Additional Language Features

Overview

In this lab, you will write an application that makes use of various additional language features in the Java programming language. You will also enhance previous lab code to use these language features where appropriate.

Source folders

Student project: `StudentAdditionalLanguageFeatures`
Solution project: `SolutionAdditionalLanguageFeatures`

Roadmap

There are 5 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Using autoboxing and unboxing
2. Defining and calling variable-argument-list methods
3. Defining and using enums
4. Using `static` imports
5. Additional suggestions

Exercise 1: Using autoboxing and unboxing

Open `AutoBoxingUnboxing.java`, and write an application that explores autoboxing and unboxing in Java.

Suggestions and requirements:

- Write a simple `main()` method that declares an `Integer` variable. Try to assign it an integer constant (e.g. 42). Does this work? Now try to assign it a (previously declared) `int` variable. Does this work?
- Try to assign your `Integer` variable to an `int` variable. What happens?
- Create an `ArrayList` object (you'll need an `import` statement to import the `java.util.ArrayList` class). Try to add a mixture of `Integer` objects and `int` variables to the `ArrayList`. Does it work?
- Write a loop to iterate through your `ArrayList`. Each time round the loop, try to assign the current element to an `Integer` variable, and also to an `int` variable. What works? Explain to yourself what's happening under the covers.

Exercise 2: Defining and calling variable-argument-list methods

Open `Varargs.java`, and write a method named `sum()`. The method should take at least 2 `double` parameters, plus any number of additional `double` values, and return the sum.

Suggestions and requirements:

- Implement `sum()` as a varargs method. Inside the method, use a loop to add up all the incoming values.
- Call `sum()` from `main()`, passing in a variety of different values.
- What happens if you pass in exactly 2 parameters?
- What happens if you pass in more than 2 parameters?
- What happens if you pass in fewer than 2 parameters?
- What happens if you pass in 2 `double` parameters, plus some illegal parameter types?

Exercise 3: Defining and using enums

Open `Employee.java`, and take a look at the existing code. This is the solution code from earlier in the course.

Enhance the `Employee` class so that it holds the employee's contract type (full-time, part-time, or casual). Define an enum type named `ContractType` to represent the allowable contract types. Modify the constructor/display capabilities of the `Employee` class, to initialize and to output the employee's contract type. Test the `Employee` class in `UseEmployee.java`.

Exercise 4: Using static imports

Still in the `Employee` class, add another method named `displayJoiningDateInfo()`. The method should make use of various `static` members of the `Calendar` class, to display detailed information about the employee's joining date.

Suggestions and requirements:

- The `Calendar` class provides extensive date/time capabilities for modern Java code. `Calendar` is an abstract class, so you can't create a `Calendar` object itself; instead, you create a `GregorianCalendar` object as follows (`GregorianCalendar` is a concrete subclass of `Calendar`):

  ```
  Calendar cal = new GregorianCalendar();
  ```

  Once you've created your `GregorianCalendar` object, you can set its date/time based on an existing `Date` object as follows (the following code snippet assumes you have a `Date` object named `joined`):

  ```
  cal.setTime(joined);
  ```

- You can invoke the `get()` method on your (`Gregorian)Calendar` object, to get various pieces of information about the date/time. The `get()` method can take a `static` value from the `Calendar` class, to indicate which piece of information you require. For example, the following code snippet gets the year value:

  ```
  int year = cal.get(Calendar.YEAR);
  ```

- Write code to get various pieces of date/time info from your (`Gregorian)Calendar` object. You'll need to make use of several `static` values from the `Calendar` class… use `static` imports to simplify your code.

Exercise 5 (If time permits): Additional suggestions

Refactor the `ContractType` enum definition, which represents the employee's contract type. Specifically:

- Add an instance variable to the `ContractType` enum, to provide a textual representation of each enum value (e.g. "full-time", "part-time", and "casual").
- Add another instance variable to the `ContractType` enum, to indicate how many days paid vacation an employee gets depending on their contract type (e.g. full-time employees get 25 days paid leave per year, part-time employees get 10 days paid leave per year, and casual employees sadly don't get any days paid leave per year).
- Add a constructor to the `ContractType` enum, to initialize the instance variables you've just added. Also modify your enum field definitions, to pass appropriate values into the constructor.
- Add methods to the `ContractType` enum, to encapsulate its instance variables as you see fit.