Assignment 01 - FileInOut

CSCI 242 - Computer Science II - Version 1.0.0

Due date is posted in the LMS.

Please read the entire assignment description before beginning the project.

Introduction

This assignment has a number of goals and objectives. Let's discuss them for a moment…

First, the transition from Cs1 to Cs2 is not an easy one to make. You learned the very basics of programming in Cs1: variables, conditionals, repetition and arrays. You may or may not realize it, but you also learned something else that will be the cornerstone of Cs2: ***objects***. We will spend approximately 75% of Cs2 talking about the various options that are available with objects and object oriented programming.

Second, we will be using a number of industry standards for Java program development that you most likely have never used before. We will be using an integrated programming environment (IDE) called IntelliJ IDEA rather than BlueJ.

Third, the documentation requirements in Cs2 are more rigorous. We will use a Java documentation tool called Javadoc. This tool allows you to write documentation that will generate its own web pages. Javadoc is used by Java programmers all over the world and is the industry standard for Java program documentation.

Fourth, you need some practice. ***No, really, you do!*** How long has it been since you coded? For most of you, it has probably been a while! So, a little practice and refresher is in order.

So, as you can see, Cs2 is quite different from Cs1 and is in many ways more difficult. We will start by taking a look at IntelliJ IDEA.

Instructions

Read the document below and perform the activities shown. The ☒ icon shows the activities you need to perform.

IntelliJ IDEA

You did all your programming in Cs1 using a simple editor named BlueJ. BlueJ is very good at helping introductory students develop small programs. However, you are no longer introductory students! In Cs2 you will be using a professional strength Integrated Development Environment (IDE) called IntelliJ IDEA. IntelliJ IDEA is developed and maintained by JetBrains to help programmers and developers manage complete projects – not just write programs.

In general, Integrated Development Environments or IDEs have many advantages over text editors and command line tools:

- They allow you to organize and navigate dozens of classes easily.
- They come with debuggers that (hopefully) help you locate and remove bugs more quickly.
- They make building graphical user interfaces easier.
- They come with plugins for doing additional tasks like testing and profiling.

While there are dozens of IDEs for Java, there are three with dominant market share: NetBeans, Eclipse and IntelliJ IDEA. Both NetBeans and Eclipse are quality IDEs. We will use IntelliJ IDEA but NetBeans and Eclipse are also available in our lab. In addition, IntelliJ IDEA is an "up and coming" IDE as it is the IDE used by Android Studio for Android App development.

The IntelliJ installation is easy. However, as you might expect, the procedure is different depending on the platform you will be using. Contact your instructor for any help needed.

Activity 1

- Follow the instruction for installing IntelliJ IDEA using the instructions in the document titled, "Installing JetBrains IDE's for Students" located at, **Home > General Course Information**.

There are a number of good resources out there that describe the main features of IntelliJ IDEA. All of the links below can be found in the LMS at, **Home > Programming Project 01 - FileInOut**.

Activity 2

- Look over the "IntelliJ IDEA IDE Information" page.
- Watch the "Overview of IntelliJ IDEA" video.

Coding Guidelines & Javadoc

One of the focuses in Cs2 is to make you better programmers. Being a better programmer is not simply about knowing how to code. There is much, much more involved. The process of software engineering is used to teach programmers how to be Software Engineers - the job you will most likely have someday! Our software engineering classes (CSCI 475 & 476) are designed to show you the process of software engineering and introduce you to some of the tools commonly used in industry.

One common tool used ***extensively*** in industry is documentation. "What?!? Wait a minute… you mean I'm going to have to write documentation when I get my first job? You mean all this in-code documentation writing wasn't just some kind of conspiracy against the students to keep us busy?!?" Yep, that's right! I know – we all like to code – that's why we're here in this class! But coding is only one part of the software engineering process. Coding falls under the "implementation" category, which accounts for only 30 – 35% of the process - coding is only one third of what you will be doing as a software engineer! The rest? Well, part of it involves writing documentation!

Javadoc is a tool for generating documentation in HTML format from comments within your source code. You write your documentation as you normally would but format the documentation in such a way that it conforms with the Javadoc standard. In addition, special "tags" are used to describe certain aspects of your documentation, such as the author, method parameters and return values.

The LMS has a document that describes the course guidelines for coding and Javadoc documentation.

Activity 3

- Read the document titled, “Coding Guidelines”. The document is located in the LMS at, **Home > General Course Information**.

Java Review

Now it is time to do some coding!

One of the things that we will be using throughout the semester is file input/output (I/O). In concept, file I/O is very important and allows you to read and write data to and from files. In this part of the assignment, we will develop a class to do file I/O and document it using Javadoc.

Requirements

The following are the requirements for the assignment.

1. You must use IntelliJ IDEA to complete the assignment.
2. You must document your assignment using Javadoc.

IntelliJ IDEA

One of the differences between Cs1 and Cs2 is that you are given more freedom to be creative in your programming assignments. What this means is that the *specifications for the assignments will become more and more sparse meaning that more will be left up to you*! But, for the moment, we will walk you through some of the process, as this might be the first time you have used IntelliJ IDEA!

So, start IntelliJ IDEA and let’s get to it!

Activity 4

- Perform the procedure below using IntelliJ IDEA.

Start IntelliJ IDEA

Obviously, in order to use the IDE, you need to run it!

1. Start IntelliJ IDEA. The way you do this depends on the platform you use.
2. Once started, IntelliJ IDEA will display the “Welcome to IntelliJ IDEA” window. This window shows the projects IntelliJ IDEA knows about. With your fresh install, you should not have any projects listed in the left hand pane.

Create a New Project

Creating a new project in IntelliJ IDEA is a breeze!

3. From the “Welcome to IntelliJ IDEA” window, select **+ Create New Project**. This brings up the “New Project” window.
4. In the “New Project” window, select **Java** in the left hand pane.
5. Click **Next**.
6. Click **Next** again (do not create a new project from a template).

7. In “Project Name”, give your project the name `A01_FileInOut`.
8. In “Project Location”, use the “…” button to select the folder that will hold the new project.
9. Click Finish.

That is it! Pretty easy, right? Let’s take a look at the project.

Look Over the New Project

Let us take a look at the structure of an IntelliJ IDEA project.

1. Expand the “Projects” folder tab by clicking on the triangle next to the project name if it is not already expanded.
2. Notice the three items in the folder: a `.idea` folder, a `src` folder and a file called `FileInOut.iml`. The `.idea` folder contains information about your project and configuration information for the IDE. You will/should not change anything in this folder. Another thing you will not need to touch is the `.iml` file. This file contains information about the project and should never be directly modified by you. The `src` folder, however, is where all the action takes place!

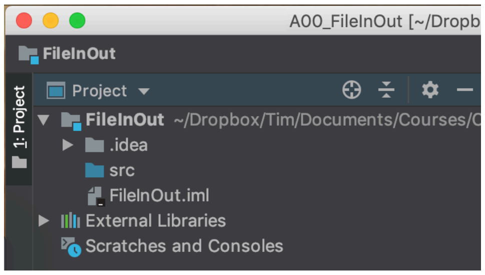


Figure 1 - The Project view

Create a New Package

In “Activity 3” above you were asked to read the document titled, “Coding Guidelines”. This document says that you should *never use the default package*. So, let us create a new package to hold our source code and name it according to “Coding Guidelines”.

3. Right click the `src` folder and select **New > Package.**
4. In the “New Package” dialog that pops up, type `edu.uwp.csci.Cs242.assignment.a01.fileinout` in the “Enter new package Name” field.
5. Click **Finish**.

Now you have a project with a proper source package to hold you Java source files.

Create a New Java Source File

The next step is to create a Java source code file. Again, this is quite easy!

6. Right click the `src` folder and select **New > Java Class**. The “Create New Class” dialog pops up.
7. Type `FileInOut` in the “Name” field.
8. Make sure “Class” is selected in the “Kind” field.
9. Click **Finish**.

IntelliJ IDEA creates a new Java source file for you with the proper package. Your code should look something like this:

```
package edu.uwp.csci.Cs242.assignment.a00.fileinout;

public class FileInOut
{
}
```

Much like BlueJ, IntelliJ IDEA color codes structural aspects of you code for easy identification when reading your code or doing code development. The actual color scheme can be changed and depends on the theme you selected when installing the program.

Java Coding

Now that we have a properly setup project, we can start writing `FileInOut`. We will do this in four parts.

Develop the Java Code Part 1 - The Research

The first step is to research how Java handles files. We will use our textbook. In particular, we need to pay attention to Liang Chapter 12, sections 10 and 11. Let us do the research first, then code.

	Activity 5 • Read Liang 12.10 and 12.11.

At this point, after reading Liang, you should have a good theoretical understanding of how files work. To summarize: the `File` object encapsulates all of the *properties* of a file – you do not actually read/write data using the `File` object. Instead, you open and close files so that you can read and write to/from them. `Scanners` are used to actually read a file and `PrintWriters` are used to actually write files. Pretty easy right?

Develop the Java Code Part 2 - The Getters and Setters

It's time to code!!! Doing your code in ***small increments*** and testing each of those increments separately leads to code the works! In the next sections, we will develop the `FileInOut` class step by step. In addition, we want our code to be ***reusable***. So, we try to make our code ***as general as possible*** trying to think of every possible use for the class.

Our `FileInOut` class will have six class-level variables. We will define four of them now and wait to define the other two until later (remember, incremental coding - step-by-step, little-by-little, piece-by-piece).

	Activity 6 • Perform the procedure below to develop your getter and setters.

10. Using IntelliJ IDEA, type in the following class level variables.
    a) `DEFAULTINFILENAME` - A `String` constant that holds the name of the default input file. The default value is "`default_in.txt`".
    b) `DEFAULTOUTFILENAME` - A `String` constant that holds the name of the default output file. The default value is "`default_out.txt`".
    c) `inFilename` - A `String` variable that holds the name of the input file.
    d) `outFilename` - A `String` variable that holds the name of the output file.

Make sure you have the right visibility modifier for each class-level variable! If you do not remember what a visibility modifier is, look back in your book!

11. Write a stubbed-out constructor. We want our constructor to do a number of things. But for the moment, we will stub it out so that is does nothing. We will come back later to fill it in. Put a comment above the constructor that says: `// TODO: write a useful constructor`.
12. Write getters and setters for each of the class-level variables. Remember, your getters should be named `get`*`VariableName`* and your setters should be named `set`*`VariableName`*. Notice that as you type in the body of the getters/setter, IntelliJ IDEA tries to help you identify possible variables to use. In the body of the getter, for example, you want to say, `this.inFilename =` … When you type `this.`, IntelliJ IDEA shows you the possible variables to choose from and lets you actually choose from a dropdown list.
13. Add the documentation for the getters and setters. Go to the line above the method and type `/**` <enter>. This will signal IntelliJ to insert Javadoc based on the method signature and its return values! Cool, right?!? Fill in the information per our "Coding Guidelines" document.

Develop the Java Code Part 3 - Finish FileInOut

Now it is your turn! Your assignment is to complete the remainder of `FileInOut`. I have created a Javadoc output page at: `http://www.cs.uwp.edu/staff/knautz/FileInOutJavadoc/`. This web page is an example of Javadoc output. Remember that Javadoc takes the comments in your code and builds a web page out of them. All of the formatting is done for you; all you need to do is document your code correctly!

Activity 7

- Perform the procedure below to develop your `FileInOut`.

14. Use your web browser to view the Javadoc web pages shown above. Look over the `FileInOut` Javadoc and understand what is happening. Combine what is shown here with the Liang chapters you read earlier. Re-read them if you need to. ***Everything you need to complete the rest of the assignment is found in the Javadoc and the textbook***.
15. Make sure you document your code per the "Coding Guidelines". Make sure you have comments throughout your code.
16. It is probably best to start your implementation with `openInFile()` and `openOutFile()`. Then, `openFiles()`. Finally the constructor.

NOTE:
`openInFile()` will require you to use Java exceptions: what happens if someone send in a zero length file name? You will also find that IntelliJ will "complain" about the `Scanner` and `PrintWriter` objects. Why? Because the `Scanner` and `PrintWriter` constructors *throw exceptions*. You must use a try/catch block to use them properly. Look at Liang Chapter 12.

Develop the Java Code Part 4 - Develop a Main for Running the Project

As you develop your `FileInOut`, you will need to test as you go along. Again, remember, incremental coding - step-by-step, little-by-little, piece-by-piece. Develop some of your `FileInOut`, then test it.

Activity 8

- Perform the procedure below to develop your `main()`.

17. Add a main to your code. Do this by adding a new class called `MainDriver`. If you do not remember how to add a new class, look at the procedure you used above.
18. The `MainDriver` class has a single method only: `main()`. Add the `main()` method to the `MainDriver` class.
19. To test `FileInOut` in `MainDriver`, what will you need to do? This is important! Answer that question before you move on! The set up for this might be a little different than what you did in Cs1. Here, the `main()` and the implementation class are in different files! Is that okay? Yes! But, again, we ask the important question: what will you need to do to use `FileInOut` in `MainDriver`?
20. Hopefully you considered the answer before reading this… If not, please take a moment to consider and answer that question; it is very important because it shows that you know and understand how objects interact with each other.
21. The answer is to instantiate a `FileInOut` object in your `main()`. Write the code to do that now. Remember that you may not have a full constructor yet. That is okay. Instantiate what you have.

22. `openInFile()` and `openOutFile()` can be tested on their own once you have instantiated your `FileInOut` object.

Add a Test Data File

Before you run your main, you will need to create some test data in, well… a file! *Java looks for input files in the root folder of the project*, not in the `src` folder of any of its subfolders.

Activity 9

- Create a test input file in the project root directory. Put whatever text in the file you like.

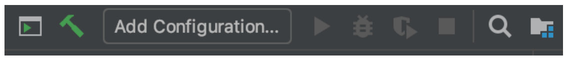
Add a Run Configuration

You are now ready to run your main. To do this, you will need to create what is called a *run configuration*. IntelliJ supports the idea of being able to run your program in any way possible. Thus, when you want to run a Java program in IntelliJ, you create a run configuration. A project can have as many run configurations as you like.

Activity 10

- Perform the procedure below to configure your run configuration.

23. In the upper right hand corner of IntelliJ is a button that currently says, “Add configuration”. Click this button.



24. Click the “+” sign in the upper left hand corner (as IntelliJ tells you) and select “Application” from the list of configuration templates that drop down.
25. In the “Name” field at the top of the right pane, type “MainDriver”.
26. In the “Main Class” field, type “MainDriver”. IntelliJ will try to help you locate the `MainDriver` class. Note that you select the `MainDriver` class rather than the `FileInOut` class because the `main()` is contained in `MainDriver`.
27. Make sure you have a JRE in the “JRE” field. If not, you will need to select one. The “default” is probably fine.
28. Select **OK**.
29. To build and run your program, click the little green triangle next to the run configuration button.

Your program should build and run. If not, you will need to debug it to get it to run. *Continue to build and run until you have it working*.

Submission

Zip up *your entire IntelliJ IDEA project folder* into a single zip file named `a1.zip` and submit that one file. Submit the zip file to the assignment folder labeled “Assignment 01 - IntelliJ, Javadoc and Exceptions with FileInOut >>> DROPBOX”. I should be able to unzip the entire project and load the entire project in IntelliJ IDEA without incident.

	Activity 11 • Submit your completed assignment to the correct LMS assignment folder.

Grading

See the document titled, “Programming Requirements and Rubric” found in the “General” section in the LMS for details on how the program will be graded.

Good software engineering is expected. Remember to follow the “Coding Guidelines” found in the LMS. Use lots of comments throughout the code, lots of method comments, and appropriate variable names.

Ω