IT 2045C Computer Programming II  
Prof. Tom Wulf   
Fall 2024 v2.0

# Lab 01 Get Classy

**20 points**

## Learning Goals:

* Get practice creating classes with IntelliJ
* Instantiate object data from a csv text file
* Persist object data to a csv text file
* Create an object version of our SafeInput console library from CP I
* Take advantage of GitHub source control to Fork the Practicum project to reuse the code for this project.

## Functional Requirements:

* Create each of the classes below within a single IntelliJ project.
* Create a constructor for each class that takes all the fields
* Create additional overloaded constructors where it makes sense to do so
* Create getters for all fields.
* Create setters for fields where it makes sense to do so
* Create the override for the toString() method (We do this for every class!)
* For data classes (Product, Person, etc) create the override for equals()
* Create the additional specified class methods
  + For each data class create String toCSV(), String toJSON(), and String toXML() (JSON and XML are common data formats like CSV)
* For each class: create a Junit test class for your class
  + IntelliJ will create a stub of tests that you can edit
  + Use the expected pattern where the test will show the expected output
  + Test the constructor, all the setter/mutator methods as well as any specified methods you were told to implement.   
    You do not have to test the getter/read methods!
* For each class, provide the UML Class Diagram. Embed it below in this document where indicated. You can use MS Word, Draw.io, chart.io to create your diagrams. Note that the Ultimate version of IntelliJ will create these for you from your classes.
* Use the project and file names specified

## Mini-lecture:

Last week, we reviewed how to save and read data records to and from a CSV text file. This week we are looking at how to create objects in java, so we will create java classes that correspond to our data records, **Person** and **Product**. We will modify the programs from the previous lab to use our new object classes.

**SPECIAL NOTE:** a good deal of the code that you wrote for the Practicum can be reused here. So, in GitHub **fork,** or **pull and reuse** your practicum project to create a new copy of it for this Lab. You can also do this manually. (It is very easy to get the entire GitHub project as a zip, or to get the individual java files from the repo.

**Create an IntelliJ project called Lab\_01 and add it to a GitHub repo with that name. As you create the work for the lab, Commit and Push your work. All files will be added to the repo.**

## Part 1: Person:

Project: Person  
Files: Person.java   
 PersonReader.java // Reads Person records from a file into an ArrayList  
 PersonGenerator.java // Creates an ArrayList of Person objects and writes to a file  
 SafeInput.java // Library of console input routines  
 PersonTest.java // Junit test file for Person class

### Fields: (No change here from the Practicum)

String firstName  
String lastName  
String ID // should never change sequence of digits  
String title // a prefix: Mr. Mrs. Ms, Prof. Dr. Hon. Etc.  
int YOB // Year of birth // Range should be 1940 - 2010

### Additional methods (All should be tested in JUnit): /\*\* ALL METHODS SHOULD HAVE COMPLETE JAVADOC BLOCKS including @return, @param and the rest of the details for each method.

### \*/

public String fullName() // returns firstName, space, lastName  
public String formalName() // returns title, space, fullName

public String getAge() // returns the age assuming the current year  
public String getAge(int year) // uses YOB to calculate age for a specified year  
 // use the Calendar object to do these. Requires a bit of a web search.

Public String toCSV() // returns a comma separated value (csv) String suitable to writing to a java text file. Be sure to use this function when you save data to the file. You can use the for each loop to traverse the ArrayList and use this function to generate the CSV record to write.

When you read the data in from the CSV text file, instantiate a Person object from each record and store the Person objects within an ArrayList of type <Person>. Similarly, when in PersonGenerator as the user enters the data fields for each person, use it to create a Person Object and again store it in an ArrayList and then write all the records to the text file when they are finished.

Similarly, code public toJSON() and public toXML() methods.

Make sure that your program uses all the methods here.

**Special Note:** as I mentioned, you can fork the previous Practicum project and reuse 90% of the code. If you read the directions carefully you should be able to see that the only modification to the Practicum Reader and Generator is to add the ArrayList and use the new People or Product object. You already have code that both reads and writes the CSV format.   
  
Create the new class files first(i.e. Person for part 1 and Product for part 2)Create the JUnit test for the class at that time. IntelliJ will stub out the tests. You have to construct tests instances in setup to use in the tests and fill out the stubs. Test the constructors, all modifier/setter methods, and any additional special methods. You do not test the getter/read methods.  
  
**Paste a single screen shot here of your IntelliJ JUnit tests. It should show that all tests passed successfully.**  
  
Then modify the java main code that you forked. That is, make sure you fork the code before you change the code in main. So you will still have a working version of the last lab and a new fork which is this lab. You can of course just copy the code manually into the new project .

In the Generator, as soon as the user inputs the field data, instantiate an object that contains that data and save it in an arrayList of either Product or Person. After you have all the data input, write it to disk from the arrayList using the toCSV() you implemented.  
  
In the Reader, as you read a line of the data file use the java String split function to separate the fields of this single CSV record into an array. Then use the array data to create an object and add it to the arrayList. Your table of Person or Product data should be generated from the ArrayList.

I’ve asked you to create toJSON() and toXML() methods as well since these are common data formats.

**Always work with the Object representation of the data within the main code of your application. You only use the text based CSV, XML, or JSON to save and retrieve the data from file storage.**

## Part 2: Product

Project: Product  
Files: Product.java ProductTest.java  
ProductReader.java ProductGenerater.java

### Fields:

String name  
String description  
String ID // should never change   
double cost

Do the same thing with Product as you did with Person.Create the JUnit tests, UML, etc.

**Paste the screen shots: your read and write program runs, and the JUnit success tests. See Below.**

## Part 3: SafeInput Object

1. In CP I we created a library of static console methods called SafeInput.java. I gave you a copy of this and we used it in the Practicum. I’ve attached a copy here in the lab.
2. Create a new java class file called SafeInputObj.java. Add a **Scanner called pipe** as a class variable. (This will work with the existing pipe parameter in the SafeInput methods and sve you a lot of editing time.)
3. Implement 2 constructors, the default with no parameters (which will automatically use System.in) and a constructor that takes a Scanner as a parameter  
   **public SafeInputObj(); // Will set the Scanner pipe to System.in  
   public SafeInputObj(Scanner scanner) // uses the scanner parameter (we won’t use this .)**
4. Now copy all the static methods from SafeInput into SafeInputObj.
   1. Remove the static keyword from each method
   2. Remove the pipe parameter from each method signature  
      (Leave the references to pipe within the method code blocks since they now use the pipe variable that belongs to the class.)
   3. Create the JUnit tests for SafeInputObj
   4. Recreate or Correct the javaDoc block for each method to reflect these changes
5. Write a short program **ObjInputTest.java** and test each of the SafeInputObj methods to show they work.   
     
   **Provide Screen shots of the working sample program output for SafeInputObj. See Below.**

Since SafeInput really isn’t a data storage class like Person or Product, it does not have the toCSV() toXML() and toJSON() methods. Also, because all of the methods are interactive it is not clear how to implement a passive Junit test suite for it.

### Part 4: UML Class Diagrams

Create UML Class Diagrams for each of your classes SafeInputObj, Product, and Person and insert them here:

**PLEASE DO NOT TAKE SCREEN SHOTS IN DARKULA MODE. SWITCH TO LIGHT MODE  
  
Screen shots**: **[PART 1 - 3]** record screen shots of your output for each of your classes here as directed:

* **Successful Junit tests runs for Person, and for Product HERE**
* **Please just the single screen shot in IntelliJ that shows that all the tests passed, NOT A PAGE FOR EACH TEST!**

Shots from your application(s):

* **Creating/Saving Person and Product data HERE**
* **Displaying Person/Product data after reading the file. HERE**
* **Clear, legible shots of your java src code showing where you create an object with the field data and add it to the arrayList that is typed for it (Person, Product) HERE  
    
    
  Runs for your test program for SafeInputObj here:**

**Your GitHub link for this lab. (Must be public until we can grade it.)**

## Submission:

Save this file as **Lastname\_Firstname\_Lab01.docx** (using your name) with your screen shots.   
In Canvas:  
1. Submit a working URL link(s) to your GitHub repository for this project. Be sure it works! Copy it above here in this document also.  
2. Submit this file with the screen shots.