Theory: Instance methods

19 minutes 5 / 12 problems solved

Start practicing

All methods in Java can be separated into two groups: ***static*** methods and ***instance*** methods. When we deal with all objects of the class, we invoke a ***static*** method. It means that this method type is associated with the class as a whole. Thus, no instance of the class is required to run the method. ***Instance*** method, on the other hand, can only be invoked through an ***instance*** of a class, so that you have to create an object first.

§1. Writing instance methods

Let's declare a class `MyClass` with one instance method `print` . Pay attention that an instance method doesn't have the keyword `static` in its declaration:

```
class MyClass {

    public void print() {
        System.out.println("instance method");
    }
}
```

An instance method is a method that belongs to ***each object that we created*** of the particular class. It can also access its fields. See how it is done in the code below:

```
class MyClass {

    int field;

    public void print() {
        System.out.println(this.field);
    }
}
```

The keyword `this` represents a particular instance of the class `MyClass` . This keyword is optional, but very useful when it comes to managing objects of the class.

What is important about instance methods is that they can take arguments and return values of any type including the same type as the defined class.

§2. Calling instance methods

To call an instance method, you need to create an **object** of the class. Let's create an instance of the class `MyClass` and call the `print` method to see how it all works.

```
public static void main(String[] args) {

   MyClass object = new MyClass();
   object.field = 10;

   object.print(); // prints "10"
}
```

We defined the `print` method so that it prints the value of `field` of the object that called this method. The keyword `this` allows us to access the value of that particular object we've just created. Since we've defined the value of `field` for our object as 10, the code prints out 10 as well.

§3. Cats

Let's go even further and consider a more difficult example of a class `Cat` .

Current topic:

✓ Instance methods — Stage 6

Topic depends on:

✓ Declaring a method — Stage 3

✓ Defining classes — Stage 3

Topic is required for:

Access modifiers — Stage 6

Static members

Annotations

Enums in Java

Inner classes

Generic programming

Rest controller

Table of contents:

Any cat has a name and a state (sleeping or not). A cat can say one of two phrases, **"meow"** or **"zzz"**, depending on its state. Sometimes, after saying **"meow"**, a cat falls asleep. However, it can be awakened by invoking the method `wakeUp`.

Here's the code that corresponds to the class of cats that behave in this way. Make sure to read the provided comments to better understand the methods of the class.

```
/**
 * The class is a "blueprint" for cats
 */
class Cat {

    String name; // the cat's name
    boolean sleeping; // the current state of the cat (by default - false)

    /**

     * The cat says "meow" if it is not sleeping, otherwise it says "zzz".

     * After saying "meow" the cat can sometimes fall asleep.

     */

    public void say() {

        if (sleeping) {

            System.out.println("zzz");

        } else {

            System.out.println("meow");



            if (Math.random() > 0.5) {

                sleeping = true;

            }

        }

    }



    /**

     * This method wakes the cat up.

     */

    public void wakeUp() {

        sleeping = false;

    }

}
```

Now, we can create an instance of this class and invoke its methods. Don't forget that when a cat is created it is awake!

```
public class CatsDemo {

    public static void main(String[] args) {

        Cat pharaoh = new Cat(); // an instance named "pharaoh"
        pharaoh.name = "pharaoh";

        for (int i = 0; i < 5; i++) {
            pharaoh.say(); // it says "meow" or "zzz"

        }



        pharaoh.wakeUp(); // invoking the instance method



        pharaoh.say();

    }

}
```

Note that the program's output can be different because we use `Math.random()` inside the `say` method. Here is an example of the output:

```
meow
meow
meow
zzz
zzz
meow
```

To sum it all up, instance methods allow programmers to manipulate particular objects of a class. They can access the fields of the class with `this` keyword, but it is optional. Instance methods are a great way to work with many objects of your classes!

Report a typo

😐 Feedback sent!

Start practicing

Comments (16) Hints (0) Useful links (0) Show discussion