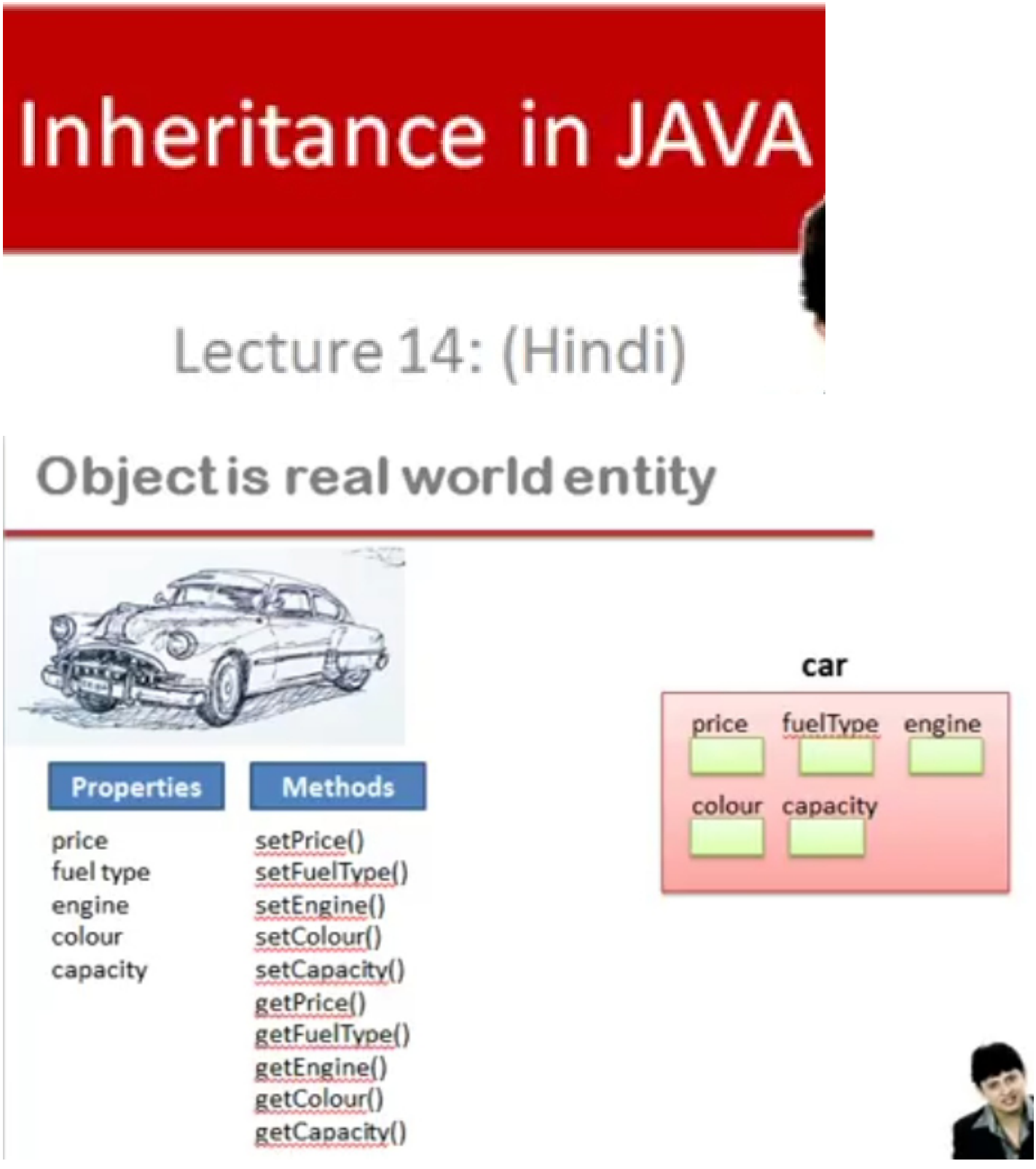
Java inheritance

Lecture 14



- Here five variable and ten function

Object is real world entity

Properties

price
fuel type
engine
Colour
capacity
alarm
navigator
safeGuard

Methods

setAlarm()
setNavigator()
setSafeGuard()
getAlarm()
getNavigator()
getSafeGuard()
setPrice()
setFuelType()
setEngine()
setColour()
setCapacity()
getPrice()
getFuelType()
getEngine()
getColour()
getCapacity()

- Oops says we which work you done don't need to do again.
- So two different class will not be create and create one class is also wrong for both car and sports car.
- **Incapsulation says in one type related all variable and function must be in one group.**
- **So we need inheritance.**
- **So we have to create a extra variable and function only and crate by use of special Syntex relationship between them.**

Syntax

```
class SubClass extends SuperClass
{

}
```

- extends is a keyword
- Base class means Super Class
- Derived Class means Sub Class

- **Here SubClass is a creating a new class(ex:-sports car)**
- SuperClass is class already created (ex:- Normal Car)
- In C++ SuperClass is a Base Class or Parent and Derived Class or Child Class.
- **Here extands keyword is used.**

Example of inheritance

Person.java - Notepad

File Edit Format View Help

```
public class Person
{
 private int age;
 private String name;
 public void setAge(int a)
 {age=a;}
 public void setName(String n)
 {name=n;}
 public int getAge()
 { return(age); }
 public String getName()
 { return(name); }
}
```

Student.java - Notepad

File Edit Format View Help

```
class Student extends Person
{
 private int rollno;
 public void setRollno(int r)
 { rollno=r; }
 public int getRollno()
 { return(rollno); }
}
```

Example.java - Notepad

File Edit Format View Help

```
public class Example
{
 public static void main(String []args)
 {
  Student s1=new Student();
  s1.setRollno(100);
  s1.setName("Rahul");
  s1.setAge(18);
  System.out.println("Rollno: "+s1.getRollno());
  System.out.println("Name: "+s1.getName());
  System.out.println("Age: "+s1.getAge());
 }
}
```

Command Prompt

```
G:\Java Programs>javac *.java

G:\Java Programs>java Example
Rollno: 100
Name: Rahul
Age: 18

G:\Java Programs>_
```

Remember

- In the Java programming language, each class is allowed to have one direct superclass, and each superclass has the potential for an unlimited number of *subclasses*
- Private members of the superclass are not accessible by the subclass and can only be indirectly accessed.
- Members that have default accessibility in the superclass are also not accessible by subclasses in other packages

- **Multiply inheritance is not possible in java.**
- **I.e no more then one parent class possible but in C++ it is possible.**

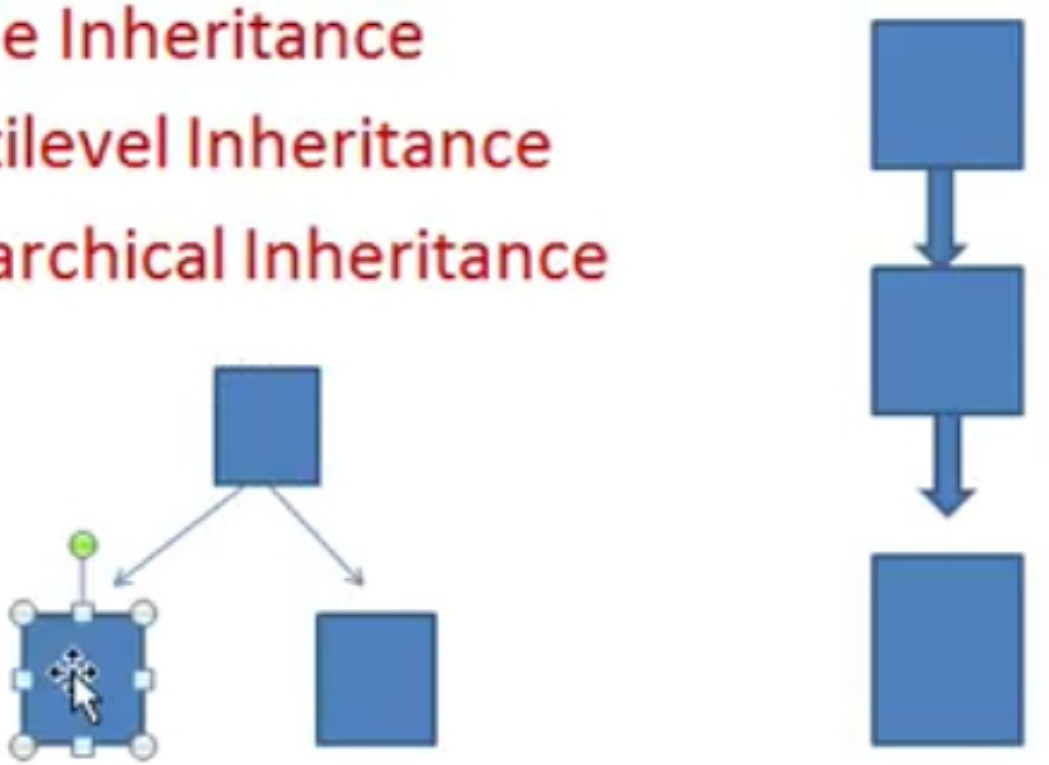
Private class is inheritable but not accessible.

Java Supports

- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance

Java Supports

- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance



- **Example of multilevel inheritance** in parent class there is SubClass and in SubClass there also have sub class.
- **Hierarchical inheritance** more then one SubClass possible.

Lecture 15

Initialization block in JAVA

Lecture 15: (Hindi)

By: SAURABH SHUKLA

Initialization Block

- There are two types of initialization blocks
  - Instance Initialization Block
  - Static Initialization Block

- Member variable and member function are two type one is instant member variable and instant member function or static member variable or static member function.
- Instant member variable and instant member function belonging to object.
- static member variable or static member function belonging to whole class.

Instance Initialization Block

```
public class Test {
 private int x;
 {
  System.out.println("Initialization Block: x="+x);
  x=5;
 }
public Test() {
  System.out.println("Constructor: x="+x);
 }
public static void main(String []args) {
   Test t1=new Test();
   Test t2=new Test();
 }
}
```

- **Instant initialization block run when we create object block.**
- **It is possible to create more then one initialization block.**
- **When we create more then one initialization block then compiler create all block in single block.**
- And sequence of code is also create in the sequence of initialization and starting of constructor line will write the initialization line code and then constructor line code.

- For every object x will be different.

Instance Initialization Block

- An *instance initializer or Initialization block* declared in a class is executed when an instance of the class is created
- return keyword cannot be used in Initialization block
- Instance initializers are permitted to refer to the current object via the keyword this and to use the keyword super

Static initialization block

```
public class Test
{
 private static int k;
 static
 {
   System.out.println("Static Initialization Block: k="+k);
   k=10;
 }
 public static void main(String []args)
 {
   new Test();
 }
}
```

- Static initialization block only accessible by class static member.
- This is not accessible by instant member.
- Static block also create more then one in class.
- Compiler can create all static block into single static block.
- Static initialization block run one time for whole class.

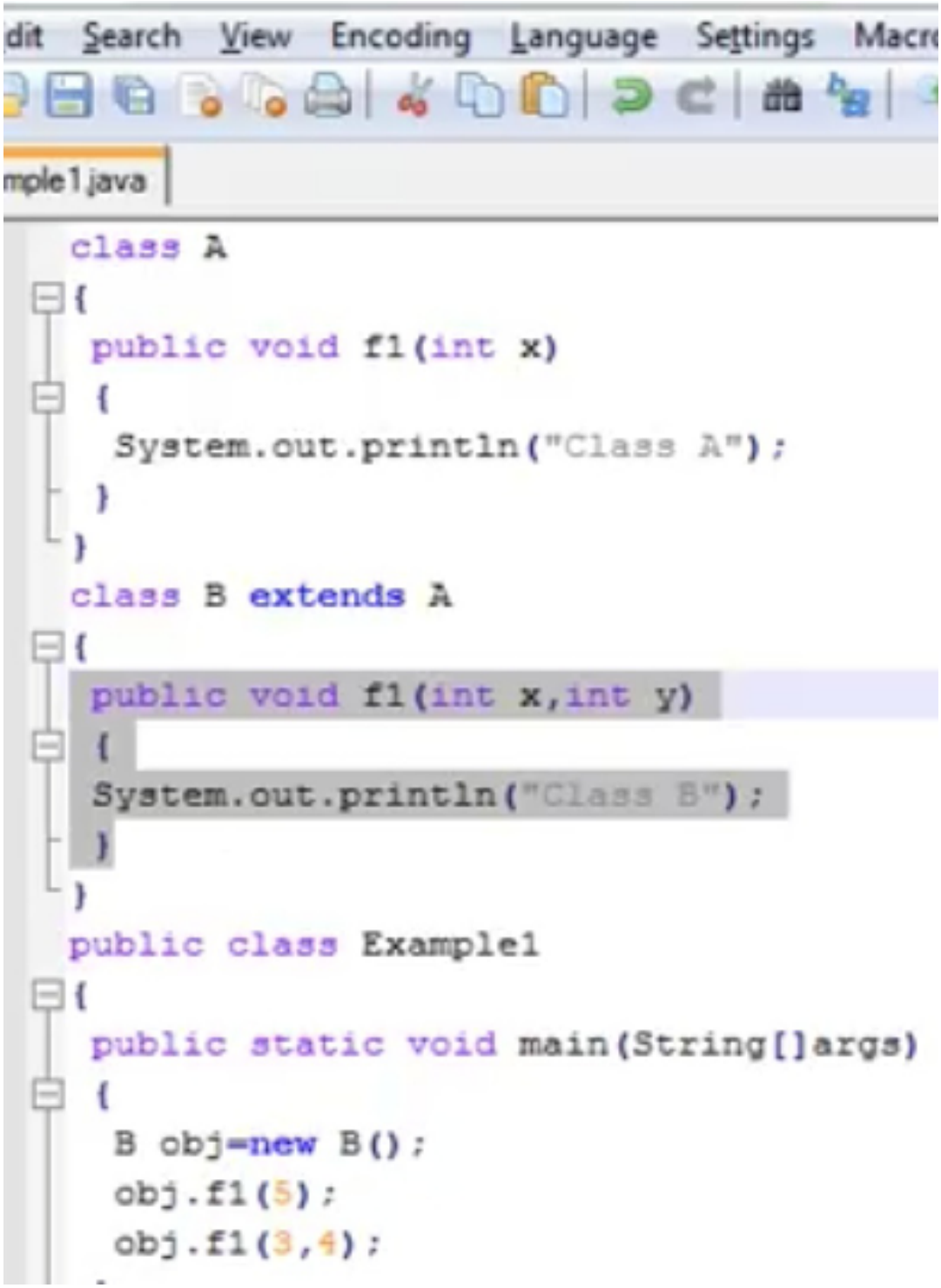
Static initialization block

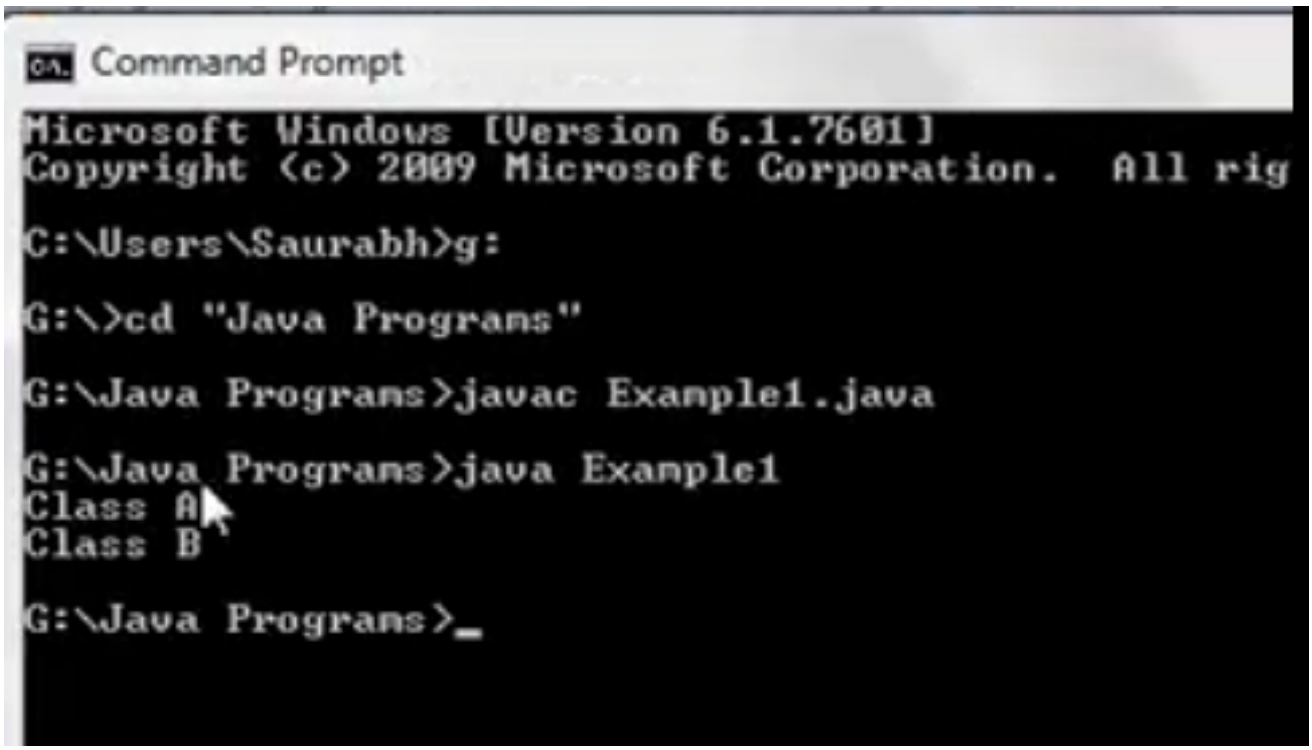
- A *static initializer* declared in a class is executed when the class is initialized
- Static initializers may be used to initialize the class variables of the class
- return keyword cannot be used in static Initialization block
- this or super can not be used in static block

Lecture 16

Function Overloading in Java

- If two methods of a class (whether both declared in the same class, or both inherited by a class, or one declared and one inherited) have the same name but signatures that are not same, then the method name is said to be *overloaded*.
- Method overloading is a way to implement polymorphism


dit Search View Encoding Language Settings Macr
mple1.java
class A
{
public void f1(int x)
{
System.out.println("Class A");
}
}
class B extends A
{
public void f1(int x,int y)
{
System.out.println("Class B");
}
}
public class Example1
{
public static void main(String[]args)
{
B obj=new B();
obj.f1(5);
obj.f1(3,4);



- In function Overloading name will same of function and argument will different.

Overriding in Java

- Method overriding is defining a method in subclass with the same signature with specific implementation in respect to the subclass.

- Why Overriding?

Example1.java

```
1  class A
2  {
3   public void f1(int x)
4   {
5    System.out.println("Class A");
6   }
7  }
8  class B extends A
9  {
10
11  public void f1(int x)
12  {
13  System.out.println("Class B");
14  }
15 }
16 public class Example1
17 {
18  public static void main(String[]args)
19  {
20   B obj=new B();
21   obj.f1(5);
22   obj.f1(3,4);
23  }
24 }
```

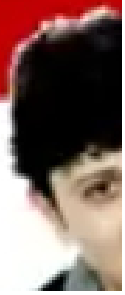
- Signature of function in parent are same in signature of function in child.

```
G:\Java Programs>javac Example1.java

G:\Java Programs>java Example1
Class B

G:\Java Programs>
```

Only second part will execute.

Lecture 17

final keyword in JAVA



Lecture 17: (Hindi)

The final keyword

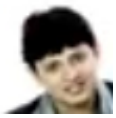
- **final** instance variable
- **final** static variable
- **final** local variable
- **final** class
- **final** methods

- In Function we create final local variable
- We also create instant member function create as final method.

final instance variable

- A java variable can be declared using the keyword final. Then the final variable can be assigned only once.

- A variable that is declared as final and not initialized is called a blank final variable. A blank final variable forces either the constructors to initialize it or initialization block to do this job.



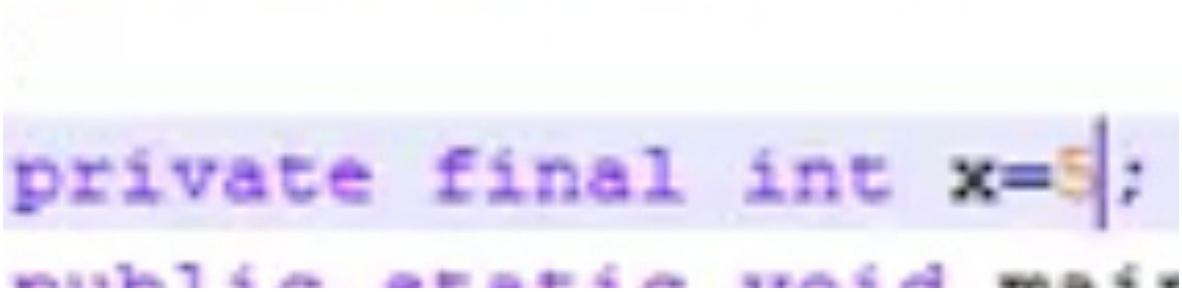
```
public class Example
{
 private  int x; //final instance member variable
 public static void main(String[] args)
 {
  Example e1=new Example();

 }
}
```

- **Here** e1 is reference variable
- **Object will be created with reference variable** e1=new Example. **Here new Example is object.**
- **By default x value is 0.**
- In java no garbage value there is blank of by default value is present.

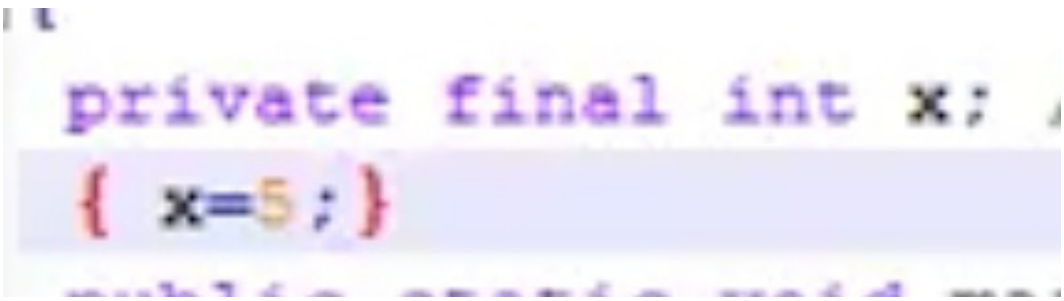
```
public class Example
{
 private final int x; //final instance member variable
 public static void main(String[] args)
 {
  Example e1=new Example();

 }
}
```

- **When we write final before variable then this int x value will be blank.**
- We can initialize the value bof x in three ways

1. **By direct**



2. **By the help of initialization block** .it run immediately run when object will create.



3. **With the help of constructor.**

3.

```
private final int x;
Example()
{ x=5;}
```

final static variable

- Static member variable when qualified with final keyword, it becomes blank until initialized.
- Final static variable can be initialized during declaration or within the static block

```
public class Example
{
 private final int x; //final instance member variable
 private final static int y; // final static member variable
 Example()
 { x=5;}
 public static void main(String[] args)
 {
  Example e1=new Example();

 }
}
```

- We can initialize two way of static variable . direct and with the help of static keywords.

```
1  public class Example
2  {
3   private final int x; //final instance member variable
4   private final static int y; // final static member variable
5   static
6   { y=4;}
7   Example()
8   { x=5;}
9   public static void main(String[] args)
10  {
11   Example e1=new Example();
12
13  }
14 }
```

final local variable

- Local variables that are final must be initialized before it's use, but you should remember this rule is applicable to non final local variables too.

- Once they are initialized, can not be altered.

```java
public class Example
{
 private final int x; //final instance member variable
 private final static int y; // final static member variable
 static
 { y=4;}
 Example()
 { x=5;}
 public void fun()
 {
   final int k; //final local variable

 }
 public static void main(String[] args)
 {
  Example e1=new Example();

 }
}
```

Final local variable can't be change.

final class

- Java classes declared as final cannot be extended. Restricting inheritance!

```java
final class Dummy
{
}
public class Example
{
 private final int x; //final instance member variable
 private final static int y; // final static member variable
 static
 { y=4;}
 Example()
 { x=5;}
 public void fun()
 {
   final int k; //final local variable

 }
 public static void main(String[] args)
 {
  Example e1=new Example();

 }
}
```

- When we write final class it cannot be extends.

final methods

- Methods declared as final cannot be overridden

```
class Dummy
{
  public final void someFunction()
  { }
}
class MoreDuumy extends Dummy
{
  public void someFunction() //error
  { }
}
```

- It show error.

Lecture 18

this keyword

- The this object reference is a local variable in instance member methods referring the caller object
- this keyword is used as a reference to the current object which is an instance of the current class
- The this reference to the current object is useful in situations where a local variable hides, or shadows, a field with the same name.

- This keyword use in c++ as pointer
- In java this is use as reference variable.

```
1  class Box
2  {
3   private int l,b,h;
4   public void setDimension(int x,int y,int z) //instance member function
5   { l=x; b=y; h=z; }
6
7  }
8  public class Example
9  {
10  public static void main(String[]args) //static member function
11  {
12   Box b1=new Box();
13   b1.setDimension(12,10,5);
14  }
15 }
```

- "This" is a object reference.

- **Here "b1" is a object reference which contains object of box.**
- Here setDimention is a caller object.
- **Here l,b,h is a instance variables but x,y,z is a local variable.**

```
class Box
{
 private int l,b,h;
 public void setDimension(int l,int b,int h) //instance member function
 { l=l; b=b; h=h; }

}
public class Example
{
 public static void main(String[]args) //static member function
 {
  Box b1=new Box();
  b1.setDimension(12,10,5);
 }
}
```

```
class Box
{
 private int l,b,h;
 public void setDimension(int l,int b,int h) //instance member function
 { this.l=l; this.b=b; this.h=h; }

}
public class Example
{
 public static void main(String[]args) //static member function
 {
  Box b1=new Box();
  b1.setDimension(12,10,5);
 }
}
```

- **Here we use this.l for instance variable l**
- **This represent caller object.**

```
class Box
{
 private int l,b,h;
 public void setDimension(int l,int b,int h) //instance member function
 { this.l=l; this.b=b; this.h=h; }
 public void sendBox()
 {
  GiftTaker gf=new GiftTaker();
  gf.acceptGift(this);
 }
}
public class Example
{
 public static void main(String[]args) //static member function
 {
  Box b1=new Box();
  b1.setDimension(12,10,5);
  b1.sendBox();
 }
}
```

- Here in sentbox function (gf.acceptaGift) caller object represent by use of "this".
- Constructor have also this because they are also instance member function.
- In static function there in no this.

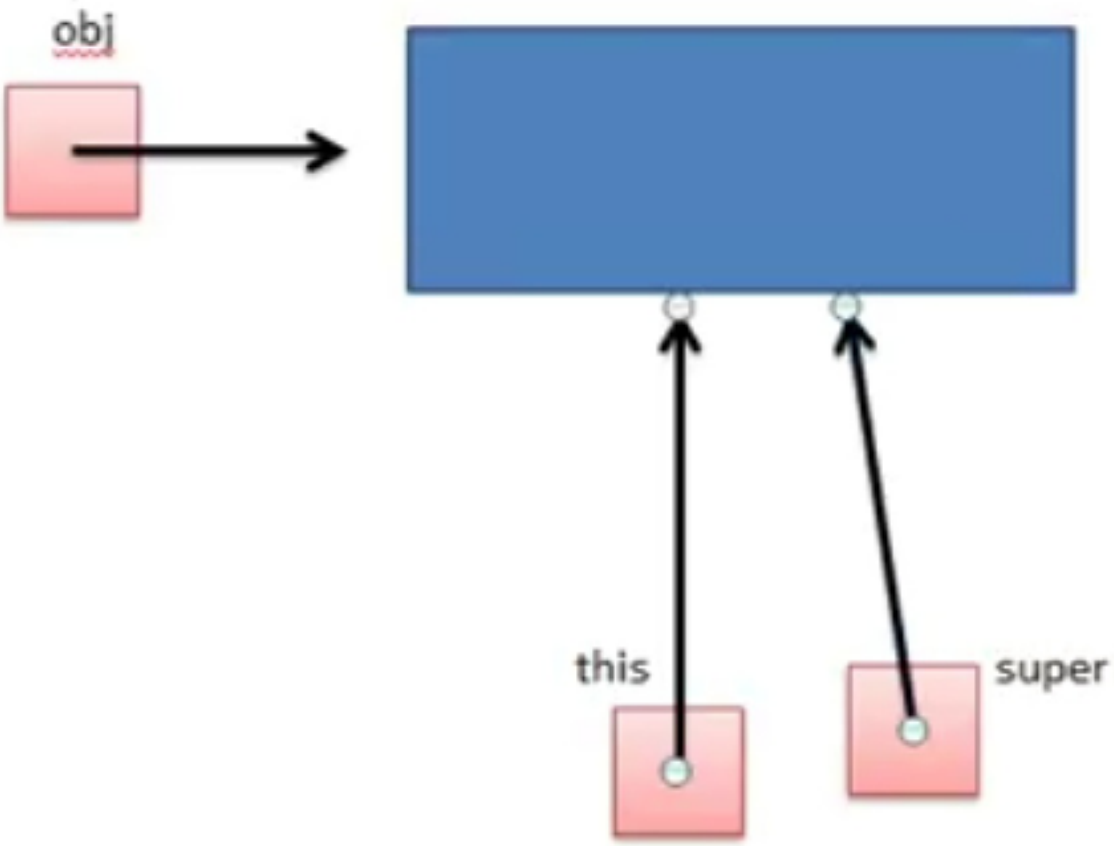
this

- If a method needs to pass the current object to another method, it can do so using the this reference.
- Note that the this reference cannot be used in a static context, as static code is not executed in the context of any object

Lecture 19

super keyword

- In inheritance, subclass object when call an instance member function of subclass only, function contains implicit reference variables this and super both referring to the current object (Object of subclass).
- The only difference in this and super is
  - this reference variable is of subclass type
  - super reference variable is of superclass type



- The only difference in this and super is
  - this reference variable is of subclass type
  - super reference variable is of superclass type

```
1  class A
2  {
3   public void f1()
4   { }
5  }
6  class B extends A
7  {
8   public  void f1()
9   {
10   super.f1();
11  }
12
13 }
14 class Example
15 {
16  public static void main(String[]args)
17  {
18   B obj=new B();
19   obj.f1();
20
21  }
22 }
```

- When chance to overriding and we call from child class version to parent class version then we use SUPER keyword.

Use of super keyword

- If your method overrides one of its superclass's methods, you can invoke the superclass version of the method through the use of the keyword super.

- It avoids name conflict between member variables of superclass and subclass

```
1  class A
2  {
3   int z;
4   public void f1()
5   { }
6  }
7  class B extends A
8  {
9   int z;
10  public  void f1()
11  {
12    super.f1();
13  }
14  public void f2()
15  {
16   int z;
17   z=2;
18   this.z=3;
19   super.z=4;
20  }
21
22 }
23 class Example
24 {
25  public static void main(String[]args)
```

```
{
 int z;
 public  void f1()
 {
   super.f1();
 }
 public void f2()
 {
  int z;
  z=2;
  this.z=3;
  super.z=4;
 }

}
class Example
{
 public static void main(String[]args)
 {
  B obj=new B();
  obj.f1();
  obj.f2();

 }
}
```

Lecture 20

Static member functions

- A class C *inherits* from its direct superclass all concrete methods m (both static and instance) of the superclass
- No method declared in C has same signature
- Ref:http://docs.oracle.com/javase/specs/jls/se8/html/jls-8.4.8

Example

```
class Parent {
  public static void f1()
  { System.out.println("hello"); }
}
class Child extends Parent{
}
public class Example {
 public static void main(String[]args)  {
   Child.f1();
 }
}
```

Function Hiding

- If subclass has a method m with the same signature as of the method present in the superclass, then method m hides the method of superclass

Example

```
class Parent {
   public static void f1()
    { System.out.println("hello"); }
 }
class Child extends Parent{
  public static void f1()
    { System.out.println("yo man"); }
}
public class Example {
 public static void main(String[]args) {
    Child.f1();
 }
}
```

Example

```
class Parent {
   public static void f1()
   { System.out.println("hello"); }
 }
class Child extends Parent{
  public static void f1()
   { System.out.println("yo man"); }
}
public class Example {
 public static void main(String[]args) {
   Child.f1();
 }
}
```

- When static is not present then overriding otherwise function hiding will be here.
- Then child version hide their parent version.

Remember

- **It is a compile-time error if a** static **method hides an instance method**
- **It is a compile-time error if an instance method overrides a** static **method.**

Static member variables

```
class Parent
{
   static int y=4;
 }
class Child extends Parent
{
  static
  { y=5; }
}
public class Example
{
 public static void main(String[]args)
 {
  System.out.println("y="+Child.y);  //y=4
 }
}
```

Static member variables do not inherit

Last modified: 29 Sep 2019