

Java OOP: Inheritance and Encapsulation

Press Space for next page →



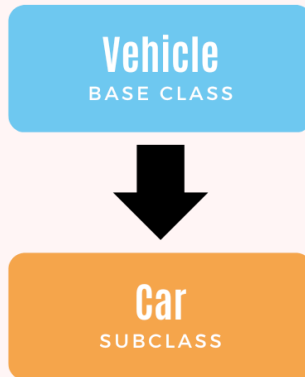
Agenda

1. Inheritance
2. Types of Inheritance
3. Code Example of Single Inheritance
4. Encapsulation
5. Getter and Setter Methods
6. Super Keyword
7. Java static keyword
8. Java static method
9. Final Keyword

Inheritance

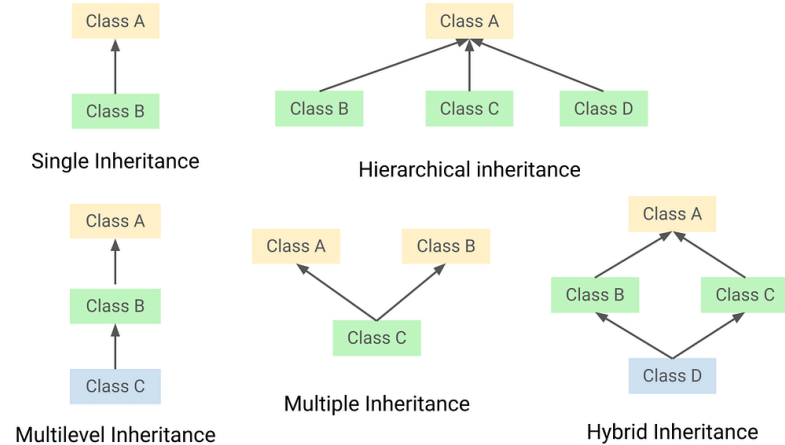
Inheritance in Java is a mechanism in which **one object acquires all the properties and behaviors of a parent object**. It is an important part of OOPs.

- When you inherit from an existing class, you can **reuse methods and fields of the parent class**.
- Moreover, you can **add new methods and fields in your current class** also.
- In the terminology of Java, a class which is inherited is called a **parent or superclass**, and the **new class is called child or subclass**.
- Inheritance represents the **IS-A relationship**.



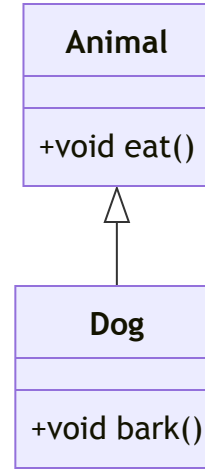
Types of Inheritance

1. **Single Inheritance:** A class inherits from only one class.
2. **Multilevel Inheritance:** A class inherits from a class which is already inherited from another class.
3. **Hierarchical Inheritance:** Multiple classes inherit from a single class.
4. **Hybrid Inheritance:** Combination of two or more types of inheritance.
5. **Multiple Inheritance:** A class inherits from more than one class.



Code Example of Single Inheritance

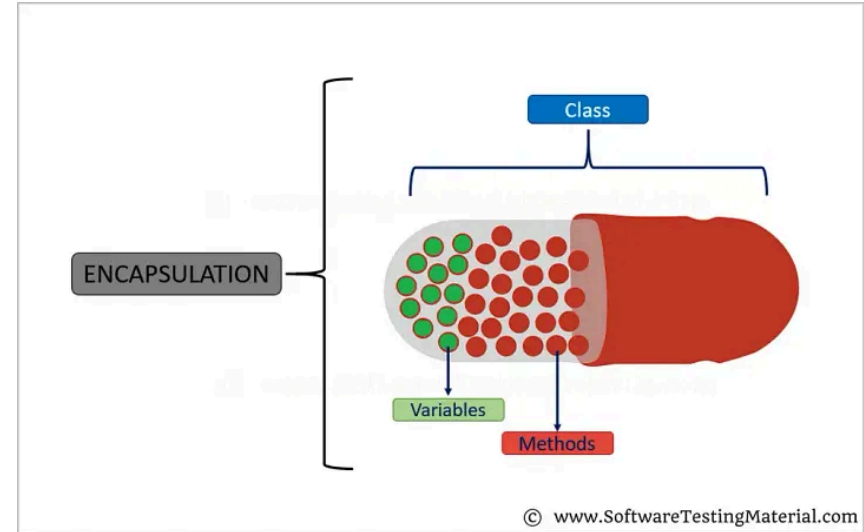
```
1  class Animal {
2      void eat() {
3          System.out.println("eating...");
4      }
5  }
6
7  class Dog extends Animal {
8      void bark() {
9          System.out.println("barking...");
10     }
11 }
12
13 class TestInheritance {
14     public static void main(String[] args) {
15         Dog dog = new Dog();
16         dog.bark();
17         dog.eat();
18     }
19 }
```



Encapsulation

Encapsulation in Java is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.

- We can create a fully encapsulated class in Java by making all the data members of the class private.
- Now we can use setter and getter methods to set and get the data in it.



Getter and Setter Methods

- Getter and setter methods are used to **access and modify the private fields of a class**.
- Usually, **class fields** are decorated with a **private access specifier**.
- Thus, to access them, **public access specifiers** are used with the getter and setter methods.

```
1  class Student {  
2      private String name;  
3  
4      public String getName() {  
5          return name;  
6      }  
7  
8      public void setName(String name) {  
9          this.name = name;  
10     }  
11 }
```

Super Keyword

- super can be used to **refer immediate parent class instance variable**.
- super can be used to **invoke immediate parent class method**.
- super() can be used to **invoke immediate parent class constructor**.

```
1  class Animal {
2      void eat() {
3          System.out.println("Animal is eating...");
4      }
5  }
6
7  class Dog extends Animal {
8      void eat() {
9          super.eat(); // Calls the parent class method
10         System.out.println("Dog is eating...");
11     }
12 }
13
14 public class TestSuper {
15     public static void main(String[] args) {
16         Dog dog = new Dog();
17         dog.eat();
18     }
19 }
```


Java static keyword

- The static keyword in Java is used for **memory management** mainly.
- We can apply static keyword with **variables, methods, blocks and nested classes**.
- The static keyword **belongs to the class** than an instance of the class.
- The static variable can be used to refer to the common property of all objects (**which is not unique for each object**), for example, the company name of employees, college name of students, etc.

```
1  class Student {
2      int studentId; // instance variable
3      String studentName;
4      static String collegeName = "XYZ"; // static variable
5
6      // constructor
7      Student(int studentId, String studentName) {
8          this.studentId = studentId;
9          this.studentName = studentName;
10     }
11
12     // method to display the values
13     void display() {
14         System.out.println(studentId + " " + studentName);
15     }
16
17     public static void main(String args[]) {
18         Student s1 = new Student(111, "Karan");
19         Student s2 = new Student(222, "Aryan");
20
21         s1.display();
22         s2.display();
23     }
24 }
```

Java static method

If you apply static keyword with any method, it is known as static method.

- A static method **belongs to the class** rather than the object of the class.
- A static method can be **invoked without the need for creating an instance of a class.**
- A static method can access **static data member** and can change the value of it.

```
1  class Student {
2      int studentId;
3      String studentName;
4      static String collegeName = "XYZ";
5      // constructor
6      Student(int studentId, String studentName) {
7          this.studentId = studentId;
8          this.studentName = studentName;
9      }
10     // static method to change the value of static v
11     static void change() {
12         collegeName = "ABC";
13     }
14     // method to display the values
15     void display() {
16         System.out.println(studentId + " " + student
17     }
18
19     public static void main(String args[]) {
20         Student.change(); // calling change method
21         Student s1 = new Student(111, "Karan");
22         Student s2 = new Student(222, "Aryan");
23
24         s1.display();
25         s2.display();
```


Final Keyword

The final keyword in java is used to **restrict the user**. The java final keyword can be used in many context. Final can be:

- **Variable:** Stops value change.
- **Method:** Stops method overriding.
- **Class:** Stops inheritance.

```
1  class Bike {
2      final int speedlimit = 90; // final variable
3      void run() {
4          speedlimit = 400; // Compile Time Error
5      }
6  }
7
8  class Main {
9      public static void main(String args[]) {
10         Bike obj = new Bike();
11         obj.run();
12     }
13 }
```

Thank you 

 [qa-june-2024-automation-with-java-slides](#)