Section 1 - Welcome

1A.1.1 Java

Programming is the science and art of making computers do useful, entertaining, and sometimes destructive tasks.

To write a ***program*** you must start with a ***computer language***, and one of the most prevalent languages today is **Java**. There are, of course, others like **C++, Python, Ajax, Visual Basic** and **Perl,** and each has its strengths. So why learn **Java**? It is highly portable. You can develop a program -- even one with a ***GUI*** (Graphical User Interface) -- on your Mac, and I can run it on my Windows or Unix computer. This makes it ideal for Internet use and commercial development. In fact, this cross-platform capability is a major reason why Java has seen an explosion in popularity in recent years.

By the end of this course you'll be able to say that you're a ***computer programmer*** -- a **Java** programmer, in particular. Of course, you won't know everything there is to know about **Java** by a long shot, but you will be able to do many things with the language, and you'll be able to converse intelligently about computer programming.

Like all programming languages, **Java** has evolved since it was first introduced. Today we use **Java 1.7** (also called **Java 7.0**). If you followed the instructions included in the Module ***Obtaining Compilers***. you have everything you need to write and run **Java 7.0** programs. In about one or two hours from now, you will have written and executed your first **Java** program.

Section 2 - Reading

1A.2.1 Reading for This Week - and Every Week

Everything you need to succeed in this course is contained in these modules, this course site, and a reference text which can either be my recommended text or any Java text you choose. Each week visit the module titled Course Syllabus (page 2) to find out what I expect you to read for that week.

1A.2.1.1 Lesson Module Reading

Every week you have at least two modules to read, the *A module*, and the *B module*. This week, that means:

- module 1A
- module 1B

Read these modules twice, because that's what it takes to digest the material. However, don't stop there, because there's more...

1A.2.1.3 Handout Modules

In addition to the usual two lesson modules each week, the first couple weeks there are a number of handout modules. These appear either near the beginning or the end of the modules list. Check the Course Schedule to see which handout modules are required each week. I'll give you a hand and list the required handouts for this week, but starting next week, you'll need to get that information from the Course Schedule module. This week, in addition to the two lesson modules, read:

- Syllabus
- Reference 1R

1A.2.2 Try Out Every Program On Your Compiler

In most pages, you'll see programs or code fragments. They will be in blocks like this:

```
System.out.println( "Hello World!");
```

You can and should select the code with your mouse, copy it and paste it into a program in your IDE. Compile and run it, to see what it does.

Sometimes, the code won't fit completely into the code box on my web pages, in which case you will see scroll bars:

```
public class Foothill
{
   public static void main (String[] args)
   {
      double rent, total, cost_this_year;
      int year;
      final double INITIAL_RENT = 1700;

      rent = INITIAL_RENT;          // our initial rent
      total = 0;                    // keeps track of how much we paid "so far"

      // loop for five years
      for (year = 1; year <= 5; year++)
      {
         cost_this_year = rent * 12;
         total += cost_this_year;

         // now increase the rent in prep for the next year
         rent *= 1.05;
      }
      System.out.println("\nYou will have paid $" + total + " after five years\n");
   }
}
```

If that happens, simply select all the code inside that box and paste it into your Java IDE or text editor to examine it fully.

Either way, I expect you to compile and run the samples as you read the modules. Ask questions if you don't understand the behaviour of the code when you run it. Do this before looking at the assignment for that week.

Exception: During the first week only, do not try to compile anything until you have read module 1B and have successfully followed the instructions on running your first program.

1A.2.1.2 Textbook Reading

I do not absolutely require textbook reading. In fact, if you are having trouble or are confused, the best thing to do is to read *only* the weekly modules (remember: read them twice), and ask questions here in the forums. However, if you feel you understand the modules pretty well, then you should supplement your understanding by reading the textbook. Read the modules for the current week here at the course site, first, before referring to your text.

The way to read the textbook is to

1. read all introductory and early chapters, and
2. find the topics covered in the modules, then look up those terms in the textbook index.

1A.2.2 How Reading Relates to Assignments

While it is important to have a good reference like this text, everything you need to do **Option A** of the assignments is contained in these modules.

Important:

Read both modules A and B before attempting the assignment.

Do not even look at the assignment until after you have read the week's modules, twice. Remember, paste and run my examples into your IDE, and make sure you can get them to compile and run. Then, ask questions about anything in modules or examples.

If you do not ask me any questions about the modules, then I will assume you understand everything in them.

1A.2.3 Resources for the Entire Course

Here are some reading resources that you can use over the entire quarter.

Recommended Style Booklet

Another source of style information is the recommended booklet, ***The Elements of Java Style*** listed in the syllabus. I recommend that you get that booklet and start to read it.

Web Resources

Besides this lecture and the text, another source for documents for this course comes from Oracle:

- Master Java Documentation Page: http://docs.oracle.com/javase/7/docs/index.html

The page may not be useful for a couple weeks, but it would be a good one to bookmark now, while you are getting set up.

Section 3 - Hello World!

1A.3.1 The Joy of Programming

You have your Java ***Integrated Development Environment***, or ***IDE***, installed, and you are ready to go. Now what?

Now it's time to have some fun.

This week we'll write a very simple Java program just to get things moving.

1A.3.2 Hello Foothill

Here is the program that you need to think about. We'll call it the ***Hello World!*** program:

```
public class Foothill
{
   public static void main(String[] args)
   {
      System.out.println("Hello World!");
   }
}
```

This will be part of your assignment this week, and you will start by getting this small program running, *and only then*, adding more to it, as specified in the assignment description. When we get to Module 1B you'll need to copy and paste the above code into your ***Java IDE*** and get it to run. But before doing that we will spend today's module, Module 1A, talking only about the concepts.

note:

Other than downloading the Eclipse compiler, which you have already done, you will not actually use it to run a program until *Module 1B,* where I show you exactly how it's done.

Here in Module 1A, I will talk about the ***Hello World!*** program, explain what it means and what it does. After that, I'll help you ***compile***, ***run*** and ***capture the output*** in Module 1B. Only after participating fully in Module 1B will you be able to start actual programming. There, I'll walk you through the steps to make it all work under the assumption that you are running an ***Eclipse IDE***.

1A.3.3 The Java Language and Java IDEs

The Java language is ***platform independent***, which means a program like the one above should run on any computer (platform). However, getting it to run means typing it (or pasting it) into a Java ***IDE*** like **Eclipse**, and using the controls of that ***IDE*** to modify, compile, debug and run it. These ***IDEs*** all work differently.

As you know, the ***IDE*** we are using in this class is **Eclipse**. There are other ***IDEs***, such as **NetBeans** and **JBuilder**, but if you use those you'll have to get support for those other ***IDEs*** from the supplier. Also, students can post questions and answers about these other IDEs in the Discussion areas if you want to help each other out.

NEXT:

- **We Study** the source code for ***Hello World!***, the little 7-line program written above.

- **We Discuss** the meaning of ***compiling***, ***running***, ***capturing source*** and ***capturing runs*** of this program. We do so in the abstract, without reference to any specific ***IDE***.
- **We Show** how this program is actually ***entered***, ***compiled***, ***run*** and ***captured*** in **Eclipse**.

Let's dive in.

Section 4 - The Anatomy of a Program

1A.4.1 The Hello World! Program, Dissected

Remember that we are trying to write a program that makes the computer do something useful and/or entertaining (let's stay away from destructive for now).

A Java program is made up of one or more ***classes***, at least one of which that has a **main()** ***method***. For the first few lectures, we will only have one ***class*** that I will call "Foothill" that has only the one **main()** ***method***.

For now, don't worry about what ***classes*** are, just that a Java program basically looks like this:

```
<import statements>

<class name>
{
   <data>

   <class method #1>
   <class method #2>
}
```

So you see that a simple program has some ***import statements*** followed by a ***class***. That class has some ***data*** and some ***methods***. Each of these so-called ***methods*** takes the following form:

```
<method header>
{
   <method body>
}
```

Look at our first program with an added (optional) import line at the top:

```
import java.io.*;

public class Foothill
{
   public static void main(String[] args)
   {
      System.out.println( "Hello World!");
   }
}
```

We see that there is only one **import** statement, no ***data***, one ***class***, and within that class, one ***method***, called **main()**. Let's forget about the import statement for now and get on to the meat: the one ***method***, **main()** in our **Foothill** class.

We see that the ***method header*** is:

```
 public static void main(String[] args)
```

and the ***method body*** consists of the statement:

```
   System.out.println( "Hello World!");
```

For now, think of *every* program as a set of program statements which are completely contained inside this **void main()** method which is inside this **Foothill** class.

Although the name of the class can change (**Foothill**, **Sample**, **Exploder**, etc.), there will always be a special method, **main()**, inside it for the next several weeks.

Every main() program, for now, will look, basically, like this:

```
   public static void main(String[] args)
   {
     <program statement>;
     <program statement>;
     <program statement>;
     <program statement>;
   }
```

In the ***Hello World!*** program's **main()** method, there is only one statement:

```
   System.out.println( "Hello World!");
```

If the program had more statements, each one would be terminated by a semicolon:

```
   int time;

   System.out.println( "Hello World!");
   System.out.println( "La Femme Nikita");
   time = 24;
   System.out.println( "Jack Bauer has " + time + " hours left." );
```

Take a guess at what ***Hello World!***'s one and only statement does:

```
    System.out.println( "Hello World!");
```

If you guessed that it prints the words "**Hello World!**" on the computer screen, congratulations! I haven't put you to sleep yet.

Section 5 - Alternate Form

1A.5.1 (Optional) Alternative Form of Hello World For Some Compilers

(**Eclipse** users can skip this section.)

Some compilers require a somewhat more complex version of your ***Hello World!*** program because the output window disappears before anyone has a chance to appreciate the run. Try the simpler version that was presented earlier, first. You can skip this section for now and only come back to it if you have trouble seeing the output of your run. If you do have trouble, and you just seea ****flash**** instead of a nice output window, you can try a second form of the program:

```
import java.io.*;

public class Foothill
{
   public static void main (String[] args)
   {
      // this needed to enable a pause before termination
      Foothill f = new Foothill();

      System.out.println( "Hello World!");

      f.Pause();  // sometimes needed to hold window up
   }

   // ------------ a method to pause execution until --
   // ------------ user hits [return] -----------------
   void Pause()
   {
      BufferedReader in
         = new BufferedReader(new InputStreamReader(System.in));
      try
      {
         String line = in.readLine();
         System.out.println(line);
      }
      catch (Exception e)
      {
      }
   }
}
```

Ignore the definition of the **Pause()** method for now. We won't explain that until we get to class methods in a couple sessions. What you are seeing, however, is a class, still **Foothill**, that has two methods, rather than our original one **main()**. The second method is a hand-made trick to force the screen to pause until you, the user, type in an enter on the keyboard. You can use the time to do a copy/paste, thus capturing a copy of the program run *before* you hit the enter key to terminate the program.