Theory: The keyword super

20 minutes 5 / 10 problems solved

Start practicing

Sometimes when we define a new subclass we need to access members or constructors of its superclass. Java provides a special keyword `super` to do this. This keyword can be used in several cases:

- to access instance fields of the parent class;
- to invoke methods of the parent class;
- to invoke constructors of the parent class (no-arg or parameterized).

Let's consider all of these cases on examples.

§1. Accessing superclass fields and methods

The keyword `super` can be used to access instance methods or fields of the superclass. In a sense, it is similar to the keyword `this`, but it refers to the immediate parent class object.

> The keyword `super` is optional if members of a subclass have different names from members of the superclass. Otherwise, using `super` is the right way to access hidden (with the same name) members of the base class.

Example. There are two classes: `SuperClass` and `SubClass`. Each class has a field and a method.

Current topic:

✓ The keyword super

Topic depends on:

✓ Multiple constructors

✓ Inheritance Stage 7

Topic is required for:

Hiding and overriding

Table of contents:

↑ The keyword super

§1. Accessing superclass fields and methods

§2. Invoking superclass constructor

Feedback & Comments

```
class SuperClass {

    protected int field;

    protected int getField() {
        return field;
    }

    protected void printBaseValue() {

        System.out.println(field);

    }

}


class SubClass extends SuperClass {



    protected int field;



    public SubClass() {

        this.field = 30;  // It initializes the field of SubClass

        field = 30;       // It also initializes the field of SubClass

        super.field = 20; // It initializes the field of SuperClass

    }



    /**
     * It prints the value of SuperClass and then the value of SubClass
     */
    public void printSubValue() {

        super.printBaseValue(); // It invokes the method of SuperClass, super is optional here

        System.out.println(field);

    }

}
```

In the constructor of `SubClass` , the superclass field is initialized using the keyword `super`. We need to use the keyword here because the subclass field hides the base class field with the same name.

In the body of the method `printSubValue` , the superclass method `printBaseValue` is invoked. Here, the keyword `super` is optional. It is required when a subclass method has the same name as a method in the base class. This case will be considered in the topic concerning overriding.

§2. Invoking superclass constructor

Constructors are not inherited by subclasses, but a superclass constructor can be invoked from a subclass using the keyword `super` with parentheses. We can also pass some arguments to the superclass constructor.

Two important points:

- invoking `super(...)` must be the first statement in a subclass constructor, otherwise, the code cannot be compiled;

- the default constructor of a subclass automatically calls the no-argument constructor of the superclass.

Example. Here are two classes `Person` and `Employee`. The second class extends the first one. Each class has a constructor to initialize fields.

```
class Person {

    protected String name;
    protected int yearOfBirth;
    protected String address;

    public Person(String name, int yearOfBirth, String address) {
        this.name = name;
        this.yearOfBirth = yearOfBirth;

        this.address = address;

    }



    // getters and setters

}


class Employee extends Person {



    protected Date startDate;

    protected Long salary;



    public Employee(String name, int yearOfBirth, String address, Date startDate, Long salary) {

        super(name, yearOfBirth, address); // invoking a constructor of the superclass



            this.startDate = startDate;

            this.salary = salary;

    }



    // getters and setters

}
```

In the provided example, the constructor of the class `Employee` invokes the parent class constructor for assigning values to the passed fields. In a way, it resembles working with multiple constructors using `this()`.

Report a typo

440 users liked this theory. **8** didn't like it. **What about you?**

Start practicing

Comments (13) Hints (2) Useful links (0) Show discussion