Theory: Getters and setters

15 minutes 5 / 7 problems solved

Start practicing

§1. Data encapsulation

According to the **data encapsulation** principle, the fields of a class are hidden from being directly accessed from other classes. The fields can be accessed only through the methods of that particular class.

To access hidden fields programmers write special types of methods: **getters** and **setters**. Getters can only read fields, setters can only write (modify) the fields. Both types of methods should be `public`.

Using these methods gives us some advantages:

- the fields of a class can be made read-only, write-only or both;
- a class can have total control over what values are stored in the fields;
- users of a class don't know how the class stores its data and don't depend on the fields.

§2. Getters and setters

Java doesn't provide any special keywords for getter and setter methods. Their main difference from the other methods is their names.

According to the JavaBeans Convention:

- **getters** start with **get**, followed by the variable name, with the first letter of the variable name capitalized;
- **setters** start with **set**, followed by the variable name, with the first letter of the variable name capitalized.

This convention applies to any types except `boolean`. A **getter** for a boolean field starts with **is**, followed by the variable name.

Example 1. The class `Account` has four fields: `id`, `code`, `balance` and `enabled`. Each field has a keyword **private** to hide the field from direct access from other classes. Also, the class has **public** getters and setters for accessing fields through these methods.

Current topic:

Stage 7
Getters and setters

Topic depends on:

Stage 6
Access modifiers

Topic is required for:

Stage 7
Inheritance

Table of contents:

↑ Getters and setters

§1. Data encapsulation

§2. Getters and setters

§3. Conclusion

Feedback & Comments

```
class Account {

    private long id;
    private String code;
    private long balance;
    private boolean enabled;

    public long getId() {
        return id;

    }



    public void setId(long id) {

        this.id = id;

    }



    public String getCode() {

        return code;

    }



    public void setCode(String code) {

        this.code = code;

    }



    public long getBalance() {

        return balance;

    }



    public void setBalance(long balance) {

        this.balance = balance;

    }



    public boolean isEnabled() {

        return enabled;

    }



    public void setEnabled(boolean enabled) {

        this.enabled = enabled;

    }

}
```

Here you can see the different getters and setters for the class `Account`. Just as the convention states, the boolean field `enabled` has a different getter name: it starts with the word `is` instead of `get`.

Let's create an instance of the class and fill the fields, then read values from the fields and output them.

```
Account account = new Account();

account.setId(1000);
account.setCode("62968503812");
account.setBalance(100_000_000);
account.setEnabled(true);

System.out.println(account.getId());      // 1000
System.out.println(account.getCode());    // 62968503812

System.out.println(account.getBalance()); // 100000000

System.out.println(account.isEnabled());  // true
```

Sometimes, **getters** or **setters** can contain a more sophisticated logic. For example, **getters** may return non-stored values (calculated at runtime), or **setters** may also in some cases modify the value of another field according to changes. But usually, getters and setters have a minimum of programming logic.

Example 2. In the following class, the setter `setName` doesn't change the current value if the passed value is `null`.

```
class Patient {

    private String name;

    public Patient(String name) {
        this.name = name;
    }

    public String getName() {

        return this.name;

    }



    public void setName(String name) {

        if (name != null) {

            this.name = name;

        }

    }

}
```

§3. Conclusion

To restrict access to fields from external code make them `private` and write suitable **getters/setters** to **read/change** only the fields you need. Do not forget to make use of the naming convention when writing them.

Note, modern IDEs (such as **IntelliJ IDEA**) can generate getters and setters automatically based on class fields.

Report a typo

530 users liked this theory. **5** didn't like it. **What about you?**

Start practicing

Comments (14) Hints (4) Useful links (1) Show discussion