TDD Practical

Objectives

The objectives of this practical session are to:

- Use IntelliJ to develop using TDD and source control

Overview

In this practical you will work on an application using JUnit and TDD techniques.

Practical

TDD with Pair Programming 

1. This is a TDD practical, but your instructor might assign you to work on this as a pair, programming on one machine, or if you have an online Git account, with a shared repository.
2. Open IntelliJ, and close any existing projects. That results in the splash screen.

   Choose ‘Check out from Version Control’ and select ‘Git’.
3. Now choose the remote Git repository URL:

TDD Practical

Objectives

The objectives of this practical session are to:

- Use IntelliJ to develop using TDD and source control

Overview

In this practical you will work on an application using JUnit and TDD techniques.

Practical

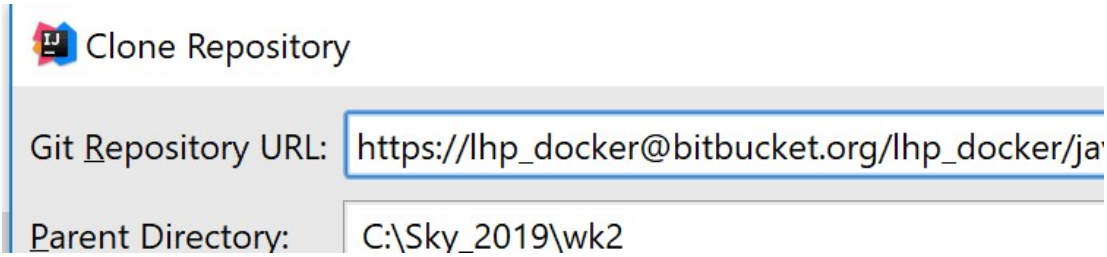
TDD with Pair Programming

1. This is a TDD practical, but your instructor might assign you to work on this as a pair, programming on one machine, or if you have an online Git account, with a shared repository.
2. Open IntelliJ, and close any existing projects. That results in the splash screen.

   Choose 'Check out from Version Control' and select 'Git'.

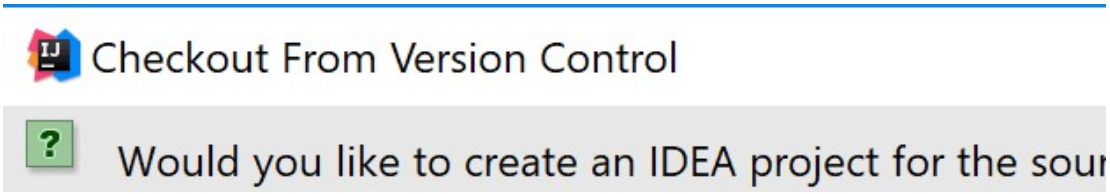
3. Now choose the remote Git repository URL:

   `https://lhp_docker@bitbucket.org/lhp_docker/java_tdd.git`

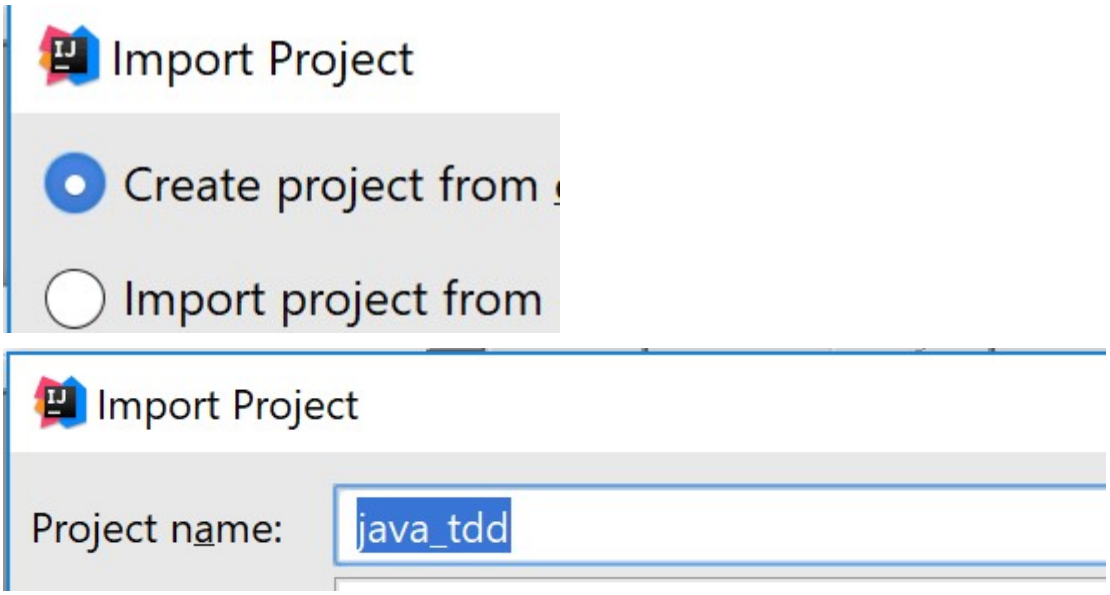


Invent your own 'Parent Directory' and 'Directory Name', but you must create the parent directory yourself. On a Mac, you can use a folder under your home. Click on **Clone**.

4. On the next screen click **Yes**.

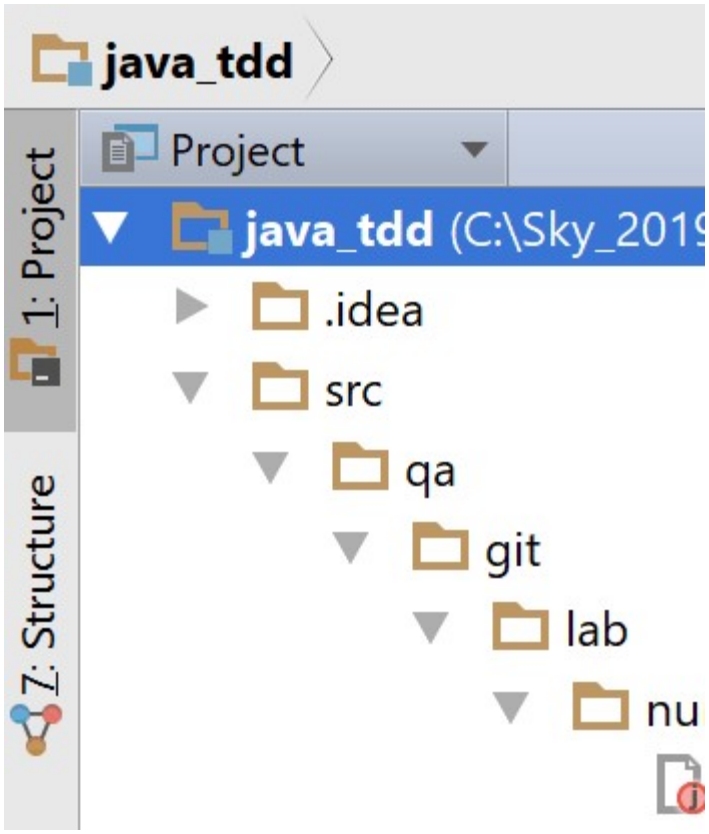


5. For the next few screens, you can just accept the defaults,



Then click Next, until the end… (These screens would configure any libraries you need. There are none so far.)

6. Ultimately you get to **Finish**. You then get a project such as this:

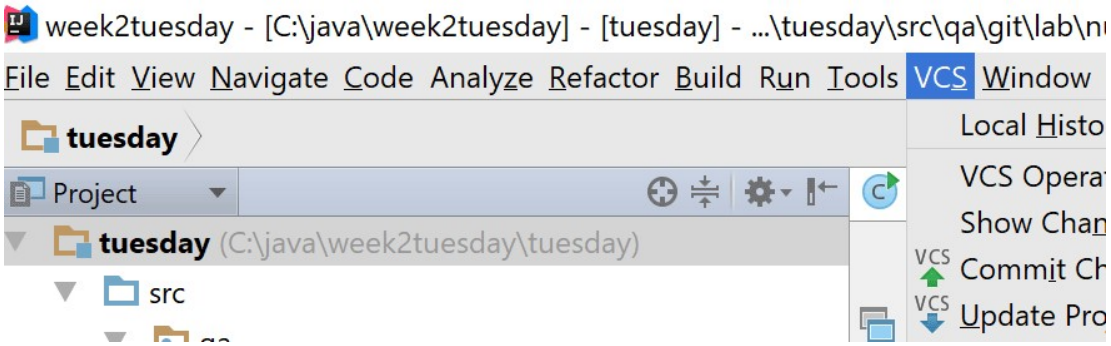


7. The `Numbers` class is similar to one you may have worked on earlier. Run the code just to see. If you modify some code, you can commit to your local repository, but it won't let you 'push' without a password! If you want, you can set up your own remote repository online if you prefer to work on multiple machines.

8. Examine the code in the `qa.tennis` package. The idea is to write an application to return tennis scores, which are not straightforward (some would say stupid...). The inputs are a series of strings of the form ABAABAB etc. where 'A' and 'B' refer to each player, winning a point.

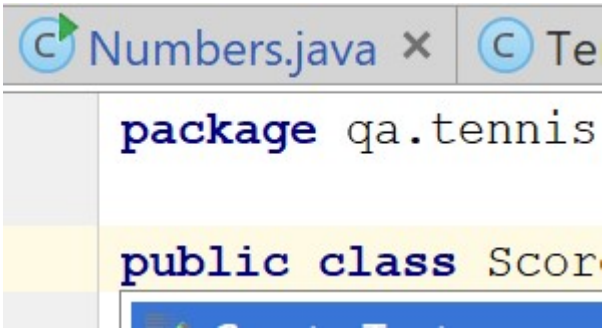
9. There is a command-line based application `ConsoleScore`, which either takes a single command-line parameter for the input string, or else accepts command-line input. There is also a graphical application `GraphicalScore`, which has two buttons. Run the graphical application, to get a feel for the task at hand.

10. Use the top menu **VCS > Git > Add**, to add all your classes to Git for staging.

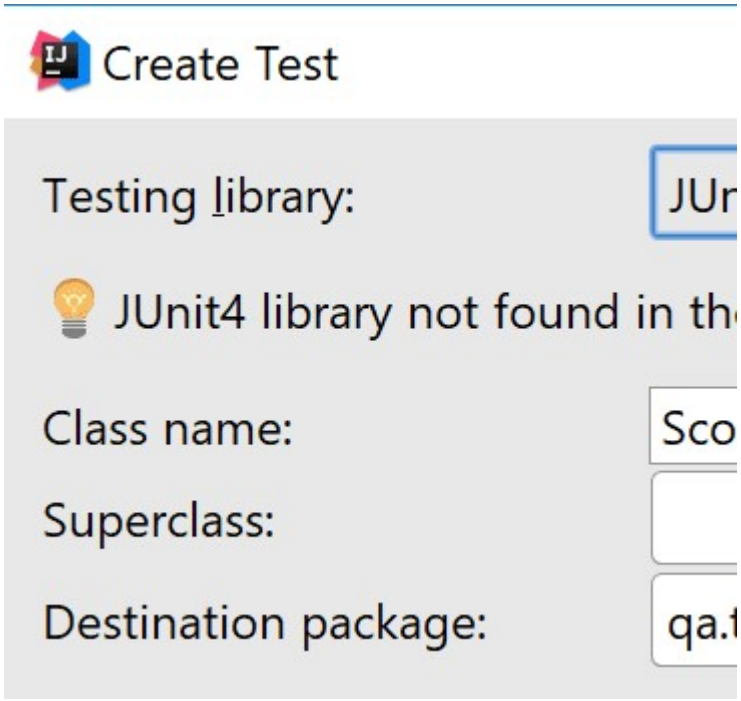


After making changes, use **VCS > Git > Commit Changes**, providing a commit message of course! You can also do this via right-clicks on your project files.

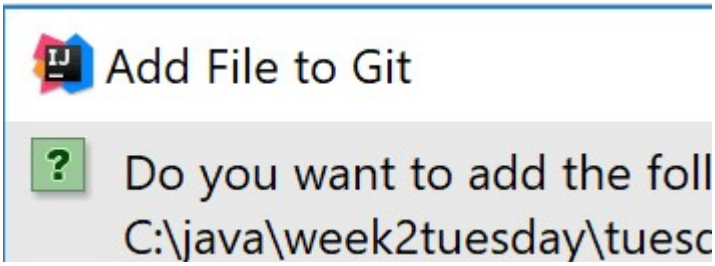
11. Open the class `Score`, and move your cursor to the beginning of the class declaration (`public class Score…`). Press **ALT + ENTER**, to get the Create Test dialog.



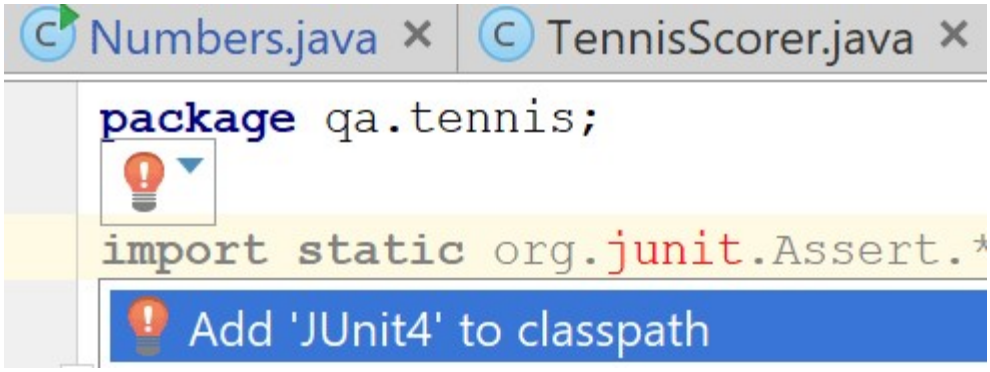
Make sure you change the Testing framework to be JUnit4 (or JUnit5).



Click **OK**, and choose to add the new class to Git.



12. In your resulting test case class, you will notice a red light-bulb, which you can use to add JUnit4 libraries to your classpath.



Use IntelliJ's own distribution.

13. In the `Score` class, locate the code in the constructor which generates random scores, and remove the two lines!
14. In your test class, create a simple test, using the `@Test` annotation:

```
@Test
public void testNewTennisScorerReturns0_0() {
    TennisScorer ts = new TennisScorer();
    assertEquals("0:0", ts.currentScore());
}
```

15. Run the test, it passes! Actually, this is not good TDD, but that was because we have provided some starter code. Normally, you'd have nothing...

16. Recall the steps of TDD:
    1. Write a test that corresponds to a requirement
    2. Run the test and see it fail
    3. Complete the code to make the test pass
    4. Run the tests to ensure they all pass
    5. Refactor your code (and your test code)
    6. Run the tests to ensure they still all pass
    7. Commit
17. Now using TDD with a Git repository, complete the application. You will each be asked to start in one developer role, but you may swap over. If your Git repository is local, you will need to use one computer only. If you managed to set up a shared repository (using your own Github or Bitbucket account, for example) then you can 'push' and 'pull' working on two computers!
18. The algorithm for tennis has a few complications, as expressed by the following rules:
    - A match is the best of 5 sets.

- Each set is the first player to 6 games, if leading by 2. If a set reaches 6-5, a player can win 7-5 or else it reaches 6-6 in which case a tiebreak is played.
- A tiebreak is first to 7 points or more, leading by 2.
- Each regular game (i.e. not tiebreak) scores for each player go up as follows: 15, 30, 40 then a game is won, but if the score reaches 40-40, this is known as "Deuce" and then players play until one player is clear by 2 points. Each score after deuce results in "Advantage" to one of the players, which then goes back to "Deuce" is that player loses the next point, and so on.
- Some tournaments do not play a tiebreak in the 5th set, the play simply continues until one player has a 2-game lead – you can choose whether you implement this.

19. In terms of console printing, the score should be formatted with a hyphen `-` between the game scores, and a colon `:` between the points scores. Spaces separate the sets. A tiebreak is denoted **`7-6`**, deuce is shown as **`40:40`** and advantage is shown **`A:40`**. For example:

    ```
    6-4 3-6 7-6 0-1 15:30
    ```

20. The GUI application uses its own representation, so there is no need to format strings in this case. Examine the code to see how this works.
21. Suggested breakdown (micro-sprints):
    - Performing validation, to ensure only 'A' and 'B' characters are processed.
    - Coding the main `TennisScorer` algorithm for individual games
    - Coding the `TennisScorer` algorithm for a set (consisting of games)
    - Coding the `TennisScorer` algorithm for tiebreaks

Each of these *could* be coded separately, and merged where appropriate back to a suitable branch of Git. It makes good sense to use several JUnit test classes. In your pair, it doesn't help to work on several things at once, so following the suggested breakdown on the master branch is fine.