Essentials of Information Technology

PC-CS-305

Inheritance

Objectives

In this chapter, we will

- Association, Composition, Aggregation
- Review inheritance
- Discuss the IS-A relationship
- Derive new classes from base classes

gauravgambhir.cse@piet.co.in

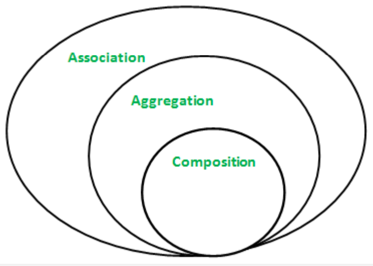
Association

- Association is relation between two separate classes which establishes through their Objects.
- Association can be one-to-one, one-to-many, many-to-one, many-to-many.
- In Object-Oriented programming, an Object communicates to other Object to use functionality and services provided by that object.
- **Composition** and **Aggregation** are the two forms of association.

gauravgambhir.cse@piet.co.in

Association

```
// Java program to illustrate the
// concept of Association
import java.io.*;

// class bank
class Bank
{
    private String name;

    // bank name
    Bank(String name)
    {
        this.name = name;
    }

    public String getBankName()
    {
        return this.name;
    }
}
```



gauravgambhir.cse@piet.co.in

Association

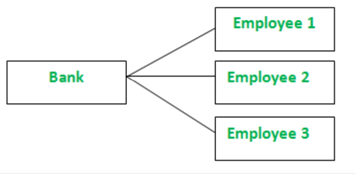
```
// employee class
class Employee
{
    private String name;

    // employee name
    Employee(String name)
    {
        this.name = name;
    }

    public String getEmployeeName()
    {
        return this.name;
    }
}
```

```
// Association between both the
// classes in main method
class Association
{
    public static void main (String[] args)
    {
        Bank bank = new Bank("Axis");
        Employee emp = new Employee("Neha");

        System.out.println(emp.getEmployeeName() +
               " is employee of " + bank.getBankName());
    }
}
```

gauravgambhir.cse@piet.co.in

Association

- In above example two separate classes Bank and Employee are associated through their Objects.
- Bank can have many employees, So it is a one-to-many relationship.



gauravgambhir.cse@piet.co.in

Aggregation

- It is a special form of Association where:
  - It represents Has-A relationship.
  - It is a unidirectional association i.e. a one way relationship. For example, department can have students but vice versa is not possible and thus unidirectional in nature.
  - In Aggregation, both the entries can survive individually which means ending one entity will not effect the other entity
- Example GFG.java

gauravgambhir.cse@piet.co.in

Aggregation

```
// Java program to illustrate
//the concept of Aggregation.
import java.io.*;
import java.util.*;

// student class
class Student
{
    String name;
    int id ;
    String dept;

    Student(String name, int id, String dept)
    {

        this.name = name;
        this.id = id;
        this.dept = dept;

    }
}
```

gauravgambhir.cse@piet.co.in

Aggregation

```
/* Department class contains list of student
Objects. It is associated with student
class through its Object(s). */
class Department
{

    String name;
    private List<Student> students;
    Department(String name, List<Student> students)
    {

        this.name = name;
        this.students = students;

    }

    public List<Student> getStudents()
    {
        return students;
    }
}
```

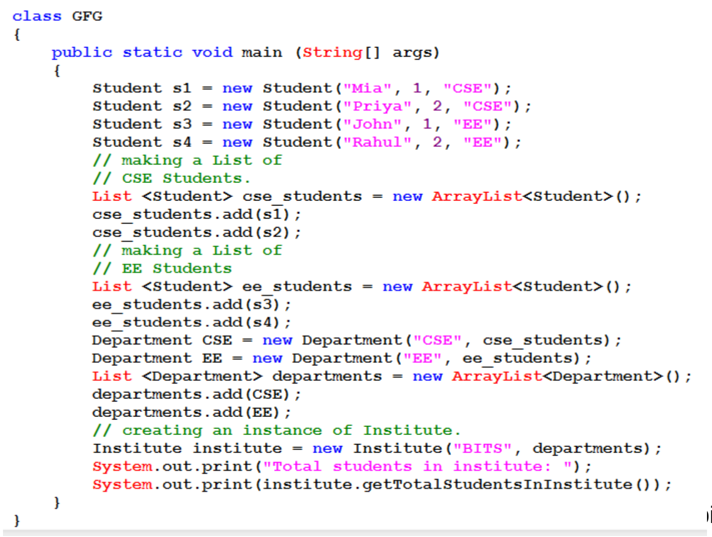
gauravgambhir.cse@piet.co.in

Aggregation

```
class Institute
{

    String instituteName;
    private List<Department> departments;

    Institute(String instituteName, List<Department> departments)
    {
        this.instituteName = instituteName;
        this.departments = departments;
    }

    // count total students of all departments
    // in a given institute
    public int getTotalStudentsInInstitute()
    {
        int noOfStudents = 0;
        List<Student> students;
        for(Department dept : departments)
        {
            students = dept.getStudents();
            for(Student s : students)
            {
                noOfStudents++;
            }
        }
        return noOfStudents;
    }

}
```

et.co.in

Aggregation



```
class GFG
{
    public static void main (String[] args)
    {
        Student s1 = new Student("Mia", 1, "CSE");
        Student s2 = new Student("Priya", 2, "CSE");
        Student s3 = new Student("John", 1, "EE");
        Student s4 = new Student("Rahul", 2, "EE");
        // making a List of
        // CSE Students.
        List <Student> cse_students = new ArrayList<Student>();
        cse_students.add(s1);
        cse_students.add(s2);
        // making a List of
        // EE Students
        List <Student> ee_students = new ArrayList<Student>();
        ee_students.add(s3);
        ee_students.add(s4);
        Department CSE = new Department("CSE", cse_students);
        Department EE = new Department("EE", ee_students);
        List <Department> departments = new ArrayList<Department>();
        departments.add(CSE);
        departments.add(EE);
        // creating an instance of Institute.
        Institute institute = new Institute("BITS", departments);
        System.out.print("Total students in institute: ");
        System.out.print(institute.getTotalStudentsInInstitute());
    }
}
```

iet.co.in

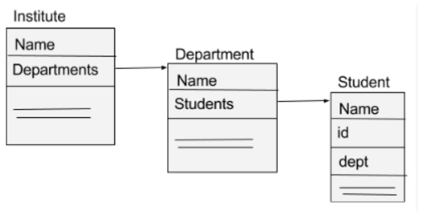
Aggregation

- In this example, there is an Institute which has no. of departments like CSE, EE. Every department has no. of students.
- So, we make a Institute class which has a reference to Object or no. of Objects (i.e. List of Objects) of the Department class.
- That means Institute class is associated with Department class through its Object(s). And Department class has also a reference to Object or Objects (i.e. List of Objects) of Student class means it is associated with Student class through its Object(s).

gauravgambhir.cse@piet.co.in

Aggregation

- It represents HAS A relationship



- **When do we use Aggregation ??**

  Code reuse is best achieved by aggregation.

gauravgambhir.cse@piet.co.in

Composition

- Composition is a restricted form of Aggregation in which two entities are highly dependent on each other.

- It represents part-of relationship.
- In composition, both the entities are dependent on each other.
- When there is a composition between two entities, the composed object cannot exist without the other entity.

- **Example** GFG1.java

gauravgambhir.cse@piet.co.in

Composition

- In above example a library can have no. of **books** on same or different subjects.
- So, If Library gets destroyed then All books within that particular library will be destroyed. i.e. book can not exist without library. That's why it is composition.

gauravgambhir.cse@piet.co.in

Aggregation vs Composition

- **Aggregation** implies a relationship where the child can exist independently of the parent.
- For example, Bank and Employee, delete the Bank and the Employee still exist.
- whereas **Composition** implies a relationship where the child cannot exist independent of the parent.
- Example: Human and heart, heart don't exist separate to a Human
- **Type of Relationship:** Aggregation relation is "has-a" and composition is "part-of" relation.
- **Type of association:** Composition is a strong Association whereas Aggregation is a weak Association.

gauravgambhir.cse@piet.co.in

Aggregation vs Composition

- Example GFG2.java
- In case of aggregation, the Car also performs its functions through an Engine. but the Engine is not always an internal part of the Car.
- An engine can be swapped out or even can be removed from the car. That' why we make The Engine type field non-final.

gauravgambhir.cse@piet.co.in

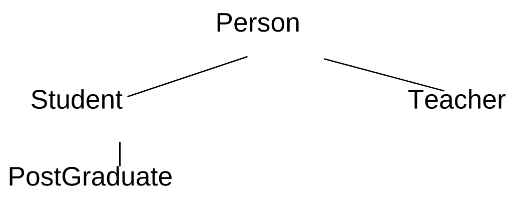
Inheritance

- Inheritance allows a new class to be defined by extending or modifying an existing class
- The new class is referred to as a derived class or sub class
- The term base class or super class refers to the class that is extended
- The derived class inherits public , protected & default members
  - both member variables and member methods
- Derived classes do not have access to any private members of the base class   **Access.java**
- Inheritance is a key concept in OOP
- **Example : SimpleInheritance.java**

gauravgambhir.cse@piet.co.in

A Simple Class Hierarchy

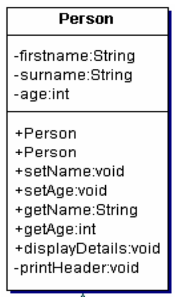
- In this section we will explore inheritance by investigating the following class hierarchy:



- A Student is a Person
- A PostGraduate is a Student and is a Person
- A Teacher is a Person

Person a Base Class

- Start with a Person
- This will be the base class of the hierarchy
  - provides useful common features



Person a Base Class

```
public class Person {
   private String firstname;
   private String surname;
   private int age;
   public Person (String theFirstname, String theSurname)
   {firstname = theFirstname; surname = theSurname; age = 0;}
    public Person (String theFirstname, String theSurname, int theAge)
   {firstname = theFirstname; surname = theSurname; age = theAge;}

   public void setName (String theFirstname, String theSurname)
   {firstname = theFirstname; surname = theSurname;}
    public void setAge (int theAge) {age = theAge;}
    public String getName () {return surname;}
    public int getAge () {return age;}
```

gauravgambhir.cse@piet.co.in

Person a Base Class (continued)

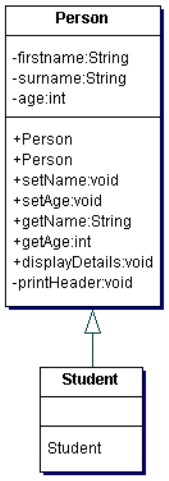
```
     public void displayDetails ()
    {
       printHeader();
      System.out.println("Name is : " + firstname + " " + surname );
       System.out.println("Age is  : " + age );
    }
    private void printHeader()
    {
      System.out.println("************************************");
      System.out.println("Person details");
      System.out.println("************************************");
    }
}
```

Example : Person.java

gauravgambhir.cse@piet.co.in

Student a Derived Class

- A Student is a Person
- The new class is publicly derived from the base class
  - all the public members of the base class are available to instances of the derived class

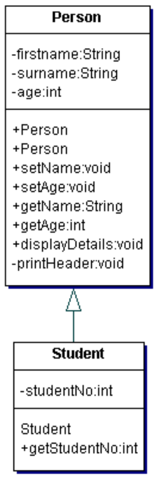


gauravgambhir.cse@piet.co.in

Student

- Student extends Person by adding:
  - one member variable studentNo
  - one member function getStudentNo()

- A Student is constructed by specifying a name, age and student number:
  - Student Alex = new Student("Alex", "Smith", 20, 199913);



gauravgambhir.cse@piet.co.in

Student Extends Person

```
public class Student extends Person {
    private int studentNo ;
    public Student (String theFirstname, String theSurname,
                        int theAge, int num)
    {
        super(theFirstname, theSurname, theAge);
        studentNo = num;
    }
    public int getStudentNo()
    {
        return studentNo;
    }
}
```

Example : Student.java

gauravgambhir.cse@piet.co.in

Extends

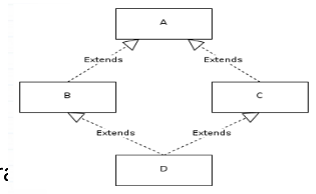
- In Java the keyword extends is used to indicate that one class is derived from another
- Only single inheritance is allowed in Java
  - That is the derived class can only extend one base class
  - Other languages allow multiple inheritance where a derived class inherits from two or more base classes

  - **Example :** Child.java

gauravgambhir.cse@piet.co.in

Diamond Problem of Multiple Inheritance

- We have two classes B and C inheriting from A, assume that B and C are overriding an inherited method and they provide their own implementation. Now D inherits from both B and C doing multiple inheritance. D should inherit that overridden method , which overridden method will be used , will it be from B or C , Here its an ambiguity.
- In Java this can never occur as there is no multiple inheritance. Even if two interface are going to have same method, the implementing class will have only one method and that too will be done by the implementor.
- **Example** A.java , B.java, C.java , D.java



gaura

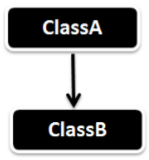
Types of Inheritance

- Single Inheritance
- Multiple Inheritance (Through Interface)
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance (Through Interface)

gauravgambhir.cse@piet.co.in

Single Inheritance

- Single Inheritance is the simple inheritance of all, When a class extends another class(Only one class) then we call it as Single inheritance.
- The below diagram represents the single inheritance in java where Class B extends only one class Class A.
- Here Class B will be the Sub class and Class A will be one and only Super class.



gauravgambhir.cse@piet.co.in

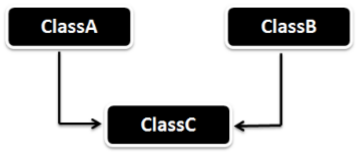
Single Inheritance

```
public class ClassA
{
    public void dispA()
    {
        System.out.println("disp() method of ClassA");
    }
}
public class ClassB extends ClassA
{
    public void dispB()
    {
        System.out.println("disp() method of ClassB");
    }
    public static void main(String args[])
    {
        //Assigning ClassB object to ClassB reference
        ClassB b = new ClassB();
        //call dispA() method of ClassA
        b.dispA();
        //call dispB() method of ClassB
        b.dispB();
    }
}
```

gauravgambhir.cse@piet.co.in

Multiple Inheritance

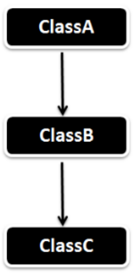
- Multiple Inheritance is nothing but one class extending more than one class.
- Multiple Inheritance is basically not supported in Java but you can achieve it using Interfaces.



gauravgambhir.cse@piet.co.in

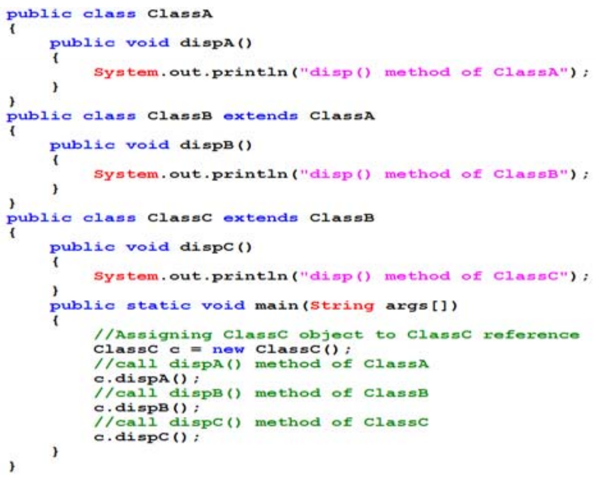
Multilevel Inheritance

- In Multilevel Inheritance a derived class will be inheriting a parent class and as well as the derived class act as the parent class to other class.
- As seen in the below diagram, Class B inherits the property of Class A and again Class B act as a parent for Class C.



gauravgambhir.cse@piet.co.in

Multilevel Inheritance

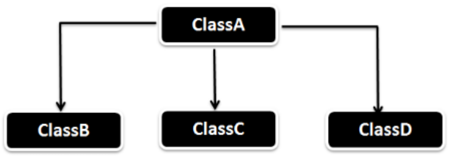


```
public class ClassA
{
    public void dispA()
    {
        System.out.println("disp() method of ClassA");
    }
}
public class ClassB extends ClassA
{
    public void dispB()
    {
        System.out.println("disp() method of ClassB");
    }
}
public class ClassC extends ClassB
{
    public void dispC()
    {
        System.out.println("disp() method of ClassC");
    }
    public static void main(String args[])
    {
        //Assigning ClassC object to ClassC reference
        ClassC c = new ClassC();
        //call dispA() method of ClassA
        c.dispA();
        //call dispB() method of ClassB
        c.dispB();
        //call dispC() method of ClassC
        c.dispC();
    }
}
```

gauravgambhir.cse@piet.co.in

Hierarchical Inheritance

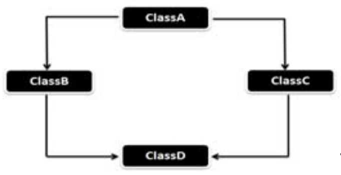
- In Hierarchical inheritance one parent class will be inherited by many sub classes.
- As per the below example Class A will be inherited by Class B, Class C and Class D. Class A will be acting as a parent class for Class B, Class C and Class D.
- Example : HierarchicalInheritanceTest.java



gauravgambhir.cse@piet.co.in

Hybrid Inheritance

- Hybrid Inheritance is the combination of both Single and Multiple Inheritance.
- Again Hybrid inheritance is also not directly supported in Java only through interface we can achieve this.
- Flow diagram of the Hybrid inheritance will look like below. As you can Class A will be acting as the Parent class for Class B & Class C and Class B & Class C will be acting as Parent for Class D.



ravgambhir.cse@piet.co.in

Super

- Student extends Person
  – Student is derived from Person
  – A student is a person plus a little bit more
- Note the use of super in the Student constructor
  – super refers to the immediate base class
  – invokes the super class constructor with the appropriate signature
  – a suitable constructor must exist in the base class otherwise the compilation will fail
- The derived class can have as many constructors as required but each must call a suitable base class constructor using super

gauravgambhir.cse@piet.co.in

Adding Functionality to Student

- The Student class can be extended by adding more member variables and more member methods
- Consider adding a method to get the full name of a student:

```
public String getFullName()
{
   return firstname + " " +  surname;
}
```

Student.java

- The method makes use of the base class member variables firstname and surname
- This FAILS; access to these base class variables is Private and therefore not available to derived classes

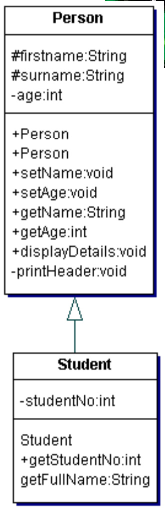
gauravgambhir.cse@piet.co.in

Person a Base Class

- Access to the base class member variables can be granted to the derived class using the keyword public or protected or not writing any access modifier:

```
public class Person {
   protected String firstname;
   protected String surname;
   private int age;
   public …
```

Example : Person.java , Student.java (uncomment getFullName() method in Student.java)



gauravgambhir.cse@piet.co.in

Using displayDetails

- Consider the following code:
  - Student Alex = new Student("Alex", “Smith”,20, 199913);
  - Alex.displayDetails ( );
- The member function displayDetails() is inherited from the base class Person
- What will be output?
- Example : Student.java

gauravgambhir.cse@piet.co.in

Overriding Base Class Methods

- It may be better to output the student number as well as the other details in response to displayDetails
- A member method of a derived class can have the same name and signature as a base class member method
- The derived class method overrides the base class method
- The base class method can still be accessed from the derived class using super.BaseClassMethod()
  - For example: super.displayDetails()

gauravgambhir.cse@piet.co.in

Overriding Base Class Methods

- The following method could be added to Student

```
public void displayDetails ()
{
     super.displayDetails();
    System.out.println("Student number is : " + studentNo );
}
```

- Note the use of super to ensure that the base class method displayDetails is called

gauravgambhir.cse@piet.co.in

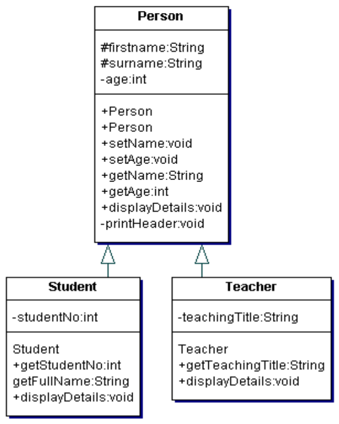
Using displayDetails

- Consider the following code now:
  - Student Alex = new Student("Alex", “Smith”, 20, 199913);
  - Alex.displayDetails ( );
- The member function displayDetails() from the base class Person is overridden in the derived class Student
- What will be output?
- Example : Student.java (uncomment displayDetails () method in student.java class)

gauravgambhir.cse@piet.co.in

Teacher another Derived Class

- A Teacher is a Person
- The new class is publicly derived from the base class
  - all the public members of the base class are available to instances of the derived class
  - **Example :** Teacher.java



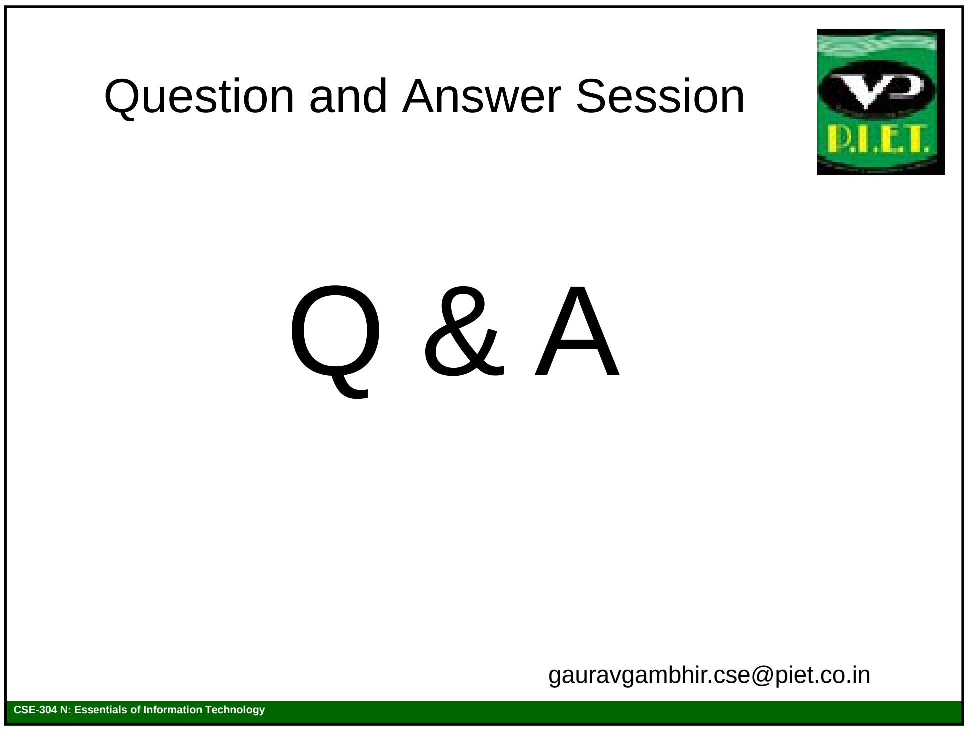
gauravgambhir.cse@piet.co.in

Summary

In this lecture we have:

- Reviewed inheritance
- Discussed the IS-A relationship
- Derived new classes from base classes

gauravgambhir.cse@piet.co.in



Question and Answer Session

Q & A

gauravgambhir.cse@piet.co.in

CSE-304 N: Essentials of Information Technology