[1. Explain programming terminology 2](#_Toc99278858)

[LS1 Explain what a program is 2](#_Toc99278859)

[1.1.1 What is a program 2](#_Toc99278860)

[LS2 Explain the different generation of languages; high-level, low-level and machine language 2](#_Toc99278861)

[1.2.1 Levels of Programming Languages 2](#_Toc99278862)

[LS3 Explain the different classifications of higher level languages; procedural and object oriented programming 3](#_Toc99278863)

[1.3.1 Language Classification 3](#_Toc99278864)

[LS4 Explain the difference between compiling and interpreting a program 4](#_Toc99278865)

[1.4.1 Translating Levels 4](#_Toc99278866)

[LS5 Describe the characteristics of Java 4](#_Toc99278867)

[1.5.1 Characteristics of Java 4](#_Toc99278868)

[Reading 5](#_Toc99278869)

[Check Your Knowledge 7](#_Toc99278870)

[2. Develop a Java Program using Java Tools and Styling Conventions 8](#_Toc99278871)

[LS1 Describe the process that is followed to create and execute a computer program 8](#_Toc99278872)

[LS2 Describe source code, bytecode and object code 10](#_Toc99278873)

[2.2.1 Overview of developing and running a program 10](#_Toc99278874)

[2.2.2 Source file 12](#_Toc99278875)

[2.2.3 Compiling 12](#_Toc99278876)

[2.2.4 Running bytecode 12](#_Toc99278877)

[LS3 Create a program 13](#_Toc99278878)

[2.3.1 Create a program 14](#_Toc99278879)

[2.3.2 Output Syntax 14](#_Toc99278880)

[2.3.3 Example 15](#_Toc99278881)

[2.3.4 Simple Program 16](#_Toc99278882)

[2.3.5 Create Source File 1 17](#_Toc99278883)

[2.3.6 Create Source File 2 25](#_Toc99278884)

[2.3.7 Reading 26](#_Toc99278885)

[LS4 Compile the program using a compiler 29](#_Toc99278886)

[LS5 Run the program using an Integrated Development Environmnet (IDE) 30](#_Toc99278887)

[LS6 Explain the difference between Semantic and Syntax errors 31](#_Toc99278888)

[LS7 Describe the styling conventions 45](#_Toc99278889)

[LS8 Use the style conventions 45](#_Toc99278890)

[3. Perform elementary programming 46](#_Toc99278891)

[LS2 Explain Variables 46](#_Toc99278892)

[LS3 Create a program that declares, initializes and assigns variables 46](#_Toc99278893)

[LS4 Create a program that uses type casting 46](#_Toc99278894)

[LS5 Create a program that uses named constants 46](#_Toc99278895)

[LS6 Create a program that displays output 46](#_Toc99278896)

[LS7 Create a program that can read variables of different types from the keyboard 46](#_Toc99278897)

[4. Use a debugging tool 46](#_Toc99278898)

[5. Create a program that uses strings and mathematical library routines 46](#_Toc99278899)

[6. Create a program that uses operators and decision statements 47](#_Toc99278900)

[7. Create a program using repetition structures 47](#_Toc99278901)

[8. Create a program using methods 47](#_Toc99278902)

[9. Troubleshoot a defective program 47](#_Toc99278903)

1. Explain programming terminology

LS1 Explain what a program is

1.1.1 What is a program

In this course, we will be using Java. Java is a programming language used to create programs. Programs are instructions for the computer to execute. A program written in java may be an application (runs on the computer), or an applet (runs in a browser over the internet).

LS2 Explain the different generation of languages; high-level, low-level and machine language

1.2.1 Levels of Programming Languages

Computers do not understand human languages; they have their own language. There are over one thousand different languages used to program computers. There are also several different levels, or generations, of languages.

The 1st generation language is Machine language, and it is binary (0's and 1's). It is code directly understood by the computer hardware (e.g. registers, central processing unit [cpu], and memory [ram] locations), but is not easily understood by people. Different computer hardware may have different behavior for the same instruction code as codes are unique to each cpu. Here is an example of what a machine language instruction looks like:

**01010100**

The 2nd generation of languages (2GL) are Assembly languages. They are low level languages, and have cryptic English-based commands that are difficult for people to understand. Assembly languages are nearly the same as Machine language, but need a slight translation (assembly) to be understood by the machine. A 2GL instruction looks something like:

***mov sales, total***

The 3rd generation languages (3GL) are higher level languages that are more "English like" than assembly language. They are easier for people to write and understand than the lower level languages. Examples of 3GLs include: Basic, Fortran, Pascal, Cobol, Visual Basic, Delphi, C, C++, C#, Java, and many others. These languages can be procedural: they allow us to tell the computer how to perform a task. Any 3rd generation language must be translated into a lower level before it can be understood by the machine. The translation can be done by either compilation or interpretation (these terms will be explained in more detail later in the course). Here is an example 3GL statement:

totalSales = totalSales + monthlySales;

Each statement can result in hundreds of machine instructions to accomplish the task. Fourth generation languages (4GL) are languages used for accessing databases or supporting artificial intelligence (AI). They tend to be declarative rather than procedural; that means they are used to state what is to be the outcome rather than how to determine the outcome. SQL , prolog, lisp and Powerhouse are example 4GL languages. A 4GL instruction looks like:

EXTRACT ALL FROM CLIENTS WHERE PREVIOUS SALES > 500

There are several development efforts to come up with the next, or 5th, generation of programming language(s). Such languages are to be natural language bases that are capable of understanding "English like" statements such as:

Give me details on those clients who ordered more than $500 in equipment last month.

or

What will be the payment on a mortgage of $350,000 over 25 years at 3.25% annual interest?

LS3 Explain the different classifications of higher level languages; procedural and object oriented programming

1.3.1 Language Classification

Higher level languages are classified as eitherprocedural or declarative. Procedurallanguages are either standardorobject-oriented.

## **Procedural**:

* language used to create a program composed of a series of statements (in the language chosen), which tell the computer how to perform a specific task
* the programs have one starting point and one ending point
* the steps are followed in a linear fashion, from beginning to end
* examples are C, Fortran, Pascal, and Basic

## **Object Oriented**:

* an extension of procedural language that models the 'real world' in terms of objects. It combines the data and its actions together into a single entity.
* emphasizes the reuse of code (known as objects), that are accessible from libraries. It supports software reuse rather than continual reinvention of code.
* not linear, rather, it is driven and controlled by the actions taken by the user
* examples are C++, C#, Python and Java

LS4 Explain the difference between compiling and interpreting a program

1.4.1 Translating Levels

It is necessary to translate a higher level of language (3GL, 4GL, 5GL) that is easy for people to read and understand, to a lower level language (1GL, 2GL) the computer can execute. Such translation is performed by compiling and/or interpreting.

## Compile

Definition: Translating a program written in a high level language into low level instructions before the program can be executed.

* The program you write as a programmer in a high level language is referred to as the source code. The low level instructions that result from compiling the source code are referred to as the object code.
* The compiled object code (also known as an executable) can be run repeatedly without having to recompile the source code.
* An example of a language that uses a compiler to translate its code is C.

## Interpreter

Definition: A language using an interpreter instead of compiler to translate code the computer can execute. The main difference with interpreted languages is that the results are not saved permanently but executed by the computer and forgotten.

* An interpreter is a simpler program than a compiler.
* A program written for an interpreter reads one instruction at a time, and converts each instruction - after it is read - into machine language, which the computer then executes.
* An example of a language that uses an interpreter to translate its code is Basic.

LS5 Describe the characteristics of Java

1.5.1 Characteristics of Java

C++ was designed for systems and applications programming (i.e. infrastructure programming), extending the procedural programming language C, which was designed for efficient execution. To C, C++ added support for object-oriented programming, exception handling, generic programming and the C++ Standard Library which includes generic containers and algorithms (the Standard Template Library or STL), and many other general purpose facilities.

Java is a general-purpose, concurrent, class-based, object-oriented programming language that is designed to minimize implementation dependencies. It relies on a Java virtual machine to be secure and highly portable. Java is a statically typed, event-driven object-oriented language that uses a syntax similar to (but incompatible with) C++. It includes a documentation system called Javadoc and a unit testing environment called JUnit.

The following list provides brief descriptions of the various characteristics of Java:

* object-oriented - centered on objects and is event driven
* interpret and compile - source code is compiled into bytecode, and the java run-time environment translates the bytecode into the machine language of the target machine
* reliable - does not support direct manipulation of pointers which could overwrite memory and corrupt data, and forces the programmer to write code to deal with run time exceptions
* secure - mechanisms were implemented to prevent java applets (a special kind of program) from damaging or corrupting the computer system
* portable - one program will run on any platform for which there is a run-time environment (Windows, OS/2, Macintosh, UNIX, IBM mainframes,...). There are no platform-specific features in Java.
* high-performance - interpreting bytecode is not as efficient as a compiled language, but improvements are constantly being made (such as just-in-time compiling (JITC), and optimizing frequently used code to boost performance)
* multi-threaded - performs several tasks simultaneously
* dynamic - at run-time, Java loads classes when they are needed and permits modification of existing classes

Reading

**Introduction to Java Programming**  
comprehensive version  
by Y. Daniel Liang. Prentice Hall, New Jersey.

ISBN: 978-0-13-376131-3 ( 10th Edition)

Earlier versions of the textbook would be acceptable for COSC180 BUT NOT for the follow on course COSC 190  
ISBN: 0-13-293652-6 (9th Ed.) or 0-13-213080-7 (8th Ed.)

|  |  |  |  |
| --- | --- | --- | --- |
| **COSC180 Reading Table** | | | |
| **Topic** | **10th Edition** | **9th Edition** | **8th Edition** |
| Explain programming terminology | 7-12 | 2-16, 1261-1264 | 2-11, 1319-1322 |
| Algorithms | 34 | 34-35 | 24-25 |
| Create a Java program | 12-18 | 16-22 | 11-16 |
| Variables | 35, 40-43, 44-45 | 40-46 | 29-33 |
| Constants | 43 | 43, 48-49 | 31-32, 35-36 |
| Input and output | 37(input), 38(output), 45-46(input), | 17, 112-115 (output) 37-40 (input) | 11, 95-97 (output) 26-29 (input) |
| Strings | 386-388 | 68-69, 336-350 | 50-51, 302-313 |
| Operators | Math Operators: 41-43,, 44-48, 54-58 Math Precedence: 50-51 Relational Operators: 76-77 Logical Operators: 93-96 Operator Precedence: 104-105, 1268-1269(Operator Precedence Chart) | 44-68, 115-117, 82-84, 101-106, 1256-1257 | 32-50, 97-98, 72-73, 88-91, 1314-1315 |
| if statements | 78-80 | 84-89 | 74-78 |
| if else statements | 80-81 | 89-91 | 79-80 |
| Nested if statements | 81-83 | 91-93 | 80-81 |
| switch statements | 100-102 | 108-111 | 93-94 |
| Conditional operator | 103-104 | 111-112 | 95 |
| for loops | 170-173 | 146-149 | 126-128 |
| while loops | 158-168 | 134-144 | 116-124 |
| do while loops | 168-170 | 144-146 | 124-126 |
| Nested loops | 176-177 | 152-153 | 129-130 |
| Defining a method | 204-206 |  |  |
| Predefined methods | 206-208 | 197-201 | 172-175 |
| Nothing passed in, nothing returned | 209-209 | 178-180 | 156-157 |
| Data passed in | 212-215 | 183-189 | 160-165 |
| Value returned | 209-211 | 189-193 | 165-168 |
| Overloading | 219-222 | 193-196 | 168-170 |
| Scope of variables | 222-223 | 196-197 | 171-172 |
| Defining and declaring | 322-329 | 296-308 | 264-274 |
| Encapsulation | 344-347 | 319-322, 375-379 | 283-286, 347-351 |
| this | 356-357 | 373-375 | 346-347 |
| Constructors | 329-329 | 296-298, 303 | 265, 270 |
| References | 347-351 | 304-308 | 270-274 |
| Instance variables | 330-333 | 304, 312-317 | 271, 278-281 |
| Static variables | 337-342 | 312-317 | 278-281 |
| Visibility modifiers | 342-344 | 317-319, 1258-1259 | 282-283, 1316-1317 |
| Wrapper classes | 380-384 | 393-397 | 476-479 |
| Packages | 343 | 317-318, Also see [Supplement III.G "Packages"](http://www.cs.armstrong.edu/liang/intro9e/supplement/SupplementPackage.doc) on companion website | 282, Also see [Supplement III.G "Packages"](http://www.cs.armstrong.edu/liang/intro8e/supplement/SupplementPackage.doc) on companion website |
| Arrays | Single Dimensional: 246-250, 256-260 Foreach Loop: 251 Multi Dimensional: 288-293 | 224-245, 264-280 | 198-216, 236-251 |

# 

Check Your Knowledge

## Check Your Knowledge

### Crossword Puzzle

Complete the crossword, then click on "Check" to check your answer. If you are stuck, you can click on "Hint" to get a free letter. Click on a number in the grid to see the clue or clues for that number.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** |  |  |  | **2** |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  | **3** |  | **4** |
|  |  | **5** |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **6** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **7** |  |  |  |  |  |  |  |  |  | **8** |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **9** |  |  |  |  |  |  |  | **10** |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | **11** |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **12** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **13** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **14** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

 Check

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Across: | | | **1.** | Cryptic, English-based language | | **5.** | A characteristic of Java; prevents programs from damaging or corrupting the computer system | | **7.** | The person who designed the first version of Java | | **8.** | An example of a 4GL | | **9.** | A Java program that is run in a browser over the internet | | **11.** | Translates higher level language to lower level language one statement at a time | | **12.** | Language classification where steps followed are linear, from beginning to end | | **14.** | A characteristic of Java; performs several tasks simultaneously | | |  |  | | --- | --- | | Down: | | | **2.** | A language directly understood by the computer, and is in binary 0's and 1's | | **3.** | A Java program that is run on your computer | | **4.** | Language classification which is event driven | | **6.** | Tranlates higher level language to lower level language in its entirety | | **10.** | A characteristic of Java where one program will run on any platform | | **13.** | The company that developed the first version of Java | |

2. Develop a Java Program using Java Tools and Styling Conventions

LS1 Describe the process that is followed to create and execute a computer program

# **Eight Steps to Good Programming**

The Eight Steps to Good Programming are as follows:

1. Define the Problem (what needs to be accomplished)

* In order to provide a solution, it is imperative to know exactly what the problem is. A precise and clear statement of the problem is necessary so you know what you are trying to solve.
* Understand what data are available.
* What are the inputs and outputs?
* What assumptions are to be made, or what constraints may be faced?
* Have a good understanding of problem solving in the context of computer applications

2. Develop an Algorithm (how)

* A finite set of unambiguous step-by-step instructions to be followed in linear fashion by a computer that generates a conclusion within a finite amount of time. This means that there are not an infinite number of instructions; the instructions must be valid for the computer executing them; the execution will not go on for ever; there will be a conclusion even though it may not be correct (may depend on data).

3. "Desk Check" or "Hand Calculation"

* Performing a pseudo-execution by hand of each algorithmic instruction to confirm the results expected so the programmer knows if the problem has been solved successfully.

4. Write the Code for the Program

* Take the algorithm and build the code around it. This involves translating the algorithmic instruction (assign 1 to the variable count) into Java (count = 1;)
* Use the algorithm as comments describing the problem-solving process. Since an algorithmic statement may require multiple program statements, the algorithm statements provide a synopsis (as comments) of what the code fragment is doing.

5. Translate the Program

* Just as essay writing has grammar and syntax rules, so does writing with a programming language. Prior to executing a program, the program is translated into the language a machine can understand, and it must follow all the grammar rules (syntax) of the programming language. If a rule is not followed, the program is said to have a syntax error, and the compiler/interpreter will indicate what syntax errors have occurred.
* Compile/interpret the program and correct any errors in the code.
* The algorithm may also contain logic errors. These will not show up as syntax errors because syntactically they are correct - they just don't make any sense logically. Consider the statement: "I floated a wallet over clouds." This statement is syntactically correct based on the rules of English grammar, but is nonsense from a logic point of view.

6. Run the Program

* The computer executes the lines of code  created by the compiler and we observe the results.
* The code may be an application (run on the computer), or an applet (run in a browser over the internet).
* At this point, your programs will be applications, but later, you will create java applets.

7. Test the Results

* Compare the results from (step 6) to the hand calculations done in step 3. If the results do not match, there is a problem in the logic, so you must return to step 2 to correct any logic errors. If your program does not do what it should, the program is said to have a semantic error**.**A semantic error is a logical error, and is a different sort of error than a syntax error.

8. Document and Maintain the Program

* This step is ongoing and does not need to be left to the end.
* Documentation includes both internal comments (algorithms or simple explanations) and external documentation (flow charts, algorithms, maintenance manuals, etc...).
* Maintenance of a program may require that the program be redesigned in situations where the problem has changed or updated if and when the underlying computer hardware or operating system is changed.

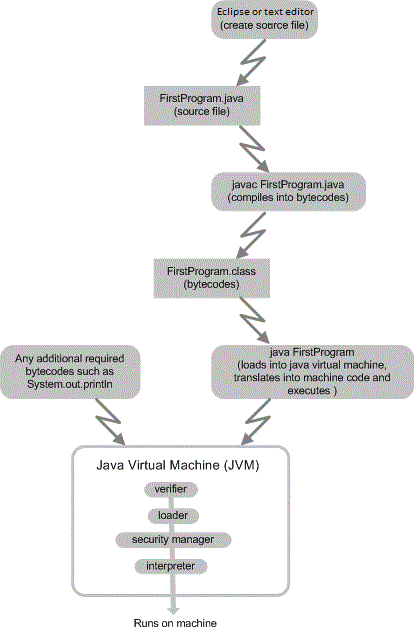
LS2 Describe source code, bytecode and object code

2.2.1 Overview of developing and running a program

# Creating, Compiling, and Running a Java Application

Now, you will look at the processes involved in creating, compiling and running a Java application. Then, you will be taken through the steps in performing these actions.

The Java programming development process consists of creating, compiling, and running programs, as is demonstrated in the following flowchart.



2.2.2 Source file

The program to display "Hello World" needs to be entered into a text file on a computer so it can be compiled. The file that contains the code is called the source file, or source code. Any text editor can be used to create the source file, and Notepad may also be used because is creates a plain text file. A word processor can be used if the file is saved as text only.

Eclipse is an integrated development environment that has a built-in editor for editing java files. Using an integrated developement environment is the best way to create and run Java programs. Eclipse is a free tool.  Go to  [http://www.eclipse.org/downloads/](https://www.eclipse.org/downloads/packages/release/kepler/sr1/eclipse-ide-java-developers) to get the Eclipse IDE.  Download the "Eclipse IDE for Java Developers" version of Eclipse.

2.2.3 Compiling

# Compiling a Source File Into Bytecodes (javac)

The source file is compiled into object code, known as bytecode, by running the compiler [javac](http://www.10ad.net/javadev/ch9.htm) at the command line, or selecting the compile option from the IDE menus. The compiling ensures all the rules of the language have been followed, and will identify any syntax errors (grammar errors). For example, if you entered **javac FirstProgram.java**at the command line, the java program source code saved in the file FirstProgram.java will be compiled. Doing this converts the human readable format into platform independent object code (bytecodes), in a file that has class as the file extension (FirstProgram.class). The bytecode created by the compiler is not the binary code of any existing computer. Rather, it is object code that can be read by all java run-time virtual machines. It is a universal language that interpreters on all machines can understand.

2.2.4 Running bytecode

# Run the Program - The Java Virtual Machine (JVM) (java)

The bytecode in the .class file can be interpreted on any machine that has a Java Virtual Machine (JVM) on it. The Java Virtual Machine processes each of the bytecodes in the .class file, and executes them one at a time.

The Java Virtual Machine is made up of several processes:

* The verifierensures the bytecodes being interpreted are valid and do not violate any language constraints.
* The class loaderloads the bytecodes into memory and includes other class libraries such as System.out if necessary.
* The security managerprovides additional security features.
* The interpreterreads the bytecodes and translates them into the machine language for a particular platform, and performs the actions specified by the statement.

The bytecodes can be run by the JVM on the same workstation as the source code, or the bytecodes can be downloaded over the Internet and run on any platform that has a JVM in a browser, applet viewer, or JDK.

## **Why the Java Virtual Machine?**

It is often unclear to people new to Java as to why it requires a compiler AND an interpreter when most other languages use only one or the other. Java uses both so it can be platform independent, portable, and dynamic. Compiling the source code into bytecodes converts it into a 'universal language'. The interpreting process is a much simpler process than in other programs, and can be made small enough to fit into an Internet Browser (Netscape or Internet Explorer), an applet viewer, a mobile device, or a Java Software Development Kit (JDK). It is then possible to download the bytecodes through the internet and have the Java Virtual Machine on the device convert them into the machine language that runs on that particular machine. The interpreter can start executing the beginning of the program, even if the entire program is not completely downloaded.

Diagram

Description automatically generated

LS3 Create a program

2.3.1 Create a program

# Introduction to Creating a Program

The previous Learning Outcome discussed the eight steps required to develop a program. This Learning Outcome will expand on Steps Four, Five, and Six by looking at writing, translating, and running a Java program.

**A Simple Program**

To begin this process you will code a simple program. The first program will be a programmer's typical first task of displaying "Hello World" on the computer screen. This is often chosen because it involves the output system of the device and ensures we can generate output. In addition it ensures that the entire development environment is configured properly.

The first program is very simple, and the problem definition, algorithm, and hand calculation do not have much detail.

1. Problem Definition: Display the message "Hello World" ( the double quotes are NOT part of the message)  
2. Algorithm: Display "Hello World"  
3. Hand Calculation: Output will be "Hello World"

The code to accomplish this task may look as follows:

**public class FirstProgram**

**{**

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

**{**

***//display Hello World - the double quotes are not part of the message -***

***//they indicate to java that the data between them is text -***

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

**}**

**}**

2.3.2 Output Syntax

# Syntax

## Output Syntax

* System is a class provided by Java that has methods for displaying to the standard output (the screen, commonly referred to as the console)
* System.out.print(data) is a method that will display 'data' to the standard output and stay on the same line
* System.out.println(data) is a method that will display 'data' to the standard output, and then continue on to the next line. Note the ln in println stands for line (l is the lowercase letter L, not the number one)
* 'data' can be any primitive type or object (including Strings) or combination

2.3.3 Example

# Example

## Output Examples:

**System.out.print("Hello World");**

The character string "Hello World" will be displayed on the screen. The quotation marks (" ") will not be displayed.

**System.out.print(totalSales);**

The value in the variable totalSales will be displayed on the screen.

**totalSales = 100;**

**System.out.print("Total sales is " + totalSales);**

In this example, the compiler will perform implicit casting because there are two different types of data: a string and a number. It will convert the value in the variable totalSales to a String, and append it to the end of the constant string "Total sales is". The output will be "Total sales is 100".

**System.out.print("The method print does not ");**

**System.out.println("advance to the next line after");**

**System.out.println("it displays the");**

**System.out.println("data");**

The above example illustrates the difference between using print and println. println will advance to the next line after it displays the data. The output for the above section of code will be:

**The method print does not advance to the next line after**

**it displays the**

**data**

2.3.4 Simple Program

# A Simple Program

**1 public class FirstProgram  
2 {  
3 public static void main(String[] args)  
4 {  
5 *//display Hello World*  
6 System.out.println("Hello World");  
7 }  
8 }**

**class**

* The first line of code defines a class called FirstProgram.
* A class is a template, or blueprint, for objects.
* Every Java program has at least one class, and the program code is contained within the class definition (enclosed in curly braces {} ).
* A class must be saved in a file with the same name as the class for the root name (exact spelling and capitalization), and java for the extension.
* This class will later be saved in a file called FirstProgram.java.
* Everything inside a class is indented one tab stop from the curly brace.

**main Method**

* The third line of code is the main method.
* Every Java application must have a user-declared method called mainthat defines where the program begins.
* The main method establishes the control for the program flow.
* The Java interpreter executes the application by calling the main method and performing the statements in the main method.
* The contents of the method are enclosed in curly braces {}, and are indented one tab stop.

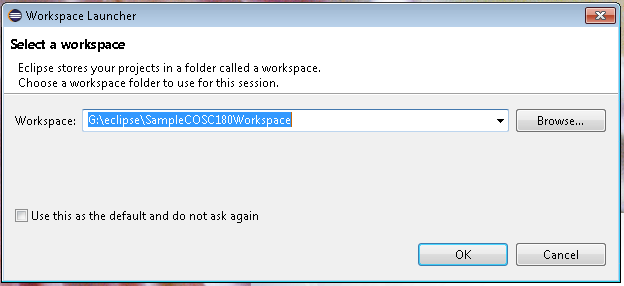
**Blocks**

* Lines 2,4,7, and 8 are the block braces. Braces, in a program, form a block that groups components of the program. In Java, each block begins with an open curly brace (i.e. { ) and ends with a closing curly brace (i.e. } ).
* Each class has a class (object) block that groups the data and methods of the class.
* Lines 2 and 8 are the block braces for the FirstProgram class.
* Each method (action) has a method block that groups the statements in the method.
* Blocks can be nested, meaning that one block can be placed within another.
* Lines 4 and 7 are the block braces for the main method. As mentioned above, the contents of a block are indented one tab stop from the curly braces.

2.3.5 Create Source File 1

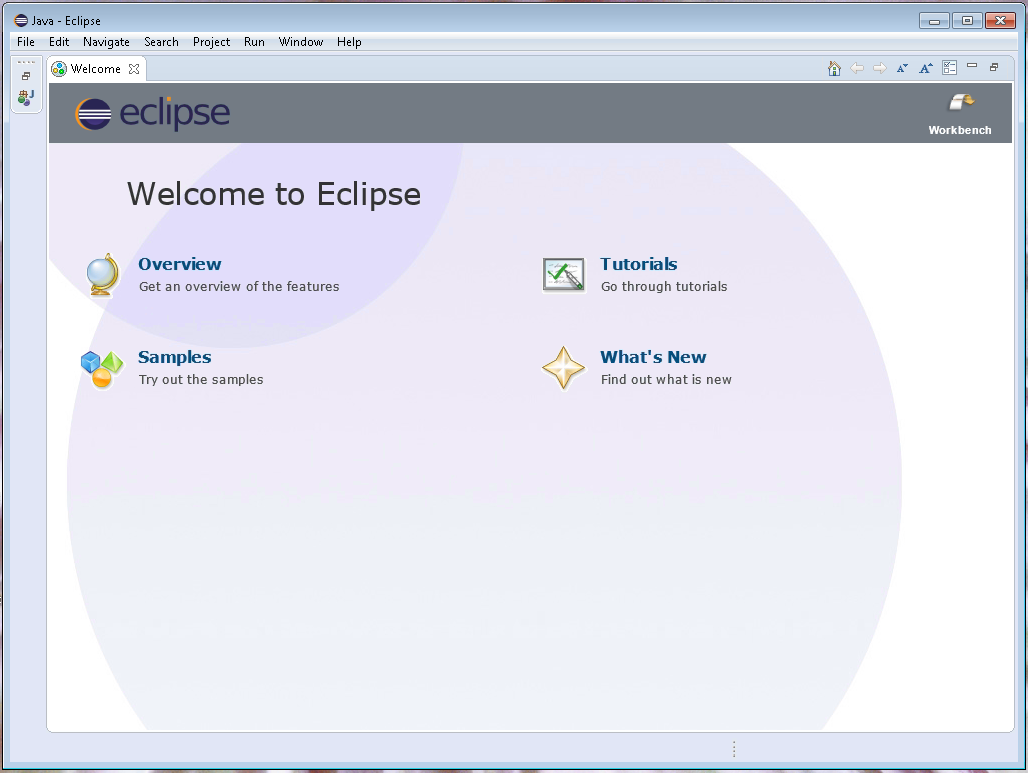
The images below may not look exactly like yours as your version of Eclipse may be different. The main screens and operations are the same.  
**Create the java source file using Eclipse**

Start Eclipse.  The Workspace Launcher window will pop up first asking you to specify a location for the workspace.   A workspace is a place where Eclipse stores all your Java files.



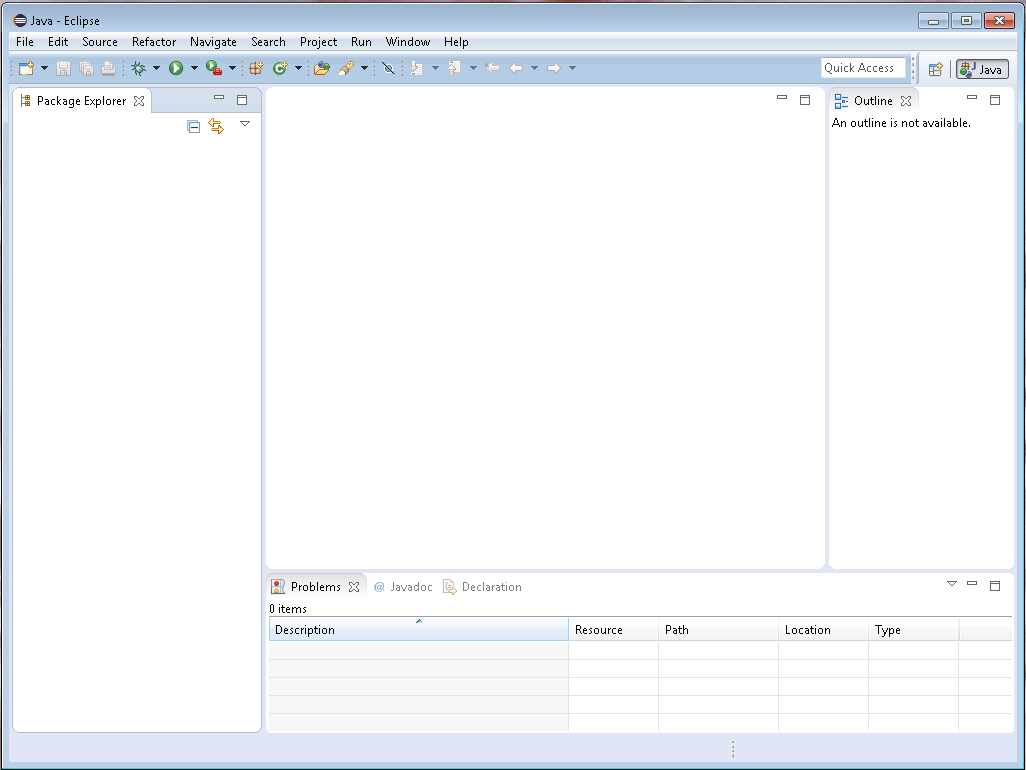
Select a folder to use as you workspace. You can select the Browse button to navigate through your file system to find an appropriate place. Make sure specify at least one folder. For example: c:\Workspace instead of c:\

Now select OK.   This will bring up the Eclipse IDE itself with a help screen displayed:

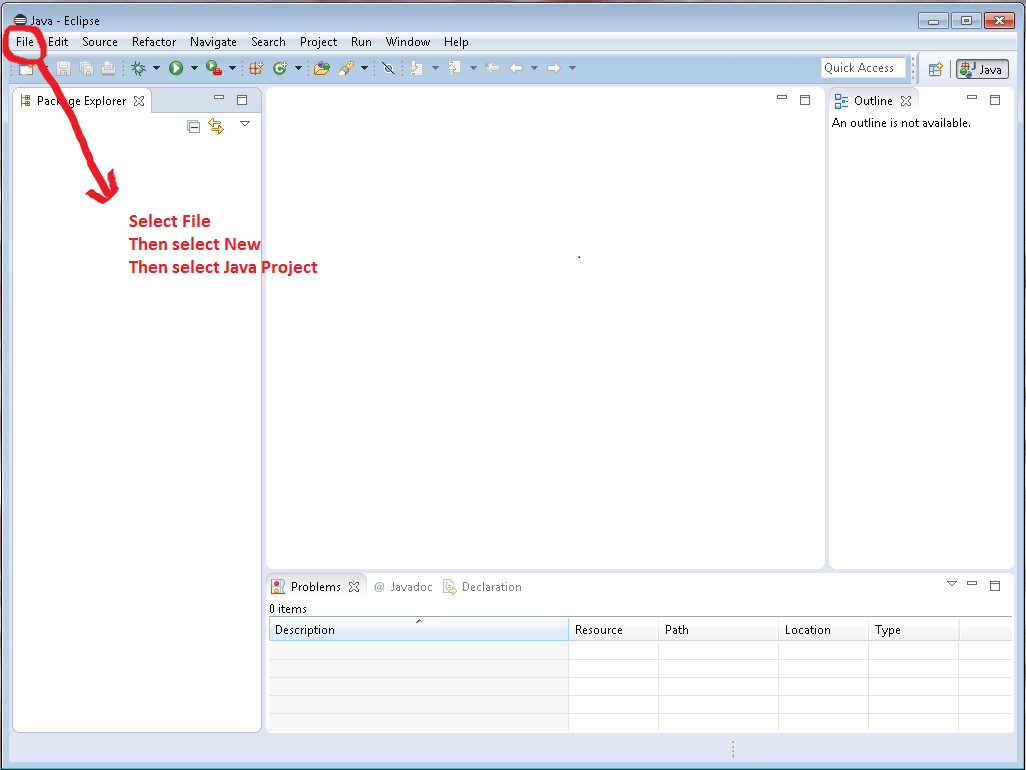


This screen has the help tab on top.  These are some beneficial resources to get help on Eclipse.  You can also get back to these by selecting Help->Welcome from the main menu.

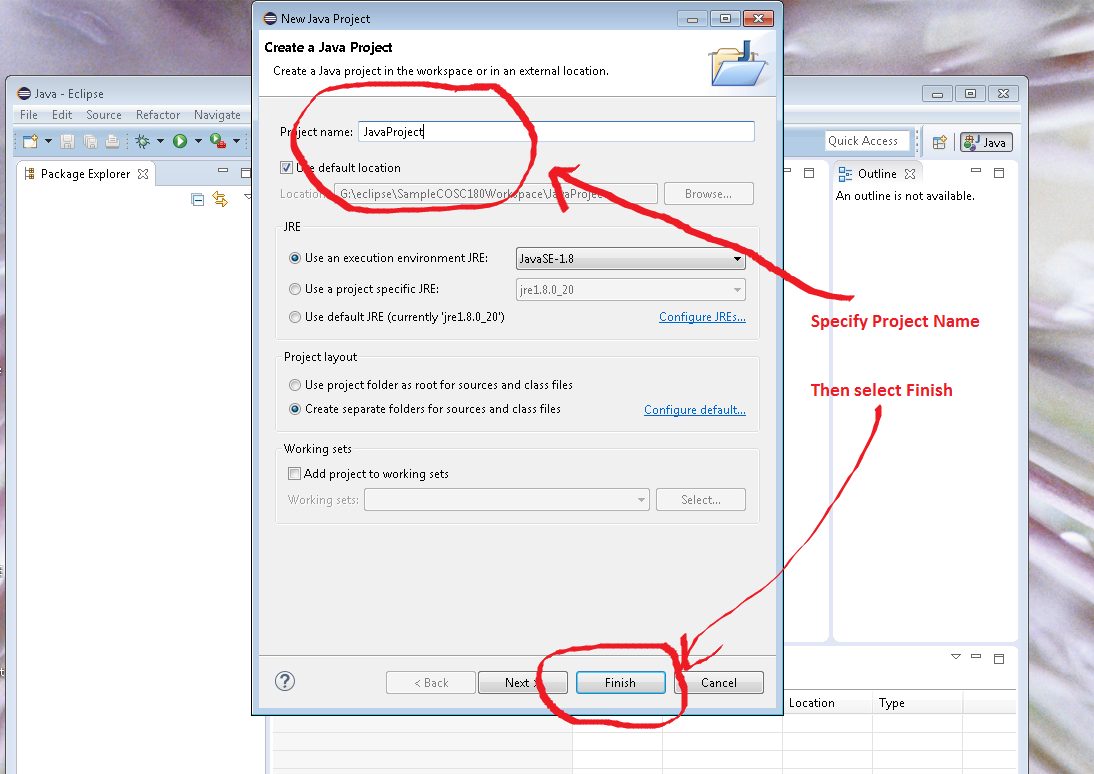
Closing the help window only by selecting the X beside the Welcome.  This will result in the main Eclipse IDE window being shown.  This looks as follows:



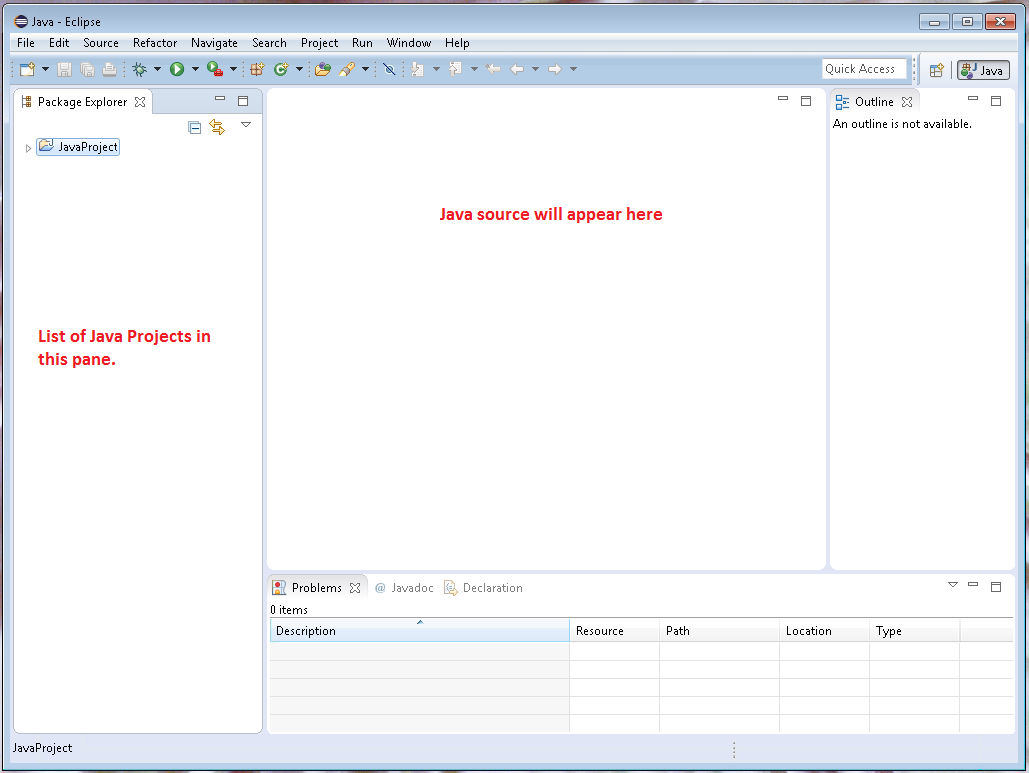
Using Eclipse to create Java programs we must always create a Javav project to hold the Java files.  The Java project appears on the left in the Package Explorer window.  To create a new Java project select File, then select New, then select Java Project.



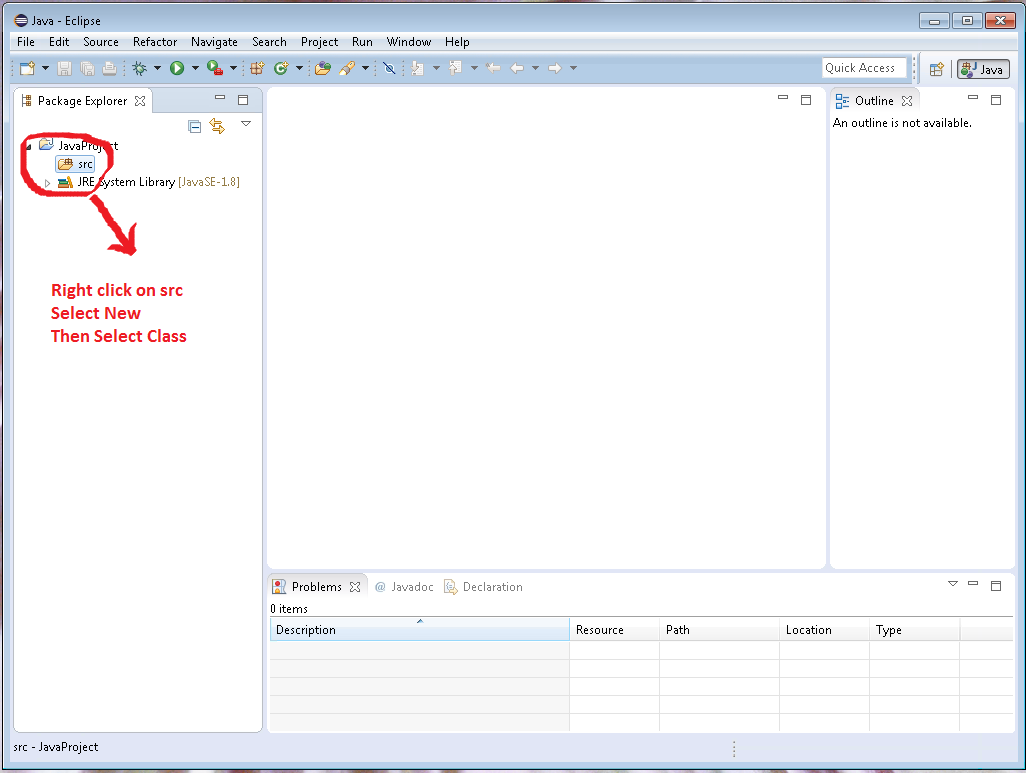
This will bring up the Java Project creation wizard:



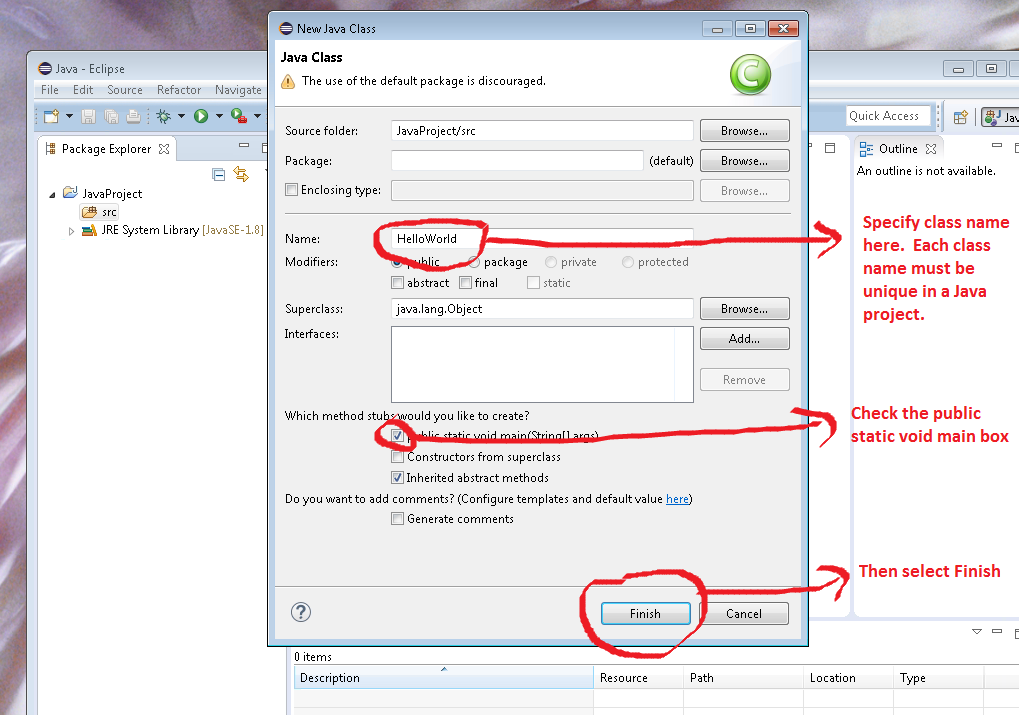
On the Java Project creation window enter the project name and select Finish.  This will create the Java project and it will now appear in the package explorer window.



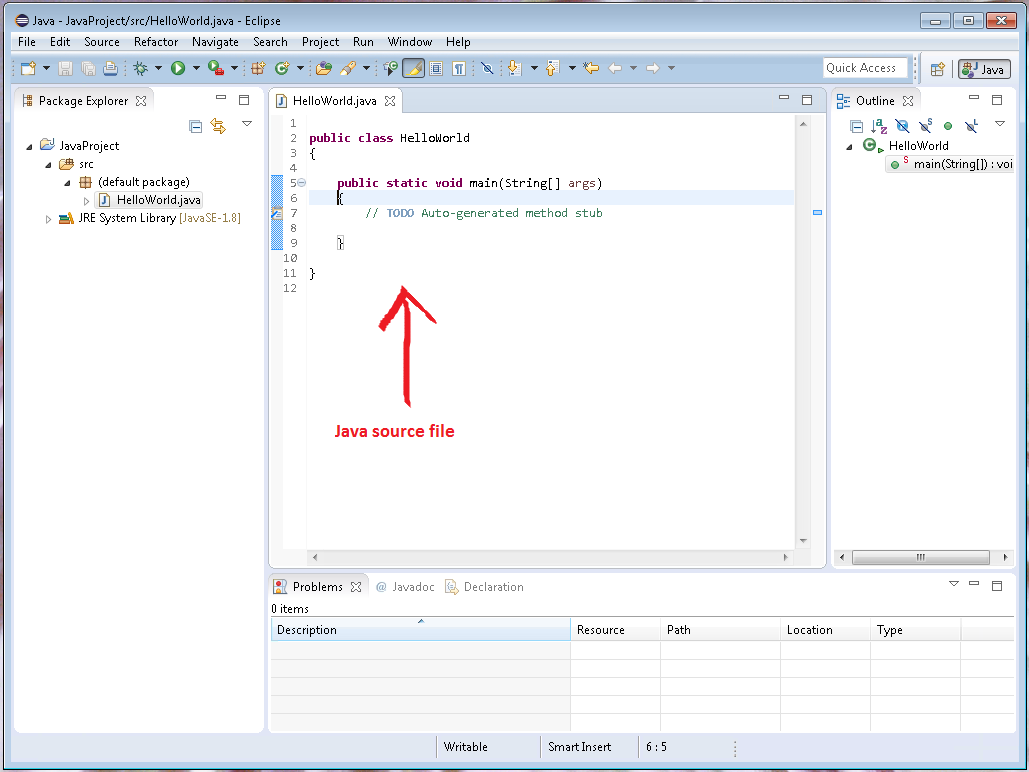
To create a Java file expand the JavaProject by select the arrow right beside it in the package explorer. This will expand the files in the Java Project.   This now allows us to create a Java program in our Java project.   You MUST always create a new Java class from within a Java Project.  Right click on src folder under the Java project and then Select New -> Class



Selecting New -> Class will bring up the Java program creation wizard. In the Java class creation wizard specify the name of your class, check the public static void main box and select Finish.



You should now have a Java program showing in the Java source code area of the screen.



2.3.6 Create Source File 2

**Creating, Compiling, and Executing a Java Application**

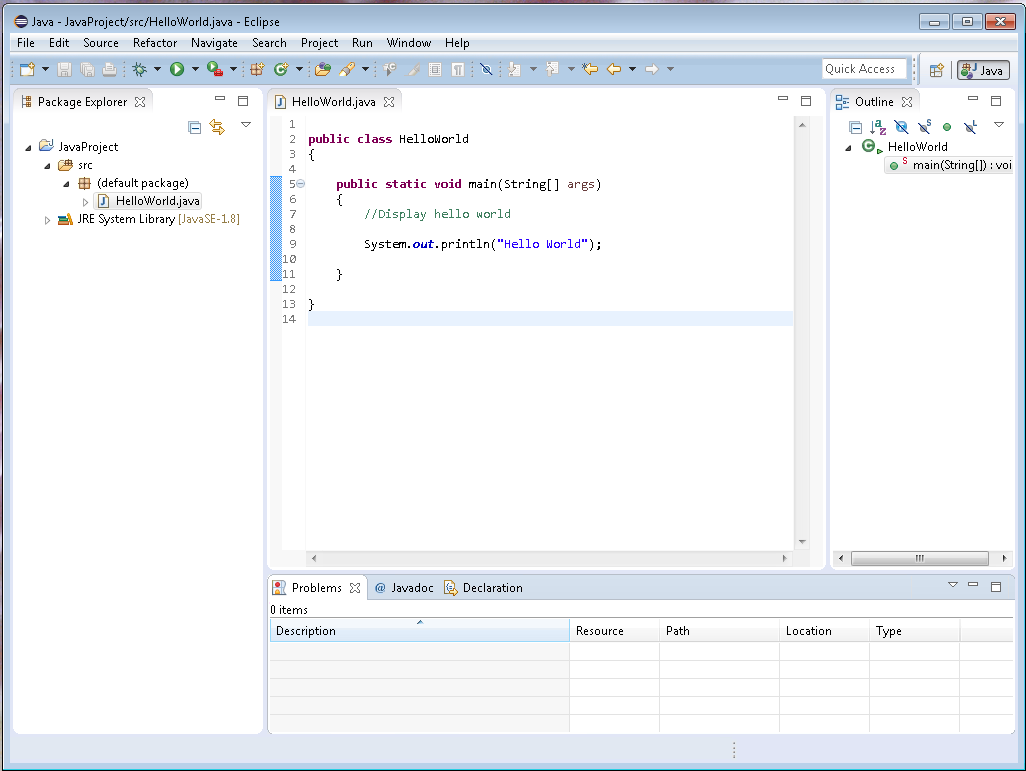
**Create the java source file using Eclipse, cont.**

Type the code for HelloWorld.java, making sure you have the identical case and spelling (i.e. Println is not the same as println, and will result in a syntax error if typed incorrectly).

Note the different coloring automatically done by Eclipse.DrJava once the file is saved with a .java extension. Comments will appear in green, and keywords appear in blue. You can take advantage of this functionality by realizing a word is misspelled when it does not appear in the expected color.

Line numbers were turned on for explanation purposes. Line numbers can be turned on and off using Window->Preferences->General->Editors then click on Text Editors.  Setting line number is about half way down the Text Editors preferences page.

Line 3, 4, and 7 are tabbed over one spot (4 spaces), and Line 5 and 6 are tabbed over 2 spots (8 spaces).



Save your file by selecting Save from the File menu or from the toolbar. The file is automatically saved to your project workspace.

2.3.7 Reading

**Introduction to Java Programming**  
comprehensive version  
by Y. Daniel Liang. Prentice Hall, New Jersey.

ISBN: 978-0-13-376131-3 ( 10th Edition)

Earlier versions of the textbook would be acceptable for COSC180 BUT NOT for the follow on course COSC 190  
ISBN: 0-13-293652-6 (9th Ed.) or 0-13-213080-7 (8th Ed.)

|  |  |  |  |
| --- | --- | --- | --- |
| **COSC180 Reading Table** | | | |
| **Topic** | **10th Edition** | **9th Edition** | **8th Edition** |
| Explain programming terminology | 7-12 | 2-16, 1261-1264 | 2-11, 1319-1322 |
| Algorithms | 34 | 34-35 | 24-25 |
| Create a Java program | 12-18 | 16-22 | 11-16 |
| Variables | 35, 40-43, 44-45 | 40-46 | 29-33 |
| Constants | 43 | 43, 48-49 | 31-32, 35-36 |
| Input and output | 37(input), 38(output), 45-46(input), | 17, 112-115 (output) 37-40 (input) | 11, 95-97 (output) 26-29 (input) |
| Strings | 386-388 | 68-69, 336-350 | 50-51, 302-313 |
| Operators | Math Operators: 41-43,, 44-48, 54-58 Math Precedence: 50-51 Relational Operators: 76-77 Logical Operators: 93-96 Operator Precedence: 104-105, 1268-1269(Operator Precedence Chart) | 44-68, 115-117, 82-84, 101-106, 1256-1257 | 32-50, 97-98, 72-73, 88-91, 1314-1315 |
| if statements | 78-80 | 84-89 | 74-78 |
| if else statements | 80-81 | 89-91 | 79-80 |
| Nested if statements | 81-83 | 91-93 | 80-81 |
| switch statements | 100-102 | 108-111 | 93-94 |
| Conditional operator | 103-104 | 111-112 | 95 |
| for loops | 170-173 | 146-149 | 126-128 |
| while loops | 158-168 | 134-144 | 116-124 |
| do while loops | 168-170 | 144-146 | 124-126 |
| Nested loops | 176-177 | 152-153 | 129-130 |
| Defining a method | 204-206 |  |  |
| Predefined methods | 206-208 | 197-201 | 172-175 |
| Nothing passed in, nothing returned | 209-209 | 178-180 | 156-157 |
| Data passed in | 212-215 | 183-189 | 160-165 |
| Value returned | 209-211 | 189-193 | 165-168 |
| Overloading | 219-222 | 193-196 | 168-170 |
| Scope of variables | 222-223 | 196-197 | 171-172 |
| Defining and declaring | 322-329 | 296-308 | 264-274 |
| Encapsulation | 344-347 | 319-322, 375-379 | 283-286, 347-351 |
| this | 356-357 | 373-375 | 346-347 |
| Constructors | 329-329 | 296-298, 303 | 265, 270 |
| References | 347-351 | 304-308 | 270-274 |
| Instance variables | 330-333 | 304, 312-317 | 271, 278-281 |
| Static variables | 337-342 | 312-317 | 278-281 |
| Visibility modifiers | 342-344 | 317-319, 1258-1259 | 282-283, 1316-1317 |
| Wrapper classes | 380-384 | 393-397 | 476-479 |
| Packages | 343 | 317-318, Also see [Supplement III.G "Packages"](http://www.cs.armstrong.edu/liang/intro9e/supplement/SupplementPackage.doc) on companion website | 282, Also see [Supplement III.G "Packages"](http://www.cs.armstrong.edu/liang/intro8e/supplement/SupplementPackage.doc) on companion website |
| Arrays | Single Dimensional: 246-250, 256-260 Foreach Loop: 251 Multi Dimensional: 288-293 | 224-245, 264-280 | 198-216, 236-251 |

LS4 Compile the program using a compiler

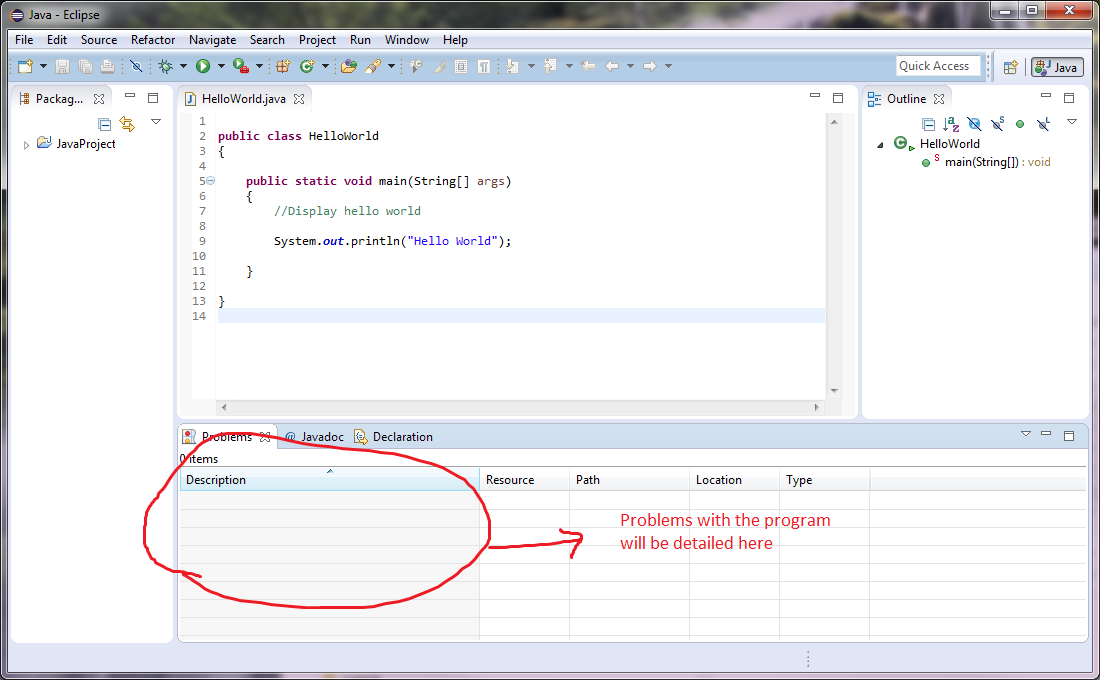
# **Creating, Compiling, and Executing a Java Application**

## **Compile the .java program into a .class file**

Java files can be compiled from the command using the javac application available as part of the JDK.

However, Eclipse is an integrated developement environment that allows creating, compiling and running the Java program.  Eclipse actually automatically compiles the program as it is written and when the file is saved.   Eclipse will automatically highlight any problems with the program as it does this.

As code is typed into the editor window of eclipse any syntax errors will be indicated by the red underlining.  When the Java file is saved all problems will be detailed in the problem view area.

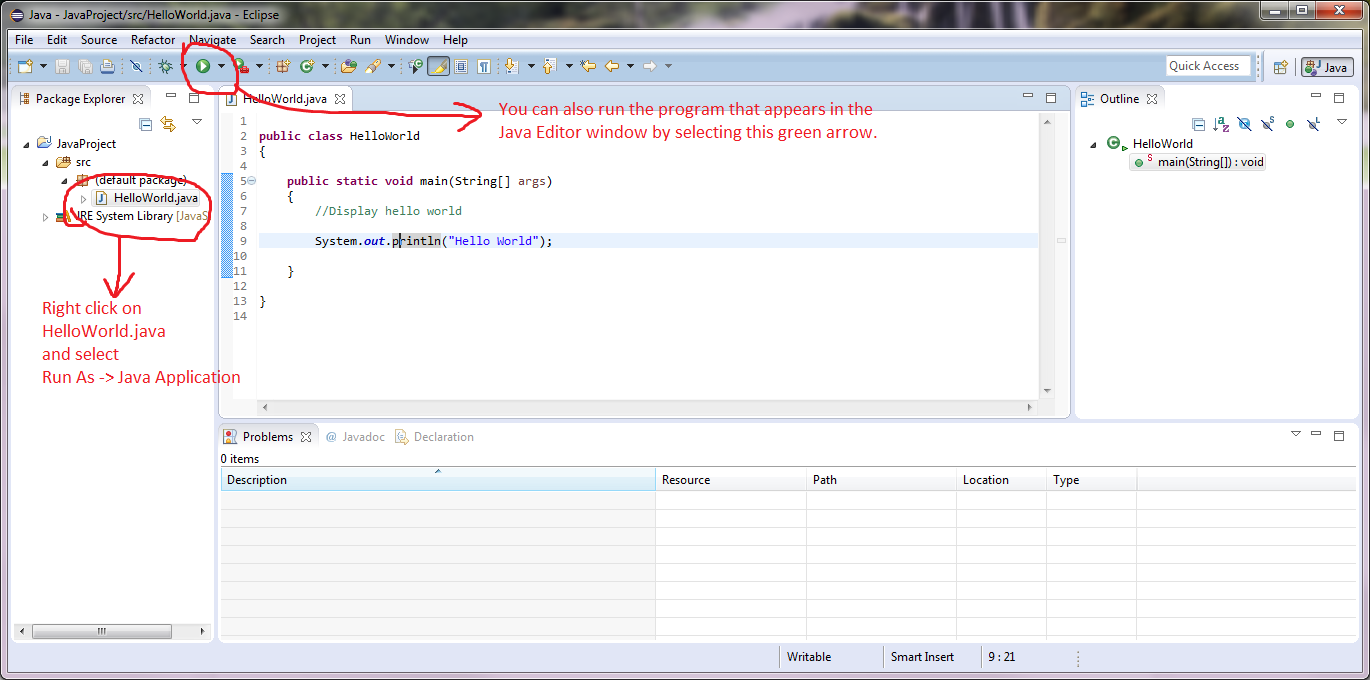


LS5 Run the program using an Integrated Development Environmnet (IDE)

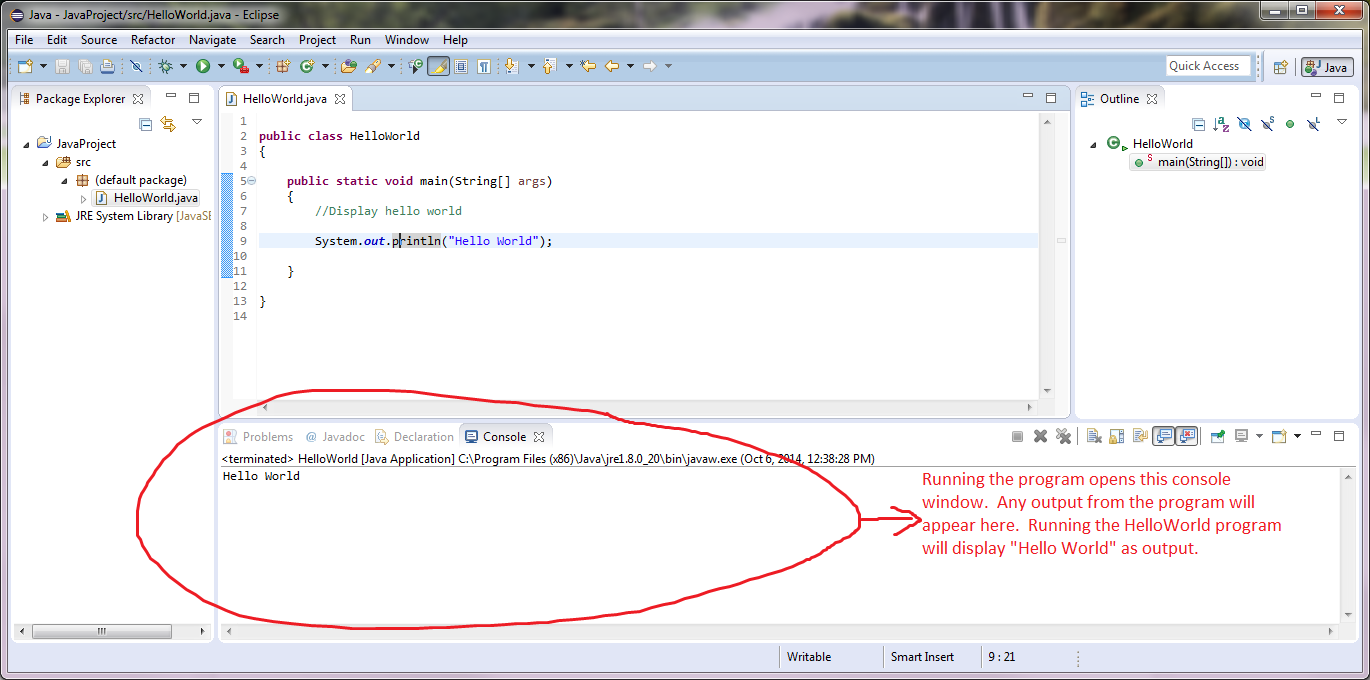
# Creating, Compiling, and Executing a Java Application

## Run Your Program

Once the Java file is error free the program can be run.  This is done by right clicking on the Java file and selecting Run As-> Java Application.  Or alternatively the green arrow on the toolbar can be selected.



Selecting one of methods of running the program will cause the Java program to be run.  Any output from the program will appear in the console view that appears at the bottom of the Eclipse window.



LS6 Explain the difference between Semantic and Syntax errors

# Programming Errors

Encountering errorsis unavoidable, even if you follow the "Eight Steps to Good Programming". Following the eight steps will minimize your errors, but even experienced programmers encounter errors in their work. Errors can be classified into three types: syntax errors, semantic errors, and runtime errors.

## Syntax Error

A syntax error occurs when one of the rules of the language is not followed. A standard example of a syntax error in Java is forgetting to place the semicolon character ";" at the end of a statement. Syntax errors are detected by the compiler in Step Five (see previous screens), and are corrected by modifying the program to better follow grammatical rules of the language. Note that a single error will often cause the compiler to complain about subsequent lines that, in fact, may be correct; so always start out by dealing with the first lines that are indicated to have problems.

## Semantic Error

Semantic errors are logic errors. The program works, but does not give the expected results. For example, if a program was designed to give the date everyday, and it gave a date of January 33, a semantic error has occurred because there are only 31 days in January.

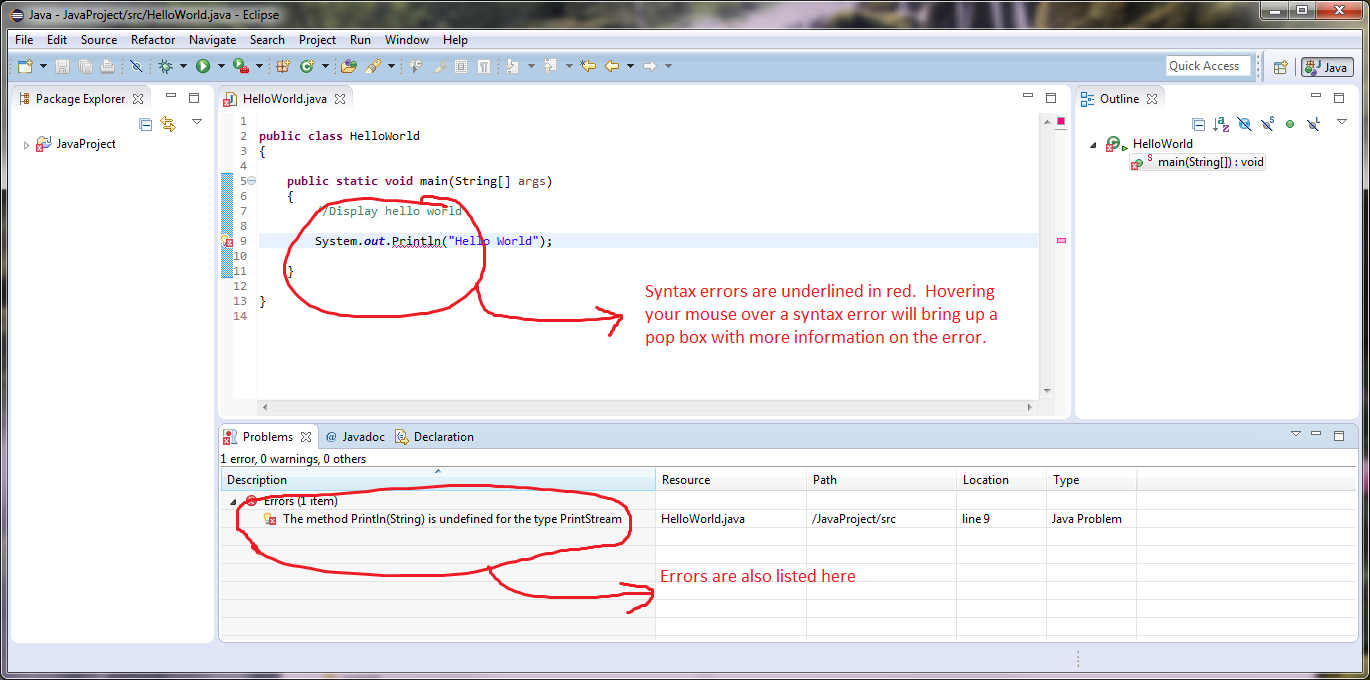
Semantic errors are detected by comparing the results obtained in Step Seven with the expected results determined in Step Three. Semantic errors are corrected by returning to Step Two and modifying the design.

## Run time Error

Run time errors cause a program to terminate abnormally. Run time errors occur while an application is running and occurs when the environment detects an operation that is impossible to perform. Input errors are typical run time errors - such as entering a character value when the program is expecting a numeric value. For example, if you have a vehicle that is running properly, and you put water in the gas tank, rather than gasoline, the water will cause the vehicle to stop running. Runt time errors, often called exceptions, are detected in Step Six and Step Seven when the program is run. Exception handling (covered in COSC 190) can be used to manage run time errors.

# Creating, Compiling, and Executing a Java Application

Eclipse is very helpful into trying to explain what error has occured.  As mentioned earlier if Println was used instead of println in the Java code a syntax error would occur.   Java highlights this as a syntax error.



Eclipse highlights the syntax errors by underlining them. Eclipse will also detail the error in the problems window at the bottom.  Note:  errors won't appear in the problems window until after the Java file is saved.  Hovering the mouse over a syntax error in the Java code will bring up a pop up box with an explanation of the error as well as suggested fixes.   The suggested fixes are NOT always the right fix but can be in some cases so use the automatic fix with some caution.

# Fixing Syntax Errors

You will now create some errors in your file on purpose so you have an opportunity to practice fixing the inevitable errors.

One error you may commonly encounter is misspelling, or incorrect capitalization of a method name, class name, or keyword (a word with special meaning in java such as **void**) . For example, the statement "System.out.println" is used to send output to the console. The word println in this statement identifies a method in the System.out class included with the Java language that will display the text in parenthesis. If you change the spelling to pintln, or change the capitalization to Println, the compiler will not find a method of that name, and will give you an error. In Eclipse syntax errors are highlighted using an underline and a red box with an x to the left of the statement as shown below.

Graphical user interface, text, application

Description automatically generated

In Eclipse if you hold you mouse over the error or over the red box a tool tip will appear which describes the syntax error. An example of this is shown below:

Graphical user interface, text, application

Description automatically generated

The error message tries to identify the error that occurred so reading the error message will help you identify the cause of the problem. If you compile from the command line or are using a different development environment, the error message may appear slightly differently. Shown below is the same error when compiled from the command line. Note the caret (^) points to the place where the compiler thinks the error occurred. After correcting the error, remember to save your file and compile the program again.

Text

Description automatically generated

# Fixing Syntax Errors

Change println to its correct spelling, then remove the closing bracket on line 8. Note that the error message tries to indicate where it thinks the error occurred. This line number can give you a clue as to where to correct the error.

Correct version of the file shown below:

Graphical user interface, text, application

Description automatically generated

The following screen shows the highlighted error after the bracket was removed.

Graphical user interface, text, application

Description automatically generated

Screen shot below shows the tool tip error that is displayed.  Missing brackets can sometimes cause unusual error messages.

Graphical user interface, text, application

Description automatically generated

# Fixing Syntax Errors

Put the closing bracket back in line 7, then change the comment indicator from // to / on line 5. Notice the coloring of the comment changes to indicate it no longer recognizes the line as a comment.

Graphical user interface, text, application

Description automatically generated

The following screen shows the tool tip message for this error.

Graphical user interface, text, application

Description automatically generated

Also notice that changing spaces in a line, inserting blank lines, or changing the comment makes no difference in the program. These changes are irrelevant to the compiler.

# Fixing Syntax Errors

Correct the comment indicator back to //. Now, change the case of the letter 'm' in main to Main. When this error is compiled, there will be no errors because it is acceptable (though undesirable) to have a method named Main. The problem occurs when you try to interpret this. The interpreter begins execution in the main method and there is no main method in this class now.

Graphical user interface, text, application, chat or text message

Description automatically generated

When you try to run the program you will get this pop up window or you may see that you do not have the option to run the program.

Graphical user interface, text, application

Description automatically generated

The following screen shows the error message in console area for incorrectly named main method.

Graphical user interface, text, application

Description automatically generated

Correct the main method name by returning the lowercase 'm'. Now, change the class name to FirstClass, rather than FirstProgram. The compiler will complain because a public class must be saved in a file with the same name as the class.

Graphical user interface, text, application

Description automatically generated

The following screen shows the tool tip message for an incorrect class name.

Graphical user interface, text, application

Description automatically generated

# Practices

Correct the syntax errors in the following four programs. If desired, you may download the files and run them through the compiler. If you have difficulty with any of the practice exercises, use the Discussion Forum to ask your classmates for help, or contact the course instructor.

**public class SyntaxError1**

**{**

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

**(**

**//display Hello World**

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

**)**

**}**

**public class SyntaxError2**

**{**

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

**{**

**/\*display Hello World**

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

**}**

**}**

**public class SyntaxError3**

**{**

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

**{**

**//display Hello World**

**System.out.println "Hello World";**

**}**

**}**

**public class SyntaxError4**

**{**

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

**{**

**//display Hello World**

**System.out.println ("Hello World")**

**}**

**}**

# Answer: Syntax Error 1

If you compiled Syntax Error1.java you would get the following syntax error:

Text

Description automatically generated

The suggestions from the compiler are not helpful in this situation. The real problem is that ( and ) were used instead of { and } in the program on lines 4 and 7. The compiler attempts to suggest where the problem in the code is, but it is not always accurate.

# Answer: Syntax Error 2

If you compiled Syntax Error2.java you would get the following syntax error:

Text

Description automatically generated

The suggestions from the compiler are somewhat more helpful in this situation. The compiler suggests that on line 5 there is an unclosed comment. This is true. /\* begins a comment, and it is necessary to have \*/ to close the comment. You will notice that all the letters after /\* are in green to indicate that the compiler thinks the rest of the program is a comment. The compiler also suggests there is a ; missing on line 10, but this is an error that is cascaded from the error on line 5. Once the first error is corrected, the second error will disappear. You can correct the error by replacing /\* with // to turn it into a one line comment, or you can place \*/ at the end of line 5 to end the block comment.

# Answer: Syntax Error 3

If you compiled Syntax Error3.java you would get the following syntax error:

Text

Description automatically generated

The suggestions from the compiler are indicating that line 6 is not a valid statement. It is expecting ( after the println not “ so it complains about the “. You can correct the error by placing ( ) around the string.

# Answer: Syntax Error 4

If you compiled Syntax Error4.java you would get the following syntax error:

Text

Description automatically generated

The suggestion from the compiler indicates a semicolon (;) is missing from the end of line 6. This is accurate. You can correct the error by placing ; at the end of line 6.

LS7 Describe the styling conventions

# Java Style Guide

A programming style guide is a standard so that written code will have a consistent look and feel from person to person. This will help you read the code provided by your instructors and classmates; it will also help your instructors and classmates read your code. This style is not the only way a program can be written, and in other organizations, you may find that a different style guide will be used. However, for programs written while you are in Computer Systems Technology, you must follow these guidelines, unless otherwise directed by your course instructors (they will provide an exception sheet, as required).

For example, some people like to have their brace { on its own line, while others prefer it on the end of the line. The compiler is unconcerned about the style of the code, so it is the preference of the programmer. One style is not always better than another, but one style may become the agreed upon style for a particular group. When you work for a company, it will have a certain style for all employees to follow. There has been a Style Guide created for all programs written in this course. You are expected to follow all the styles specified in this guide when you hand in assignments, and marks **will be deducted** if you do not follow these guidelines. Many of these styles can be set in the Global Options of DrJava and Eclipse. Once the style has been specified in the project properties in Eclipse, it can be used to convert all of your program into the requested style.

LS8 Use the style conventions

# Practice

Using DrJava or Eclipse, write a program that says, **This if My First Java Program**and **I can print on many lines**. Compile and run your program. If you have difficulty, use the Discussion Forum to ask your classmates for advice, or contact the course instructor.

**This is My First Java Program**

**I can print on many lines.**

3. Perform elementary programming

LS2 Explain Variables

LS3 Create a program that declares, initializes and assigns variables

LS4 Create a program that uses type casting

LS5 Create a program that uses named constants

LS6 Create a program that displays output

LS7 Create a program that can read variables of different types from the keyboard

4. Use a debugging tool

5. Create a program that uses strings and mathematical library routines

6. Create a program that uses operators and decision statements

7. Create a program using repetition structures

8. Create a program using methods

9. Troubleshoot a defective program