Lab Worksheet 1. Java Functional Interfaces and Unit Tests

Exercise 1: `MyPredicate` and `Predicate<T>` examples

(a) Download the Java source code provided on Moodle under today's Lab session. These are the two files we also discussed during the lecture: the functional interface `MyPredicate` and the class `SelectPositive`, which implements the interface. On line 17, we make the method call `printSelected()`, where the first argument is an integer array, and the second argument is a call to create a new object of the `SelectPositive` class, thus enabling the `printSelected()` method to call the apply method of the functional interface. Replace the "`new SelectPositive()`" with a lambda expression, which (again) checks if a value is positive.

(b) For this exercise, we will print the negative numbers of a list, but instead of using the custom `MyPredicate()`, we will use the Java built-in functional interface `Predicate<T>`.

The `Predicate<T>` functional interface represents a predicate (i.e. a Boolean-valued function) of one argument. It contains a `test()` functional method. Using the built-in functional interface saves us a few lines of code, since we do not have to create our own custom interface and implement its methods.

Steps involved:

(i) Create a new class named `SelectNegative`. Copy the `printSelected()` method from before, and modify it so that the second argument is the Java built-in `Predicate<T>` functional interface (instead of our user-defined `MyPredicate`, which we had been using until now).

(ii) In the `printSelected()` method you will also need to replace the `apply()` method call of `MyPredicate` with the `test()` method call of `Predicate<T>`.

(iii) Create the `main` method and make a call to the `printSelected()` method, so that it prints the elements of the list with a negative value.

Exercise 2: `Function<T,R>` functional interface

Create a class named `Adder`, and a method

```
int[] adder(int[] list, Function<Integer, Integer>)
```

The first argument of the method is an integer array, and the second is an object of the Java built-in `Function<T, R>` functional interface, which takes an integer as input, and outputs another integer.

Call the method `adder()`, so that it adds the value 2 to all elements of its array. The method should return the updated array.

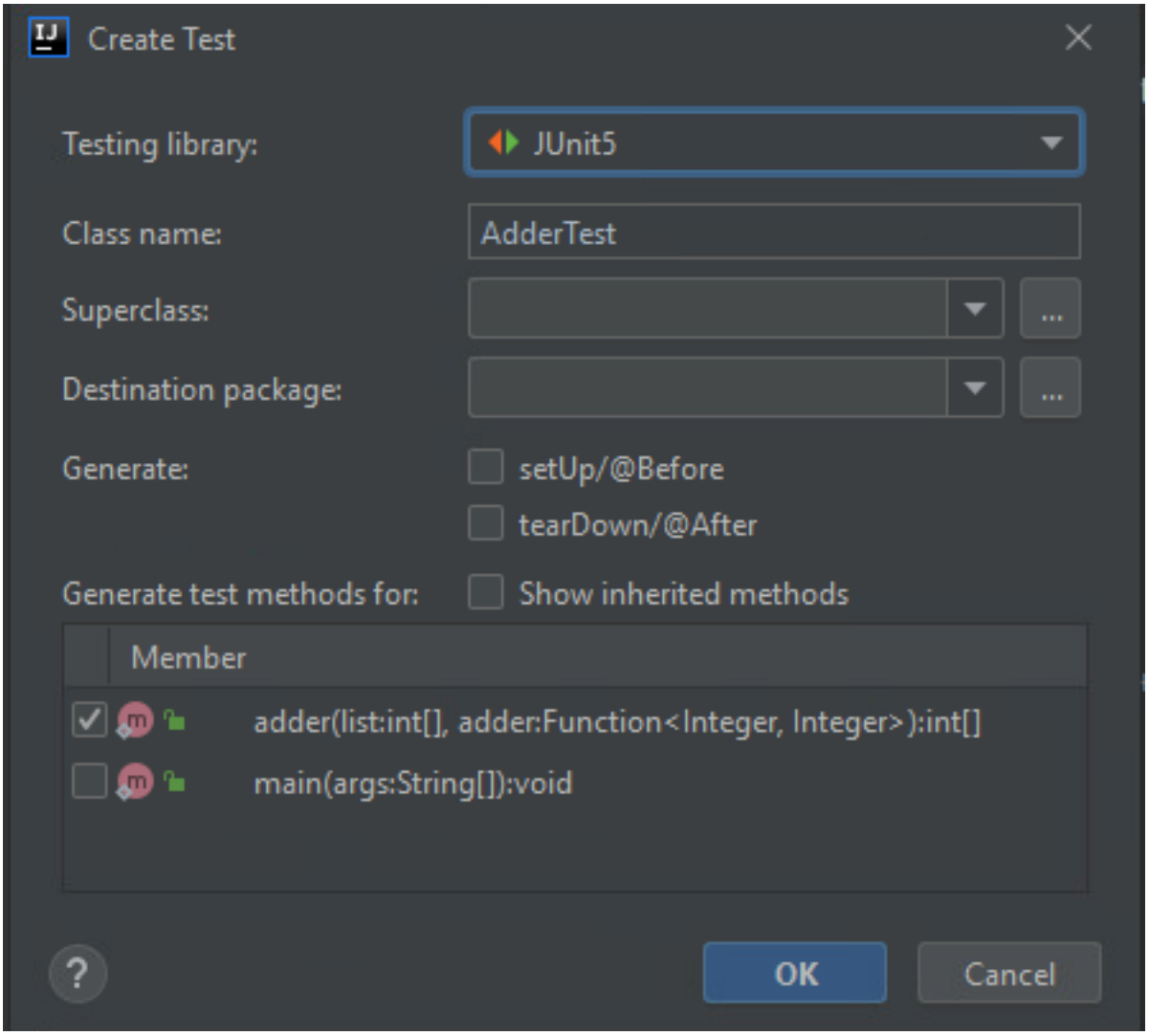
Exercise 3: Unit Tests

Unit testing is an approach to testing a unit of code in isolation, as opposed to testing the whole application. Unit testing is carried out by developers and not by end users and is often used for automated regression testing. Unit testing is particularly useful because it is often easier to test a small piece of code (typically a class or a method) and it also enables testing before integration of modules, hence making the integration smoother.

Unit tests rely on test data that describe the correct behaviour of the unit. Typically, test data is generated manually. When developing test data, it is common to use the code coverage (the percentage of lines of code executed at least once during the tests) as an indicator of how thorough the testing is.

Write a JUnit test, which checks whether the `adder()` method executes correctly. You should use the `assertArrayEquals()` statement, since you will be checking for array equality.

To produce unit tests in IntelliJ, put the cursor on the line where the `adder()` method was written. Then right-click and select "generate", and the "Test…". The testing library should be 'JUnit5.4'. If you are seeing a message that Junit5 is not in the classpath, click on "Fix", so that it gets installed. At the bottom of the window you'll see a list of the methods of the `Adder` class, for which you want to generate test methods. Select the `adder()` method and press OK. (See also these instructions: https://www.jetbrains.com/help/idea/configuring-testing-libraries.html).



To run the test, right-click the `Adder` file in the project tree and select "Run '`AdderTest`'".