Object Methods

by Sophia

WHAT'S COVERED

In this lesson, you will explore how to create and use an object's method in a larger program. Specifically, this lesson covers:

1. Accessor Methods
2. Mutator Methods
3. Other Methods

1. Accessor Methods

Accessor methods are public methods in a class that provide read access to the data in the class's attributes (to the extent that read access is appropriate). Depending on the design requirements, an accessor method may just return the data from the attribute, or it may carry out any processing or conversion that is needed.

The convention in Java is that accessor methods should begin with the word "get" and then indicate which attribute's data is returned. For instance, the `UserAccount` code below has a `getUserName()` method that returns the username and a `getDateJoined()` method that returns the date the user joined. In the case of an accessor method that returns a boolean value, though, the name usually begins with the word "is." The method in the `UserAccount` class that returns the boolean indicating whether the user is active or not is called `isActiveUser()`, as demonstrated below:

```
import java.time.LocalDate;

public class UserAccount {
  private String userName;
  private String password;
  private LocalDate dateJoined;
  private boolean activeUser;

  public UserAccount(String userName, String password) {
```

```
    this.userName = userName;
    this.password = password;
    this.dateJoined = LocalDate.now();
    this.activeUser = true;
  }

  // Allow read-only access to user name
  public String getUserName() {
    return userName;
  }

  // Allow read-only access to date joined
  public LocalDate getDateJoined() {
    return dateJoined;
  }

  // Allow activeUser to be read & set (can change)
  public boolean isActiveUser() {
    return activeUser;
  }

  public void setActiveUser(boolean activeUser) {
    this.activeUser = activeUser;
  }
}
```

To define what the methods are in the class, use the same syntax that has already been seen:

EXAMPLE

```
returnType <methodame>(self, <parameter(s)>) {
  /* Statements */
}
```

TERM TO KNOW

Accessor Method ("Getter" Method)
A public method that reads and returns the value in an attribute.

2. Mutator Methods

Mutator methods (also called "setter methods") are methods that are implemented to allow data in the class's attributes to be changed. Mutator methods are only provided if the value of the attributes is allowed to be changed after the object has been constructed. A mutator method requires a parameter for passing in the new value of the attribute.

In the `UserAccount` class that you have been looking at, there is a mutator method, `setActiveUser()`. The method is defined like this in the class with a `boolean` parameter that allows setting the value of the `activeUser` attribute:

```
public void setActiveUser(boolean activeUser) {
    this.activeUser = activeUser;
}
```

To create an account with the username "sophia" and the password "mypass", but then set the activeUser attribute to false rather than the default true, you could write code like this in the application's driver class:

```
public class UserAccountExample {
  public static void main(String[] args) {
    UserAccount account = new UserAccount("sophia", "mypass");
    account.setActiveUser(false);
    System.out.println("User name: " + account.getUserName());
    System.out.println("Is active user: " + account.isActiveUser());
    System.out.println("Date joined: " + account.getDateJoined());
  }
}
```

If you type this code into a file named UserAccountExample.java and run it, the output should look like this:

```
User name: Sophia
Is active user: false
Date joined: 2024-04-29
```

REFLECT

As the output shows, the user's active status has been set to false (rather than the default true).

TERM TO KNOW

Mutator Method ("Setter" Method)
A public method that changes the value in an attribute.

3. Other Methods

In addition to the constructors, accessor methods, and mutator methods that have been discussed, a class can include other methods that carry out relevant operations on the data in the class's attributes. In an upcoming tutorial about debugging classes, you will look into a `toString()` method, but for the moment, let's look at another method that involves passing values via method parameters and getting back a value returned by the method. Here is a method that checks if the username and password match the values for an account:

EXAMPLE

```
public boolean checkCredentials(String userName, String password) {
  return this.userName.equals(userName) && this.password.equals(password);
}
```

Note that when comparing two String values for equality, the code needs to use the equals() method provided by the String class.

TRY IT

Directions: Try adding this method to the UserAccount class and then calling it from the `main()` in the application class.

BRAINSTORM

How could you use this to produce code that indicates if the login was successful using a username and password entered?

Let's write a small program that prompts the user to enter a username and password. The program then creates a new UserAccount object, prompts the user again to enter a username and password, and then checks to see if the login is correct. You have not read any user input for a while, so let's use a Scanner to read input.

TRY IT

Directions: Type the following code into a file named UserLoginExample.java:

```
import java.util.Scanner;

class UserLoginExample {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.println("Create New User Account: ");
    System.out.print("Enter User Name: ") ;
    String user = input.nextLine();
    System.out.print("Enter Password: ");
    String passwd = input.nextLine();
```

```
    System.out.println("Creating account...");
    UserAccount acct = new UserAccount(user, passwd);

    System.out.println("Now Check the Login: ");
    System.out.print("Enter User Name: ");
    String userToCheck = input.nextLine();
    System.out.print("Enter the Password: ");
    String passwordToCheck = input.nextLine();

    boolean result = acct.checkCredentials(userToCheck, passwordToCheck);
    if(result) {
      System.out.println("Login successful.");
    }
    else {
      System.out.println("Login failed!");
    }
  }
}
```

Run the `UserLoginExample` program. The result of a valid login should look like this (keep in mind that both the username and password are case-sensitive):

```
Create New User Account:
Enter User Name: Sophia
Enter Password: mypass
Creating account...
Now Check the Login:
Enter User Name: Sophia
Enter the Password: mypass
Login successful.
```

A failed login attempt should look like this:

```
Create New User Account:
Enter User Name: Sophia
Enter Password: mypass
Creating account...
Now Check the Login:
Enter User Name: Sophia
Enter the Password: MyPassword
Login failed!
```

REFLECT

The code above shows how to write a method that checks the username and password stored in the object in a case-sensitive manner. While passwords are typically case-sensitive, think about how the `toUpperCase()` or `toLowerCase()` method provided by the `String` class could be used to check the username in a non-case-sensitive manner.

THINK ABOUT IT

In looking at the results, were they what you expected to see?

SUMMARY

In this lesson, you learned how to create methods that provide access to the data in attributes in a class (called **accessor methods**, also called "getter" methods). You have also seen how to **create mutator** or "setter" **methods** that allow the values of attributes to be changed after objects have been created. Finally, you have seen an example of **another method** that carries out a useful process using the data in an object.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source py4e.com/html3/

TERMS TO KNOW

Accessor Method ("Getter" Method)

A public method that reads and returns the value in an attribute.

Mutator Method ("Setter" Method)

A public method that changes the value in an attribute.