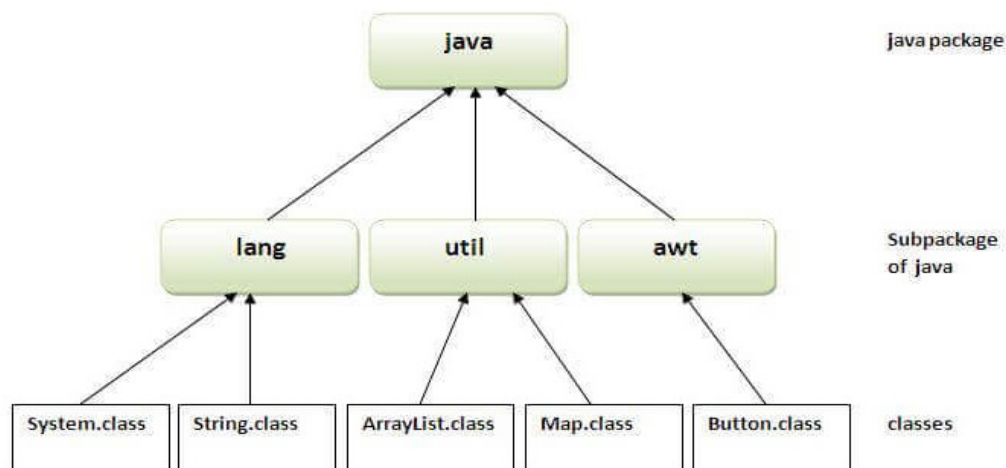


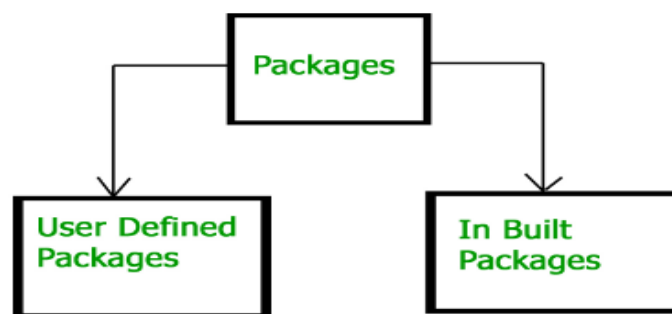
# Unit 3 : Package, String and Exception Handling

## Que 1: Explain package in detail.

- 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.



### Types of packages:



### Built-in Packages

These packages consist of a large number of classes which are a part of Java API. Some of the commonly used built-in packages are:

- 1) **java.lang**: Contains language support classes (e.g. classes which define primitive data types, math operations). This package is automatically imported.
- 2) **java.io**: Contains classes for supporting input / output operations.
- 3) **java.util**: Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.
- 4) **java.applet**: Contains classes for creating Applets.
- 5) **java.awt**: Contains classes for implementing the components for graphical user interfaces (like button , ; menus etc).
- 6) **java.net**: Contains classes for supporting networking operations.

### **User-defined packages**

- These are the packages that are defined by the user.
- First we create a directory myPackage (name should be same as the name of the package).
- Then create the MyClass inside the directory with the first statement being the package names.

#### **Syntax:-**

```
package nameOfPackage;
```

- Name of the package must be same as the directory under which this file is saved

### **Simple example of java package**

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

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

### **How to compile java package**

javac -d . Simple.java

- The -d switch specifies the destination where to put the generated class file.

### **How to run java package program**

**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.

### **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.

### **Que 2: Explain Access Protection in java**

- The access modifiers in Java specify the accessibility or scope of a field, method, constructor, or class.
- We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

| Accessibility Location<br>Access Specifier | Same Class | Same Package |                 | Other Package |                 |
|--|------------|--------------|-----------------|---------------|-----------------|
|  |            | Child class  | Non-child class | Child class   | Non-child class |
| Public                                     | Yes        | Yes          | Yes             | Yes           | Yes             |
| Protected                                  | Yes        | Yes          | Yes             | Yes           | No              |
| Default                                    | Yes        | Yes          | Yes             | No            | No              |
| Private                                    | Yes        | No           | No              | No            | No              |

There are four types of Java access modifiers

**Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

### Example

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is a compile-time error.

```
class A{
    private int data=40;
    private void msg(){System.out.println("Hello java");}
}
```

```
public class Simple{
```

```

public static void main(String args[]){
    A obj=new A();
    System.out.println(obj.data);//Compile Time Error
    obj.msg();//Compile Time Error
}
}

```

### **Default:**

- If you don't use any modifier, it is treated as default by default.
- The default modifier is accessible only within package.
- It cannot be accessed from outside the package.
- It provides more accessibility than private. But, it is more restrictive than protected, and public.

### **Example of default access modifier**

- In this example, we have created two packages pack and mypack.
- We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

```

//save by A.java
package pack;
class A{
    void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.*;
class B{

```

```

public static void main(String args[]){
    A obj = new A();//Compile Time Error
    obj.msg();//Compile Time Error
}
}

```

### **Protected:**

- The access level of a protected modifier is within the package and outside the package through child class.
- If you do not make the child class, it cannot be accessed from outside the package.

### **Example of default access modifier**

- In below example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

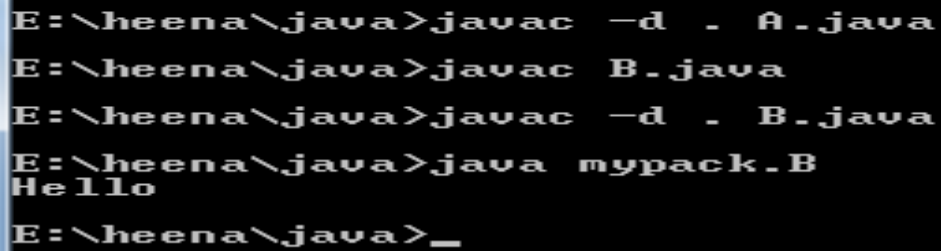
```

//save by A.java
package pack;
public class A
{
    protected void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.*;

class B extends A{
    public static void main(String args[]){
        B obj = new B();
    }
}

```

```
    obj.msg();  
  }  
}
```



```
E:\heena\java>javac -d . A.java  
E:\heena\java>javac B.java  
E:\heena\java>javac -d . B.java  
E:\heena\java>java mypack.B  
Hello  
E:\heena\java>_
```

### Public:

- The access level of a public modifier is everywhere.
- It can be accessed from within the class, outside the class, within the package and outside the package.
- The public access modifier is accessible everywhere.
- It has the widest scope among all other modifiers.

### Example of public access modifier

//save by A.java

```
package pack;
```

```
public class A{
```

```
    public void msg(){System.out.println("Hello");}
```

```
}
```

```
//save by B.java
```

```
package mypack;
```

```
import pack.*;
```

```
class B{
```

```
public static void main(String args[]){
```

```
    A obj = new A();
```

```
    obj.msg();
```

```
}
```

```
}
```

### **Que 3 : Explain java .lang Package**

- java.lang is a special package, as it is imported by default in all the classes that we create.
- There is no need to explicitly import the lang package. It contains the classes that form the basic building blocks of Java.:

### **Que 4 : Explain java.lang.Object Class**

- The java.lang.Object class is the root of the class hierarchy.
- Every class has Object as a superclass.
- All objects, including arrays, implement the methods of this class.

The Object class provides many methods. They are as follows:

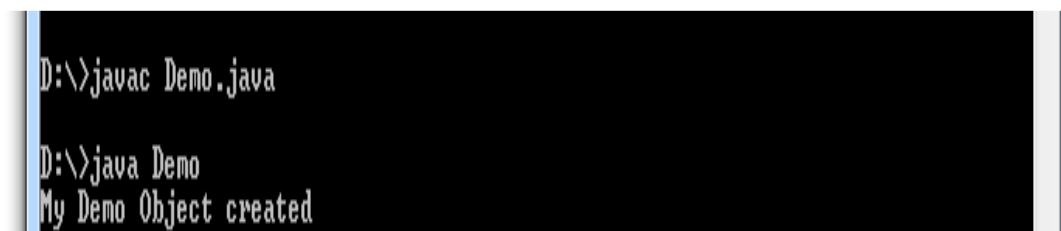
| Method | Description |
|--------|-------------|
|--------|-------------|



|  |   |
|--|---|
| public final Class getClass()                              | returns the Class class object of this object. The Class class can further be used to get the metadata of this class. |
| public int hashCode()                                      | returns the hashcode number for this object.  |
| public boolean equals(Object obj)                          | compares the given object to this object.   |
| protected Object clone() throws CloneNotSupportedException | creates and returns the exact copy (clone) of this object.  |
| public String toString()                                   | returns the string representation of this object.   |

### Example

```
class Demo {
    public String toString()
    {
        return "My Demo Object created";
    }
    public static void main(String args[])
    {
        System.out.println(new Demo());
    }
}
```




```
D:\>javac Demo.java
D:\>java Demo
My Demo Object created
```

```
class Demo {

    public static void main(String args[])
    {
        Demo d=new Demo();
```

```
System.out.println(d);  
}}
```



```
D:\>javac Demo.java  
D:\>java Demo  
Demo@173a10f  
D:\>
```

#### Que 4 : Explain Wrapper classes in Java

- The **wrapper class in Java** provides the mechanism *to* convert primitive into object and object into primitive.
- **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically.
- The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

#### Use of Wrapper classes in Java

##### Change the value in Method:

- Java supports only call by value.
- So, if we pass a primitive value, it will not change the original value.
- But, if we convert the primitive value in an object, it will change the original value.

##### Serialization:

- If we have a primitive value, we can convert it in objects through the wrapper classes.

##### Synchronization:

- Java synchronization works with objects in Multithreading.

##### java.util package:

- The java.util package provides the utility classes to deal with objects.
- The eight classes of the java.lang package are known as wrapper classes in Java.
- The list of eight wrapper classes are given below:

| Primitive Type | Wrapper class |
|----------------|---------------|
|----------------|---------------|

|         |                  |
|---------|------------------|
| boolean | <u>Boolean</u>   |
| char    | <u>Character</u> |
| byte    | <u>Byte</u>      |
| short   | <u>Short</u>     |
| int     | <u>Integer</u>   |
| long    | <u>Long</u>      |
| float   | <u>Float</u>     |
| double  | <u>Double</u>    |

### Autoboxing

- The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.

### Wrapper class Example: Primitive to Wrapper(Object)

```
public class JavaExample{
    public static void main(String args[]){
        //Converting int primitive into Integer
        object
        int num=100;
        Integer obj=Integer.valueOf(num);

        System.out.println(obj);
    }
}
```

**Output:**

100

## Unboxing

- The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing.
- It is the reverse process of autoboxing.
- Since Java 5, we do not need to use the `intValue()` method of wrapper classes to convert the wrapper type into primitives.

## Wrapper class Example: Wrapper to Primitive

```
//Java program to convert object into primitives
//Unboxing example of Integer to int
public class JavaExample{
    public static void main(String args[]){
        //Creating Wrapper class object
        Integer obj = new Integer(100);

        //Converting the wrapper object to primitive
        int num = obj.intValue();

        System.out.println(obj);
    }
}
```

## Output:

100

## Creating Wrapper Objects

- To create a wrapper object, use the wrapper class instead of the primitive type.
- To get the value, you can just print the object:

## Example

```
public class Main
{
    public static void main(String[] args)
    {
        Integer myInt = 5;
        Double myDouble = 5.99;
```

```
Character myChar = 'A';  
System.out.println(myInt);  
System.out.println(myDouble);  
System.out.println(myChar);  
}  
}
```

### **OUTPUT**

5  
5.99  
A

- Since you're now working with objects, you can use certain methods to get information about the specific object.
- For example, the following methods are used to get the value associated with the corresponding wrapper object: `intValue()`, `byteValue()`, `shortValue()`, `longValue()`, `floatValue()`, `doubleValue()`, `charValue()`, `booleanValue()`.

### **Example**

```
public class Main  
{  
    public static void main(String[] args)  
    {  
        Integer myInt = 5;  
        Double myDouble = 5.99;  
        Character myChar = 'A';  
        System.out.println(myInt.intValue());  
        System.out.println(myDouble.doubleValue());  
        System.out.println(myChar.charValue());  
    }  
}
```

}

- Another useful method is the `toString()` method, which is used to convert wrapper objects to strings.
- In the following example, we convert an `Integer` to a `String`, and use the `length()` method of the `String` class to output the length of the "string":

#### **Example**

```
public class Main {  
    public static void main(String[] args) {  
        Integer myInt = 100;  
        String myString = myInt.toString();  
        System.out.println(myString.length());  
    }  
}
```

#### **Que 5: Explain String class in java**

- Java `String` class provides a lot of methods to perform operations on strings such as `compare()`, `concat()`, `equals()`, `split()`, `length()`, `replace()`, `compareTo()`, `intern()`, `substring()` etc.

#### **Java String class methods**

- The `java.lang.String` class provides a lot of methods to work on string.
- By the help of these methods, we can perform operations on string such as trimming, concatenating, converting, comparing, replacing strings etc.

#### **important methods of String class.**

#### **Java String `toUpperCase()` and `toLowerCase()` method**

- The java string toUpperCase() method converts this string into uppercase letter
- string toLowerCase() method into lowercase letter.

```
public class Testmethodofstringclass{
    public static void main(String args[]){

        String s="Sachin";
        System.out.println(s.toUpperCase());//SACHIN
        System.out.println(s.toLowerCase());//sachin
        System.out.println(s);//Sachin(no change in original)
    }
}
```

#### **Java String trim() method**

- The string trim() method eliminates white spaces before and after string.

```
public class Testmethodofstringclass1{
    public static void main(String args[]){

        String s=" Sachin ";
        System.out.println(s);// Sachin
        System.out.println(s.trim());//Sachin
    }
}
```

#### **Java String charAt() method**

- The string charAt() method returns a character at specified index.

```
public class Testmethodofstringclass3{
    public static void main(String args[]){

        String s="Sachin";
        System.out.println(s.charAt(0));//S
        System.out.println(s.charAt(3));//h
    }
}
```

#### **Java String valueOf() method**

- The string `valueOf()` method converts given type such as int, long, float, double, boolean, char and char array into string.

```
int a=10;
String s=String.valueOf(a);
System.out.println(s+10);
```

**Output:**

1010

### **Java String replace() method**

- The string `replace()` method replaces all occurrence of first sequence of character with second sequence of character.

```
String s1="Java is a programming language. Java is a platform.
Java is an Island.";
String replaceString=s1.replace("Java","Kava");
System.out.println(replaceString);
```

**Output:**

Kava is a programming language. Kava is a platform. Kava is an Island.

### **Que 5: Explain StringBuffer Class in java**

- The `StringBuffer` class is used for representing changing strings.
- we can use `StringBuffer` to append, reverse, replace, concatenate and manipulate Strings or sequence of characters.
- Corresponding methods under `StringBuffer` class are respectively created to adhere to these functions.

### **Important Constructors of StringBuffer class**

| Constructor | Description |
|-------------|-------------|
|-------------|-------------|



|                            |   |
|----------------------------|---|
| StringBuffer()             | creates an empty string buffer with the initial capacity of 16.       |
| StringBuffer(String str)   | creates a string buffer with the specified string.                    |
| StringBuffer(int capacity) | creates an empty string buffer with the specified capacity as length. |

**Table 6.6** Methods of StringBuffer Class

| Method name with signature                            | Method details  |
|---|---|
| int capacity()  | Returns the current capacity of the storage available for characters in the buffer. When the capacity is approached, the capacity is automatically increased. |
| StringBuffer append(String str)                       | Appends String argument to the buffer   |
| StringBuffer replace (int sindx,int eIndx,String str) | The characters from start to end are removed and the string is inserted at that position  |
| StringBuffer reverse()                                | Reverses the buffer character by character  |
| Char charAt(int index)                                | Returns the character at the specified index  |
| Void setCharAt(int indx,char c)                       | Sets the specified character at the specified index   |

### **Example**

```

class demo
{
    public static void main(String[] args)
    {
        StringBuffer sb=new StringBuffer("gujarat");

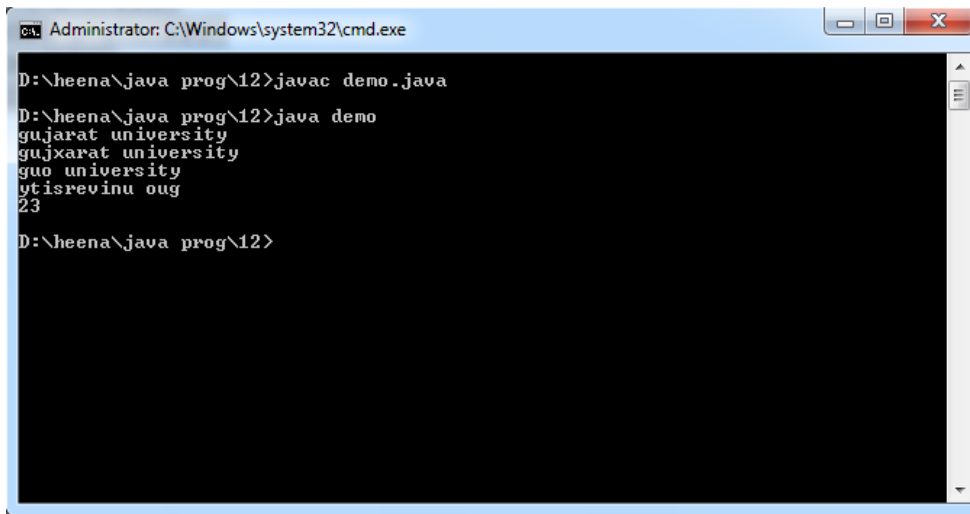
        System.out.println(sb.append(" university"));
        System.out.println(sb.insert(3,'x'));
        System.out.println(sb.replace(2,8,"o"));
        System.out.println(sb.reverse());
        System.out.println(sb.capacity());

    }
}

```

}

## OUTPUT



```
Administrator: C:\Windows\system32\cmd.exe
D:\heena\java prog\12>javac demo.java
D:\heena\java prog\12>java demo
gujarat university
gujxarat university
guo university
ytisrevinu oug
23
D:\heena\java prog\12>
```

### **Que 6 : Explain Exception Handling in Java**

- In Java, an exception is an event that disrupts the normal flow of the program.
- 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.
- 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:

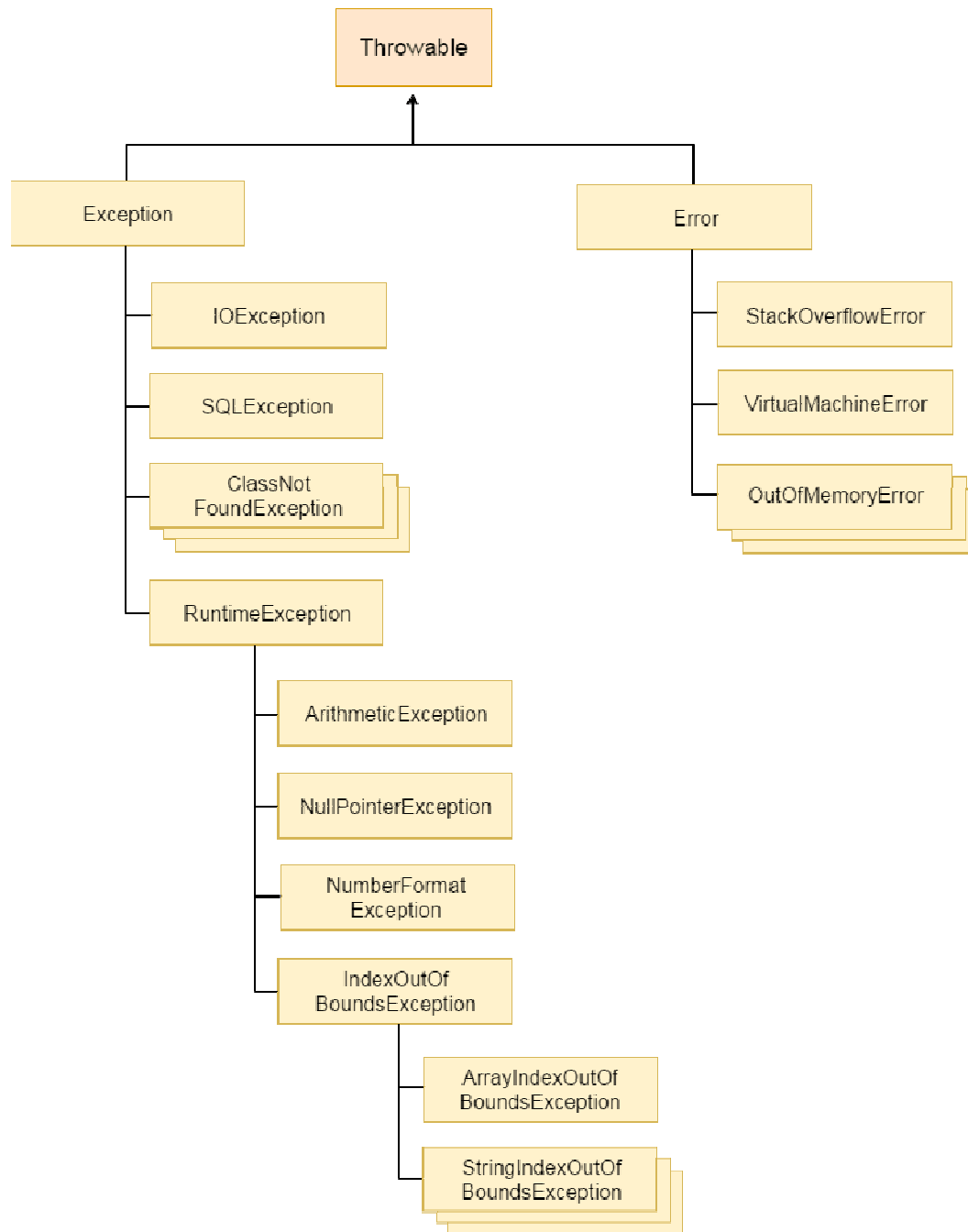
```
statement 1;
statement 2;
```

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

### **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

- 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.

### Que 6: Explain EXCEPTION HANDLING TECHNIQUES

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

| Keyword | Description   |
|---------|---|
| try     | The try statement allows you to define a block of code to be tested for errors while it is being executed.  |
| catch   | The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.   |
| finally | The finally statement lets you execute code, after try...catch, regardless of the result:   |
| throw   | <p>The throw statement allows you to create a custom error.</p> <p>The throw statement is used together with an exception type. There are many exception types available in Java: <code>ArithmeticException</code>, <code>FileNotFoundException</code>, <code>ArrayIndexOutOfBoundsException</code>, <code>SecurityException</code>, etc:</p> |
| 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.   |

### Syntax of Java try-catch

```

Try
{
//code that may throw an exception
}
catch(Exception_class_Name ref)
{
}

```

### Example

```
public class TryCatchExample2 {  
  
    public static void main(String[] args) {  
        try  
        {  
            int data=50/0; //may throw exception  
        }  
        //handling the exception  
        catch(ArithmeticException e)  
        {  
            System.out.println(e);  
        }  
        System.out.println("rest of the code");  
    }  
}
```

**Compile by:** javac TryCatchExample2.java

**Run by:** java TryCatchExample2

java.lang.ArithmeticException: / by zero  
rest of the code

### Syntax of Java try-catch-finally

```
try {  
    // Protected code  
} catch (ExceptionType1 e1) {  
    // Catch block  
} catch (ExceptionType2 e2) {  
    // Catch block  
} catch (ExceptionType3 e3) {  
    // Catch block  
}finally {  
    // The finally block always executes.  
}
```

### Example

```

class TestFinallyBlock1{
    public static void main(String args[]){
        try{
            int data=25/0;
            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**

Compile by: javac TestFinallyBlock1.java

Run by: java TestFinallyBlock1

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

at TestFinallyBlock1.main(TestFinallyBlock1.java:4)  
finally block is always executed

### **Syntax throw**

throw exception;

### **Example**

```

public class Main {
    static void checkAge(int age) {
        if (age < 18) {
            throw new ArithmeticException("Access denied - You must be
at least 18 years old.");
        } else {
            System.out.println("Access granted - You are old enough!");
        }
    }
}

public static void main(String[] args) {

```



```
    checkAge(15);  
  }  
}
```

**OUTPUT**

```
Exception in thread "main" java.lang.ArithmeticException: Access
denied - You must be at least 18 years old.
    at Main.checkAge(Main.java:4)
    at Main.main(Main.java:12)
```

### Syntax of java throws

```
return_type method_name() throws exception_class_name{
//method code
}
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);
        }
    }
}
```

### Output:

java.io.IOException: IOException Occurred

### Que 7: User Defined Exception

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

- Java custom exceptions are used to customize the exception according to user need.

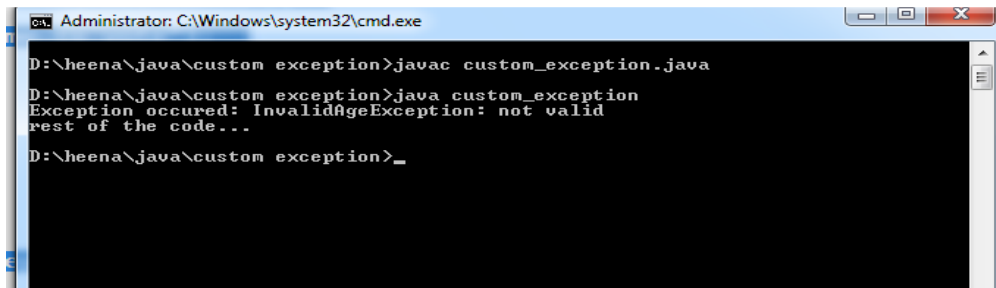
### Example

```
class InvalidAgeException extends Exception
{
    InvalidAgeException(String s)
    {
        super(s);
    }
}
class custom_exception
{
    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 occurred: "+m);
        }

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

## Output



```
Administrator: C:\Windows\system32\cmd.exe
D:\heena\java\custom exception>javac custom_exception.java
D:\heena\java\custom exception>java custom_exception
Exception occurred: InvalidAgeException: not valid
rest of the code...
D:\heena\java\custom exception>_
```