Khulna Khan Bahadur Ahsanullah University

Object-oriented programming

CSE 1203

Lecture -9

Final Keyword in Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many contexts. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. 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. We will have detailed learning of these. Let's first learn the basics of final keyword.

1) Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Example of final variable

There is a final variable speed limit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
1. class Bike9{
2.  final int speedlimit=90;//final variable
3.  void run(){
4.   speedlimit=400;
5.  }
6.  public static void main(String args[]){
7.  Bike9 obj=new  Bike9();
8.  obj.run();
9.  }
10.}//end of class
```

Output: Compile Time Error

2) Java final method

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

Example of final method

```
1. class Bike{
2.   final void run(){System.out.println("running");}  }
3. class Honda extends Bike{
4.   void run(){System.out.println("running safely with 100kmph");}
5.   public static void main(String args[]){
6.   Honda honda= new Honda();
7.   honda.run();
8.   }  }
```

Output: Compile Time Error

3) Java final class

If you make any class as final, you cannot extend it.

Example of final class

```
1. final class Bike{}
2. class Honda1 extends Bike{
3.   void run(){System.out.println("running safely with 100kmph");}
4.   public static void main(String args[]){
5.   Honda1 honda= new Honda1();
6.   honda.run();
7.   }  }
```

Output: Compile Time Error

Q) Is final method inherited?

Yes, final method is inherited but you cannot override it. For Example:

```
1. class Bike{
2.   final void run(){System.out.println("running...");}
3. }
4. class Honda2 extends Bike{
5.    public static void main(String args[]){
6.     new Honda2().run();
7.    }
8. }
```

Output: running...

Q) What is blank or uninitialized final variable?

- A final variable that is not initialized at the time of declaration is known as blank final variable.
- If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful.
- For example, PAN CARD number of an employee.

It can be initialized only in constructor.

Example of blank final variable

```
1. class Student {
2. int id;
3. String name;
4. final String PAN_CARD_NUMBER;
5. ...
6. }
```

Que) Can we initialize blank final variable?

Yes, but only in constructor. For example:

```
1. class Bike10{
2.   final int speedlimit;//blank final variable
3.
4.   Bike10(){
5.   speedlimit=70;
6.   System.out.println(speedlimit);
7.   }
8.
9.   public static void main(String args[]){
10.    new Bike10();
11. }
12.}
```

Output: 70

4. static blank final variable

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

Example of static blank final variable

```
1. class A{
2.   static final int data;//static blank final variable
3.   static{ data=50;}
4.   public static void main(String args[]){
5.     System.out.println(A.data);
6.  } }
```

What is final parameter?

If you declare any parameter as final, you cannot change the value of it.

```
1. class Bike11{
2.   int cube(final int n){
3.    n=n+2;//can't be changed as n is final
4.    n*n*n;  }
5.   public static void main(String args[]){
6.     Bike11 b=new Bike11();
7.     b.cube (5);
8.  }  }
```

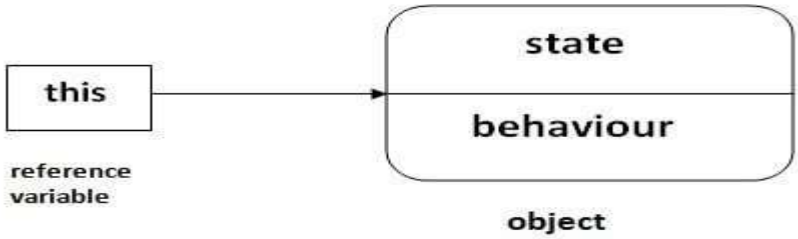
Output: Compile Time Error

Q) Can we declare a constructor final?

No, because constructor is never inherited.

This keyword in Java

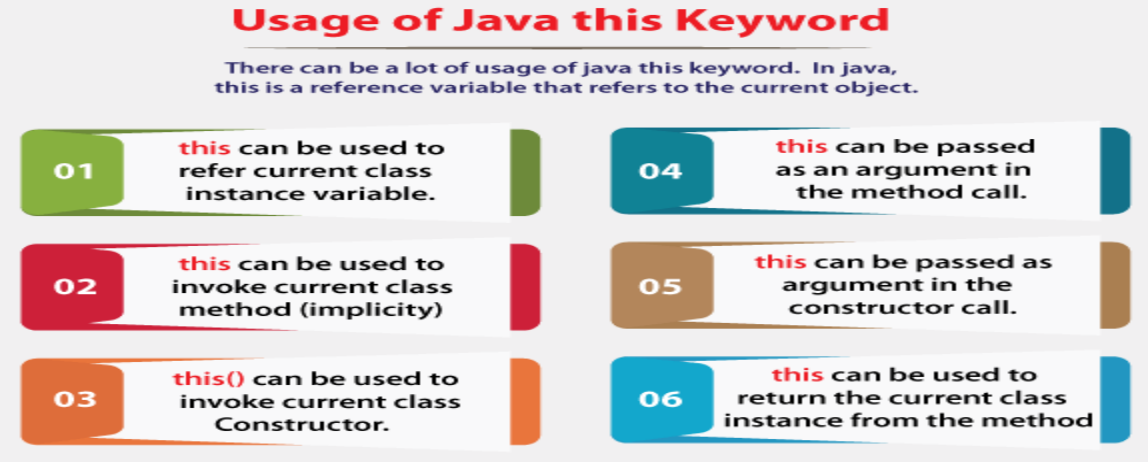
There can be a lot of usage of **Java this keyword**. In Java, this is a **reference variable** that refers to the current object.



Usage of Java this keyword

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.



1) this: to refer current class instance variable

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:

```
1. class Student{
2. int rollno;
3. String name;
4. float fee;
5. Student(int rollno,String name,float fee){
6. rollno=rollno;
7. name=name;
8. fee=fee;
9. }
10.void display(){System.out.println(rollno+" "+name+" "+fee);}
11.}
12.class TestThis1{
13.public static void main(String args[]){
14.Student s1=new Student(111,"ankit",5000f);
15.Student s2=new Student(112,"sumit",6000f);
16.s1.display();
17.s2.display();
18.}}
```

Output:

```
0 null 0.0
0 null 0.0
```

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.

Solution of the above problem by this keyword

```
1. class Student{
2. int rollno;
3. String name;
4. float fee;
5. Student(int rollno,String name,float fee){
6. this.rollno=rollno;
7. this.name=name;
8. this.fee=fee;  }
9. void display(){System.out.println(rollno+" "+name+" "+fee);}
10.}
11.class TestThis2{
12.public static void main(String args[]){
13.Student s1=new Student(111,"ankit",5000f);
14.Student s2=new Student(112,"sumit",6000f);
15.s1.display();
16.s2.display();
17.}}
```

Output:

```
111 ankit 5000.0
112 sumit 6000.0
```

If local variables (formal arguments) and instance variables are different, there is no need to use this keyword like in the following program:

Program where this keyword is not required

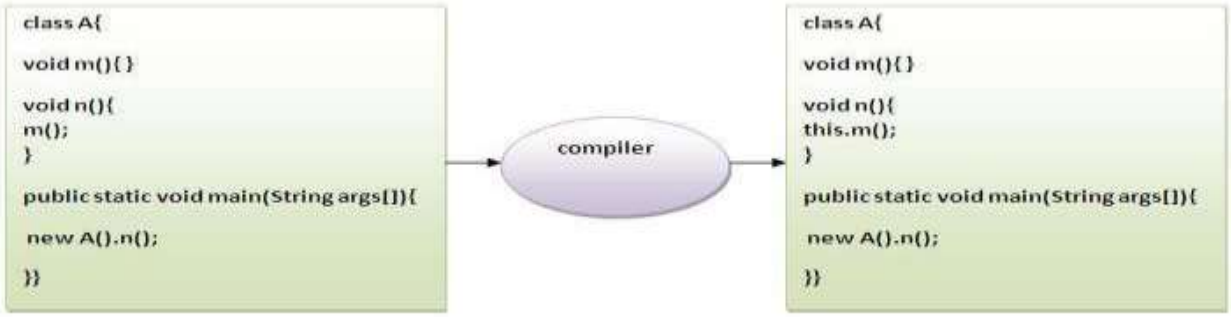
```
1. class Student{
2. int rollno;
3. String name;
4. float fee;
5. Student(int r,String n,float f){
6. rollno=r;
7. name=n;
8. fee=f;
9. }
10.void display(){System.out.println(rollno+" "+name+" "+fee);}
11.}
12.class TestThis3{
13.public static void main(String args[]){
14.Student s1=new Student(111,"ankit",5000f);
15.Student s2=new Student(112,"sumit",6000f);
16.s1.display();
17.s2.display();
18.}}
```

Output:

```
111 ankit 5000.0
112 sumit 6000.0
```

2) this: to invoke current class method

You may invoke the method of the current class by using this keyword. If you don't use this keyword, compiler automatically adds this keyword while invoking the method. Let's see the example




```
1. class A{
2. void m(){System.out.println("hello m");}
3. void n(){
4. System.out.println("hello n");
5. //m();//same as this.m()
6. this.m();
7. }  }
8. class TestThis4{
9. public static void main(String args[]){
10.A a=new A();
11.a.n();
12.}}
```

Output:

```
hello n
hello m
```

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:

```
1. class A{
2. A(){System.out.println("hello a");}
3. A(int x){
4. this();
5. System.out.println(x);
6. }  }
7. class TestThis5{
8. public static void main(String args[]){
9. A a=new A(10);
10.}}
```

Output:

```
hello a
10
```

Calling parameterized constructor from default constructor:

```
1. class A{
2. A(){
3. this(5);
4. System.out.println("hello a");
5. }
6. A(int x){
```

```
7. System.out.println(x);
8. }
9. }
10.class TestThis6{
11.public static void main(String args[]){
12.A a=new A();
13.}}
```

Output:

```
5
hello a
```

Real usage of this () constructor call

The this () constructor call should be used to reuse the constructor from the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining. Let's see the example given below that displays the actual use of this keyword.

```
1. class Student{
2. int rollno;
3. String name,course;
4. float fee;
5. Student(int rollno,String name,String course){
6. this.rollno=rollno;
7. this.name=name;
8. this.course=course;
```

```
9.  }
10.Student(int rollno,String name,String course,float fee){
11.this(rollno,name,course);//reusing constructor
12.this.fee=fee;
13.}
14.void display(){System.out.println(rollno+" "+name+" "+course+" "+fee);}
15.}
16.class TestThis7{
17.public static void main(String args[]){
18.Student s1=new Student(111,"ankit","java");
19.Student s2=new Student(112,"sumit","java",6000f);
20.s1.display();
21.s2.display();
22.}}
```

Output:

```
111 ankit java 0.0
112 sumit java 6000.0
```

Rule: Call to this() must be the first statement in constructor.

```
1.  class Student{
2.  int rollno;
3.  String name,course;
4.  float fee;
5.  Student(int rollno,String name,String course){
6.  this.rollno=rollno;
```

```
7.  this.name=name;
8.  this.course=course;
9.  }
10.Student(int rollno,String name,String course,float fee){
11.this.fee=fee;
12.this(rollno,name,course);//C.T.Error
13.}
14.void display(){System.out.println(rollno+" "+name+" "+course+" "+fee);}
15.}
16.class TestThis8{
17.public static void main(String args[]){
18.Student s1=new Student(111,"ankit","java");
19.Student s2=new Student(112,"sumit","java",6000f);
20.s1.display();
21.s2.display();
22.}}
```

Output:

```
Compile Time Error: Call to this must be first statement in constructor
```

4) this: to pass as an argument in the method

This keyword can also be passed as an argument in the method. It is mainly used in the event handling. Let's see the example:

```
1. class S2{
2.   void m(S2 obj){
3.   System.out.println("method is invoked");
4.   }
5.   void p(){
6.   m(this);
7.   }
8.   public static void main(String args[]){
9.   S2 s1 = new S2();
10.  s1.p();
11.  }
12.}
```

Output:

```
method is invoked
```

Application of this that can be passed as an argument:

In event handling (or) in a situation where we have to provide reference of a class to another one. It is used to reuse one object in many methods.

5) this: to pass as argument in the constructor call

We can pass this keyword in the constructor also. It is useful if we have to use one object in multiple classes. Let's see the example:

```
1.  class B{
2.    A4 obj;
3.    B(A4 obj){
4.      this.obj=obj;
5.    }
6.    void display(){
7.      System.out.println(obj.data);//using data member of A4 class
8.    }
9.  }
10.
11.class A4{
12.  int data=10;
13.  A4(){
14.   B b=new B(this);
15.   b.display();
16.  }
17.  public static void main(String args[]){
18.   A4 a=new A4();
19.  }
20.}
```

Output:10

6) this keyword can be used to return current class instance

We can return this keyword as an statement from the method. In such case, return type of the method must be the class type (non-primitive). Let's see the example:

Syntax of this that can be returned as a statement

```
1. return_type method_name(){
2. return this;
3. }
```

Example of this keyword that you return as a statement from the method

```
1. class A{
2. A getA(){
3. return this;
4. }
5. void msg(){System.out.println("Hello java");}
6. }
7. class Test1{
8. public static void main(String args[]){
9. new A().getA().msg();
10.}
11.}
```

Output:

```
Hello java
```