Theory: Defining classes

9 minutes 2 / 4 problems solved

Skip this topic Start practicing

When programmers are writing a real program, they use standard classes as building blocks. However, they often need to declare new program-specific classes to better represent the domain area. In this topic, we will see how you can create a **custom class** in Java.

Declaring new classes

A new class is declared with the `class` keyword followed by the name of the class. For example, this is how you would create a class named `Nothing` :

```
class Nothing {
    // empty body
}
```

A class body can include fields, methods, and constructors. **Fields** store data, **methods** define behavior and **constructors**allow us to create and initialize new objects of the class. Fields and methods are considered to be class members. Not all Java classes have fields and methods so sometimes you will see classes without them.

The source code of a class is placed in a `.java` file. Usually, a source code file contains only one class and has the same name as that class, but sometimes a file can contain more classes.

Writing fields

A **field** is a variable that stores data. It may have any type, including primitive types (int, float, boolean and so on) and classes (even the same class). A class can have as many fields as you need.

Let's declare a class `Patient` :

```
/**
 * The class is a "blueprint" patients
 */
class Patient {

    String name;
    int age;
    float height;
    String[] complaints;
}
```

This class represents a patient in a hospital information system. It has four fields for storing important information about the patient: `name` , `age` , `height` , and `complaints` . All objects of the class `Patient` have the same fields, but their values are different for each object.

Creating objects

Let's create an **instance** of the class `Patient` using the keyword **new**:

```
Patient patient = new Patient();
```

When you create a new object, each field is initialized with the default value of the corresponding type.

```
System.out.println(patient.name); // it prints null
System.out.println(patient.age); // it prints 0
```

Creating multiple objects of the same class

The following program creates two objects of the class `Patient` and prints the information about them.

Note that both classes are placed in the same file named `PatientDemo.java`.

```
public class PatientDemo {

    public static void main(String args[]) {

        Patient john = new Patient();

        john.name = "John";
        john.age = 30;
        john.height = 180;

        System.out.println(john.name + " " + john.age + " " + john.height);

        Patient alice = new Patient();

        alice.name = "Alice";
        alice.age = 22;
        alice.height = 165;

        System.out.println(alice.name + " " + alice.age + " " + alice.height);
    }
}

class Patient {

    String name;
    int age;
    float height;
}
```

In the code above, we've created two patients, John and Alice, defined the values of their fields and then printed out the information about them. So, the output of the code above is:

```
John 30 180
Alice 22 165
```

Summary

In this topic, we've learned how to create classes in Java. Custom classes can be very useful because they allow you to define fields and methods that work best for your purposes.

Fields keep the current state (data) of the instances of the class and their values can be different for different instances. You can create objects of the class, assign values to their fields and use those objects in your programs. All in all, classes are a very powerful tool and we hope that you'll use them in your projects!