# SE-1011 Lab 2: Our First Java Program

## Due by 23:00 on Tuesday, September 20, 2011

# 1. Objectives

- Carry out the basic steps required to create a working Java program, using the Eclipse integrated development environment.
- Use Eclipse to create a Java program containing a main class with a **`main`** method.

# 2. Assignment

## *2.1. Overview*

In this lab, you will use the Java program authoring tool **Eclipse** (which you have installed previously) to create a simple Java program. When you use Eclipse, you begin by creating a project that contains the files that constitute your Java program. Eclipse can have many projects open simultaneously, but in this lab you'll only create and work on a single project called **Lab2**.

A video showing how to construct a new project is available on the course website. When you are completed with this step, in your development workspace you will have a directory named Lab2 created which will contain two subdirectories, namely src and bin. (We'll learn more about what these mean later.)

Recall from lecture that, as a general rule, every Java program must have a *main class* - that is, a class that contains at least one method named **`main`** - in order to be able to run. When you run a Java program - say, by double-clicking on a program icon - the Operating System (Windows, in our case) and the Java Runtime Environment (JRE) send a "run" message to the program by calling the **`main`** method in the *main class*. The *main class* can have any reasonable name.

## *2.2. Problem description*

You are to write a program which will calculate a student's GPA. A student should be assumed to be taking four classes. The program must ask the student for their name, and details about the names of the 4 courses, their grades, and the credits for each of the 4 courses. The program should generate a screen execution matching that shown in Figure 2.

Your grade point average (GPA) is calculated by dividing the total amount of quality points earned by the total amount of credit hours attempted. Your grade point average may range from 0.0 to a 4.0. For example, a score of an A will give you 4 quality points, a B will give you 3 quality points, a C will give you 2 quality points, and a D will give you 1 quality points. The quality points for each class is multiplied by the number of credits for each course. A sample of this is provided below.

| Example Student Transcript: Jacques Costeaux | | | |
|---|---|---|---|
| Course | Credit Hours | Grade | Quality Points |
| Biology | 3 | A | 12 |
| Biology Lab | 1 | B | 3 |
| English 101 | 3 | C | 6 |
| Mathematics | 3 | F | 0 |
| **10 Total Credit Hours Attempted** | | | **21 Total Quality Points** |
| | | | **21 Quality points / 10 credits = 2.10 GPA** |

Figure 1 Sample Transcript

Enter your name:
*Jacques Costeaux*
Enter the name for course 1.
*Biology*
Enter the number of credits for Biology
*3*
Enter the number of quality points (A = 4, B=3, C=2, D=1, F=0) for Digital Logic Design using TTL devices
*4*
Enter the name for course 2.
*Biology Lab*
Enter the number of credits for Biology Lab
*1*
Enter the number of quality points (A = 4, B=3, C=2, D=1, F=0) for Software development 1
*3*
Enter the name for course 3.
*English 101*
Enter the number of credits for English 101
*3*
Enter the number of quality points (A = 4, B=3, C=2, D=1, F=0) for Modern Literary Marvels of the 1700's
*2*
Enter the name for course 4.
*Mathematics*
Enter the number of credits for Markov Models for Mathematical Minors
*3*
Enter the number of quality points (A = 4, B=3, C=2, D=1, F=0) for Markov Models for Mathematical Minors
*0*
Jacques Costeaux 's GPA is 2.10

**Figure 2: Sample program execution.**

# 3. Assignment Specifics

The first step of this assignment is to draw an algorithm showing the steps necessary to calculate a student's GPA for four classes. This can either be written as an algorithm or shown visually as a flowchart.

Once this step has been completed, you are to translate this algorithm into a Java application. For right now, all code will be written within the main method.

# 4. A note on writing Java code

Java code may look like English, but it's still a computer language that is primarily designed to be understandable by a computer. When well-written, it is easy for a person to understand what the instructions are doing as well. Good software developers observe certain rules when writing Java code as part of good software development practices - so get started on the right track. Here are some basic rules for writing Java instructions that you should follow for starters:

- Use logical-sounding identifiers (nouns or noun phrases) for your **`variables`**.
- Put **comments** inside your Java program more fully explain what you're doing.
- Indent your code appropriately. Eclipse has an auto-indent command called "Correct Indentation" in the "Source" menu - ask if you have trouble using it. There also is a Format operation which behaves similarly in the same menu.

Finally, **test** your program and make certain that it works correctly and conforms to all of the requirements stated above.

# 5. Lab Deliverables

Submit the following materials in electronically using the course upload script on the instructor's website:

1. A lab report (in pdf format) containing the following
    a. A diagram showing your program flow. This can be hand drawn and scanned or developed using Visio. If you are unable to scan or use visio, this can be submitted in hardcopy to the instructor.
    b. A short description of what went wrong and what went right during the lab.
    c. A description of what you learned from this lab.
    d. A capture of your programming running. To obtain this, run your program in Eclipse and copy the output of the window to Word by right clicking on the selected text.
    e. A copy of the Java Source code you wrote. Code must be commented as appropriate.
2. A zip file containing the java project you developed.

If you have any questions, consult your instructor.

# Appendix A: JavaDoc standards

All source code submitted to Dr. Schilling should meet the minimum documentation standards outlined below.

- The beginning of each source file should contain:
  - The name of the file/class.
  - The date it was originally written.
  - The original author.
  - The course for which it was written.
- All Java classes must have a "class description" comment in javadoc style with an @author tag.
- All fields must be declared individually and have an associated javadoc comment.
- All methods must be proceeded by a javadoc comment that makes appropriate use of the following tags:
  - @param
  - @return
  - @throws
- Within each method, documentation should be provided for any code whose purpose is not immediately obvious to someone with your current level of programming knowledge.

An example of this JavaDoc standard is given on the next page.

```
/*
 * Course: SE1011
 * Section: 2
 * Term: Fall 2008
 * @author Brittany Spears
 * Assignment: Lab 2 : The squariator
 * Date: 09/09/08
 * This class implements lab 2, which is a program that prints out the square of
 * the numbers between 1 and 5.
 */

package edu.msoe.se1011.lab2.schilling;

/**
 * Class description goes here.
 * @author     Firstname Lastname
 */
public class HelloWorld extends HelloCountry {
    /* A class implementation comment can go here. */


    /** counter variable comment */
    private static int counter;

    /**
     * classVar2 documentation comment that happens to be
     * more than one line long
     */
    private static Object classVar2;

    /**
     * ...constructor Blah documentation comment...
     */
    public static void main (String[] args) {
        // ...implementation goes here...
    }

    /**
     * ...method doSomething documentation comment...
     * @param someParam description
     * @return Returns the number of widgets
     */
    public int doSomething (Object someParam) {
        // ...implementation goes here...
        return i;
    }
}
```

# Appendix B: Preliminary Grading Rubric

| | Weight Factor | Rubric Score | |
|---|---|---|---|
| Algorithm Design | 2 | | Submitted Algorithm<br>- Algorithm is neat and legible<br>- Algorithm correctly describes a solution to the given problem<br>- Algorithm shows all steps and does not skip any steps |
| Lab Report | | | |
| | 1 | | Things Gone Right / Things Gone Wrong<br>- The problems encountered during the lab described appropriately<br>- The areas of the lab which did not pose problems described properly |
| | 1 | | Material learned<br>- Material learned from the lab clearly delineated and explained.<br>- Details of new concepts which are better understood explained |
| | 1 | | Output<br>- Output shows proper operation of the program under normal usage<br>- Output matches assigned functionality |
| Source Code | | | |
| | 2 | | Comments<br>- Step by step comments describe the algorithm used to solve the problem<br>- Algorithm properly translated into comments |
| | 1 | | JavaDoc<br>- File includes overall JavaDoc comments which list the author, course, date, and program description<br>- Method(s) include JavaDoc matching standard |
| | 1 | | Variables<br>- Variables declared properly<br>- Variable types appropriate for problem<br>- Variable names properly reflect variable usage<br>- Variables have appropriate cases. |
| | 1 | | Formatting<br>- Indentation of file correct<br>- Bracketing of program correct |