Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Java : Inheritance/5b1bc62813ce7007ceda2369

TUTORIAL

Java : Inheritance

Topics

1.2 Inheritance Implementation

Hierarchy is a ranking or ordering of abstractions. The most important hierarchies in a complex system are its class structure (the "is a" hierarchy) and its object structure (the "part of' hierarchy).

Inheritance is the most important "is a" hierarchy, and as we noted earlier, it is an essential element of object systems. Basically, inheritance defines a relationship among classes, one class shares the structure or behavior defined in one or more classes (denoting single inheritance and multiple inheritance, respectively). Inheritance thus represents a hierarchy of abstractions, in which a subclass inherits from one or more superclasses. Typically, a subclass augments or redefines the existing structure and behavior of its superclasses. Semantically, inheritance denotes an "is-a" relationship. For example, a bear "is a" kind of mammal, a house "is a" kind of tangible asset, and quick sort "is a" searching algorithm. Lets understand inheritance with the help of an Employee and Manager class. Suppose you work for a company at which managers are treated differently from other employees. Managers are, of course, just like employees in many respects. Both employees and managers are paid a salary. However, while employees are expected to complete their assigned tasks in return for receiving their salary, managers get bonuses if they actually achieve what they are supposed to do. This is the kind of situation that cries out for inheritance. Why? Well, you need to define a new class, Manager, and add functionality. But you can retain some of what you have already programmed in the Employee class, and all the fields of the original class can be preserved. More abstractly, there is an obvious “is–a” relationship between Manager

and Employee. Every manager is an employee: This “is–a” relationship is the hallmark of inheritance. The general form of a class declaration that inherits a superclass is shown here:

```
class subclass-name extends superclass-name
{
        // body of class
}
```

You can only specify one superclass for any subclass that you create. Java does not support the inheritance of multiple superclasses into a single subclass. (This differs from C++, in which you can inherit multiple base classes.) You can, as stated, create a hierarchy of inheritance in which a subclass becomes a superclass of another subclass. However, no class can be a superclass of itself. Here is how you define a Manager class that inherits from the Employee class. You use the Java keyword extends to denote inheritance.

```
class Manager extends Employee
{
        // added methods and fields
}
```

The keyword extends indicates that you are making a new class that derives from an existing class. The existing class is called the superclass, base class, or parent class. The new class is called the subclass, derived class, or child class. The terms superclass and subclass are those most commonly used by Java programmers, although some programmers prefer the parent/child analogy, which also ties in nicely with the “inheritance” theme. Our Manager class has a new field to store the bonus, and a new method to set it:

```
class Manager extends Employee
{
        . . .
        public void setBonus(double b)
        {
        bonus = b;
        }
        private double bonus;
}
```

So, inheritance provides a way to enhance the previously defined entities for some more specialized tasks. One example of it is as shown below:

Inheritance Implementation

Java

```
// Create a superclass.
class A {
  int i, j;
  void showij() {
    System.out.println("i and j: " + i + " " + j);
  }
}

// Create a subclass by extending class A.
class B extends A {
  int k;
  void showk() {
    System.out.println("k: " + k);
  }
  void sum() {
    System.out.println("i+j+k: " + (i+j+k));
  }
}

class Main{
  public static void main(String args[]) {
    A superOb = new A();
    B subOb = new B();
    // The superclass may be used by itself.
    superOb.i = 10;
    superOb.j = 20;
    System.out.println("Contents of superOb: ");
    superOb.showij();

    System.out.println();
    /* The subclass has access to all public members of
its superclass. */
    subOb.i = 7;
    subOb.j = 8;
```

```
34      subOb.k = 9;
35      System.out.println("Contents of subOb: ");
36      subOb.showij();
37      subOb.showk();
38      System.out.println();
39      System.out.println("Sum of i, j and k in subOb:");
40      subOb.sum();
41    }
42  }
43
```

The output from this program is shown here:

```
Contents of superOb:
i and j: 10 20
Contents of subOb:
i and j: 7 8
k: 9
Sum of i, j and k in subOb:
i+j+k: 24
```

As you can see, the subclass B includes all of the members of its superclass, A. This is why subOb can access i and j and call showij( ). Also, inside sum( ), i and j can be referred to directly, as if they were part of B. Even though A is a superclass for B, it is also a completely independent, stand-alone class. Being a superclass for a subclass does not mean that the superclass cannot be used by itself. Further, a subclass can be a superclass for another subclass.