VIT-AP UNIVERSITY

Course code : **CSE2005**

Course title : **Object Oriented Programming**

Inheritance

Objectives

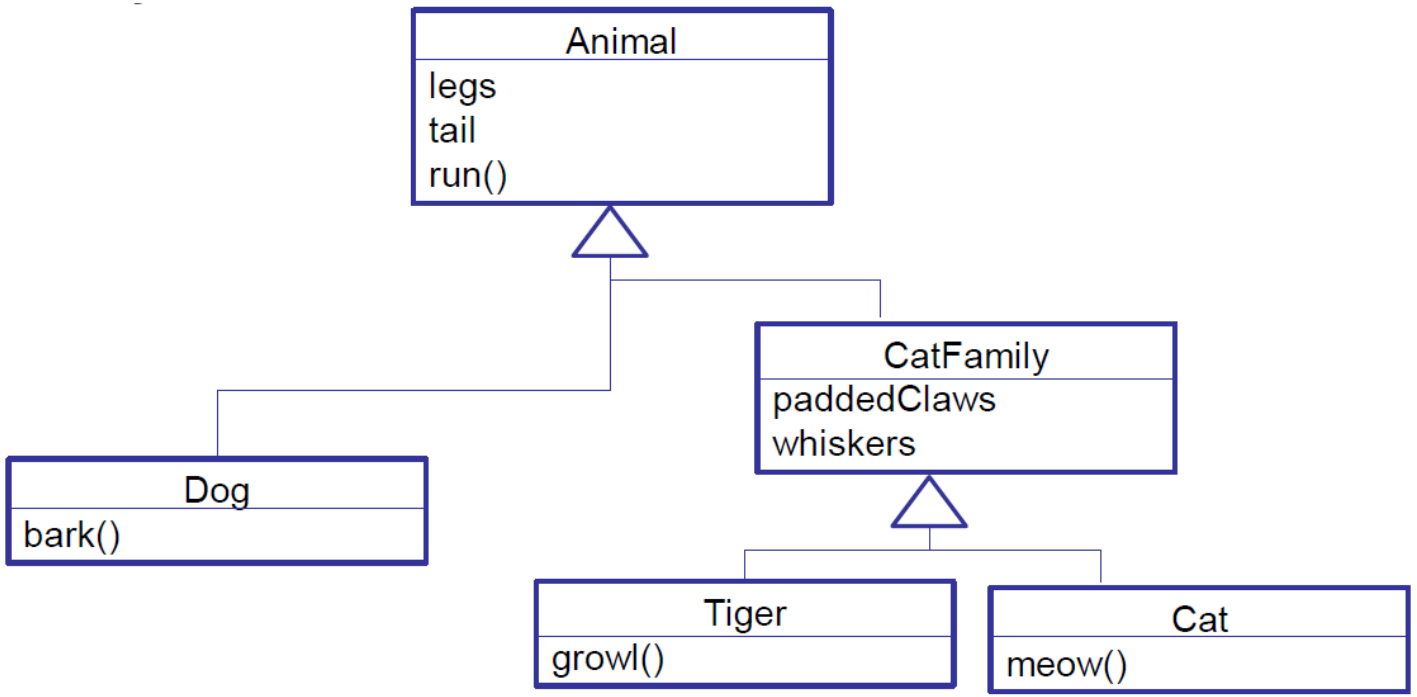
This session will give the knowledge about

- Describe Java's inheritance model and its language syntax
- Describe the usage of the keyword super
- Define a multilevel hierarchy

Inheritance

Inheritance defines relationship among classes, wherein one class share structure or behavior defined in one or more classes.

- Grady Booch



Inheritance

Using inheritance, you can create a general class at the top

This class may then be inherited by other, more specific classes

Each of these classes will add only those attributes and behaviors that are unique to it

Generalization/ Specialization

- In keeping with Java terminology, a class that is inherited is referred to as a superclass
- The class that does the inheriting is referred to as the subclass
- Each instance of a subclass includes all the members of the superclass
- The subclass inherits all the properties of its superclass
- Association, Aggregation, Composition are the terms used to signify the relationship between classes

Association

- Association is a relationship between two objects. The association between objects could be
  - one-to-one
  - one-to-many
  - many-to-one
  - many-to-many
- Types of Association
  - Aggregation
  - Composition
- ***Example:*** A Student and a Faculty are having an association

Aggregation

- Aggregation is a special case of association
- A directional association between objects
- When an object ‘has-a’ another object, then you have got an aggregation between them
- Aggregation is also called a “Has-a” relationship.
- Example: College has a Student Object

Composition

- Composition is a special case of aggregation
- In a more specific manner, a restricted aggregation is called composition
- When an object contains the other object, if the contained object cannot exist without the existence of container object, then it is called composition
- Example: A class contains students. A student cannot exist without a class. There exists composition between class and students

IS-A Relationship

- It can be implemented by inheriting the properties from the super class to derived class
- Derived class instance can access all the properties of base class
  - Employee is a Person
  - Manager is an Employee

HAS-A Relationship

- HAS-A relationship is expressed with containership
- Containership simply means using instance variables that refer to other objects
- Example:
  - The class House will have an instance variable which refers to a Kitchen object
  - It means that, House HAS-A Kitchen
  - Note that, something like Kitchen HAS-A House is not valid in this context

HAS-A Relationship

- Let us take one personal computer.
- It has a monitor, CPUbox, keyboard and mouse, etc.
- Technically we can say that,
  - Personal Computer class HAS-A monitor.
  - Personal Computer class HAS-A CPUbox
  - The most important point is : the 4 independent components like monitor, keyboard, CPUbox and mouse cannot function separately on its own.
  - But, by combining them, we are creating a new type of useful class called Personal Computer.

Java's Inheritance Model

Java uses the single inheritance model.

In single inheritance, a subclass can inherit from one (and only one) superclass

Code Syntax for Inheritance:

```
package vit.demo;
class Base{
}
class Derived extends Base {
 }
```

Single Inheritance

```
package vit.demo;

class Person{
        String name="myname";
        int age=21;
}

class Student extends Person {
        String roll="19bcd1234";
}
```

```
class Main{
        public static void main(String ar[]){
            Student student=new Student();

            System.out.println(student.name);
            System.out.println(student.age);
            System.out.println(student.roll);
        }
}
```

Guess the output

```
package vit.demo;
class Base{
    private int pktmoney;
    int money;
    void fill(int pktmoney,int money){
        pktmoney=pktmoney;
        money=money;
    }
}
class Derived extends Base{
    int total;
    void sum(){
        total=pktmoney+  money;
    }
}
```

```
class Main{
    public static void main(String ar[]){
        Derived derived=new Derived();
        derived.fill(200, 2000);
        derived.sum();
        System.out.println(derived.total);
    }
}
```

Using super

The creation and initialization of the superclass object is a prerequisite to the creation of the subclass object.

When a subclass object is created,

- It creates the superclass object
- Invokes the relevant superclass constructor.
    - The initialized superclass attributes are then inherited by the subclass object

Using super

- finally followed by the creation of the subclass object
  - initialization of its own attributes through a relevant constructor subclass
- The constructors of the superclass are never inherited by the subclass
- This is the only exception to the rule that a subclass inherits all the properties of its superclass

Example

```
package vit.demo;
class Person {
    String name = "myname";
    int age = 21;
    Person(){
            System.out.println(name+" "+age);
    }
}
class Student extends Person {
    String roll = "19bcd1234";
    Student(){
            System.out.println(roll);
    }
}
```

```
class Faculty extends Person {
    String id="vit101";
    Faculty(){
            System.out.println(id);
    }

}

class Main {
    public static void main(String ar[]) {
            Student student = new Student();
            Faculty faculty = new Faculty();
    }
}
```

Using super to Call Superclass Constructors

- super() if present, must always be the first statement executed inside a subclass constructor.
- It clearly tells you the order of invocation of constructor in a class hierarchy.
- Constructors are invoked in the order of their derivation

Example

```
package vit.demo;
class Person {
    String name = "myname";
    int age = 21;
    Person(String name,int age){
            this.name=name;
            this.age=age;
    }
}
class Student extends Person {
    String roll = "19bcd1234";
    Student(){
            super("sam",23);
            System.out.println(roll+" "+name+" "+age);
    }
}
```

```
class Faculty extends Person {
    String id="vit101";
    Faculty(){
            super("rahul",30);
            System.out.println(id+" "+name+" "+age);
    }

}
class Main {
    public static void main(String ar[]) {
            Student student = new Student();
            Faculty faculty = new Faculty();
    }
}
```

Example

```
package vit.demo;
class Person {
    String name = "myname";
    Person(String name){
            this.name=name;
    }
}
class Student extends Person {
    String roll = "19bcd1234";
    String name = "tom";
    Student(){
            super("sam");
            System.out.println(roll+" "+name);
    }
}
```

```
class Faculty extends Person {
    String id="vit101";
    Faculty(){
            super("rahul");
            System.out.println(id+" "+name);
    }

}
class Main {
    public static void main(String ar[]) {
            Student student = new Student();
            Faculty faculty = new Faculty();
    }
}
```

Defining a Multilevel Hierarchy

Java allows us to define multiple layers in an inheritance hierarchy

We can define a superclass and a subclass, with the subclass in turn becoming a superclass for another subclass

Consider the following example

- Employee
- Manager is a Employee
- Director is a Manager

Example

```
package vit.demo;
class Person {
   String name = "myname";
   Person(){
          System.out.println(name);
   }
}
class Employee extends Person {
   String id = "emp123";
   Employee(){
          System.out.println(id);
   }
}
```

```
class Manager extends Employee {
   int salary=100000;
   Manager(){
          System.out.println(salary);
   }
}

class Main {
   public static void main(String ar[]) {
          Manager manager=new Manager();
   }
}
```

Guess the output

```
package vit.demo;
class Person {
   String name = "myname";
   Person(){
         System.out.println(name);
   }
}
class Employee extends Person {
   String id = "emp123";
   Employee(){
         System.out.println(id);
   }
}
```

```
class Manager extends Employee {
   int salary=100000;
   Manager(){
         System.out.println(salary);
   }
}

class Main {
   public static void main(String ar[]) {
         Employee employe=new Employee();
   }
}
```

Guess the output

```
package vit.demo;

class Person {
    String name = "person";
}
class Employee extends Person {
    String name = "employee";
}
class Manager extends Employee {
    String name = "manager";
}
```

```
class Main {
    public static void main(String ar[]) {
        Manager obj = new Manager();
        if(obj instanceof Manager)
            System.out.println(obj.name+" is a manager");
        else if(obj instanceof Employee)
            System.out.println(obj.name+" is an employe");
    }
}
```

Guess the output

```
package vit.demo;
class A1{
    A1(){
            System.out.println("a1 default");
    }
    A1(int a){
            System.out.println("a1"+a);
    }
}
class B1 extends A1{
    B1(){
            System.out.println("b1 default");
    }
    B1(int a){
            super(1000);
            System.out.println("b1"+a);
    }
}
```

```
class C1 extends B1{
        C1(){
                System.out.println("c1 default");
        }
        C1(int a){
                super(100);
                System.out.println("c1"+a);
        }
}
class Main {
        public static void main(String ar[]) {
                C1 c=new C1();
        }
}
```

Guess the output

```
package vit.demo;
class A1{
    A1(){
            System.out.println("a1 default");
    }
    A1(int a){
            System.out.println("a1"+a);
    }
}
class B1 extends A1{
    B1(){
            System.out.println("b1 default");
    }
    B1(int a){
            super(1000);
            System.out.println("b1"+a);
    }
}
```

```
class C1 extends B1{
        C1(){
                System.out.println("c1 default");
        }
        C1(int a){
                super(100);
                System.out.println("c1"+a);
        }
}
class Main {
        public static void main(String ar[]) {
                C1 c=new C1(10);
        }
}
```

Guess the output

```
package vit.demo;
class A1{
    A1(){
            System.out.println("a1 default");
    }
    A1(int a){
            System.out.println("a1"+a);
    }
}
class B1 extends A1{
    B1(){
            System.out.println("b1 default");
    }
    B1(int a){
            super(1000);
            System.out.println("b1"+a);
    }
}
```

```
class C1 extends B1{
        C1(){
                System.out.println("c1 default");
        }
        C1(int a){
                System.out.println("c1"+a);
        }
}
class Main {
        public static void main(String ar[]) {
                C1 c=new C1(10);
        }
}
```

Guess the output

```
package vit.demo;
class A1{
    A1(){
            System.out.println("a1 default");
    }
    A1(int a){
            System.out.println("a1"+a);
    }
}
class B1 extends A1{
    B1(){
            System.out.println("b1 default");
    }
    B1(int a){
            System.out.println("b1"+a);
    }
}
```

```
class C1 extends B1{
        C1(){
                super(1000);
                System.out.println("c1 default");
        }
        C1(int a){
                System.out.println("c1"+a);
        }
}
class Main {
        public static void main(String ar[]) {
                C1 c=new C1(10);
        }
}
```

Summary

We have discussed about

- Describe Java’s inheritance model and its language syntax
- Describe the usage of the keyword super
- Define a multilevel hierarchy