Essentials of Information Technology

PC-CS-305

Object Oriented Concepts

Objectives

In this lecture, we will

- Discuss the basic object oriented concepts
- Define classes, create and use instances of classes
- Examine the structure and syntax of a simple Java program using classes

gauravgambhir.cse@piet.co.in

Introduction to Classes

- The class is at the core of Java.
- It is the logical construct upon which the entire Java language is built because it defines the shape and nature of an object.
- The class forms the basis for object-oriented programming in Java.
- A class defines a new data type. Once defined, this new type can be used to create objects of that type.

gauravgambhir.cse@piet.co.in

Introduction to Classes

- A class is a *template for an object, and an object is an instance of a class.*
- In real world there are many object all of same kind, those objects are represented using class.
- **Class** : is a blue print from which individual objects are created.

  Class generally consist of field and methods.

  field : is the property of object which we create.

  method: is the operation that an object can perform.

gauravgambhir.cse@piet.co.in

Introduction to Objects and Classes

Example:

```
class Builder{
   String name;
   int age;
   void buildWall(){
    }
}
```

```
class BuilderDemo {
  public static void main(String args[])
  {
     Builder p1 = new Builder();
     Builder p2=new Builder();
      p1.name="Bob";  p2.name ="Sumit";
      p1.age=32;          p2.age=42;
   }
}
```

gauravgambhir.cse@piet.co.in

Introduction to Objects and Classes

- In above example from Builder class two real object are created p1 and p2.
- name and age are properties of object and value of these, vary with each object i.e. p1.name is different from p2.name.
- The data, or variables, defined within a **class are called *instance variables*** because each instance of the class (that is, each object of the class) contains its own copy of these variables. (e.g name , age)

gauravgambhir.cse@piet.co.in

Objects and Classes in Java

- however, you have used classes from the Java libraries to construct objects
- you used the `Scanner` class for input

```
Scanner kybd = new Scanner(System.in);
int hoursWorked = kybd.nextInt();
```

See BoxDemo3.java (example of instance variable & instance method)

gauravgambhir.cse@piet.co.in

Objects and Classes in Java

- Java provides the class `Scanner` that knows how to read input from an input stream
- the `Scanner` class defines methods that can be used to read the input

```
next()
nextInt()
nextDouble()
```

- these methods can be called on any `Scanner` object
  - `kybd.nextInt()` reads the next integer from the keyboard

gauravgambhir.cse@piet.co.in

Objects and Classes

This is analogous to the builder situation

- Builder is a class that knows how to do something
  - Build walls
- `Scanner` is a class that knows how to do something
  - get input from an input stream
- Bob(p1) is an instance of builder
  - We can ask Bob(p1) to build a wall
- `kybd` is an instance of `Scanner`
  - We can ask `kybd` to find the next integer typed in on the keyboard

gauravgambhir.cse@piet.co.in

Objects and Classes

- Using classes makes performing complex tasks simple
- In order to use a class you need to know:
  - Where the class is located
  - How to build an instance of the class – an object
  - What instances of the class can do
  - How to ask the instance to do something
- A class has a set of methods that define the functionality that it can provide
  - For example `nextInt` is a method of the `Scanner` class
- A method is invoked by sending a message to an object
  - `num1 = kybd.nextInt();`

gauravgambhir.cse@piet.co.in

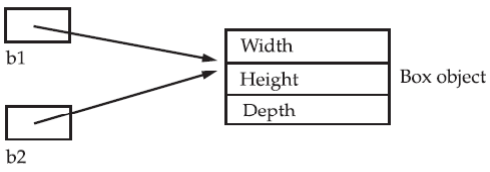
Reference Variables

- Object reference variables act differently than you might expect when an assignment takes place.
- Box b1 = new Box();
  Box b2 = b1;
- You might think that **b1 and b2 refer to separate and distinct objects. However,** this would be wrong. Instead, after this fragment executes, **b1 and b2 will both refer to the** *same object.*

gauravgambhir.cse@piet.co.in

Reference Variables

- It simply makes **b2 refer to the same object as does b1. Thus, any changes** made to the object through **b2 will affect the object to which b1 is referring, since they are the** same object.



- ***REMEMBER :*** *When you assign one object reference variable to another object reference variable**,** you are not creating a copy of the object, you are only making a copy of the reference.*

gauravgambhir.cse@piet.co.in

Pass By Value & Pass By Reference

- **Pass by Value** : Actual parameter expressions that are passed to a method are evaluated and a value is derived. Then this value is stored in a location and then it becomes the formal parameter to the invoked method. This mechanism is called pass by value.

- **Pass By Reference** : In pass by reference, the formal parameter is just an alias to the actual parameter. It refers to the actual argument. Any changes done to the formal argument will reflect in actual argument and vice versa.

gauravgambhir.cse@piet.co.in

Pass By Value & Pass By Reference

- Whenever parameters of methods are Java primitive type at that time value are passed by value .

- So value stored in number variable is copied to the num variable and num variable is stored in different memory location and is independent of other values

- So is we change the value of num in display method , the change is only visible in display method, it will not effect number variable.

gauravgambhir.cse@piet.co.in

Pass By Value & Pass By Reference

```
class PBV
{
    public static void main(String[] args)
    {
        int number=25;
        System.out.println("Before calling display method number =" + number);
        display(number);
        System.out.println("After calling display method number =" + number);
    }

    public static void display(int num)
    {
        System.out.println("Inside display method num =" + num);
        num=100;
        System.out.println("Inside display method num =" + num);
    }
}
```

```
---------- Java Run ----------
Before calling display method number =25
Inside display method num =25
Inside display method num =100
After calling display method number =25

Output completed (0 sec consumed) - Normal Termination
```

gauravgambhir.cse@piet.co.in

Pass By Value & Pass By Reference

```
class PBV
{
    public static void main(String[] args)
    {

        int[] value= {125,635};
        System.out.println("before method First element of array= "+value[0]);
        displayArray(value);
        System.out.println("after method  First element of array= "+value[0]);
    }


    public static void displayArray(int [] values){
        System.out.println("Inside method first element is  = "+values[0]);
        values[0]=100;
        System.out.println("Inside method first element is  = "+values[0]);
    }
}
```

```
---------- Java Run ----------
before method First element of array= 125
Inside method first element is  = 125
Inside method first element is  = 100
after method  First element of array= 100

Output completed (0 sec consumed) - Normal Termination
```

gauravgambhir.cse@piet.co.in

Pass By Value & Pass By Reference

- When an object is passed as argument, that object itself is not passed as argument to the invoked method.
- Internally that object's reference is passed as value and it becomes the formal parameter in the method.

gauravgambhir.cse@piet.co.in

Test Pass by Value vs Pass by Reference

- Let's run a simple swap test and check for pass by value vs. pass by reference.
- Let us pass two arguments and swap them inside the invoked method, then check if the actual arguments are swapped.
- If the actual arguments are affected then the mechanism used is pass by reference otherwise it is pass by value.

gauravgambhir.cse@piet.co.in

Test Pass by Value vs Pass by Reference

```
class Animal {
    String name;

    public Animal(String name) {
        this.name = name;
    }

    public String toString() {
        return name;
    }
}
```

gauravgambhir.cse@piet.co.in

Test Pass by Value vs Pass by Reference

```
public class Swap {

    public static void main(String args[]) {
        Animal a1 = new Animal("Lion");
        Animal a2 = new Animal("Crocodile");

        System.out.println("Before Swap:- a1:" + a1 + "; a2:" + a2);
        swap(a1, a2);
        System.out.println("After Swap:- a1:" + a1 + "; a2:" + a2);
    }

    public static void swap(Animal animal1, Animal animal2) {
        Animal temp = new Animal("");
        temp = animal1;
        animal1 = animal2;
        animal2 = temp;
    }

}
```

```
---------- Java Run ----------
Before Swap:- a1:Lion; a2:Crocodile
After Swap:- a1:Lion; a2:Crocodile

Output completed (0 sec consumed) - Normal Termination
```

gauravgambhir.cse@piet.co.in

Test Pass by Value vs Pass by Reference

- Same code will swap the objects in C++ since it uses pass by reference.

```
void swap(Type& arg1, Type& arg2) {
    Type temp = arg1;
    arg1 = arg2;
    arg2 = temp;
}
```

gauravgambhir.cse@piet.co.in

Test Pass by Value vs Pass by Reference

```
class Animal {
    String name;

    public Animal(String name) {
        this.name = name;
    }

    public String toString() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

gauravgambhir.cse@piet.co.in

Test Pass by Value vs Pass by Reference

```
public class Swap {

    public static void main(String args[]) {
        Animal a = new Animal("Lion");

        System.out.println("Before Modify: " + a);
        modify(a);
        System.out.println("After Modify: " + a);
    }

    public static void modify(Animal animal) {
        animal.setName("Tiger");
    }

}
```

```
---------- Java Run ----------
Before Modify: Lion
After Modify: Tiger

Output completed (0 sec consumed) - Normal Termination
```

gauravgambhir.cse@piet.co.in

Test Pass by Value vs Pass by Reference

- Java passes the object reference ‘by value’.
- When an object is passed as argument to a method, actually the reference to that object is passed.
- The formal parameter is a mapping of the reference to the actual parameter.

gauravgambhir.cse@piet.co.in

What is Object Oriented Programming?

- Object oriented Programming an approach to program Design and Implementation that focuses upon:
  - Organisation of data
  - Access of data
- In contrast traditional programming focuses upon the organisation of Algorithms
- The three main characteristics of OOP are:
  - Encapsulation
  - Inheritance
  - Polymorphism

gauravgambhir.cse@piet.co.in

Characteristics of O.O.P.

Encapsulation

It conceal the details for example to drive a car we need to know how to accelerate, brake and steer. Car doesn't have to tell about how it burns the fuel, generate electricity and rotate tires to driver when all he want is to drive to destination.

gauravgambhir.cse@piet.co.in

Characteristics of O.O.P.

- **Encapsulation**
  - Hide internal state of one object from another
  - Is a technique of making the fields in a class private and providing access to the fields via public methods.
  - If a field is declared private it cannot be accessed by anyone outside the class there by hiding the fields within the class.
  - Encapsulation is also known as data hiding.
- **CLASS** is the unit of encapsulation
- An **Object** is an instance of a class

Characteristics of O.O.P.

- Encapsulation Example
  - EncapsulationDemo.java , RunEncapsulation.java
- The user of the class don't know how the class store its data. A class can change the data type of a field and users of the class don't need to change any of their code.

Characteristics of O.O.P.

- Advantage of Encapsulation
  - Encapsulated Code is more flexible and easy to change with new requirements.
  - By providing only getter and setter method access, you can make the class read only.
  - Encapsulation in Java makes unit testing easy.
  - A class can have total control over what is stored in its fields. Suppose you want to set the value of marks field i.e. marks should be positive value, than you can write the logic of positive value in setter method.
  - Encapsulation allows you to change one part of code without affecting other part of code.

gauravgambhir.cse@piet.co.in

Characteristics of O.O.P.

Inheritance

- the ability to construct **DERIVED** classes from existing **BASE** classes
- the derived class inherits the member variables and member methods of the base class
- the derived class can
  - add additional member variables
  - add new member methods or override or extend the functionality of existing base class methods
  - **Example** Animal.java , Dog.java

gauravgambhir.cse@piet.co.in

Characteristics of O.O.P.

Polymorphism

Polymorphism is taking many different forms , its the ability of one object to be treated and used like another object. For example we treat duck as animal not just as duck similarly we treat dog and cat as animal. we ask each animal to speak, they speak in their own language

gauravgambhir.cse@piet.co.in

Characteristics of O.O.P.

Polymorphism

- one method can have a different function associated with it for each class in a hierarchy
- which function to call can be determined at run time based upon the class of the object against which the method is called
- **Example** Animal.java, Dog.java, Duck.java, Cat.java

gauravgambhir.cse@piet.co.in

Designing Object Oriented Programs

The general approach for designing and developing object oriented programs is:

- determine the set of classes required
- establish the essential instances of the classes
- specify the exchange of messages between instances

Create **CLASSES**

Create **INSTANCES** of classes (**OBJECTS**)

Specify the **MESSAGES** to be exchanged by objects

gauravgambhir.cse@piet.co.in

Classes

- A **CLASS** is a user defined type which encapsulates
  - the representation of the type
  - the operations which access or update objects of that type
  - operations for creating and deleting objects
- A class is a structure that can have functions as members as well as data elements
- The functions are called:
  - member functions
  - or member methods

gauravgambhir.cse@piet.co.in

Classes

- A class Person could be set up to hold and manage information about a person
- There are many common attributes that could be used to describe a person
  - name, age
  - sex, occupation
- These attributes are held as member variables of the class
- In this example we will use just two attributes
  - name and age

gauravgambhir.cse@piet.co.in

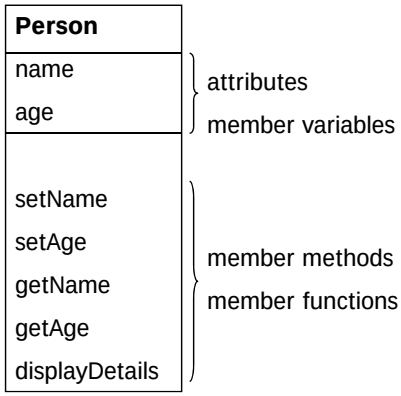
Classes

- There are many things that we might like to do with the information about a person
  - set the value of the name
  - set the value of the age
  - get the value of the name

  } accessing the attributes
  - get the value of the age
  - display details of the person
- These describe the functionality expected of an instance of the class
  - implemented by member functions

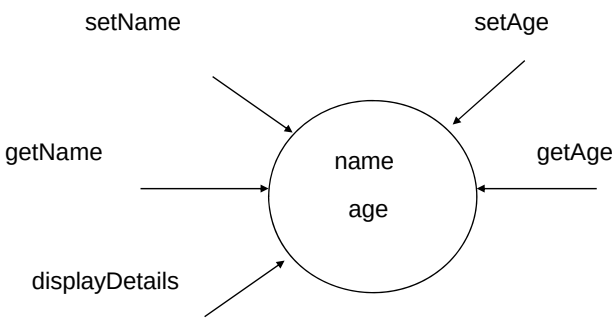
gauravgambhir.cse@piet.co.in

Person



gauravgambhir.cse@piet.co.in

Person



gauravgambhir.cse@piet.co.in

Classes in Java

```
public class Person {
    String name;
    int age;
    public void setName (String theName) {name = theName;}
     public void setAge (int theAge) {age = theAge;}
     public String getName () {return name;}
     public int getAge () {return age;}
     public void displayDetails ()
    {    System.out.println("Name is : " + name );
          System.out.println("Age is  : " + age );
    }
}
```

gauravgambhir.cse@piet.co.in

Creating an Object

- An object is an instance of a class
- The tutor is an example of a person
  - has a name
  - has an age

- In Java an object is constructed by instantiating a class:

```
Person theTutor = new Person( );
```

the class

a reference to the object

construct an object

an instance of the class

gauravgambhir.cse@piet.co.in

Using the Methods of an Object

- The member functions of a class can be called against an instance of the class:
  - theTutor.setName (“Pete”) ;
  - theTutor.setAge (53) ;
  - theTutor.displayDetails ( ) ;
- This is often described as sending a message to the object
  - asking the object to do something
  - the object itself knows how to do what is asked

The Heap and Garbage Collection

- All objects in Java are created on the heap
  - Space is allocated dynamically using the keyword new
- Consider the following code:

  Person thePerson;
- The variable thePerson can hold a reference to an object of type Person
  - Initially it is a null reference
- An object can be constructed on the heap using new

  thePerson = new Person ( );
- A constructor is invoked when new is used
  - Constructs the object

Testing a Class

- One way to test a class is to build a simple driver class
  - The driver contains a main
  - In the main method the class to be tested is instantiated
  - Then each of the methods of the class are invoked
- Why can the driver class method main be used without instantiating the driver class?
- In the driver main is:
  - public static void main (String [] args) …
- Methods that are static are at the class level and can be invoked without instantiating the class

Testing the class

```
public class TestPerson {
   public static void main(String []args)
   {
      System.out.println("Testing Person class");
      System.out.println("---------------------------------------");
       // instantiating the class
      // constructing an object of type Person
      Person   theTutor = new  Person();
       // using the set methods
      theTutor.setName ("Pete") ;
       theTutor.setAge (53) ;
```

Testing the class (continued)

```
        System.out.println("Testing displayDetails "+
                           " name Pete age 53 expected");
        theTutor.displayDetails () ;
        System.out.println("Testing getName - Pete expected");
        System.out.println(theTutor.getName( ));
        System.out.println("Testing getAge - 53 expected");
        System.out.println(theTutor.getAge( ));
    }
}
```


Summary

In this lecture we have:

- Discussed the module organisation
- Introduced why object-orientation is desirable
- Reviewed the basic object oriented concepts
- Defined classes, created and used instances of classes
- Examined the structure and syntax of a simple Java program using classes

Question and Answer Session

Q & A

gauravgambhir.cse@piet.co.in