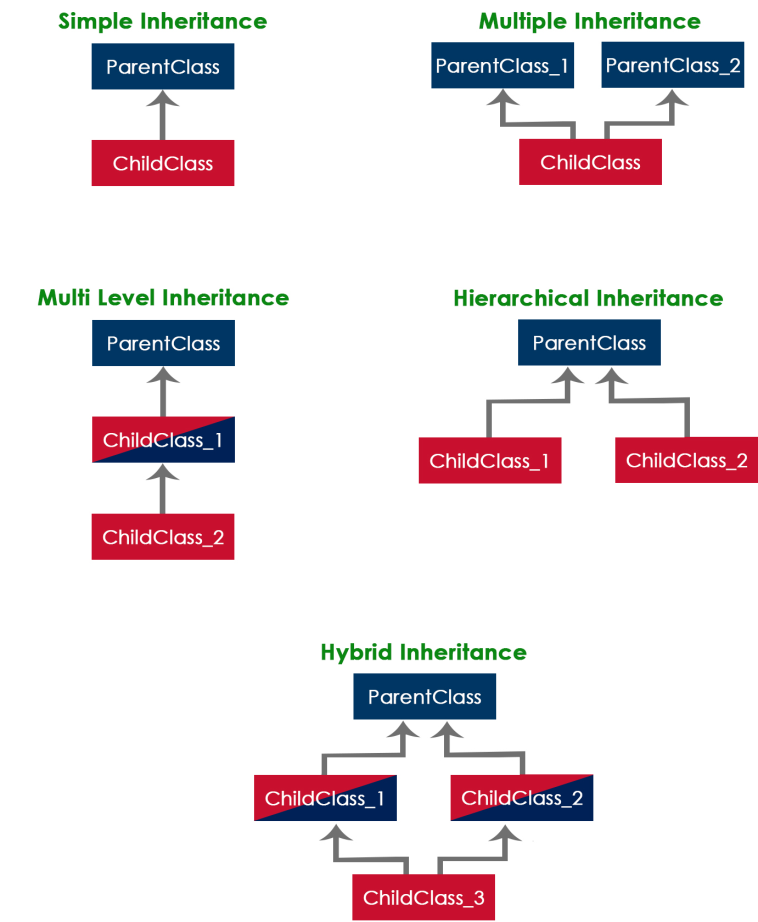
**INHERITANCE**

* 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.
3. **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.

1. **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.

# 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 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()

{ }

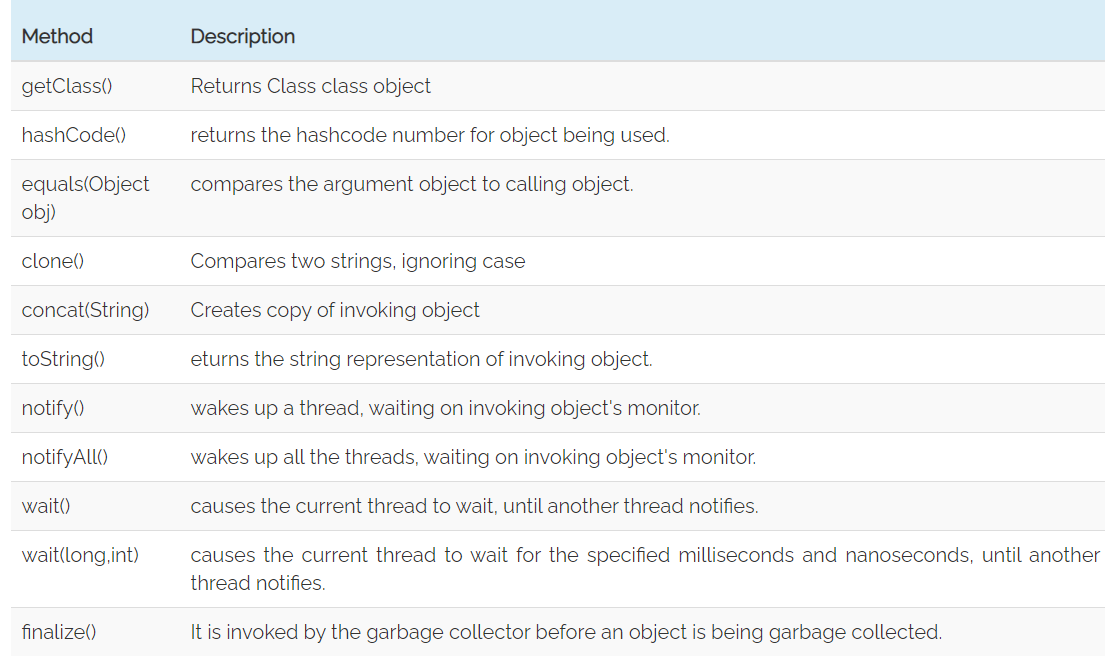
}

* 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.

## Methods of Object class



# 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.
* It holds the principle of substitutability.

## 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.
* It holds the principle of substitutability.

## Construction

* In this form of inheritance the child class may change the behaviour defined by the parent class (overriding).
* It does not hold the principle of substitutability.

## Extension

* This is another form of inheritance where the child class may add its new properties.
* It holds the principle of substitutability.

## Limitation

* In this form of inheritance the subclass restricts the inherited behaviour.
* It does not hold the principle of substitutability.

## 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, the program to jump 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.