Essentials of Information Technology

PC-CS-305

Abstract Classes

Objectives

In this lecture, we will
- Discuss Abstract classes
- Build class hierarchies based on abstract classes
- Introduce interfaces
- Discuss implementing and extending within a single class

gauravgambhir.cse@piet.co.in

What are Abstract Classes

- Abstract classes are classes which cannot be instantiated, means you cannot create new instances of an abstract class.
- The purpose of an abstract class is to function as a base for subclasses.

gauravgambhir.cse@piet.co.in

Declaration of Abstract class

```
public abstract class MyAbstractClass {

}
```

A class is declared abstract by adding the abstract keyword.

Its not possible to create instance of abstract class.

The following code is no longer valid.

```
MyAbstractClass c = new MyAbstractClass();
```

If you try to compile the code above the compiler will generate an error.

gauravgambhir.cse@piet.co.in

Declaring abstract methods

- An abstract class can have abstract methods. A method is declared abstract by adding abstract keyword in front of method declaration.

```
public abstract class MyAbstractClass {
                public abstract void abstractMethod();
}
```

An abstract method has no implementation. Its just a method signature.

If a class has an abstract method , the whole class must be declared abstract.

gauravgambhir.cse@piet.co.in

Declaring abstract methods

- Not all the methods have to be abstract even if class is abstract.
- An abstract class can have a mixture of abstract and non abstract classes
- Subclasses of an abstract class must implement (override) all abstract methods of its abstract superclass.
- The non-abstract methods of the superclass are just inherited as they are. They can also be overridden, if needed.

gauravgambhir.cse@piet.co.in

Declaring abstract methods

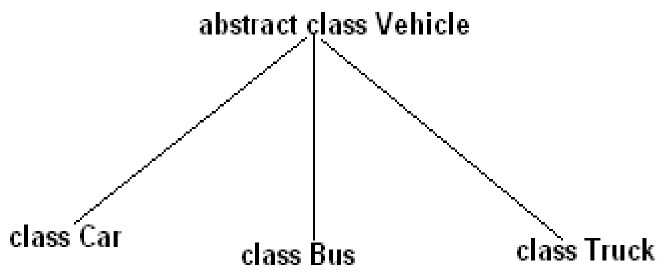
Signature of subclass of MyAbstractClass

```
public class MySubClass extends MyAbstractClass {
     public void abstractMethod() {
          System.out.println("My method implementation");
     }
}
```

Example : MyAbstractClass.java , MySubClass.java

gauravgambhir.cse@piet.co.in

Where we use abstract class



- You have a method like takepassenger() in Car and Bus, sellTicket() in Bus and collectGoods() in Truck, these methods would be different in all cases.

gauravgambhir.cse@piet.co.in

Where we use abstract class

- You have a method buyFuel() in Vehicle, which is abstract because Car might use petrol/gasoline, a truck might use Diesel and Bus might use electricity, you might have methods like turnRight() turnLeft() go() stop() implemented in vehicle because they are the same in all three subclasses.
- You would never want an object of Vehicle class , only Car Bus and Truck, but all three of these objects are also objects of Vehicle.
- **Example :** Vehicle.java, Car.java, Bus.java, Truck.java

gauravgambhir.cse@piet.co.in

Where we use abstract class

```
abstract class  Vehicle{

    abstract void buyFuel();

    void turnRight(){

    }

    void turnLeft(){
    }

    void go(){
    }

    void stop(){
    }
}
```

gauravgambhir.cse@piet.co.in

Where we use abstract class

```
class Car extends Vehicle {

     void takePassenger(){
     }

     void buyFuel(){
          System.out.println("Petrol cost Rs 70/lit");
     }
 }
class Bus extends Vehicle
{
     void takePassenger(){
     }

     void sellTicket(){
     }

     void buyFuel(){
          System.out.println("Electricity cost Rs 5/unit");
     }
}
```

Where we use abstract class

```
class Truck extends Vehicle
{
     void collectGoods(){
     }

     void buyFuel(){
          System.out.println("Diesel cost Rs 40/lit");
     }
}
```

Abstract vs. Final methods

- Abstract methods are always overridden, final methods can never be overridden.
- **Example** AbstractFinalDemo.java , AbstractFinalChild.java

gauravgambhir.cse@piet.co.in

Abstract vs. Final methods

```
abstract class AbstractFinalDemo {

    abstract void abstractmethod();

    final void finalmethod(){
    }
}

class AbstractFinalChild extends AbstractFinalDemo{

    void abstractmethod(){
    }

    final void finalmethod(){
    }


}
```

gauravgambhir.cse@piet.co.in

Examples of Abstract Class

- Player.java, GameObject.java
- Shape.java, Rectangle.java, Circle.java

```
public abstract class  GameObject
{
    public static void main(String[] args)
    {
        //abstract class cannot be instantiated
        //GameObject g = new GameObject();
    }
}

public class  Player extends GameObject
{
    public static void main(String[] args)
    {
        Player p = new Player();
    }
}
```

ımbhir.cse@piet.co.in

Interfaces

- An interface is a collection of abstract methods.
- An interface is similar to abstract class but its pure abstract class, means it doesn't provide definition of any method.
- An abstract class is always extended, but in case of interface a class implements an interface there by inheriting the abstract methods of interface.
- An interface is not a class , writing an interface is similar to writing a class
- **Example I.java**

gauravgambhir.cse@piet.co.in

Interfaces

- A class describes the attributes and behaviors of an object, an interface contain behavior that a class implements.
- All the methods of the interface need to be defined in the class that implements the interface, otherwise the class is declared as abstract.
- Interface is a contract between implementing class and the outside world.

gauravgambhir.cse@piet.co.in

Interfaces

```
// interface is an agreement that class has to implement
interface I
{
//1.Methods in interface
  void abc();//By default methods are public abstract
  int def();

//2.Constants in interface
  int x=10;//By default public static final i.e constant

//3. Instance fields in interface
//  int y;

//4. Methods with body in interface
  //will give compile time error
  /*
  void pqr(){
  }
  */

//5. Constructors in interface
 /*  I(){
     }
  */
}
```

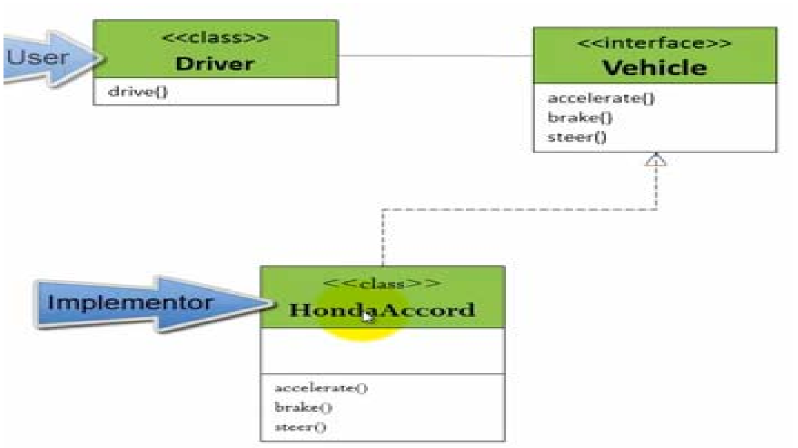
[illegible]t.co.in

Interfaces

```
class P implements I // means class follows the agreement
{

  public void abc(){
    System.out.println(x);
    //interface cannot be instantiated
    //I i = new I();
  }

  public int def(){
      return 0;
  }

}
```

gauravgambhir.cse@piet.co.in

Understanding Interface with example



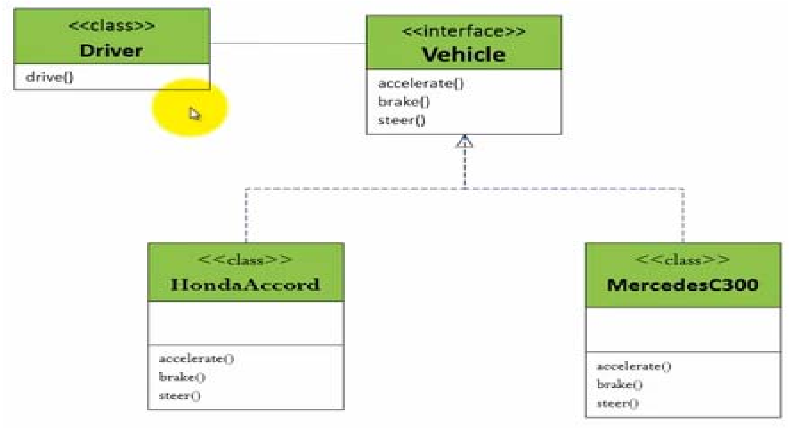
gauravgambhir.cse@piet.co.in

Understanding Interface with example

- Vehicle is an interface that defines the contract that includes operations accelerate , brake and steer.
- Driver drives the vehicle using these methods defined in vehicle contract.
- Vehicle contract is implemented by Honda Accord class, so this class has to implement accelerate , brake and steer methods.
- So the interface is helping us to separate the user of interface from implementer of interface.
- Now if the driver want to drive the mercedesc300 instead of Honda Accord, so a new class is created mercedes c300 which implements the vehicle interface, changes in driver are minimal, it is this reason we are encourage to use interface, instead of class directly.
- Example Vehicle.java , Mercedesc300.java, HondaAccord.java

gauravgambhir.cse@piet.co.in

Understanding Interface with example



gauravgambhir.cse@piet.co.in

Understanding Interface with example

```
public interface Vehicle
{
    public void accelerate();
    public void brake();
    public void steer();

}

 public class  HondaAccord implements Vehicle
 {
     public void accelerate(){
         System.out.println("In HondaAccord.accelerate()");
     }
     public void brake(){
     }
     public void steer(){
     }
 }
```

gauravgambhir.cse@piet.co.in

Understanding Interface with example

```
public class  Mercedesc300 implements Vehicle
{
    public void accelerate(){
        System.out.println("In Mercedesc300.accelerate()");
    }
    public void brake(){
    }
    public void steer(){
    }
}
   public class  Driver
   {
       public static void main(String[] args)
       {
           Vehicle myVehicle1=new HondaAccord();
           Vehicle myVehicle2=new Mercedesc300();
           myVehicle2.accelerate();
       }
   }
```

gauravgambhir.cse@piet.co.in

Interface similar to class

An interface is similar to class in following ways

An interface can contain any number of methods.

An interface is written in a file with a .java extension, with name of interface matching name of file.

The byte code of an interface appears in a .class file.

gauravgambhir.cse@piet.co.in

Interface different from class

Interface is different from class in following ways

You cannot instantiate an interface.

An interface doesn't contain any constructor.

All the methods in an interface are abstract.

An interface cannot contain instance fields, the only field that can appear in an interface must be declared both static and final.

An interface is not extended by class it is implemented by a class.

An interface can extend multiple interfaces.

gauravgambhir.cse@piet.co.in

Declare an interface

An interface keyword is used to declare an interface

Example

```
public interface  NameOfInterface
{
        //Any number of final , static fields
        // Any number of abstract method declarations.
}
```

Fields in the interface are implicitly final and static

Methods in the interface are implicitly public and abstract

gauravgambhir.cse@piet.co.in

Interface Properties

Interface have the following properties.

An interface is implicitly abstract, you don’t need to use the abstract keyword when declaring an interface.

Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.

Methods in an interface are implicitly public.

gauravgambhir.cse@piet.co.in

Interface Example

```
interface Animal
{
        public void eat();
        public void travel();
}
```

By default both the methods are abstract.

Implementing Interface

- When a class implements an interface , think of class as signing a contract agreeing to perform the specific behavior of the interface.
- If the class doesn't perform all the behavior of the interface , the class must declare itself as abstract.
- A class uses the implements keyword to implement an interface.
- The implements keyword appears in the class declaration following the extends portion of the declaration.

Example

```
public class MammalInt implements Animal{
   public void eat(){
      System.out.println("Mammal eats");
   }
   public void travel(){
      System.out.println("Mammal travels");
   }
   public int noOfLegs(){
      return 0;
   }
   public static void main(String args[]){
      MammalInt m = new MammalInt();
      m.eat();
      m.travel();
   }
}
```

gauravgambhir.cse@piet.co.in

Rules while implementing interface

- A class can implement more than one interface at a time
- A class can extend only one class , but implement many interfaces.
- An interface can extend another interface , similar to the way that a class can extend another class.

gauravgambhir.cse@piet.co.in

Extending interfaces

- An interface can extend another interface , similar to the way that a class can extend another class.
- The extend keyword is used to extend an interface and the child interface inherits the methods of the parent interface.

```
public interface Sports {
        public void setHomeTeam(String name);
        public void setVisitingTeam(String name);
}
```

gauravgambhir.cse@piet.co.in

Extending interfaces

```
public interface Football extends Sports {
        public void homeTeamScored(int points);
        public void visitingTeamScored(int points);
        public void endOfQuarter(int quarter);
}

public interface Hockey extends Sports {
        public void homeGoalScored();
        public void visitingGoalScored();
        public void endOfPeriod(int period);
        public void overtimePeriod(int ot);
}
```

gauravgambhir.cse@piet.co.in

Extending interfaces

- The Hockey interface has four methods , but it inherits two from sports, thus a class that implements Hockey needs to implement all six methods. Similarly a class that implements Football needs to define the three methods from Football and two methods from Sports

Extending Multiple Interfaces

- A Java class can only extend one parent class. Multiple inheritance is not allowed. Interfaces are not classes and an interface can extend more than one parent interface.

- For example if the Hockey interface extended both Sports and Event it would be declared as

```
public interface Hockey extends Sports, Event
{
}
```

Example I1.java

Abstract class vs. Interface

- When we talk about abstract classes we are defining characteristics of an object type, specifying **what an object is**
- In the case of an interface we define a capability and we bond to provide that capability, we are talking about establishing a contract about **what the object can do.**

- An abstract class can have shared state or functionality.
- A good abstract class will reduce the amount of code that has to be rewritten because it's functionality or state can be shared.
- An interface is only a promise to provide the state or functionality.
- The interface has no defined information to be shared

gauravgambhir.cse@piet.co.in

Abstract class vs. Interface

- Abstract classes allow you to create a blueprint, and allow you to additionally CONSTRUCT (implement) properties and methods you want ALL its descendants to possess.

- An interface on the other hand only allows you to declare that you want properties and/or methods with a given name to exist in all classes that implement it - but doesn't specify how you should implement it.

gauravgambhir.cse@piet.co.in

Abstract class vs. Interface

- When you derive an Abstract class, the relationship between the derived class and the base class is 'is a' relationship. e.g., a Dog is an Animal, a Sheep is an Animal which means that a Derived class is inheriting some properties from the base class.

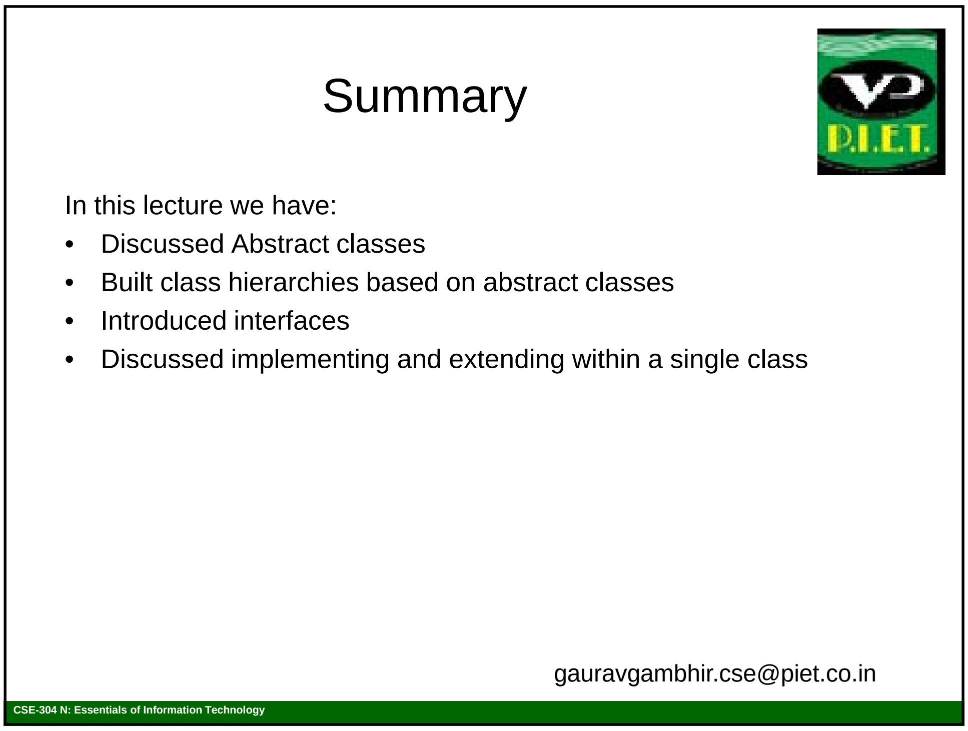
- Whereas for implementation of interfaces, the relationship is "can be". e.g., a Dog can be a spy dog. A dog can be a circus dog. A dog can be a race dog. Which means that you implement certain methods to acquire something.

gauravgambhir.cse@piet.co.in

Abstract class vs. Interface

- Example :Consider a base class called Customer which has abstract methods like CalculatePayment(), CalculateRewardPoints() and some non-abstract methods like GetName(), SavePaymentDetails().
- Specialized classes like RegularCustomer and GoldCustomer will inherit from the Customer base class and implement their own CalculatePayment() and CalculateRewardPoints() method logic, but re-use the GetName() and SavePaymentDetails() methods.
- You can add more functionality to an abstract class(non abstract methods that is) without affecting child classes which were using an older version. Whereas adding methods to an interface would affect all classes implementing it as they would now need to implement the newly added interface members.
- An abstract class with all abstract members would be similar to an interface.

gauravgambhir.cse@piet.co.in

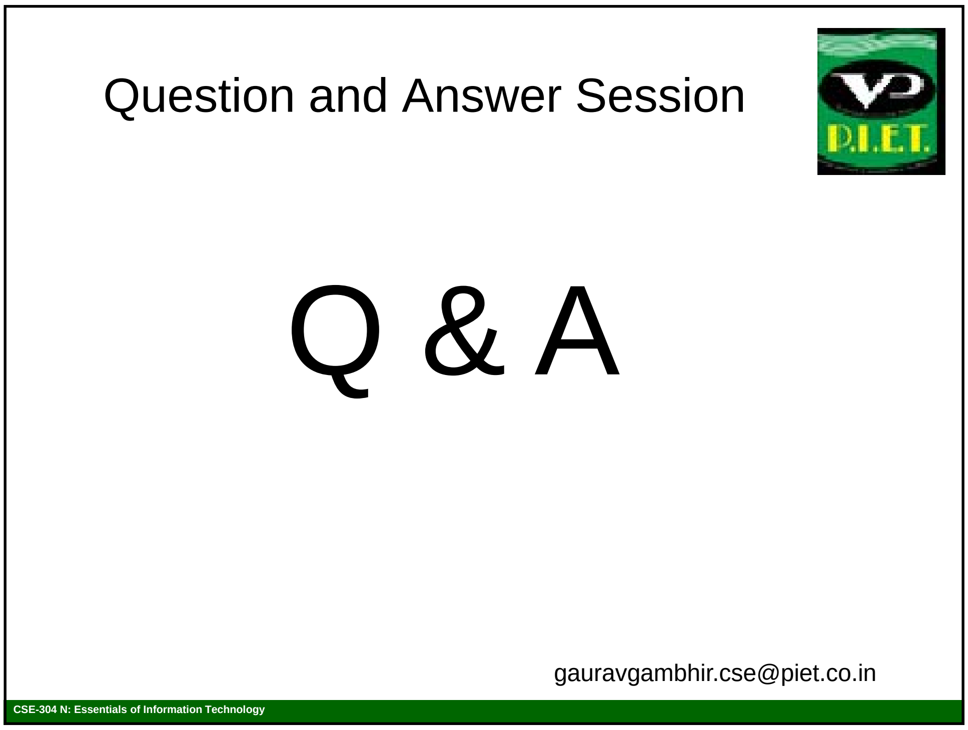


Summary

In this lecture we have:

- Discussed Abstract classes
- Built class hierarchies based on abstract classes
- Introduced interfaces
- Discussed implementing and extending within a single class

gauravgambhir.cse@piet.co.in



Question and Answer Session

Q & A

gauravgambhir.cse@piet.co.in