LECTURE NOTES

UNIT 3

PACKAGES AND EXCEPTION Package naming, type imports, package access
Package contents, package objects and specifications, examples
Creating exception types, throw, throws
Try, catch and finally
Custom exception, when to use exception

1.0. JAVA PACKAGE

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

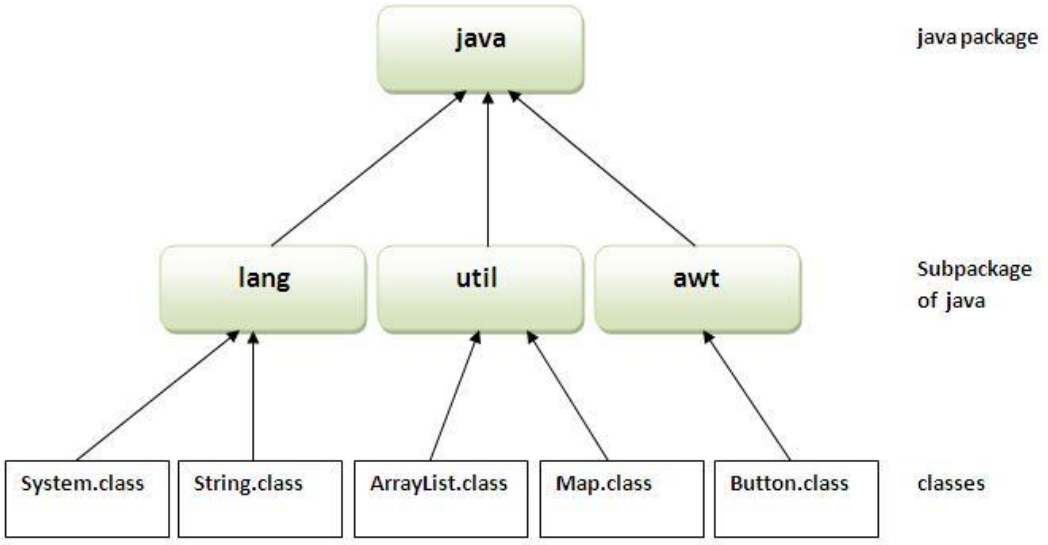
Here, we will have the detailed learning of creating and using user-defined packages.

1.1.1. ADVANTAGE OF JAVA PACKAGE

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.



SIMPLE EXAMPLE OF JAVA PACKAGE

The **package keyword** is used to create a package in java.

```
1. //save as Simple.java
2. package mypack;
3. public class Simple{
4.  public static void main(String args[]){
5.    System.out.println("Welcome to package");
6.   }
7. }
```

HOW TO COMPILE JAVA PACKAGE

If you are not using any IDE, you need to follow the **syntax** given below:

```
1. javac -d directory javafilename
```

For **example**

```
1. javac -d . Simple.java
```

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

HOW TO RUN JAVA PACKAGE PROGRAM

You need to use fully qualified name e.g. mypack.Simple etc to run the class.

To Compile: javac -d . Simple.java

To Run: java mypack.Simple

```
Output:Welcome to package
```

The -d is a switch that tells the compiler where to put the class file i.e. it represents destination. The . represents the current folder.

HOW TO ACCESS PACKAGE FROM ANOTHER PACKAGE?

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;
3. fully qualified name.

1) USING PACKAGENAME.*

If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

EXAMPLE OF PACKAGE THAT IMPORT THE PACKAGENAME.*

```
1. //save by A.java
2.
3. package pack;
4. public class A{
5.   public void msg(){System.out.println("Hello");}
6. }
1. //save by B.java
2.
3. package mypack;
4. import pack.*;
5.
6. class B{
7.   public static void main(String args[]){
8.    A obj = new A();
9.    obj.msg();
10.  }
11. }
```

```
Output:Hello
```

2) USING PACKAGENAME.CLASSNAME

If you import package.classname then only declared class of this package will be accessible.

EXAMPLE OF PACKAGE BY IMPORT PACKAGE.CLASSNAME

```
1. //save by A.java
2.
3. package pack;
4. public class A{
5.   public void msg(){System.out.println("Hello");}
6. }
1. //save by B.java
2.
3. package mypack;
4. import pack.A;
5.
6. class B{
7.   public static void main(String args[]){
8.    A obj = new A();
9.    obj.msg();
10.  }
11. }
```

```
Output:Hello
```

---

3) USING FULLY QUALIFIED NAME

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

EXAMPLE OF PACKAGE BY IMPORT FULLY QUALIFIED NAME

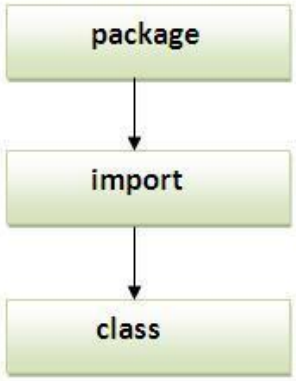
```
1. //save by A.java
2.
3. package pack;
4. public class A{
5.   public void msg(){System.out.println("Hello");}
6. }
1. //save by B.java
2.
3. package mypack;
4. class B{
5.   public static void main(String args[]){
6.    pack.A obj = new pack.A();//using fully qualified name
7.    obj.msg();
8.   }
```

9. }

```
Output:Hello
```

If you import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the subpackages. Hence, you need to import the subpackage as well.

Sequence of the program must be package then import then class.



1.2.SUBPACKAGE IN JAVA

Package inside the package is called the **subpackage**. It should be created **to categorize the package further**.

Let's take an example, Sun Microsystem has definded a package named java that contains many classes like System, String, Reader, Writer, Socket etc. These classes represent a particular group e.g. Reader and Writer classes are for Input/Output operation, Socket and ServerSocket classes are for networking etc and so on. So, Sun has subcategorized the java package into subpackages such as lang, net, io etc. and put the Input/Output related classes in io package, Server and ServerSocket classes in net packages and so on.

EXAMPLE OF SUBPACKAGE

1. **package** com.javatpoint.core;

```
2. class Simple{
3.   public static void main(String args[]){
4.    System.out.println("Hello subpackage");
5.   }
6. }
```

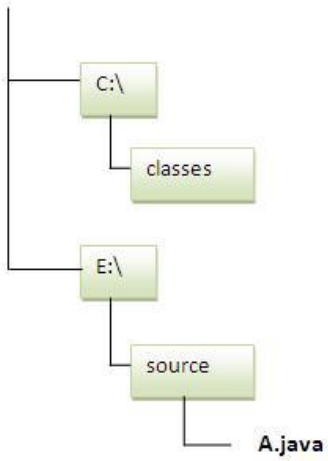
To Compile: javac -d . Simple.java

To Run: java com.javatpoint.core.Simple

```
Output:Hello subpackage
```

HOW TO SEND THE CLASS FILE TO ANOTHER DIRECTORY OR DRIVE?

There is a scenario, I want to put the class file of A.java source file in classes folder of c: drive. For example:




```
1. //save as Simple.java
2.
3. package mypack;
4. public class Simple{
5.  public static void main(String args[]){
6.     System.out.println("Welcome to package");
7.    }
8. }
```

To Compile:

e:\sources> javac -d c:\classes Simple.java

To Run:

To run this program from e:\source directory, you need to set classpath of the directory where the class file resides.

e:\sources> set classpath=c:\classes;.;

e:\sources> java mypack.Simple

ANOTHER WAY TO RUN THIS PROGRAM BY -CLASSPATH SWITCH OF JAVA:

The -classpath switch can be used with javac and java tool.

To run this program from e:\source directory, you can use -classpath switch of java that tells where to look for class file. For example:

e:\sources> java -classpath c:\classes mypack.Simple

```
Output:Welcome to package
```

WAYS TO LOAD THE CLASS FILES OR JAR FILES

There are two ways to load the class files temporary and permanent.

- Temporary
  - By setting the classpath in the command prompt
  - By -classpath switch
- Permanent
  - By setting the classpath in the environment variables
  - By creating the jar file, that contains all the class files, and copying the jar file in the jre/lib/ext folder.

```
1. //save as C.java otherwise Compilte Time Error
2.
3. class A{}
4. class B{}
```

```
5. public class C{}
```

HOW TO PUT TWO PUBLIC CLASSES IN A PACKAGE?

If you want to put two public classes in a package, have two java source files containing one public class, but keep the package name same. For example:

```
1. //save as A.java
2.
3. package javatpoint;
4. public class A{}
1. //save as B.java
2.
3. package javatpoint;
4. public class B{}
```

2.0. 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 exception, its type and the difference between checked and unchecked exceptions.

WHAT IS EXCEPTION

Dictionary Meaning: Exception is an abnormal condition.

In java, 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 ClassNotFound, IO, SQL, Remote etc.

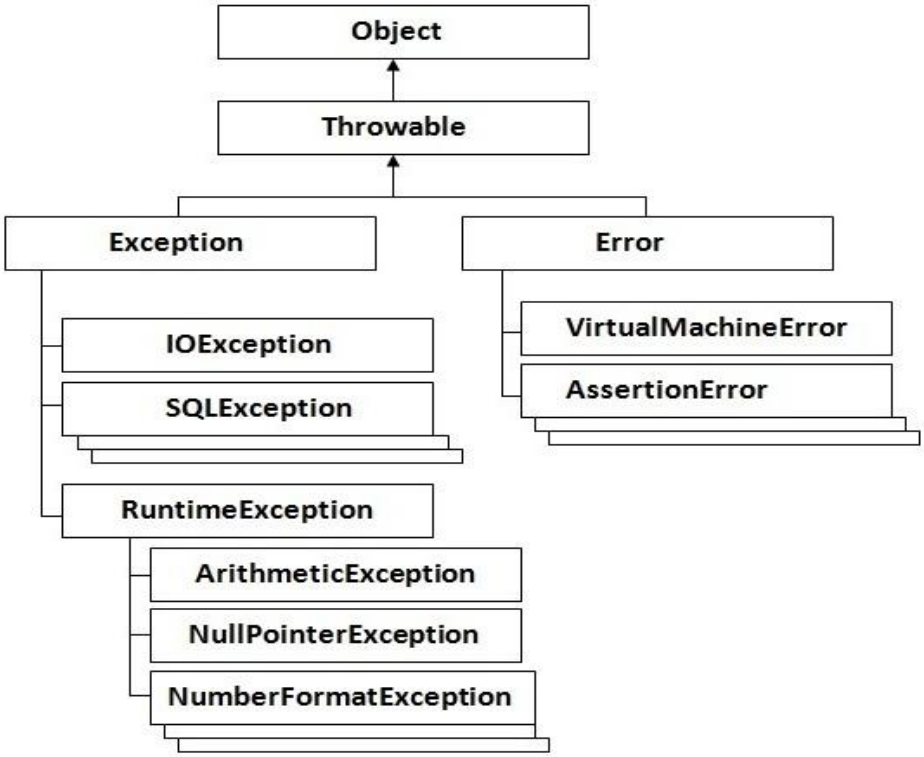
Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. 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 is 10 statements in your program and there occurs an exception at statement 5, rest of the code will not be executed i.e. statement 6 to 10 will not run. If we perform exception handling, rest of the exception will be executed. That is why we use exception handling in java.

HIERARCHY OF EXCEPTION CLASSES



2.1.TYPES OF EXCEPTION

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says 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 that extend 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.

COMMON SCENARIOS WHERE EXCEPTIONS MAY OCCUR

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

1) Scenario where ArithmeticException occurs

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

```
1. int a=50/0;//ArithmeticException
```

2) Scenario where NullPointerException occurs

If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

```
1. String s=null;
2. System.out.println(s.length());//NullPointerException
```

3) Scenario where NumberFormatException occurs

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

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

4) Scenario where ArrayIndexOutOfBoundsException occurs

If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

```
1. int a[]=new int[5];
2. a[10]=50; //ArrayIndexOutOfBoundsException
```

2.2.USE OF TRY-CATCH BLOCK IN EXCEPTION HANDLING:

Five keywords used in Exception handling:

1. try
2. catch
3. finally
4. throw
5. throws

TRY BLOCK

Enclose the code that might throw an exception in try block. It must be used within the method and must be followed by either catch or finally block.

SYNTAX OF TRY WITH CATCH BLOCK

```
try{
...
}catch(Exception_class_Name reference){}
```

Syntax of try with finally block

```
try{
...
}finally{}
```

catch block

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

Problem without exception handling

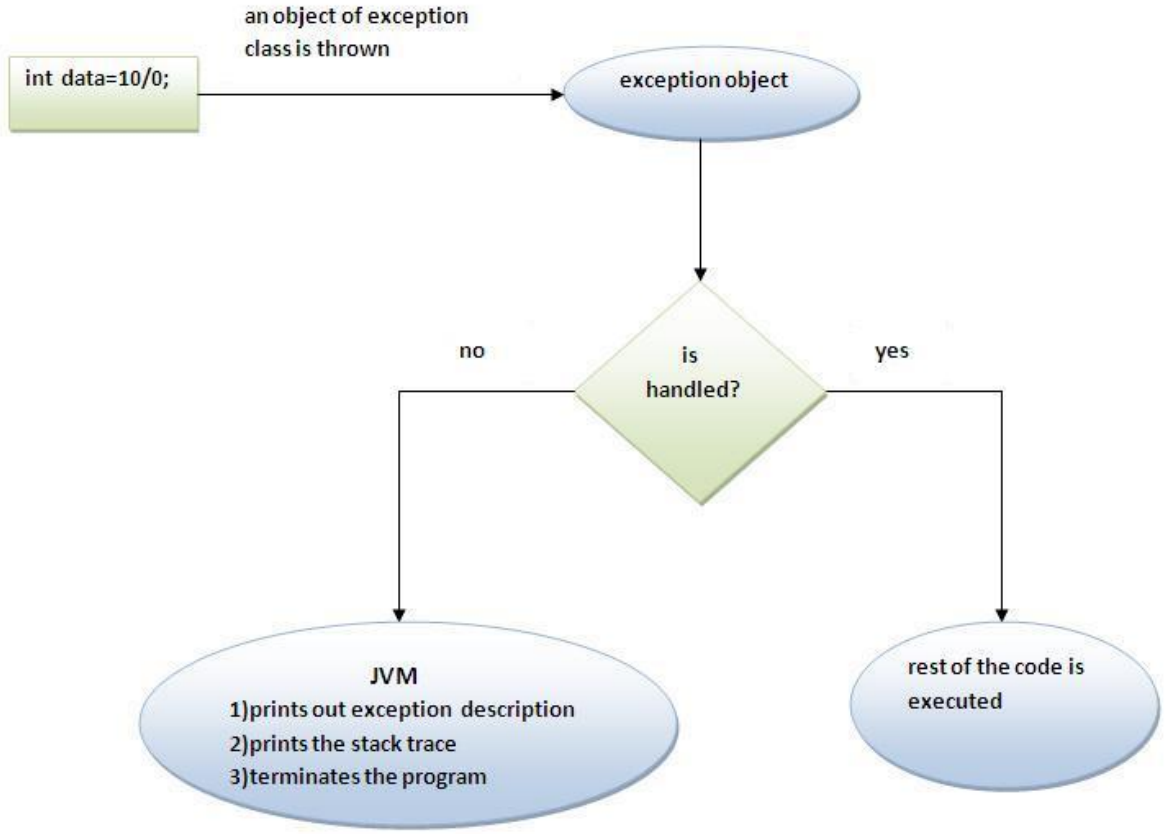
```
public class Testtrycatch1{
  public static void main(String args[]){
      int data=50/0;

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

Test it Now

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

As displayed in the above example, rest of the code is not executed i.e. rest of the code... statement is not printed. Let's see what happens behind the scene:

What happens behind the code int a=50/0;



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:

- Prints out exception description.
- Prints the stack trace (Hierarchy of methods where the exception occurred).
- 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.

SOLUTION BY EXCEPTION HANDLING

```
1. public class Testtrycatch2{
2.   public static void main(String args[]){
3.    try{
4.       int data=50/0;
5.
```

```
6.    }catch(ArithmeticException e){System.out.println(e);}
7.
8.    System.out.println("rest of the code...");
9.  }
10. }
```

Test it Now

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

Now, as displayed in the above example, rest of the code is executed i.e. rest of the code... statement is printed.

2.3.MULTIPLE CATCH BLOCK:

If you have to perform different tasks at the occrence of different Exceptions, use multple catch block.

Example of multiple catch block

```
1.  public class TestMultipleCatchBlock{
2.    public static void main(String args[]){
3.     try{
4.      int a[]=new int[5];
5.      a[5]=30/0;
6.     }
7.     catch(ArithmeticException e){System.out.println("task1 is completed");}
8.     catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}
9.     catch(Exception e){System.out.println("common task completed");}
10.
11.    System.out.println("rest of the code...");
12.  }
13. }
```

Test it Now

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

```
1.  class TestMultipleCatchBlock1{
2.    public static void main(String args[]){
3.     try{
4.      int a[]=new int[5];
5.      a[5]=30/0;
6.     }
7.     catch(Exception e){System.out.println("common task completed");}
8.     catch(ArithmeticException e){System.out.println("task1 is completed");}
9.     catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}
10.
11.    System.out.println("rest of the code...");
```

```
12. }
13. }
```

Test it Now

```
Output:Compile-time error
```

2.4.NESTED TRY BLOCK:

try block within a try block is known 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

Syntax:

```
1. ....
2. try
3. {
4.     statement 1;
5.     statement 2;
6.     try
7.     {
8.         statement 1;
9.         statement 2;
10.    }
11.    catch(Exception e)
12.    {
13.    }
14. }
15. catch(Exception e)
16. {
17. }
18. ....
```

Example:

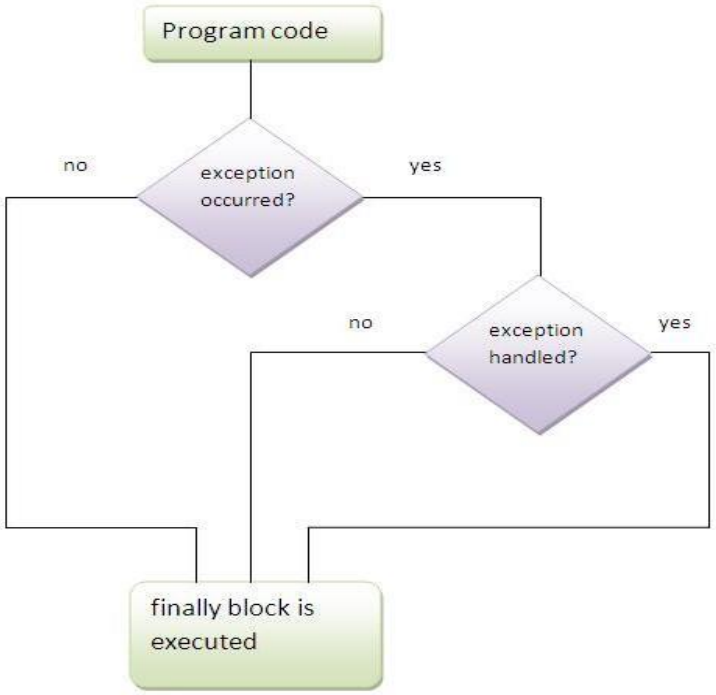
Example of nested try block

```
1.  class Excep6{
2.   public static void main(String args[]){
3.    try{
4.      try{
5.       System.out.println("going to divide");
6.       int b =39/0;
7.      }catch(ArithmeticException e){System.out.println(e);}
8.
```

```
9.      try{
10.     int a[]=new int[5];
11.     a[5]=4;
12.     }catch(ArrayIndexOutOfBoundsException e){System.out.println(e);}
13.
14.     System.out.println("other statement);
15.   }catch(Exception e){System.out.println("handeled");}
16.
17.   System.out.println("normal flow..");
18.  }
19. }
```

2.5.FINALLY BLOCK

The finally block is a block that is always executed. 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 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.

---

case 1

Program in case exception does not occur

```
1.  class TestFinallyBlock{
2.    public static void main(String args[]){
3.    try{
4.     int data=25/5;
5.     System.out.println(data);
6.    }
7.    catch(NullPointerException e){System.out.println(e);}
8.
9.    finally{System.out.println("finally block is always executed");}
10.
11.   System.out.println("rest of the code...");
12.   }
13. }
```

Test it Now

```
Output:5
       finally block is always executed
       rest of the code...
```

case 2

Program in case exception occured but not handled

```
1.  class TestFinallyBlock1{
2.    public static void main(String args[]){
3.    try{
4.     int data=25/0;
5.     System.out.println(data);
6.    }
7.    catch(NullPointerException e){System.out.println(e);}
8.
9.    finally{System.out.println("finally block is always executed");}
10.
11.   System.out.println("rest of the code...");
12.   }
13. }
```

Test it Now

```
Output:finally block is always executed
       Exception in thread main java.lang.ArithmeticException:/ by zero
```

case 3

Program in case exception occured and handled

```
1.  public class TestFinallyBlock2{
2.    public static void main(String args[]){
3.    try{
4.     int data=25/0;
5.     System.out.println(data);
6.    }
7.    catch(ArithmeticException e){System.out.println(e);}
8.
9.    finally{System.out.println("finally block is always executed");}
10.
11.   System.out.println("rest of the code...");
12.   }
13. }
```

Test it Now

```
Output:Exception in thread main java.lang.ArithmeticException:/ by zero
       finally block is always executed
       rest of the code...
```

2.6.THROW KEYWORD

The throw keyword is used to explictily throw an exception.

We can throw either checked or uncheked exception. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

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.

```
1.  public class TestThrow1{
2.
3.    static void validate(int age){
4.      if(age<18)
5.       throw new ArithmeticException("not valid");
6.      else
7.       System.out.println("welcome to vote");
8.    }
9.
10.   public static void main(String args[]){
```

```
11.      validate(13);
12.      System.out.println("rest of the code...");
13.  }
14. }
```

Test it Now

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

2.7.EXCEPTION PROPAGATION:

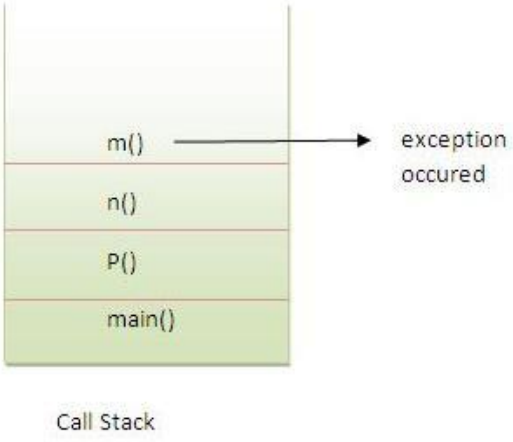
An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method,If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack.This is called exception propagation.

Program of Exception Propagation

```
1.  class TestExceptionPropagation1{
2.    void m(){
3.      int data=50/0;
4.    }
5.    void n(){
6.      m();
7.    }
8.    void p(){
9.     try{
10.     n();
11.    }catch(Exception e){System.out.println("exception handled");}
12.   }
13.   public static void main(String args[]){
14.    TestExceptionPropagation1 obj=new TestExceptionPropagation1();
15.    obj.p();
16.    System.out.println("normal flow...");
17.   }
18. }
```

Test it Now

```
Output:exception handled
       normal flow...
```



In the above example exception occurs in m() method where it is not handled,so it is propagated to previous n() method where it is not handled, again it is propagated to p() method where exception is handled.

Exception can be handled in any method in call stack either in main() method,p() method,n() method or m() method.

Program which describes that checked exceptions are not propagated

```
1.  class TestExceptionPropagation2{
2.    void m(){
3.      throw new java.io.IOException("device error");//checked exception
4.    }
5.    void n(){
6.      m();
7.    }
8.    void p(){
9.     try{
10.      n();
11.    }catch(Exception e){System.out.println("exception handeled");}
12.   }
13.   public static void main(String args[]){
14.    TestExceptionPropagation2 obj=new TestExceptionPropagation2();
15.    obj.p();
16.    System.out.println("normal flow");
17.   }
18. }
```

Test it Now

```
Output:Compile Time Error
```

2.8.THROWS KEYWORD

The **throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

SYNTAX OF THROWS KEYWORD:

```
1. void method_name() throws exception_class_name{
2.  ...
3. }
```

QUE) WHICH EXCEPTION SHOULD WE DECLARE?

Ans) checked exception only, because:

- **unchecked Exception:** under your control so correct your code.
- **error:** beyond your control e.g. you are unable to do anything if there occurs VirtualMachineError or StackOverflowError.

ADVANTAGE OF THROWS KEYWORD:

Now Checked Exception can be propagated (forwarded in call stack).

Program which describes that checked exceptions can be propagated by throws keyword.

```
1. import java.io.IOException;
2. class Simple{
3.   void m()throws IOException{
4.     throw new IOException("device error");//checked exception
5.   }
6.   void n()throws IOException{
7.     m();
8.   }
9.   void p(){
```

```
10.   try{
11.    n();
12.   }catch(Exception e){System.out.println("exception handled");}
13.  }
14.  public static void main(String args[]){
15.   Simple obj=new Simple();
16.   obj.p();
17.   System.out.println("normal flow...");
18.  }
19. }
```

```
Output:exception handled
       normal flow...
```

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

There are two cases:

1. **Case1:**You caught the exception i.e. handle the exception using try/catch.
2. **Case2:**You declare the exception i.e. specifying throws with the method.

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.

```
1.  import java.io.*;
2.  class M{
3.   void method()throws IOException{
4.    throw new IOException("device error");
5.   }
6.  }
7.
8.
9.  class Test{
10.   public static void main(String args[]){
11.    try{
12.     Test t=new Test();
13.     t.method();
14.    }catch(Exception e){System.out.println("exception handled");}
15.
16.    System.out.println("normal flow...");
17.  }
18. }
```

```
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 does not occur

```
1. import java.io.*;
2. class M{
3.  void method()throws IOException{
4.   System.out.println("device operation performed");
5.  }
6. }
7.
8.
9. class Test{
10.   public static void main(String args[])throws IOException{//declare exception
11.    Test t=new Test();
12.    t.method();
13.
14.    System.out.println("normal flow...");
15.  }
16. }
```

```
Output:device operation performed
       normal flow...
```

B)Program if exception occurs

```
1. import java.io.*;
2. class M{
3.  void method()throws IOException{
4.   throw new IOException("device error");
5.  }
6. }
7.
8.
9. class Test{
10.   public static void main(String args[])throws IOException{//declare exception
11.    Test t=new Test();
12.    t.method();
13.
14.    System.out.println("normal flow...");
15.  }
16. }
```

```
Output:Runtime Exception
```

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.