

2021/2022(1)

IF184301 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

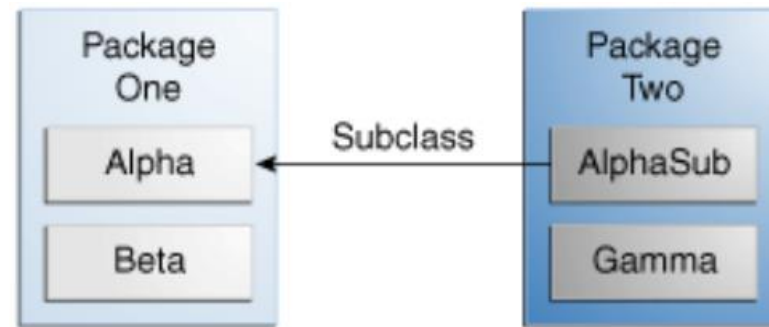
- Access level modifiers determine whether other classes can use a particular field or invoke a particular method
 - At the top level—`public`, or *package-private* (no explicit modifier, *default*).
 - At the member level—`public`, `private`, `protected`, or *package-private* (no explicit modifier,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

Access Levels

Modifier	Class	Package	Subclass	World
<code>public</code>	Y	Y	Y	Y
<code>protected</code>	Y	Y	Y	N
<i>no modifier</i>	Y	Y	N	N
<code>private</code>	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
<i>no modifier</i>	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 x
1 package animal;
2 public class Animal {
3     // Information hiding = encapsulation
4     // This class property set to be private -> cannot be accessed
5     // directly. Only can be accessed by getter(s) & setter(s)
6     private String name = "No name"; // Default name: No name
7     private int age = 0; // Default age: 0
8     private boolean vaccinated = false; // Default: false
9     private double price = 10.0; // Default: 10.0
10
11     // Overloading: a class has more than one method of
12     // the same name & their parameter(s) are different
13
14     // Constructor: special case of function, where its name
15     // is the same with the class' name
16     Animal() { // Overloading
17         System.out.println("Animal: Default constructor");
18     }
19     Animal(String name) { // Overloading
20         this.name = name;
21         System.out.println("Constructor with name");
22     }
23     Animal(boolean vaccinated) { // Overloading
24         this.vaccinated = vaccinated;
25         System.out.println("Constructor with vaccinated status");
26     }
27     Animal(String name, int age) { // Overloading
28         this.name = name;
29         this.age = age;
30         System.out.println("Constructor with name & age");
31
32
33     // Overriding: same method name & same parameter(s)
34     // By using overriding, Run Time Polymorphism can be achieved
35     public void sound() { // Polymorphism
36         System.out.println("Animal makes a sound");
37     }
38     public String getName() {
39         return name;
40     }
41     public void setName(String name) {
42         this.name = name;
43     }
44     public boolean isVaccinated() {
45         return vaccinated;
46     }
47     public void setVaccinated(boolean vaccinated) {
48         this.vaccinated = vaccinated;
49     }
50     public int getAge() {
51         return age;
52     }
53     public void setAge(int age) {
54         this.age = age;
55     }
56     public double getPrice() { // Overloading
57         return price;
58     }
59     public double getPrice(String name) { // Overloading
60         if (name.equals("No name")) {
61             return price + 100;
62         } else {
63             return price + 110;
64         }
65     }
66 }
```

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 ×
1 package animal;
2 public class Cat extends Animal {
3     // Overriding: same method name & same parameter(s)
4     // By using overriding, Run Time Polymorphism can be achieved
5
6     @Override
7     public void sound() { // Polymorphism
8         System.out.println("Cat says: meow");
9     }
10
11     // An example of Cat's method/function
12     public void catMethod() {
13         System.out.println("Cat: catMethod");
14     }
15 }
```

```
Cow.java ×
1 package animal;
2 public class Cow extends Animal {
3     // Overriding: same method name & same parameter(s)
4     // By using overriding, Run Time Polymorphism can be achieved
5
6     @Override
7     public void sound() { // Polymorphism
8         System.out.println("Cow says: moo");
9     }
10 }
```

```
Dog.java ×
1 package animal;
2 public class Dog extends Animal {
3     // Overriding: same method name & same parameter(s)
4     // By using overriding, Run Time Polymorphism can be achieved
5
6     @Override
7     public void sound() { // Polymorphism
8         System.out.println("Dog says: woof");
9     }
10 }
```


AnimalTest: Overloading & overriding

```
AnimalTest.java x 29
1 package animal;
2 public class AnimalTest {
3     public static void main(String[] args) {
4         Animal animal = new Animal(); // Create an Animal object
5         // Overloading
6         Animal animal2 = new Animal("Bibi"); // With a name
7         Animal animal3 = new Animal(true); // With a vaccinated status
8         Animal animal4 = new Animal("Chiki", 3); // With a name & age
9         System.out.println();
10
11        Animal cat = new Cat(); // Create a Cat object
12        Animal cow = new Cow(); // Create a Cow object
13        Animal dog = new Dog(); // Create a Dog object
14        System.out.println();
15
16        // Polymorphism
17        animal.sound();
18        cat.sound();
19        cow.sound();
20        dog.sound();
21        System.out.println();
22
23        // Overloading
24        System.out.println(animal);
25        System.out.println("Age: " + animal.getAge());
26        System.out.println("Price: " + animal.getPrice());
27        System.out.println("Vaccinated: " + animal.isVaccinated());
28        System.out.println();
29
30        System.out.println(animal2);
31        System.out.println("Age: " + animal2.getAge());
32        System.out.println("Price: " + animal2.getPrice(animal2.getName()));
33        System.out.println("Vaccinated: " + animal2.isVaccinated());
34        System.out.println();
35
36        System.out.println(animal3);
37        System.out.println("Age: " + animal3.getAge());
38        System.out.println("Price: " + animal3.getPrice(animal3.isVaccinated()));
39        System.out.println("Vaccinated: " + animal3.isVaccinated());
40        System.out.println();
41
42        System.out.println(animal4);
43        System.out.println("Age: " + animal4.getAge());
44        System.out.println("Price: " + animal4.getPrice(animal3.getName(),
45            animal3.getAge()));
46        System.out.println("Vaccinated: " + animal4.isVaccinated());
47        System.out.println();
48
49        System.out.print("Showing that a superclass can call ");
50        System.out.println("a subclass method");
51        System.out.println("-----");
52        System.out.println("a1 is defined by superclass: Animal, instantiated by superclass: Animal");
53        Animal a1 = new Animal();
54        System.out.println("b1 is defined by subclass: Cat, instantiated by subclass: Cat");
55        Cat b1 = new Cat();
56        System.out.println("b1 calls the Cat's method: catMethod()");
57        b1.catMethod();
58        // System.out.println("Try: a1 calls the subclass' method: catMethod() by casting to Cat");
59        // ((Cat) a1).catMethod(); // Exception will happen
60        System.out.println("c1 is defined by superclass: Animal, instantiated by subclass: Cat");
61        Animal c1 = new Cat();
62        // c1.catMethod(); // The subclass' method: catMethod() is undefined for the type Animal
63        ((Cat) c1).catMethod(); // Need to cast, so that an instance of superclass can call
64        // the subclass' method
65    }
66 }
```

AnimalTest: The result

```
Problems @ Javadoc Declaration Search Console X
<terminated> AnimalTest (1) [Java Application] D:\Program\Java\JDK\bin\je
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: 160.0
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
```