IntelliJ Dojo Lab Book

Daniel Hinojosa

Table of Contents

Lab 1: Tool Windows without your mouse

1. Open the **Project** tool, Close the **Project** Tool
2. Open the **Run** tool, keep it open. Go to the Editor, remember `ESC`!
3. Maximize the editor (`⌘`+`⇧`+`F12` / `CTRL`+`SHIFT`+`F12`)
4. Run `mvn compile` quickly, both by going to the Maven tool window `⌘`+`E` / `CTRL`+`E`/ , select the **Maven** tool, and select **compile**
5. Close all tool windows
6. Close all the tabs through either `Shift`+`Shift` or (`⌘`+`⇧`+`A` / `CTRL`+`SHIFT`+`A`)
7. Run `mvn test` using `Run Anything` using `Ctrl`+`Ctrl`

Lab 2: Find your file without your mouse

1. Use "Find Action" (`⌘`+`⇧`+`A` / `CTRL`+`SHIFT`+`A`) to run "Close All Tabs"
2. Locate files quickly, and bring them up quickly, ready for you to edit using (`⌘`+`⇧`+`O` / `CTRL`+`N`)
3. Open `Deck` using (`⌘`+`⇧`+`O` / `CTRL`+`N`)
4. Use "Find Symbol" (`⌘`+`⌥`+`O` / `CTRL`+`ALT`+`SHIFT`+`N`) and locate `game.initialDeal`

Lab 3: Find the Errors

1. Go to `DeckTest` using the techniques we covered previously
2. Find all minor errors quickly with `F2`
3. Fix them using (`⌥` + `⏎` / `ALT` + `ENTER`)

Lab 4: Create the files quickly without your mouse and efficiently

1. Be sure to use (`⌘` + `N` / `ALT` + `INSERT`) for creation of elements.
2. Create an enum in `com/jitterted/ebp/blackjack` called `Currency` with the values `Dollar` , and `Euro` , for a challenge add the symbols `$` and `€`
3. Now we will create a Java class in the `com/jitterted/ebp/blackjack` called `Chip`
4. Create two fields, one `amount` of type `int` , and `currency` of type `Currency` , the `enum` that you created
5. Add a `toString` , `equals` , and `hashCode` for `Chip` using (`⌘` + `N` / `ALT` + `INSERT`)

Lab 5: Using the blob!

1. Find the file `Game.java` using the techniques that we covered
2. Go to the method `main` . Hint: You can quickly search methods by either (`⌘` + `F12` / `CTRL` + `F12`)
3. Highlight only the text `System.console().readLine()` using increase code selection `⌥` + `↑` / `CTRL` + `W`)
4. Now use surround block on the selection (`⌘` + `⌥` + `T` / `CTRL` + `ALT` + `T`), and wrap it in a `try` / `catch` / `finally` block that will catch an `IOError` and print the `message` of the exception

Lab 6: Go to Implementations and Usages

1. Go to `Deck` and then go to the `size` method using the techniques that you've learned
2. Which other methods are using this method? (`⌘` + `B` , `⌥` + `F7` / `CTRL` + `B` , `CTRL` + `F7`)
3. Jump to one of the methods that uses `size`
4. Go to `Game` and then go the constructor of `Game` , from here go to the implementation of `Deck`

Lab 7: Moving Code

1. Make `main` the last method using (`⇧` + `⌘` + `↑↓` / `CTRL` + `SHIFT` + `UP/DOWN`)
2. Rearrange the other methods to your liking

Lab 8: Documentation and Info

1. Go to Maven Tool Window and "Download Sources and/or Documentation", you'll need to use the mouse on this one
2. Dismiss the Maven Tool Window
3. Using "Find Symbol" go to `displayFinalGameState` . What is the type of `dealerHand` ?

4. Go to `displayHand` using whatever maneuver you like, put your cursor over `joining`, what is the documentation for this method?
5. What is the type of `Collectors.joining( ansi().cursorUp(6).cursorRight(1).toString())`

Lab 9: Efficient Running and Testing

1. Run the `main` method in `Game` using the keyboard
2. Open `CardTest` and run `withValueOfQueenHasNumericValueOf10` only
3. Go to the `Card` class
4. Run `withValueOfQueenHasNumericValueOf10` but don't go back to the `CardTest`!
5. Open `CardTest` and run `withValueOfQueenHasNumericValueOf10` and `withNumberCardHasNumericValueOfTheNumber` only
6. Open `Game`, use recent files to go back to the game
7. Run the same two tests we just did without going back to `CardTest`
8. Run all the tests in `CardTest`
9. Run all the tests in the `com.jitterted.ebp.blackjack`

Lab 10: Toggling your Test

1. Go to `Card`
2. Go back to `CardTest` using the toggle, (`⌘` + `SHIFT` + `T` / `CTRL` + `SHIFT` + `T`)
3. Go back to `Card` using the toggle, (`⌘` + `SHIFT` + `T` / `CTRL` + `SHIFT` + `T`)
4. Create a test called `WalletTest`, remember (`⌘` + `N` / `ALT` + `INSERT`)?
5. Create a test method called `walletHasZeroBalance` method (`⌘` + `N` / `ALT` + `INSERT`). In the test, implement a test that creates a `Wallet` that returns a `balance`. Assert that the `balance` is `0`.
6. Use (`⌥` + `⏎` / `ALT` + `ENTER`) to create a `Wallet`.

Lab 11: Debugging

1. Go to `HandValueAceTest` and the `handWithOneAce` method using the techniques that we coverd
2. Put a breakpoint on the `assertThat(game.handValueOf(cards)).isEqualTo(11 + 5);` (`⌘` + `F8` / `CTRL` + `F8`)
3. Debug the method (`^` + `⇧` + `D` / `CTRL` + `SHIFT` + `F9`), what are the contents of cards?
4. Stop the debugging session
5. Remove the breakpoint

Lab 12: Refactoring

1. Go to initial deal method in main
2. Highlight the lines the first round of cards into a method call dealRound in the method using the blob
3. Make it so that it is called twice
4. Extract another method in the first part of the main method from `AnsiConsole.systemInstall()` to `Hit Enter to Start` and create a method called display welcome screen

Lab 13: Hey Emmett!

1. Open the Emmett cheatsheet: https://docs.emmet.io/cheat-sheet/
2. Create a resources folder in `src/main` and add a file called *index.html*, if one is not already created.
3. Create an HTML template using `html:5`!
4. Create a `<p>` with the content "Favorite Food" follow by an unordered list with `5` `li` items in whatever format you like listing some of your favorite foods
5. Try different combinations
6. Find other creative Emmet combinations using the cheat sheet
7. Try your hand at styles by creating *styles.css*

Lab 14: Live Templates

1. Create a Live template for yourself
2. Are there some classes that you create? Perhaps a standard way to do log files?
3. How about a fixed thread pool? `ExecutorService ec = Executors.newFixedThreadPool(10);`
4. How about using `???` and create `throw new UnsupportedOperationException("Not Implemented");` for always failing initial tests in TDD?
5. If you like to use AssertJ, perhaps for `assertThat` or `assertThatThrownBy`?

Credit to Ted Young for this wonderful project, https://github.com/tedyoung and https://moretestable.com