Going Further with Classes

Objectives

The objectives of this practical session are to:

1 Define overloaded constructors, overloaded instance methods, class variables, class methods

Reference Material

This practical session is based on material in the *Going Further with Classes* chapter.

Overview

This practical consists of two parts plus an optional part. In Part 1, you will take a pre-defined class called `Customer` similar to the one that you defined yourself in the last practical. It has two instance variables for the customer's name and account number, respectively, but this time only the name is passed as an argument to the constructor.

First, you will add another instance variable to that class, to represent the customer status; the current known values are "A" for active, "I" for inactive and "H" for on-hold. The existing constructor will set the status to the default value of "A". You will also need to provide a second (i.e. overloaded) constructor to initialise the status to "A" or "H", as passed in as a second argument - in other words, this constructor will receive both name and status as arguments.

You will add code to automatically generate the account number when constructing a new customer object, by incrementing a static variable (class variable), `lastUsedAccountNumber`, which you will add to the `Customer` class. You will also provide a static method (class method) which allows the user to reset the value in the class variable.

You will need to define a test class (similar to what you had in the previous exercises) in order to test the constructors and methods of your `Customer` class.

Practical

Part 1. Overloading Constructors

1. Create a new IntelliJ project and paste the given class into it. Make sure you're using the correct package structure.

2. Look at the class definition `Customer` and notice the comments that indicate where you should insert your code. Note that the existing constructor takes a single argument to set the customer's name, and does not update the account number. Until we make some changes (below), every customer will have an account number set to zero.

3. Create a CustomerTest class with a main method where you can create a couple of instances of Customer class and test how they work. Is your assumption from the previous step correct?

4. In the `Customer` class definition, declare another `private` instance variable to hold the customer's status as a `char`, and provide an accessor method `isHeld()` which returns boolean *true* if the status is equal to 'H', otherwise *false*. Also provide a `getStatus()` method that returns the `char`.

5. Define a second constructor that has two arguments to initialise the customer's status as well as their name. If the value received is not 'A' or 'I' or 'H' (for active, inactive or on-hold), then set it to 'H'. Remember that you can use `this()` to avoid duplicating code in multiple constructors. Modify the first constructor (which does *not* receive a `char` argument) to set the status to 'A'.

6. Save the project and fix any compiler errors before proceeding. Write up some code in your main method to prove that your code is working correctly.

Part 2. Overloading Instance Methods

1. Notice the method `changeDetails()` provided in the `Customer` class. In your main method add code to call the `changeDetails()` method to verify that the method changes the customer's name.

2. Now you need to add another instance method to the `Customer` class to change both the name and status. As you have already learnt, there's no need to specify different names for methods as long as they have different signatures (i.e. different numbers and/or types of arguments). This is known as method *overloading*.

3. Examine the `Customer` class and, where indicated by the comments, define a second `changeDetails()` method that takes a `String` argument and a `char` argument to change the customer's name and status, respectively.

4. Back in `CustomerTest`, add code to call the second `changeDetails()` method for one of your `Customer` variables, updating the name to some new value, and the status to 'I'.

5. Congratulations! You have now learnt how to define and use overloaded constructors and overloaded methods.

Using Class Variables and Class Methods

1. Add a class variable to the `Customer` class to hold the last used account number. This needs to be a class variable because there is only one last-used-number, and it's the same for all customers. You could initialise this variable to some reasonable value, such as 1000, in the variable declaration. What would be a suitable type for this variable?

2. Modify the constructors so that the last-used account number is incremented by 1, and assigned to the instance variable `accountNumber`. Bearing in mind that one constructor may be called from the other, you can implement this requirement by changing code in just one of the constructors. Test your work in the `main` method.

3. Define a class method 'setLastUsedAccountNumber' to allow the static variable to be reset to a new value.

4. In the `CustomerTest` class, call your class method to set a different last-used account number, e.g. 2000.

5. After calling the class method, create two new customer objects - assigning the object references to the variables you used before: `customer1` and `customer2`. (This is for teaching purposes - you would not normally do this, at least not in this kind of program. What has happened to the two customer objects you built earlier?) Save the project, execute `CustomerTest` and make sure that your class method really works.