# Java Object-Oriented Concepts Unit

## Agile Approach Checklist for Console Applications

SOFTWARE GUILD

The Learning House

427 S 4th Street #300

Louisville KY 40202

# Agile Approach Checklist for Console Applications

## Overview

This document outlines an approach that can be used to break down a lab and implement it in a stepwise and Agile way. This will allow you to deliver demonstrable, testable code to your user in small increments.

This approach specifically covers the following labs:

- Address Book (we will use this lab as an example)
- DVD Library
- Vending Machine
- Baseball League
- Flooring Mastery Project

## Assumptions

Our applications are mainly concerned with Creating, Reading, Updating, and Deleting (CRUD) data to and from persistent storage (files or databases).

1. Our programs (even the simple ones) must be split into tiers or layers as practice for more complicated problems. This leads us to create projects with the following features:
    a. Data Transfer Objects (DTO) also known as domain objects (e.g. Address)
    b. Data Access Object that handles storing data in memory and reading and writing data from persistent storage (e.g. AddressBook).
    c. Some kind of user interface (e.g. the console).
    d. Logic that orchestrates the objects in the program (this will be found in the Controller).

## Requirement Steps

### Step 0

Create user stories from the problem statement/requirements. For Address Book, it would be:

- Add an address
- View an address
- Delete an address
- List all addresses
- Show the number of addresses in the address book
- Save to file
- Read from file

## Design Steps

### Step 1

Analyze problem statement and identify required classes. For Address Book, we have the following:

- Address (DTO)
- AddressBook (Data Access Object)
- ConsoleIO (helper class to interact with the console, from your library jar)
- AddressBookController (this class orchestrates the program)
- AddressBookRunner (this class has a main method that instantiates AddressBookController and calls the execute method)

### Step 2

Create UML Class Diagrams based on Step 1. Flesh out the classes by defining properties and methods for each.

### Step 3

Create a flowchart for the user interaction process (e.g. displaying menus and reacting to user menu choices).

### Step 4

Create file format to match domain object(s) (e.g. Address). For example, for storing address, we might have something like this:

```
<firstName>::<lastName>::<streetAddress>::<city>::<state>::<zip>
```

## Construction Steps

### Step 5

Create a menu system (in the execute method of AddressBookController) with stubbed out code for each menu choice. For example, when the user presses the choice to add an address, the system simply prints a message saying “AddAddress: To Be Implemented.” This can be delivered to the user for testing and feedback.

### Step 6

Pick a user story to implement. For example: “Add address.” For this story, we’ll need to do the following (each User Story will be different — the first one is always the most work):

1. Create the Address class (domain object)
2. Create the AddressBook class
    a. Include a HashMap or ArrayList class level variable

b. Implement the addAddress(...) method. **Write a unit test for this method** (this will require that you also implement the getAddress(...) method).

3. Add code into the AddressBookController to:
    a. Instantiate AddressBook object for storing Address information
    b. Read in address information from user
    c. Create a new Address object
    d. Put address information from user into the Address object
    e. Add the new Address to the AddressBook

**Repeat step 6 for each user story.**