Theory: Multiple constructors

23 minutes 9 / 9 problems solved

Start practicing

§1. Constructor overloading

Sometimes we need to initialize all fields of an object when creating it, but other times it might be appropriate to initialize only one or several fields. Fortunately, for this purpose, a class can have multiple constructors that assign values to the fields in different ways.

You can define as many constructors as you need. Each constructor should have a name that matches the class name but the parameters should be different. The situations when a class contains multiple constructors is known as **constructor overloading**.

Here is an example:

```
public class Robot {
    String name;
    String model;

    public Robot() {
        this.name = "Anonymous";
        this.model = "Unknown";
    }


    public Robot(String name, String model) {

        this.name = name;

        this.model = model;

    }

}
```

The class `Robot` has two constructors:

- `Robot()` is a no-argument constructor that initializes fields with default values;
- `Robot(String name, String model)` takes two parameters and assigns them to the corresponding fields.

To create an instance of the class `Robot` we can use either of the two constructors:

```
Robot anonymous = new Robot(); // name is "Anonymous", model is "Unknown"
 Robot andrew = new Robot("Andrew", "NDR-114"); // name is "Andrew", model is "NDR-114"
```

Bear in mind that you cannot define two constructors with the same number, types, and order of the parameters!

§2. Invoking constructors from other constructors

We can also invoke a constructor from another one. It allows you to initialize one part of an object by one constructor and another part by another constructor.

Calling a constructor inside another one is done using `this`. For example:

2 required topics

- Overloading — In project
- Constructor — In project — 13

2 dependent topics

- The keyword super — In project
- Reducing boilerplate code with Lombok

```
this(); // calls a no-argument constructor
```

If you called a constructor that has parameters you can pass some arguments:

```
this("arg1", "arg2"); // calls a constructor with two string arguments
```

> Remember, the statement for invoking a constructor should be the first statement in the body of a caller constructor.

Here is an extended example of the `Robot` class:

```
public class Robot {
    String name;
    String model;
    int lifetime;

    public Robot() {
        this.name = "Anonymous";
        this.model = "Unknown";
    }



    public Robot(String name, String model) {

        this(name, model, 20);

    }



    public Robot(String name, String model, int lifetime) {

        this.name = name;

        this.model = model;

        this.lifetime = lifetime;

    }

}
```

Now, the class has three constructors:

- `Robot()` is a no-argument constructor;
- `Robot(String name, String model)` is a two-argument constructor that invokes another constructor;
- `Robot(String name, String model, int lifetime)` is a three-argument constructor that fills all fields.

The second constructor invokes the third one and passes `name`, `model`, and `lifetime = 20` to it. The third constructor, in its turn, initializes all fields of the created object.

Let's add an output to the third constructor and see the result:

```
public Robot(String name, String model, int lifetime) {
    this.name = name;
    this.model = model;
    this.lifetime = lifetime;
    System.out.println("The third constructor is invoked");
}
```

Let's now create an instance using the two-argument constructor.

Table of contents:

```
1  Robot andrew = new Robot("Andrew", "NDR-114");
```

The program outputs:

```
1  The third constructor is invoked
```

§3. Conclusion

In this topic, we've covered constructor overloading — creating multiple constructors for the class. Constructor overloading allows us to create an object of the class in different ways depending on the circumstances.

We can also invoke constructors inside other constructors. All in all, Java provides many useful features for writing constructors and defining interactions between them.

Report a typo

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

Start practicing

Comments (8) Hints (0) Useful links (0) Show discussion