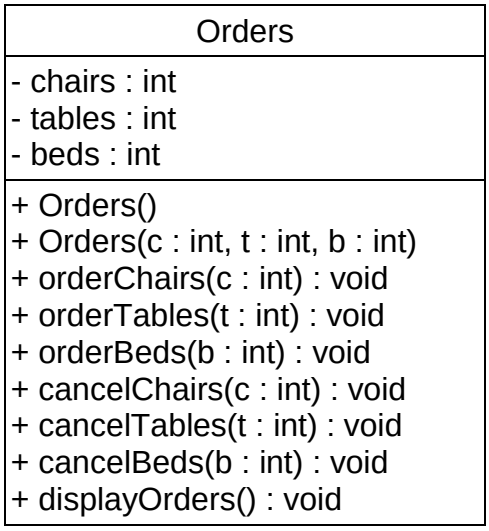
CIS 255 • Java Programming
Laboratory Assignment 9: More Work with Classes

In this laboratory assignment you will work with Java class syntax to create a class to track the number of various items ordered from a warehouse.

Background

A furniture company needs to track the number of chairs, tables, and beds ordered from a warehouse over a given period of time. Instead of "setting" the quantities ordered with mutators, the methods should either add to the existing sums or indicate the number to be subtracted due to an order cancellation. There should also be a method to display a summary of the orders placed so far.

The following is a UML class diagram listing the fields and methods required:



Step 1: Launch TextPad and Save a New Document

You'll be using the starter file when you instantiate the class later on, but for now, you need to launch TextPad (or a similar Java code editor) and start a new document. Remember that TextPad highlights syntax based on the file extension, so you need to save the file as a Java file before you start typing. The name of your class will be `Orders`, so save the file as **Orders.java** where you want the file to be stored.

Step 2: Write the Class Header and Declare the Fields

Start your file with comments at the top that include your name, the course number, the laboratory assignment, and a brief description of the program's contents. You may also wish to include the date (perhaps after the laboratory assignment number). As you continue your code, you may go ahead and add comments to the statements or groups of statements that you believe deserve additional explanation, or you may wait until after you have written your entire class definition to add comments.

This program will not require any `import` statements unless you decide to display the results and error messages using dialogs (in which case you need the `import` statement for the class `JOptionPane`); otherwise, the next item you write should be the class header.

After the header, write the two brackets for the class body. After the opening curly bracket, declare three fields, one each for the number of chairs, tables, and beds ordered thus far (whole numbers). All three of these should be `private`.

Step 3: Define the Constructors

The class requires two constructors: a no-arg constructor and a constructor with three parameters corresponding to each of the types of furniture. The no-arg constructor is a `public` method with no return type, a name matching the class name, and empty parentheses. In the body of this constructor, assign each field a "default" value of 0 indicating that no items have been ordered thus far.

The second constructor is a `public` method with no return type, a name matching the class name, and a parameter list in parentheses including three variables capable of receiving whole numbers. Make sure the parameters do not shadow the fields. In the body of this constructor, use a series of `if` statements to verify that none of the parameters are negative before assigning them to the fields. If a particular parameter is negative, display an error message and assign the default value of 0 to the corresponding field. (Check the parameters individually – a non-negative value for one parameter should be accepted even if one or more of the other parameters is negative.)

Step 4: Define the Order Methods

For each of the three types of furniture, define a method to allow the programmer to add an order for that type of item to the object. Each method should have a single whole-number parameter with a name that doesn't shadow any of the fields. If the value of that parameter is non-negative, **add** its value to the corresponding field; if not, display an error message and leave the field as-is. These methods will not return a value.

In each one of these methods, you are adding the parameter's value to the existing value of the field, not replacing it! Example: if 5 chairs have already been ordered, and a call to the method to order chairs has an argument of 12, the field that stores the number of chairs should increase to 17.

The names of the methods other than the constructor do not have to begin with set and get. Sometimes a class contains methods that allow interaction with the fields other than strictly assigning values to, or returning, the values of fields.

Step 5: Define the Cancellation Methods

For each of the three types of furniture, write a method to allow the programmer to cancel an order for that type of item to the object. Each method should have a whole-number parameter with a name that doesn't shadow any of the fields. If the value of that parameter is non-negative, **subtract** its value to the corresponding field; if not, display an error message and leave the field as-is. These methods will not return a value.

Although not required, you may also want to make sure that the quantity of the cancellation does not exceed the number of that type of furniture ordered so far (e.g., you can't cancel 15 beds if only 10 beds have been ordered thus far); if it does, display an error message.

Step 6: Define the Display Method

Finally, write a `void` method with no parameters that contains output statements to display the current order counts for each type of furniture. The output should appear similar to this:

```
Furniture Orders So Far:
Tables:  12
Chairs:  63
Beds:  5
```

You don't have to call the accessor methods when writing code in the same class. This method has direct access to the fields in the same manner as any other method within the class.

Step 7: Compile Your Class Definition

To determine if there are any syntax errors in your class definition, and to generate the byte code once your source code is free of syntax errors, select the item **Compile Java** from the **External Tools** submenu of the Tools menu. If the Tool Output area indicates that there are syntax errors in your code, double-click the line that describes each syntax error to see the line that contains the error (or, in some cases, the line after the error).

Remember that you cannot execute an object-oriented class definition that does not contain a `main` method. The `main` method will be in a separate source code file.

Step 8: Download the Starter File and Open It in TextPad

On Blackboard, click the link for the starter file to download it. Make sure you insert your first initial and last name where indicated (**FLast_Lab09.java**) and that you save the file in the same location where `Orders.java` is saved. Once you have downloaded the file, open it in TextPad or a similar Java code editor.

Step 9: Customize Your Code and Add `import` Statements

Edit the comments at the top of the code to include your name, the course number, the laboratory assignment number, and a brief description of the program's contents. You may also wish to include the date (perhaps after the laboratory assignment number). Also, modify the class name to match your file name. As you continue your code, you may go ahead and add comments to the statements or groups of statements that you believe deserve additional explanation, or you may wait until after you have written your entire program to add comments.

This program will deal with interactive input, so you will need to decide whether to use the console or dialogs; include the `import` statement that corresponds to your choice.

Be consistent in your method of interaction with the user. Using both dialogs and the console in the same program may confuse the user.

Step 10: Declare Variables in `main` for Primitive Values and Input / Output

You'll need at least one variable to store the user's input for one order. If using the console for input and output, you will also need a `Scanner` object for `System.in`; if using dialogs, you will need a temporary `String` variable for the value returned by `showInputDialog` before it is converted to the proper type.

Step 11: Create an Instance of the Class Using Constructors

Your program should contain at least one `Orders` object. You may use either of the constructors; if you choose to use the second constructor, you can hard-code the initial values for each type of furniture, or you can prompt the user for input for each initial value.

Step 12: Modify an Object using the Order and Cancellation Methods

For each field, perform at least one order and at least one cancellation (there should be at least three orders and three cancellations). Overall, at least one order and at least one cancellation should be based on user input; the values for the other orders and cancellations may be hard-coded.

Step 13: Display the Order Totals

Invoke the display method to show the order totals for each type of furniture.

Step 14: Compile Your Code and Execute the Byte Code

When you compile a program that makes use of another class definition, the code in this program will be validated against the syntax of the class definition. You will receive errors if you attempt to use a constructor or another method in a way that is inconsistent with its definition within the other class. Compile and correct these errors until you have successfully removed any errors that exist.

In the **External Tools** submenu of the **Tools** menu, select **Run Java Application**. Enter some test values to verify that the results match what you believe they should be. To test multiple sets of input, you'll need to run the program multiple times. Make sure that you test some invalid (negative) values for the constructor (if you used the second constructor) and for the orders and cancellations so that you can verify the program's behavior.

Step 15: Add Inline Comments and Submit Your Code

Make sure that **each file** has **three or more inline comments** (a total of **six comments**) near the statements that you believe warrant the additional description (especially those that deal with classes and objects). These can be simple line comments at the end of a line of code that needs explanation, or block comments where each precedes a series of statements described in the comment.

Follow the instructions on the course website to attach and submit your source code.

Make sure you complete the submission process! If you attach your file without submitting the assignment, your file will not be received!