Lab: Advanced Language Constructs

Preamble

To do this lab, first download and open the NetBeans project template provided on Campus.

For each exercise, please test thoroughly your code. You can do this in a separate class, e.g. `Test`, or in the method `main` of the class you have to test.

1 Generics

The class `generics.IntegerSequence` in the project template implements a sequence of integers, with basic operations like `add`, `remove` and `get`. Assume that some application you have to develop requires not only sequences of integers, but also sequences of reals, sequences of strings, etc.

1.1 Solutions

Give three solutions to this requirement, along with their benefits and drawbacks.

1.2 Generic class

Implement the solution based on generics. Test your solution by defining three variables: `integerSeq` (sequence of integers), `stringSeq` (sequence of strings) and `objectSeq` (sequence of objects).

`String` is a subclass of `Object`. Does this imply that a sequence of `String` is a subclass of a sequence of `Object`? How can you check that? Explain what you observe.

1.3 New feature: max() method

Add to your class the method `max()`, which returns the greatest element in the sequence. You are faced with two problems; the first one is conceptual, the second one is practical. How do you solve them?

1.4 New feature: removeAll() method

Add to your class the method `removeAll()`, which removes from a sequence all the elements of another sequence passed as a parameter, and returns whether the sequence was modified. The parameter sequence can be of any type: we must be able to remove from a sequence of `Number` the elements of a sequence of `Integer` (which is a subclass of `Number`) or vice versa.

2 Annotations

In this exercise, we define the annotation `@TestSpec` for use in conjunction with the basic automatic testing test tool we will develop in exercise 3.3. Given some test name, e.g. "test1", "test2", etc. the tool will execute the methods of a class in the order specified by the `@TestSpec` annotations in the class. Note: To make things simple, we assume there can only be one `@TestSpec` annotation for each method.

2.1 Declaring annotations

Based on the description above, give the values of the annotation, their type and their possible default value. Also, give the target of the annotation and its retention policy.

Next, develop the annotation in the package `annotations.definition`.

2.2 Using annotations

Tag the methods of the class `annotations.TestClass` with the annotation `@TestSpec` using values of your choice.

Also, check that (1) you cannot use the annotation for a class or an attribute, (2) you cannot specify several annotations `@TestSpec` for the same method.

2.3 Processing annotations

So far, the annotations we declared and used are of no use. We need to develop some program that will retrieve them using reflection and perform some processing based on their values. This is the topic of exercise 3.3.

3 Reflection

3.1 Class analysis

Implement the method `printMethods` of the class `reflection.ClassIntrospector`. This method takes a class as a parameter and prints the name and the number of parameters of all the methods declared in the class. Test your implementation with the classes `Object`, `Sequence` and `ArrayList`.

Next, complete the method `printMethods` so that it also prints the same information for all the parent classes up to and including `Object`.

3.2 Code execution

In the class `ClassIntrospector`, develop the method `invokeDefaultConstructor`, which calls the default (i.e. no-arg) constructor of the class passed as a parameter and returns the newly-instantiated object. If the class does not have a default constructor, or if invoking the default constructor throws some exception, the method must throw `IllegalArgumentException`. Make sure you carefully define the method's prototype before you develop it.

Test your method using various classes.

3.3 Test tool

In the class `reflection.ClassTester`, develop the method `testClass`, which takes two parameters: a class to test (of type `Class`), and the name of a test to execute on the class (of type `String`).

The method must: (1) instantiate an object of the class, (2) call on the newly-instantiated object all the methods of the class with an annotation @TestSpec corresponding to the name of the test; methods are called in the order specified by the annotations.

To make things easier, we make the following assumptions: (1) the parameter class has a default constructor, (2) all the methods tagged with the annotation @TestSpec are public instance method; they all return void and they do not take any parameter.

Test your method with the class `TestClass` of exercise 2.2.

3.4 Improving the test tool

Make the annotation `@TestSpec` repeatable, so that the following scenario is now possible: test “test1” will call methods `foo()` and `bar()`, whereas test “test2” will call methods `baz()` and `bar()`.