
# Fundamentals of Object Oriented Programming using


# (CSN-103)

## Keywords in Java

**Dr. PRADUMN KUMAR PANDEY**

2022-12-20

# Keywords in Java

- Java has a set of keywords that are reserved words that cannot be used as variable name, method name, class name, object name or any other identifiers

# Keywords in Java

1. abstract
2. assert
3. boolean
4. break
5. byte
6. case
7. catch
8. char
9. class
10. continue
11. default
12. do
13. double
14. else
15. enum
16. extends
17. final
18. finally
19. float
20. for
21. if
22. implements
23. import
24. instanceof
25. int
26. interface
27. long
28. native
29. new
30. package
31. private
32. protected
33. public
34. return
35. short
36. static
37. strictfp
38. super
39. switch
40. synchronized
41. this
42. throw
43. throws
44. transient
45. try
46. void
47. volatile
48. while

# package

- 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.
- There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.
- The package keyword is used to create a package in java.
- If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

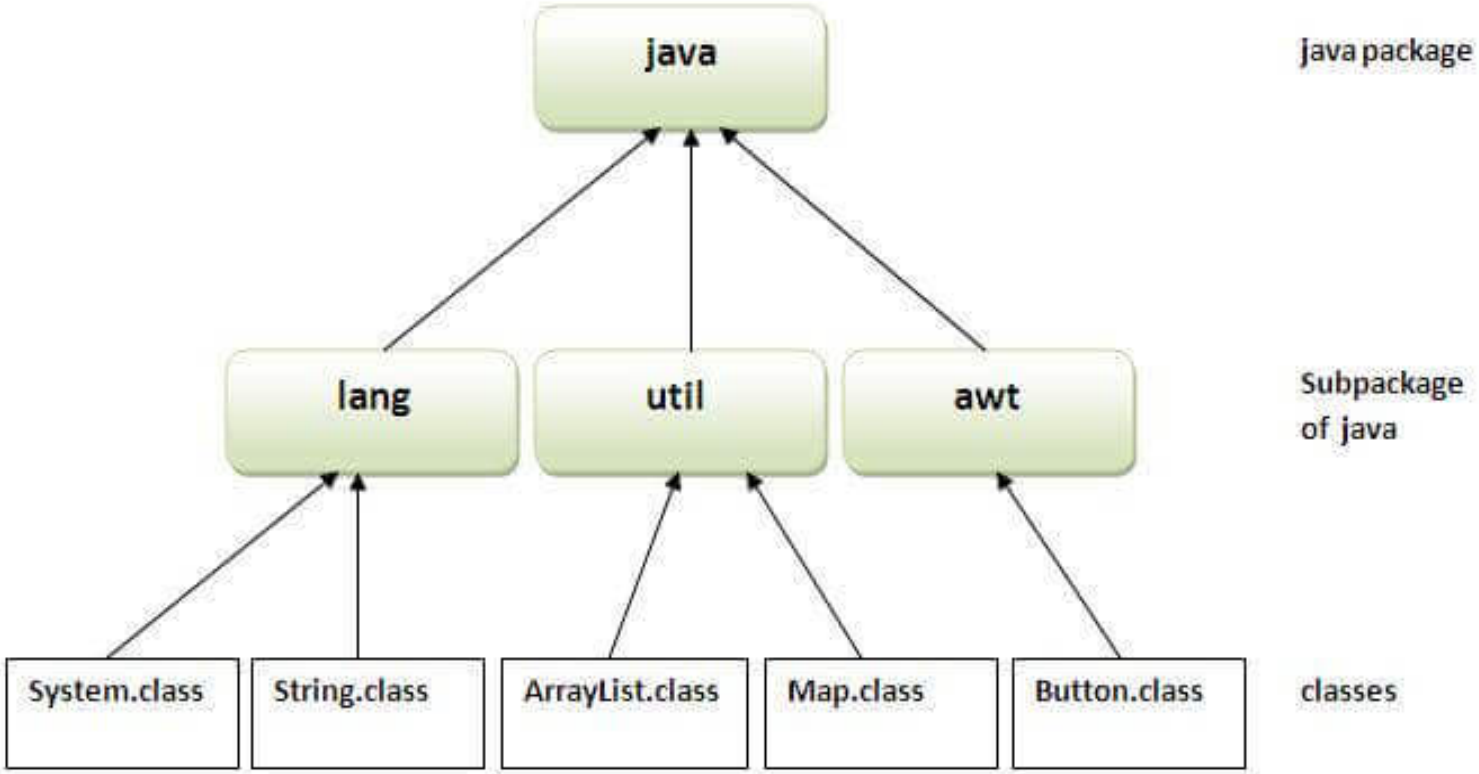
# Advantage of Java Package

- Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- Java package provides access protection.
- Java package removes naming collision.

# import

- The import keyword is used to make the classes and interface of another package accessible to the current package.
- If you import package.classname then only declared class of this package will be accessible.

# contd..


java
Java package
lang
util
awt
Subpackage
of java
System.class
String.class
ArrayList.class
Map.class
Button.class
classes

# Simple example of java package

```java
//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();
    }
}
```

Output:

Welcome to package

# this

- In Java, this is a reference variable that refers to the current object.
- this can be used to refer current class instance variable.
- this can be used to invoke current class method (implicitly)
- this() can be used to invoke current class constructor.
- this can be passed as an argument in the method call.
- this can be passed as argument in the constructor call.
- this can be used to return the current class instance from the method.

# Understanding the problem without this keyword

```
class Student
{
    int rollno;
    String name;
    float fee;
    Student(int rollno,String name,float fee)
    {
        rollno=rollno;
        name=name;
        fee=fee;
    }
    void display()
    {
        System.out.println(rollno+" "+name+" "+fee);
    }
}
class TestThis1
{
    public static void main(String args[])
    {
        Student s1=new Student(111,"ankit",5000f);
        Student s2=new Student(112,"sumit",6000f);
        s1.display();
        s2.display();
    }
}
```

Output:
0 null 0.0

0 null 0.0

# Solution of the above problem by this keyword

```
class Student
{
    int rollno;
    String name;
    float fee;
    Student(int rollno,String name,float fee)
    {
       this.rollno=rollno;
       this.name=name;
       this.fee=fee;
    }
    void display()
    {
        System.out.println(rollno+" "+name+" "+fee);
    }
}
class TestThis1
{
    public static void main(String args[])
    {
        Student s1=new Student(111,"ankit",5000f);
        Student s2=new Student(112,"sumit",6000f);
        s1.display();
        s2.display();
    }
}
```

Output:
111 ankit 5000.0

112 sumit 6000.0

# Program where this keyword is not required

```
class Student
{
    int rollno;
    String name;
    float fee;
    Student(int r,String n,float f)
    {
       rollno=r;
       name=n;
       fee=f;
    }
    void display()
    {
        System.out.println(rollno+" "+name+" "+fee);
    }
}
class TestThis1
{
    public static void main(String args[])
    {
        Student s1=new Student(111,"ankit",5000f);
        Student s2=new Student(112,"sumit",6000f);
        s1.display();
        s2.display();
    }
}
```

Output:
111 ankit 5000.0

112 sumit 6000.0

# static

- The static keyword is a non-access modifier used for methods and attributes.
- The static keyword in Java is used for memory management mainly.
- The static keyword belongs to the class than an instance of the class.
- Static methods/attributes can be accessed without creating an object of a class.

# contd..

- The static can be:
  - Variable (also known as a class variable)
  - Method (also known as a class method)
  - Block
  - Nested class

# Java 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.
- The static variable gets memory only once in the class area at the time of class loading.

# Understanding the problem without static variable

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

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

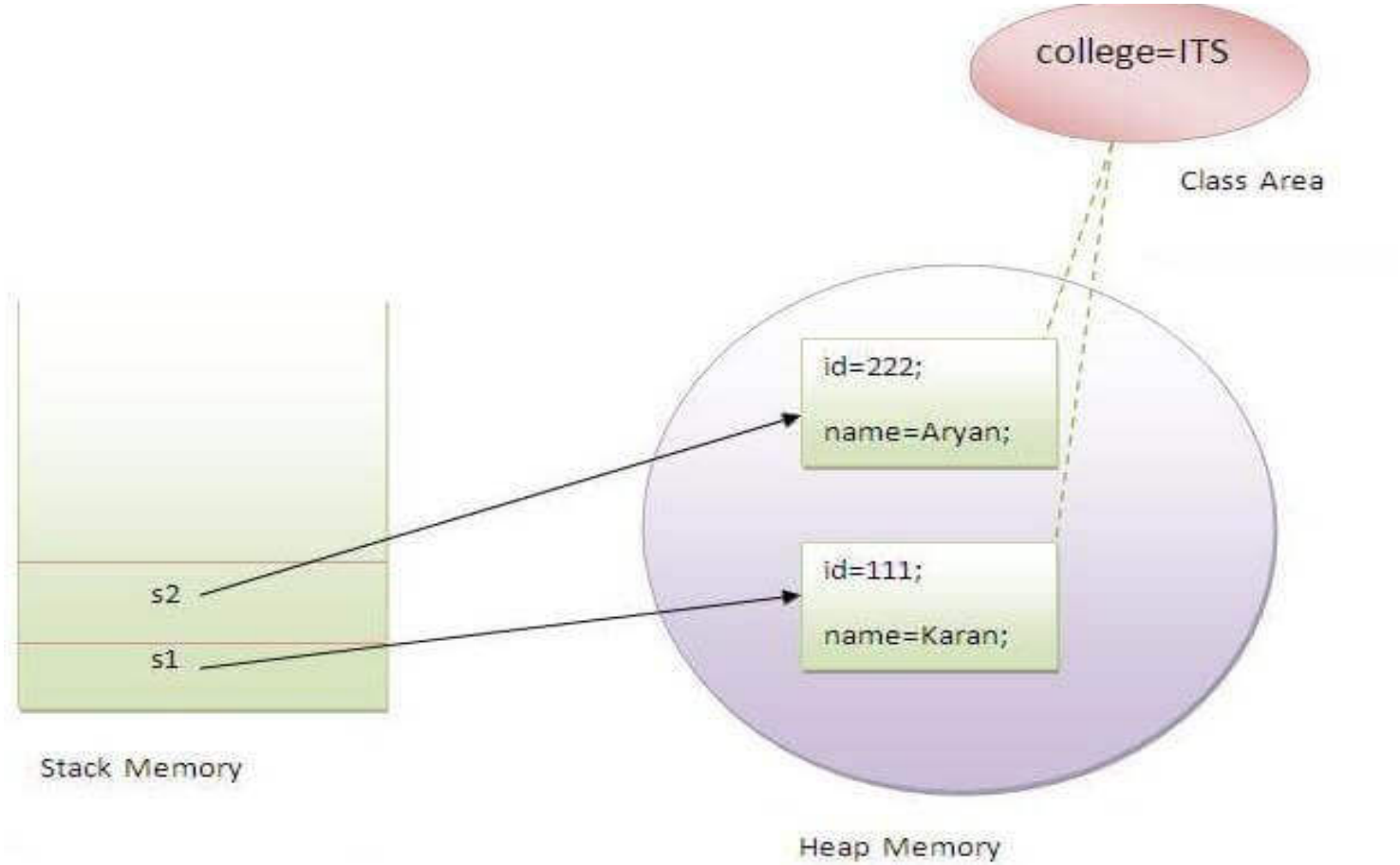
# Program to demonstrate the use of static variable

```
class Student
{
    int rollno;     //instance variable
    String name;
    static String college ="ITS";    //static variable
    Student(int r, String n)
    {
        rollno = r;
        name = n;
    }
    void display (){System.out.println(rollno+" "+name+" "+college);}
}
public class TestStaticVariable1
{
    public static void main(String args[])
    {
        Student s1 = new Student(111,"Karan");
        Student s2 = new Student(222,"Aryan");
 //we can change the college of all objects by the single line of code
 //Student.college="BBDIT";
        s1.display();
        s2.display();
    }
}
```

Output:
111 Karan ITS

222 Aryan ITS

# contd..


college=ITS
Class Area
id=222;
name=Aryan;
id=111;
name=Karan;
s2
s1
Stack Memory
Heap Memory

# Java 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.
- A static method can access static data member and can change the value of it.

# Restrictions for the static method

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

# Example of static method

```
class Student{
     int rollno;
     String name;
     static String college = "ITS";
     static void change(){
        college = "BBDIT";
     }
     Student(int r, String n){
        rollno = r;
        name = n;
     }
     void display(){System.out.println(rollno+" "+name+" "+college);}
}
public class TestStaticMethod{
    public static void main(String args[]){
        Student.change();
        Student s1 = new Student(111,"Karan");
        Student s2 = new Student(222,"Aryan");
        Student s3 = new Student(333,"Sonoo");
        s1.display();
        s2.display();
        s3.display();
    }
}
```

Output:
111 Karan BBDIT
222 Aryan BBDIT

333 Sonoo BBDIT

# NOTE

**Why is the Java main method static?**

It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.

# Java static block

- Is used to initialize the static data member.
- It is executed before the main method at the time of class loading.

# Example of static block

```
class A2
{
    static
    {
        System.out.println("static block is invoked");
    }
    public static void main(String args[])
    {
        System.out.println("Hello main");
    }
}
```

Output:
static block is invoked

Hello main

# public

- A Java public keyword is an access modifier.
- It can be assigned to variables, methods, constructors, and classes.
- It is the most non-restricted type of access modifier.
- The public access modifier is accessible everywhere. So, we can easily access the public inside and outside the class and package.

# contd..

- If you are overriding any method, overridden method (i.e., declared in the subclass) must not be more restrictive. So, if you assign public to any method or variable, that method or variable can be overridden to sub-class using public access modifier only.
- If a program contains multiple classes, at most one class can be assigned as public.
- If a class contain a public class, the name of the program must be similar to the public class name.

# whether public variable and method is accessible or not outside the class

```
class A
{
    public String msg="Try to access a public variable outside the class";
    String info;
    public void display()
    {
        System.out.println("Try to access a public method outside the class");
        System.out.println(info);
    }
    public A(String info)
    {
        this.info=info;
    }
}
public class PublicExample1
{
    public static void main(String[] args)
    {
        A a=new A("Try to create the instance of public constructor outside the class");
        System.out.println(a.msg);
        a.display();
    }
}
```

Output:

Try to access a public variable outside the class
Try to access a public method outside the class

Try to create the instance of public constructor outside the class

# whether public variable and method is accessible or not outside the package

```
//save by A.java
package com.java;
public class A {
    public String msg="Try to access a public variable outside the package";
    String info;
    public void display()  {
        System.out.println("Try to access a public method outside the package");
        System.out.println(info);
    }
    public A(String info)  {
        this.info=info;
    }
}
//save by PublicExample1.java
package com.javatpoint;
import com.java.A;
public class PublicExample2 {
    public static void main(String[] args) {
        A a=new A("Try to create the instance of public constructor outside the package");
        System.out.println(a.msg);
        a.display();
    }
}
```

Output:

Try to access a public variable outside the package
Try to access a public method outside the package

Try to create the instance of public constructor outside the package

# private

- A Java private keyword is an access modifier.
- It can be assigned to variables, methods, and inner classes.
- It is the most restricted type of access modifier.
- The private access modifier is accessible only within the same class.
- We can't assign private to outer class and interface.

# contd..

- The best use of private keyword is to create a fully encapsulated class in Java by making all the data members of that class private.
- If we make any class constructor private, we cannot create the instance of that class from outside the class.
- If we are overriding any method, overridden method (i.e., declared in the subclass) must not be more restrictive.
- According to the previous point, if we assign a private modifier to any method or variable, that method or variable can be overridden to sub-class using all type of access modifiers. However, still, we can't invoke private method outside the class.

# whether the private variable is accessible or not outside the class

```
class A
{
    private String msg="Try to access the private variable outside the class";
}
public class PrivateExample1
{
    public static void main(String[] args)
    {
        A a=new A();
        System.out.println(a.msg);
    }
}
```

Output:
Exception in thread "main" java.lang.Error: Unresolved compilation problem:

The field A.msg is not visible

# protected

- A Java protected keyword is an access modifier. It can be assigned to variables, methods, constructors and inner classes.
- The protected access modifier is accessible within the package. However, it can also accessible outside the package but through inheritance only.
- We can't assign protected to outer class and interface.

# contd..

- If you make any constructor protected, you cannot create the instance of that class from outside the package.
- If you are overriding any method, overridden method (i.e., declared in the subclass) must not be more restrictive.
- According to the previous point, if you assign protected to any method or variable, that method or variable can be overridden to sub-class using public or protected access modifier only.

# whether the protected variable is accessible or not outside the package.

```java
//save by A.java
package com.java;
public class A
{
    protected String msg="Try to access the protected variable outside the package";
}
//save by ProtectedExample1.java
package com.javatpoint;
import com.java.A;
public class ProtectedExample1
{
    public static void main(String[] args)
    {
        A a=new A();
        System.out.println(a.msg);
    }
}
```

Output:
Exception in thread "main" java.lang.Error: Unresolved compilation problem:

The field A.msg is not visible

# whether the protected variable is accessible or not outside the class and within the package.

```
class A
{
    protected String msg="Try to access the protected variable outside the class within the package";
}
public class ProtectedExample2
{
    public static void main(String[] args)
    {
        A a=new A();
        System.out.println(a.msg);
    }
}
```

Output:

Try to access the protected variable outside the class within the package

# whether the protected method is accessible or not outside the package.

```java
//save by A.java
package com.java;
public class A
{
    protected void msg()
    {
        System.out.println("Try to access the protected method outside the package ");
    }
}
//save by ProtectedExample3.java
package com.javatpoint;
import com.java.A;
public class ProtectedExample3
{
    public static void main(String[] args)
    {
        A a=new A();
        a.msg();
    }
}
```

Output:

Exception in thread "main" java.lang.Error: Unresolved compilation problem:

The method msg() from the type A is not visible

# extends

- Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).
- Inheritance represents the IS-A relationship which is also known as a parent-child relationship.
- The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

# Why use inheritance in java

- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods of the parent class. Moreover, you can add new methods in your current class also.
- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

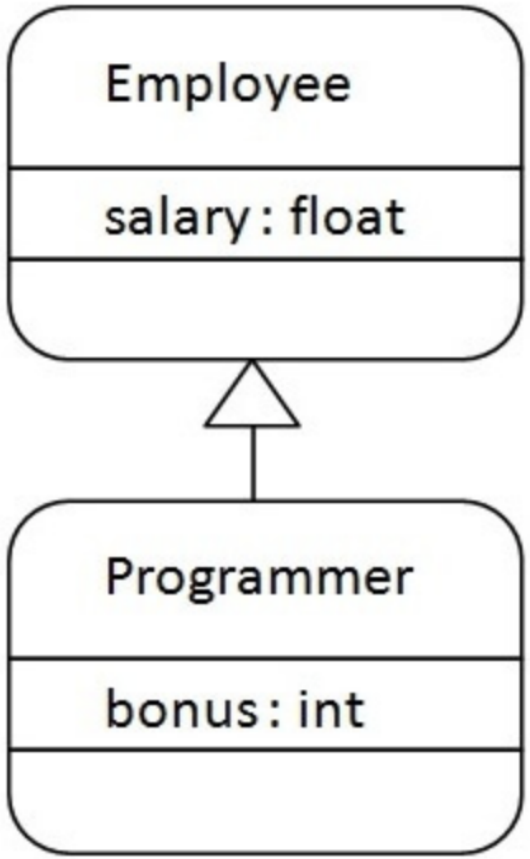
# Terms used in Inheritance

- Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

# Syntax

```
class Subclass-name extends Superclass-name
{
    //methods
}
```

# contd..



# Example

```
class Employee
{
    float salary=40000;
}
class Programmer extends Employee
{
    int bonus=10000;
    public static void main(String args[])
    {
        Programmer p=new Programmer();
        System.out.println("Programmer salary is:"+p.salary);
        System.out.println("Bonus of Programmer is:"+p.bonus);
    }
}
```

Output:
Programmer salary is:40000.0

Bonus of programmer is:10000

# abstract

- The abstract keyword is a non-access modifier, used for classes and methods.
- Class: An abstract class is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- Method: An abstract method can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).
- Note - We cannot declare abstract methods in non abstract class.

# Dont's

- An abstract keyword cannot be used with variables and constructors.
- If a class is abstract, it cannot be instantiated.
- If a method is abstract, it doesn't contain the body.
- We cannot use the abstract keyword with the final.
- We cannot declare abstract methods as private.
- We cannot declare abstract methods as static.
- An abstract method can't be synchronized.

# Do's

- An abstract keyword can only be used with class and method.
- An abstract class can contain constructors and static methods.
- If a class extends the abstract class, it must also implement at least one of the abstract method.
- An abstract class can contain the main method and the final method.
- An abstract class can contain overloaded abstract methods.
- We can declare the local inner class as abstract.
- We can declare the abstract method with a throw clause.

# Syntax

```
abstract class Employee
{
    abstract void work();
}
```

# Abstract class containing the abstract method

```
abstract class Vehicle
{
    abstract void bike();
}
class Honda extends Vehicle
{
    @Override
    void bike()
    {
        System.out.println("Bike is running");
    }
}

public class AbstractExample1
{
    public static void main(String[] args)
    {
        Honda obj=new Honda();
        obj.bike();
    }
}
```

Output:

Bike is running

# Abstract class containing the abstract and non-abstract method

```
abstract class Vehicle
{
    abstract void bike();
    void car()
    {
        System.out.println("Car is running");
    }
}
class Honda extends Vehicle
{
    @Override
    void bike()
    {
        System.out.println("Bike is running");
    }
}
public class AbstractExample2
{
    public static void main(String[] args)
    {
        Honda obj=new Honda();
        obj.bike();
        obj.car();
    }
}
```

Output:

Bike is running

Car is running

# Abstract class containing the constructor

```
abstract class Vehicle
{
    String msg;
    Vehicle(String msg)
    {
        this.msg=msg;
    }
    void display()
    {
        System.out.println(msg);
    }
}
class Honda extends Vehicle
{
    Honda(String msg)
    {
        super(msg);
    }
}
public class AbstractExample3
{
    public static void main(String[] args)
    {
        Honda obj=new Honda("Constructor is invoked");
        obj.display();
    }
}
```

Output:

Constructor is invoked

# Abstract class containing overloaded abstract methods

```
abstract class Vehicle
{
    abstract void display();
    abstract void display(String msg);
}
class Honda extends Vehicle
{
    @Override
    void display()
    {
        System.out.println("abstract method is invoked");
    }
    @Override
    void display(String msg)
    {
        System.out.println(msg);
    }
}
public class AbstractExample4
{
    public static void main(String[] args)
    {
        Honda obj=new Honda();
        obj.display();
        obj.display("overloaded abstract method is invoked");
    }
}
```

Output:

abstract method is invoked

overloaded abstract method is invoked

# Inner abstract class

```
class Vehicle
{
    abstract class Car
    {
        abstract void display();
    }
class Honda extends Car
{
    @Override
    void display()
    {
        System.out.println("inner abstract class is invoked");
    }
}
}
public class AbstractExample5
{
    public static void main(String[] args)
    {
        Vehicle obj=new Vehicle();
        Vehicle.Car c=obj.new Honda();
        c.display();
    }
}
```

Output:

inner abstract class is invoked

# Nested abstract class

```
abstract class Vehicle
{
    abstract class Car
    {
        abstract void display();
    }
}
class Honda extends Vehicle
{
    class FourWheller extends Car
    {
        @Override
        void display()
        {
            System.out.println("nested abstract class is invoked");
        }
    }
}
public class AbstractExample6
{
    public static void main(String[] args)
    {
        Vehicle obj=new Honda();
        Honda h=(Honda)obj;
        Honda.FourWheller fw=h.new FourWheller();
        fw.display();
    }
}
```

Output:

```
nested abstract class is invoked
```

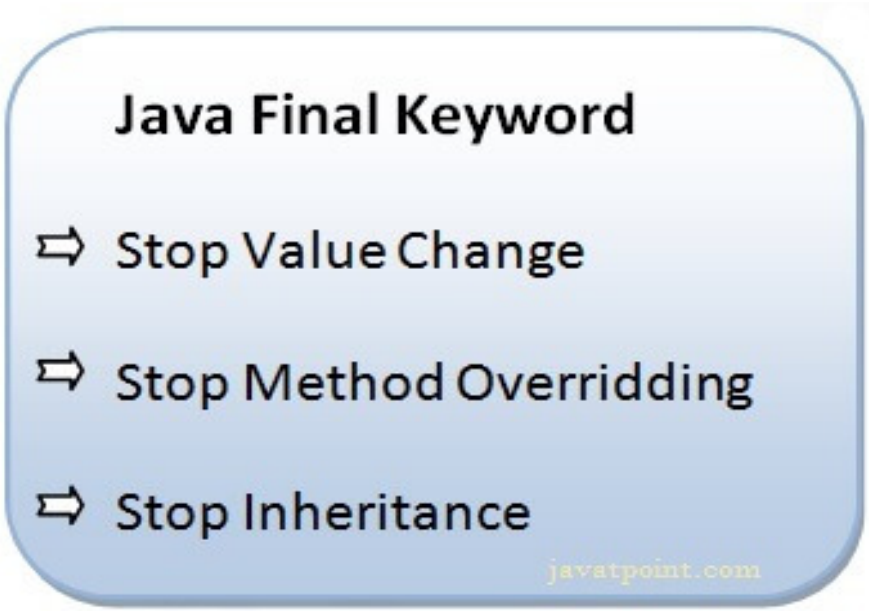
# final

- The final keyword in java is used to restrict the user.
- Final can be: variable, method, class

# Java final variable

- A final variable that have no value it is called blank final variable or uninitialized final variable.
- If you make any variable as final, you cannot change the value of final variable (It will be constant).
- It can be initialized in the constructor only.
- The blank final variable can be static also which will be initialized in the static block only.

# contd..


Java Final Keyword
Stop Value Change
Stop Method Overridding
Stop Inheritance
javatpoint.com

# Example of final variable

```
class Bike9
{
    final int speedlimit=90;
    void run()
    {
    speedlimit=400;
    }
    public static void main(String args[])
    {
    Bike9 obj=new  Bike9();
    obj.run();
    }
}
```

Output:

Compile Time Error

# Java final method

- If you make any method as final, you cannot override it.

# Example of final mehod

```
class Bike
{
    final void run()
    {
        System.out.println("running");
    }
}
class Honda extends Bike
{
    void run()
    {
        System.out.println("running safely with 100kmph");
    }
    public static void main(String args[])
    {
        Honda honda= new Honda();
        honda.run();
    }
}
```

Output:

Compile Time Error