COMPSCI 220 Programming Methodology

Project Assignment 01: Setup and Basic Scala

Overview

This project assignment is an introductory assignment on Scala programming in the IntelliJ IDE. You are provided a starter project where you will complete several Scala functions that exercise your understanding of basic Scala programming. These functions are based on concepts introduced in the first four chapters of *Scala for the Impatient*. You should reference these chapters as you work through each of these functions. You will also learn how to use the *activator* build system, the *grading assistant*, and how to run tests.

Learning Objectives

- To learn and exercise Scala variables and values.
- To learn and exercise Scala's basic types.
- To learn and exercise Scala control flow.
- To learn and exercise Scala for comprehensions.
- To learn and exercise basic Scala function declarations.
- To learn basic usage of the IntelliJ IDE.
- To learn how to compile using *activator*.
- To learn how to run tests using *activator*.
- To learn how to run the *grading assistant* to generate a report and package your submission.

General Information

Read this entire document. If, after a careful reading, something seems ambiguous or unclear to you, then communicate to the course staff immediately. Start this assignment as soon as possible. Do not wait until 5pm the night before the assignment is due to tell us you don't understand something, as our ability to help you will be minimal.

Reminder: Copying partial or whole solutions, obtained from other students or elsewhere, is academic dishonesty. Do not share your code with your classmates, and do not use your classmates' code. If you are confused about what constitutes academic dishonesty you should re-read the course syllabus and policies. We assume you have read the course information in detail and by submitting this assignment you have provided your virtual signature in agreement with these policies.

You are responsible for submitting project assignments that compile and are configured correctly. If your project submission does not follow these policies exactly you may receive a grade of zero for this assignment.

Policies

- For many assignments, it will be useful for you to write additional Scala files. Any Scala file you write that is used by your solution **MUST** be in the provided *`src/main`* directory you submit to Moodle.
- The course staff are here to help you figure out errors (not solve them for you), but we won't do so for you after you submit your solution. When you submit your solution, be sure to remove all compilation errors from your project. Any compilation errors in your project will cause the auto-grader to fail your assignment, and you will receive a zero for your submission. **No Exceptions**!

Test Files

In the *`src/test/scala`* directory, we provide several ScalaTest test suites that will help you keep on track while completing the assignment. We recommend you run the tests often and use them to help create a checklist of things to do next. *But, you should be aware that we deliberately do not provide you the full test suite we use when grading*.

We recommend that you think about possible cases and add new test cases to these files as part of your programming discipline. Simple tests to add will consider questions such as:

- Do your methods handle edge cases such as integer arguments that may be positive, negative, or zero? Many methods only accept arguments that are in a particular range.
- Does your code handle unusual cases, such as empty or maximally-sized data structures?

More complex tests will be assignment-specific. To build good test cases, think about ways to exercise functions and methods. Work out the correct result for a call of a method with a given set of parameters by hand, then add it as a test case. **Note that we will not be looking at your test cases** (unless otherwise specified by the assignment documentation), they are just for your use and will be removed by the auto-grader during the evaluation process.

You should not modify any source files in the *`src/test/scala`* directory as the auto-grader will be copying in the original public unit tests and additional private unit tests.

Before submitting, **make sure that your program compiles** with and passes all of the original tests. If you have errors in these files, it means the structure of the files found in the *`src`* directory have been altered in a way that will cause your submission to lose some (or all) points.

Import Project into IntelliJ

Begin by downloading the starter project, extracting it from the zip file, and importing it into IntelliJ. To import your project into IntelliJ you must run IntelliJ. If the IDE opens a previously created project, simply close the IDE window (do not close the IDE). It will then bring up a prompt with the following options:

- Create New Project
- Import Project
- Open
- Check out from version control

You should select **Import Project**. You will then need to find the downloaded project in the file menu and select **OK**. You should then select **Import project from external model** and then highlight **SBT**.

On the next screen make sure *Use auto-import* is selected and then click *Finish*. IntelliJ will then initialize your project (give it a minute) and then show you your project structure on the left.

The imported project may have some errors, but these should not prevent you from getting started. Specifically, we may provide unit tests for source files (e.g., classes, methods, functions) that do not yet exist in your code. You can still run the other unit tests.

The project should normally contain the following root items:

- ***`src/main/scala`***: This is the source folder where all code you are submitting must go. You can change anything you want in this folder (unless otherwise specified in the problem description and in the code we provide), you can add new files, etc.

- ***src/instructor/scala***: This folder contains support code that we encourage you to use (and must be used to pass certain tests). **You must not change or add anything in this folder**. The auto-grader will replace your submitted *src/instructor* directory with the original during the evaluation process. If you make changes or add/remove anything it can lead to problems in your submission and will result in a 0 for the assignment.
- ***src/test/scala***: The test folder where all of the public unit tests are available.
- ***tools/grading-assistant.jar***: This is the grading assistant that you can run to give you an estimate of your score as well as package your project to be submitted to Moodle. More on this later. **Do not remove**.
- ***activator***: This is a Unix helper script that can be used to run the activator interactive build tool. You can use this on Linux and Mac OSX. **Do not remove**.
- ***activator.bat***: This is a Windows helper script that can be used to run the activator interactive build tool. You can use this on Windows. **Do not remove**.
- ***activator-launch-1.3.2.jar***: This is a Scala/Java library that runs the activator tool and is used by the previously mentioned scripts. **Do not remove**.
- ***build.sbt***: This is the build definition file. It provides build information to activator to build your project. **Do not remove**.
- ***.gitignore***: This is a special file used by *git* source control to ignore certain files. You should not touch this file and please **do not remove**. This file may not be present if the assignment does not call for git usage.

Testing, Grading Assistant, and Console

As mentioned previously, you are provided a set of unit tests that will test various aspects of your implementation. You should get in the habit of running the tests frequently to see how you are doing and to understand where you might be going wrong. The ScalaTest testing framework is built-in to the *activator* tool and you can easily run the tests by issuing the following command from the command line:

```
> ./activator test
```

This will compile your code and run the public ScalaTest unit tests. After you compile and run the tests you will notice that a *target* directory has been created. The *target* directory contains the generated class files from your source code as well as information and results of the tests. *Activator* uses this directory so it must not be removed. After you run the tests you can get a grade report from the Jeeves tool by issuing this command:

```
> scala -cp tools/grading-assistant.jar autograder.jeeves.Jeeves
```

This will print a report to the console showing you the tests you passed, tests you failed, and a score based on the public tests. Although this gives you a good estimate as to what your final score might look like, it does not include points from our private tests. You may run Jeeves as often as you like, however, you must run the tests before your run Jeeves to give you an updated result.

Another very useful approach to test and play with the code you write is the console. You can run the console with this command:

```
> ./activator console
```

This will load up the Scala REPL (read-eval-print-loop). You can type code directly into the console and have it executed. If you want to cut and paste a larger segment of code (e.g., function declaration) you simply type *:paste* in the console, then paste in your code, then type control-D.

Part 1 – Implementing Functions

After you download, extract, and import your project into IntelliJ you will be able to navigate the project structure. This assignment comes with a single source file for you to modify:

- ***src/main/scala/basics/ScalaBasics.scala***: this file contains a single Scala object declaration with stubs for functions that you will need to implement.

Begin by opening this file in the IDE. Take a look at each of the function stubs defined. You will notice that after each functions there is an empty body designated by ***`???`***. This allows the Scala compiler to compile your project, but indicates that the function is not implemented. Indeed, if you were to execute these functions it will result in thrown exceptions telling you that the function was not implemented. Your job will be to replace each ***`???`*** by your implementation of the function.

You will also notice that Scaladoc documentation precedes each function declaration. We will cover Scaladoc at a later time, however, you will be able to read the documentation easily and each gives you constraints and hints on how to implement the function. The following is a summary of the functions you must implement.

- ***`ScalaBasics.add`***: You are to implement a simple add function that returns the sum of the two integer parameters.

- ***`ScalaBasics.inRange`***: Here you must implement a function that returns a *`Range`* type starting at *`start`* and ending at *`end`*. The documentation provides a hint on where you should look to find the answer to this.

- ***`ScalaBasics.oddRange`***: This function returns a *`Range`* type that returns *n* odd integers starting at 1.

- ***`ScalaBasics.minWhile`***: You must implement a function that returns the minimum integer in the given non-empty *`Array`* parameter. Your implementation must conform to the following constraints:

    - You must use a while loop.
    - You may use both immutable (val) and mutable (var) variables.
    - You must use an *`if`* expression.

- ***`ScalaBasics.minFor`***: You must implement a function that returns the minimum integer in the given non-empty *`Array`* parameter. Your implementation must conform to the following constraints:

    - You must use a for loop (not a for comprehension).
    - You may use both immutable (val) and mutable (var) variables.
    - You may not use an *`if`* expression. (hint: you should find an appropriate function in the Scala library to determine the *min*).

- ***`ScalaBasics.minRecursive`***: You must implement a function that returns the minimum integer in the given non-empty *`Array`* parameter. Your implementation must conform to the following constraints:

    - You may not use any loops.
    - You may not use any immutable (val) or mutable (var) variables.
    - You are allowed to use an if expression.
    - Your implementation must be recursive.

    - You may not use any library calls.

- ***`ScalaBasics.base36`***: In the real world it is often useful to be able to generate unique folder and file names. An easy way to do this is to use a *`BigInt`* and convert the *`BigInt`* into base 36 (which uses valid characters for folder and file names). You must implement a function that returns a base 36 string representation of the given *`BigInt`* value.

- ***`ScalaBasics.splitInHalf`***: You must implement a function that takes a string as input and returns a 2-tuple (pair) where the first element in the pair is the first half of the given string and the second element is the second half of the string. If the string has an odd number of characters it the first element will be the shorter half. Your implementation must conform to the following constraints:

    - You may not use any loops.
    - You may not use recursion.
    - You may not use any immutable (val) or mutable (var) variables.

- ***`ScalaBasics.isPalindrome`***: You must implement a function that returns *true* if the given string parameter is a palindrome[1] and *false* otherwise. You should *normalize* the string before you determine if it is a palindrome. That is, you must remove the characters '.' (period), '?' (question mark), ',' (comma), ';' (semi-colon), '-' (dash), ' ' (space), and ''' (single quote) to eliminate punctuation. Your implementation must conform to the following constraints:

    - You must use a for comprehension.
    - You may not use any other loops.
    - You may not use any mutable (var) variables.

- ***`ScalaBasics.sumChars`***: You must implement a function that sums the integer ASCII code of the given characters. This function uses a variable number of character parameters. That is, the function should accept any number of characters as arguments. Your implementation must conform to the following constraints:

    - You must use a for loop.
    - You may use mutable (var) variables.

- ***`ScalaBasics.wordCounter`***: You must implement a function that counts the number of space delimited words in the provided array of strings. This function takes an array of strings that represent lines in a text file. It returns a *`Map`* from *`String`* to *`Int`* where the *`String`* is a word found across all lines and the *`Int`* is the number of times that word was seen (see function documentation for an example). You may assume that words are delimited by spaces and you need not worry about punctuation. You can implement this however you wish. We do not provide a test for this function, however, you are welcome to implement your own by mimicking the structure of the provided public tests. We will be covering ScalaTest in more detail at a later time.

As you implement these functions you should run the tests frequently as described above. You should also take advantage of the REPL (read-eval-print) loop and test out your functions by cut and pasting them from your source code into the console to test. If you prefer not to cut and paste you can also import the functions using the following import from the console:

```
scala> import basics.ScalaBasics._
```

[1] https://goo.gl/Y2vhu

and then proceed by calling the individual functions:

```
scala> add(4, 5)
```

Submission

When you have completed the changes to your code, you must run the following command to package up your project for submission:

```
> scala -cp tools/grading-assistant.jar submission.tools.PrepareSubmission
```

This will package up your submission into a zip file called `submission-DATE.zip`, where `DATE` is the date and time you packaged your project for submission. After you do this, log into Moodle and submit the generated zip file.

After you submit the file to Moodle you should ensure that Moodle has received your submission by going to the activity page for the assignment and verifying that it has indeed been uploaded to Moodle. To further ensure that you have uploaded the correct zip file you should download your submission from Moodle and verify that the contents are what you expect.

We do not allow re-submissions after the assignment due date **even** if you failed to submit the proper file or forgot. There are no exceptions so be very sure that you have submitted properly.