## What is Eclipse?

In the context of computing, Eclipse is an integrated development environment (IDE) for developing applications using the Java programming language and other programming languages such as C/C++, Python, PERL, Ruby etc.

The Eclipse platform which provides the foundation for the Eclipse IDE is composed of plug-ins and is designed to be extensible using additional plug-ins. Developed using Java, the Eclipse platform can be used to develop rich client applications, integrated development environments and other tools. Eclipse can be used as an IDE for any programming language for which a plug-in is available.

The Java Development Tools (JDT) project provides a plug-in that allows Eclipse to be used as a Java IDE, PyDev is a plugin that allows Eclipse to be used as a Python IDE, C/C++ Development Tools (CDT) is a plug-in that allows Eclipse to be used for developing application using C/C++, the Eclipse Scala plug-in allows Eclipse to be used an IDE to develop Scala applications and PHPeclipse is a plug-in to eclipse that provides complete development tool for PHP.

**Downloading and installing**

The main page is <http://www.eclipse.org/> ; start there for everything.

Click on "downloads" in either the main panel or the navigation panel; this will bring you to a page where you can select either the Main Eclipse download site or one of several mirrors. I suggest that you get the "Latest Release." I'm using **Release 3.0.1**, which is the latest release at the time I'm writing this.

Installation is simple, but you must already have the Java SDK installed. Unzip the file and double-click **eclipse.exe**.

**Getting started**

The following is slightly modified from <http://www.eclipse.org/eclipse/faq/eclipse-faq.html#users_4>

**Helloworld**

**package** one;

**public** **class** helloworld {

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

System.***out***.println("Hello world");

}

}

1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Program 1. Write a Java program to demonstrate the creation of class for student information.**

**package** sss;

**import** java.util.Scanner;

**public** **class** student

{

**public** String vtuno;

**public** String fullName;

**public** **int** sem;

**public** String branchcode;

**public** String address;

**public** student()

{

Scanner scanner=**new** Scanner(System.***in***);

System.***out***.print("VTU NO:");

vtuno=scanner.nextLine();

System.***out***.print("FullName:");

fullName=scanner.nextLine();

System.***out***.print("Address:");

address=scanner.nextLine();

System.***out***.print("branch:");

branchcode=scanner.nextLine();

System.***out***.print("semester:");

sem=scanner.nextInt();

}

**public** **void** show()

{

System.***out***.println("Entered Data");

System.***out***.println("VTU No. "+vtuno);

System.***out***.println("FullName:"+fullName);

System.***out***.println("sem:"+sem);

System.***out***.println("Branch:"+branchcode);

System.***out***.println("Address:"+address);

}

**public** **static** **void** main(String[] args)

{

student std=**new** student();

std.show();

}

}

**Output:**

VTU NO:3pd15cs092

FullName:xyz

Address:kalaburagi

branch:cse

semester:6

Entered Data

VTU No. 3pd15cs092

FullName:xyz

sem:6

Branch:cse

Address:kalaburagi

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Program 2. Writ and execute a java program to calculate sum of series of natural numbers.**

1. **public** **class** SumOfNaturalNumber1
2. {
3. **public** **static** **void** main(String[] args)
4. {
5. **int** i, num = 10, sum = 0;
6. //executes until the condition returns true
7. **for**(i = 1; i <= num; ++i)
8. {
9. //adding the value of i into sum variable

sum = sum + i;

}

//prints the sum

System.out.println("Sum of First 10 Natural Numbers is = " + sum);

}

}

**Output:**

Sum from: 1

Sum up to: 10

Sum of Natural Numbers = 55

**Program 3: Write and execute java program to demonstrate the scope of variables.**

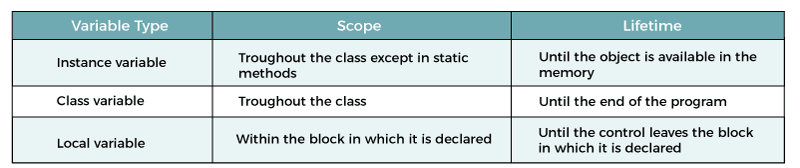
Scope of Variables in Java

In programming, **scope of variable** defines how a specific variable is accessible within the program or across classes. In this section, we will discuss the **scope of variables in Java**.

Scope of a Variable

In programming, a variable can be declared and defined inside a class, method, or block. It defines the scope of the variable i.e. the visibility or accessibility of a variable. Variable declared inside a block or method are not visible to outside. If we try to do so, we will get a compilation error. Note that the scope of a variable can be nested.

* We can declare variables anywhere in the program but it has limited scope.
* A variable can be a parameter of a method or constructor.
* A variable can be defined and declared inside the body of a method and constructor.
* It can also be defined inside blocks and loops.
* Variable declared inside main() function cannot be accessed outside the main() function



1. **public** **class** Demo
2. {
3. //instance variable
4. String name = "Andrew";
5. //class and static variable
6. **static** **double** height= 5.9;
7. **public** **static** **void** main(String args[])
8. {
9. //local variable

**int** marks = 72;

}

}

In [Java](https://www.javatpoint.com/java-tutorial), there are three types of variables based on their scope:

1. Member Variables (Class Level Scope)
2. Local Variables (Method Level Scope)

### Member Variables (Class Level Scope)

These are the variables that are declared inside the class but outside any function have class-level scope. We can access these variables anywhere inside the class. Note that the access specifier of a member variable does not affect the scope within the class. Java allows us to access member variables outside the class with the following rules:

| **Access Modifier** | **Package** | **Subclass** | **Word** |
| --- | --- | --- | --- |
| **public** | Yes | Yes | Yes |
| **protected** | Yes | Yes | No |
| **private** | No | No | No |
| **default** | Yes | No | No |

### Local Variables (Method Level Scope)

These are the variables that are declared inside a **method, constructor**, or **block** have a **method-level** or **block-level** scope and cannot be accessed outside in which it is defined. Variables declared inside a pair of curly braces **{}** have block-level scope.

**Program 3: Write and execute java program to demonstrate the scope of variables.**

**public** **class** VariableScopeExample1

1. {
2. **public** **static** **void** main(String args[])
3. {
4. **int** x=10;
5. {
6. //y has limited scope to this block only
7. **int** y=20;
8. System.out.println("Sum of x+y = " + (x+y));
9. }

//here y is unknown

y=100;

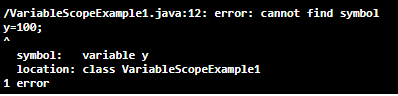
//x is still known

x=50;

}

}

**Output:**



We see that **y=100** is unknown. If you want to compile and run the above program remove or comment the statement **y=100.** After removing the statement, the above program runs successfully and shows the following output.

Sum of x+y = 30

There is another variable named an **instance** variable. These are declared inside a class but outside any method, constructor, or block. When an instance variable is declared using the keyword **static** is known as a static variable. Their scope is class level but visible to the method, constructor, or block that is defined inside the class.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Program 4: Write and execute a java program to find the biggest name in the array of strings.**

public class JavaLongestStringInStringArray {

public static String getLongestString(String[] array) {

int maxLength = 0;

String longestString = null;

for (String s : array) {

if (s.length() > maxLength) {

maxLength = s.length();

longestString = s;

}

}

return longestString;

}

public static void main(String[] args) {

String[] toppings = {"Cheese", "Pepperoni", "Black Olives"};

String longestString = getLongestString(toppings);

System.out.format("longest string: '%s'\n", longestString);

}

}

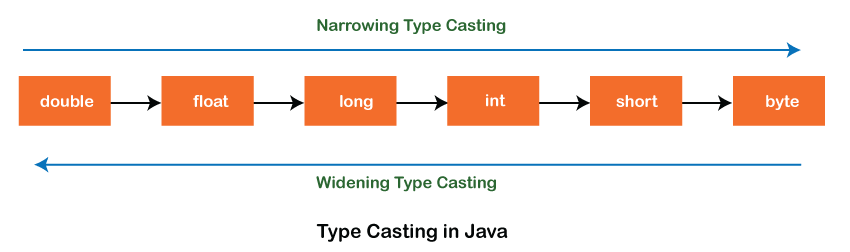
**Output: longest string: 'Black Olives'**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Program 5: Write and execute a java program to demonstrate data type casting.**

# Type Casting in Java

In Java, **type casting** is a method or process that converts a data type into another data type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer. In this section, we will discuss **type casting** and **its types** with proper examples.



## Type casting

Convert a value from one data type to another data type is known as **type casting**.

## Types of Type Casting

There are two types of type casting:

* Widening Type Casting
* Narrowing Type Casting

### Widening Type Casting

Converting a lower data type into a higher one is called **widening** type casting. It is also known as **implicit conversion** or **casting down**. It is done automatically. It is safe because there is no chance to lose data. It takes place when:Prime Ministers of India | List of Prime Minister of India (1947-2020)

* Both data types must be compatible with each other.
* The target type must be larger than the source type.

**byte** -> **short** -> **char** -> **int** -> **long** -> **float** -> **double**

For example, the conversion between numeric data type to char or Boolean is not done automatically. Also, the char and Boolean data types are not compatible with each other. Let's see an example.

**WideningTypeCastingExample.java**

**public** **class** WideningTypeCastingExample

1. {
2. **public** **static** **void** main(String[] args)
3. {
4. **int** x = 7;
5. //automatically converts the integer type into long type
6. **long** y = x;
7. //automatically converts the long type into float type
8. **float** z = y;
9. System.out.println("Before conversion, int value "+x);
10. System.out.println("After conversion, long value "+y);
11. System.out.println("After conversion, float value "+z);
12. }
13. }

**Output**

Before conversion, the value is: 7

After conversion, the long value is: 7

After conversion, the float value is: 7.0

In the above example, we have taken a variable x and converted it into a long type. After that, the long type is converted into the float type.

### Narrowing Type Casting

Converting a higher data type into a lower one is called **narrowing** type casting. It is also known as **explicit conversion** or **casting up**. It is done manually by the programmer. If we do not perform casting then the compiler reports a compile-time error.

**double** -> **float** -> **long** -> **int** -> **char** -> **short** -> **byte**

Let's see an example of narrowing type casting.

In the following example, we have performed the narrowing type casting two times. First, we have converted the double type into long data type after that long data type is converted into int type.

**NarrowingTypeCastingExample.java**

1. **public** **class** NarrowingTypeCastingExample
2. {
3. **public** **static** **void** main(String args[])
4. {
5. **double** d = 166.66;
6. //converting double data type into long data type
7. **long** l = (**long**)d;
8. //converting long data type into int data type
9. **int** i = (**int**)l;
10. System.out.println("Before conversion: "+d);
11. //fractional part lost
12. System.out.println("After conversion into long type: "+l);
13. //fractional part lost
14. System.out.println("After conversion into int type: "+i);
15. }
16. }

**Output**

Before conversion: 166.66

After conversion into long type: 166

After conversion into int type: 166

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Regular Laboratory exercises (for SEE)**

**1 Write and execute a JAVA program to demonstrate use of any five string functions. Use both parameterized constructors for passing string inputs.**

**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.

Java String is a powerful concept because everything is treated as a string if you submit any form in window based, web based or mobile application.

**Java StringBuffer class**

Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

### 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. |

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

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

String s="Sachin";

System.out.println(s.toUpperCase());//SACHIN

System.out.println(s.toLowerCase());//sachin

System.out.println(s);//Sachin(no change in original)

### What is mutable string

A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

### a) StringBuffer append() method

The append() method concatenates the given argument with this string.

**class** StringBufferExample{

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

StringBuffer sb=**new** StringBuffer("Hello ");

sb.append("Java");//now original string is changed

System.out.println(sb);//prints Hello Java

}

}

### 2) StringBuffer insert() method

The insert() method inserts the given string with this string at the given position.

**class** StringBufferExample2{

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

StringBuffer sb=**new** StringBuffer("Hello ");

sb.insert(1,"Java");//now original string is changed

System.out.println(sb);//prints HJavaello

}

}

### 3) StringBuffer replace() method

The replace() method replaces the given string from the specified beginIndex and endIndex.

**class** StringBufferExample3{

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

StringBuffer sb=**new** StringBuffer("Hello");

sb.replace(1,3,"Java");

System.out.println(sb);//prints HJavalo

}

}

### 4) StringBuffer delete() method

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

**class** StringBufferExample4{

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

StringBuffer sb=**new** StringBuffer("Hello");

sb.delete(1,3);

System.out.println(sb);//prints Hlo

}

}

### 5) StringBuffer reverse() method

The reverse() method of StringBuilder class reverses the current string.

**class** StringBufferExample5{

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

StringBuffer sb=**new** StringBuffer("Hello");

sb.reverse();

System.out.println(sb);//prints olleH

}

}

### 6) StringBuffer capacity() method

The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity\*2)+2. For example if your current capacity is 16, it will be (16\*2)+2=34.

**class** StringBufferExample6{

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

StringBuffer sb=**new** StringBuffer();

System.out.println(sb.capacity());//default 16

sb.append("Hello");

System.out.println(sb.capacity());//now 16

sb.append("java is my favourite language");

System.out.println(sb.capacity());//now (16\*2)+2=34

 i.e (oldcapacity\*2)+2

}

}

**package** sss;

**import** java.util.Scanner;

**public** **class** Stringops {

**public** **static** **void** main(String[] args)

{

StringBuffer s=**new** StringBuffer();

String str2;

Scanner input=**new** Scanner(System.***in***);

s.append("java programming ");

System.***out***.println("the current capacity of StringBuffer " +s.capacity());

System.***out***.println("initial String without append: " +s);

System.***out***.println("enter a String to append: ");

str2=input.nextLine();

s.append("" +str2);

System.***out***.println("initial String after append: " +s);

s.reverse();

System.***out***.println("initial String after reversing: " +s);

**int** i=0;

**do**

{

s.replace(i, i+1, s.substring(i,i+1).toUpperCase());

i=i+1;

}

**while**(i>0 && i<s.length());

System.***out***.println("after reversing:"+s);

}

}

**OUTPUT:**

the current capacity of StringBuffer 34

initial String without append: java programming

enter a String to append:

lab

initial String after append: java programming lab

initial String after reversing: bal gnimmargorp avaj

after reversing:BAL GNIMMARGORP AVAJ

1. **Write and execute a JAVA Program to demonstarate Inheritance.**

# Java - Inheritance

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

## extends Keyword

**extends** is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

**Syntax**

class Super {

.....

}

class Sub extends Super {

.....

.....

}

## Sample Code

Following is an example demonstrating Java inheritance. In this example, you can observe two classes namely Calculation and My\_Calculation.

Using extends keyword, the My\_Calculation inherits the methods addition() and Subtraction() of Calculation class.

**Types of inheritance in Java: Single, Multiple, Multilevel**

Below are Various types of inheritance in Java. We will see each one of them one by one with the help of examples and flow diagrams.

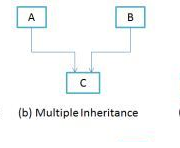
### 1) Single Inheritance

**Single inheritance** is damn easy to understand. When a class extends another one class only then we  call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a **parent class** of B and B would be  a **child class** of A.

### Single Inheritance

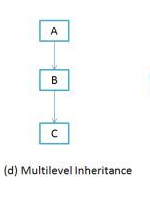
### 2) Multiple Inheritance

“**Multiple Inheritance**” refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with “multiple inheritance” is that the derived class will have to manage the dependency on two base classes.



### 3) Multilevel Inheritance

**Multilevel inheritance** refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of of B and B is a child class of A



**Program 6a :**

package sss;

class sup

{

int x; sup(int x)

{

this.x=x;

}

void display()

{

System.out.println("sup+x = " +x);

}

}

class supr extends sup

{

int y;

supr(int x,int y)

{

super(x);

this.y=y;

}

void display()

{

System.out.println("sup-x = "+x); System.out.println("sup-y = "+y);

}

}

class prog6a

{

public static void main(String args[])

{

supr s1 = new supr(100,200); s1.display();

}

}

# OUTPUT:

sup-x = 100

sup-y = 200

**b. Simple Program on Java for the implementation of Multiple inheritance using interfaces to calculate the area of a rectangle and triangle.**

# Java - Interfaces

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.

Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is similar to a class in the following ways −

* An interface can contain any number of methods.
* An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
* The byte code of an interface appears in a **.class** file.
* Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including −

* You cannot instantiate an interface.
* An interface does not contain any constructors.
* All of the methods in an interface are abstract.
* An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
* An interface is not extended by a class; it is implemented by a class.
* An interface can extend multiple interfaces.

**Program 6b**

package sss;

interface compute

{

double calculate();

}

class Rectangle implements compute

{

double l,b;

void getvalues()

{

l = 10.5F;

b=7.3F;

}

public double calculate()

{

return (l\*b);

}

}

class TRI extends Rectangle implements compute

{

public double calculate()

{

return (0.5\*b\*l);

}

}

class prog6b

{

public static void main(String[] args)

{

Rectangle R = new Rectangle();

R.getvalues();

System.out.println("Area of Rectangle = " + R.calculate());

TRI T =new TRI();

T.getvalues();

System.out.println("Area of triangle = " + T.calculate());

}}

**OUTPUT:**

Area of Rectangle = 76.65000200271606

Area of triangle = 38.32500100135803

1. **Write and execute a JAVA Program to demonstrate exception handling(both built-in and user-defined exceptions)**

{

curBalance=500;

}

void deposit()

{

Scanner s=new Scanner(System.in);

System.out.println("Enter the amount:");

amt=s.nextInt();

curBalance+=amt;

System.out.println("\nBalance: "+curBalance);

}

void withdraw()**Exception Handling in Java**

1. [Exception Handling](https://www.javatpoint.com/exception-handling-in-java)
2. [Advantage of Exception Handling](https://www.javatpoint.com/exception-handling-in-java#exceptionad)
3. [Hierarchy of Exception classes](https://www.javatpoint.com/exception-handling-in-java#exceptionhierarchy)
4. [Types of Exception](https://www.javatpoint.com/exception-handling-in-java#exceptiontypes)
5. [Scenarios where exception may occur](https://www.javatpoint.com/exception-handling-in-java#exceptionscenarios)

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.

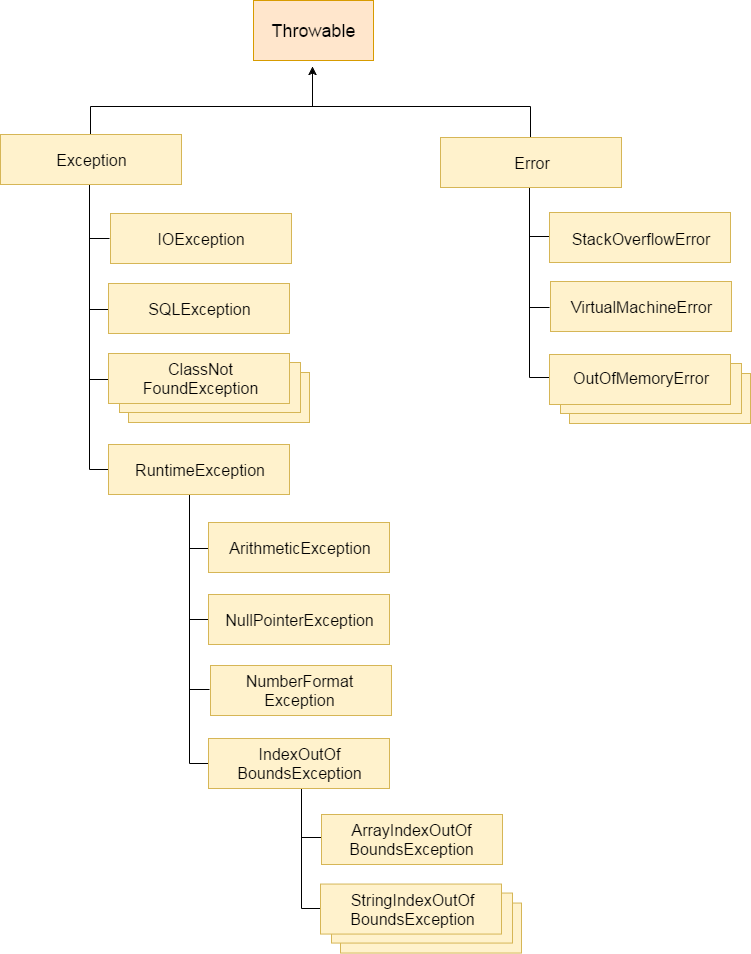
**Advantage of Exception Handling**

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 statement will be executed. That is why we use exception handling in java.

**Hierarchy of Java Exception classes**

****

**Program 3** :

package sss;

import java.util.\*;

class Account

{

int curBalance,amt;

public Account()

{

Scanner s=new Scanner(System.in);

System.out.println("\nEnter the amount:");

amt=s.nextInt();

try

{

if((curBalance-amt)<500) throw new LessBalanceException(amt);

curBalance-=amt;

System.out.println("\nBalance left: "+curBalance);

}

catch (LessBalanceException le)

{

System.out.println(le);

}

}

void checkBal()

{

System.out.println("\nBalance in your a/c: "+curBalance);

}

}

class LessBalanceException extends Exception

{

int am;

public LessBalanceException(int x)

{

am=x;

System.out.println("Less Balance : "+x);

}

public String toString()

{

return("You cannot with draw the amount,less balance"+am+ "Now");

}

}

class Main3

{

public static void main(String[] ar)

{

int ch;

Scanner s=new Scanner(System.in);

Account a=new Account();

while(true) {

System.out.println("1:Deposit\t2:Withdrawl\t3:Balance\t4:Exit\n"); System.out.println("Enter your choice:"); ch=s.nextInt();

switch(ch)

{

case 1: a.deposit(); break;

case 2: a.withdraw(); break;

case 3: a.checkBal(); break;

case 4: return;

default: System.out.println("Invalid choice\n"); return;

}

}

}}

**OUTPUT:**

1:Deposit 2:Withdrawl 3:Balance 4:Exit

Enter your choice:

1

Enter the amount:

300

Balance: 800

1:Deposit 2:Withdrawl 3:Balance 4:Exit

Enter your choice:

2

Enter the amount:

50

Balance left: 750

1:Deposit 2:Withdrawl 3:Balance 4:Exit

Enter your choice:

3

Balance in your a/c: 750

1:Deposit 2:Withdrawl 3:Balance 4:Exit

Enter your choice:

4

1. **Write and Execute a JAVA program to implement inheritance(single level and multilevel)**
2. **Write and execute a JAVA program to demonstrate polymorphism through method overloading.**

**package** sss;

**class** Pattern

{

// method without parameter

**public** **void** display() {

**for** (**int** i = 0; i < 10; i++) {

System.***out***.print("\*");

}

}

// method with single parameter

**public** **void** display(**char** symbol) {

**for** (**int** i = 0; i < 10; i++) {

System.***out***.print(symbol);

}

}

}

**class** Main {

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

Pattern d1 = **new** Pattern();

// call method without any argument

d1.display();

System.***out***.println("\n");

// call method with a single argument

d1.display('#');

}

}

**OUTPUT:**

\*\*\*\*\*\*\*\*\*\*

##########

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **Write and execute a JAVA program to demonstrate method overriding.**

**package** sss;

**class** Language {

**public** **void** displayInfo() {

System.***out***.println("Common English Language");

}

}

**class** Java **extends** Language {

@Override

**public** **void** displayInfo() {

System.***out***.println("Java Programming Language");

}

}

**class** Main {

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

// create an object of Java class

Java j1 = **new** Java();

j1.displayInfo();

// create an object of Language class

Language l1 = **new** Language();

l1.displayInfo();

}

}

**Output:**

Java Programming Language

Common English Language

1. **Write a Java applet program and required applet program and required HTML file to create banner applet.**