Assignment 1 (AcesUp Game)

Introduction

For Sprint 1, your team is tasked with creating a proof of concept demo of a card game called Aces Up using the Ninja framework (http://www.ninjaframework.org/).

The rules for Aces Up are here: Note: Aces are high.

- Deal four cards in a row face up.
- If there are two or more cards of the same suit, discard all but the highest-ranked card of that suit.
- Repeat step 2 until there are no more pairs of cards with the same suit.
- Whenever there are any empty spaces, you may choose the top card of another pile to be put into the empty space. After you do this, go to Step 2.
- When there are no more cards to move or remove, deal out the next four cards from the deck face-up onto each pile.
- Repeat Step 2, using only the visible (top) cards on each of the four piles.
- When the last four cards have been dealt out and any moves made, the game is over. The fewer cards left in the tableau, the better. To win is to have only the four aces left.

When the game ends, the number of discarded cards is your score. The maximum score (and thus the score necessary to win) is 48, which means all cards have been discarded except for the four aces, thus the name of the game.

The rules of the game are from here: https://en.wikipedia.org/wiki/Aces_Up
You can also play online here: http://justsolitaire.com/Aces_Up_Solitaire/

Your boss wants to know if it is possible to implement this game using the ninja framework. You will have two weeks to implement this game. Since this is just a proof of concept, we will not be developing an extensive UI at this time.

Tasks for this Assignment

To complete this assignment, each group needs to do the following:

- **Clone https://github.com/nelsonni/AcesUp_Stage1 to local.** This will be the repo for your project. This repo will be the basis for your grade in this assignment.
- **Turn the assignment into User Stories.** From the Introduction Section, each team should turn the assignment into a series of user stories. We will talk about them in class, but a good introduction to User Stories can be found here: http://www.subcide.com/articles/how-to-write-meaningful-user-stories/. Once each team has developed their user stories, they will create a page on their Github Project wiki called "User

Stories", where they will record all their user stories.

Each User Story must have the following format:

```
Card: ("As a [user], I want [function], so that [value]")
Conversation: (Details that clarify the user story)
Confirmation: (Tests to know when the User Story is completed)
```

Your user stories will be evaluated on whether they follow INVEST.

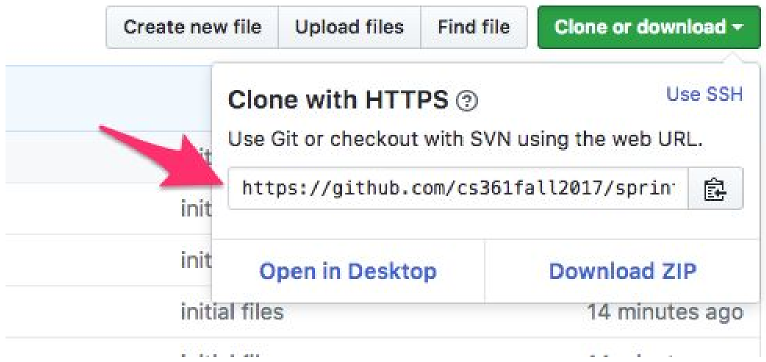
- **Create Tasks from User Stories.** From the user stories, each team will create a series of features that need to be implemented in the code. These will be added to the Github issue tracking system. Each of these must be assigned to a specific team member, as well as have a time estimate given in the body of the issue.
- **Implement Code in Feature Branches.** Each Team member will create a branch for each feature they are going to implement. When the feature is completed, the team member will create a pull request to the group repo. They should use that pull request to close the specific feature. If at any time, a bug is found, it should be reported using the Github issue tracker.

Configuring IntelliJ IDEA and Importing Maven-based Projects

When working with any of the project repositories for the first time, it is necessary to configure your environment and import the project properly before beginning development. The following steps assume that you will be using IntelliJ IDEA as your development environment, but other IDEs can be used as well (we won't be able to support other IDEs however, so select your IDE accordingly).

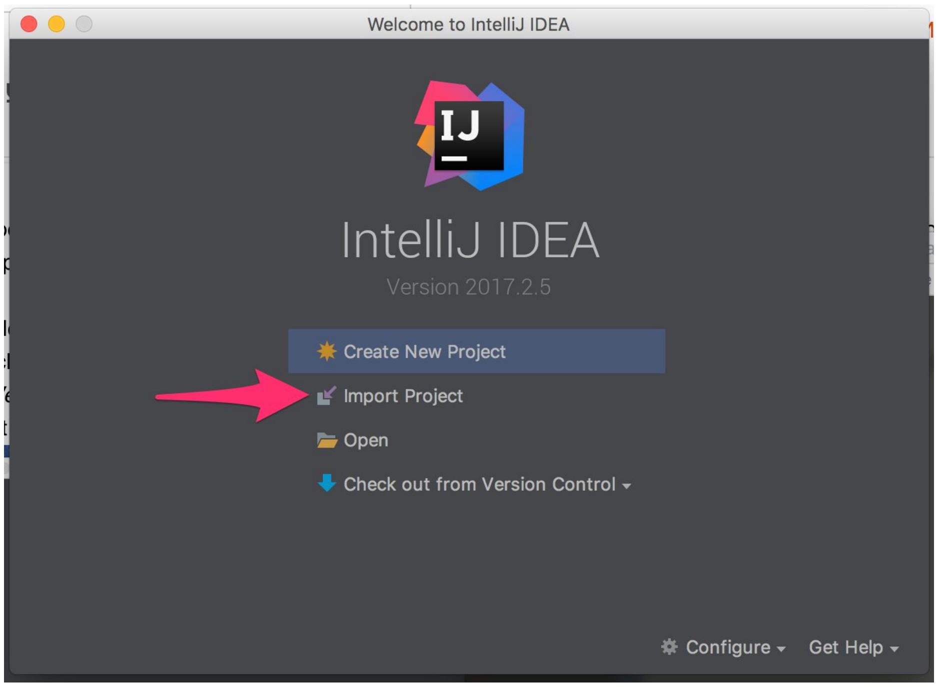
WARNING: If you use the *Open* option within the *Welcome to IntelliJ IDEA* window, you will be able to view the project files, but IntelliJ will not recognize those files as being part of a Maven project and will not import the Ninja framework. Please follow the instructions below to properly import the project.

1. Navigate to the repository page on GitHub (i.e. *https://github.com/user/repo*).
2. Select the **Clone or download** button and copy either the HTTPS or SSH URL.

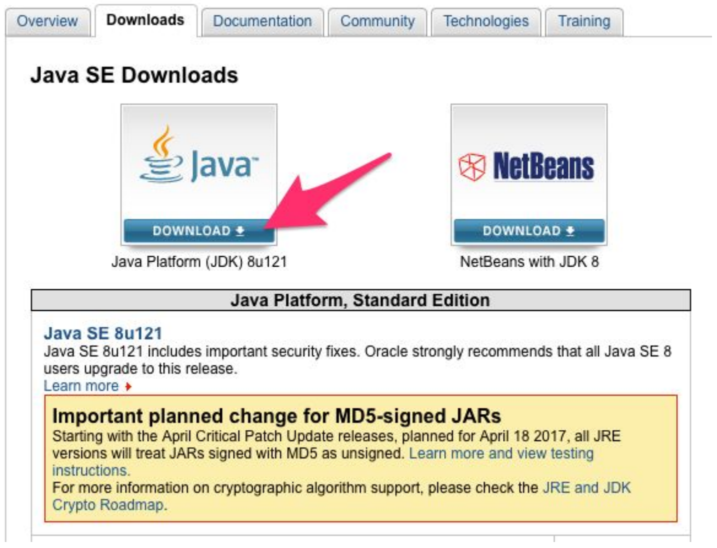


3. Clone the repository to your local system either via a Git UI client or through a terminal. The terminal method would involve launching a terminal window, navigating to the folder/directory that you wish clone the repository files into, and running the command: **git clone *url*** (where *url* should be the HTTPS or SSH URL).
4. Download and install IntelliJ IDEA (link (Links to an external site.)Links to an external site.).
5. Launch IntelliJ IDEA. The initial launch has configuration steps, complete these as needed, and get to the *Welcome to IntelliJ IDEA* screen.

6. Select the **Import Project** option.



7. From the file browser window, navigate to the repository that you cloned to your local system in steps 1-3, and select the **pom.xml** file within the main project folder. Select the **Open** button to continue.
8. If this is a clean installation of IntelliJ IDEA, you will need to add a Java SDK in order for IntelliJ IDEA to be able to compile Java source code. Typically when loading a new project in IntelliJ IDEA a popup dialog will appear indicating that the project does not have an SDK configured for it; this dialog will all provide options for adding and configuring an SDK for that project.
9. If you do not currently have a copy of the Java SDK installed on your system, you will first need to download and install the **Java Platform (JDK)** from Oracle (http://www.oracle.com/technetwork/java/javase/downloads/index.html).



Java SE 8u121

Java SE 8u121 includes important security fixes. Oracle strongly recommends that all Java SE 8 users upgrade to this release.

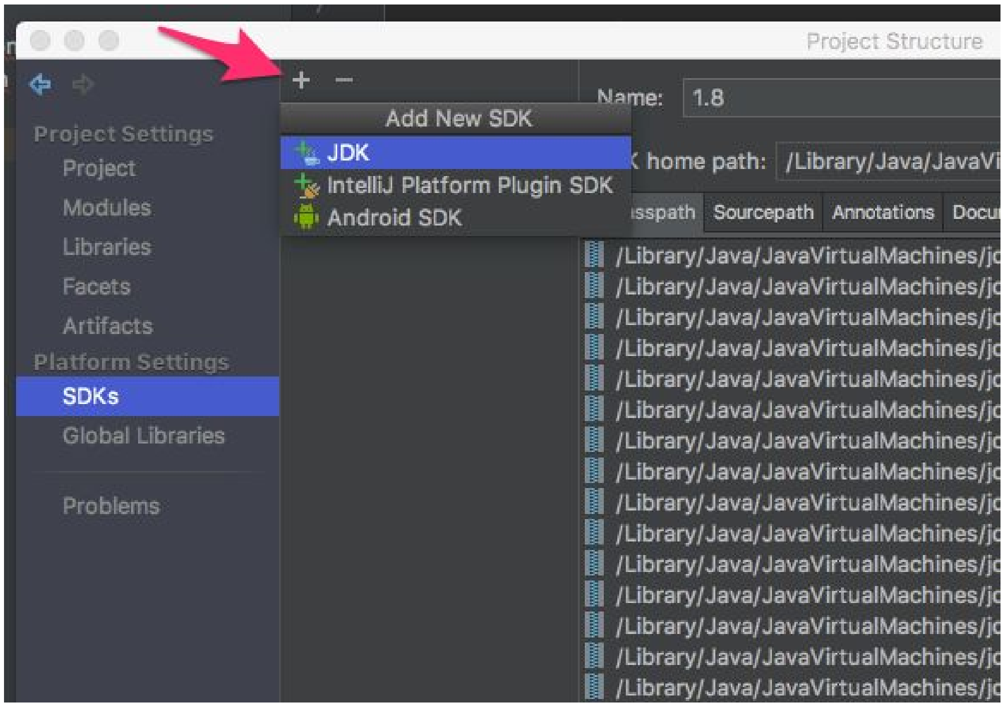
Learn more

Important planned change for MD5-signed JARs

Starting with the April Critical Patch Update releases, planned for April 18 2017, all JRE versions will treat JARs signed with MD5 as unsigned. Learn more and view testing instructions.

For more information on cryptographic algorithm support, please check the JRE and JDK Crypto Roadmap.

10. If the SDK configuration dialog did not appear, or you closed out of it, the appropriate menu for adding a Java SDK to IntelliJ IDEA can be found by going to **File** → **Project Structure...** and selecting the **SDK** option under the *Platform Settings* group on the left-hand side of the window. Select the *+* icon at the top of the middle pane, and the **JDK** option.



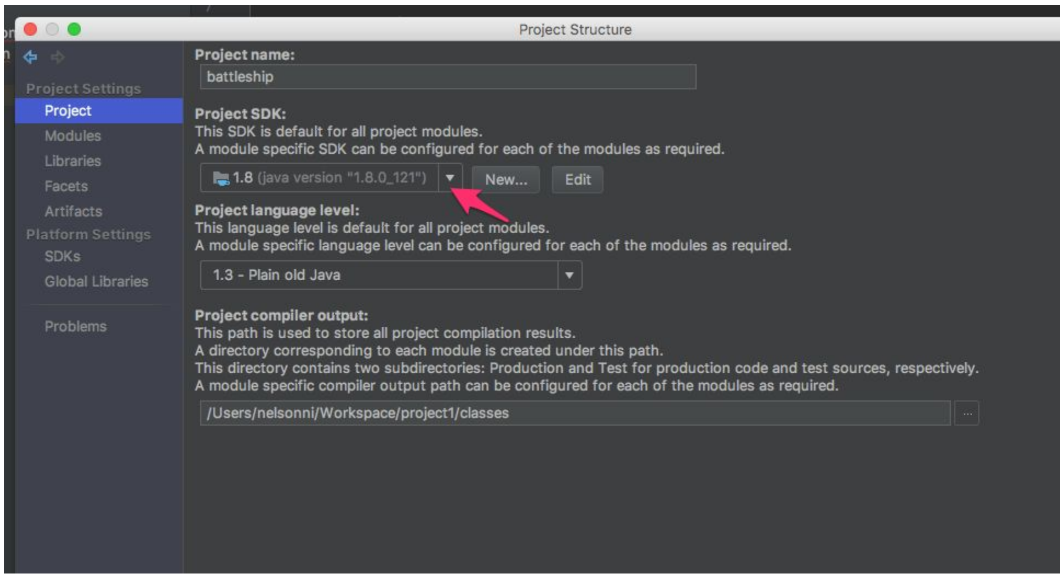
Name: 1.8

home path: /Library/Java/JavaVi

```
/Library/Java/JavaVirtualMachines/jc
/Library/Java/JavaVirtualMachines/jc
/Library/Java/JavaVirtualMachines/jc
/Library/Java/JavaVirtualMachines/jc
/Library/Java/JavaVirtualMachines/jc
/Library/Java/JavaVirtualMachines/jc
/Library/Java/JavaVirtualMachines/jc
/Library/Java/JavaVirtualMachines/jc
/Library/Java/JavaVirtualMachines/jc
/Library/Java/JavaVirtualMachines/jc
/Library/Java/JavaVirtualMachines/jc
/Library/Java/JavaVirtualMachines/jc
/Library/Java/JavaVirtualMachines/jc
/Library/Java/JavaVirtualMachines/jc
/Library/Java/JavaVirtualMachines/jc
```

11. The **JDK** option will launch a file browser window. Navigate to the location of the installed Java SDK and select the *jdk1.8.0_121.jdk* folder (version numbers and the folder name might be slightly different on your system) before selecting the **Open** button. Note: For macOS users, the Java SDK folder is typically located in */Library/Java/JavaVirtualMachines/*.
12. Once added, select the **Project** option under the *Project Settings* group on the left-hand side of the window. Select the drop-down menu under *Project SDK* in the main pane and a **1.8** or

similarly named instance of the Java SDK should be available; select it and select the **OK** button.



Project Structure

Project name: battleship

Project SDK:
This SDK is default for all project modules.
A module specific SDK can be configured for each of the modules as required.
1.8 (java version "1.8.0_121")

Project language level:
This language level is default for all project modules.
A module specific language level can be configured for each of the modules as required.
1.3 - Plain old Java

Project compiler output:
This path is used to store all project compilation results.
A directory corresponding to each module is created under this path.
This directory contains two subdirectories: Production and Test for production code and test sources, respectively.
A module specific compiler output path can be configured for each of the modules as required.
/Users/nelsonni/Workspace/project1/classes

13. Your project should now be properly configured for development and use from within IntelliJ IDEA.