM executes finally block(if any).

Note: finally block must be followed by try or catch block.

Why use finally block?

Finally block can be used to put "cleanup" code such as closing a file,closing connection etc.

Example :

```
      class Simple
      {
            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...
```

1.9 throw keyword

The throw keyword is used to explictily throw an exception.throw key word is mainly used to inform the JVM where an Exception is going to be rasied.We can throw either checked or uncheked exception. The throw keyword is mainly used to throw custom exception.

Example of throw keyword

In this example, we have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

Example :

```
Class AgeException
{
      static void validate(int age)
      {
         if(age<18)
            {
             throw new ArithmeticException("not valid");
             }
           else
            {
            System.out.println("welcome to vote");
             }
          }
```

```
	public static void main(String args[]){
	validate(13);
	System.out.println("rest of the code...");
	}
}
```

Output : Output:Exception in thread main java.lang.ArithmeticException:not valid.

1.10 throws keyword

The **throws keyword** is used to declare an exception. Throws keyword is used beside method. Throws keyword is used if we want escape from the exception.

So if a particular exception is raised in that method then if we use throws keyword beside that method. Then the method which called that particular method will handle the Exception. This is also know as escaping from Exception and the Calling method will handle the Exception

Syntax of throws keyword:
Void method_name() throws exception_class_name{

...
}

Here we can declare only checked Exceptions, because handling of unchecked exception is programmers responsibility.

If you are calling a method that declares an exception, you must either caught or declare the exception.
There are two cases:

Case1: You caught the exception i.e handling the Exception using try and catch blocks
Case2: You escpaing the Exception by using throws keyword.

Case1: You handle the exception

- In case you handle the exception, the code will be executed fine whether exception occurs during the program or not.

Example :

```
import java.io.*;
class HandlingException
{
 void method()throwsIOException
	{
	throw new IOException("device error");
	}
}


class Testing extends HandlingException
{
	 public static void main(String args[]){
	try
	{
		Testing test=new Testing();
		test.method();
		catch(Exception e)
	{
		System.out.println("exception handled");
	}
```

```
            System.out.println("normal flow...");
      }
}
```

Output:exception handled
normal flow...

Case2: You declare the exception

A) In case you declare the exception, if exception does not occur, the code will be executed fine.

B) In case you declare the exception if exception occures, an exception will be thrown at runtime because throws does not handle the exception.

A)Program if exception doesnt occurs

Example :
import java.io.*;

```
class HandlingExceptions
{
       void method()throws IOException
      {
      System.out.println("device operation performed");
      }
}


class Testing extends HandlingException{
      public static void main(String args[])throws IOException{//declare exception
      Testing test=new Testing();
      test.method();
      System.out.println("normal flow...");
      }
}
```

Output: device operation performed
normal flow...

B)Program if exception occurs Exceptions Example :

```
importjava.io.*;
class HandlingExceptions
{
      void method()throws IOException
      {
      throw new IOException("device error");
      }
}


class Testing
{
      public static void main(String args[])throws IOException
       {
      Testing test=new Testing();
      t.method();
      System.out.println("normal flow...");
      }
}
```

Difference between throw and throws:

throw keyword	throws keyword
1.throw is used to explicitly throw an exception.	throws is used to declare an exception.
2.checked exception can not be propagated without throws.	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 exception	You can declare multiple exception e.g.
public void method()throws IOException,SQLException.	

1.11 Custom Exception

If you are creating your own Exception that is known as custom exception or user-defined exception.

Example :

```
Class InvalidAgeException extends Exception
{
      InvalidAgeException(String s){
super(s);
 }
}
class CustomExceptionDemo
{
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...