Lecture 20

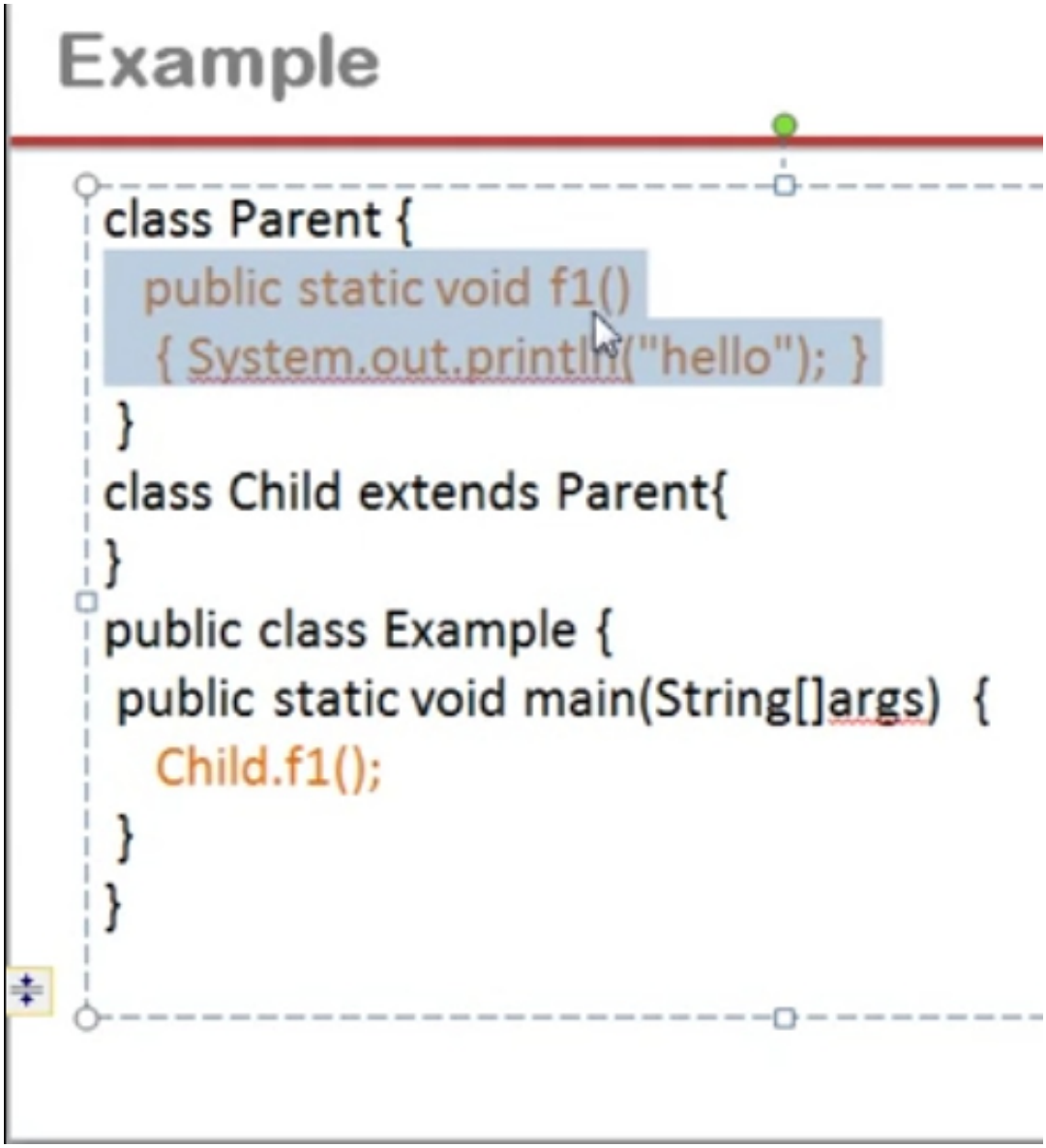
static members in inheritance in JAVA

Lecture 20: (Hindi)

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();
 }
}
```

- When there will same function signature not create in child then parent function will inharit either that function will static function or not.
- We call parent class by Child.fi()

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();
 }
}
```

- There will no over riding ,Over riding will only when there will not static function.
- Here function hiding will happen.

- Here both should static, not will apply one static and another non static it shows compile time error.

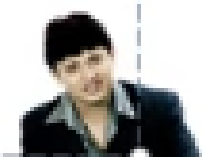
Remember

- 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



- Static member variable do not inharit they should hide.
- Static member variable do not inharit

- Static member function will inharit when will we create same signature in child class function then that function is called function hiding.

Lecture 21

- When object of subclass created then subclass constrictor will call but subclass constrictor will also call parent class.
- In inheritance when we create child class object then also parents class gets memory.

- When we create child class object then child constrictor will call but child class constrictor before they run their code the call parent class constrictor.

Example.java

```
1  class A
2  {
3   int a;
4  }
5  class B extends A
6  {
7   int b;
8  }
9  class Example
10 {
11  public static void main(String[] args)
12  {
13   B obj=new B();
14  }
15 }
```

Constructor in inheritance

- Sub class's constructor invokes constructor of super class.
- Explicit call to the super class constructor from sub class's constructor can be made using super().
- You can write a subclass constructor that invokes the constructor of the superclass, either implicitly or by using the keyword super.

Example.java

```
1  class A
2  {
3   int a;
4   public A()
5   { System.out.println("A");}
6  }
7  class B extends A
8  {
9   int b;
10  public B()
11  { System.out.println("B");}
12 }
13 class Example
14 {
15  public static void main(String[] args)
16  {
17   B obj=new B();
18  }
19 }
```

Command Prompt

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

G:\Java Programs\Example>java Example
A
B

G:\Java Programs\Example>_
```

Example.java

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

- **Super();** represent parent class constrictor.(the line will be first line in constrictor of super).
- This means parent class constrictor will run.

Scenarios

- Implicit constructors in superclass and subclass
- Implicit constructor in subclass and explicit constructor in superclass
- Implicit constructor in superclass and explicit constructor in subclass
- Explicit constructor in superclass and subclass

Parameterized Constructors

- Subclass constructor may take arguments for its use as well as for the constructor of superclass.

Constructor chaining in JAVA

Lecture 22: (Hindi)

Constructor chaining

- Constructor can call other constructors of the same class or superclass
- Constructor call from a constructor must be the first step. (call should appear in the first line)
- Such series of invocation of constructors is known as constructor chaining.

Example.java

```
1  class A
2  {
3   public A()
4   {
5    System.out.println("A 1");
6   }
7  }
8  class B extends A
9  {
10  public B()
11  {
12
13   System.out.println("B 1");
14  }
15 }
16 public class Example
17 {
18  public static void main(String []args)
19  {
20    B o1=new B();
21  }
22 }
```

Command Prompt

```
G:\Java Programs\example>javac Example.java

G:\Java Programs\example>java Example
Error: Could not find or load main class Example

G:\Java Programs\example>javac Example.java

G:\Java Programs\example>java Example
A 1
B 1

G:\Java Programs\example>_
```

Example.java

```
{
 public A()
 {
  System.out.println("A 1");
 }
}
class B extends A
{
 public B()
 {
  this(4);
  System.out.println("B 1");
 }
 public B(int k)
 {
  System.out.println("B 2");
 }
}
public class Example
{
 public static void main(String []args)
 {
   B o1=new B();
 }
```

- This keyword is use for represent same class constrictor.

```
G:\Java Programs\example>javac Example.java

G:\Java Programs\example>java Example
A 1
B 2
B 1

G:\Java Programs\example>_
```

super() or this()

- First line of constructor is either super() or this() (by default super())
- Constructor never contains super() and this() both.

Lecture 23

- **In C++ there is no abstract class keyword.**
- **In both language the concept of abstract class is different.**
- In C++ for making abstract in class must be at least one pure virtual function.
- But in java there is no virtual keyword there is abstract keyword.

- Abstract class is that class which not create object which cannot be instantiated.

Abstract class

- Abstract classes are declared with the abstract keyword.
- An abstract class cannot be instantiated.

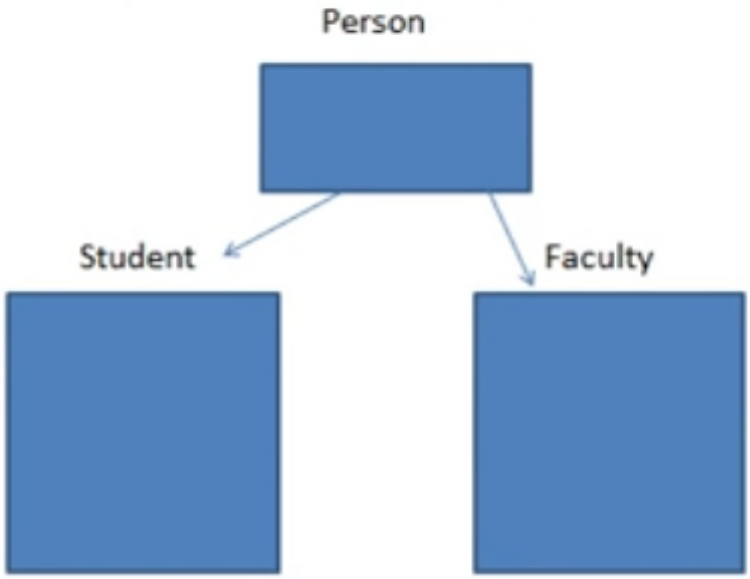
Example

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

class AbstractExample1{
    public static void main(String[] args)
    { Person p=new Person(); } //can't instantiated
}
```

Why Abstract Class Create?

Why Abstract class?



Abstract class

- **Java Abstract classes** are used to declare common characteristics of subclasses.
- It can only be used as a superclass for other classes that extend the abstract class.
- Like any other class, an abstract class can contain fields that describe the characteristics and methods that describe the actions that a class can perform.

Abstract class

- You can not create object of abstract class but you can create reference variable of abstract class

Lecture 24

Abstract Methods in JAVA

Lecture 24: (Hindi)

Abstract methods

- An abstract class can include methods that contain no implementation. These are called abstract methods. The abstract method declaration must then end with a semicolon rather than a block.

Abstract methods

- If a class has any abstract methods, whether declared or inherited, the entire class must be declared abstract

- **When abstract method is in class then the class must be abstract.**
- **And when child class inherit the abstract method then child class must be abstract.**
- **We cannot create an object which have abstract class.**

What is wrong with the code?

```
abstract class Person{
 ...
 abstract void show();
}
abstract class Student extends Person{
 ...
}
 class AbstractExample3{
  public static void main(String[] args){
  Student s=new Student();
  }
}
```

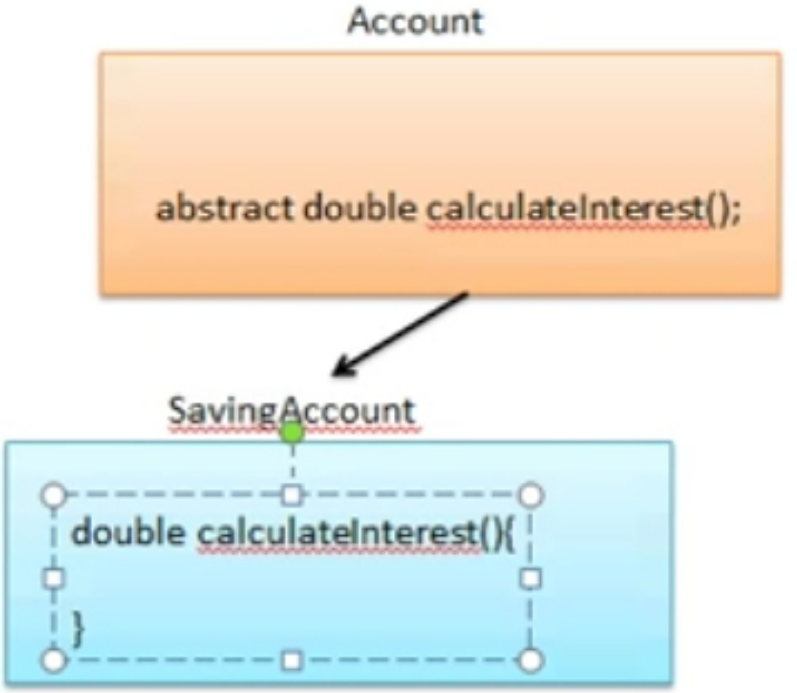
- So there will not be last line.

What is wrong with the code?

```
abstract class Person{
        abstract void show();
}
class Student extends Person{
 void show()
 { //some code }
}
 class AbstractExample3{
  public static void main(String[] args){
  Student s=new Student();
  s.show();
  }
}
```

- **Here show() is abstract method**
- **For creating with student class we are not make it abstract because we define of abstract class from parent class.**
- In student class we override of person class method show,we write code in it.

Why abstract methods?



Lecture 25

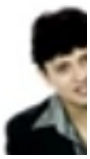
Interface

- Interface definition begins with a keyword interface.

```
interface SomeName
{



}
```

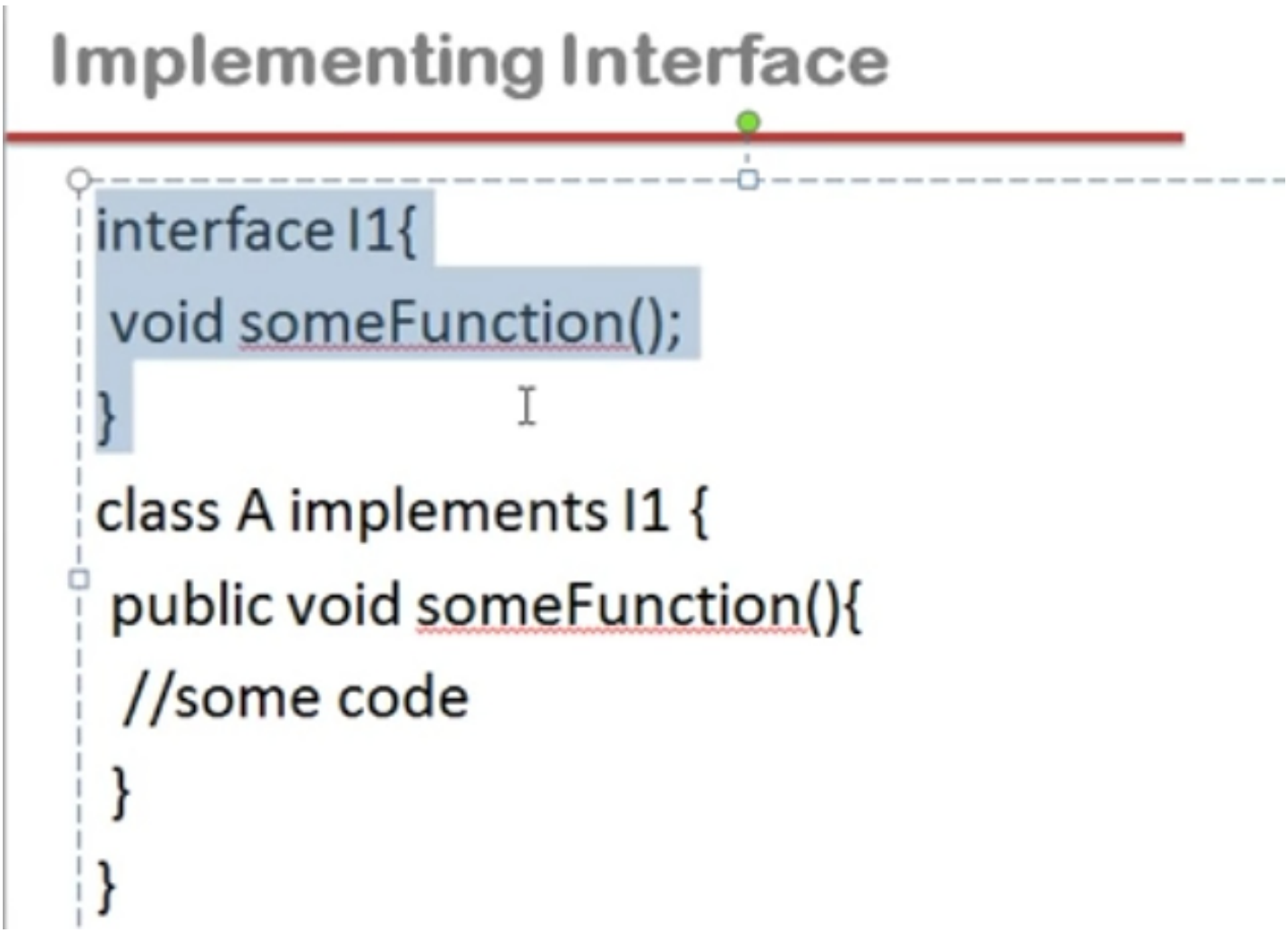


Interface

- Interfaces just specify the method declaration (implicitly public and abstract) and can only contain fields (which are implicitly public static final).

```
interface SomeName
{
  int x;
  void someFunction();
}
```

- In interface there is only function decoration only not implementation.
- **In interface by default all members are public.**
- **Here (variable x) int x; (is final , which when value is set then not changeable)**



Interface

- An interface like that of an abstract class cannot be instantiated.
- Interface do not have constructors.

- Abstract class and interface both haven't object but in abstract class have constrictor, because of there child class object is create, as we know whenever subclass object is create then subclass constrictor will run And they call there parent constrictor.
- But in interface variable always be static. That means static variable doesn't related to object. So in abstract class we create that variable which haven't static i.e instance variable i.e object variable.
- **So in abstract class constrictor is access by object variable because constrictor run for object.**
- So in interface no instance member variable so there will no role of constrictor.(all variable is static so no need of constrictor)

Interface

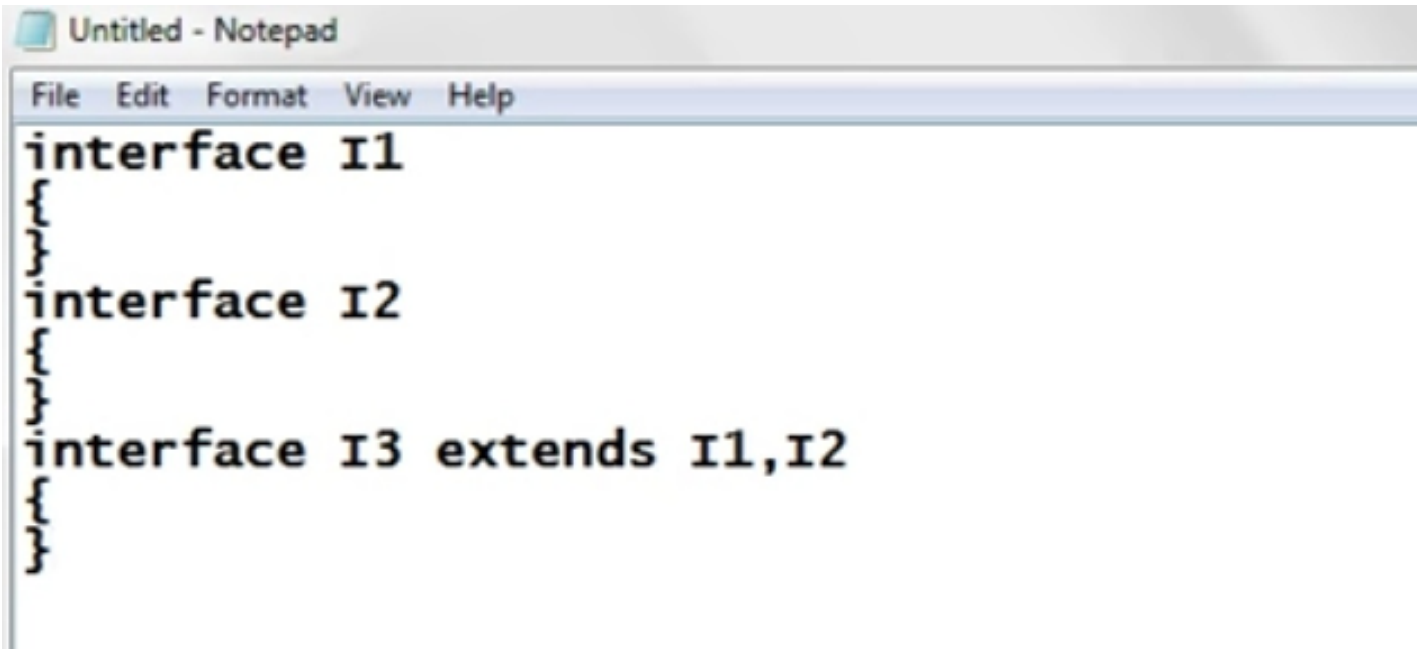
- If a class that implements an interface does not define all the methods of the interface, then it must be declared abstract and the method definitions should be provided by the subclass that extends the abstract class.

Example

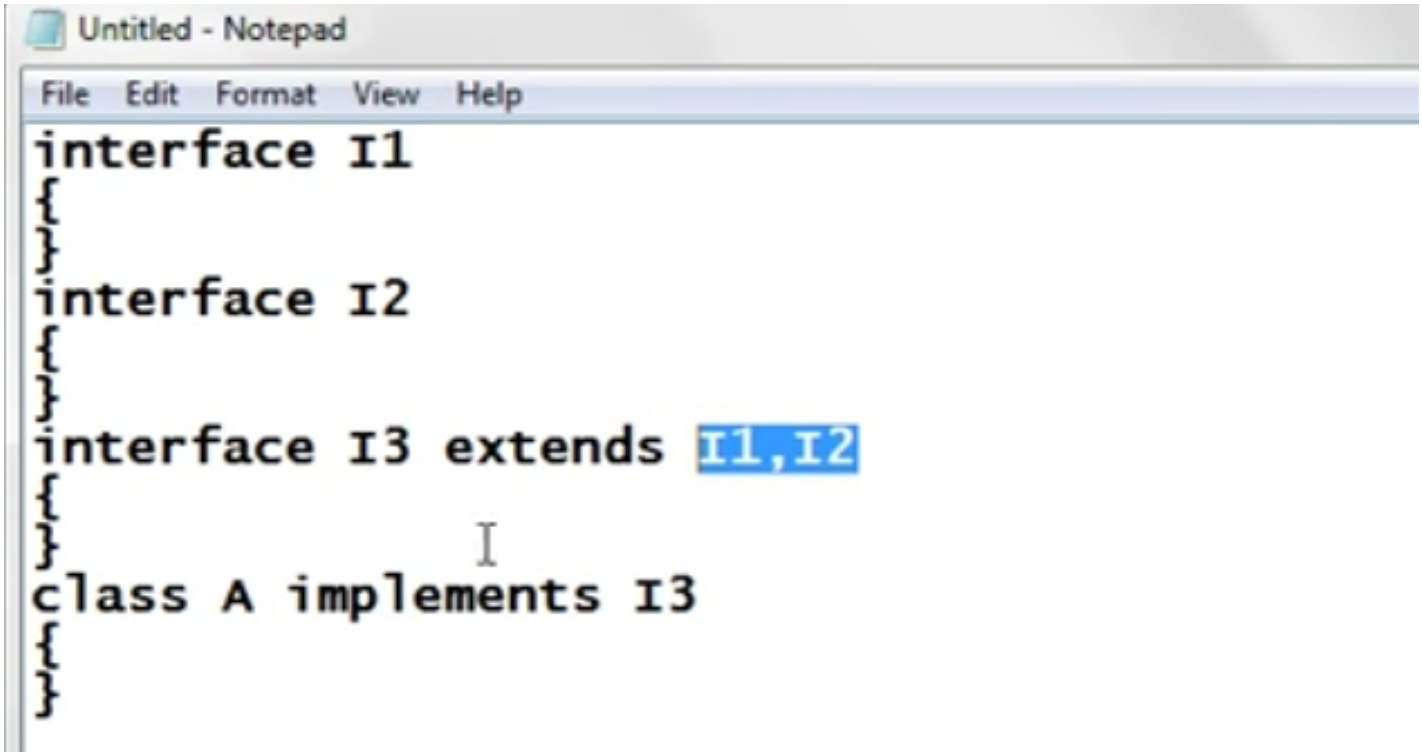
```
interface Admission
{
 int registration();
 int batchAllotment();
 int iCardGeneration();
}
```

Extending and implementing

- Multiple extension is allowed when extending interfaces i.e. one interface can extend none, one or more interfaces.



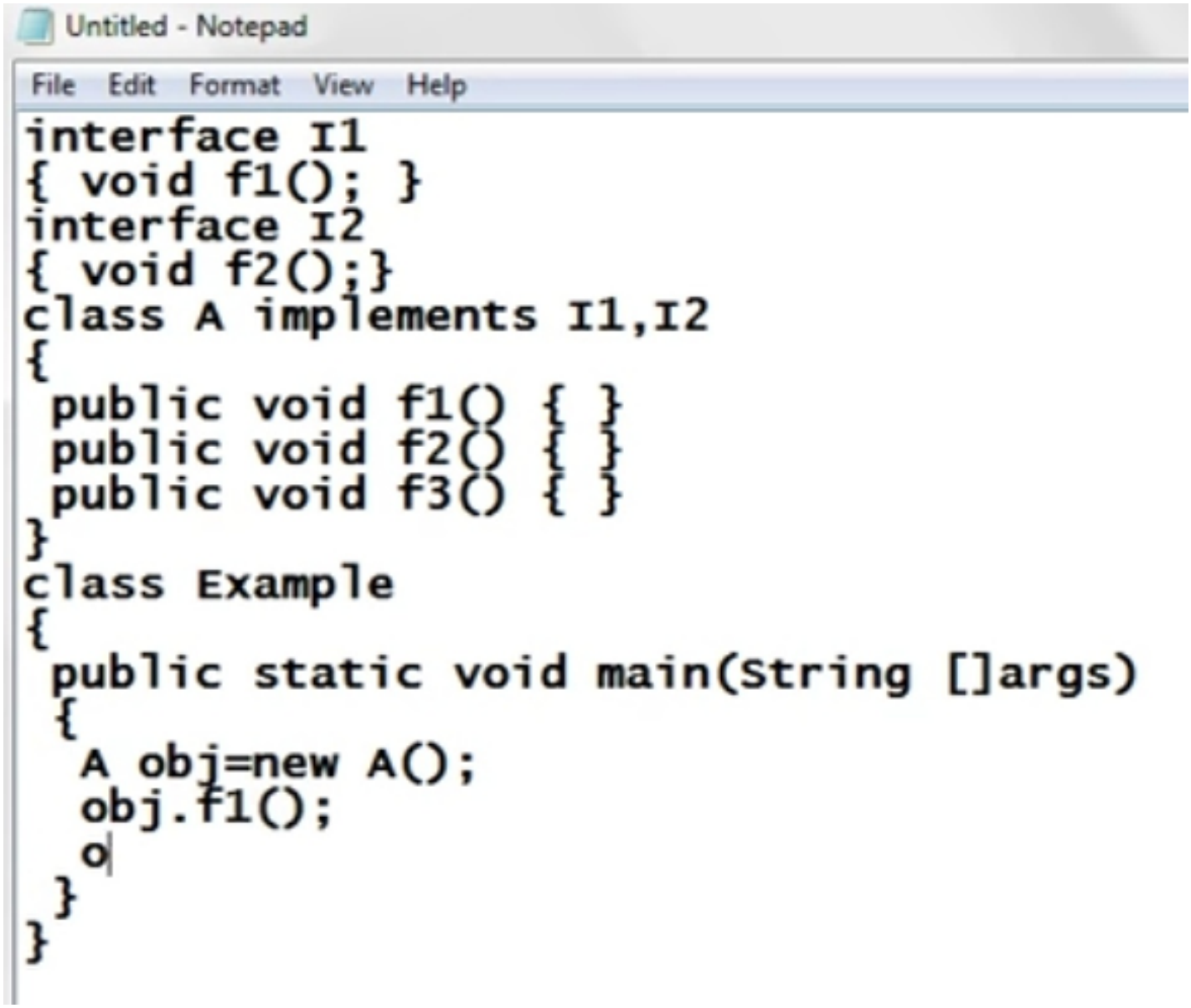
- **It is possible in interface to extends more then on interface.**

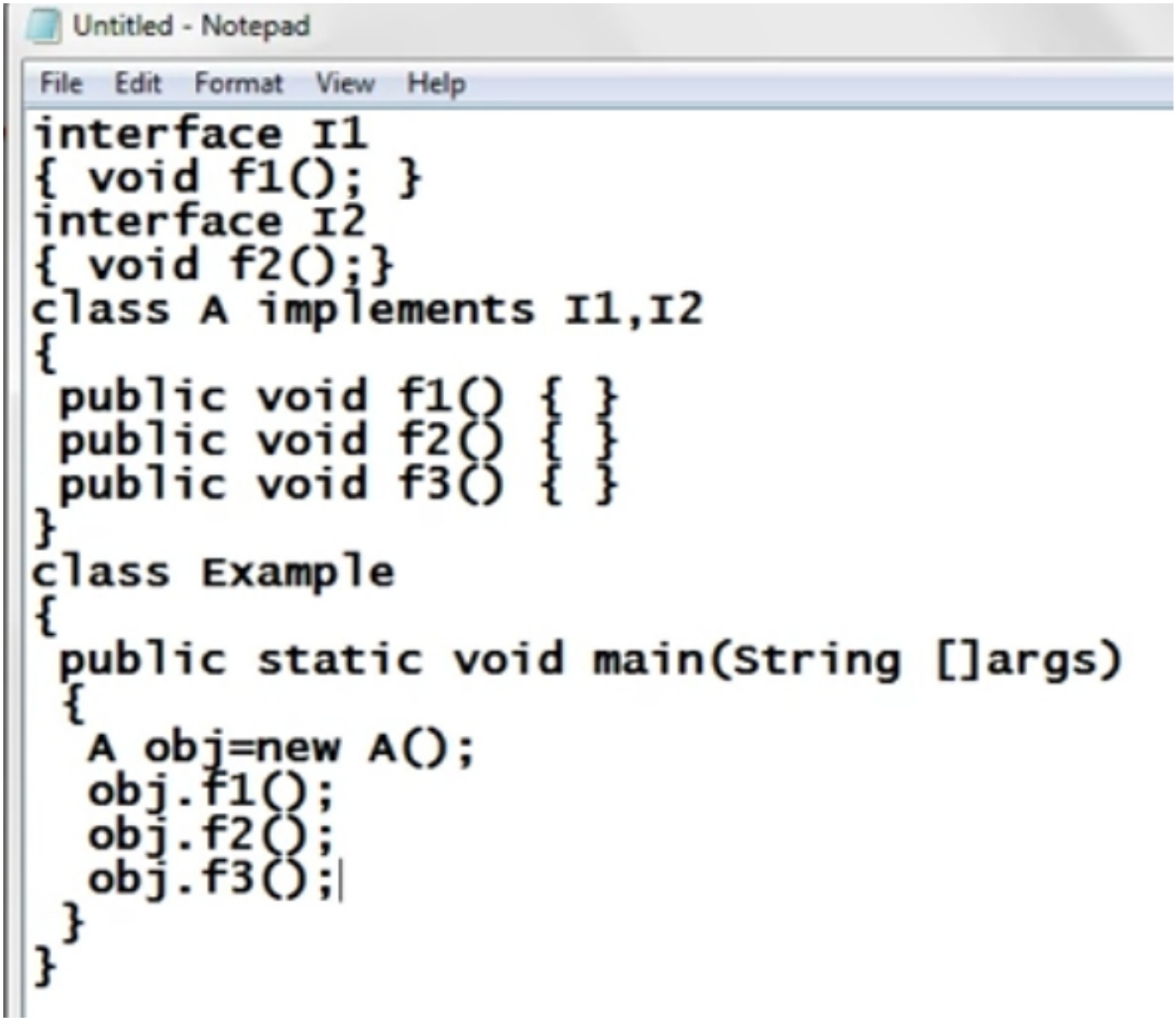


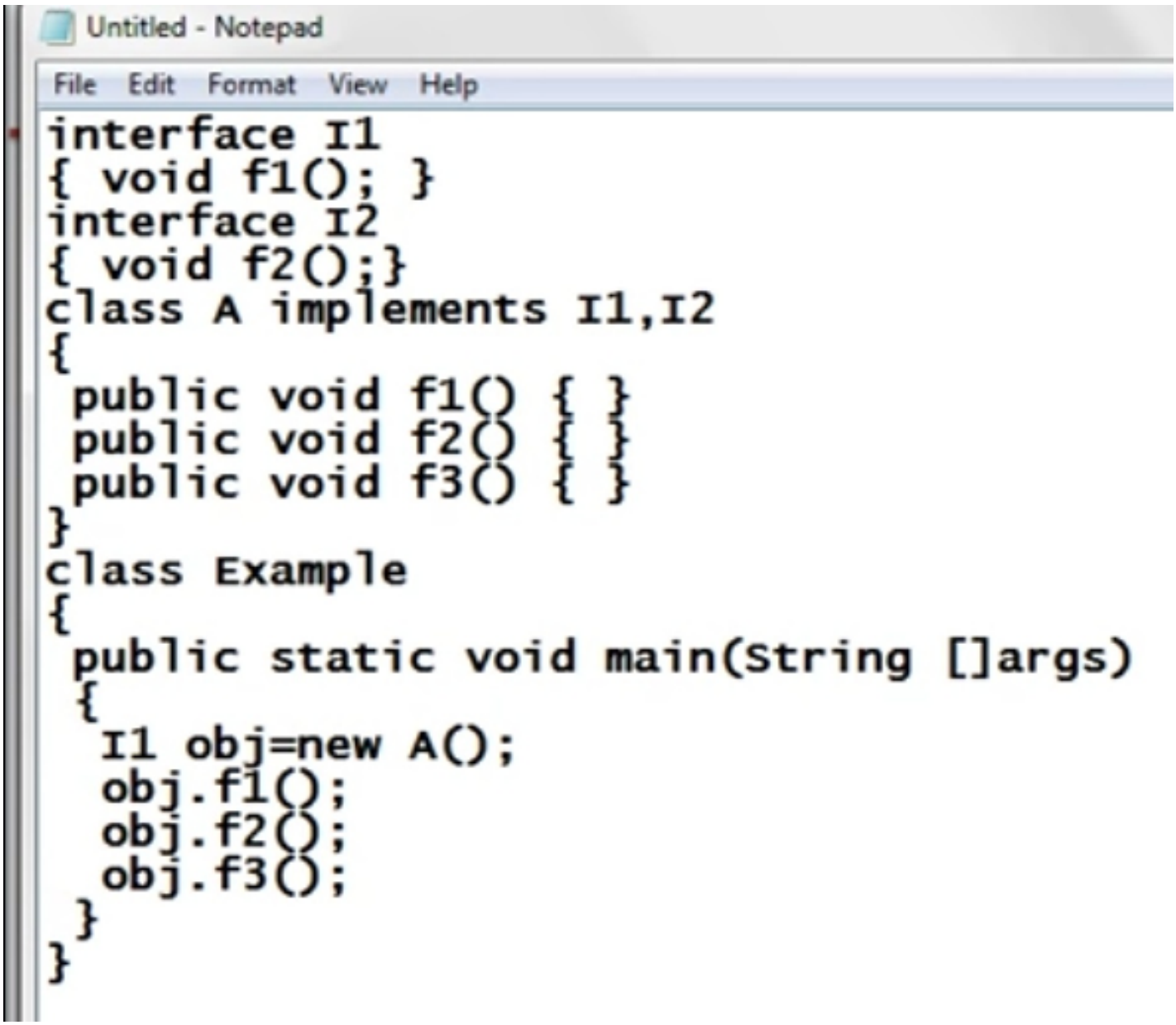
- Class implements interface.
- Class extends one class but implements more then interface.

Object Reference

- You can not create object of any interface but creation of object reference is possible.
- Object reference of interface can refer to any its subclass type.


Untitled - Notepad
File Edit Format View Help
interface I1
{ void f1(); }
interface I2
{ void f2();}
class A implements I1,I2
{
public void f1() { }
public void f2() { }
public void f3() { }
}
class Example
{
public static void main(String []args)
{
A obj=new A();
obj.f1();
o
}
}


Untitled - Notepad
File Edit Format View Help
interface I1
{ void f1(); }
interface I2
{ void f2();}
class A implements I1,I2
{
public void f1() { }
public void f2() { }
public void f3() { }
}
class Example
{
public static void main(String []args)
{
A obj=new A();
obj.f1();
obj.f2();
obj.f3();
}
}



- **In Interface** object will not create but reference variable will create.
- In this program For I1 reference variable f1() will call but f2() and f3() will not call they get error.

Untitled - Notepad

File Edit Format View Help

```
interface I1
{ void f1(); }
interface I2
{ void f2();}
class A implements I1,I2
{
 public void f1() { }
 public void f2() { }
 public void f3() { }
}
class Example
{
 public static void main(String []args)
 {
  I2 obj=new A();
  obj.f1(); //error
  obj.f2();
  obj.f3(); //error
 }
}
```

Lecture 26

Interface and Abstract class

- Abstract class can have any access modifiers for members. Interface can have only public members

```
interface Calculation
{
 double PI=3.14;
 int add(int a, int b);
 int sub(int a, int b);
}
```

```
abstract class Person
{
 private String name;
 public void setName(String n)
{ name=n; }
 public String getName()
 { return(name); }
}
```

Interface and Abstract class

- Abstract class may or may not contain abstract method. Interface can not have defined method.

```
interface Calculation
{
 double PI=3.14;
 int add(int a, int b);
 int sub(int a, int b);
}
```

```
abstract class Person
{
 private String name;
 public void setName(String n)
{ name=n; }
 public String getName()
 { return(name); }
}
```

- In interface all function by default abstract.

Interface and Abstract class

- Abstract class can have static or non static members. Interface can have only static member variables.

```
interface Calculation
{
 double PI=3.14;
 int add(int a, int b);
 int sub(int a, int b);
}
```

```
abstract class Person
{
 private String name;
 public void setName(String n)
{ name=n; }
 public String getName()
 { return(name); }
}
```

Interface and Abstract class

- Abstract class may or may not contain abstract method. Interface can not have defined method.

```
interface Calculation
{
 double PI=3.14;
 int add(int a, int b);
 int sub(int a, int b);
}
```

```
abstract class Person
{
 private String name;
 public void setName(String n)
{ name=n; }
 public String getName()
 { return(name); }
}
```

Interface and Abstract class

- Abstract class can have final or non final members. Interface can have only final member variables.

```
interface Calculation
{
 double PI=3.14;
 int add(int a, int b);
 int sub(int a, int b);
}
```

```
abstract class Person
{
 private String name;
 public void setName(String n)
{ name=n; }
 public String getName()
 { return(name); }
}
```

Interface and Abstract class

- Interface do not have constructor unlike abstract class .

```
interface Calculation
{
 double PI=3.14;
 int add(int a, int b);
 int sub(int a, int b);
}
```

```
abstract class Person
{
 private String name;
 public void setName(String n)
{ name=n; }
 public String getName()
 { return(name); }
}
```

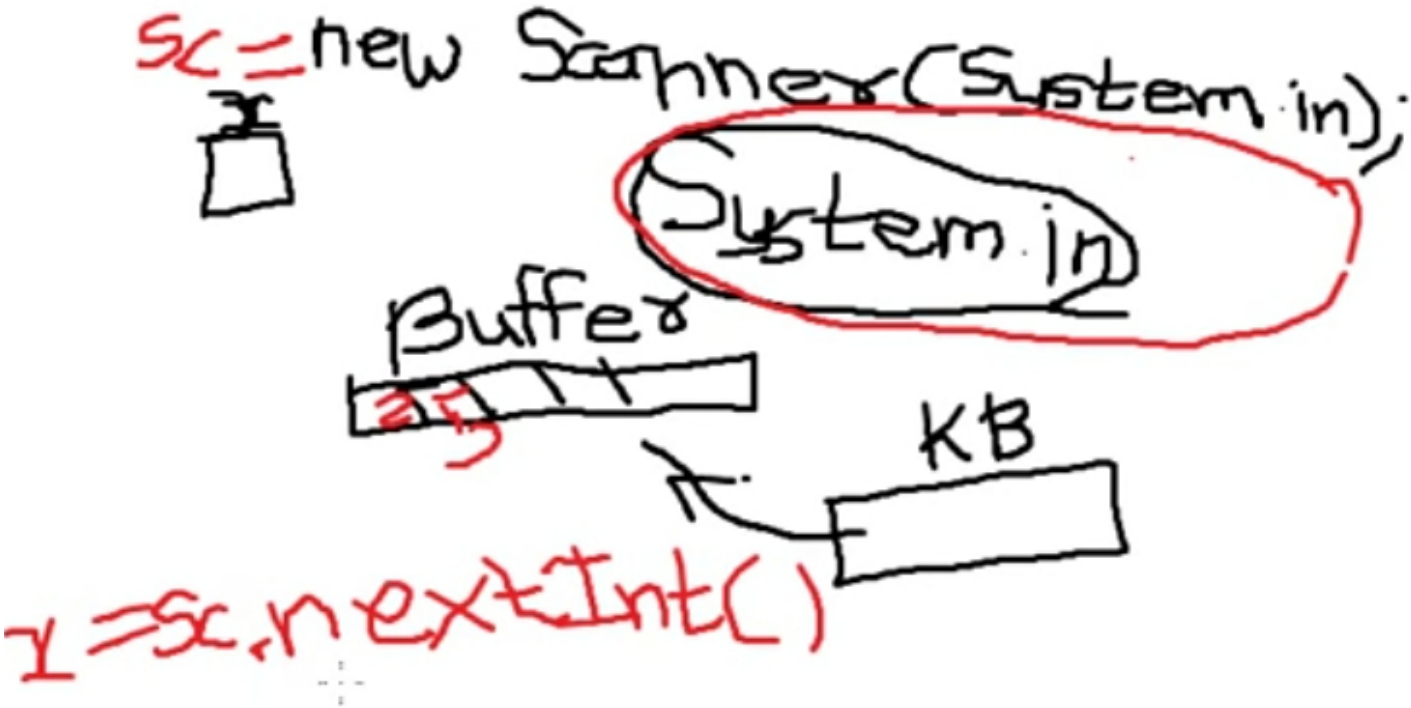
Lecture 27

Taking input from keyboard in JAVA

Lecture 27: (Hindi)

Scanner class

- We can read java input from System.in using Scanner class
- Scanner is final class, that is can not be extended.
- Scanner class is a part of java.util package



- **Here System.in is object (in is object reference).**
- **Data input through keyword is not directly transfer to variable,it is through as intermediate location (Buffer).**
- **In object of scanner class we pass there constrictor is System.in.**

- **System.in responsible for deta transfer through keyword.**
- Scanner class have function nextInt().
- nextInt function collect data from buffer.

Scanner class

- A Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace
- The resulting tokens may then be converted into values of different types using the various next methods.

Example

```
import java.util.*;
class InputExample1{
 public static void main(String []args){
  System.out.println("Enter your name and age:");
  Scanner sc=new Scanner(System.in);
  String name=sc.next();
  int age=sc.nextInt();

  System.out.println("Name: "+name+" Age: "+age);
 }
}
```