Lab 1
Using Eclipse and Submitting Files

Welcome to your first lab session!

Completion of the lab exercises is very important in this course. The labs give you an opportunity to make sure you understand the course material and they give you a chance to practice new skills before using them to write your larger lecture programs. Therefore, you should take your lab work seriously.

Note:
This set of instructions assumes that you are familiar with the Windows operating systems and can copy files, run programs, etc. If that is not the case, please get individual help from your lab instructor.

Preparing for lab

In most cases, there will be one or more pre-lab exercises: work that you are expected to complete **before** coming to lab. You need to have a paper copy of the pre-lab to turn in as you walk into lab. Your GA will look at your pre-lab to be sure it is completed (it does not have to be correct, but a reasonable attempt is required for points). You will then be provided with the answers to check your work. Be sure to check the course website (www.itk.ilstu.edu\it168\ *the last slash is important*) for the pre-lab assignment well before your lab so that you can complete the pre-lab work. Your lab instructor will show you where to find the pre-lab in class today. Pre-lab assignments will typically be made available by Friday morning the week before lab.

<u>Your computer log-in</u>

You will be using your ULID and password for an H: drive account to use for 168 work only. ***Do not share it with anyone.***

The H: drive account will provide you with a portion of a networked hard disk that you can access from virtually any computer on which you can access ADILSTU - including the lab and where you live. If you live off campus, you must use the VPN client to access the ISU network. Detailed instructions are posted on the website in the files: "Working on Your Personal Comuter" and "Accessing IT Drives". Another document, "Using IT's Virtual Computers" has instructions for using the virtual computer to get the same desktop you see in lab. (An option to use instead of downloading software to your computer.)

The purpose of the H: drive is to give you a place to store your course work that is accessible from multiple places. You must not store anything else on this disk (particularly not multi-media files) of any kind.

The account also gives you access to the department's T: drive. This contains information provided by instructors for various IT courses. Any files provided to you will be available on the **T:** drive in the **it168/Labs** folder. You can copy things from the T: drive, but you may not modify anything on the drive.

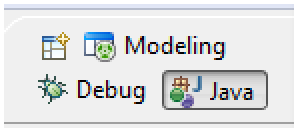
- Log in and create three folders in your H: drive:
    - work
    - Preferences
    - Lab01

Opening Eclipse in the Lab

On your desktop, there is a folder named Java tools. Open that folder, and double-click on the program labeled Eclipse (H drive).

Instructions for downloading the open source software that is used for developing Java programs in this class are available on the website.

In the top right corner, check to be sure you are in the "Java" perspective.

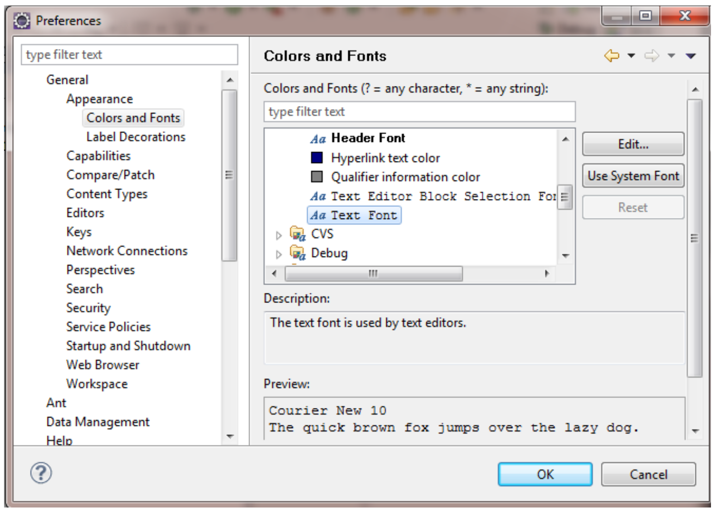


If not, click on the window with the + sign for the menu.

Set Eclipse Preferences

Eclipse preferences allow you to customize settings in your workspace and save them for the next time you return to the workspace. You can also export your settings to be used in any other workspace you create.

- From the Menu select: Window -> Preferences
- Expand: General
    - Appearance
        - Colors and Fonts
        - Basic
        - Text Font
        - Click the Edit button

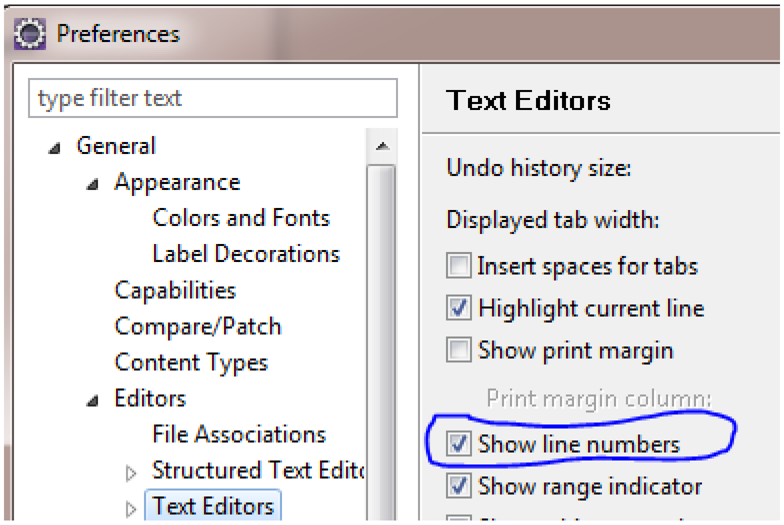


The text font is used by text editors.

```
Courier New 10
The quick brown fox jumps over the lazy dog.
```

- This will allow you to change the font, font style, size, and/or color.
    - Leave the font as Courier New.
    - Change the font size if size 10 is too small.

- Editors -> Text Editors
    - Check "Show line numbers"


Preferences

type filter text

- General
  - Appearance
    - Colors and Fonts
    - Label Decorations
  - Capabilities
  - Compare/Patch
  - Content Types
  - Editors
    - File Associations
    - Structured Text Editc
    - Text Editors

Text Editors

Undo history size:

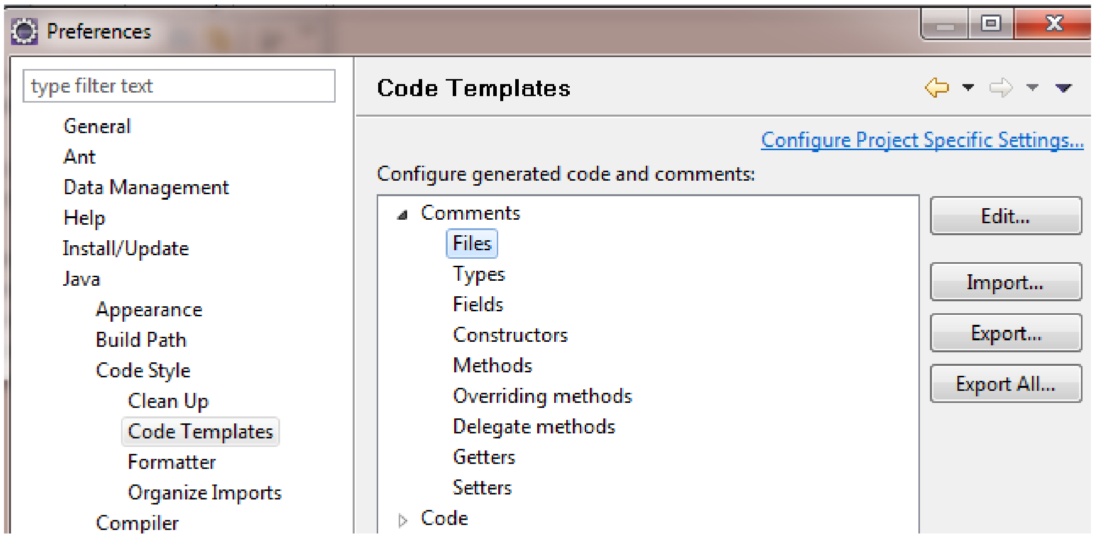
Displayed tab width:

- [ ] Insert spaces for tabs
- [✓] Highlight current line
- [ ] Show print margin

Print margin column:

- [✓] Show line numbers
- [✓] Show range indicator

- Expand: Java
  - Code Style
  - Code Templates
  - Expand: Comments
  - Files
  - Click on the "Edit" button


Preferences

type filter text

- General
- Ant
- Data Management
- Help
- Install/Update
- Java
  - Appearance
  - Build Path
  - Code Style
    - Clean Up
    - Code Templates
    - Formatter
    - Organize Imports
  - Compiler

Code Templates

Configure Project Specific Settings...

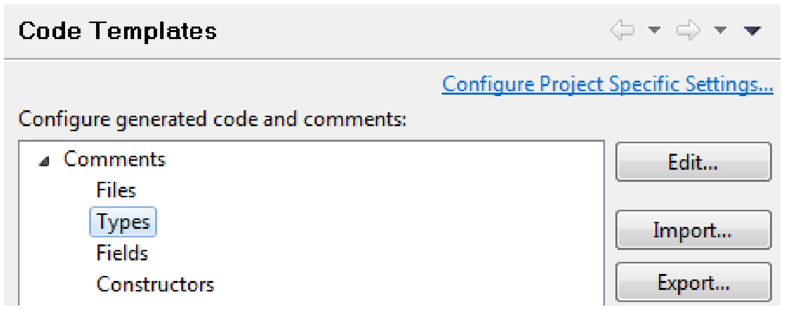
Configure generated code and comments:

- Comments
  - Files
  - Types
  - Fields
  - Constructors
  - Methods
  - Overriding methods
  - Delegate methods
  - Getters
  - Setters
- Code

- Modify the comment to look like the example below.
    - Change from a Javadoc comment to a general comment by removing one of the *s from the first line. (You want /* not /**).
    - Put your name after programmer.
    - Variables will be used to fill in the current date and file name for you.
        - Place the cursor where the variable is to be inserted.
        - Click on the "Insert Variable . . ." button
            - Double-click on "date"
        - Place the cursor after the File name
            - Insert Variable . . .
                - Double-click on "file_name"

```
/*
 * File name:  ${file_name}
 *
 * Programmer:  insert your first and last name
 *  ULID:  your ulid
 *
 *  Date:  ${date}
 *
 *  Class: IT 168
 *  Lecture Section:   your lecture section
 *  Lecture Instructor:   your lecture instructor
 *  Lab Section:
 *  Lab Instructor:
 */
```

- Note: ${file_name} and ${date} are inserted variables and will automatically be replaced with the file name and date when the class is created.
- Now click on "Types" under Comments

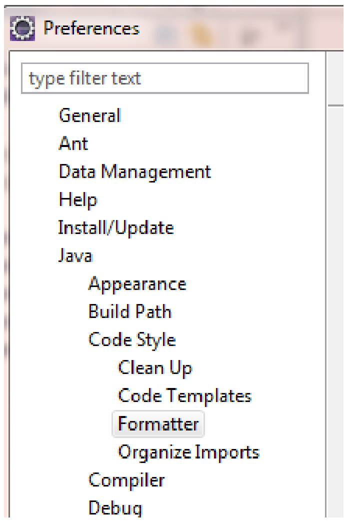

Code Templates
Configure Project Specific Settings...
Configure generated code and comments:
Comments
Files
Types
Fields
Constructors
Edit...
Import...
Export...

- Click on the "Edit" button

- Modify the comment to look like the example below.
    - This is a Javadoc comment, so it should start with /**
    - We're using the placeholder *<insert class description here>* as a temporary reminder that each time we create a class we need to insert a good class description. **Never leave this in your final source code - always replace it with something meaningful.**
    - Remove the variable associated with the author tag and replace it with ***your full name***.

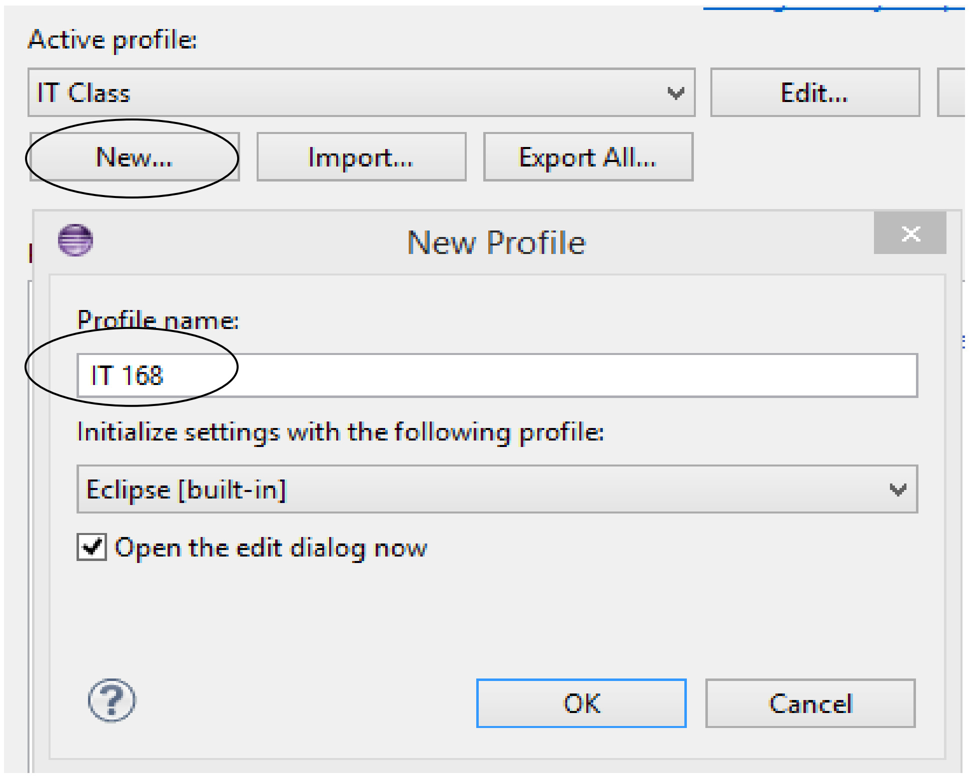
```
/**
 *  <insert class description here>
 *
 *  @author  your name
 *
 *  ${tags}
 */
```

- Java -> Code Style -> Formatter



- Click on the New button, enter a Profile name with the class name (IT168), and click OK.
- Here you can customize the editor to match the specific requirements for the class.
- Things that can be changed are: braces, tab width, blank lines, etc.

.



Active profile:

IT Class

Edit...

New... | Import... | Export All...

New Profile

Profile name:

IT 168

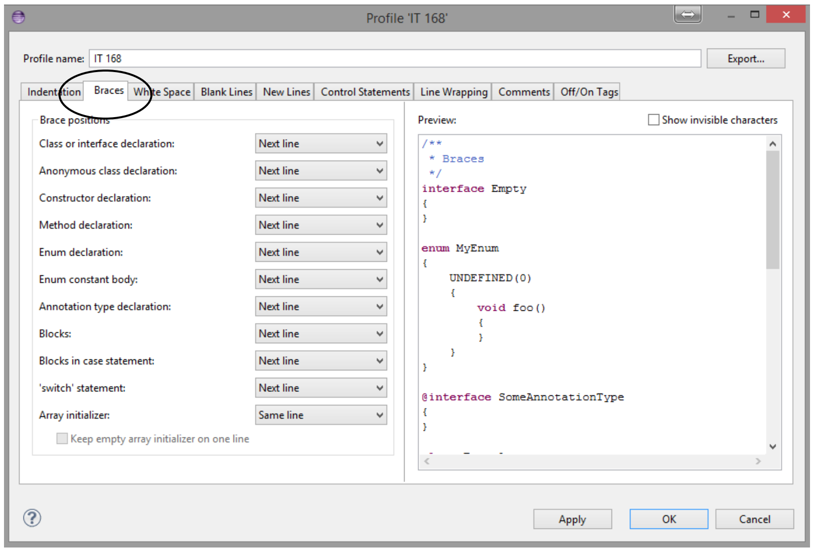
Initialize settings with the following profile:

Eclipse [built-in]

[✓] Open the edit dialog now

OK | Cancel

- Select the Braces tab and set each brace position except the last one (i.e., *Array initializer*) to Next Line. Do not close this window yet.



Profile 'IT 168'

Profile name: IT 168 | Export...

Indentation | Braces | White Space | Blank Lines | New Lines | Control Statements | Line Wrapping | Comments | Off/On Tags

Brace positions

Class or interface declaration:	Next line
Anonymous class declaration:	Next line
Constructor declaration:	Next line
Method declaration:	Next line
Enum declaration:	Next line
Enum constant body:	Next line
Annotation type declaration:	Next line
Blocks:	Next line
Blocks in case statement:	Next line
'switch' statement:	Next line
Array initializer:	Same line

☐ Keep empty array initializer on one line

Preview: ☐ Show invisible characters

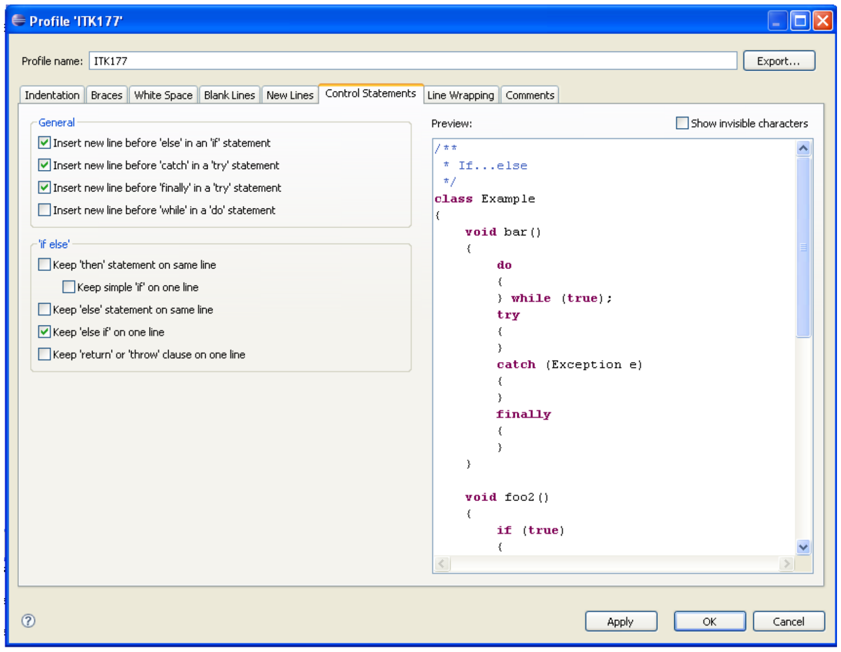
```
/**
 * Braces
 */
interface Empty
{
}

enum MyEnum
{
    UNDEFINED(0)
    {
        void foo()
        {
        }
    }
}

@interface SomeAnnotationType
{
}
```

Apply | OK | Cancel

- Select the Control Statements tab and make it match this window.



Profile name: ITK177

General
- [✓] Insert new line before 'else' in an 'if' statement
- [✓] Insert new line before 'catch' in a 'try' statement
- [✓] Insert new line before 'finally' in a 'try' statement
- [ ] Insert new line before 'while' in a 'do' statement

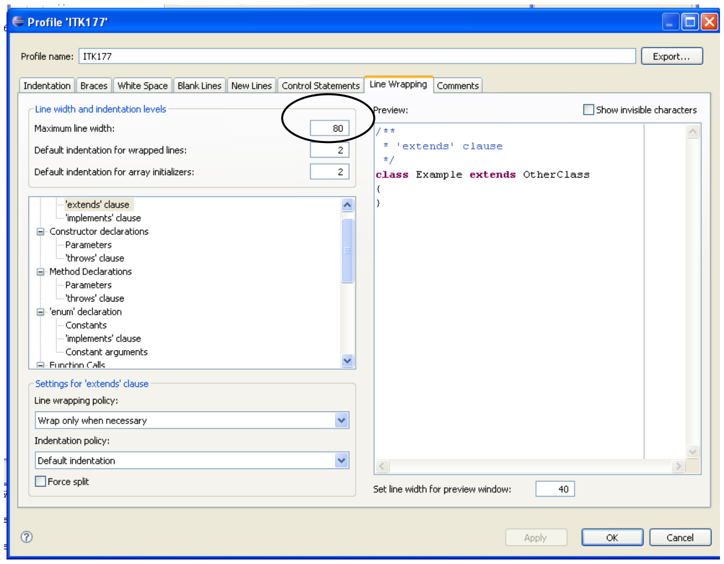
'if else'
- [ ] Keep 'then' statement on same line
  - [ ] Keep simple 'if' on one line
- [ ] Keep 'else' statement on same line
- [✓] Keep 'else if' on one line
- [ ] Keep 'return' or 'throw' clause on one line

Preview: [ ] Show invisible characters

```
/**
 * If...else
 */
class Example
{
    void bar()
    {
        do
        {
        } while (true);
        try
        {
        }
        catch (Exception e)
        {
        }
        finally
        {
        }
    }

    void foo2()
    {
        if (true)
        {
```

- Select the Line Wrapping tab and make sure the maximum width is set to 80. Now click OK to close the window.



Profile name: ITK177

Line width and indentation levels
Maximum line width: 80
Default indentation for wrapped lines: 2
Default indentation for array initializers: 2

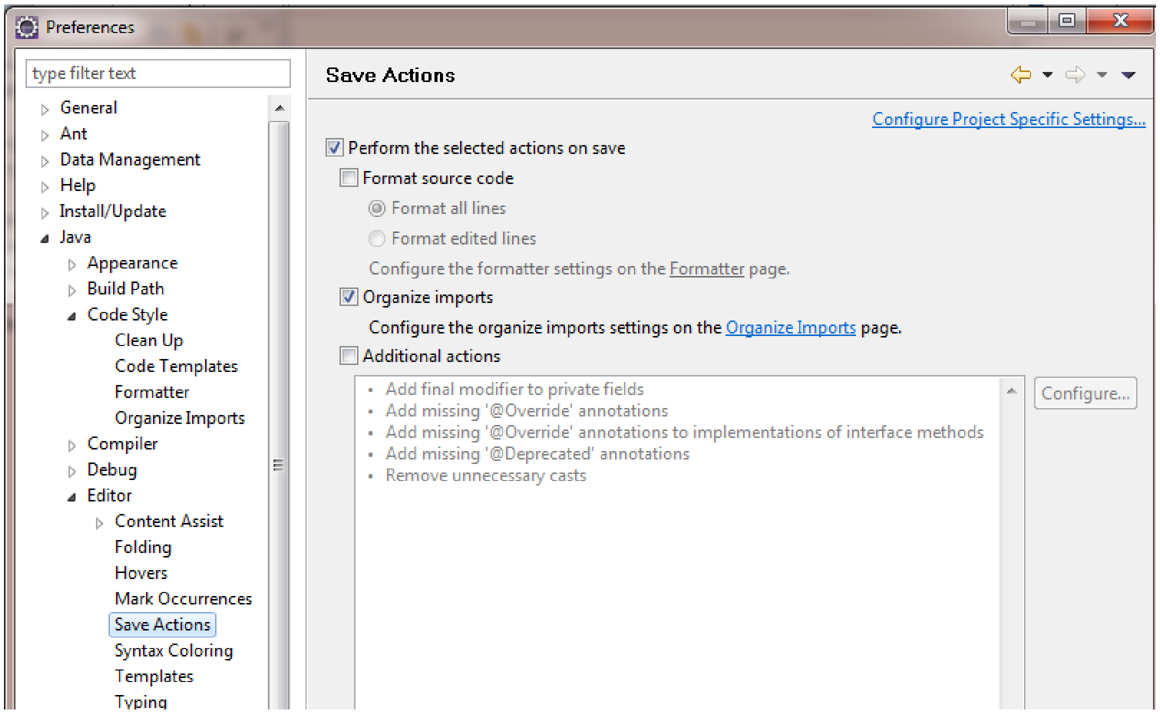
Settings for 'extends' clause
Line wrapping policy: Wrap only when necessary
Indentation policy: Default indentation
[ ] Force split

Preview: [ ] Show invisible characters

```
/**
 * 'extends' clause
 */
class Example extends OtherClass
{
}
```

Set line width for preview window: 40

- Java
    - Editor
        - Save Actions
        - Match the following:



[✓] Perform the selected actions on save

[ ] Format source code

(●) Format all lines

( ) Format edited lines

Configure the formatter settings on the Formatter page.

[✓] Organize imports

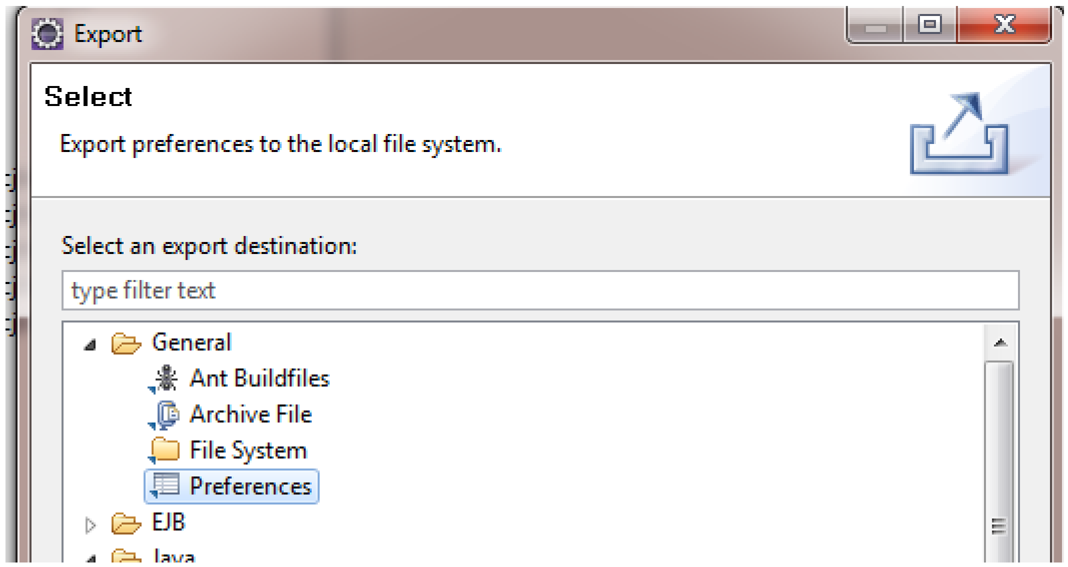
Configure the organize imports settings on the Organize Imports page.

[ ] Additional actions

- Add final modifier to private fields
- Add missing '@Override' annotations
- Add missing '@Override' annotations to implementations of interface methods
- Add missing '@Deprecated' annotations
- Remove unnecessary casts

- Click on the "Apply" button
- OK

- After you get Eclipse the way you like it, export the preferences so you can import them into a new workspace instead of having to set them all again.
    - File -> Export



Export

Select

Export preferences to the local file system.

Select an export destination:

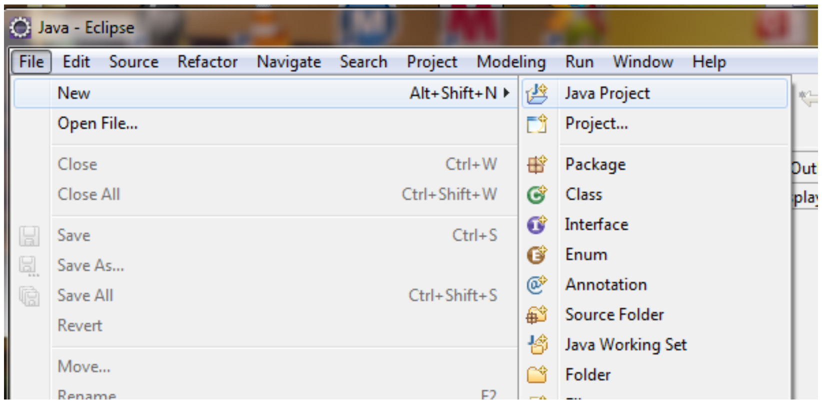
type filter text

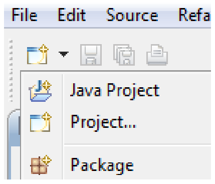
- General
    - Ant Buildfiles
    - Archive File
    - File System
    - Preferences
- EJB
- Java

    - General -> Preferences
        - Next
        - Browse to your H: drive and create a Preferences folder to store the file in.
        - Give the file a name
        - Save
        - Finish
    - Later, in another workspace you can import the file:
        - File -> Import . . .
        - General -> Preferences
        - Next
        - Browse to find the preference file you saved
        - Save
        - Finish

<u>Create a Java Project</u>

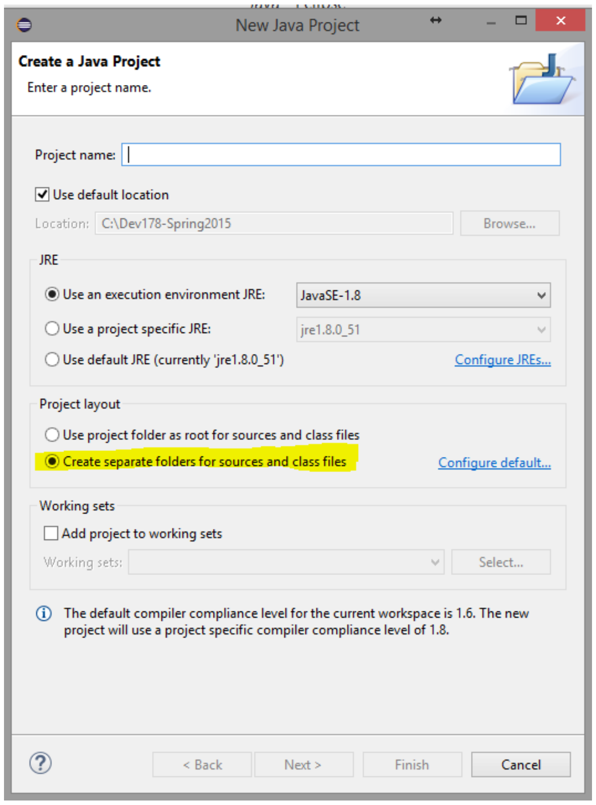
- File -> New -> Java Project
    - (Or File -> New -> Project and choose Java Project in the next dialog box)



- or click on the New icon and choose Java Project 

- For the "Project name", name it: "Lab01"
    - You need to use this same naming convention throughout the whole semester for each lab and program.
        - Example program name: "Program1"
    - Note that project names cannot have any blanks in the name.
- Location
    - You want to make sure you are aware of where you are storing your project.
    - Your default location should be H:\work
        - On your own computer you have 3 choices
            - work on your hard drive
            - create a workspace on a flash drive to carry between computers.
            - Log in to the ISU network and use your assigned H: drive.
                - As long as you have logged into the ISU network, this allows you to access your files from multiple computers. network.

- Under project layout, check "Create separate folders for sources and class files"



New Java Project

Create a Java Project

Enter a project name.

Project name:

[✓] Use default location

Location: C:\Dev178-Spring2015

JRE

(●) Use an execution environment JRE: JavaSE-1.8

( ) Use a project specific JRE: jre1.8.0_51

( ) Use default JRE (currently 'jre1.8.0_51')

Project layout

( ) Use project folder as root for sources and class files

(●) Create separate folders for sources and class files

Working sets

[ ] Add project to working sets

The default compiler compliance level for the current workspace is 1.6. The new project will use a project specific compiler compliance level of 1.8.

- Finish

Create a Package

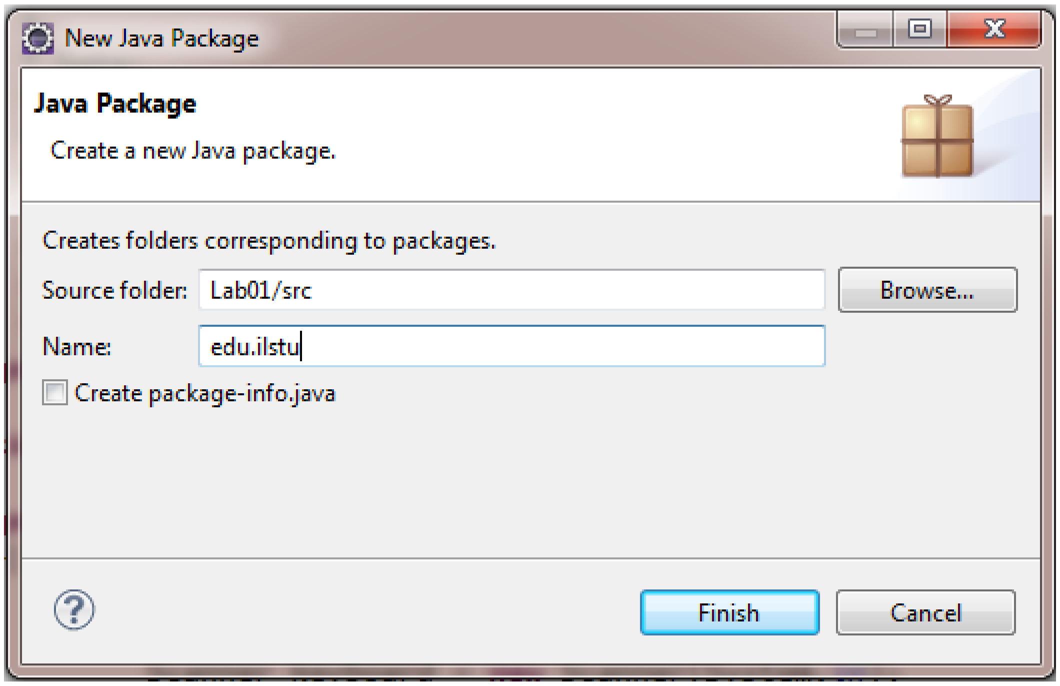
When you have a small project with few files, things are fairly simple. However, when working on a large project with many files, it is a good idea to have a way to organize them. This is done in your workspace by using *packages*, similar to organizing your hard drive with folders.

Advantages:

- Makes it easier to find and use types
- Avoids naming conflicts
- Can be used to control access

To create a package:

- Highlight the *src* folder in your project.
- File -> New -> Package
    - You can also right-click to get the menu.
- Name the package: **edu.ilstu**
    - Note: Package names must be all lowercase letters or a period
- Finish


New Java Package

Java Package

Create a new Java package.

Creates folders corresponding to packages.

Source folder: Lab01/src

Name: edu.ilstu

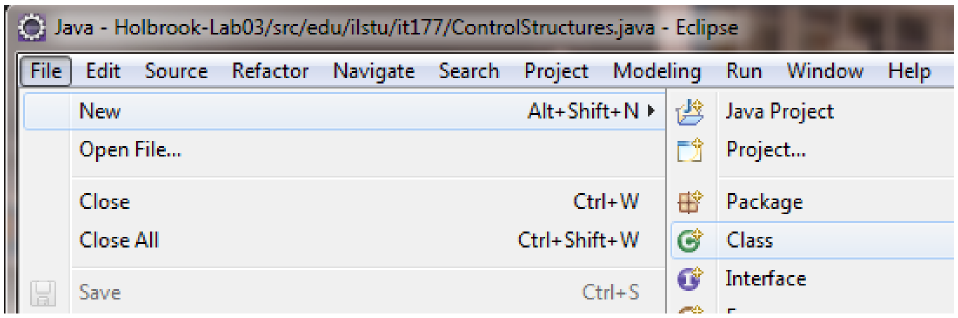
Create package-info.java

If you do not use a *package* statement, your java files end up in an unnamed (default) package. Generally, an unnamed package is only for small or temporary applications such as the type you will write in this class. You will not be required to use a package, but it is a good idea to get into the habit of using a named package.

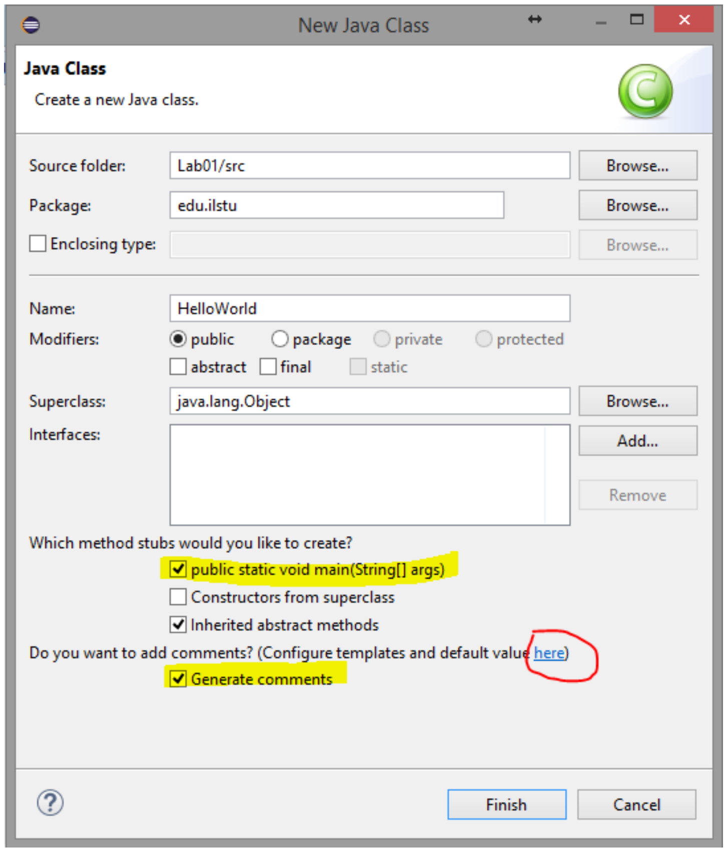
Package names are written in all lowercase letters. Convention is for a company to use their reversed Internet domain name to begin their package names. The one we will use is slightly modified, but similar. You can also add more qualifier(s) as needed to distinguish packages from each other.

Create a new Java file in a project

- Right-click on the package name where the file should be saved
- New -> Class


Java - Holbrook-Lab03/src/edu/ilstu/it177/ControlStructures.java - Eclipse

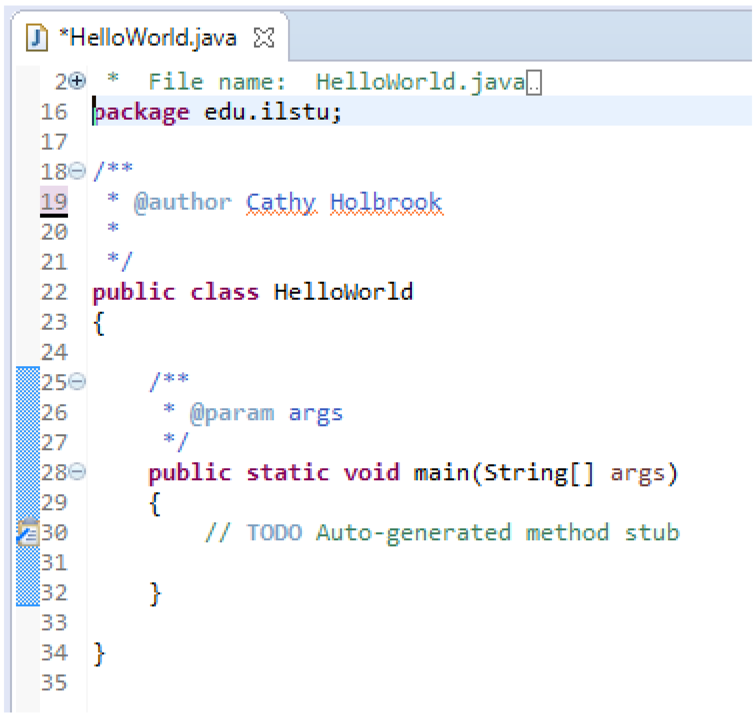
- Name: HelloWorld
    - Be sure that there are no spaces in the name
- Select the box for "`public static void main(String[] args)`" under "Which method stubs would you like to create?"
    - You will **NOT** want to check this for all classes that are created. How you determine that will be discussed in class.
- Select the box for "Generate comments" under "Do you want to add comments?"
    - This will insert the comment blocks that you set up in preferences so that all you will need to do is replace the "<insert class description here>"
    - You **WILL** want to check this for all classes that are created unless you want to create your comment blocks from scratch.
        - By clicking on the here link right above the "Generate comments" line, you can set the default to have this checked so you don't have to do it each time.



Source folder: Lab01/src
Package: edu.ilstu
☐ Enclosing type:
Name: HelloWorld
Modifiers: ● public ○ package ○ private ○ protected
☐ abstract ☐ final ☐ static
Superclass: java.lang.Object
Interfaces:
Which method stubs would you like to create?
☑ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
☑ Generate comments

- Finish

- Your file should look like this:



```
 2⊕  *  File name:  HelloWorld.java
16  package edu.ilstu;
17
18⊖ /**
19   * @author Cathy Holbrook
20   *
21   */
22  public class HelloWorld
23  {
24
25⊖     /**
26       * @param args
27       */
28⊖     public static void main(String[] args)
29     {
30         // TODO Auto-generated method stub
31
32     }
33
34  }
35
```

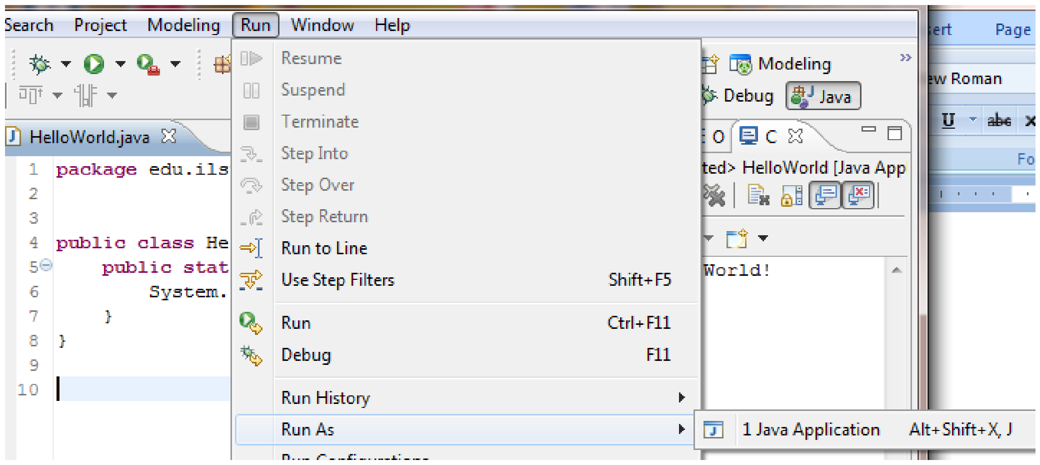
- Some things to notice:
    - The line numbers on the left go from 2 to 16. That, along with the 2⊕ indicates that some rows are hidden. Click on the plus sign to expand the lines and you will see your whole comment block.
        - This type of comment block will be green in your editor. Comment blocks that are blue are JavaDocs (more about that later).
    - The asterisk before the file name in the tab indicates that the file has not been saved since the last change. Be sure to save often.

- Fill in the parts of the comment blocks that is specific for each class. This should always be the first thing that you do when writing the code for a new class.
    - Replace "<insert class description here>" with
        - This is the traditional first program for people to run when attempting to learn a new programming language.
- Write the code for the program:
    - Replace the "//TODO code application logic here" with:
        - `System.out.println("Hello World!");`
            - You need to be very precise with Java. Type the line exactly as you see it here and don't forget the punctuation!

Running a Java Program

- Select the file of the program you want to run.
    - Click on HelloWorld.java.
        - This can either be the one open in the editor or the file name in Package Explorer on the left.
    - There are 3 ways you can run a program:

1. On the menu: Run -> Run as -> Java Application

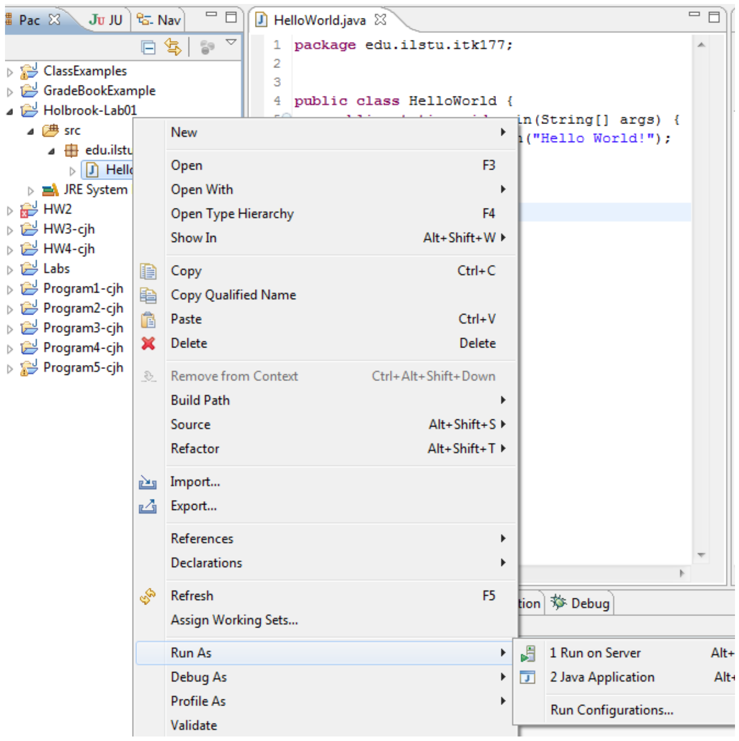


```
package edu.ils


public class He
    public stat
        System.
    }
}
```

```
World!
```

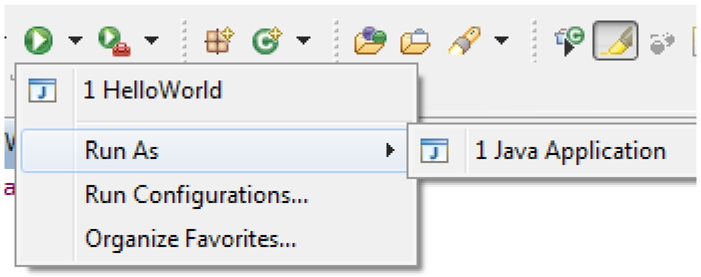
2. Right-click on the file to get a pop-up menu
    a. Run As -> Java Application



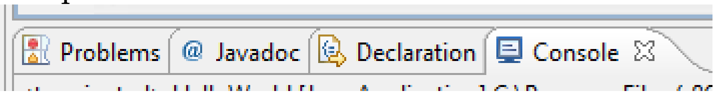
```
package edu.ilstu.itk177;


public class HelloWorld {
    ...in(String[] args) {
    ...("Hello World!");
```

3. Click on the arrow on the right of the Run icon
   a. Run As -> Java Application



- Click on the "Console" tab (in the bottom window on the screen) to be able to see the output.



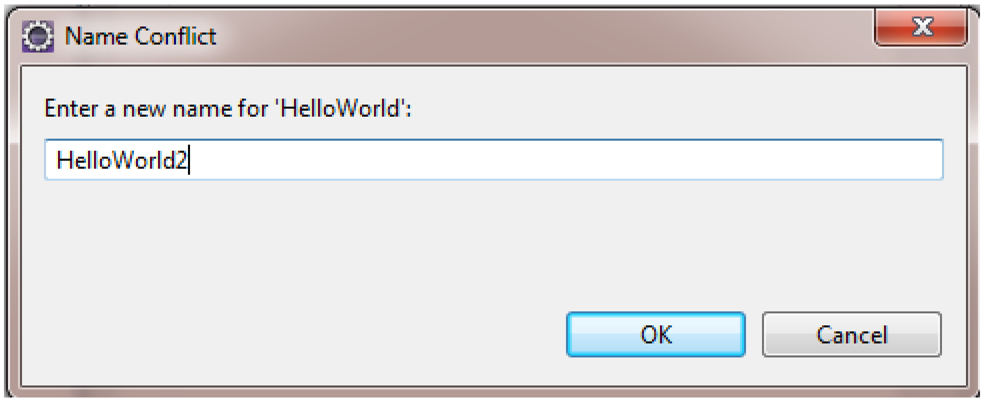
- What is your output?

- Save

- After the lab is complete, this is one of the files that will be submitted.
  - All files for the lab will need to be put into a zip file to be submitted at one time.

<u>Modifying a Java Program</u>

- Make a copy of HelloWorld.java and save it with a new name
    - Right-click on the file name in the Package Explorer
    - Click on "Copy"
    - Right-click on the package name
    - Click on "Paste"
    - Change the name in the dialog box to HelloWorld2



Enter a new name for 'HelloWorld':

HelloWorld2

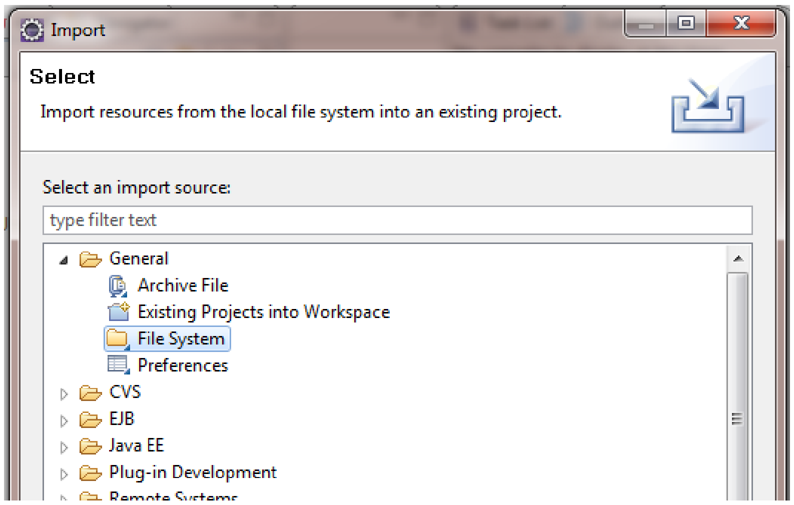
- Double-click on HelloWorld2.java to open the file.
- Add a new line of code to the existing program after the "Hello World!" statement.
    - Type the following:
        - `System.out.println("Welcome to IT 168!");`
- Save
- Run (see previous instructions if needed)

- This file will need to be submitted when the lab is completed.

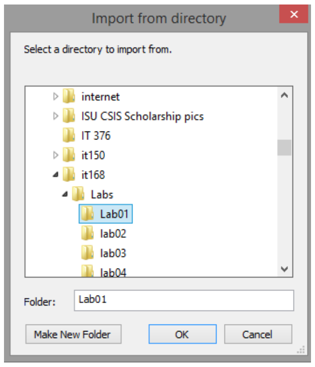
<u>Import a Java file into a project</u>

Follow this procedure any time you are provided with a file that is to be part of your project.

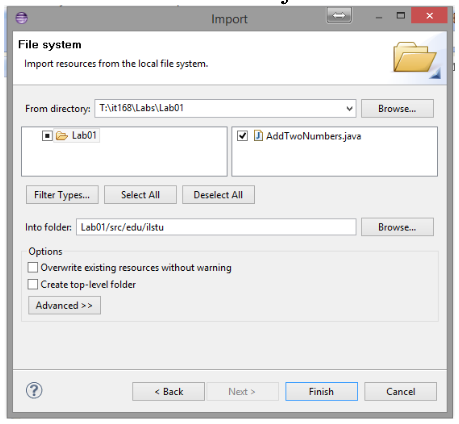
- You will import the file `AddTwoNumbers.java` from T:\IT168\Labs\Lab01 drive.
- In Eclipse, select the package
- File (or right-click) -> Import -> General -> File System
- Next



- Browse to the T: drive folder listed above.
  - Note that you just go to the folder.



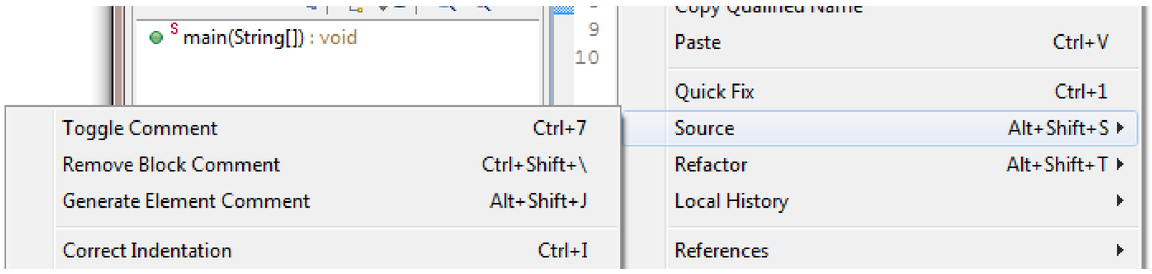
- Click in the AddTwoNumbers.java box



  - An advantage of choosing the folder is that multiple files can be imported at the same time.

- Double check that the "Into folder" is correct.
- Finish

- Double-click on the AddTwoNumbers.java file to open it in the edit window.
- Complete the Beginning and Class comment blocks.
  - Fill in the blanks in the beginning comment block.
    - The only way to automatically generate this comment block is when creating a new file.
  - Add a comment block for the class.
    - Place the cursor on a line between the lines
      - `import java.util.Scanner;`
      - `public class AddTwoNumbers`
    - Right-click to get the menu and choose Source / Generate Element Comment



- This will insert the comment block for a class that you created when setting up your preferences.
- All you need to do is replace the `<insert class description here>` with a short description of the class. For this just enter:
  - `Adds two numbers read from the keyboard.`
- Save
- Run (see previous instructions if needed)
  - There will be a prompt in the Console window. Follow the directions and hit ENTER to get the result.

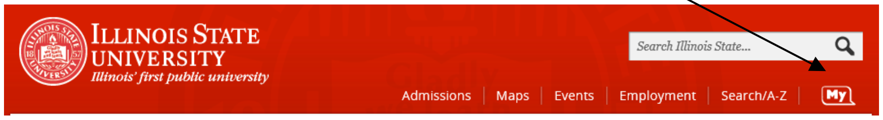
Submit the Files using ReggieNet

Step 1: First you need to prepare your files for submission. Java file names must match the name given to the class definition. If the file name is changed, the class will not function. You will be required to zip all your submitted files into a single zip file. **You are to zip individual files only. DO NOT zip a folder containing the files as this will make grading more time-consuming and your grade will be reduced for it.**

Follow these steps to zip your lab files:

- Typically you will be submitting multiple files for each lab or program. You are required to submit only one file so you must **wait until your entire lab or program is complete to submit.**
- Locate all files to be submitted in your `H:work\Lab01\src` folder. (Keep clicking on edu and ilstu if you used a package.) Remember that you are to submit the **.java** file **NOT** the **.class** files found in the `bin` folder for all Java classes. Any **.class** files submitted will not receive credit.
- Copy them to the Lab01 folder you created.
- Select all files to be submitted. You can do this using **Ctrl-click**.
- Right-click on the set of selected files and choose **Send To -> Compressed (zipped) Folder**.
- Take care in the naming of the zip file. Each lab will dictate the name to use. Failure to use this name will also reduce your grade.
- **Zip your lab 1 files to a file named Lab01.** Your project should include the following files:
    - HelloWorld.java
    - HelloWorld2.java
    - AddTwoNumbers.java

Step 2: Submit the zipped file through ReggieNet to the Lab 01 Assignment.

- Go to your ReggieNet account for this class and log in.
    - Go to https://reggienet.illinoisstate.edu/
        - Alternative ways are to go to http://ilstu.edu or http://illinoisstate.edu and click on the My in the right top or the home page:
        - 

    - Go to *My Sites* and select the course.

- Click *Assignments* under *LEARNING ACTIVITIES & ASSESSMENT* on the left.
    - An *Assignment List* is displayed which contains assignments for the course. This list includes both new and old assignments.
- Click the assignment you want to submit and upload your zipped file.
    - The top half of the assignment page includes details like the due date, grading method, and instructions. These details are determined by your instructor.
    - The bottom half of the page explains your options for submitting the assignment. You will need to upload the zipped file with all of the files for that assignment.
- When you are done, click *Submit*.
    - Labs are due by the beginning of the following lab, so you can resubmit freely if needed until the beginning of the next lab. **Note that the due date and time may be for the last lab of the week for your instructor, so what is posted in ReggieNet may not be the same as your due date!!**
- After submitting your assignment, you will receive an email notification which serves as a receipt. The email notification includes a Submission ID which can be used to verify that you submitted the assignment.
    - It would be a good idea to keep this email until your grade has been posted.