FAST

National University of Computer and Emerging Sciences Peshawar

Lecture # 13

Software Construction and Development (Java Programming)

Instructor: Muhammad Abdullah Orakzai

DEPARTMENT OF COMPUTER SCIENCE

الذی علم بالقلم۔ علم الانسان ما لم یعلم۔

(Java super and final keyword)

Contents

1) super keyword

2) final keyword

super keyword in java

- super keyword is used to call overrided method of super class.

- super keyword in java is reference variable that is used to refer immediate super class object.

- Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by **super** reference variable.

Usage of super keyword in java

1. super keyword in java is reference variable that is used to refer immediate super class object (instance) variable.
2. super() is used to invoke immediate parent class constructor.
3. super is used to invoke immediate super class method.

1) super is used to refer immediate parent class instance variable

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

```
class Animal
{
String color="white";
}
```

1) super is used to refer immediate parent class instance variable…

```
class Dog extends Animal{
 String color="black";
 void printColor(){
 System.out.println(color);       //prints color of Dog class
 System.out.println(super.color);    //prints color of Animal class
 }
```

1) super is used to refer immediate parent class instance variable…

```
class TestSuper1{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
}}
```

Output:

```
black
white
```

In the above example, Animal and Dog both classes have a common property color. If we print color property, it will print the color of current class by default. To access the parent property, we need to use super keyword.

2) super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

```
class Animal{
void eat(){
System.out.println("eating...");
}
}
```

2) super can be used to invoke parent class method…

```
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
void bark(){System.out.println("barking...");}
void work(){
super.eat();
bark();
}
}
```

2) super can be used to invoke parent class method…

```
class TestSuper2{
public static void main(String args[]){
Dog d=new Dog();
d.work();
}}
```

Output:

```
eating...
barking…
```

In the above example Animal and Dog both classes have eat() method if we call eat() method from Dog class, it will call the eat() method of Dog class by default because priority is given to local.

To call the parent class method, we need to use super keyword.

3) super is used to invoke parent class constructor

The super keyword can also be used to invoke the parent class constructor. Let's see a simple example:

```
class Animal{
Animal()
{
System.out.println("animal is created");
}
}
```

3) super is used to invoke parent class constructor…

```
class Dog extends Animal{
Dog(){
super();
System.out.println("dog is created");
}
}
```

3) super is used to invoke parent class constructor…

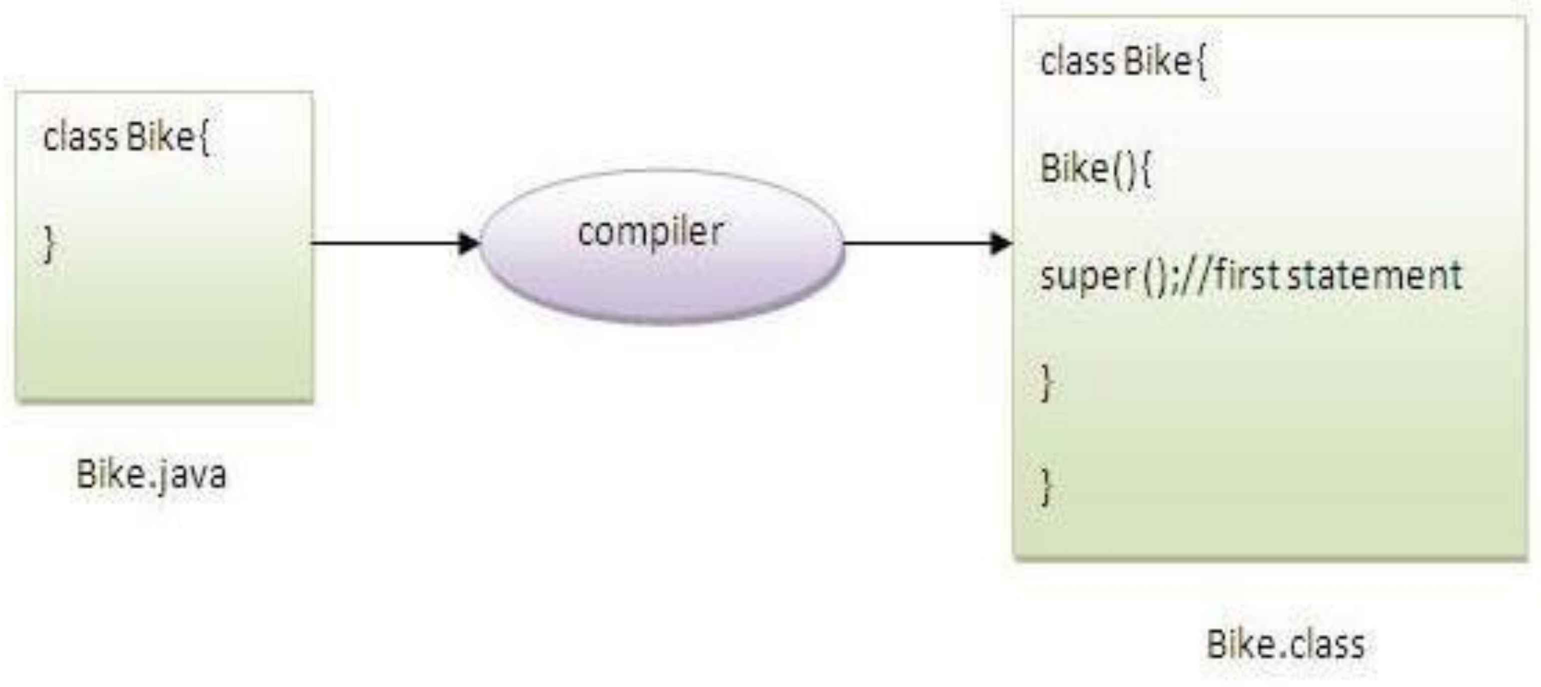
```
class TestSuper3{
public static void main(String args[]){
Dog d=new Dog();
}}
```

Output:

```
animal is created
dog is created
```

3) super is used to invoke parent class constructor...

Note: super() is added in each class constructor automatically by compiler if there is no super() or this().



```
class Bike{

}
```

```
class Bike{

Bike(){

super();//first statement

}

}
```

3) super is used to invoke parent class constructor…

As we know well that default constructor is provided by compiler automatically if there is no constructor. But, it also adds super() as the first statement.

Another example of super keyword where super() is provided by the compiler implicitly.

```
class Animal{
Animal()
{
System.out.println("animal is created");
}
}
```

3) super is used to invoke parent class constructor…

```
class Dog extends Animal{
Dog(){
System.out.println("dog is created");
}
}
```

3) super is used to invoke parent class constructor…

```
class TestSuper4{
public static void main(String args[]){
Dog d=new Dog();
}}
```

Output:

```
animal is created
dog is created
```

super example: real use

- Let's see the real use of super keyword. Here, Emp class inherits Person class so all the properties of Person will be inherited to Emp by default.

- To initialize all the property, we are using parent class constructor from child class.

- In such way, we are reusing the parent class constructor.

super example: real use…

```
class Person{
int id;
String name;
Person(int id,String name){          //constructor
this.id=id;
this.name=name;
}
}
```

super example: real use...

```
class Emp extends Person{
float salary;
Emp(int id,String name,float salary){
super(id,name);        //reusing parent constructor
this.salary=salary;
}
void display(){System.out.println(id+" "+name+" "+salary);}
}
```

super example: real use…

```
class TestSuper5{
public static void main(String[] args){
Emp e1=new Emp(1,"ankit",45000f);
e1.display();
}
}
```

Output:

```
1 ankit 45000
```

final keyword

The final keyword in java is used to restrict the user.

The java final keyword can be used in many context.

Final can be;

1. Variable
2. Method
3. Class

final keyword…

- The final keyword can be applied with the variables, a final variable that have no value is called blank final variable or uninitialized final variable.
- It can be initialized in constructor only.
- The blank final variable can be static also which will be initialized in static block only.

final keyword…

1) Stop value change

(if you have final variable then you cannot change its value) .

2) Stop method overriding

(if you have final method then you cannot override this method).

3) Stop inheritance

(if you have final class then you cannot inherit this class).

Java final variables

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

Note

Final variable is normally written in upper case i.e. COUNT , PI etc.

The following variable can be made final:

1) Local Variable
2) Instance Variable
3) Static Variable

Example of final variable

There is a final variable speedlimit, 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.

```
class Bike9{
 final int speedlimit=90;        //final variable
 void run(){
  speedlimit=400;
 }
```

Example of final variable…

```
public static void main(String args[]){
 Bike9 obj=new  Bike9();
 obj.run();
 }
}//end of class
```

Output: Compile Time Error

Class variables or static variables

Declaration

```
static final int COUNT ;
```

Initialization

```
static final int COUNT = 300;
```

Class variables or static variables…

OR

```
static final int COUNT;       // static blank final variable
static
{
COUNT =  300;
}
```

static blank final variable

- A static final variable that is not initialized at the time of declaration is known as static blank final variable.

- **Note:** static blank final variable can be initialized in static block only.

static blank final variable

Example

```
class A
{
static final int data;   // static blank final variable
static                // static block
{
  data = 40 ;
}
```

static blank final variable…

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

{

System.out.println(A.data);

}

}
```

Instance variables

Declaration

```
final int COUNT ;
```

Initialization

```
class X

{

static final int COUNT = 300;

}
```

Instance variables…

OR By Using constructor

```
class X
{
 final int COUNT;       // blank final variable
   X()                          //constructor
       {
            COUNT = 300;
       }
}
```

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

```
class Student{
int id;
String name;
final String PAN_CARD_NUMBER;
...
}
```

Blank or uninitialized final variable

Example

```
class  Bike

{

final int speedLlimit;   //   blank final variable

}
```

Blank or uninitialized final variable

```
Bike()
{
final int speedLimit = 70;
System.out.println(speedLimit);
}
public  static void main(String args[])
{
new Bike();                    // nameless object created
}
}
```

Output: 70

Local variables

Variables within the method are called local variables.

```
class X{
void disp()
{
 final int COUNT =  300;
//Or
final int COUNT;
COUNT = 300
}
}
```

final parameter

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

```
void cube(final int n)
{
 n=n+2;  // cannot be change
 n*n*n;
}
```

final parameter…

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

Output: Compile Time Error

Java final method

If you make any method as final , it cannot be overridden.

Example

```
final disp()

{

Statement(s);

}
```

Example of final method

```
class Bike
{
  final void run(){System.out.println("running");}
}
```

Example of final method…

```
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

Is final method inherited?

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

```
class Bike{
  final void run(){System.out.println("running...");}
 }
 class Honda2 extends Bike{
   public static void main(String args[]){
    new Honda2().run();
   }
 }
```

Output: running...

Java final class

If you make any class as final , it cannot be inherited or extended.

Example

```
final class  X{
-------
}
class Y extends X   // compile time error
```

Note: final method can be inherited but you cannot override it.

Example of final class

```
final class Bike{}

class Honda1 extends Bike{
  void run(){System.out.println("running safely with 100kmph");}

  public static void main(String args[]){
  Honda1 honda= new Honda1();
  honda.run();
  }
}
```

Output: Compile Time Error

THANK YOU