ECSE-321 Intro to Software Engineering

Software Tutorials

Prepared by Trevor Ahmedali (trevor.ahmedali@mail.mcgill.ca)
Modified by Hussein Moghnieh (hussein.moghnieh@mail.mcgill.ca) on Sep. 2005
Modified by Kangbin Wang (kangbin.wang@mail.mcgill.ca) on Sep. 2006
Modified by Jun Ouyang (jun.ouyang@mail.mcgill.ca) on Jan.2009

Winter 2009

NetBeans 5.0 IDE

Create a New Project

- Choose **File > New Project.**
- In **Categories**, choose "**General"**, and in **Projects** choose **"Java Class Library".** Click **Next**.
- Under Project Name enter `Hello_World` **,** change the project location to "`C:\my_project`". Click **Finish**.

The `Hello_World` project opens in both the Projects window and the Files window.

- Expand the Hello_World project node.
- Right-click the **Source Packages** node and choose **New > Java Class.** Name the class `myMain`**.** Click **Finish.** `myMain.java` opens in the Source Editor.(Figure 1)
- In the static main function, type the following:
  `System.out.println("hello world");`
- In the **Project Explorer**, right click on **"**`Hello_World`**"** and click **run.** Select `myMain` to be your main class.

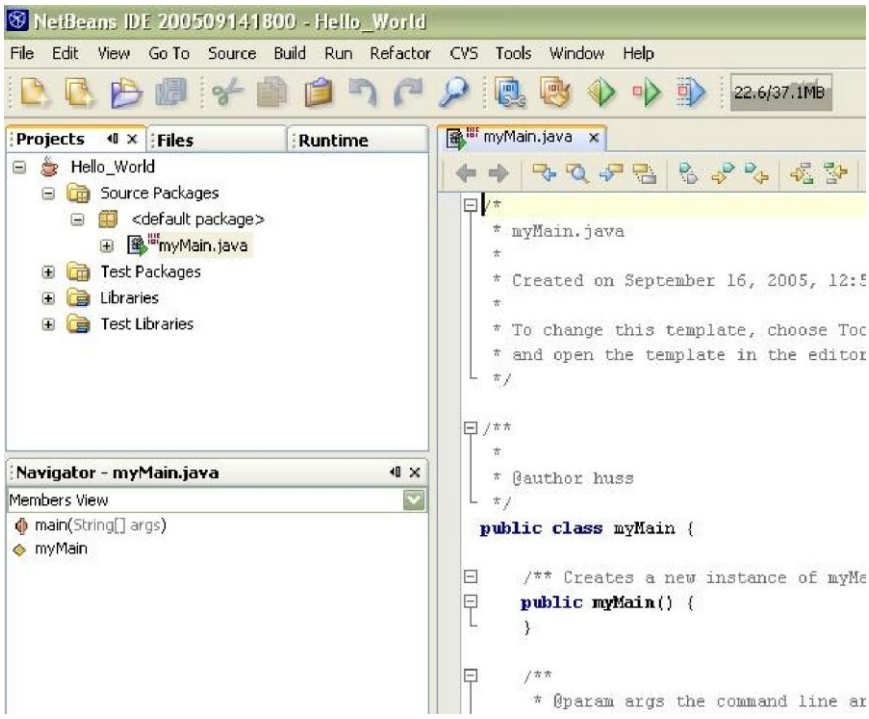

Figure 1: Creating NetBeans Project

Using NetBeans 5.0 CVS Client

Import the project to CVS (Creating new CVS Module)

- In the Project Explorer, highlight the project **Hello_World**
- Choose **CVS > Import into Repository**
- In CVS Root click Edit.
- In the Edit CVS Root box , fill as shown in Figure 2.

Figure 2. Edit CVS Root

- Click Ok. Check "Use internal SSH" and provide your UNIX password.
- Folder to import should be: **C:\my_project\Hello_World** as shown in Figure 3.
- Repository Folder, enter Hello_World

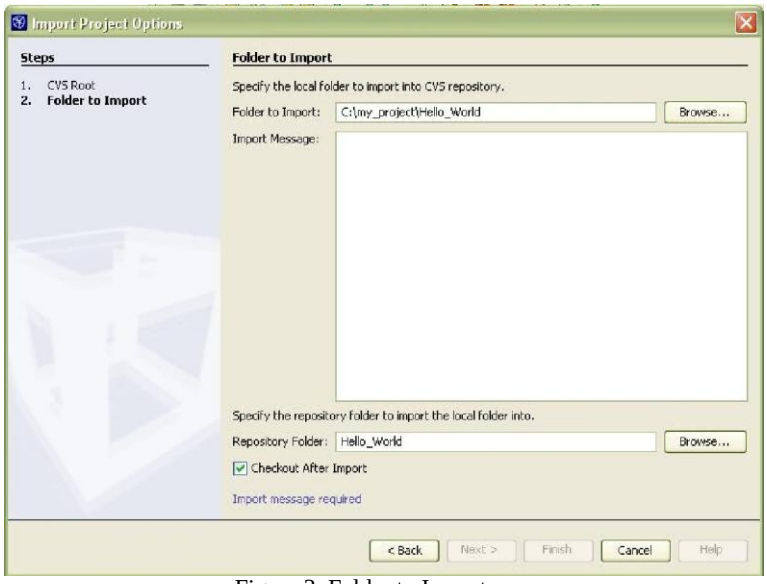

Figure 3: Folder to Import

- **Click Finish**. Your module will be imported to the CVS server, and it will be checked back to your local directory.

- Right click on any file or directory, choose **CVS** to see the various CVS command.(Figure 4)

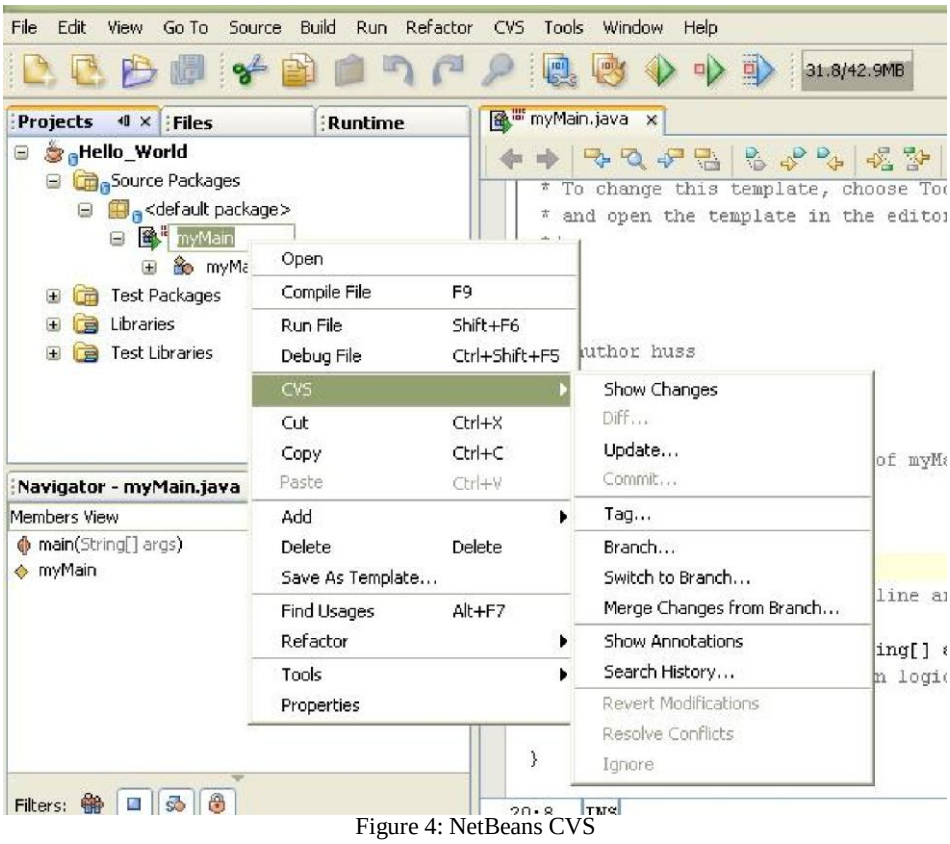

Figure 4: NetBeans CVS

Editing Source Code: Expanding Abbreviations

- Display line numbers in the Source Editor by right-clicking in the left margin and choosing **Show Line Numbers**.
- Place the cursor at the end of the main method declaration (on or near line 22) and press Enter.
- Type `sout` and press Space. The `sout` abbreviation expands to `System.out.println("");`
- Type `Hello World! It's between` the quotation marks (be sure to include a space after `It's`).
- To see or customize the list of abbreviations: ○ Choose **Tools** > **Options**.
  - ○ Click on Edit Icon and select code completion tab

Editing Source Code: Use Code Completion

- Type a space and then type `+ new` after `"Hello World! It's "`.
- Type another space and start typing the word `java`. The code completion box should open and display all of the matching packages. If the code completion box does not open, press **Ctrl-Space** to manually open it.
- Use the arrow keys to select `java` in the code completion box and press Enter. Then type a period (`.`). The code completion box should open again, listing all of the available subpackages of the Java package.
- Finish entering `java.util.Date().toString()` using code completion. When you are done, the line should read:

```
System.out.println("Hello World! It's " +
new java.util.Date().toString());
```

Editing Source Code: View Errors

- Place the cursor at the end of the main method declaration (should be line 22) and press Enter.
- Copy the following comment and paste it in the new line right above the `"Hello World!"` message

```
Print out "Hello World!" and the time
```

- After a second or so, the string is underlined with a red line and has a *red "x" icon* in the left margin, meaning that it contains errors.
- Hold the mouse over the icon to view the error.

Editing Source Code: Comment Out Code

- The line that you pasted in the last step should really be a comment and should therefore be enclosed in comment symbols. Put the insertion point at the beginning of the line and press **Ctrl-Shift-T** to comment the line out. Note that this also works if you've selected several lines at once.

Editing Source Code: Reformat Code

- The spacing for the comment line does not match the spacing for the rest of the file. Press **Ctrl-Shift-F** to reformat the entire file. If any lines are selected when you press **Ctrl-Shift-F**, only those lines are reformatted.

Editing Source Code: Create Import Statement

- Go back to the Hello World message and delete `java.util` from `java.util.Date().toString()`.
- Place the insertion point anywhere in the word `Date` and press **Alt-Shift-I**.
- In the "Fast Import" dialog box, choose **java.util.Date** and click **OK**. The import statement is created. This allows you to automatically import the correct package for a class without having to know the complete package hierarchy.

Compile the Program

- Choose **Build** > **Compile** (**F9**).
- The Output window opens at the bottom of the IDE and displays any compiler output, including compilation errors. You can double-click any error to go to its location in the source code.

Run the Program

- Choose **Build** > **Execute** (**F6**).
- The IDE runs the program and prints the output to the Output window. You should see the following message in the Output window (with a different date, of course):

```
Hello World! It's Wed Dec 10 18:15:20 CET 2003
```

Debugging: Add a Breakpoint to a File

- Click in the left margin of the file on line 27 (**Ctrl-Shift-F8**).
- A red breakpoint icon appears in the margin and the line is highlighted.

Debugging: Add a Watch to a Variable

- Right-click `Date` on line 26 and choose **New Watch** (**Ctrl-Shift-F7**).
- Make sure that the "Watch Expression" field contains `Date` and click **OK** in the dialog box.

Debugging: Start a Debugging Session

- Choose **Debug** > **Start Session** > **Run in Debugger** (**Alt-F5**).
- The IDE runs the file until the breakpoint is reached. Use the debugger views at the bottom of the IDE to monitor your program's state, such as the value of the `Date` variable (listed under both Local Variables and Watches).

Debugging: Step Through Program Execution

- Use **Step Into** (**F7**) and **Step Over** (**F8**) to execute the program one line at a time. **Step Into** will continue the debugging one line at a time *inside* an upcoming method call, whereas **Step Over** will treat the whole method call as a single statement and continue debugging *afterwards*.

Debugging: Stop a Debugging Session

- Choose **Debug** > **Finish Sessions** (**Shift-F5**).

Other Helpful Tips

- When coding applications, you can include comments with Javadoc `@todo` tags (or simply just comments with the keyword `TODO`) for things that are left to be done or that aren't working. This makes it easy for others (or yourself later on) to find the problems.
- To search documents for `TODO` tags, choose **Windows** > **To Do** from the main window. This displays a list of "to do" tasks and lets you jump straight to one by double-clicking on it.

NetBeans Links

- NetBeansTM IDE 5.0 Tutorials
  http://www.netbeans.org/kb/50/

JBuilder X IDE

Activating JBuilder X

- Before you can use JBuilder, you must first register for a free activation.
- Go to http://www.borland.com/products/downloads/download_jbuilder.html#
- In the "Keys Only" table, click the **Foundation** link.
- Click the **New User** button in the form that appears, and register for an account. Use a valid e-mail address, because the activation file will be sent to that address.
- In a few minutes, you should receive an e-mail from Borland at the address you entered. It will contain the activation file as an attachment. Save it on the desktop (or in some another folder).
- When you start JBuilder, select the option for activation file and select the file you just saved. You should now be able to use JBuilder.

Create a New Project

- Choose **File** > **New Project** to open the Project wizard.
- In Step 1 of the Project wizard, type `HelloWorld` in the "Name" field.
- Accept **jpx** as the project file type.
- In "Directory", enter a directory for this tutorial project. The end of the directory path will automatically be the same as the project name you just entered.
- Uncheck the **Generate Project Notes File** option. If you check this option, the Project wizard creates an HTML file for project notes and adds it to the project (not needed now, but you might want to do that for your project).
- Click **Next** to go to Step 2.
- Accept the default paths in Step 2. Note where the compiled class files, project files, and source files will be saved.
- Click **Next** to go to Step 3.
- Type `Hello World` in the Title field of the Javadoc fields.
- Fill in the following Class Javadoc fields as well: **Description** (a short description, e.g. "This is the Hello World tutorial"), **Company** (McGill or your project group name), **@author** (your name, obviously). The information in the class Javadoc fields appears in the project HTML file and as optional header comments in the source code.
- Click **Finish**. A file, `HelloWorld.jpx`, is generated by the wizard and appears in the project pane located in the upper left of JBuilder's IDE.

Generating Your Source Files

- Choose **File** > **New** to open the Object Gallery.
- Select **General** on the left-hand part of the dialog box.
- For now, we will select a simple class, but later on you can use the other options for your course project (*Application*, for example, will let you build a graphical interface for your project): select **Class** and click **OK**.
- In the Class Wizard, enter `helloworld` (it must be lowercase, according to Java coding conventions) under "Package", if it isn't already there.
- In "Class name", enter `HelloWorldClass`.
- Check **Public** and **Generate main method**.
- Uncheck **Generate default constructor**.
- Click **OK**. The wizard creates the class file you specified and places it in your project. It appears as a node in the Project pane, under the package you specified.

Editing Source Code: Use Code Completion

- If the `HelloWorldClass.java` file isn't already open, locate it in the Project pane on the left-hand side of JBuilder and double-click it to open it.
- Place the cursor at the end of the main method declaration (should be line 13) and press Enter to create a new line.
- Type:

```
String str;
System.out.println("Hello World! It's " + str);
```

- After `String str` (should be line 14), type a space and then type = `new` after `String str` but before the semicolon.
- Type another space and start typing the word `java.` (note the period). The code completion pop-up box (called "CodeInsight" in JBuilder) should open and display all of the matching packages. If the CodeInsight pop-up box does not open, press **Ctrl-H** to manually open it.
- Use the arrow keys to select `util` in the pop-up and press Enter. Then type a period (`.`). The CodeInsight pop-up should open again, listing all of the available classes in the `java.util` package.
- Finish entering `java.util.Date().toString()` using code completion. When you are done, the line should read:

```
String str = new java.util.Date().toString();
```

Editing Source Code: View Errors

- Place the cursor at the end of the main method declaration (should be line 13) and press Enter.
- Type the following comment in the new line right above the "`String str`" statement (it should be line 14):

```
Print out "Hello World!" and the time
```

- The string is underlined with a red line and has a *red exclamation mark icon* in the left margin, meaning that it contains errors. Note also that the error is displayed in the **Errors** folder of the **Structure** pane (bottom-left window pane).
- Hold the mouse over the text that's underlined in red to view the error.
- **Note:** if the error icon shows a yellow wrench beside the exclamation mark, it means JBuilder thinks it might be able to fix the error itself (this is *not* the case with the error we just made, though). In such a case, left-click the error icon in the left margin to see possible solutions (the solutions are not guaranteed to be correct, however!).

Editing Source Code: Comment Out Code

- The line that you typed in the last step is a comment and should therefore be enclosed in comment symbols. Put the insertion point at the beginning of the line (should be line 14) and press **Ctrl-/** to comment the line out.
- Note that this also works if you've selected several lines at once. Furthermore, pressing **Ctrl-/** on a line(s) will un- comment it if it was already a comment.

Editing Source Code: Reformat Code

- If the spacing or indentation for the comment line does not match the spacing for the rest of the file, you can choose **Edit** > **Format All** to reformat the entire file.
- To change the formatting rules used by JBuilder (indentation, braces, spacing, etc.), you can go to **Project** > **Project Properties** and select **Formatting**.

Compile the Program

- Choose **Project** > **Make Project "HelloWorld.jpx"** (**Ctrl-F9**) to compile the project.
- If there are errors or warnings, the Messages window opens at the bottom of the IDE and displays any compiler output, including compilation errors (this might be in a tab called "Build HelloWorld.jpx"). You can double-click any error to go to its location in the source code.

Run the Program

- Choose **Run** > **Run Project** (**F9**).
- The first time you run a project, a dialog box called "Runtime Configurations" will come up. You must create a new configuration for your project. Click **New**.
- Enter in a name for the configuration (or leave the default).
- Make sure "Type" is set to **Application**.
- Beside "Main Class", click the button with the ellipses (**…**) and select **HelloWorldClass**, then press **OK**.
- Press **OK**.
- Press **OK** to exit the "Runtime Configurations" dialog box.

- You only had to do that because it was the first time running the project. Now, choose **Run** > **Run Project** (**F9**) again to run the program.
- The IDE runs the program and prints the output to the Messages window at the bottom of the IDE. You should see the following message in the Output window (with a different date, of course):

```
Hello World! It's Tue Sep 07 09:18:46 EDT 2004
```

Debugging: Add a Breakpoint to a File

- Click in the left margin of the file on line 16 to add a breakpoint (**F5**).
- A red breakpoint icon appears in the margin and the line is highlighted.

Debugging: Start a Debugging Session

- Choose **Run** > **Debug Project** (**Shift-F9**).
- The IDE runs the file until the breakpoint is reached. Use the debugger views at the bottom of the IDE (in the Messages window) to monitor your program's state, such as the value of any variables you have put watches on (see next step).

Debugging: Add a Watch to a Variable

- Right-click on `str` on line 15 and choose **Add Watch** (**Ctrl-Shift-F7**). **Note:** unlike NetBeans, you can only add a watch to a variable once the debugger has started!
- Make sure that the "Expression" field contains `str` and click **OK** in the dialog box.
- You can now view the value of `str` at the current point of execution by going to the "Data watches" tab on the left side of the Messages window. This can be very useful when you want to know the value of a variable at a certain point in the program.
- If the variable is an object (as in our case), rather than a primitive data type (like an `int` or `double`), then you can expand its entry in the "Data watches" view to show the values of its members.

Debugging: Step Through Program Execution

- Use **Step Into** (**F7**) and **Step Over** (**F8**) to execute the program one line at a time. **Step Into** will continue the debugging one line at a time *inside* an upcoming method call, whereas **Step Over** will treat the whole method call as a single statement and continue debugging *afterwards*.

Debugging: Stop a Debugging Session

- Click the **Reset Program button** (looks like a red square "stop" button) (**Ctrl-F2**) at the bottom of the Messages pane.

Other Helpful Tips

- When coding applications, you can include comments with Javadoc `@todo` tags for things that are left to be done or that aren't working. This makes it easy for others (or yourself later on) to find the problems. On the line where you want to enter the comment, just write `todo` and press **Ctrl-J**, and then JBuilder will format it into a proper Javadoc comment.
- To search documents for `@todo` tags, choose **Search** > **View Todos** from the main window. This displays a list of "to do" tasks and lets you jump straight to one by double-clicking on it.

JBuilder Links

- Introducing JBuilder:
  http://info.borland.com/techpubs/jbuilder/jbuilderx/introjb/contents.html
- JBuilder complete documentation:
  http://info.borland.com/techpubs/jbuilder/jbuilderx/index1280x1024.html

Eclispe 3.4 IDE

Installation

Download Eclipse Ganymede from the website http://www.eclipse.org/ and unpack it to any directory. No installation procedure is required. Eclipse requires an installed Java as of version 1.5 It is recommended to use Java 6 (also known as Java 1.6).

Create A Project in Eclipse.

- Choose **File > New Project.**
- In **Select wizard**, choose **Java Project,** click **Next.**
- In the **Project Name:** type your project name **:** `Hello_World`
- Click **"Create Project from existing source"** and browse to your local drive**,** create a directory and call it**:** `my_eclipse_project`. Click **Finish**
- Expand the Hello_World Project Node
- Right click and choose **new Class.** Make the Class name **"**`myMain`**".** Select the option: **public static void main(String[] args)**. Click **Finish**
- Write the following command in the static main method:
  `System.out.println("Hello World");`
- Right click the `Hello_World` project and choose**, run as: Run.**
- Double click on **Java Application**. A new run configuration will be created. In the **main class** edit box, click on the search button and choose "myMain"
- Click **Run**.

Running Eclipse Project

Run configuration

A *run configuration* is a set of guidelines that Eclipse uses for running a particular Java program. A particular run configuration stores the name of a main class, the values stored in the main method's args parameter, the JRE version to be used, the CLASSPATH, and many other facts about a program's anticipated run.

- Select a class that contains a main method.
- On Eclipse's menu bar, choose **Run>Run**.
- In the Configurations pane (on the left side of the Run dialog), double-click the Java Application branch. Click on **Search** button next to the Main Class edit box and select `myMain` from the list. (Figure 5)
- Click **Run**.

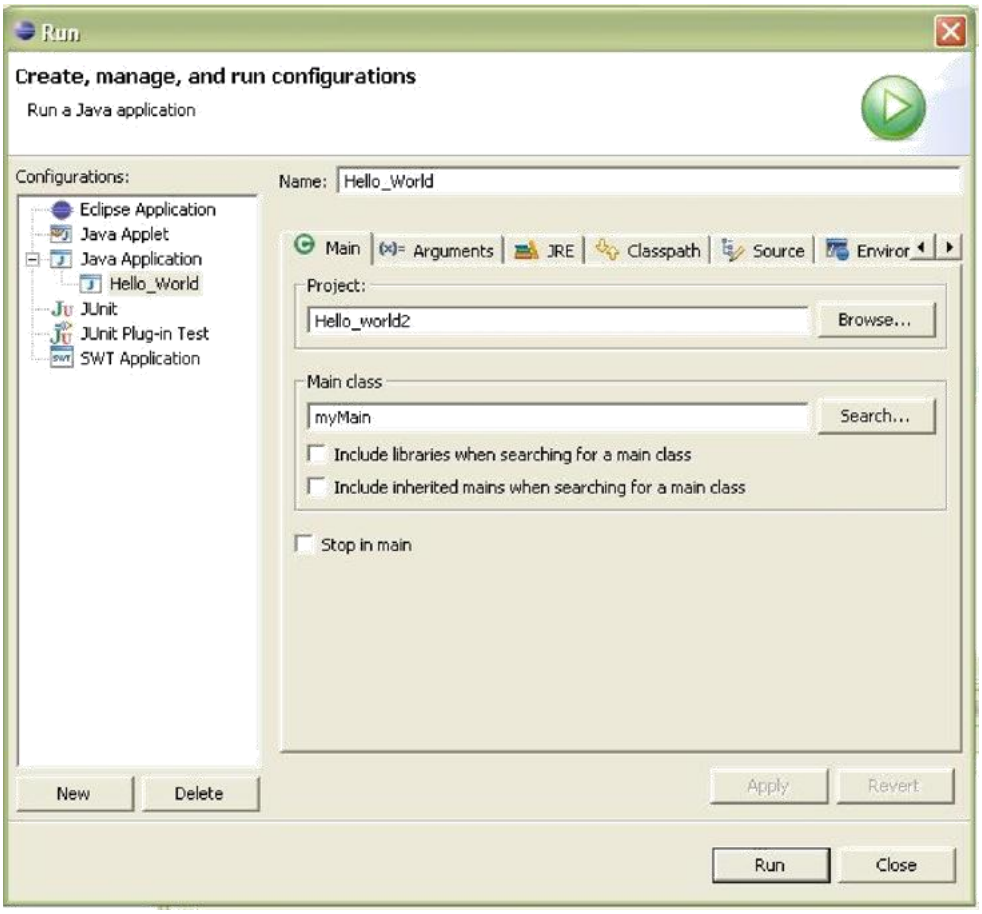

Figure 5: run configuration

Note: Now that you've run the HelloWorld program and run configuration, that configuration shows in the Run tool's dropdown list for easy access. As you add more configurations, they will be added as well, each with a number. The one marked number "1" will be the one selected if you just hit the Run tool button itself. (Figure 6)

Fig 6: running Java Project

Eclipse Code Template (Code completion)

- In the static main function, write : `sysout` and click **Ctrl + space** Sysout will be replaces by: `system.out.println();`
- To view the list of ready made template and add you own template. Click **Window->Preferences->Java->Editor->Templates**

Debugging in Eclipse

Modify the main method with the following source.

```
int count = 3;
for (int i = 0; i < count; i++)
{
System.out.println("Hello" + i);
}
```

- Right-click in the editor and select **Save**.
- Right-click on the debug column to the left of the for loop. Select **Toggle Breakpoint**.

A small blue ball will appear (Figure 7) next to the line that has a breakpoint. However, setting a breakpoint is not sufficient to debug a program. You must also run the program in debug mode. This is done by using the Debug tool rather than the Run tool.

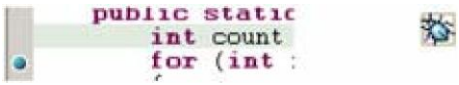

Figure 7: Adding a breakpoint

- Click the **Debug** tool button. The debuger perspective window will open (Figure 8)

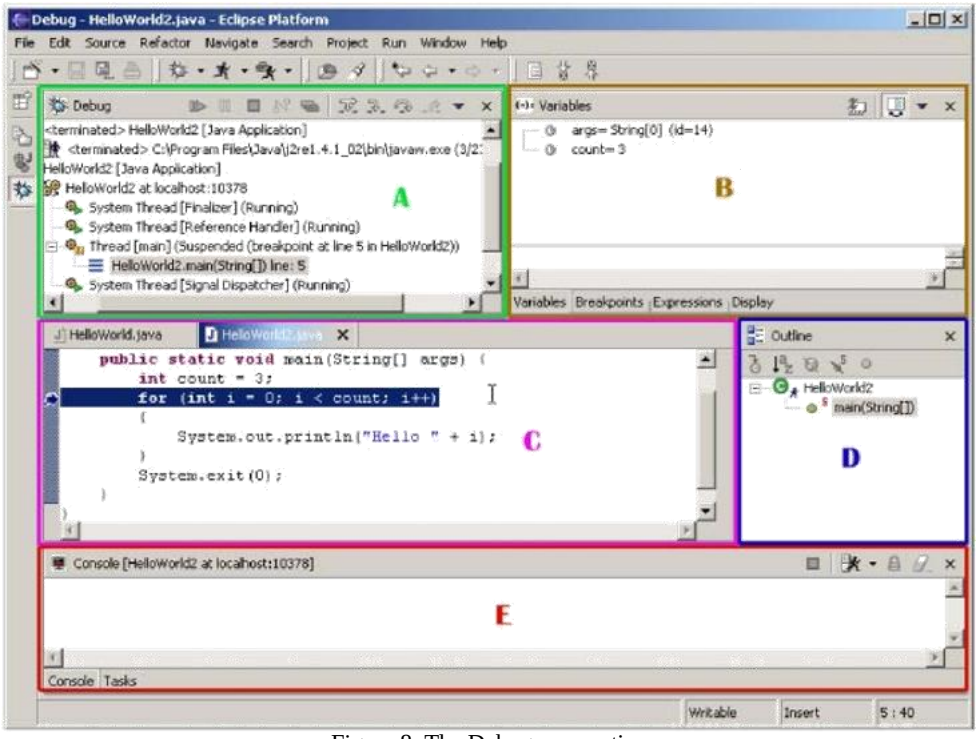

Figure 8: The Debug perspective

A. The Debug window, which shows the state of all processes, living or dead, on the machine.(Figure 9)
B. The Display window. This particular window has several stacked views that you can use to interact with the program. By default, it shows the Variables view.(Figure 10)

C., D., & E. The Editor, Outline, and Console views. These are actually all from the Java perspective, just resized and repositioned a bit.(Figure 8)

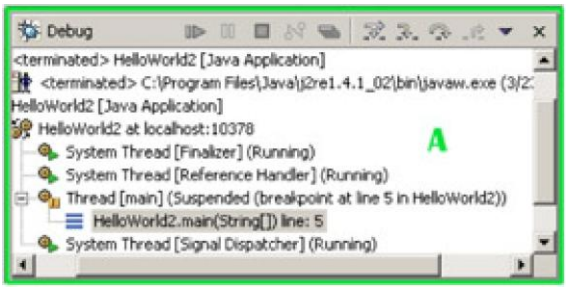

Figure 9: The Debug view.

The Debug view shows the various processes running (or no longer running). view) by a small blue arrow.

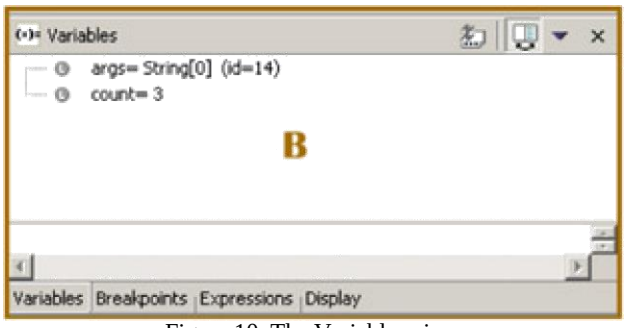

Figure 10: The Variables view.

This view shows the variables. You can not only inspect the contents of any variable; you can also interact with them, changing the contents.

Eclipse Helpful Tips

- To comment one or multiple lines, select the lines and type : **Ctrl + /**
- To comment a paragraph type: **Ctrl + Shift + /**
- To Reformat the source code: Select the code you want to format and press **Ctrl+Shift+F**
- Import code completion: Write the following statement:
  `System.out.println("Time:"+new  Date().toString());`
  On the left margin, a red X appears signaling an error. Left Click on it and it will list the different options to solve the error. Choose "Import 'Date'(java.util)".

For more information, please refer to the following link.

http://www.vogella.de/articles/Eclipse/article.html

Javadoc

Introduction

Javadoc is the Java programming language's tool for generating API documentation. Java API documentation describes important elements of your code, such as methods, parameters, classes, fields, and so forth. You can insert special Javadoc comment tags into your code so that they will be automatically included in the generated documentation. Describing your code within the code itself rather than in a separate document helps to keep your documentation current, since you can regenerate your documentation as you modify it.
A **doclet** is a program that uses the Javadoc tags in the source code to produce documentation. The *standard doclet* that comes with Java produces HTML documentation (exactly like the Java API webpages).

Using Javadoc Comments in Your Source

Javadoc comments are comments that start with **`/**`** and end with **`*/`** (note the extra leading asterix). You can include Javadoc comments in the source code, ahead of declarations for any class, interface, method, constructor, or field. You can also create doc comments for each package and another one for the overview, though their syntax is slightly different.

Documentation comments are recognized only when placed *immediately before* class, interface, constructor, method, or field declarations. Documentation comments placed in the body of a method are ignored.

A doc comment is composed of a main description followed by a tag section: the *main description* begins after the starting delimiter `/**` and continues until the *tag* section. The *tag* section starts with the first block tag, which is defined by the first @ character that begins a line (ignoring leading asterisks, white space, and leading separator `/**`).

Comments are written in HTML: this means that text will be interpreted as HTML code, so you can use HTML tags.

Summary sentences: the first sentence of each doc comment should be a summary sentence, containing a concise but complete description of the declared entity.

Javadoc Tags

Here are some useful Javadoc tags to use in your Javadoc comments:

- **`@author`** ***name*****:** Use this to enter your name. Can be used in overview, package, or class Javadoc comments.
- **`@param`** ***parameter-name description*****:** Use one of these tags for each parameter in a method's Javadoc comment, where *parameter-name* is the name of the parameter as it appears in the method declaration, and *description* is a concise description of it (description may use multiple lines).
- **`@return`** ***description*****:** Adds a "Returns" section with the *description* text. This text should describe the return type and permissible range of values. This tag is valid only in a doc comment for a method.
- **`@throws`** ***class-name description*****:** Used to describe an exception of *class-name* (e.g. `IOException`) thrown by a method. Can only be used in method Javadoc comments.
- **`@version`** ***version-text*****:** Adds a "Version" subheading with the specified *version-text* (e.g. `1.0`) to the generated docs when the `-version` option is used. This tag is intended to hold the current version number of the software that this code is part of. It can be used in overview, package, or class Javadoc comments.

Running Javadoc from the Command Line

The javadoc program will be located in the same place as the rest of Java on your machine.

You can run it on a whole package or subpackage (e.g. `helloworld.utils`):

`javadoc [-sourcepath path] [-classpath path] [packages]`

Or you can run it on specific source files (e.g. `C:\user\*.java`):

`javadoc [-sourcepath path] [-classpath path] [source_files]`

The **–sourcepath** option is used to provide the path to the start of the package. For example, suppose you want to document a package called `com.mypackage` whose source files are located at: `C:\user\src\com\mypackage\*.java`. In this case you would specify the sourcepath to `C:\user\src`, the directory that contains `com\mypackage`, and then supply the package name `com.mypackage`:

```
javadoc -sourcepath C:\user\src com.mypackage
```

The –classpath option is used when your files reference other classes outside of their package. For example, if you want to document com.mypackage, whose source files reside in the directory `C:\user\src\com\mypackage`, and if this package relies on a library in `C:\user\lib`, you would run:

```
javadoc -classpath \user\lib -sourcepath \user\src com.mypackage
```

Note that javadoc has more available options, but you probably won't need those. You can find their descriptions at http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/javadoc.html#options.

Javadoc in NetBeans

- To generate the javadoc files:
  - Build > Generate Javadoc for "your project name"
- To check for missing javadoc comments:
  - Right-click any source in the package exolpoer select **Tools** > **Auto Comment**.
  - The list on the left shows all entities that should have comments. The icons indicate if the comments are incomplete (yellow icon) or missing (red icon). The box at the bottom-left describes the problem.
  - You can fill in the Javadoc comments on the right of the window.
  - Use the **Refresh** button to re-check your changes.
- The generated javadoc are stored in "<project home>\doc\javadoc" directory within your project. Click on *index.html* to view the javadoc.

Javadoc in JBuilder

JBuilder handles all the compiling of the Javadoc information when you build your project. It also includes a few handy shortcuts to using Javadoc with your project:

- To start a Javadoc block comment, position the cursor the line above the method or class you want to document, and type `/**` then press Enter. JBuilder automatically creates a Javadoc comment block and fills it in with the relevant Javadoc fields for you to fill out.
- You can also accomplish the same thing by right-clicking on the method or class you want in the editor window and selecting **Edit Javadoc for …** from the menu.
- When you're in a Javadoc comment block, type `@` then a pop-up box will appear listing the possible Javadoc fields for you to use.
- To insert a Javadoc "@todo" comment, you simply need to write **`todo`** and then press **Ctrl-J**. JBuilder turns that line into a proper Javadoc @todo comment so you just have to type in the item's description.
- When Javadoc HTML files are made (i.e. when you compile your project), you can view them by selecting the **Doc** tab at the bottom of the editing window of the file you want to see.

Javadoc in Eclipse

To generate Javadoc in Eclipse:

- Highlight your project name in Package Explorer.
- Click **Navigate ->Generate Java Doc**.
- In the **Wizard**, click **finish** to accept the default settings

To view the generated java docs. Expand the **doc** folder in the **package explorer**, and right lick on `index.html`, choose open in the system browser.

Javadoc Links

- Javadoc Tool Homepage: http://java.sun.com/j2se/javadoc/
- How to Write Doc Comments for Javadoc (official Sun conventions): http://java.sun.com/j2se/javadoc/writingdoccomments/index.html

WinCVS

Introduction

CVS is a version control system. It tracks the history of changes to your source files, so you can easily go back to an earlier version. It is also useful for working in groups, since you don't have to worry about overwriting each other's code. Each developer works on their own local copy of the code, in a ***working directory*** (also called ***sandbox***), and CVS will merge the changes into a central location, called a ***repository***.

A user can ***checkout*** a file from the repository to edit in his working directory. When finished, the user can send his version back in to the repository (called ***committing*** or ***merging***). To indicate that he is no longer working with the file, he can ***release*** or ***checkin*** the file. Finally, if the version on the repository is newer, the user can ***update*** his local files to include the new changes.

WinCVS is a specific client program for working with a CVS repository. Since CVS itself is actually a command-line program, WinCVS just provides a GUI front-end for it.

Configuring WinCVS

- When you start WinCVS for the first time, the WinCVS Preferences dialog box should appear. If it doesn't, go to **Admin** > **Preferences**.
- In the **Global** tab, uncheck "**Prune empty directories**".
- In the **WinCvs** tab, check "External diff program" and enter in the following path: C:\Program Files\Microsoft Visual Studio .NET 2003\Common7\Tools\Bin\windiff.exe
- Click **OK**.

Logging in to the Server

- Before any CVS commands can be issued to the CVS server, you must first login. Go to **Admin** > **Login**.
- In Login Settings menu, check "**CVSROOT**" and click on the right button after the edit box.
- In protocol box, choose "SSH". Repository Path, write: */CVSREP/<group directory>*. i.e. */CVSREP/softeng0* . The group directory will be given to you during the tutorial session.
- In username, enter your UNIX username and password. In hostname enter: *Xenon.ece.mcgill.ca* . Click Ok to save the settings.
- Look at the messages in the output window to see if you logged in correctly. If it was successful, it will say something like:

```
cvs -q login
(Logging in to user@server)
*****CVS exited normally with code 0*****
```

- If it was unsuccessful, it will look something like:

```
cvs -q login
(Logging in to user@server)
cvs [login aborted]: authorization failed: server
snow rejected access
*****CVS exited normally with code 1*****
```

Creating a CVS Repository

- Each group will have to initialize their repository **only once**. Meaning , CVS will add special directory called **CVSROOT** in */CVSREP/<group directory>*
- You must first be logged in. Go to **Create** > **Create new repository**.
- Verify the parameters in the **General** tab and click **OK**.

Creating a New (Empty) Module

Files and directories (folders) can only be added to an existing module in the repository. Before any files or folders can be added to the repository, it is therefore necessary to create at least one module. The easiest way is to import an empty folder into the repository.

- Create an empty folder somewhere. Create a dummy text file in the directory (WinCvs doesn't allow an empty directory to be checked out)
- Go to **Admin** > **Command Line**.
- Check CVSROOT and make sure your login settings are written in the Edit box next to it.
- Check the box that reads "**Click to specify in which local directory the command applies**".
- Click the **Change folder** button and select the empty folder you created.
- In the "Enter a cvs line command" field, type:
  `cvs import –m "Create Module" module1 vtag rtag`
  (where `module1` is the name of the new module)
- Click **OK**.
- The module will be created on the UNIX server and the file is added. A good practice is login to your UNIX account using any SSH tool and go to the directory */CVSREP/<group directory>* to see the changes made so far.
- You need to check out the module you just created to your local drive to make the modification. Follow the next step to checkout the module

Checkout a Module

- You must first know the name of the module you want to check out.
- Choose **Create** > **Checkout module**.
- In the "**Checkout settings**" dialog box, enter the name of the module to be checked out. (module1)
- Fill in "**Local folder to checkout to**" with the local directory you want to store the module in. It can be any directory or simply the "C:\" directory

- In the "**Checkout settings**" tab, you can use the "**Checkout (sticky) options**" to retrieve versions other than the most recent (for example, if you need to go back to an older version because of a new bug).
- Click **OK**.
- You can now edit the checked out files in the local directory you selected, using NetBeans, JBuilder or Eclipse.

Updating code before Committing

Before you commit a file, you should update to make sure you have the latest version as the one in the CVS repository.

- Right-click the file you want to commit, and choose update.
- If the file on the server is newer than your local copy , winCVS will try to merge both versions. If winCVS failed to do the merging automatically, due to conflict in the same line of code, proceed to the next step to resolve the conflict.

Checking Diffs Prior to Committing

Before you commit a file back into the repository, you may want to verify what's different between your local copy and the repository copy.

- Right-click the file or folders you want to check, and choose **Diff selection**.
- In the "**Diff settings**" dialog box that appears, check **Use the external diff** to use the graphical diff packaged with Microsoft Visual Studio (you already set it up when you configured WinCVS). Otherwise, WinCVS uses a text diff, whose output is similar to the Unix command `diff`.
- Click **OK**.

Commit/Merge a File or Folder

- Select the file or folder you want to commit back to the repository.
- Choose **Modify** > **Commit**.
- Enter a log message that describes the changes you made since you last checked out the file/folder.
- If you selected a folder to commit, but you don't want to commit its subfolders, check **Do not recurse**.
- If you want to force the commited file/folder to be a certain version, you can go to the "**Commit options**" tab and check **Force revision/branch** and then enter a version number.
- Click **OK**.
- Note that you can only commit files if their original version was the most recent version from the repository. For example, if I check out version 1.2 of a file, and someone else checks it out and commits it (making that version 1.3), then I can't commit my 1.2-based file until I update the module first.

Update Your Local Working Directory

- Prior to running the update command, it may be useful to know which files are out of date. The Query Update command lists the actions that will occur if update were to be run on the currently selected file or folder. To run it, go to **Query** > **Query update** (**F4**). Filenames listed with a "U" in front of them indicate which files will be overwritten by newer files checked out from the repository.
- If this is okay, go to **Modify** > **Update Selection** (**Ctrl-U**) to update the currently selected file or folder to the newest version (if there's a newer version in the repository).
- The "**Update settings**" dialog box will appear. Most of the options aren't useful and should be unchecked, but check **Create missing directories** (in case new directories were added to the repository).
- If you definitely want to overwrite your code with whatever's the most recent version in the repository (even if the local code was newer), then check **Get the clean copy**.
- Click **OK**.

WinCVS Links:

- WinCVS homepage: http://www.wincvs.org
- WinCVS Daily Use Guide: http://ikon.as/wincvs-howto/
- WinCVS 1.3 Quick Start Guide: http://www.devguy.com/fp/cfgmgmt/cvs/startup/