Object Oriented Programming
CS F213
Amit Dua

BITS Pilani
Pilani Campus

Slides Taken from the slides prepared by Dr. Jennifer

Questions from prev. class

- Can we access the static(class) methods/variables from the instance/object of the class?

With static methods?

```
class Base {
  static void fun() {
     System.out.println("Base
    fun");
  }
}

class Derived extends Base {
  static void fun() {
     System.out.println("Derived
    fun");
  } }
```

```
class Main {
// Base b1 = new Base()
public static void main(String[ ]
args) {
      Base obj = new Derived();
      obj.fun();
//b1.fun();
  }
}
```

Base fun

non-static variable b1 cannot be referenced from a static context

Ques:

```
class A{
void show()   {
System.out.println("Hello");    }}
class B extends A{
private void show()    { System.out.println("Bye");}}
class Main{
public static void main(String[] args)
{
A a1 = new B();
a1.show();    }}
```

error: show() in B cannot override show() in A
attempting to assign weaker access privileges

Default Methods

```
interface Printable{
void print();
default void show()
{
System.out.println("Within Show");
}
}

class trial implements Printable {

public void print()
{
System.out.println("Within Print");
}
}
```

```
public class test {
public static void main(String[]
    args) {
trial t = new trial();
t.print();
t.show();
}

}
```

Static Methods in Interfaces


```
interface Printable{
void print();
static void show()
{
System.out.println("Within
    Printable Show");
}
}

class trial implements Printable {
public void print()
{
System.out.println("Within Print");
}
}

public class test {
public static void main(String[]
    args) {
trial t = new trial();
t.print();
Printable.show();
}
}
```

Question:
Can we replace Printable.show() with t.show()?

Static Methods in Interfaces

```
interface Printable{
void print();
static void show()
{
System.out.println("Within
    Printable Show");
}
}
class trial implements Printable {
public void print() {
System.out.println("Within Print");}
static void display()
{
System.out.println("Within
    Display");
}}
```

```
public class test {
public static void main(String[]
    args) {
trial t = new trial();
t.print();
t.display();
t.show();     //Error
 }
}
```

Default Methods & Multiple Inheritance

innovate achieve lead

```
interface Printable{
void print();
default void show()
{
System.out.println("Within
    Printable Show");
}
}
interface Showable{
default void show()
{
System.out.println("Within
    Showable Show");
}
void print();
}
```

```
class trial implements Printable,Showable{
public void show()  {
Printable.super.show();
Showable.super.show(); }

public void print() {
System.out.println("Within Print"); }}

public class test {
public static void main(String[] args) {
trial t = new trial();
t.print();
t.show();
}
}
```

Question:
What happens if super keyword is omitted?

innovate achieve lead

BITS Pilani

Pilani Campus

Nested Interfaces

Nested Interfaces

- Interface can be declared within another interface or class
- Nested interface cant be accessed directly, it is referred by the outer interface or class
- Nested interface must be public if it is declared inside the interface but it can have any access modifier if declared within the class.
- Nested interfaces are declared static implicitly.

Class Implementing Outer Interface


```
interface Printable{
void print();
interface Showable{
void show(); }
}

class trial implements Printable {
public void print()
{
System.out.println("Within Print");
}
public void show() {
System.out.println("Within Show");
}
}
```

```
public class test {
public static void main(String[]
    args) {
trial t = new trial();
t.print();
t.show();
}
}
```

Question:
What happens when implementation of show() is removed from class trial?

Class Implementing Outer Interface

```
interface Printable{
void print();
interface Showable{
void show(); }
}

class trial implements Printable {
public void print()
{
System.out.println("Within Print");
}
}
```

```
public class test {
public static void main(String[]
    args) {
trial t = new trial();
t.print();
}
}
```

Answer: Nothing happens. Outer interface does not have access to inner interface.

Class Implementing Inner Interface

```
interface Printable{
void print();
interface Showable{
void show();}
}
class trial implements
    Printable.Showable {
public void print1()
{
System.out.println("Within Print");
}
public void show()
{
System.out.println("Within Show");
}
}
```

```
public class test {
public static void main(String[]
    args) {
trial t = new trial();
t.print1();
t.show();
}

}
```

Note: If we omit the implementation of show() method, we get compilation error

Interface within the Class

```
class Printable{
public void print()
{
System.out.println("Within Print");
}
interface Showable{
void show();}
}

class trial implements
    Printable.Showable {
public void show()
{
System.out.println("Within Show");
} }
```

```
public class test {
public static void main(String[]
    args) {
trial t = new trial();
t.show();
t.print();   //print undefined for the type
    trail
}
}
```

Interface within the Class

```
class Printable{
public void print()
{
System.out.println("Within Print");
}
interface Showable{
void show();}
}

class trial extends Printable
    implements Printable.Showable
    {
public void show()
{
System.out.println("Within Show");
} }
```

```
public class test {
public static void main(String[]
    args) {
trial t = new trial();
t.show();
t.print();
}
}
```

Output:
Within Show
Within Print

Class within the Interface

```
interface Showable{
class Printable{
public void print()
{
System.out.println("Within Print");
}}
void show();
}

class trial extends
    Showable.Printable {
public void show()
{
System.out.println("Within Show");
} }
```

```
public class test {
public static void main(String[]
    args) {
trial t = new trial();
t.show();
t.print();
}
}
```

Output:
Within Show
Within Print

Class within the Interface

```
interface Showable{
class Printable{
public void print()
{
System.out.println("Within Print");
}}
void show1();
}
class trial extends
    Showable.Printable implements
    Showable {
public void show()
{
System.out.println("Within Show");
} }
```

```
public class test {
public static void main(String[]
    args) {
trial t = new trial();
t.show();
t.print();
}
}
```

Error:
Class trial should implement the method show1()

What happens if the class does not implement all members of the Interface?

innovate achieve lead

```
interface Printable{
void print();
void show();
}
abstract class trial implements Printable {
public void print() {
System.out.println("Within Print");
}}

public class test {
public static void main(String[] args) {
trial t = new trial();
t.print();}
}
```

Error:
Cannot Instantiate trial

(Because trail is an abstract class)

What happens if the class does not implement all members of the Interface?


```
interface Printable{
void print();
void show(); }

abstract class trial implements Printable {
public void print() {
System.out.println("Within Print");}
}

public class test extends trial {
public void show() {
System.out.println("Within Show");}
public static void main(String[] args) {
test t = new test();
t.print();
t.show();}}
```

Output:
Within Show
Within Print

Ques

- What is the output?

```
class Parent{
static int A=50;
static void show() {
System.out.println(A);
}}

class Child extends Parent{
int A=10;
int show() {
System.out.println(A);
}}
```

Compilation Error:
Instance method cannot override a static method from parent

Ques

```
class Parent{
static int A=50;
static void show() {
System.out.println(A);
}}

class Child extends Parent{
int A=10;
}
class test{
public static void main(String args[]) {
Child c = new Child();
c.show();
System.out.println(c.A);    }
}
```

Output:
50
10

Warning:
The static method show() from the type Parent should be accessed in a static way

Ques

```
interface Printable{
int data=20;
class Showable{
void show()
{
System.out.println("Interface Variable "+data);
}
}
}
class test extends Printable.Showable{
public static void main(String args[]) {
test c = new test();
c.show();
}}
```

Output:
Interface Variable 20

innovate achieve lead

BITS Pilani

Pilani Campus

Nested Classes

Inner Classes

- Nested classes are used to logically group classes or interfaces in one place, for more readability and maintainability.
- Nested class can access all members of the outer class including the private data members and methods.
- Two types:
  - Non-static nested class (inner class)
  - Static nested class

Member Inner class - Example

innovate achieve lead

```
class Outer{
int data = 30;
private int val = 20;
class Inner{
void show() {
System.out.println("Data=:"+data+"Value="+val);} }
}
class test{
public static void main(String args[]) {
Outer o = new Outer();
Outer.Inner in = o.new Inner();
in.show();
} }
```

Member Inner Class

- Compiler creates two class files of the inner class
  - Outer.class and Outer$Inner.class
- To instantiate the inner class, the instance of the outer class must be created
- The inner class have a reference to the outer class, thus it can access all the data members of the outer class.

Anonymous Inner Class

- Class with no name
- Used when a method or interface is to be overridden

Anonymous class - Example

```
abstract class Outer{
int data = 30;
abstract void show();
void print() {
System.out.println("Within Print");
}
}
class test {
public static void main(String args[]) {
Outer o = new Outer() {
void show() {
System.out.println("Data=:"+data);}
};
o.show();
o.print();
}}
```

Anonymous Class

- The name of the class created is decided by the compiler
- In the given example, the anonymous class extends the ‘Outer’ class and gives implementation for the show() method.
- The object of the anonymous class can be referred by the reference variable ‘o’
- Anonymous class cannot have additional methods because it is accessed using the reference to the ‘Outer’ class

Anonymous Inner Class using Interface-Example


```
interface Outer{
int data = 30;
void show();
}
class test {
public static void main(String args[]) {
Outer o = new Outer() {
public void show() {
System.out.println("Data=:"+data);
//data =25;          // Error
}
};
o.show();
}
}
```

Local Inner Class-Example

```
class Outer{
private int data = 30;
void show() {
int val =50;
class inner{
void print() {
System.out.println("Value= "+val+"Data="+data);}}
inner i =new inner();      // Creating a named type
i.print(); }
}
class test {
public static void main(String args[]) {
Outer o = new Outer();
o.show();
//o.print();  //Error}
}
```

Note:
Local inner class can be instantiated only within the method it is defined.

Static Inner Class-Example

```
class Outer{
static int data = 30;
private static int val = 20;
static class Inner{
void show() {
System.out.println("Data=:"+data+"Value="+val);
}
}
}
class test{
public static void main(String args[]) {
Outer.Inner in = new Outer.Inner();
in.show();
}
}
```

Static Inner Class

- A static class created inside a class.
- It can access the static data members of the outer class including the private members.
- It cannot access the non-static members and methods.
- The object of the outer class need not be created, because static methods or classes can be accessed without object.
- **Note:** Only inner classes can be prefixed with the static keyword.

Take Home Exercise: What happens when the Interface Showable is private? Update the following code.


```
class Printable{
public void print()
{
System.out.println("Within Print");
}
interface Showable{
void show();}
}

class trial extends Printable
    implements Printable.Showable
    {
public void show()
{
System.out.println("Within Show");
} }
```

```
public class test {
public static void main(String[]
    args) {
trial t = new trial();
t.show();
t.print();
}
}
```

Output:
Within Show
Within Print

innovate achieve lead

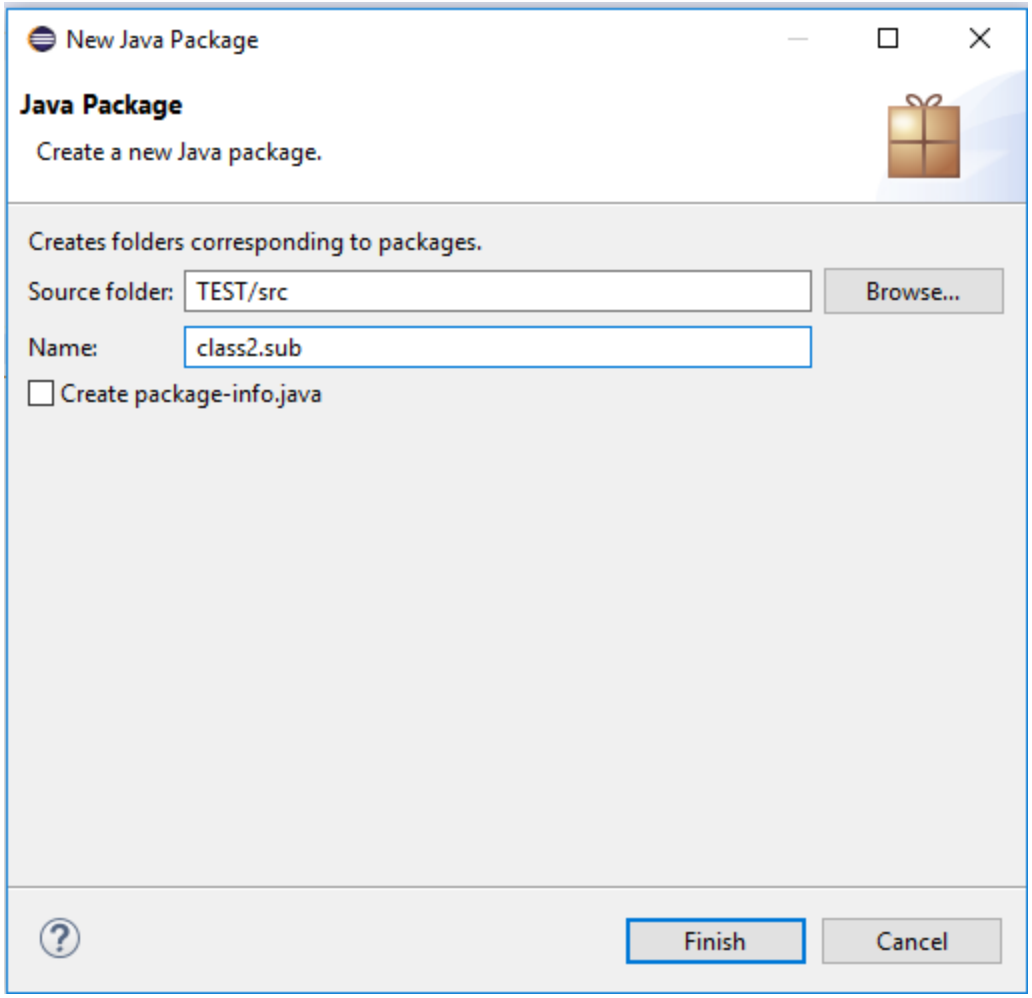
BITS Pilani

Pilani Campus

Packages

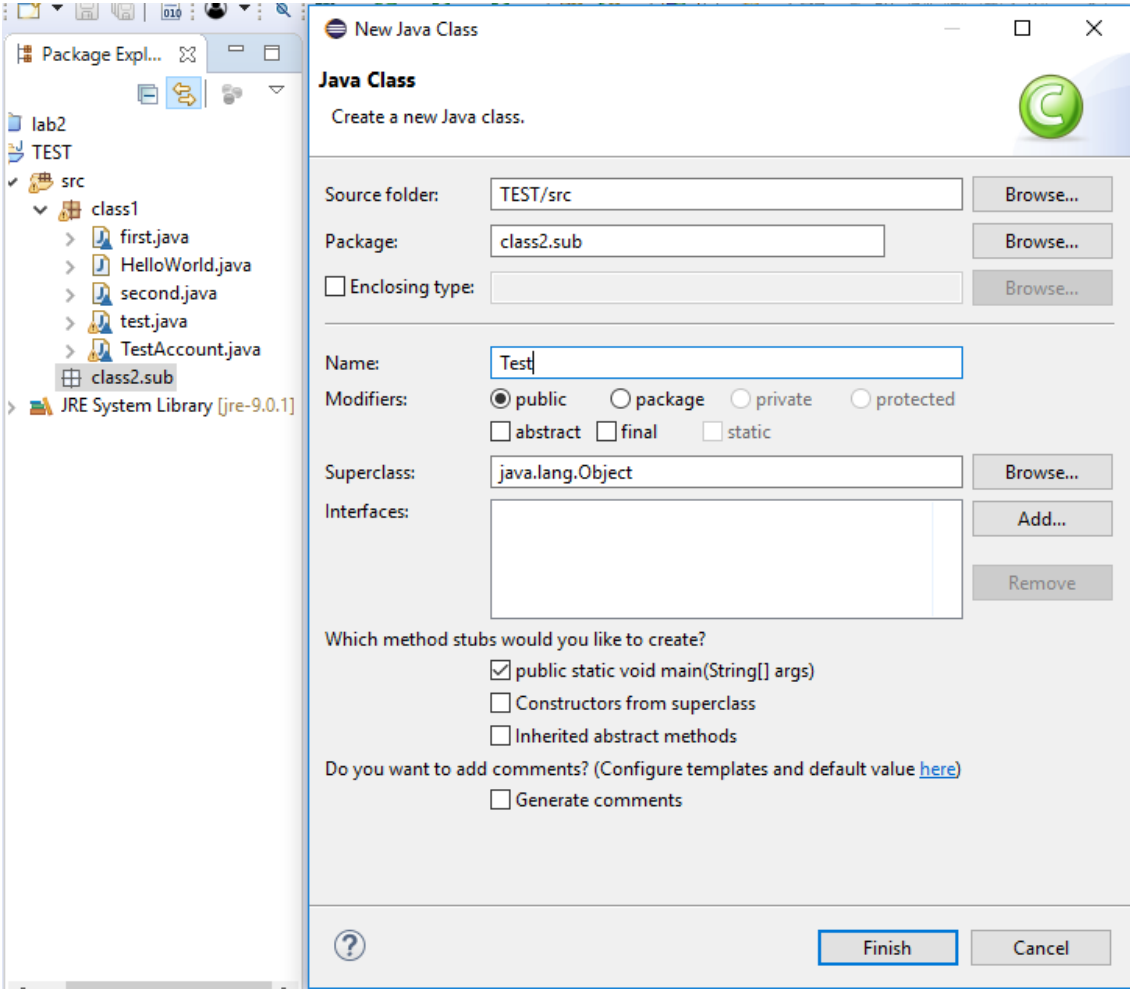
Create a package & sub package

Project → New → Package



Create a class within the package

Package → New → class



Class within the package

```
package class2.sub;

public class Test {

public static void main(String[] args) {
// TODO Auto-generated method stub

}

}
```

Importing a package

```
package class1;

public class HelloWorld
{
 public void show() {
  System.out.println("Within class
     1's show");
  }
}
```

```
package class2.sub;
import class1.*;

public class Test {

public static void main(String[]
     args) {
HelloWorld h = new HelloWorld();
h.show();

}

}
```

Importing a class

```
package class1;

public class HelloWorld
{
 public void show() {
  System.out.println("Within class
     1's show");
  }
}
```

```
package class2.sub;
import class1.HelloWorld;

public class Test {
public static void main(String[]
     args) {
HelloWorld h = new HelloWorld();
h.show();
}
}
```

Take Home Exercise: Learn how to execute the same code from the command prompt.

Access Modifiers

innovate achieve lead

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y