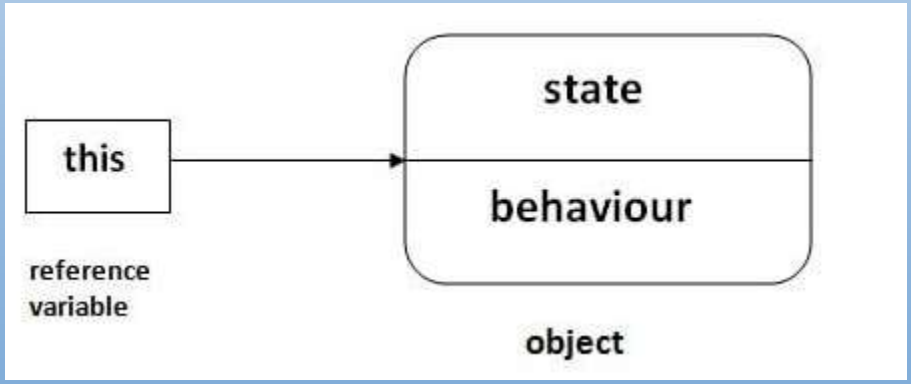
this KEYWORD AND Access Control

- THIS KEYWORD IN JAVA
- THERE CAN BE A LOT OF USAGE OF **THIS KEYWORD**. IN JAVA, THIS IS A **REFERENCE VARIABLE** THAT REFERS TO THE CURRENT OBJECT.



USAGE OF JAVA THIS KEYWORD

HERE IS GIVEN THE 6 USAGE OF JAVA THIS KEYWORD.

- 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.

- THIS KEYWORD. THIS : A REFERENCE TO THE IMPLICIT PARAMETER (THE OBJECT ON WHICH A METHOD/CONSTRUCTOR IS CALLED)

- SYNTAX: TO REFER TO A FIELD: this. field
- TO CALL A METHOD: this. method (parameters);
- TO CALL A CONSTRUCTOR this(parameters);
- FROM ANOTHER CONSTRUCTOR:. PRECONDITIONS.

1) THIS: TO REFER CURRENT CLASS INSTANCE VARIABLE

•The this Keyword Can Be Used To Refer Current Class Instance Variable. If There Is Ambiguity Between The Instance Variables And Parameters, this Keyword Resolves The Problem Of Ambiguity.

LET'S UNDERSTAND THE PROBLEM IF WE DON'T USE THIS KEYWORD BY THE EXAMPLE GIVEN BELOW:

```
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
student s2=new student(112,"sumi
;
s1.display();
s2.display();
}
}
```

Output: 0 null 0.0 0 null 0.0

Note: In the above example, parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable.

```
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 testthis2
{
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 testthis3{
public static void main(string args[]){
student s1=new student(111,"ankit",5000f);
student s2=new student(112,"sumit",6000f);
s1.display();
s2.display();
}
}
```

2) THIS: TO INVOKE CURRENT CLASS METHOD

YOU MAY INVOKE THE METHOD OF THE CURRENT CLASS BY USING THE THIS KEYWORD. IF YOU DON'T USE THE THIS KEYWORD, COMPILER AUTOMATICALLY ADDS THIS KEYWORD WHILE INVOKING THE METHOD.

```
class a{
void m()
{system.out.println("hello m");
}
void n(){
system.out.println("hello n");
this.m();
}
 }
```

```
class testthis4{
public static void main(string args[])
{
a a=new a();
a.n();
}
}
```

3) this() : to invoke current class constructor

The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

Calling default constructor from parameterized constructor:

```
class a{
a()
{
system.out.println("hello a");
}
a(int x)
{
this();
system.out.println(x);
} }
```

```
class testthis5
{
public static void main(string args[]
{
a a=new a(10);
}
}
```

calling parameterized constructor from default constructor:

```
class a{
a()
{
this(5);
system.out.println("hello a");
}
a(int x)
{
system.out.println(x);
}
}
```

```
class testthis6
{
public static void main(string args[])
{
a a=new a();
}
}
```

ACCESS CONTROL

Access control is a mechanism, an attribute of encapsulation which restricts the access of certain members of a class to specific parts of a program. Access to members of a class can be controlled using the *access modifiers.* There are four access modifiers in Java. They are:

1. public
2. protected
3. default
4. private

If the member (variable or method) is not marked as either *public* or *protected* or *private*, the access modifier for that member will be *default*.

Among the four access modifiers, *private* is the most restrictive access modifier and *public* is the least restrictive access modifier. Syntax for declaring a access modifier is shown below:

access-modifier data-type variable-name;

Access Modifiers In Java

Access Modifier	Within the Class	Other Classes [Within the Package]	In Subclasses [Within the package and other packages]	Any Class [In Other Packages]
public	Y	Y	Y	Y
protected	Y	Y	Y	N
default	Y	Y	Same Package – Y Other Packages - N	N
private	Y	N	N	N

Y – Accessible
N – Not Accessible

- **Default**: When no access modifier is specified for a class, method, or data member – It is said to be having the **default** access modifier by default.
  - The data members, classes, or methods that are not declared using any access modifiers i.e. having default access modifier are accessible **only within the same package**.

In this example, we will create two packages and the classes in the packages will be having the default access modifiers and we will try to access a class from one package from a class of the second package.

```
// Java program to illustrate default modifier
package p1;

// Class Geek is having Default access modifier
class Geek
{
        void display()
        {
                System.out.println("Hello
World!");
        }
}
```

```
// Java program to illustrate error while
// using class from different package with
// default modifier
package p2;
import p1.*;

// This class is having default access modifier
class GeekNew
{
        public static void main(String args[])
        {
                // Accessing class Geek
from package p1
                Geek obj = new Geek();

                obj.display();
        }
}
```

Output: Compile time error

- **Private**: The private access modifier is specified using the keyword **private**.
  - The methods or data members declared as private are accessible only **within the class** in which they are declared.
  - Any other **class of** the **same package will not be able to access** these members.
  - Top-level classes or interfaces can not be declared as private because
    1. private means "only visible within the enclosing class".
    2. protected means "only visible within the enclosing class and any subclasses"

```
// Java program to illustrate error while
// using class from different package with
// private modifier
package p1;

class A
{
private void display()
        {
                System.out.println("GeeksforGeeks");
        }
}

class B
{
public static void main(String args[])
        {
                A obj = new A();
                // Trying to access private method
                // of another class
                obj.display();
        } }
```

Output

```
error: display() has private
access in A
          obj.display();
```

- **protected**: The protected access modifier is specified using the keyword **protected**.
  - The methods or data members declared as protected are **accessible within the same package or subclasses in different packages.**

In this example, we will create two packages p1 and p2. Class A in p1 is made public, to access it in p2. The method display in class A is protected and class B is inherited from class A and this protected method is then accessed by creating an object of class B.

```
// Java program to illustrate
// protected modifier
package p1;


// Class A
public class A
{
protected void display()
        {
        System.out.println("GeeksforGeeks");
        }
```

```
// Java program to illustrate
// protected modifier
package p2;
import p1.*; // importing all classes in package

// Class B is subclass of A
class B extends A
{
public static void main(String args[])
{
        B obj = new B();
        obj.display();
}

}
```

```
GeeksforGeeks
```