Java and Eclipse
Download/Installation Instructions

You may want to print these instructions before proceeding. You should read each step completely before performing the action it describes. Because these products/instructions may change slightly every few months, I would appreciate it if you would let me know if you find any discrepancies between these instructions and what actually happens when you try to follow them.

IMPORTANT: Install Java first, BEFORE you install Eclipse.

Java

The following instructions were composed for installing Sun Java 1.8.0_101 (aka Java SE Development Kit (JDK) 8, Update 101). Note: Java is already installed as part of Mac OS X, so no further installation is necessary if you have recently purchased one of these machines. You can determine whether you have the right version by typing

```
java -version
```

to the unix command prompt; the response should be something like **java version "1.8.0_101"** If you have an earlier number (high teens or twenties) that is fine.

A typical Java installation takes about 450 megabytes of disk space (plus the < 1 Mb download, which you should keep on your machine, in case you need to reinstall Java). The installation process reports the exact size.

Downloading

1. Click Java Downloads
2. Click the **Java Platform (JDK) 8u101/8u102 (Java Download)** button.

   A new page will be displayed (based on what operating system you are using); in this document we will assume you are using **Windows** .

3. Click the **Accept License Agreement** and then choose the file to download based on your version of windows
4. Click **Save**. A **Save As** pop-up window will appear.
5. Select the place to save this file (the **Desktop** is a convenient place; you can move it later) and then click **Save**.

6. Click **Close**.
7. Start the **Installing** instructions directly below.

If the page starting with **Download Java for Windows** (from step 2) or any other Java-related windows still appear in your browser(s), you can close/terminate it.

Installing

1. Double click the "coffee cup with steam" icon labeling the file **jdk-8u101-windows-x64.exe**.

   An **Open File - Security Warning** pop-up window will appear.

2. Click **Run**.

   A **Java Setup - Welcome** pop-up window will appear.

3. Click the **Install>** button.

   A **Java Setup - Yahoo! Toolbar** pop-up window will appear.

4. Uncheck the **Install the Yahoo! Toolbar** and click **Next>**.

   A **Java Setup - Progress** pop-up window will appear. A green progress bar will record the progress let to right; it will pause and the end and finally disappear (on my machine this took about 30 seconds).

   A **Java Setup - Complete** pop-up window will appear

5. click **Close**.

   A page with **Java** in red at the top will be displayed in your browser (it might have to bring up a new browser to display this page). Beneath the red top the page should display **Verify Java Version** in red. Beneath it is the message **Check to ensure that you have the recommended version of Java installed for your operating system.**

6. Click the **Verify Java version** red button.

   The message beneath the red top the page should display **Verified Java Version** in red. Beneath it is the message **You have the recommended Java installed (Version 8 Update 101).**

If you follow these instructions, Java will be successfully installed. To try to verify installation, click **Start** and then **Run...** and then type **C:\WINDOWS\system32\cmd.exe** in the **Run** pop-up

window (if it is not already there) and then click **OK**. In the pop-up window type **java -version** and press enter; it should print in that window **java version "1.8.0_101"** followed by two other lines.

You should keep the "coffee cup with steam" icon labeling the file **jdk-8u60-windows-x64.exe** somewhere on your computer in case you need to reinstall Java (not likely necessary).

You may **Close** or **Terminate** (**X**) any Java-related windows remaining from this process.

The installed files are typically stored (in Windows) in the folder **C:\Program Files\Java\jre8**

You may now download and install the Eclipse IDE and then test it with your installed version of Java.

Add the Java compiler to your PATH variable (Instructions for Windows 8):

- Select Start -> Control Panel -> search control panel, for "environment" -> Edit the system environment variables -> Environment Variables
- Under "System variables", find the one named "Path". Select Edit.
- Find the path to your Java installation (with help from TA). On some computers this path is "`C:\Program Files\Java\jdk1.8.0_101\bin`" but on others it is "`C:\Program Files (x86)\Java\jdk1.8.0_101\bin`".
- Without deleting the existing path, prepend your Java installation path to the beginning, followed by a semicolon. So it should look like "`C:\Program Files\Java\jdk1.8.0_101\bin`;originalpath"
- Ok, Ok

Eclipse

The following instructions were composed for installing Eclipse 3.6.2 on Windows (this version is also called **Helios**), but **the same instructions can be used to install the last version of Eclipse (Eclipse Neon)**. The process for Mac/Linux should be similar.

A typical installation takes about 300 Mb of disk space (plus the 170 Mb download, which you should keep on your machine, in case you need to reinstall it).

Downloading

1. Click Eclipse

   A page with **Eclipse Downloads** in purple will be displayed near the top of your browser.

2. If you are using some version of Windows (32-bit or 64-bit), continue below; otherwise look above the column of "green arrows" for the label **Eclipse Helios (3.6.2) Packages for** and in the pull-down list to its right, choose either **Linux** or **Mac OS X (Cocoa)**.
3. Under the heading **Eclipse Classic 3.6.2 (171Mb)** (probably the third icon from the top), click the appropriate Operating System for your machine -I'll assume a **Windows 32 bit** installation in these instructions.

   A page with **Eclipse downloads - mirror selection**, will be displayed in your browser.

4. After the label **Download eclipse-SDK-3.6.2-win32.zip from:** click what is next: typically **[United States]** followed by the name of some college or computer organization in purple (it chooses a random one each time this page is displayed to distribute the work). It might take 10-30 seconds to respond (as indicated below).
5. In the **File Download** pop-up window, click **Save**.

   A **Save As** pop-up window will appear.

6. Select the place to save this file (the **Desktop** is a convenient place; you can move it later) and then click **Save**.

   The file will be named **eclipse-SDK-3.6.2-win32.zip**.

   A pop-up window will appear showing **% of eclipse-SDK-3.6-win32.zip from ...** It should reach **100%** relatively quickly if you are using a hard-wired internet connection, more slowly if you are using wireless (my cable connection at home took about 10 minutes at .4 MB/second). Don't worry about the exact time as long as the download continues to make steady progress). At the very end a "copying" pop-up window will briefly appear and then disappear.

When it shows the **Download Complete** pop-up window, click **Close**.

- If the page with the label **Eclipse downloads - mirror selection** (from part 2) or any Eclipse-related windows still appear in your browser(s), you can close/terminate them. You should keep this file (**eclipse-SDK-3.6.2-win32.zip**) somewhere on your computer in case you need to reinstall Eclipse.

Installing

1. Unzip this file that you just downloaded.

   On my machines, I can

   - Right-click the file.
   - Move to the **WinZip** command.
   - Click **Extract to here**

   Unzipping creates a folder named **eclipse**; on my machine this took about 30 seconds. The result is a folder named **eclipse**. You can leave this folder here or move it elsewhere on your disk drive. Again, I recommend putting the file and resulting folder in the **C:\Program Files** directory.

   You should know how to use your machine to zip and unzip files.

2. Create a shortcut on your desktop to the **eclipse.exe** file in this **eclipse** folder. On most Windows machines, you can
   - Right-click the file **eclipse.exe**
   - Drag it to the desktop.
   - Release the right button.
   - Click **Create shortcut here**

   Now you are ready to perform a **one-time only** setup of Eclipse.

3. Double-click the shortcut to Eclipse that you just created. In the **Workspace Launcher** window, in the box following **Workspace:**, should appear something like **C:\Documents and Settings*username*\workspace** (where *username* is your login on the machine). If you want, you can type in (or browse) another location for the workspace file to be created, but I advise accepting the default.

   Check the box labeled **Use this as the default and do not ask again**. Aside: you will be using one workspace during the semester, checking projects in and out of this workspace. If you ever want to re-enable the display of this window, once Eclipse starts, you can

- Select **Window | Preferences**.
- Click the **+** in front of **General** or double-click **General** (after the +)
- Click **Startup and Shutdown**
- Check the box labeled **Prompt for workspace on startup**.

4. Click **OK**
5. Terminate (**X**) the **Welcome** window
   You can always get it back by selecting **Help | Welcome**
6. Click the **Window** menu, and then click the **Customize Perspective...** (the first item in the third group).

   Click the **Tool Bar Visibility** tab.

   Click the **+** disclosure icon all the way to the left of **Java Element Creation** (third from the top).
   Click the checkbox for **New Java Project**, whose contents should change to a check.
   Click **OK**

7. Terminate (**X**) the Eclipse window.

Note that if you install Eclipse on any platform(s), you will have to follow the directions related to **stdlib** there too.

Eclipse Nomenclature

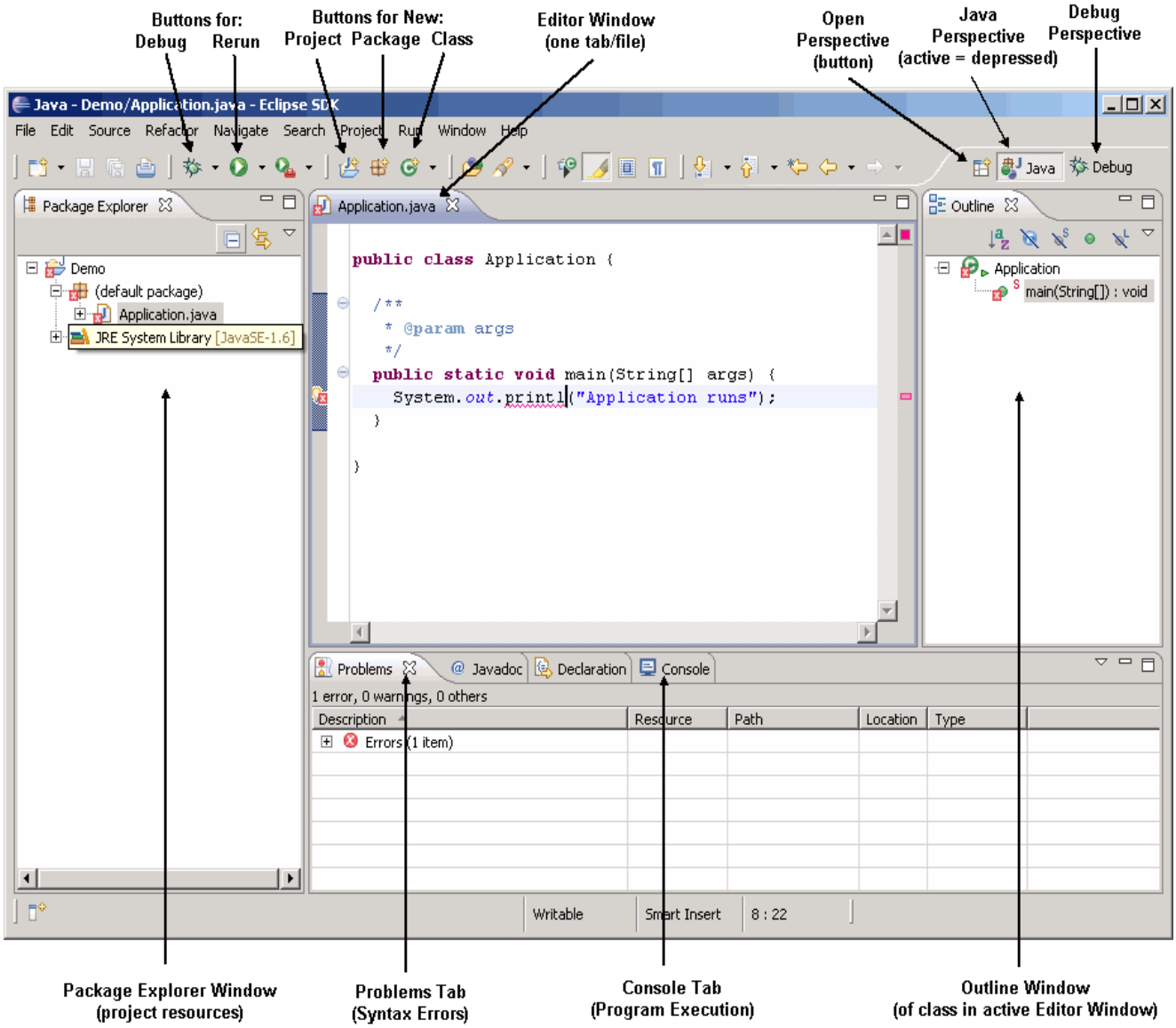
This section contains a terse description of Eclipse using **highlighted** technical terms (start becoming familiar with them) to describe its basic layout and operation. Because Eclipse is an *industrial-strength* tool, and we are using it in an academic setting (an early programming course), we will focus on its simpler aspects only. The most important terms that we will discuss and use are **workbench**, **workspace**, **perspective**, **view**, and **tool bar**.

Workbench/**Workspace**: These two terms are closely connected, to the point of having the same prefix. A **workbench** (or more accurately, a workbench window -see the window below) is the Eclipse interface to a **workspace**. A workspace is a folder that comprises a collection of files/subfolders that store the workspace's preferences (how the workbench window appears on the screen and how it displays/manipulates it contents) and projects (collections of related programming resources -primarily Java classes). We interact with a workspace -view and manipulate its preferences and projects- through a workbench window.

Preferences specify how a workbench window displays a workspace; projects specify the software that we can develop using the workbench window. In the section above, we started Eclipse and created a new, "empty" workspace; actually, it is not really empty: it stores all the standard initial preferences for the workbench window, but no projects. Then, the workbench window displayed this "empty" workspace.

So, a workspace stores information about each of its projects. It knows where all their resources are located, whether they are inside or outside the workspace. The workspace also records whether each project it contains is open or closed for use. Finally a workspace stores preferences that apply to all its projects, and to the workbench using the workspace.

Eclipse is general: we can have any number of workbench windows open, each referring to a unique workspace. For simplicity, we will always use just one workbench window, and it will always refer to the same workspace. In fact, in the following discussion we often will say "Eclipse" when we mean "workbench window": e.g. We use Eclipse to interact with a workspace. Below is an example of Eclipse using all the standard preferences, with labels affixed to many of its interesting features. The rest of this section will explain its layout and operation.



Perspective: At any given time, Eclipse displays one **perspective** of the many that it can display. Each different perspective is suited to one specific programming task. The perspective shown above in Eclipse is the **Java** perspective, which we use to write and run Java code. Notice that the name of the perspective, **Java** appears depressed (it is active) on the upper-rightmost tabs. It it followed by another tab, indicating the **Debug** perspective; we can switch to this perspective being active by clicking its tab. A third perspective that we will use, less frequently than these two, is named **Java Browsing** (not shown here).

View: Each perspective contains a variety of **views** that allow us to view, navigate, and edit information about a program. Each view is shown in a pane in the Eclipse window. So, views are not just for looking: we can use views to change information too. A view may appear as a single tab in its own window, or it may be one tab in a tabbed notebook window, containing many views, of which only one is active at a time- the top one. The **Java** perspective contains a variety of standard views. Going clockwise from the top left,

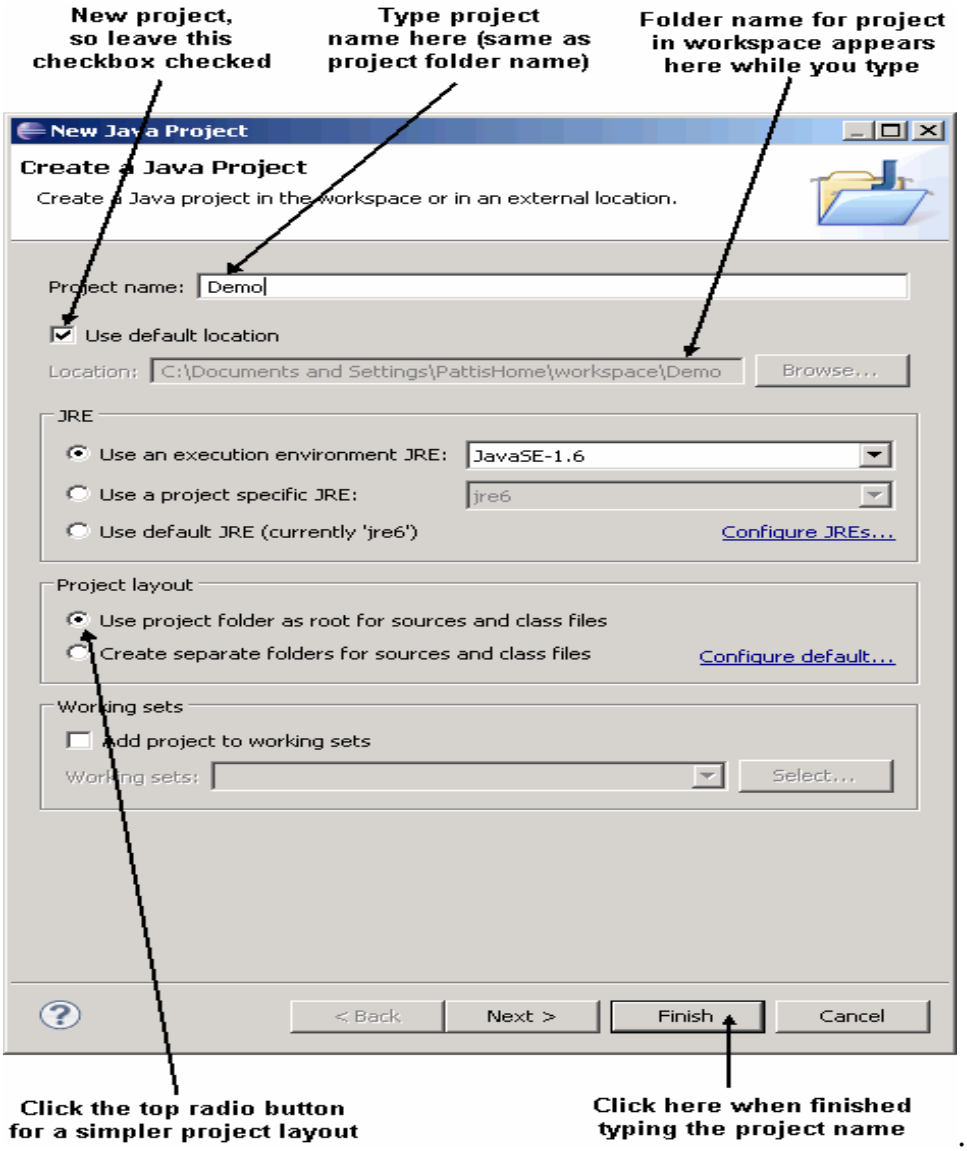
- The **Package Explorer** view is the only tab in a window that shows all the code (classes and libraries, and their files) under a project name. Here it shows a folder named Demo, a file named **Application.java** in the **(default package)**, and a connection to the **JRE System Library** we just updated with **stdlib.jar**.
- An **Editor** view is one tab (per file being edited) in a window comprising only editor views; here the only file is named **Application.java** so there is only one tab in this editor window. The tab contains the name of the resource being edited; if we hover over the tab that views a **.java** file, Eclipse displays the name of the project, the package containing the class (nothing if it is in the default package) and finally the class name.
- The **Outline** view is the only tab in a window that shows a high-level outline (mostly imports, instance variables, methods, and nested classes) of the class specified in the active editor tab. The colored icons to the left of the names specify properties: e.g., access modifiers, whether or not they are overriding an inherited method.
- The **Problems** view (it is one tab in a window, containing other tabs of which **Console** is the most important) shows a list of all the errors the Java compiler found when it tried to compile a project. The **Console** view shows the input to and output of the program when it runs.

Because these views are related, some information is propagated into multiple views: e.g., the red indicators that there are syntax errors in the code. When they disappear from one window, they often simultaneously disappear from the others.

Tool Bar: The workbench tool bar appears under the menu bar that includes the drop-down menus labeled **File**, **Edit**, **Source**, etc. The tool buttons here act as short cuts for common operations in a perspective; we can also invoke these operations with the pull-down menus, but these buttons are faster. The picture above labels the **Debug** and **Rerun** buttons, as well as the **New Java Project**, **New Java Package**, and **New Java Class** buttons (described in the next sections). The tool buttons on this tool bar change when we change the perspective. We can customize tool bars within a perspective, but we will not cover this topic here.

Starting a New Programming Project

To start a new project, click **File – New - Java Project.** Then, the following **New Java Project** window will appear.

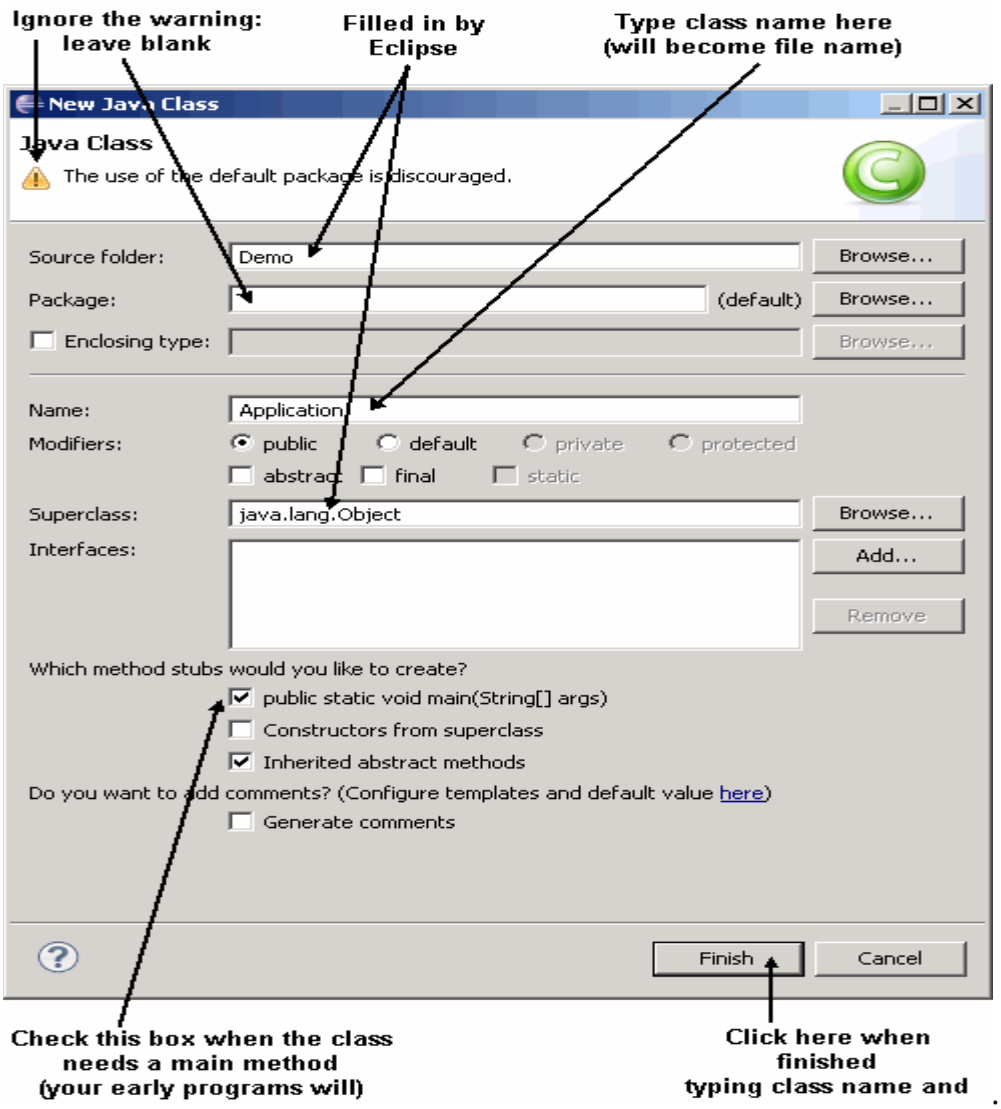


In this window, I have started creating a new project whose name is **Demo**; this project will be stored in a folder named **Demo** in the workspace. I have also clicked the **Use project folder as root for sources and class files** radio button: this is the simplest form for a project, and the one we want to use.

For a simple project, just click **Finish** when you are done typing the project name; clicking **Next** leads to another window of options, which we would default anyway and are ultimately accepted

just by clicking **Finish** here.

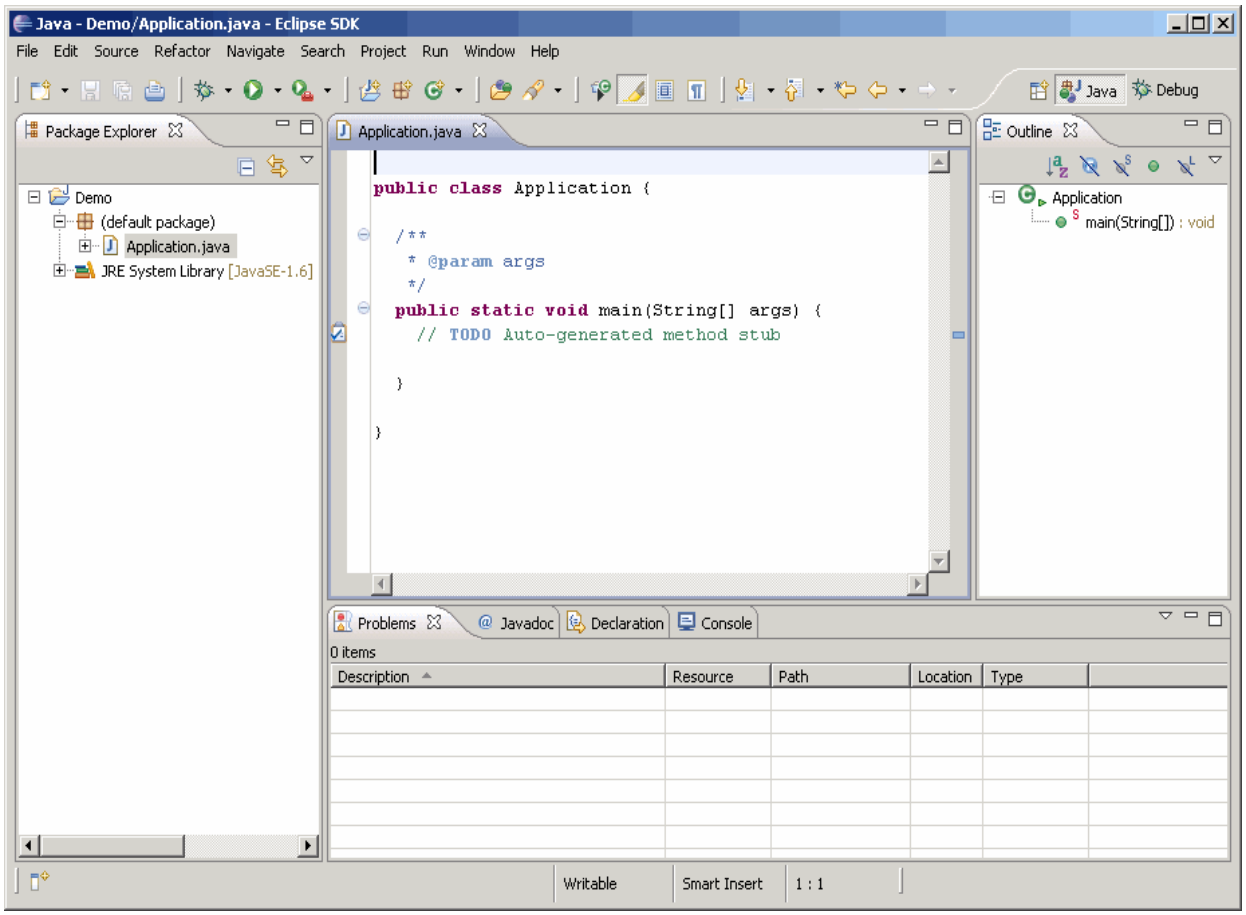
To create a new class inside this project (your programs will all be written inside classes), click the **New Java Class** button on the tool bar for the **Java** perspective. The following **New Class Window** will appear.



In this window I have created a class whose name is **Application**, inside the default package, inside the project named **Demo**. While Eclipse does not recommend using the **default** package for advanced programmers, as beginners we will use the default package for a while.

Because I put a check in the appropriate checkbox, Eclipse will automatically fill-in a **main** method for the **Application** class. Click **Finish** when you are done typing the name and checking the box.

Eclipse will update to look like the following.



Using Jar Libraries in Eclipse

This handout describes how to use/install Jar (Java ARchive) libraries/files in Eclipse. In EPL131, we will need to use/install one Jar library/file: **stdlib.jar**

Note that you will need to know the location of the standard workspace that you are using for your Eclipse projects. The name of this workspace appears in the first pop-up window (labelled **Workspace Launcher**) that Eclipse displays. The default location for this workspace is **C:\Documents and Settings*yourname*\workspace**.

Method: Installing a Jar Library in all Eclipse Projects

(Use this method to install the **stdlib.jar** library)

1. Download (right-click and **Save Target As**) **stdlib.jar** from **http://introcs.cs.princeton.edu/java/stdlib/** into the same folder that you are using as your workspace (or move this file into the workspace folder after downloading it elsewhere).
2. Start Eclipse and select **Window | Preferences**
3. Click the **+** in front of **Java** or double-click **Java** (after the +)
4. Click **Installed JREs**
5. Under the **Name** column, double-click **jre6**.
6. Click the **Add External Jars...** button.
7. In the **Jar Selection** window, navigate to your workspace folder, and double-click the file **stdlib.jar**.
8. In the **Edit JRE** window, click **OK**
9. In the **Installed JREs** window, click **OK**

You must perform these operations just once on machines that you control (like your home or portable computers). When the Jar files are installed in this way, they will be made available to every new project that you create in Eclipse.

You can check for these Jar files if you disclose the + in front of the project's name, and then disclose the + in front of **JRE System Library[jre6]**: observe the "jar" icon, the name of the Jar file to its right, and the location of that Jar file at the far right.

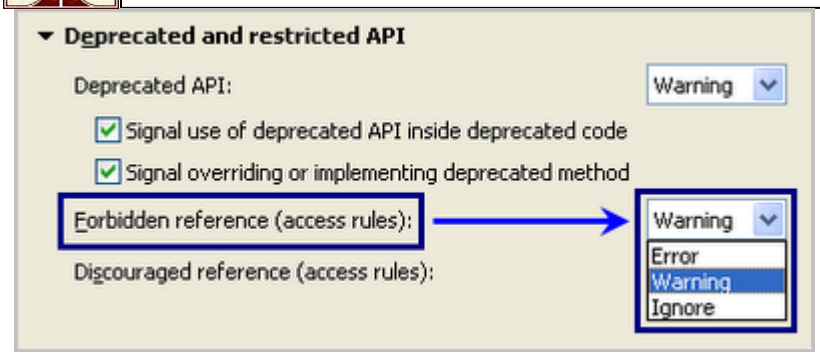
If you have the following Problem

"Access restriction: Class is not accessible due to restriction on required library"; error message may be shown while developing Java projects in Eclipse IDE. Error message is self-explanatory; some classes cannot be loaded into the project since restriction rules are being imposed on those classes.

How to solve

This error message can be removed by changing a setting inside Eclipse IDE. Open up the dialog box shown below, using any of the following paths.

- Windows -> Preferences -> Java -> Compiler -> Errors/Warnings
- (Project) Properties -> Java Compiler -> Errors/Warnings



Locate the "Forbidden reference (access rules)" option under "Deprecated and restricted API" section in the dialog box. This option decides how to handle access rules defined inside Eclipse. By default it is set to "Error" which causes Eclipse to complain about references to any restricted classes. Choosing any other option (Warning or Ignore) will remove these error messages.

Be careful

"Warning" or "Ignore" options will only hide the potential issue in the project, by allowing the project to use any classes ignoring predefined access rules. To completely resolve this issue, analyze the project and located the use of restricted classes and take necessary actions (either remove those references or access rules).

Exporting Javadocs

JAVA DOCuments are web pages (HTML files) that Java can automatically generate from Javadoc-compliant code comments. The ability to automatically generate easily readable documentation directly from the code files was a major advancement in software development technology when it came out.

- How to Write Doc Comments for the Javadoc Tool
- Javadoc tags (@xxx) -- Note: most basic HTML tags can be imbedded in Javadocs and will be interpreted correctly.

Eclipse can easily create Javadoc web pages through its "Export" capabilities:

1. Highlight the project for which you wish to generate documentation web pages.
2. On the main Eclipse menu, select **File/Export...**
3. In the Export dialog that comes up, select **Java/Javadoc** then click **Next.**
4. In the Javadoc Generation dialog that comes up, set the following options:
    1. **Javadoc command:** If this is your first time to export Javadocs, this field may be blank.
        1. Click the **Configure...** button
        2. Browse to the location of the javadoc executable which is generally located in the **bin** subdirectory of the Java JDK installation directory. Here

are some typical locations for the javadoc executable on different systems:

- **Windows**: **C:\Program Files\Java\jdk1.6.0_24\bin\javadoc.exe** (exact directory name depends on the version of the JDK installed)
- **Mac**: Separately, open a command window and type "which javadoc". This will tell you the location of the javadoc executable.
- **Linux**: Separately, open a command window and type "which javadoc". This will tell you the location of the javadoc executable. On Ubuntu, a typical executable is /usr/bin/javadoc.

3. Highlight the javadoc executable and click **Open**.

2. **Select types for which Javadoc will be generated:** Be sure that the project that you want to generate documentation for is checked, and only that project.
3. **Generate Javadoc for memebers with visibility**: Select **Private**.
4. **Use standard doclet**: Checked -- generate standard HTML documentation files.
5. **Destination**: Click **Browse...** and navigate to your project directory. Just below your project directory, create a "**doc**" directory (same level as "**src**" and "**bin**"), highlight the **doc** directory and click **OK**.
6. Click **Next** to go to the next configuration window.

5. In the next Javadoc Generation configuration window, set the following options:
   1. **Document title**: Enter a meaningful title that is pertinent to your project.
   2. **Basic Options**: All boxes should be checked (default).
   3. **Document these tags**: All boxes should be checked (default).
   4. **Select referenced archives and project to which links should be generated**: No boxes need to be checked (default).
   5. **Style sheet**: Unchecked (default).
   6. Click **Finish** as all the options in the next configuration window should be set to their defaults.
      - An Update Javadoc Location dialog may pop up. Check that the location described is indeed where you want the javadocs to be located and click "**Yes To All**".

A new tab will open in Eclipse showing the top level Javadoc web that was generated (doc\index.html). Browse through the Javadocs to make sure that what was generated matches what you thought you were generating!

Commit your project to source control, being sure to include the doc directory.

Exporting JAR Files

Java **AR**chive files are special zip files that the Java system uses to bundle up entire code directory hierarchies into an single file for easy deployment. Instead of a whole messy and complex directory structure of files, all you need is a single file to run your application or

applet. Code and any required resource files, such as images or sound clips can all be bundled into JAR files. The compiled class files are required to run the program but the source files can be optionally included as well.

To create a JAR file of your project from Eclipse, you need to "export" it.

1. Click on **File/Export...**
2. In the pop-up dialog window, select **Java/JAR file** as the export destination. Click **Next**.
3. Select the files you want to be in your JAR file. Be sure that only the project you want is selected if you have multiple projects in your workspace. You probably do NOT want any above the source directory, so be sure those files are unchecked.
4. Slecte or type in the name and destination of the JAR file you wish to create.
5. Be sure that "**Export generated class files and resources**" is checked off and optionally, "compress the contents of the JAR file". Click **Next**.
6. The default "JAR Packaging Options" are fine. Click **Next**.
7. Be sure that "**Generate the manifest file**" is checked. Either option for sealing the JAR file is fine.
8. Specify the **Main class** by typing in or browsing to find the class with the main() method. This is typically the controller in your MVC architecture. This will enable you to run the application by using just the JAR file (double-clicking it or "java -jar MyJar.jar" for you command-line aficionados). Click **Finish** to create the JAR file.