2023/2024(1)
EF234302 Object Oriented Programming

Lecture #9a

Access Control & Polymorphism

Misbakhul Munir IRFAN SUBAKTI
司馬伊凡
Мисбакхул Мунир Ирфан Субакти

Controlling access to members of a class

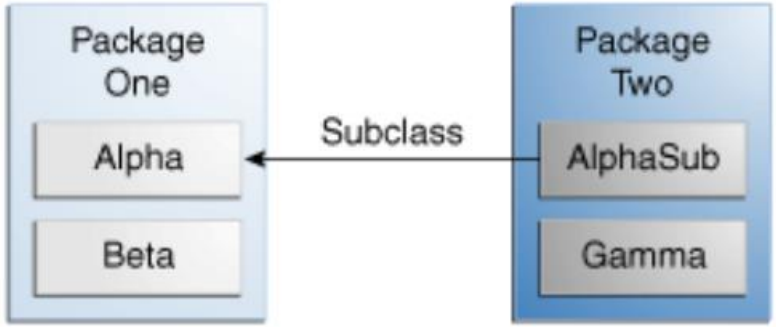
- Access level modifiers determine whether other classes can use a particular field or invoke a particular method
  - At the top level—`public`, or *package-private* (no explicit modifier, *default*).
  - At the member level—`public`, `private`, `protected`, or *package-private* (no explicit modifier, *default*).

Access Levels

Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

Access level: Illustration

- Assume we put some properties/`variables` & methods/`functions` in class `Alpha`



Visibility

Modifier	Alpha	Beta	Alphasub	Gamma
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

Access level: Tips

- If other programmers use your class, you want to ensure that errors from misuse cannot happen. Access levels can help you do this.
  - Use the most restrictive access level that makes sense for a particular member. Use `private` unless you have a good reason not to.
  - Avoid `public` fields *except* for constants. Public fields tend to link you to a *particular* implementation and *limit your flexibility* in changing your code.

Polymorphism

- Polymorphism means "*many forms*", and it occurs when we have many classes that are related to each other by *inheritance*.
- **Inheritance** lets us inherit attributes and methods from another class.
- **Polymorphism** uses those methods to perform different tasks. This allows us to perform a single action in different ways.
- For example, think of a superclass called `Animal` that has a method called `sound()`. Subclasses of Animals could be Cats, Cows, Dogs, etc.
- They also have their own implementation of an animal sound (the cat meows, the cow moos, etc.)

Animal: Superclass

Animal.java

```java
package animal;
public class Animal {
    // Information hiding = encapsulation
    // This class property set to be private -> cannot be accessed
    // directly. Only can be accessed by getter(s) & setter(s)
    private String name = "No name"; // Default name: No name
    private int age = 0; // Default age: 0
    private boolean vaccinated = false; // Default: false
    private double price = 10.0; // Default: 10.0

    // Overloading: a class has more than one method of
    // the same name & their parameter(s) are different

    // Constructor: special case of function, where its name
    // is the same with the class' name
    Animal() { // Overloading
        System.out.println("Animal: Default constructor");
    }
    Animal(String name) { // Overloading
        this.name = name;
        System.out.println("Constructor with name");
    }
    Animal(boolean vaccinated) { // Overloading
        this.vaccinated = vaccinated;
        System.out.println("Constructor with vaccinated status");
    }
    Animal(String name, int age) { // Overloading
        this.name = name;
        this.age = age;
        System.out.println("Constructor with name & age");
    }
    // Overriding: same method name & same parameter(s)
    // By using overriding, Run Time Polymorphism can be achieved
    public void sound() { // Polymorphism
        System.out.println("Animal makes a sound");
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public boolean isVaccinated() {
        return vaccinated;
    }
    public void setVaccinated(boolean vaccinated) {
        this.vaccinated = vaccinated;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public double getPrice() { // Overloading
        return price;
    }
    public double getPrice(String name) { // Overloading
        if (name.equals("No name")) {
            return price + 100;
        } else {
            return price + 110;
        }
    }
```

`Animal`: Superclass (continued)

```
65      public double getPrice(String name, int age) { // Overloading
66          if (name.equals("No name")) {
67              if (age > 1) {
68                  return price + 200;
69              } else {
70                  return price + 150;
71              }
72          } else {
73              if (age > 1) {
74                  return price + 300;
75              } else {
76                  return price + 200;
77              }
78          }
79      }
80      public double getPrice(boolean vaccinated) { // Overloading
81          if (vaccinated) {
82              return price + 400;
83          } else {
84              return price;
85          }
86      }
87      public void setPrice(double price) {
88          this.price= price;
89      }
90      @Override
91      public String toString() { // Used for printing an instance
92          return "[" + name + "]";
93      }
94  }
```

Cat, Cow, Dog: Subclasses

Cat.java

```java
package animal;
public class Cat extends Animal {
    // Overriding: same method name & same parameter(s)
    // By using overriding, Run Time Polymorphism can be achieved

    @Override
    public void sound() { // Polymorphism
        System.out.println("Cat says: meow");
    }

    // An example of Cat's method/function
    public void catMethod() {
        System.out.println("Cat: catMethod");
    }
}
```

Cow.java

```java
package animal;
public class Cow extends Animal {
    // Overriding: same method name & same parameter(s)
    // By using overriding, Run Time Polymorphism can be achieved

    @Override
    public void sound() { // Polymorphism
        System.out.println("Cow says: moo");
    }
}
```

Dog.java

```java
package animal;
public class Dog extends Animal {
    // Overriding: same method name & same parameter(s)
    // By using overriding, Run Time Polymorphism can be achieved

    @Override
    public void sound() { // Polymorphism
        System.out.println("Dog says: woof");
    }
}
```

AnimalTest: Overloading & overriding

AnimalTest.java

```java
package animal;
public class AnimalTest {
    public static void main(String[] args) {
        Animal animal = new Animal();  // Create an Animal object
        // Overloading
        Animal animal2 = new Animal("Bibi");  // With a name
        Animal animal3 = new Animal(true);   // With a vaccinated status
        Animal animal4 = new Animal("Chiki", 3);  // With a name & age
        System.out.println();

        Animal cat = new Cat();  // Create a Cat object
        Animal cow = new Cow();  // Create a Cow object
        Animal dog = new Dog();  // Create a Dog object
        System.out.println();

        // Polymorphism
        animal.sound();
        cat.sound();
        cow.sound();
        dog.sound();
        System.out.println();

        // Overloading
        System.out.println(animal);
        System.out.println("Age: " + animal.getAge());
        System.out.println("Price: " + animal.getPrice());
        System.out.println("Vaccinated: " + animal.isVaccinated());
        System.out.println();

        System.out.println(animal2);
        System.out.println("Age: " + animal2.getAge());
        System.out.println("Price: " + animal2.getPrice(animal2.getName()));
        System.out.println("Vaccinated: " + animal2.isVaccinated());
        System.out.println();

        System.out.println(animal3);
        System.out.println("Age: " + animal3.getAge());
        System.out.println("Price: " + animal3.getPrice(animal3.isVaccinated()));
        System.out.println("Vaccinated: " + animal3.isVaccinated());
        System.out.println();

        System.out.println(animal4);
        System.out.println("Age: " + animal4.getAge());
        System.out.println("Price: " + animal4.getPrice(animal4.getName(),
                animal4.getAge()));
        System.out.println("Vaccinated: " + animal4.isVaccinated());
        System.out.println();

        System.out.print("Showing that a superclass can call ");
        System.out.println("a subclass method");
        System.out.println("-----------------------------------------------------");
        System.out.println("a1 is defined by superclass: Animal, instantiated by superclass: Animal");
        Animal a1 = new Animal();
        System.out.println("b1 is defined by subclass: Cat, instantiated by subclass: Cat");
        Cat b1 = new Cat();
        System.out.println("b1 calls the Cat's method: catMethod()");
        b1.catMethod();
//      System.out.println("Try: a1 calls the subclass' method: catMethod() by casting to Cat");
//      ((Cat) a1).catMethod(); // Exception will happen
        System.out.println("c1 is defined by superclass: Animal, instantiated by subclass: Cat");
        Animal c1 = new Cat();
//      c1.catMethod(); // The subclass' method: catMethod() is undefined for the type Animal
        ((Cat) c1).catMethod(); // Need to cast, so that an instance of superclass can call
                                // the subclass' method
    }
}
```

`AnimalTest`: The result

```
Animal: Default constructor
Constructor with name
Constructor with vaccinated status
Constructor with name & age

Animal: Default constructor
Animal: Default constructor
Animal: Default constructor

Animal makes a sound
Cat says: meow
Cow says: moo
Dog says: woof

[No name]
Age: 0
Price: 10.0
Vaccinated: false

[Bibi]
Age: 0
Price: 120.0
Vaccinated: false

[No name]
Age: 0
Price: 410.0
Vaccinated: true

[Chiki]
Age: 3
Price: 310.0
Vaccinated: false
```

```
Showing that a superclass can call a subclass method
-----------------------------------------------------
a1 is defined by superclass: Animal, instantiated by superclass: Animal
Animal: Default constructor
b1 is defined by subclass: Cat, instantiated by subclass: Cat
Animal: Default constructor
b1 calls the Cat's method: catMethod()
Cat: catMethod
c1 is defined by superclass: Animal, instantiated by subclass: Cat
Animal: Default constructor
Cat: catMethod
```