A way of viewing world

- an idea to illustrate the object-oriented programming concept with an example of a real-world situation.

Agents and Communities

- An object-oriented program is structured as a community of interacting agents, called objects. Where each object provides a service (data and methods) that is used by other members of the community.

Messages and Methods

- In object-oriented programming, every action is initiated by passing a message to an agent (object), which is responsible for the action.
- The receiver is the object to whom the message was sent.
- In response to the message, the receiver performs some method to carry out the request.
- Every message may include any additional information as arguments.

Responsibilities

- behaviors of an object described in terms of responsibilities.

Classes and Instances

- all objects are instances of a class.

Classes Hierarchies

- classes can be organized into a hierarchical inheritance structure. A child class inherits properties from the parent class

Method Binding, Overriding, and Exception

- In the class hierarchy, both parent and child classes may have the same method which implemented individually.
- the implementation of the parent is overridden by the child.
- Or a class may provide multiple definitions to a single method to work with different arguments (overloading).
- search for the method to invoke in response to a request (message) begins with the class
- If no suitable method is found, the search is performed in the parent class of it.
- The search continues up the parent class chain until either a suitable method is found or the parent class chain is exhausted.

- If a suitable method is found, the method is executed. Otherwise, an error message is issued.

Summary of Object-Oriented concepts

- OOP stands for Object-Oriented Programming
- OOP is a programming paradigm in which every program follows the concept of object.
- object-oriented programming paradigm core concepts
    - **Encapsulation**
    - **Inheritance**
    - **Polymorphism**
    - **Abstraction**

Encapsulation

- Encapsulation is the process of combining data and code into a single unit (object / class).
- In OOP, every object is associated with its data and code.
- In programming, data is defined as variables and code is defined as methods.
- The java programming language uses the *class* concept to implement encapsulation.

Inheritance

- Inheritance is the process of acquiring properties and behaviors from one object to another object or one class to another class.
- In inheritance, we derive a new class from the existing class.
- the new class acquires the properties and behaviors from the existing class.
- In the inheritance concept, the class which provides properties is called as parent class and the class which recieves the properties is called as child class.
- The parent class is also known as base class or supre class. The child class is also known as derived class or sub class.
- In the inheritance, the properties and behaviors of base class extended to its derived class, but the base class never receive properties or behaviors from its derived class.

```
class childclassname extends parentclassname
{
// body of child class
}
```

Polymorphism

- Expressing something in multiple forms
- Poly+morphs = many+forms
- In java polymorphism can be expressed using overloading and overriding
    - Overloading-
        1. Constructor overloading – a class having more than one constructor with different number of parameters
        2. Method overloading – a class having more than one method with same name and different number of parameters
    - Overriding - Method overriding – If a child class has methods with same name and same number of parameters as the parent class methods then the child class method is said to override the super class method.

Abstraction

- Abstraction is hiding the internal details and showing only essential functionality.
- In the abstraction concept, we do not show the actual implementation to the end user, instead we provide only essential things.
- Abstraction can be illustrated using abstract classes and interfaces.

Java buzzwords

- Java has many advanced features, a list of key features is known as Java Buzz Words.

- **Simple**
- **Secure**
- **Portable**
- **Object-oriented**
- **Robust**
- **Architecture-neutral (or) Platform Independent**
- **Multi-threaded**
- **Interpreted**
- **High performance**
- **Distributed**
- **Dynamic**

Simple

- Java programming language is very simple and easy to learn, understand, and code.
- Most of the syntaxes in java follow basic programming language C and object-oriented programming concepts are similar to C++.
- In a java programming language, many complicated features like pointers, operator overloading, structures, unions, etc. have been removed.

Secure

- java provides a feature "applet" which can be embedded into a web application. The applet in java does not allow access to other parts of the computer, which keeps away from harmful programs like viruses and unauthorized access.
- Java is said to be more secure programming language because it does not have pointers concept

Portable

- Portability is one of the core features of java which enables the java programs to run on any computer or operating system.
- For example, an application developed using java runs on a wide variety of CPUs, operating systems, and browsers connected to the Internet.

Object-oriented

- Java is said to be a pure object-oriented programming language.
- In java, everything is an object.
- It supports all the features of the object-oriented programming paradigm.

Robust

- Java is more robust because the java code can be executed on a variety of environments
- java has a strong memory management mechanism, dynamic memory allocation and automatic memory deallocation by garbage collector
- java is a strictly typed language
- it has a strong set of exception handling mechanism

Architecture-neutral

- Java program runs on systems with different architectures (32-bit or 64-bit)

Platform Independent

- Java follows "write once; run anywhere, any time, forever".
- The java provides JVM (Java Virtual Machine) to achieve architectural-neutral or platform-independent.
- The JVM allows the java program created using one operating system can be executed on any other operating system.

Multi-threaded

- Java supports multi-threading programming, which allows us to write programs that do multiple operations simultaneously.

Interpreted

- Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode.
- The byte code is interpreted to any machine code so that it runs on the native machine.

High performance

- Java provides high performance with the help of features like JVM, interpretation, and its simplicity.

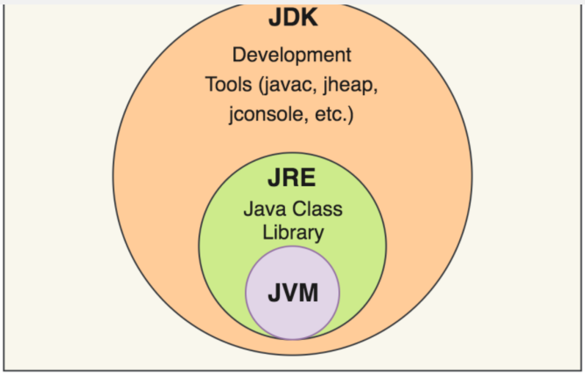
Distributed

- Java programming language supports TCP/IP protocols which enable the java to support the distributed environment of the Internet.
- Java also supports Remote Method Invocation (RMI), this feature enables a program to invoke methods across a network.

Dynamic

- Java is said to be dynamic because the java byte code may be dynamically updated on a running system and it has a dynamic memory allocation and deallocation (objects and garbage collector).

Data Type	Size	Range
byte	1 byte	Stores whole numbers from -2^7 and a maximum



JDK=JRE+DEVELOPMENT TOOLS

JRE=JVM+LIBRARY CLASSES

DATA TYPES

		-128 to 127	value of 2^{7}-1.
short	2 bytes	Stores whole numbers from -32,768 to 32,767	-2^{15} and a maximum value of 2^{15}-1.
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647	-2^{31} and a maximum value of 2^{31}-1.
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	-2^{63} and a maximum value of 263-1.
Float single precision	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits	-1.4e-0.45 to 3.4e+38
Double double precision	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits	-4.9e-324 to 1.8e+308
boolean	1 bit	Stores true or false values	
char	2 bytes	Stores a single character/letter or ASCII values	UNICODE

Type Conversion and Type Casting :

- when you assign a value of one primitive data type to another type.
- two types
    - **Widening Casting** (automatically) - converting a smaller type to a larger type size

```
int x = 9;

    double y = x;
```

    - **Narrowing Casting** (manually) - converting a larger type to a smaller size type

```
double x = 9.78;

    int y = (int) x;
```

Type Promotion Rules:

1. all byte, short, and char values are promoted to int
   Ex: byte a=4
   byte b=5
   byte c=a*b; [incompatible types]

   ‘a’& ‘b’ get automatically promoted to int whereas we are storing in byte so incompatabile.
   So type casting has to be done
   byte c= (byte) (a*b);

2.
    - if one operand is a long, the whole expression is promoted to long.

- If one operand is a float, the entire expression is promoted to float.
- If any of the operands is double, the result is double.

FIRST JAVA PROGRAM

```
class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello, World!");

    }

}
```

1. Save with .java extension
2. Can be saved using any name
3. Compiled as javac HelloWorld.java (if the file name is HelloWorld.java)
4. This creates a .class file called HelloWorld.class which contains the bytecode
5. The program is executed as java HelloWorld

NOTE: If you have saved program as First.java
Then compile as javac First.java
But run as java HelloWorld

ARRAYS

- Array is a collection of similar type of elements which has contiguous memory location.
- The elements of an array are stored in a contiguous memory location.
- Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Single Dimensional Array

Datatype arrayVar=**new** datatype[size];

int a[]=**new int**[5];

- We can declare, instantiate and initialize the java array
  int a[]={33,3,4,5};

Passing Array to a Method

//finding minimum element in an array

```
class MinArray{
 void min(int arr[]){
int min=arr[0];
// length is an attribute of an array which gives array length
for(int i=1;i<arr.length;i++)
if(min>arr[i])
  min=arr[i];
System.out.println(min);
}  }
class ArrayDemo1
{
public static void main(String args[])
{
MinArray ob=new MinArray();
int a[]={33,3,4,5};//declaring and initializing an array
ob.min(a);//passing array to method  }}
```

Multidimensional Array

```
int[][] arr=new int[3][3];
int arr[][]={{1,2,3},
              {2,4,5},
              {4,4,5}
              };
```

//Java Program to demonstrate the addition of two matrices in Java

```
class Testarray5{
public static void main(String args[]){
int a[][]={{1,3,4},{3,4,5}};
int b[][]={{1,3,4},{3,4,5}};
  int c[][]=new int[2][3];
  for(int i=0;i<2;i++){
for(int j=0;j<3;j++){
c[i][j]=a[i][j]+b[i][j];
System.out.print(c[i][j]+" ");
}
System.out.println();}
}}
```

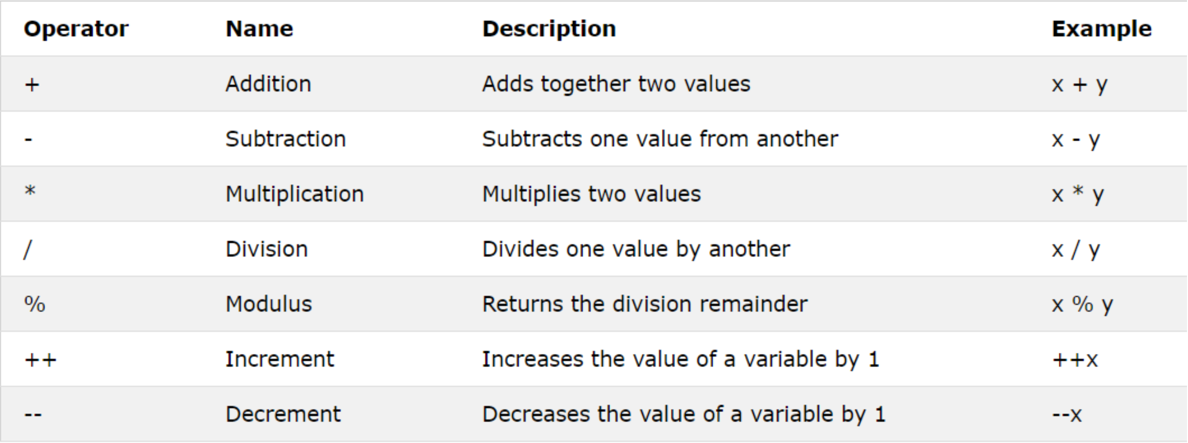
//Java Program to multiply two matrices

```
public class MatrixMultiplicationExample{
public static void main(String args[]){
int a[][]={{1,1,1},{2,2,2},{3,3,3}};
int b[][]={{1,1,1},{2,2,2},{3,3,3}};

int c[][]=new int[3][3];
for(int i=0;i<3;i++){
for(int j=0;j<3;j++){
c[i][j]=0;
for(int k=0;k<3;k++)
{
c[i][j]+=a[i][k]*b[k][j];
}
System.out.print(c[i][j]+" ");
}
 System.out.println();
}
}}
```

Java Operators

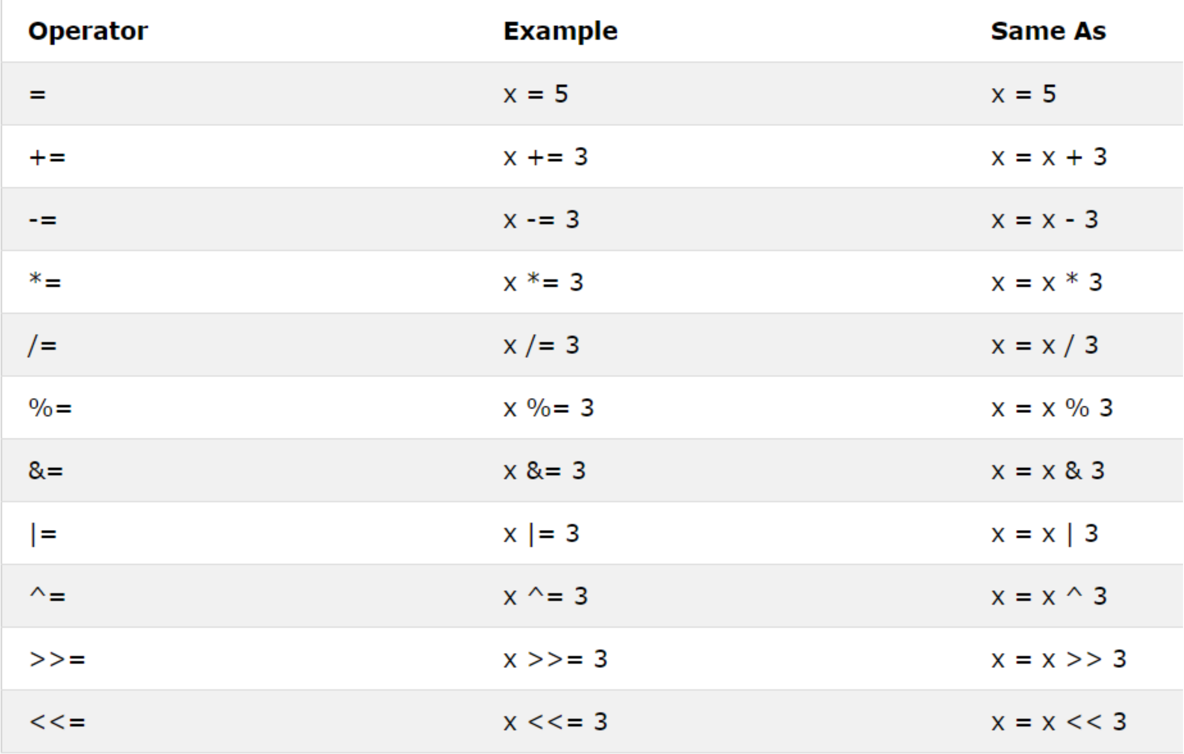
- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators

Arithmetic Operators



Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Assignment Operator

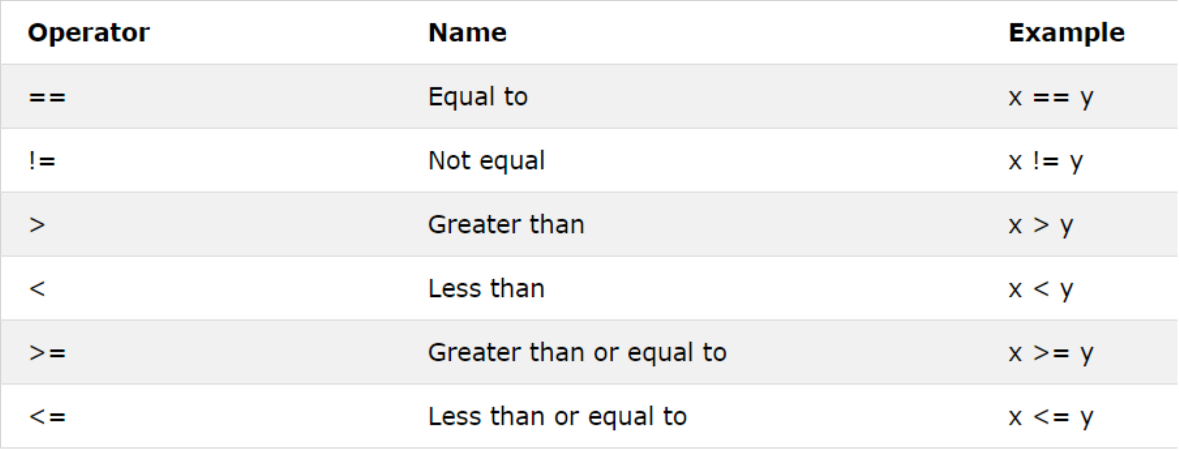


Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

a=a+1

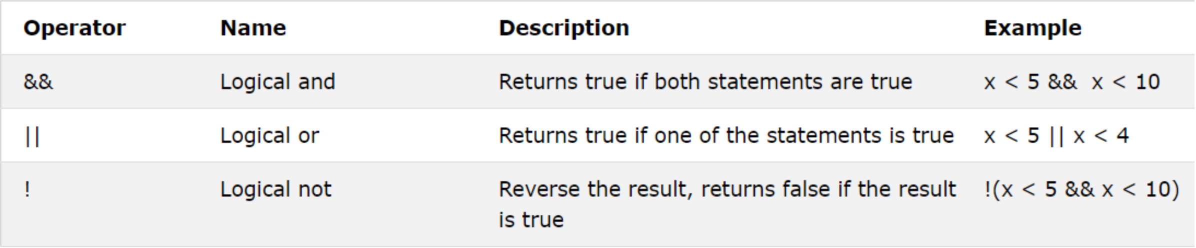
a+=1

Relational Operators



Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical Operators



Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Bitwise Operators

~	Unary bitwise complement
<<	Signed left shift
>>	Signed right shift
>>>	Unsigned right shift or right shift with zero fill
&	Bitwise AND
^	Bitwise exclusive OR
|	Bitwise inclusive OR

STRING HANDLING

- String **objects** represents sequence of characters.
- every string that we create is actually an object of type **String**.
- string objects are **immutable** that means once a string object is created it cannot be changed.

Creating a String object

1. Using a String literal

String s1 = "Hello Java";

2. Using new Keyword

String s1 = new String("Hello Java");

3.

- `String str= "Hello";`
- `String str2 = str;`

Concatenating String

i. Using **concat()** method
ii. Using + operator

Using concat() method

- `String s = "Hello";`
- `        String str = "Java";`
- `        String str1 = s.concat(str);`

```
Output: HelloJava
Str.concat(s)---JavaHello
```

Using + operator

- `String s = "Hello";`
- `        String str = "Java";`
- `        String str1 = s+str;`

-
  ```
  String str2 = "Java"+11;
  ```

-
  ```
  System.out.println(str1);
  ```

-
  ```
  System.out.println(str2);
  ```

```
Output:    HelloJava
          Java11
```

String Comparison

1. Using equals() method
2. Using == operator
3. By compareTo() method

Using equals() method

```
String s = "Hell";

      String s1 = "Hello";

      String s2 = "Hello";

      s1.equals(s2);    //true

      b =   s.equals(s1) ; // false
```

Using == operator

```
String s1 = "Java";

      String s2 = "Java";

      String s3 = new String ("Java");

    (s1 == s2);     //true

  (s1 == s3);      //false
```

compareTo() method

compareTo() method compares values and returns an integer value which tells if the string compared is less than, equal to or greater than the other string.

Syntax:

```
int compareTo(String str)

String s1 = "sree";

      String s2 = "Sree";

      String s3 = "sree";

      int a = s1.compareTo(s2)

      System.out.println(a);

      a = s1.compareTo(s3);     //return 0 because s1 == s3

      System.out.println(a);

      a = s2.compareTo(s1);

      System.out.println(a);
```

charAt() method

String charAt() function returns the character located at the specified index.

```
String str = "lastdaynightout";

        System.out.println(str.charAt(2));
```

equalsIgnoreCase() method

String equalsIgnoreCase() determines the equality of two Strings, ignoring their case (upper or lower case doesn't matter with this method).

```
String str = "java";

        System.out.println(str.equalsIgnoreCase("JAVA"));
```

Returns true

indexOf() method

String indexOf() method returns the index of first occurrence of a substring or a character.

indexOf() method has four override methods:

- int indexOf(String str): It returns the index within this string of the first occurrence of the specified substring.
- int indexOf(int ch, int fromIndex): It returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
- int indexOf(int ch): It returns the index within this string of the first occurrence of the specified character.
- int indexOf(String str, int fromIndex): It returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

```
String str="late night studying ";

       System.out.println(str.indexOf('u'));   //3rd form

   System.out.println(str.indexOf('t', 3));    //2nd form

        String subString="ght";

  System.out.println(str.indexOf(subString)); //1st form

System.out.println(str.indexOf(subString,3));   //4th form
```

length() method

String length() function returns the number of characters in a String.

```
String str = "Count me";

        System.out.println(str.length());// 8
```

replace() method

String replace() method replaces occurances of character with a specified new character.

```
String str = "Change me";

        System.out.println(str.replace('m','M'));
```

//Change Me

substring() method

String substring() method returns a part of the string.

1. public String substring(int begin);

2. public String substring(int begin, int end);

```
String str = "0123456789";

        System.out.println(str.substring(4));

        System.out.println(str.substring(4,7));
```

Output

```
456789
4567
```

toLowerCase() method

String toLowerCase() method returns string with all uppercase characters converted to lowercase.

```
String str = "ABCDEF";

        System.out.println(str.toLowerCase()); //abcdef
```

toUpperCase() method

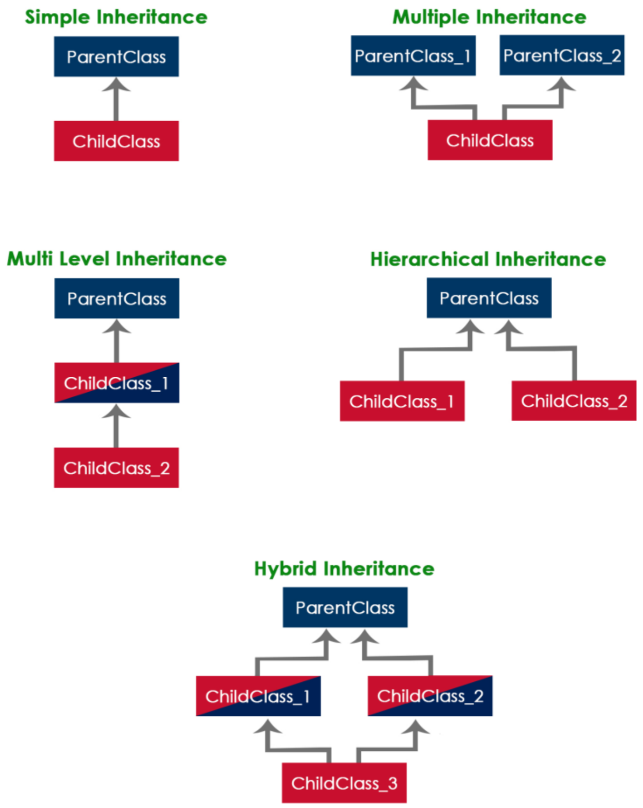
```
String str = "abcdef";

        System.out.println(str.toUpperCase());   // ABCDEF
```

toString()—that converts into string

INHERITANCE

- using the inheritance concept, we can use the existing features of one class in another class.
- code re-usability is the advantage
- In inheritance, we use the terms like parent class, child class, base class, derived class, superclass, and subclass.
  - The Parent class is the class which provides features to another class. The parent class is also known as Base class or Superclass.
  - The Child class is the class which receives features from another class. The child class is also known as the Derived Class or Subclass.

- five types of inheritances :
  - **Simple Inheritance (or) Single Inheritance**
  - **Multiple Inheritance (**java does not support multiple inheritance However, it provides an alternate with the concept of interfaces.)
  - **Multi-Level Inheritance**
  - **Hierarchical Inheritance**
  - **Hybrid Inheritance**



Syntax

class ChildClassName **extends** ParentClassName

```
{   ...
   //Implementation of child class
   ...
}
```

- A class extends only one class. Extending multiple classes is not allowed in java.

//Program to illustrate Single Inheritance

```
class ParentClass{ int a;
            void setData(int a) {
                        this.a = a;          }}
class ChildClass extends ParentClass{
            void showData() {
                        System.out.println("Value of a is " + a);          }}
```

```
public class SingleInheritance {
        public static void main(String[] args) {
                ChildClass obj = new ChildClass();
                obj.setData(100);
                obj.showData();    }}




// PROGRAM TO ILLUSTRATE MULTILEVEL INHERITANCE
class ParentClass{
        int a;
        void setData(int a) {
                this.a = a;          }}
class ChildClass extends ParentClass{
        void showData() {
                System.out.println("Value of a is " + a);         }}
class GrandChildClass extends ChildClass{
        void display() {
                System.out.println("Inside GrandChildClass!");}}
class MultipleInheritance {
        public static void main(String[] args) {
        GrandChildClass obj = new GrandChildClass();
                obj.setData(100);
                obj.showData();
                obj.display();       }}
```

Access Modifiers

- The access specifiers (also known as access modifiers) used to restrict the scope or accessibility of a class, constructor, variable, method or data member of class and interface.
- There are four access specifiers
    - **default (or) no modifier**
    - **public**
    - **protected**
    - **private**

- The public members can be accessed everywhere.
- The private members can be accessed only inside the same class.
- The protected members are accessible to every child class (same package or other packages).
- The default members are accessible within the same package but not outside the package.

Example

```
class ParentClass{
        int a = 10;
        public int b = 20;
        protected int c = 30;
        private int d = 40;
                void showData() {
                System.out.println("Inside ParentClass");
                System.out.println("a = " + a);
                System.out.println("b = " + b);
                System.out.println("c = " + c);
                System.out.println("d = " + d);
        }}
class ChildClass extends ParentClass{
        void accessData() {
                System.out.println("Inside ChildClass");
                System.out.println("a = " + a);
                System.out.println("b = " + b);
                System.out.println("c = " + c);
                //System.out.println("d = " + d);      // private member can't be accessed
        }}
public class AccessModifiersExample {
```

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

        ChildClass obj = new ChildClass();

        obj.showData();

        obj.accessData();       }}
```

Constructors in Inheritance

- The default constructor of a parent class called automatically by the constructor of its child class.
- That means when we create an object of the child class, the parent class constructor executed, followed by the child class constructor executed.
- Example:

```
class  A
{
  A()
{ System.out.println("in A");}}
class B extends A
{
  B()
{
System.out.println("in child class B");}}
class C extends B
{
  C()
{
System.out.println("in grand child C");}}

class ConstructorCallingDemo
{
public static void main(String args[])
{
 C ob=new C();
}}
```

➢ NOTE: if the parent class contains both default and parameterized constructor, then only the default constructor is called automatically by the child class constructor.

- To call parameterized constructor of parent class inside child class constructor we have to “super”keyword.

super keyword

- “Super” is a keyword used to refers to the parent class object.
- the super keyword is used for the following purposes.
    - **To refer parent class data members**
    - **To refer parent class methods**
    - **To call parent class constructor**

NOTE: The super keyword is used inside the child class only.

SUPER KEYWORD USES:

Advantages:

1. To solve the naming conflicts in the inheritance. When both parent class and child class have members (A. data members and also B. member functions) with the same name, then the super keyword is used to refer to the parent class version.
2. To invoke and initialize super class private members inside subclass constructor.

1. **A. Example (Parent class and child class with data members of same name)**

```
class ParentClass{
        int num = 10;  }
class ChildClass extends ParentClass{
                int num = 20;
        void showData() {
                System.out.println("Inside the ChildClass");
                System.out.println("ChildClass num = " + num);
                System.out.println("ParentClass num = " + super.num);
        }}
public class SuperKeywordExample {
        public static void main(String[] args) {
                ChildClass obj = new ChildClass();
```

```
                        obj.showData();
          //System.out.println("ParentClass num = " + super.num); //super can't be used here
}}
```

1. **B. Example (Parent class and child class with methods of same name)**

METHOD OVERRIDING:

- When both parent class and child class have methods with same signature (method name and same number and same order of parameters) then child class is said to override the parent class method.
  Example:

```
class Parent
{
 void fun()
{
   System.out.println("in parent class fun() method");
}}
class Child extends Parent
{
  void fun()
{
  System.out.println("in child class overridden fun() method");
}}
class MethodOverridingDemo
{
public static void main(String args[])
{
Child ob=new Child();
ob.fun();//invokes only the child overridden fun() method
}}
```

  Output: in child class overridden fun() method

- To invoke the super class fun() method we can use super keyword

```
class Parent
{
 void fun()
{
   System.out.println("in parent class fun() method");
}}
class Child extends Parent
{
  void fun()
{
  super.fun(); //invoking the super class overridden method
 System.out.println("in child class overridden fun() method");
}}
class MethodOverridingDemo
{
public static void main(String args[])
{
Child ob=new Child();
ob.fun();
}}
```

Output:

```
 in parent class fun() method
 in child class overridden fun() method
```

- the above program invokes the child class fun() method in which the parent class fun() method is invoked using super keyword.

2. Initializing super class private members inside child class constructor using super keyword

```
class Box
{
private double width,height,depth;
Box()
{  width=10;height=10;depth=10;}
Box(double w,double h,double d)
{    width=w;height=h;depth=d;    }
void volume()
{ System.out.println(width*height*depth);}
}
class Boxw extends Box
{
  double weight;
  Boxw()
{
  super();
  weight=10;}
Boxw(double w,double h,double d,double wt)
{
 //width=w;height=h;depth=d;  // private data access in childclass will give error
super(w,h,d);
 weight=wt;  }
void displayweight()
{

 System.out.println(weight); }
}

class SuperDemo3
{
 public static void main(String args[])
{
  Boxw ob1=new Boxw();
  ob1.volume();
ob1.displayweight();
Boxw ob2=new Boxw(1,2,3,4);
  ob2.volume();
  ob2.displayweight();}}
```

using final with inheritance

- ➔ final keyword can be used for variables,methods and classes.
- ➔ final variables cannot be changed i.e they become constants.
- ➔ final methods cannot be overridden.
- ➔ final classes cannot be inherited.

```
class A

{  final int a=10;}

class FinalDemo1

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

 A ob=new A();

   ob.a=100;}}
```

error: cannot assign a value to final variable a

```
   ob.a=100;

class A

{

final  void meth(){}

}

class B extends A
```

```
{
  void meth(){}
}
class FinalDemo2
{
  public static void main(String args[])
{
 B ob=new B();
   ob.meth();
}}
error: meth() in B cannot override meth() in A
  void meth(){}
       ^
  overridden method is final
1 error
final class A
{      }
class B extends A
{     }
class FinalDemo3
{  public static void main(String args[])
{ B ob=new B(); }}
error: cannot inherit from final A
class B extends A
            ^
1 error
```

Polymorphism

-> In java, polymorphism implemented using method overloading and method overriding.

Ad hoc polymorphism

- The ad hoc polymorphism is a technique used to define the same method with different implementations and different arguments.
- In a java programming language, ad hoc polymorphism carried out with a method overloading concept.
- In ad hoc polymorphism the method binding happens at the time of compilation.
- Ad hoc polymorphism is also known as compile-time polymorphism.
- Every function call binded with the respective overloaded method based on the arguments.

One more form of Method Overloading

```
class A
{
  void meth(int a){}
}
class B extends A{
  void meth(int a,int b){ }   }
class MethOverloadingDemo{
public static void main(String args[ ])
{ B ob=new B();
  ob.meth(10);
ob.meth(20,30);
}}
```

Pure polymorphism

- The pure polymorphism is a technique used to define the same method with the same arguments but different implementations.
- In a java programming language, pure polymorphism carried out with a method overriding concept.
- In pure polymorphism, the method binding happens at run time.

- Pure polymorphism is also known as run-time polymorphism.
- Every function call binding with the respective overridden method based on the object reference.

Super class reference variable referring to child class object

```
//Dynamic Method Dispatch or Run-time Polymorphism

class A

{

  void show()

{

  System.out.println("A says hi");

}

}

class B extends A

{

void show()

{

  System.out.println("B says hi");

}

}

class C extends A

{

void show()

{

  System.out.println("C says hi");

}

void disp()
{
System.out.println("C says hello too");
}
}
class DMD
{
public static void main(String args[])
{
A ob;
ob=new B();// super class reference variable referring to subclass object
ob.show();// the execution of the overidden method depends on the type of the object and not on type of the reference variable
ob=new C();
ob.show();
//ob.disp();// cannot access non-overidden methods of child class
}}
```

Abstract Class

- An abstract class is a class that created using abstract keyword.
- In other words, a class prefixed with abstract keyword is known as an abstract class.
- In java, an abstract class may contain abstract methods (methods without implementation) and also concrete methods (non-abstract methods) (methods with implementation).

Syntax

```
abstract class <ClassName>{
   ...
}
```

```
import java.util.*;
```

```
abstract class Shape {
        int length, breadth, radius;
        Scanner input = new Scanner(System.in);
        abstract void printArea();
}
class Rectangle extends Shape {
        void printArea() {
                System.out.print("Enter length and breadth: ");
                length = input.nextInt();
                breadth = input.nextInt();
                System.out.println("The area of Rectangle is: " + length * breadth);
        }}
class Triangle extends Shape {
        void printArea() {
                System.out.print("Enter Base And Height: ");
                length = input.nextInt();
                breadth = input.nextInt();
                System.out.println("The area of Triangle is: " + (length * breadth) / 2);
        }}
class Cricle extends Shape {
        void printArea() {
                System.out.print("Enter Radius: ");
                radius = input.nextInt();
                System.out.println("The area of Cricle is: " + 3.14f * radius * radius);
        }}

public class AbstractClassExample {
        public static void main(String[] args) {
                Rectangle rec = new Rectangle();
                rec.printArea();

                Triangle tri = new Triangle();
                tri.printArea();
                Cricle cri = new Cricle();
                cri.printArea();
        }
}
```

- The child class of an abstract class should compulsorily define the abstract methods of the abstract class.
- If the child class does not define then the child class also should be declared abstract and its child has to define the methods.

EX:

```
 abstract   class A
{   abstract void fun1();   }
 abstract class B extends A
{
}
class C extends B
{
  void fun1()
{ }
}
```

- An abstract class must be created with abstract keyword.
- An abstract class may contain abstract methods and non-abstract methods.
- An abstract class may contain final methods that cannot be overridden.
- An abstract class may contain static methods, but the abstract method can not be static.
- An abstract class may have a constructor that gets executed when the child class object created.
- An abstract method must be overridden by the child class, otherwise, it must be defined as an abstract class.
- An abstract class cannot be instantiated but can be referenced.

Object Class

- the Object class is the super most class of any class hierarchy.
- The Object class in the java programming language is present inside the **java.lang** package.
- Every class in the java programming language is a subclass of Object class by default.

Meth

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.
public final void notify()	wakes up single thread, waiting on this object's

	monitor.
public final void notifyAll()	wakes up all the threads, waiting on this object's monitor.
public final void wait(long timeout)throws InterruptedException	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait(long timeout,int nanos)throws InterruptedException	causes the current thread to wait for the specified milliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait()throws InterruptedException	causes the current thread to wait, until another

}

	thread notifies (invokes notify() or notifyAll() method).
protected void finalize()throws Throwable	is invoked by the garbage collector before object is being garbage collected.

Static keyword

The static keyword can be used for

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested class

1) Java static variable

If you declare any variable as static, it is known as a static variable.

- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.

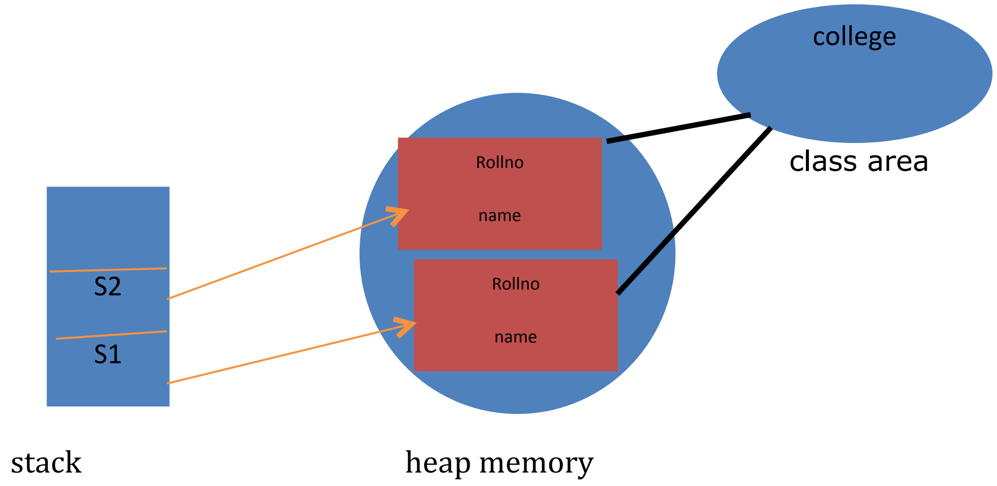
  It makes program **memory efficient** (i.e., it saves memory).

```
class Student{
    int rollno;
    String name;
    String college="MGIT";
}
```

- Suppose there are 500 students in the college, now all instance data members will get memory each time when the object is created.

- All students have unique rollno and name, so instance data member is good in such case.
- Here, "college" refers to the common property of all objects.
- If we make it static, this field will get the memory only once.

```
//Java Program to demonstrate the use of static variable
class Student{
   int rollno;//instance variable
   String name;  //instance variable
   static String college ="MGIT";//static variable
   //constructor
   Student(int r, String n){
   rollno = r;
   name = n;
   }
   //method to display the values
   void display (){System.out.println(rollno+" "+name+" "+college);}
}
 class TestStaticVariable1{
 public static void main(String args[]){
 Student s1 = new Student(111,"sree");
 Student s2 = new Student(222,"Adi");
 //we can change the college of all objects by the single line of code
 //Student.college="CBIT";
 s1.display();
 s2.display();
 }
}
```



2) Java static method

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class i.e without creating an object.
- A static method can access static data member and can change the value of it.

There are two main restrictions for the static method. They are:

1. The static method cannot use non static data member or call non-static method directly.
2. this and super cannot be used in static context.

```
//Java Program to demonstrate the use of a static method.

class Student{

    int rollno;

    String name;

    static String college = "MGIT";

    //static method to change the value of static variable

    static void change(){

    college = "CBIT";

//rollno=525; //non-static variable cannot be accessed from static method

    }

      Student(int r, String n){

    rollno = r;

    name = n;      }

    void display(){System.out.println(rollno+" "+name+" "+college);}

}
```

```
 class TestStaticMethod{

  public static void main(String args[]){

  Student.change();

// change();

  Student s1 = new Student(111,"sree");

  Student s2 = new Student(222,"Aadi");

        s1.display();

  s2.display();

     } }
```

Why is the Java main method static?

3) Java static block

- in order to initialize your **static variables**, you can declare a static block that gets executed exactly once, when the class is first loaded.

```
// Java program to demonstrate use of static blocks
class StaticBlockDemo
{
    // static variable
    static int a = 10;
    static int b;

    // static block
    static {
        System.out.println("Static block initialized.");
        b = a * 4;
    }
```

```
        public static void main(String[] args)
        {
          System.out.println("from main");
          System.out.println("Value of a : "+a);
          System.out.println("Value of b : "+b);
        }
      }
```

Forms of Inheritance

- The substitutability means that when a child class acquires properties from its parent class, the object of the parent class may be substituted with the child class object. For example, if B is a child class of A, anywhere we expect an instance of A we can use an instance of B.

Different forms of inheritance in java.

- Specialization
- Specification
- Construction
- Extension
- Limitation
- Combination

Specialization

- The subclass is a special case of the parent class.

Specification

- In this form of inheritance, the parent class just specifies which methods should be available to the child class but doesn't implement them.
- The java provides concepts like abstract and interfaces to support this form of inheritance.

Construction

- In this form of inheritance the child class may change the behaviour defined by the parent class (overriding).

Extension

- This is another form of inheritance where the child class may add its new properties.

Limitation

- In this form of inheritance the subclass restricts the inherited behaviour.

Combination

- In this inheritance the subclass inherits properties from multiple parent classes.
- Java does not support multiple inheritance type.

Benefits of Inheritance

- Inheritance helps in code reuse. The child class may use the code defined in the parent class without re-writing it.
- Inheritance can save time and effort as the main code need not be written again.
- An inheritance leads to less development and maintenance costs.
- In inheritance base class can decide to keep some data private so that it cannot be altered by the derived class.

Costs of Inheritance

- Inheritance decreases the execution speed due to the increased time and effort it takes while the program jumps through all the levels of overloaded classes.
- Inheritance makes the two classes (base and inherited class) get tightly coupled. This means one cannot be used independently of each other.
- The changes made in the parent class will affect the behaviour of child class too.
- The overuse of inheritance makes the program more complex.