CSE118 Spring 2023

Assignment 1

In this assignment you will write two simple Kotlin classes and tests to confirm they functions as expected. You will also gain familiarity with a well-known development tool for building and testing Kotlin projects.

This assignment is worth 10% of your final grade.

Installation

Download and install Intellij IDEA Community Edition: https://www.jetbrains.com/idea/download

Setup

Download the starter code archive from Canvas and expand into an empty folder. I recommend creating a folder for the class and individual folders beneath that for each assignment.

The starter code archive contains the following files that you will modify and two empty resources folders that can be ignored:

```
src/main/kotlin/CalendarArray.kt
src/main/kotlin/PostfixCalculator.kt
src/test/kotlin/CalendarArrayTest.kt
src/test/kotlin/PostfixCalculatorTest.kt
```

Do NOT modify the following file:

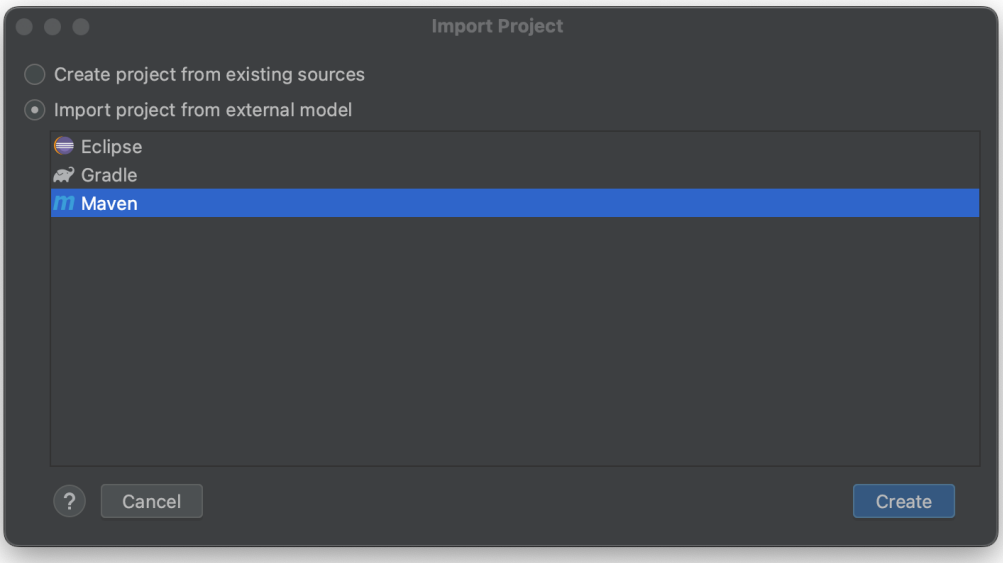
```
pom.xml
```

Start Intellij IDEA and import the project:

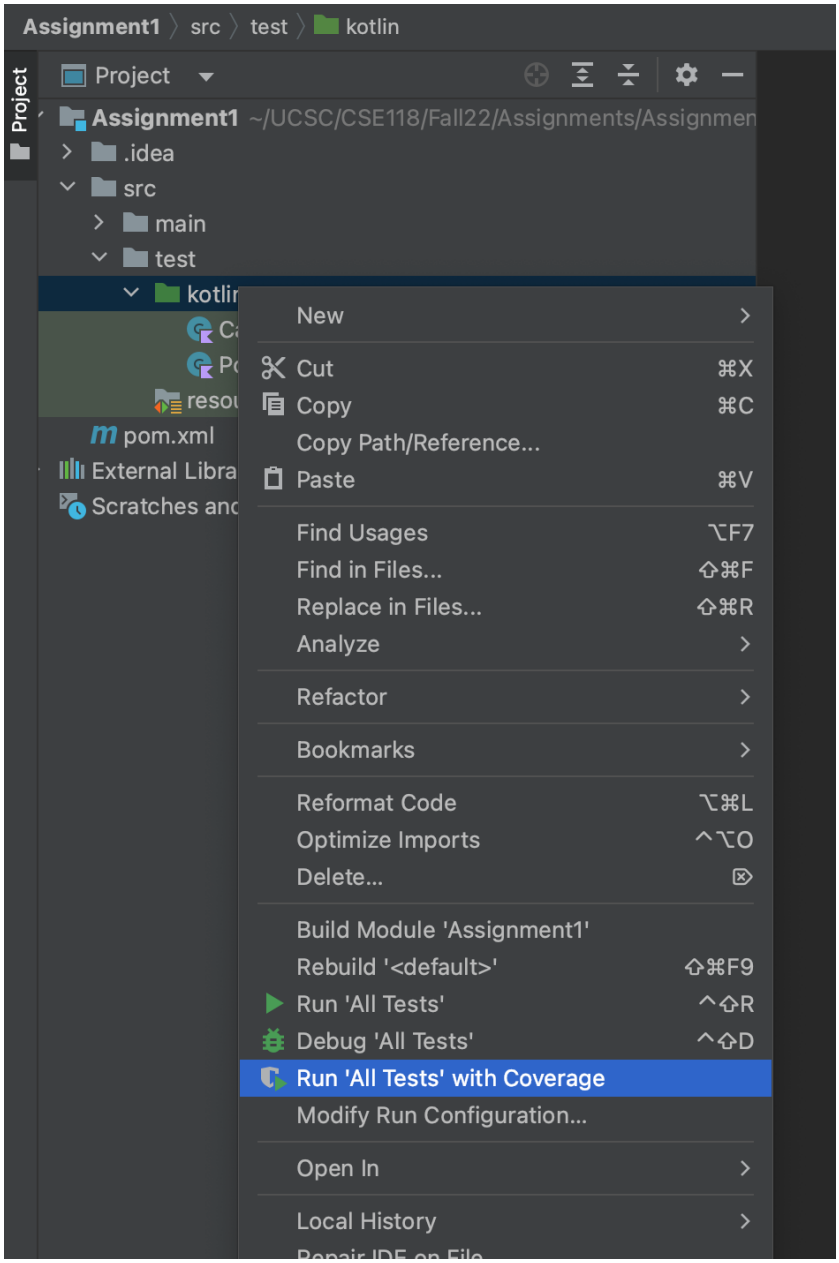
File → New → Project from Existing Sources

Select the folder where you expanded the starter code and click **Open**.

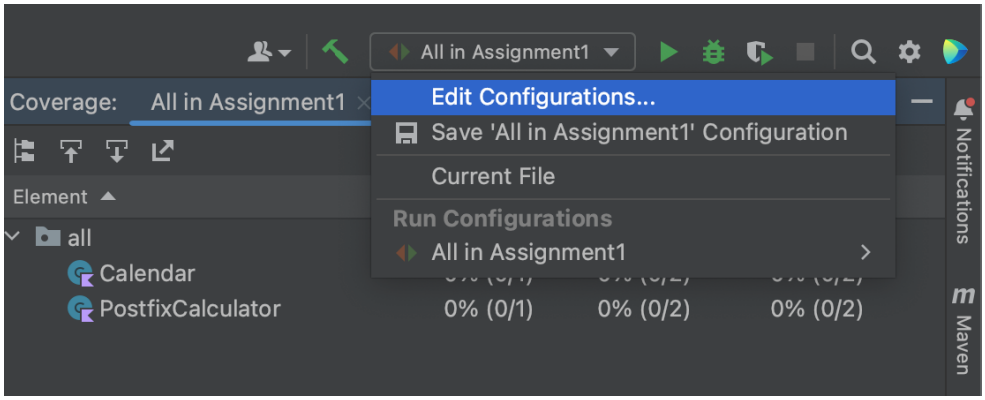
Select the option to import from a Maven external model and click **Create**:



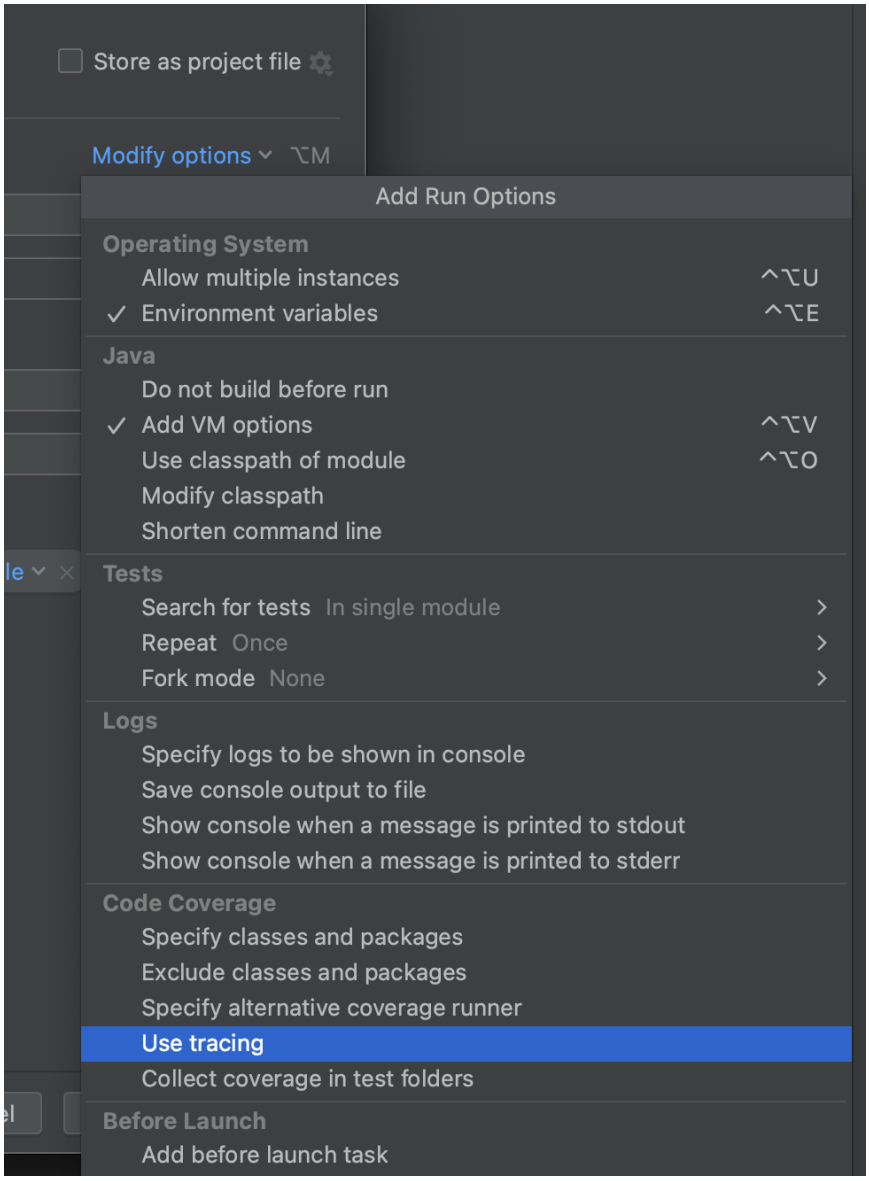
Select the context menu (right-click) on the Kotlin test folder and select **Run 'All Tests' with Coverage.**



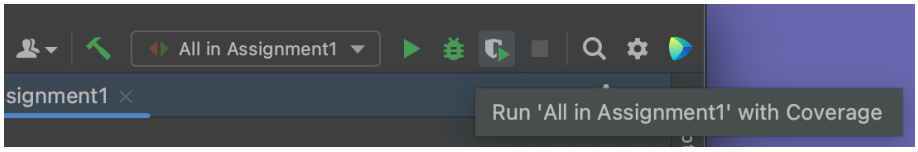
Edit the test execution configuration:



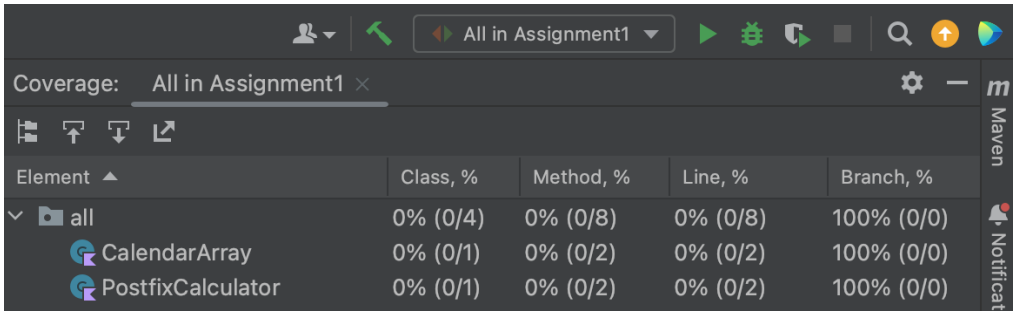
Select **Modify options** then **Use tracing** which will close the menu. Finally click **OK**.



Run all the tests again:



And confirm branch coverage is now shown:



Requirements

Postfix Calculator:

Details of postfix / reverse Polish Notation (RPN) calculations can be found in many places on-line, but if you are unfamiliar the notation, a good place to start is: https://en.wikipedia.org/wiki/Reverse_Polish_notation

Your postfix calculator is required to support the following operators when `PostfixCalculator.parse()` is invoked by your tests:

+	addition
-	subtraction
*	multiplication
/	division
^	power

For example, after executing the following Kotlin statements:

```
val result = PostfixCalculator().parse("5 6 2 ^ 2 - *")
```

The variable `result` should have the value `170.0`.

Whilst you are welcome to implement additional operators, they will not be tested by the automated grading system. Take care not to waste time implementing operators that will gain you no credit.

Calendar Array:

Modify `CalendarArry.generate()` so it returns a populated 6x7 two-dimensional array of integers representing the days that would be displayed in a UI date selection component.

For example, after executing the following Kotlin statements:

```
val days = CalendarArray().generate(YearMonth.of(2023, 4))
```

The variable `days` should have the following content:

```
26 27 28 29 30 31  1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30  1  2  3  4  5  6
```

The numbers before and after those shown here in **bold** being the last few days of the previous month and the first few days of the next month.

Code Coverage:

Your Postfix Calculator and Calendar Array implementations should exhibit 100% class, method, line, and branch coverage when *your* tests are executed.

Note that the intent is for you to adopt a test-driven approach such that no additional work is required to satisfy this requirement.

What steps should I take to tackle this?

Review the lecture handouts and recordings, study the sample code for the Infix Calculator, ask question on Slack and consult any on-line resources you find useful. If you get stuck, come along to section and office hours to ask questions and get some help.

How much code will I need to write?

A model solution that satisfies all requirements adds approximately 50 lines of code to the implementation classes and and around 100 lines to the test classes.

Grading scheme

The following aspects will be assessed:

1. (60%) **Does it work?**

    a. Postfix Calculator (30%)
    b. Calendar Array (30%)

2. (40%) **Is it well written?**

    a. Code Coverage (40%)

3. (-100%) **Did you give credit where credit is due?**

    a. Your submission is found to contain code segments copied from on-line resources or created by code generation tools and you failed to give clear and unambiguous credit to the original author(s) in your source code You will also be subject to the university academic misconduct procedure as stated in the class academic integrity policy. (-100%).

    b. Your submission is determined to be a copy of a past or present student's submission. (-100%)

    c. Your submission is found to contain code segments copied from on-line resources or created by code generation tools that you did give a clear an unambiguous credit to in your source code, but the copied code constitutes too significant a percentage of your submission:

        - < 25% copied code No deduction
        - 25% to 50% copied code (-50%)
        - > 50% copied code (-100%)

What to submit

On the console (PowerShell on Windows), navigate to the folder you extracted the starter code into and run the appropriate command to create the submission archive:

Windows:

```
$ Compress-Archive -Path src -DestinationPath Assignment1.Submission.zip
```

Linux:

```
$ zip -r Assignment1.Submission.zip src
```

Mac:

```
$ zip -x "*.DS_Store" -r Assignment1.Submission.zip src
```

**** UPLOAD** `Assignment1.Submission.zip` **TO THE CANVAS ASSIGNMENT AND SUBMIT ****

