06_1 Inheritance

Object-Oriented Programming

Inheritance

- Reduce duplicate code by reusing already well-developed classes to create new ones

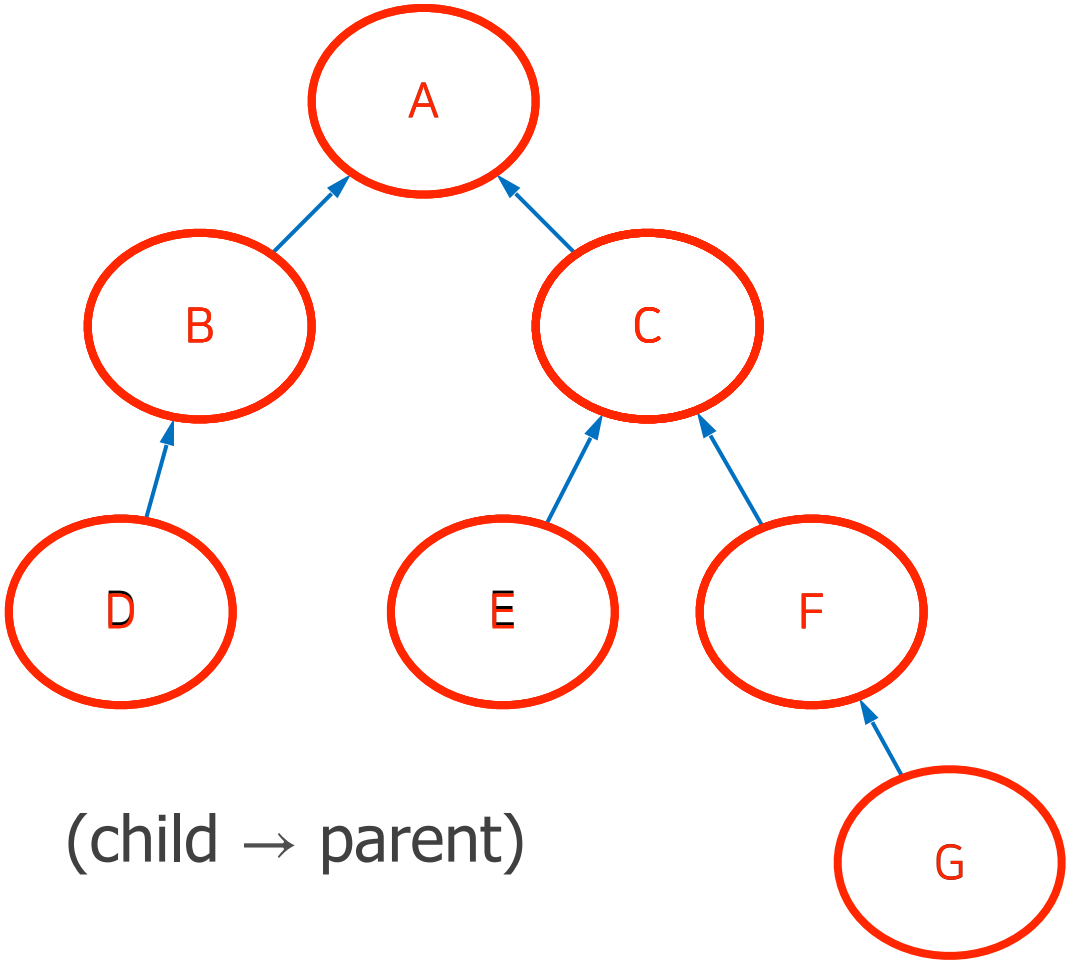
```
// A: parent class (= base class = super class)
public class A {
    int field1;              // A has field1
    void method1() { }       // A has method1()
}

// B: child class (= derived class = sub class)
public class B extends A { // B inherits from A
    int field2;              // B has field1 and field2
    void method2() { }       // B has method1() and method2()
}
```

Inheritance Rule

1. Only one parent class is allowed
2. Private fields and methods in the parent class cannot be accessed directly by the child class.
3. If the parent class exists in a different package, fields and methods with default (package) access cannot be directly accessed from the child class.

Class Hierarchy



(child → parent)

- A is a parent of B, C
- A is a grand parent of D, E, F
- A is an ancestor of B, C, D, E, F, G

- G is a child of F
- G is a grand child of C
- G is a descendant of A, C, F

- B is not an ancestor of E
- D is not a descendant of C

Example: AnimalTest (1/7)

```
public class Animal {
    private String name;  // private access
    private int age;      // private access

    // Default constructor
    public Animal() { }

    // Constructor with parameters
    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Accessors and Mutators
    public String getName() {
        return name;
    }
```

Example: AnimalTest (2/7)

```
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    // Method makesound()
    public void makeSound() {
        System.out.println("Some generic animal sound");
    }
    @Override // why? see later chapters
    public String toString() {
        return "Animal{name='" + name + "', age=" + age + '}';
    }
}
```

Example: AnimalTest (3/7)

```
public class Cat extends Animal { // inherit Animal class
    private String color;  // more instance variable

    public Cat() {
        super();    // must call the parent constructor first
    }

    // name and age are private in the parent class, so cannot directly accessed
    public Cat(String name, int age, String color) {
        super(name, age); // call the parent constructor instead of direct access
        this.color = color;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }
```

Example: AnimalTest (4/7)

```
    // Method overriding: change the original method
    @Override
    public void makeSound() {
        System.out.println("Meow");
    }

    // toString method
    @Override
    public String toString() {
        return "Cat{name='"+getName()+ "', age=" + getAge() + ", color='" + color + "'}";
    }
}

public class Dog extends Animal {
    private String breed; // more instance variable

    // Default constructor
    public Dog() {
        super();  // call the parent constructor
    }
```

Example: AnimalTest (5/7)

```
    // Constructor with parameters
    public Dog(String name, int age, String breed) {
        super(name, age);
        this.breed = breed;
    }

    // Accessors and Mutators
    public String getBreed() {
        return breed;
    }

    public void setBreed(String breed) {
        this.breed = breed;
    }

    // Method overriding
    @Override
    public void makeSound() {
        System.out.println("Bark");
    }
```

Example: AnimalTest (6/7)

```
    public String toString() {
        return "Dog{name='"+getName()+"', age=" + getAge() + ", breed='" + breed + "'}";
    }
}

public class AnimalTest {
    public static void main(String[] args) {
        Animal animal = new Animal("Generic Animal", 5);  // Animal object
        System.out.println(animal);
        animal.makeSound();  // print “Some generic animal sound”

        Dog dog = new Dog("Buddy", 3, "Golden Retriever");  // Dog object
        System.out.println(dog);
        dog.makeSound();     // print “Bark”

        Cat cat = new Cat("Whiskers", 2, "Black");  // Cat object
        System.out.println(cat);
        cat.makeSound();     // print “Meow”
    }
}
```

Example: AnimalTest (7/7)

```
OUTPUT:

Animal{name='Generic Animal', age=5}
Some generic animal sound

Dog{name='Buddy', age=3, breed='Golden Retriever'}
Bark

Cat{name='Whiskers', age=2, color='Black'}
Meow
```

Example: VehicleTest (1/4)

```
public class Vehicle {
    private String brand;
    private int year;

    public Vehicle() { }

    public Vehicle(String brand, int year) {
        this.brand = brand;
        this.year = year;
    }

    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }
```

Example: VehicleTest (2/4)

```
    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }

    public void startEngine() {
        System.out.println("The engine is starting...");
    }

    @Override
    public String toString() {
        return "Vehicle{brand='" + brand + "', year=" + year + '}';
    }
}
```

Example: VehicleTest (3/4)

```
public class Car extends Vehicle {
    private int doors; // more instance variable

    public Car() {
        super();
    }

    public Car(String brand, int year, int doors) {
        super(brand, year);
        this.doors = doors;
    }

    public int getDoors() {
        return doors;
    }

    public void setDoors(int doors) {
        this.doors = doors;
    }
```

Example: VehicleTest (4/4)

```
    // method overriding
    @Override
    public void startEngine() {
        super.startEngine(); // call the parent's method
        System.out.println("The car engine is starting...");
    }

    @Override
    public String toString() {
        return "Car{brand='"+getBrand()+"', year="+getYear() + ", doors=" + doors + '}';
    }
}
```

```
OUTPUT:
Vehicle{brand='Generic Vehicle', year=2010}
The engine is starting...
Car{brand='Toyota', year=2020, doors=4}
The engine is starting...
The car engine is starting...
```

’protected’ Access members

- Can be directly accessed from
  - other classes in the same package
  - any descendant classes

Example: ProtectedExample

```
public class Parent {
    protected String name = "Parent Name";
    protected void display() {
        System.out.println("This is a protected method in Parent class.");
    }
}

class Child extends Parent {
    public void showName() {
        System.out.println("Name: " + name); // read parent's protected variable
        display(); // call the parent's protected method
    }
}

public class ProtectedExample {
    public static void main(String[] args) {
        Child c = new Child();
        c.showName();
    }
}
```

```
OUTPUT:
Name: Parent Name
This is a protected method in Parent class.
```

Summary of Access Modifiers

- private: access only from the same class
- default (package): access only from the same package
- protected: access from the same package, from the descendant class in other packages
- public: no restriction

Access Modifier	The same Class	The same Package	Descendants	Everywhere
public				
protected				
default (package)				
private				