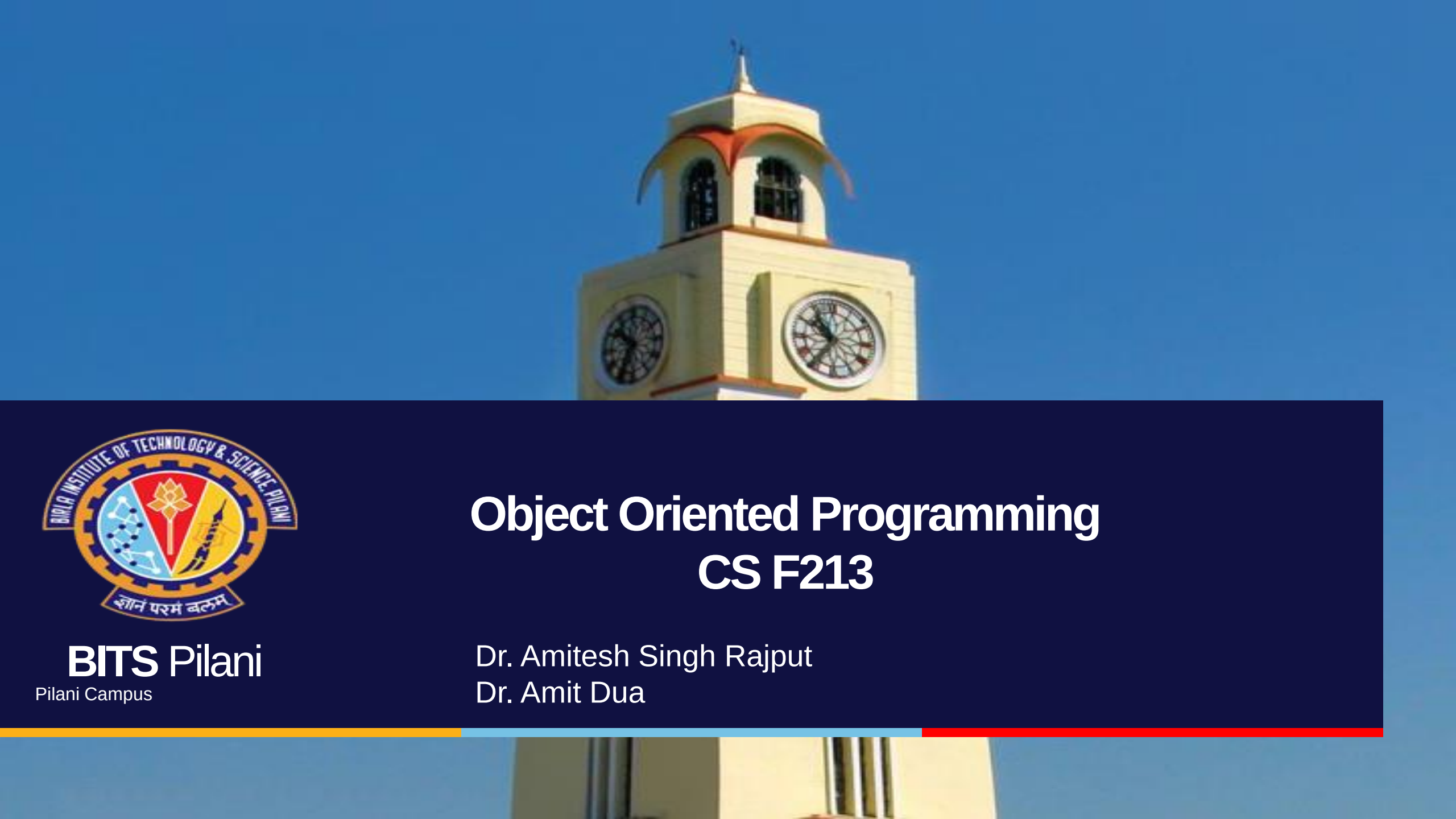

Object Oriented Programming
CS F213

BITS Pilani
Pilani Campus

Dr. Amitesh Singh Rajput
Dr. Amit Dua

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();
   }
}
```

Default Methods & Multiple Inheritance

innovate achieve lead

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

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

```
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

Static Methods in Interfaces

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

Apart from default method, we can have definition for static methods in an interface.

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

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

```
Within Print
Within Printable
```

Static Methods in Interfaces


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

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

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

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

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?

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 1");
 }
 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();
 }
}
```

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

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

```
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();
}
```

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

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

```
Error:
Class trial should implement the method
show1()
```

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{
 static void show(){
   System.out.println("Within Static Show");
 }
}

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

```
class trial implements Printable,Showable{

}

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

```
Output:
Within default Show
```

Ques

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

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

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

```
Output:
Within Show
Within default Show
Within Static Show
```

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


```
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.
- Only a single object is created.
- Used when a method or interface is to be overridden.
- **Syntax:** The syntax of an anonymous class expression is like the invocation of a constructor, except that there is a class definition contained in a block of code.

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();
}}
```

- 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);
         }
        };
   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();
   i.print();
 }
}
```

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

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

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();
   }
}
```

- 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

Thank You!