Java Classes Practical

Objectives

The objectives of this practical session are to:

- Define a new class
- Provide a class with constructors, methods and variables
- Create objects of that class
- Test the instance methods of a class

Overview

This practical consists of two parts plus an optional part. In Part 1, you will define a simple class called `Employee` that will model an employee of a company. You will provide that class with just two instance variables for the employee's name and age, respectively. You will also provide a constructor to initialise these variables with some values. To test this class, you will also need to define another class called `EmployeeTest`. Here, you will create two objects of the `Employee` class and display the details of each employee. Since, the `Employee` class does not (as yet) have any instance methods, you will have to access its instance variables directly.

As you have learnt, it is bad practice to allow a user of your class to access your instance variables directly. So, in Part 2 of this practical, you will provide the `Employee` class with two instance methods to return the values of the employee's name and age, respectively. Then, you will modify the `EmployeeTest` class to make use of these new instance methods rather than accessing the instance variables directly. You will also add another instance method to the `Employee` class to allow a user of the class to increment the employee's age. The company isn't really interested in an employee once he or she reaches retirement age, but, to be kind, you are going to freeze their age once they reach retirement age. In other words, your method will not increment the employee's age beyond, say, 65.

Practical

Part 1. Defining a Class and Creating Objects of that Class

1. In IntelliJ, reopen the first project you created, the one with the "Employee Details:" print statement. Alternatively, create it afresh.
2. You should have two classes in your package, one called `EmployeeTest`, and another called `Employee`.
3. In the `Employee` class definition, declare two `public` instance variables: a `String` variable to hold the employee's last name and an `int` variable to hold the employee's age. Note that code assist is available through **CTRL+SPACE** (e.g. when you're halfway through typing 'String'!).

4. Return to the class definition for `EmployeeTest`. In the `main()` method, declare two variables of the `Employee` class. Then, use the `new` operator to create *two* objects of this class. Assign suitable names and ages for each Employee.
5. Add code to display the details of each employee (using `System.out.println()`). Since you haven't provided any instance methods yet, you'll have to access the instance variables directly. Use the '`\t`' escape character to insert one or more tabs between the employee's name and age.
6. Save the project and fix any compiler errors. Execute the program `EmployeeTest`. Check that it displays the correct details of each employee.
7. Go back to the `Employee` class and change the access modifiers of the two instance attributes to `private`. Does the program still run? Why? Why not?
8. Still in the Employee class, use the top menu **Code > Generate > Getter and Setter**, to get the editor to create accessor methods for both variables.
9. Use the **Code** menu again to generate a constructor that has two arguments to initialise both of the employee's instance variables.
10. Save the project and fix any compiler errors before proceeding.
11. Return to the class definition for `EmployeeTest`. In the `main()` method, declare two more variables of the `Employee` class. Then, use the `new` operator to create *two* objects of this class, use your newly created constructor.
12. Add code to display the details of each employee (using `System.out.println()`).
13. Save the project and fix any compiler errors. Execute the program `EmployeeTest`. Check that it displays the correct details of each employee.
14. Extra: Can you make sure that an employee is older than 18 years old?

Part 2. Defining Instance Methods in a Class

1. In the `Employee` class definition, provide another public instance method, `incAge()`, that increments the employee's age. Do a "sanity check" before performing the operation, e.g. check that the employee's age is less than the retirement age of 65.
2. In the `EmployeeTest` class, after displaying the initial details of each employee, call the `incAge()` method of one of the employees and then redisplay the details of both employees.
3. Save and run, to check that your `incAge()` method performs correctly.

4. Enclose your call to `incAge()` in a `for` loop, and verify that the age doesn't increase beyond 65.
5. In the `Employee` class definition, declare another `private` instance variable to hold the employee's first name. Modify the `getLastName()` method so that it returns a string containing the employee's full name, i.e. first name concatenated with last name.
6. Change your constructor so that you can assign values to all three attributes.
7. In `EmployeeTest`, modify the creation of your two `Employee` objects to use the new overloaded constructor that allows you to assign a full name to an employee. Execute the `EmployeeTest` class and check that the full name of each employee is displayed.
8. Add a new variable to the `Employee` class for employees' retirement age. You could initialise this variable to some reasonable value.
9. Modify the `incAge()` method so that it won't increment an employee's age beyond the value held in the retirement age class variable.
10. Define a new method to allow this retirement age to be changed.
11. In the `main()` method of the `EmployeeTest` class, call your new method to set the initial retirement age.
12. To test all this, perhaps you could have two loops, changing the retirement age between them, to see how the behaviour of one of your employees is different!