UNIT-IV Interface, Package and Exception Handling

1. What is an Interface?

An interface is a blue print of a class. An *interface* is a way of describing what classes should do, without specifying how they should do it. In Java, an *interface* is not a class but a set of requirements for the class that we want to conform to the interface.

The interface is a mechanism to achieve fully abstraction in java. There can be only abstract methods in the interface. It is used to achieve fully abstraction and multiple inheritance in Java.

Why we use Interface ?

1. It is used to achieve fully abstraction.
2. By interface, we can support the functionality of multiple inheritance.
3. It can be used to achieve loose coupling.

Properties of Interface

- An interface is implicitly abstract. So we no need to use the abstract keyword when declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.
- All the data members of interface are implicitly public static final.

How interface is different from class ?

1. We cannot instantiate an interface.
2. An interface does not contain any constructors.
3. All methods in an interface are abstract.
4. An interface cannot contain instance fields. Interface only contains public static final variables.
5. An interface is can not extended by a class; it is implemented by a class.
6. An interface can extend multiple interfaces. That means interface support multiple inheritance

2. Defining and implementing interfaces.

The **interface** keyword is used to declare an interface.

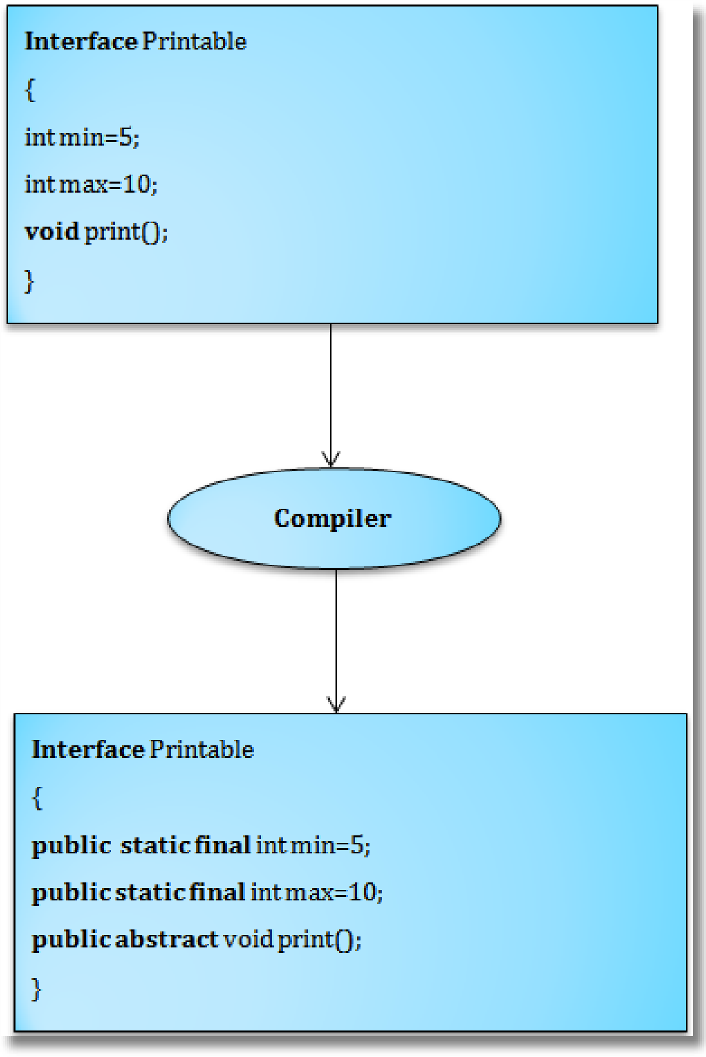
```
interface InterfaceName
{
 datatype variablename=value;
 //Any number of final, static fields.
returntype methodName(list of parameters
//Any number of abstract method declarations
}
```

In the above syntax **Interface** is a keyword interface name can be user defined name the default signature of variable is public static final and for method is public abstract. Compiler will be added implicitly public static final before data members and public abstract before method.

E.g.

```
interface printable
{
int min=5;
int max=10;
void print();
}
```

The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.



```
Interface Printable
{
int min=5;
int max=10;
void print();
}
```

```
Interface Printable
{
public static final int min=5;
public static final int max=10;
public abstract void print();
}
```

Implementing Interfaces:

A class uses the **implements** keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

The following is the syntax of implementing interface.

```
class ClassName implements InterfaceName
{
 returnType methodName(List of Parameters)
 {
   // method body
 }
}
```

E.g.

```
class PrintTest implements Printable
{
 public void print()
 {
 System.out.println("Implementation of print() method");
 }
}
```

Interface Methods

Every interface method is always public and abstract whether we are declaring or not. Hence the following declarations are equals inside interface.

1. void print();
2. public void print();
3. abstract void print();
4. public abstract void print();

Interface Variables

- An interface can contain variables also. The main purpose of interface variables is to define constants.
- Every interface variable is always public, static and final by default whether we are declaring or not.

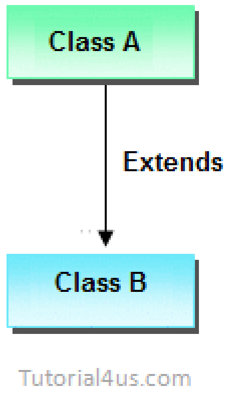
- For every interface variables compulsory we should perform initialization at the time of declaration otherwise we will get compile time error. E.g. int i; //invalid

Rules for implementation interface

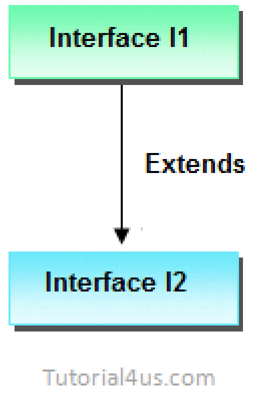
- A class can implement more than one interface at a time.
- A class can extend only one class, but implement many interfaces.
- An interface can extend another interface, similarly to the way that a class can extend another class.

Relationship between class and Interface

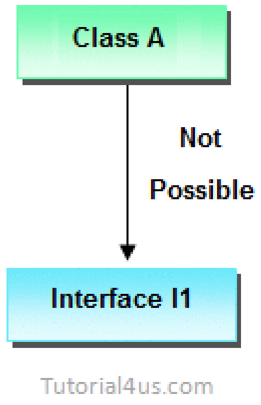
- Any class can extends another class



- Any Interface can extends another Interface.



- Any class can Implements another Interface



- Any Interface can not extend or Implements any class.

3. Inner Classes

If one class is existing within another class is known as inner class or nested class

Syntax

```
class  Outerclass_name
{
.....
.....

class  Innerclass_name1
{
.....
.....
}
}
```

The main purpose of using inner class

To provide more security by making those inner class properties specific to only outer class but not for external classes.

To make more than one property of classes private properties.

Private is a keyword in java language, it is preceded by any variable that property can be access only within the class but not outside of it (provides more security).

If more than one property of class wants to make as private properties than all can capped under private inner class.

```
class  Outerclass_name
{
private  class Innerclass_name
{
.....
.....   //private properties
}
}
```

Note: No outer class made as private class otherwise this is not available for JVM at the time of execution.

There are two types of nested classes: *static* and *non-static*.

A static nested class is one that has the **static** modifier applied. Because it is static, it must access the members of its enclosing class through an object. That is, it cannot refer to members of its enclosing class directly. Because of this restriction, static nested classes are seldom used.

The most important type of nested class is the *inner* class. An inner class is a non-static nested class. It has access to all of the variables and methods of its outer class and may refer to them directly in the same way that other non-static members of the outer class do.

Rules to access properties of inner classes

Inner class properties can be accessed in the outer class with the object reference but not directly.

Outer class properties can be access directly within the inner class.

Inner class properties can't be accessed directly or by creating directly object.

Note: In special situation inner class property can be accessed in the external class by creating special objects with the reference of its outer class.

Example

```
class   A //outer class
{
void  fun1()
{
System.out.println("Hello fun1()");
B  ob=new  B();
ob.x=10
System.out.println("x= "+ob.x);
ob.fun2();
}
void fun3()   // outer class fun3()
{
System.out.println("Hello fun3()");
}
class  B  // inner class
{
int  x;    // inner class variable
void fun2()       //inner class fun2()
{
System.out.println("Hello fun2()");
fun3(); //outer class properties can be access directly
}
}
}
class  C // external class
{
void fun3()
{
System.out.println("Hello fun3()");
}
}

class IncDemo
{
public static void main(String args[])
{
A  oa=new  A();
oa.fun1();
C   oc=new   C();
oc.fun3();
}
}
```

Output

```
Hello fun1()
X=10
Hello fun2()
Hello fun3()
```

1. Creating Accessing and using Packages

Package is a collection of classes, interfaces and sub-packages.

The purpose of package concept is to provide common classes and interfaces for any program. In other words if we want to develop any class or interface which is common for most of the java programs than such common classes and interfaces must be place in a package.

Packages in Java are the way to organize files when a project has many modules. Same like we organized our files in Computer. For example we store all movies in one folder and songs in other folder, here also we store same type of files in a particular package for example in awt package have all classes and interfaces for design GUI components.

Advantage of package

1. Package is used to categorize the classes and interfaces so that they can be easily maintained
2. Application development time is less, because reuse the code
3. Application memory space is less (main memory)
4. Application execution time is less.
5. Application performance is enhance (improve)
6. Redundancy (repetition) of code is minimized
7. Package provides access protection.
8. Package removes naming collision.

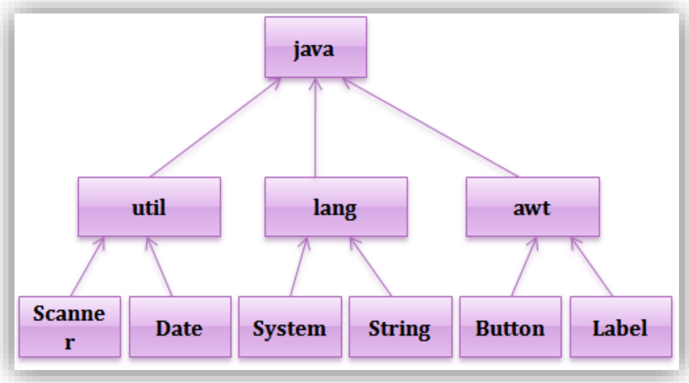
Type of package

Package are classified into two types which are given below.

1. Predefined or built-in package
2. User defined package

Predefined or built-in package

These are the package which are already designed by the Sun Microsystem and supply as a part of java API, every predefined package is collection of predefined classes, interfaces and sub-package.



User defined package

If any package is design by the user is known as user defined package. User defined package are those which are developed by java programmer and supply as a part of their project to deal with common requirement.

How to create user defined packages.

To create a new package java provides a keyword "**package**". While creating a new package we should follow some rules these rules are:

1. package statement should be the first statement of any package program
2. Package program should not contain any main class (that means it should not contain any main())
3. Every package program should be save either with public class name or public Interface name.

The following is the syntax of creating package.

Syntax:

```
package <packagename>;
```

E.g.

```
package mypack;
```

The following is the simple example of package.

Simple.java

```
package arithmetic;
public class Simple
{
        public void add(int x, int y)
        {
         System.out.println("Addition is : "+(x+y));
}
public void sub(int x, int y)
{
 System.out.println("Substraction is : "+(x-y));
}
public void multi(int x, int y)
{
 System.out.println("Multiplication is : "+(x*y));
}
public void division(int x, int y)
{
 System.out.println("division is : "+(x/y));
}
}
```

How to compile the package.

Syntax:

```
javac –d directoryPath FileName.java
```

In the above syntax the **"-d"** switch specifies the destination where to store the generated .class file. You can use any existing directory name. If you want to keep the package within the same directory you can use **.(dot)**.

E.g.

```
javac –d . Simple.java
```

Accessing and using the packages.

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

1. Using import package.*;
2. Using import package.ClassName;
3. Using Fully Qualified Name

1. **Using import package.*;**

If we use package.* then all the classes and interfaces of this package will be accessible but not sub-packages.

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

AccessPack.java

```
import arithmetic.*;
class AccessPack
{
        public static void main(String[] args)
        {
                Simple s=new Simple();
                s.add(10,20);
                s.sub(10,20);
                s.multi(10,20);
                s.division(10,20);
        }
}
```

If we compile and run AccessPack.java file it will display following output.

javac AccessPack.java

java AccessPack

O/P

```
        Addition is : 30
        Substraction is : -10
        Multiplication is : 200
        division is : 0
```

2. **Using import package.ClassName;**

If we import package.ClassName then only declared class of this package will be accessible.

```
import arithmetic.Simple;
class AccessPack
{
        public static void main(String[] args)
        {
                Simple s=new Simple();
                s.add(10,20);
                s.sub(10,20);
                s.multi(10,20);
                s.division(10,20);
        }
}
```

If we compile and run AccessPack.java file it will display following output.

javac AccessPack.java

java AccessPack

O/P

```
        Addition is : 30
        Substraction is : -10
        Multiplication is : 200
        division is : 0
```

1. **Using Fully Qualified Name**

If we use fully qualified name then only declared class of this package will be accessible. Now there is no need to import statement. But we 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 both the packages contain Date class.

E.g.

```
class AccessPack
{
public static void main(String[] args)
{
arithmetic.Simple s=new arithmetic.Simple();
                s.add(10,20);
                s.sub(10,20);
                s.multi(10,20);
                s.division(10,20);
        }
}
```

Sub-Packages

Package inside the package is called the sub-package. It should be created to categorized the package further.

Let's take an example, Sun Microsystems has defined a package named java that contains many classes like String, System, Reader, Writer, Socket etc. These classes represents a particular group. E.g. Reader and Writer classes are for I/O operations, Socket and ServerSocket classes for networking and so on. So SUN Microsystems has sub-categorized the java package into sub-packages such as java.lang, java.io, java.net etc.

The following is the syntax of creating sub-package.

Syntax:

```
package <packagename>.<subpackagename>;
```

E.g.

```
package maths.arithmetic;
```

The following is the example of creating the subpackage.

```
package maths.arithmetic;
public class SimpleMath
{
public void add(int x, int y)
{
System.out.println("Addition : "+(x+y));
}
}
import maths.arithmetic.SimpleMath;
class AccessSubPack
{
        public static void main(String[] args)
        {
                SimpleMath s=new SimpleMath();
                s.add(10,25);
        }
}
```

```
javac -d . SimpleMath.java
javac AccessSubPack.java
java AccessSubPack
```

O/P

```
Addition : 35
```

Exception Handling (5)

Exception handling is a mechanism that enables programs to detect and handle errors before they occur.

The **exception handling in java** is one of the powerful mechanisms to handle the runtime errors so that normal flow of the application can be maintained.

1. **What is error? Types of errors.**

Many Times a Program has to face some errors An Error is an Situation when a Compiler either doesn't Execute statements or either Compiler will Produce Wrong Result .

1. Syntax Error or Compile Time Error:-

The Compile Time Error or Syntax are denoted by compiler at the time of compilation The Compile Time Errors include Missing Semicolon, missing parenthesis , used undefined variable etc These are all Compile Time Errors

2. Run Time Error:-

Run Time Errors are Caught by Interpreter or at the time of Execution when we Execute out Program the Error is Encountered Run Time Error includes operations those are not possible but a user trying to do them .At that Situation error may occurred Like Divide a Number by zero, Accessing Element of Array Which Doesn't Exist Trying to Open a File that doesn't Exists. All these are Examples of Run Time Errors.

3. Logical Error:- These types of Error doesn't halt or Stops the Execution of Java Program but they produce wrong Results But there is no any syntax missing or either any Operation which is not possible but his type of Error is occurred when these is some mistake in logic of user Like he wants to Multiply two Numbers and he puts Addition Mark instead of Multiplication then it is called as Logical Error.

2. **What is exception?**

Exception is a condition which is Responsible for Occurrence of Error like Divide by Zero is a condition that never be possible So we can call it an Exception which halts or stops the Execution of program.

An exception is an event that may cause abnormal termination of the program during its execution.

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.

Exception handling doesn't mean repairing an exception. We have to define alternative way to continue rest of the program normally this defining alternative way is nothing but **Exception Handling.**

Advantage of Exception Handling

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

The main purpose of exception handling is graceful termination of program.

Exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

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

Types of Exception

1) Scenario where ArithmeticException occurs

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

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

```
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.

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

```
String s=null;
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.

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

```
String s="abc";
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:

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

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

3. **Catching exception and exception handling**

Exception handling in Java is accomplished by using five keywords

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

1. **try block**

In Java, the code that may generate (or throw) an exception is enclosed in a try block.

It is one of the block in which we write the block of statements which causes executions at run time in other words try block always contains problematic statements.

Important points about try block

1. If any exception occurs in try block then program control comes out to the try block and executes appropriate catch block.
2. After executing appropriate catch block, even through we use run time statement, program control never goes to try block to execute the rest of the statements.
3. Each and every try block must be immediately followed by catch block that is no intermediate statements are allowed between try and catch block.

```
try
{
 .......
}
catch(ExceptionClassName ref)
{
}
```

4. Each and every try block must contains at least one catch block. But it is highly recommended to write multiple catch blocks for generating multiple user friendly error messages.
5. One try block can contains another try block that is nested or inner try block can be possible.

2. **Catch block**

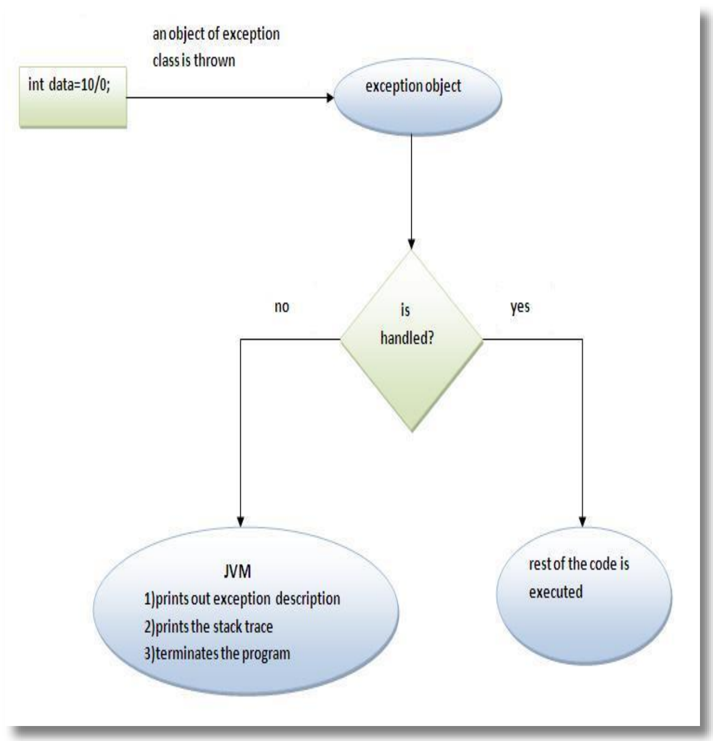
Java catch block is used to handle the Exception. It must be used after the try block only.

You can use multiple catch block with a single try.

Important points about catch block

1. Catch block will execute provided or exception occurs in try block.
2. It is highly recommended to write multiple catch blocks for generating multiple user friendly error messages to make our application strong.
3. At any point of time only one catch block will execute out of multiple catch blocks.
4. In programmatically in the catch block as a Java programmer we declare an object of sub class and it will be internally referenced by JVM.

```
class ExceptionDemo
{
public static void main(String[] args)
{
int a=10, ans=0;
try
{
ans=a/0;
}
catch (Exception e)
{
System.out.println("Denominator not be zero");
}
}
}
```


int data=10/0;
an object of exception class is thrown
exception object
is handled?
no
yes
JVM
1)prints out exception description
2)prints the stack trace
3)terminates the program
rest of the code is executed

* ***try with multiple catch blocks***

It is possible to define try with multiple catch blocks.

The way of handling an exception is varied from exception to exception. Because a single try block may contain more than one exception. Hence it is highly recommended to define a separate catch block for every exception.

It means we should define try with multiple catch blocks.

The following is the syntax of defining try with multiple catch blocks.

```
Syntax:
try
{
  ---------;  --------;  --------;
}
catch(ExceptionType1)
{
}
catch(ExceptionType2)
{
}
```

The following example demonstrates the use of try block with multiple catch blocks.

```
class ExceptionTest1
{
public static void main(String[] args)
{
int totalMarks=400;
int x=10;
int numbers[]={101,102,103,104,105};
try
{
float percentage=totalMarks/x;
int rollNo=numbers[5];
System.out.println("Percentage : "+percentage);
System.out.println("Roll No : "+rollNo);
}
catch(ArithmeticException ae)
{
 ae.printStackTrace();
}
catch(ArrayIndexOutOfBoundsException ae1)
{
 ae1.printStackTrace();
}
System.out.println("Hi");
}
}
```

O/P

```
javac ExceptionTest1.java
java ExceptionTest1
java.lang.ArrayIndexOutOfBoundsException: 5
        at ExceptionTest1.main(ExceptionTest1.java:11)
Hi
```

Important Note:

1. If try with multiple catch blocks presents then the order of catch blocks is very important. It should be child to parent by mistake if we are taking from parent to child we will get **CE: Exception XXX has already been caught.**

2. At a time only one Exception is occurred and at a time only one catch block is executed.

* ***Finally block***

It is one of the block in which we write the block of statements which will relinquish (released or close or terminate) the resource (file or database) where data store permanently.

It is never recommended to maintain cleanup code such as closing a file, closing connection etc. inside the try block because there is no guarantee for the execution of every statement inside the try always.

It is not recommended to place the cleanup code within the catch block because it only executed if there is an exception in try block.

We require a place to maintain a cleanup code which should be executed always irrespective of whether exceptions raised or not or whether the exception handled or not handled.

Such type of best place is nothing but the finally block.

```
Syntax:
try
{
  ---------;  --------;  --------;
}
catch(ExceptionType1)
{
}
finally
{
  // Cleanup code
}
```

Example of finally block

```
class TestFinallyBlock
        {
  public static void main(String args[]){
  try
                {
     int data=25/0;
                 System.out.println(data);
                }
        catch(ArithmeticException e)
                {
                        System.out.println(e);
                }
        finally
                {
                        System.out.println("finally
block is always executed");
                }
        System.out.println("rest of the code...");
 }
}
```

O/P

```
javac TestFinallyBlock.java
java TestFinallyBlock
java.lang.ArithmeticException: / by zero
finally block is always executed
rest of the code...
```

The main objective of finally block is to maintain cleanup code.

Important points about finally block

1. Finally block will execute compulsory
2. Writing finally block is optional.
3. For each try block there can be zero or more catch blocks, but only one finally block.
4. It is highly recommended to the java programmer to write on finally block for the entire java program
5. In some of the circumstances one can also write try and catch block in finally block.

**

4. **Creating User Defined Exception or Java Custom Exception**

If you are creating your own exceptions it is known as customized exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

By the help of custom exception, you can have your own exception and message.

To create customized exception Java provides a keyword called as “**throw**”.

To create user defined exceptions first you must create a class which extends Exception class. In which class you should define a parameterized constructor used to receive a error message as a String type. Call the super class constructor by passing the error message as a parameter.

The following is the syntax of creating user defined exception class.