RYERSON UNIVERSITY

School of Computer Science

CPS109 Course Notes Set 1

Alexander Ferworn Updated Fall 15

Class Notes 1

Alexander Ferworn

Unrelated Facts Worth Remembering

This section precedes all the "Class Notes". They are just my ramblings and can be ignored.

You will learn this

Believe that change can happen despite overwhelming evidence indicating otherwise.

Table of Contents

1 How this course works

There is no real mystery about passing this course. You need to do five things,

1.) Read the material BEFORE you come to class.
2.) Come to class and listen annotating what is in the documents with your own comments so you understand them. I teach from them, using the book to support them.
3.) Do the labs and assignments honestly. You can get help from a TA or a friend but at the end of the day your work must be your own.
4.) Use the course textbook to figure out the labs and assignments since the labs and assignments are taken out of the course text book.
5.) Write, compile and execute lots of Java.

You will find that this document and all the documents that you see in this course support the lectures in this subject. I have liberally stolen bits and pieces out of a lot of publications and have put a lot of work into it. This means that the best thing that you can do is to print out each document, as it becomes relevant (and available from the web site) and bring it to class with you.

I guarantee that you will see me using these documents to support any discussion in class.

2 What is computer science?

This course is called *Computer Science 1*. So, what is computer science? Well there are many answers to that question.

When we ask this question, consider that we don't have car science or telephone science, dishwasher science, etc. Perhaps we should ask whether or not computer science is part of some other disciplines like mathematics, physics, neuro-science, electrical engineering, linguistics, logic or philosophy. In fact, some computer scientists consider this to be the case and consider it their goal to so thoroughly integrate the various parts of computer science back into its parent disciplines so that it no longer has a separate identity as a separate science. Well judging by what is happening in the world today that is unlikely to happen.

2.1 Computer Science Misconceptions

Before we elaborate on what computer science is, let's consider some possible definitions of the field of computer science and discuss the deficiencies of each.

2.1.1 Computer Science is not the study of computers

Much of the important scientific work that lays the foundation of computer science was done between 1915 and the mid 1940's. This was, of course, before electronic computers had been built. The scientists doing this work were, for the most part, mathematicians, and, in fact, the first computers were actually women who had the job title "computer".

Figure 1 The first "computers"

In fact, I will go a little further to suggest that even before the term “computer” even existed, the concept of “programmer” existed—if I can be granted some leeway as to what the programmer was programming—and the first programmer was also a woman.

Ada Lovelace (1815-1852) was the only legitimate child of Annabella Milbanke and the poet Lord Byron. Her mother, Lady Byron, had mathematical training (Byron called her his 'Princess of Parallelograms') and insisted that Ada, who was tutored privately, study mathematics too.

Ada met Charles Babbage at a party in 1833 when she was seventeen and was entranced when Babbage demonstrated the small working section of his “Analytical Engine” to her. In1843 she published a translation from the French of an article on the Analytical Engine by an Italian engineer, Luigi Menabrea, to which Ada added extensive notes of her own. The Notes included the first published description of a stepwise sequence of operations for solving certain mathematical problems and Ada is often referred to as “the first programmer”.

Figure 2 The first programmer: Ada Lovelace

Ada speculated that the Engine “might act upon other things besides numbers... the Engine might compose elaborate and scientific pieces of music of any degree of complexity or extent”. The idea of a machine that could manipulate symbols in accordance with rules and that number could represent entities other than quantity mark the fundamental transition from calculation to computation. She has been referred to as 'prophet of the computer age'. Certainly she was the first to express the potential for computers outside mathematics[1].

Given this example, one might correctly surmise that much of the theoretical basis for computer science was developed before computers existed, so a more inclusive definition is needed.

2.1.2 Computer Science is not the study of the programming of computers

This is a popular view held my many, perhaps because the programming of computers is one of the more visible activities of many of those involved in computer science. However, this definition is not nearly inclusive enough since many activities that a computer scientist might engage in involve no programming at all.

For example, it may be necessary to give a mathematical proof of correctness of an algorithm, or a systems analyst may need to study a process in detail before beginning a design of a data processing system. A computer engineer may need to design a simulation model for a cache memory system that will buffer the speed difference between a fast processor and its memory system. A researcher may try and determine the least difficult way for a robot to find its way out of a maze.

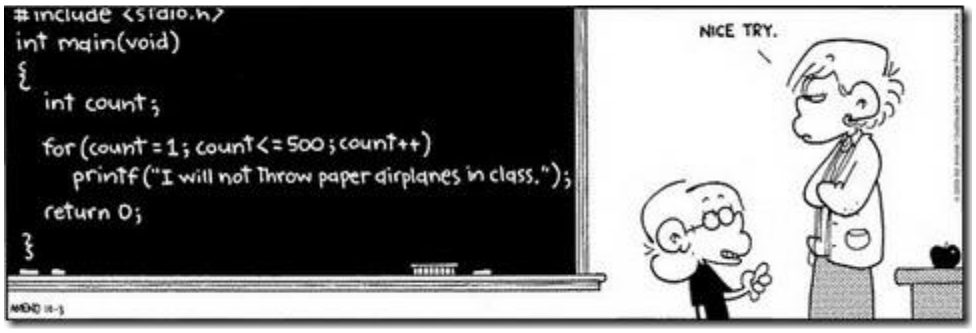
[1] Adapted from http://www.computerhistory.org/babbage/adalovelace/

Programming is of fundamental importance to computer science since it is the requisite laboratory skill that most computer scientists possess in order to be able to pursue their work, but that work often involves much more than just programming.

In the previous paragraph a number of terms were used that you may not be familiar with. Do not worry about this for the moment as definitions for most of these will be given later.

2.1.3 Computer Science is not the study of the uses and applications of computers and software

Since the personal computer became widely available in the early 1980's, the pervasive application of computers to nearly all fields has popularized the view that computer science is concerned primarily with computer applications. It is thought by some that computers, without their widespread general applicability to a variety of fields, would not constitute a sufficient body of knowledge to warrant being considered a body of scientific information. They also argue that it is sufficient to be trained in a few computer applications software packages in order to work as a computer specialist.

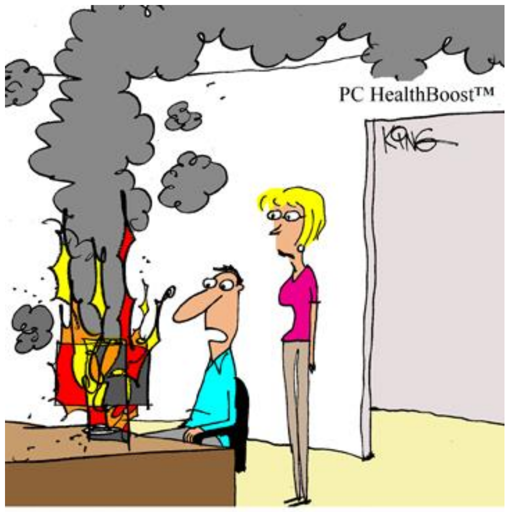


```
#include <stdio.h>
int main(void)
{
  int count;
  for(count = 1; count <= 500; count++)
     printf("I will not Throw paper airplanes in class.");
  return 0;
}
```

Clearly, the field of computer science is much larger than just computer application software--we do more than just “apps”!

2.1.4 Computer Science is not the study of problem solving via mechanisms

In this definition we have the idea that computer science is concerned with problem solving and it is certainly true that problem solving is a principle focus of computer science. This definition restricts the field to problem solving using machines. Did you know that putting things in order--from smallest to largest--sorting, is a fundamental idea in computer science? Computer scientists have been (and continue to be) obsessed with how fast you one can arrange items from smallest to largest using a computer.



"I'm no computer expert, but I don't think this should be happening."

2.2 Computer Science is the Study of Algorithms

As humans, we experience many problems in our lives. Some are trivial (I wonder what time it is?) some are extremely complex (what am I doing to do with the rest of my life?) and some have no solution (how can I become a streetcar?). Problems are the stock in trade for computer scientists where their solutions are expressed in three ways:

Plan	Solution to a problem expressed in prose for the purposes of informing other people. Usually descriptive in nature with some prescription.
Algorithm	An unambiguous, a set of rules for solving a problem in a finite number of steps. Usually prescriptive in nature with little description
Program	The algorithmic solution to the problem translated into a computer programming language

A plan is a detailed proposal for achieving a goal. It is a means for gaining a common understanding of how a goal will be achieved. Because humans are the target audience many components of a plan are assumed to be understood by all the humans hearing it, the plan tends to be **descriptive** of what needs to happen to achieve success and does not necessarily prescribe how every step is to be achieved.

An algorithm is an abstract recipe, **prescribing** a process that might be carried out by a human, by a computer or by some other means. In essence, it is the formalization of a plan.

Algorithms have the following characteristics

- Algorithms are precise. Each step is explicit and unambiguous

- Algorithms are effective. The task is always done as required.
- Algorithms have a finite number of steps.
- Algorithms (usually) terminate.

Using this definition one might consider some of the underlying areas of computer science to involve the design, structure, analysis, efficiency and correctness of algorithms, the mathematical properties of algorithms, the data they manipulate, the languages in which they are implemented. Finally, one might study the limitations of algorithms.

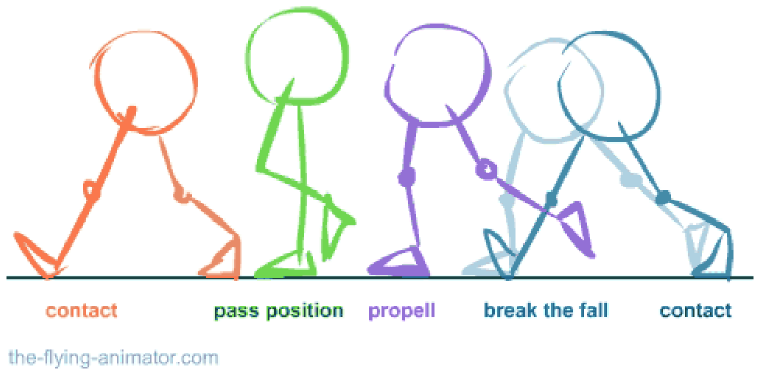
More on what a program is below.

2.2.1 An example: The Walking Problem

Do you remember how you learned how to walk? Well no one else does either. The problem is that most people are so good at it that they don't even think about the process.

2.2.1.1A Plan for walking

Starting from the standing position, lift one leg and lean forward, letting the leg swing past the stationary leg until your foot contacts the floor. Push forward with the other foot, lifting your leg and swinging it forward until it passes the first leg and the other foot contacts the floor. Repeat these motions until it is time to stop.



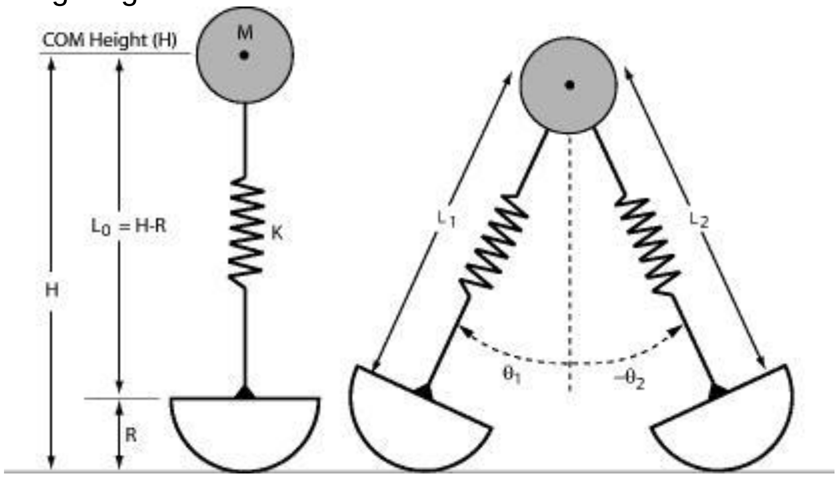
2.2.1.2An Algorithm for walking

The following assumptions are made:

- There are 2 legs,
- There is nothing above the hip,
- Gravity exists to hold the legs to the surface,
- The surface being walked over is flat and perpendicular to the force of gravity,
- Each leg is capable of deforming to reduce its length as indicated below, and

- Each leg is capable of creating a motive force at the hip to propel a leg forward with sufficient force to achieve angle Θ.

Given the following diagram:



Where:

M: hip pivot point at the joint,
K: knee,
H: height of the leg from M to surface being walked on,
R: raised foot distance,
L_0: H-R,
L_1: leg 1 with height H
L_2: leg 2 with height H
Θ: Angle from vertical for leg L_n
T: Torque applied at M to achieve Θ

The algorithm is given:

Step 1: Starting from the position indicated in the left diagram above
Step 2: Let n = 1
Step 3: Reduce length of L_n to L_0
Step 4: Apply T to M of L_n
Step 5: Wait until angle of L_n from vertical is – Θ: Increase length of L_n to H
Step 6: If n == 1 let n = 2 else let n = 1
Step 7: Go to step 3.

2.3 Pseudocode

Algorithms present the solution to a problem as a logical sequence of steps, usually presented with rigor and mathematical notation. In many instances, it is possible to translate an algorithm directly into a computer programming language. Sometimes it is desirable to clarify an algorithm by first writing it in Pseudocode: an English-like language which has no formal definition but might be best described as “programmer slang”.

The reasons for making this intermediate translation is to make it a little easier to understand some of the notation for people who might be somewhat unfamiliar with the formalities of algorithms and to start the design process that informally employs programming language constructs (like decisions) and will eventually become a computer program.

3 Computer Science Concept: Abstraction

Abstraction is a powerful tool of notation that is particularly useful in handling complexity.

Abstraction is the naming of an idea so that it may be easily referred to in more complex settings.

For example, in mathematics the idea of computing the average of a sequence of n numbers x1, x2, ..., xn is sometimes written as,

avg(xi).

Similarly, the derivative of a function f(x) is written as

f'(x).

If we examine our walking algorithm from the previous section we used abstraction repeatedly. For example the term L was used for a leg. A human leg is very complex, we do not know how it works exactly but when we name it L, we get a kind of power over the concept and it can be used without further definition (unless we have to describe an algorithm to create one).

4 Computer Science Concept: Conceptual Models and Modeling

A conceptual model represents “concepts” and the relationships between them. Conceptual models are built without regard to design or implementation details. For example, what programming language to use or on what computer or computers the program will run on. The aim of a conceptual model is to express the meaning of terms and concepts used by experts to discuss the problem, and to find the correct relationships between different concepts. The conceptual model attempts to clarify the meaning of various, usually ambiguous terms, and ensures that problems with different interpretations of the terms and concepts cannot occur[2].

[2] Adapted from https://en.wikipedia.org/wiki/Conceptual_model_(computer_science)

Computer scientists build conceptual models so that they can study, analyze and predict the behavior of the systems they are studying. Wherever possible, these models will be designed using some notation that is useful, not only for thinking and analysis, but also as a means of implementing that model on a computer.

In our walking algorithm, you will notice that the diagram showing the stick figures is actually a model of what the system is doing at any particular time.

5 What is a Computer?

Now that we have the computer science mumbo-jumbo out of the way we still have a problem. What is a computer? Well we could use a dictionary to help us.

5.1 Potential Definitions

- (Collins English Dictionary): 'A device, usually electronic, that processes data according to a set of instructions'
- (Que's Computer Users Dictionary): 'A machine capable of following instructions to alter data in a desirable way and to perform at least some of these operations without human intervention'.
- (Microsoft Press Computer Dictionary, 3rd Ed.: 'Any machine that does three things:accepts structured input, processes it according to prescribed rules, and produces the results as output'.

5.2 Digital vs. Analog

Actually, for the longest time (in dog years at least) people have considered two types of computers.

- Digital Computers -store data in discrete units and perform arithmetical and logical operations at very high speed. For example, your digital watch is a digital computer in disguise.
- Analog Computers – have continuous rather than discrete input; widely used in laboratory equipment to monitor ongoing, continuous changes and record/graph these. For example, your little brother is an analog computer as are you as is a mercury thermometer, a wind up clock, a…

In this course we will be dealing solely with digital computers.

5.3 Acting like a Computer

In order to prepare for some of the challenges of modern computer architecture[3] it may help to take some time out and act like a computer. This can be done with the aid of an

[3] A big word indicating the functional components of a computer and how they interact.

everyday device and a simplified problem. The device is the common numeric handheld calculator and the problem is to add a set of numbers (3) together.

The numbers or "data" are as follows

4 ,
7, and
9 .

The concept of "add a set of numbers together" is in fact a series of discrete operations performed in a particular order that form an algorithm. The steps are :

1 clear the calculator
2 load first number
3 add
4 load second number
5 add
6 load third number
7 add
8 stop

The data is

1	4
2	7
3	9

5.4 *The Von Neuman Architecture (the modern computer)*

This was the way it was done in the 1940's. The calculations were much more complex, involving the evaluation of mathematical equations for the prediction of artillery shell trajectories, statistics, etc. Also the data was more extensive, involving census data, results from studies or experiments, etc. However the process was much the same. A human operator of a calculator--the "computer" followed the program or menu of operations and applied them to a set of data.

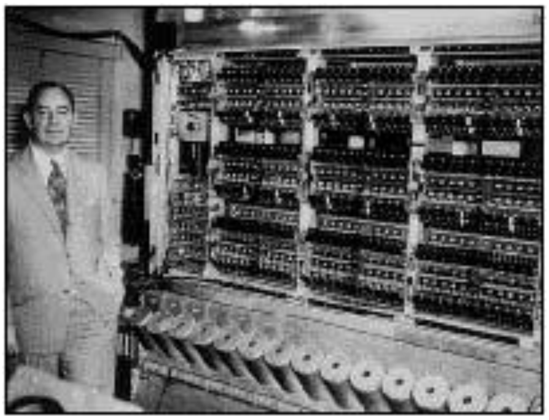


Figure 3 John Von Neuman, 1903-57

"If people do not believe that mathematics is simple, it is only because they do not realize how complicated life is."

John Von Neuman

John Von Neuman--who is considered by some to be the father of the modern computer--proposed that the program of operations be coded and stored in a device which would be attached to a "sequencer" which would fetch the coded instruction, decode it and cause the calculator to perform the coded operation. Furthermore, he proposed that the data be encoded and placed in the same storage device along with the instructions.

This automating of the calculation process allows it to proceed at the native speed of the calculating device rather than the speed of the human operating the calculator. This is extremely significant in that refinements in the computing process are left to the engineers or "plumbers". They can concentrate on the refinement of their device and "we" can concentrate on its application or "cooking" (Did I just equate engineers with plumbers?)

Thus was born the modern computer that has remained more or less unchanged apart from the process refinements in the manufacturing of integrated circuits that is the art that keeps things hopping in the computer hardware field. Computers have become faster, cheaper, physically smaller and logically larger at an exponential rate.

5.4.1 The Von Neuman Architecture Hardware Components

Hardware is the equipment associated with a computer system. Modern digital computer will inevitably contain;

- Input/Output Peripherals
- Memory
- Processing Unit(s)

5.4.2 Input/Output peripherals

These are the devices attached to a computer that allow you to tell the computer things and let it tell you things. Sometimes a device (physical piece of hardware) will perform both functions at different times (disk drive for example).

- Keyboards,
- mice,
- track balls,
- Tape drives,
- CD ROMS,
- printers
- hammers ;-)

5.4.3 Memory

Computer memories are capable of storing data over time.

The memory of a computer is not a single entity (although in many cases you can think of it that way). It is a collection of units dispersed in space and function with a spectrum of speeds, capacities and costs.

Memory can only store bits because the tiny circuitry that makes up each spot in memory only knows how to be on or off (1 or 0) (Actually on or off or +5V or 0V).

Primary memory refers to memory that is typically on-board the circuit board that is your CPU board. **Secondary memory** is everything else.

If you forget all that, you can think of memory as a long narrow sheet of paper. You can only write one thing on each line of the paper and then you must go to the next line. The same is true for reading it. So, to get from any one line to any other you just need to add or subtract an integer value from where you are to where you want to go....you can jump around in memory.

5.4.4 Processing Unit

The Central Processing Unit (CPU) is responsible for executing the programs that you write and that are resident in your computer already (like the operating system which is also a program that runs your program).

5.5 *Hooking the whole thing up*

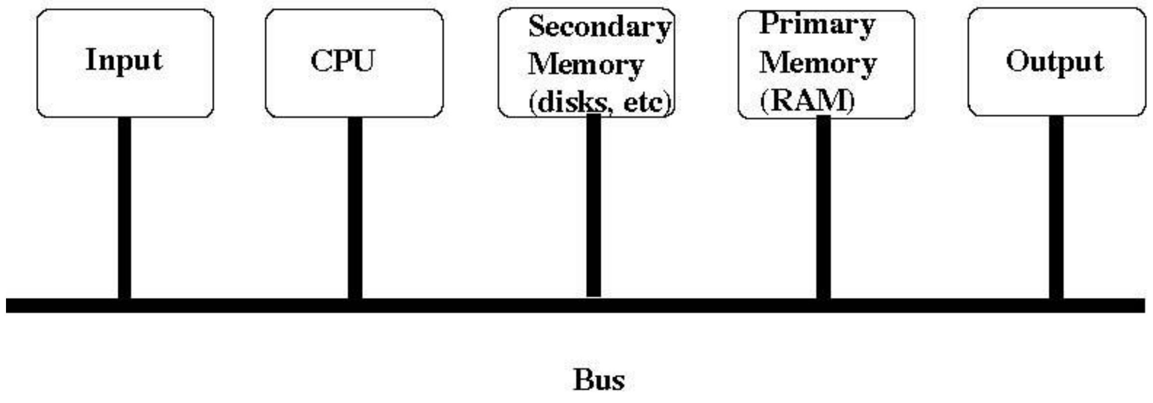


Figure 4 Single bus computer architecture

All these devices share data across a bunch of wires called a **bus**. The width of a bus is the amount of bits that it can carry in parallel at once. So, a 16-bit bus can carry 16 bits at once.

6 Computer Programs

A program is an algorithm described to a computer. How this happens is a very long story but no matter how you tell the story, you must remember 2 things: Computers are,

- exceedingly fast
- surprisingly stupid

6.1 *Binary and Machine Codes*

Because of their architecture computers,

- Only understand binary (1’s and 0’s)(on and off)
- must be told everything exactly the way they want it

Of course, binary digits (bits) can only be on or off, so some clever people came up with the notion of grouping several bits together. Now you can encode more than two things in this way (Why is it that you can only encode 2 things with 1 bit?).

Suppose you had two bits grouped together. You could represent 4 things. For example we could encode how well you are doing in this course.

Mark	Meaning
11	Excellent work
10	Good work
01	Poor work
00	Oh no

So, we could encode everything as grouped 1's and 0's.

The standard is to group 8 bits into something called a **byte**. 8 bits give you the possibility to represent 256 things (0 to 255).

To confuse things a bit more, each byte can represent either a number (data) or an instruction. Forget this for the moment and we will assume each byte is a number representing an instruction. Instructions represented in this way are called **byte codes**.

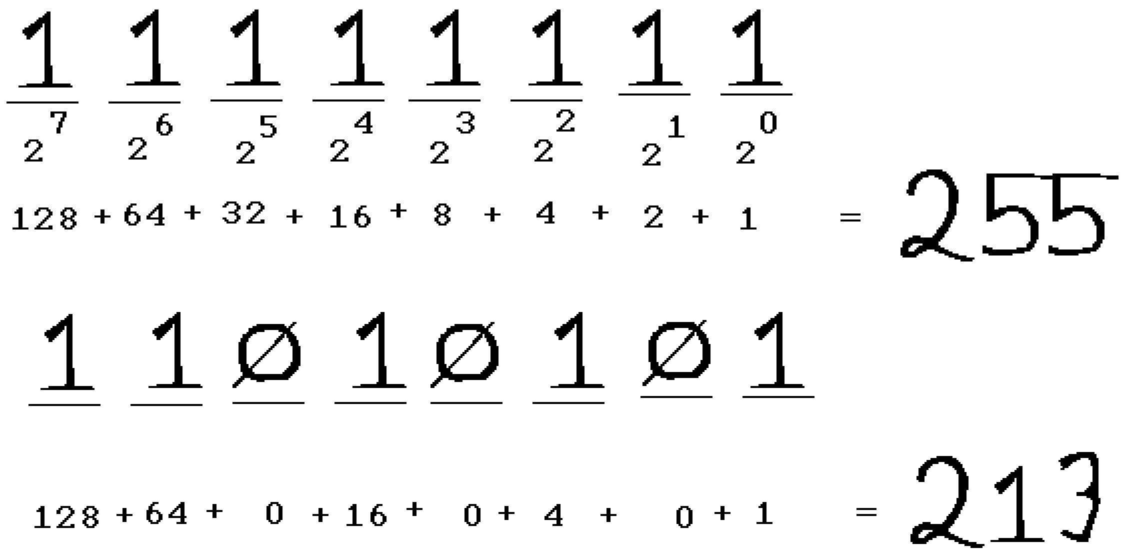


Figure 5 Converting between binary and decimal numbers

6.2 Language Interpretation

These byte codes are called **Machine Codes** or **Machine Language.** Computers inherently understand machine codes because they can be executed by the machine's processing unit directly. It just reads a code, **interprets** what it means and does that action.

6.2.1 Operations and Operands

Each instruction to the computer usually has two parts the command being performed (**operation**) and the data it is being performed on (**operands**).

If you make the bytes represent words you can tell a computer about your algorithm. Here is an example dictionary of bytes/words (this is not a real machine language just an example).

Example Operations

Byte Code	Operation	# Operands
00000000	OPEN	2
00000010	PULL	2
00000011	PICKUP	1
00000100	OPEN	1
00000101	MOVE	1
00000110	DROP	1

Example Operands

Byte Code	Operand
00000000	DOOR
00000001	ROOM
00000010	SHEETS
00000011	BED
00000100	BROTHER
00000101	TOPOFBED
00000110	HALL
00000111	BOTTOMOFBED

We could tell the computer our algorithm for dragging us to bed (see above)

```
00000000 00000000 00000001
00000101 00000011
00000011 00000010
00000010 00000010 00000111
00000110 00000010
00000101 00000100
00000011 00000100
00000101 00000101
00000110 00000100
00000011 00000010
00000101 00000101
00000110 00000010
00000101 00000110
```

6.3 Assembly Language

This is all well and good but there must be an easier way...who can remember all those codes and what happens if we accidentally swap a few zeros and ones…disaster! An improvement was **assembly language** that shortened all the commands into assembly codes and grouped some of the related stuff together.

```
OPEN DOOR ROOM
MOVE HEADOFBED
PICKUP SHEETS
MOVE FOOTOFBED
DROP SHEETS
MOVE BROTHER
PICKUP BROTHER
MOVE HEADOFBEAD
DROP BROTHER
MOVE FOOTOF BED
PICKUP SHEETS
MOVE HEADOFBED
DROP SHEETS
LEAVE
```

6.3.1 Translation and Interpretation

Computers can almost execute this stuff directly (interpret it) since only a simple **translation** is required from the **mnemonic** (English-like command). This process is illustrated below;

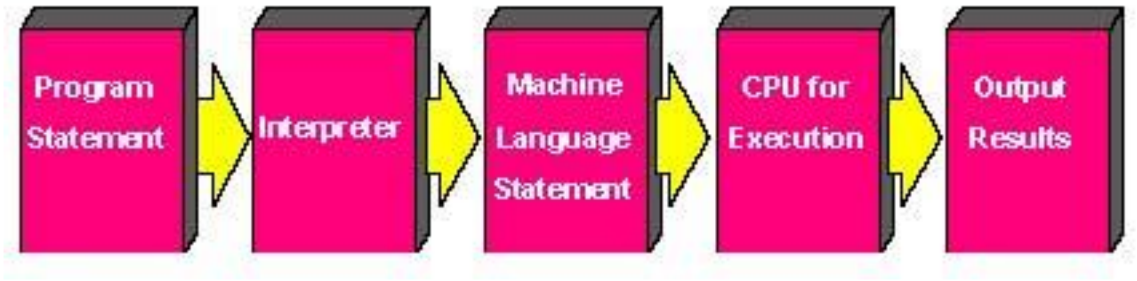


Figure 6 The process of instruction interpretation

6.4 High-Level languages

It soon became clear that for all complex problems the code would be horrendously long to write and next to impossible to fix if something went wrong. That's where high-level languages come in.

High-level languages hide some of the obscurity of the machine. In Machine Code and Assembly Language the programmer must specify everything. This is a problem because many times the same code is written again and again which could lead to errors as things are mistyped or later modified but only in some places. In addition machine code and assembly languages are specific to individual CPUs. If you move from one type of CPU to another the codes can be quite different.

Higher level languages (like Java) avoid this problem by making each command group together a whole bunch of lower-level machine codes--you don't have to keep repeating these codes yourself--avoiding introducing more errors.

```
import brothers.*;
Public class movebrother
{
     public static void main(String[] args)
     brother = new Brother;
     brother.move("bed");
     brother.cover;
}
```

There are of course even higher-level languages--the higher you go, the more stuff is done for you. The repetitious stuff now happens in the background.

Language Classification	**Examples**
Machine Code	68000, 8080
Assembly Code	68000, 8080
Third Generation Languages	Java, C, C++, COBOL, FORTRAN, Lisp
Fourth Generation Languages	SQL, Hypertalk, Smalltalk, TKL/TK
Markup Languages	HTML, SGML

6.4.1 Compilation, linking and executing

Computers cannot interpret high level languages directly because the codes are more complex and rely on functionality which may be hidden someplace either in the depths of the language or in a library of common routines. The computer must now go through several steps;

- It must translate the high-level language program into machine code routines, so that it can understand it. This is call **compiling** the program. Each one of those high-level language commands is replaced with the lower level code it represents. This is great because the compiler will not allow you to enter a program which is syntactically incorrect. A **syntax error** is one in which you break the language rules and is the only type of error picked up by a compiler
- It must get the code resources that the program needs from libraries of common code. These functions are not actually part of the high-level language but may be written in it. For example, in C there is no command to sort a list of numbers but you could write a program to do it. If you wanted to reuse this program you could store it in a library from which other programs could use it. This is called **linking** or **link editing**. You need to link because the language is only so big and you might need to call sub-programs which are stored in libraries of commonly used code which you link to. Linking allows you detect error like--the darn thing isn't in the library. This is a **linking error**.

- Finally the computer is able to execute the translated and linked program--called **execution** or go! Note: at this stage the machine code is being executed--not the high level language. An error at this point is called an **execution error** and normally has something to do with the logic of your program--meaning that although what you have written makes sense syntactically and links fine you still cannot solve the problem because what you are saying in the program makes no sense **semantically**. An error may also mean there is some sort of physical problem. For example if your program is designed to print something on a printer and you run it without an actual printer attached to the computer you will receive an execution or **run time error**.

This process is illustrated below;

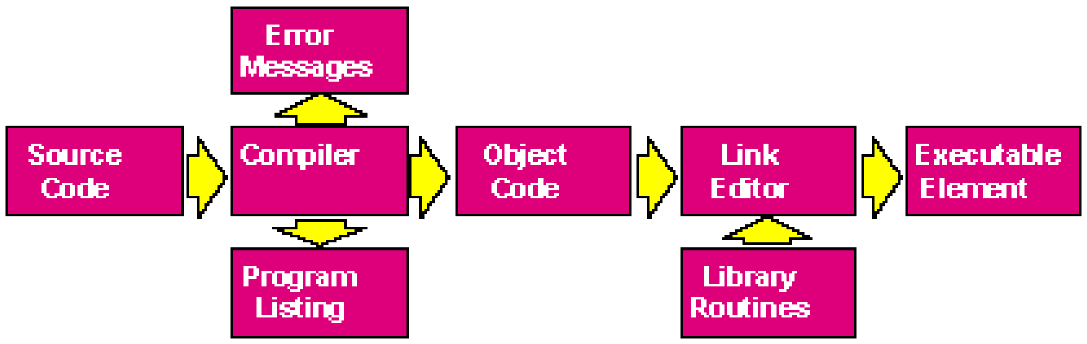


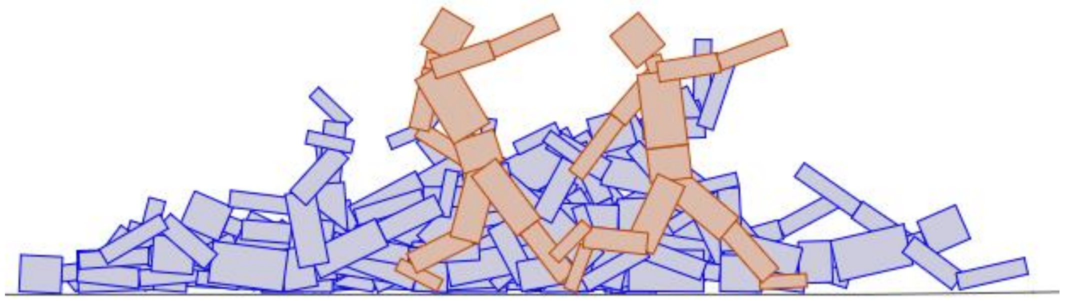
Figure 7 The compilation process

6.5 A program for walking

The interesting thing about algorithms is that they are successful if tested by writing the program that implements the algorithm for such a program could be written for our walking algorithm but a much more interesting implementation can be found here:

http://rednuht.org/genetic_walkers/

This site implements a form of machine learning where a genetic algorithm learns how to walk (kind of drunkenly).



7 What is Java?

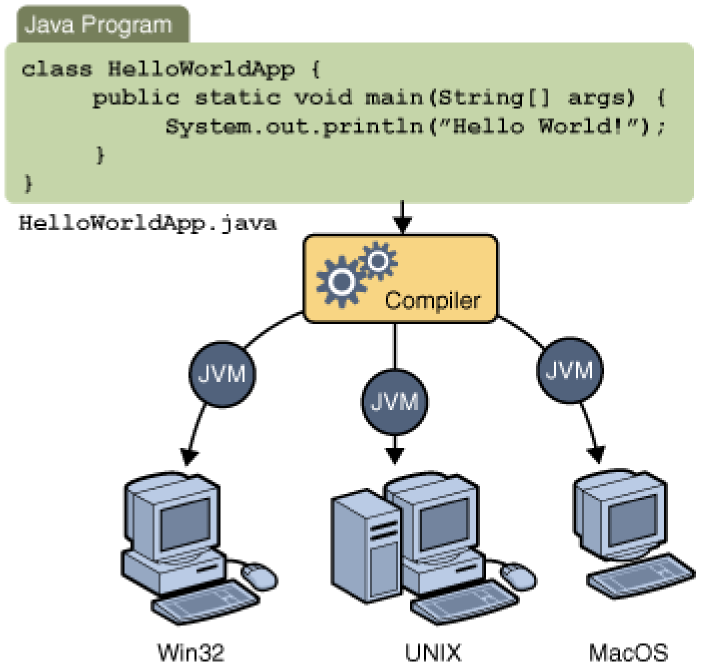
Java is the high-level programming language we use in this course to teach you some of the basic concepts of computer science. It shares many of the characteristics of other high-level programming languages but it also has some interesting distinctions that we will discuss as we proceed.

7.1 History

The Java programming language and environment is designed to solve a number of problems in modern programming practice. Java started as a part of a larger project to develop advanced software for consumer electronics. These devices are small, reliable, portable, distributed, real-time embedded systems. When developers at Sun Microsystems started the project they intended to use C++ (another high-level language), but encountered a number of problems. Initially these were just compiler technology problems, but as time passed more problems emerged that were best solved by changing the language being used.

7.2 Java Virtual Machine

The designers of Java chose to use a combination of compilation and interpretation in order to make Java programs run on computers in a way slightly different than either compilation or interpretation. Programs written in Java are compiled into machine language, but it is a machine language for a computer that doesn't really exist. This "virtual" computer is called the **Java Virtual Machine (JVM)**. The machine language for the Java virtual machine is called Java **bytecode**.



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

Figure 8 Purpose of the JVM

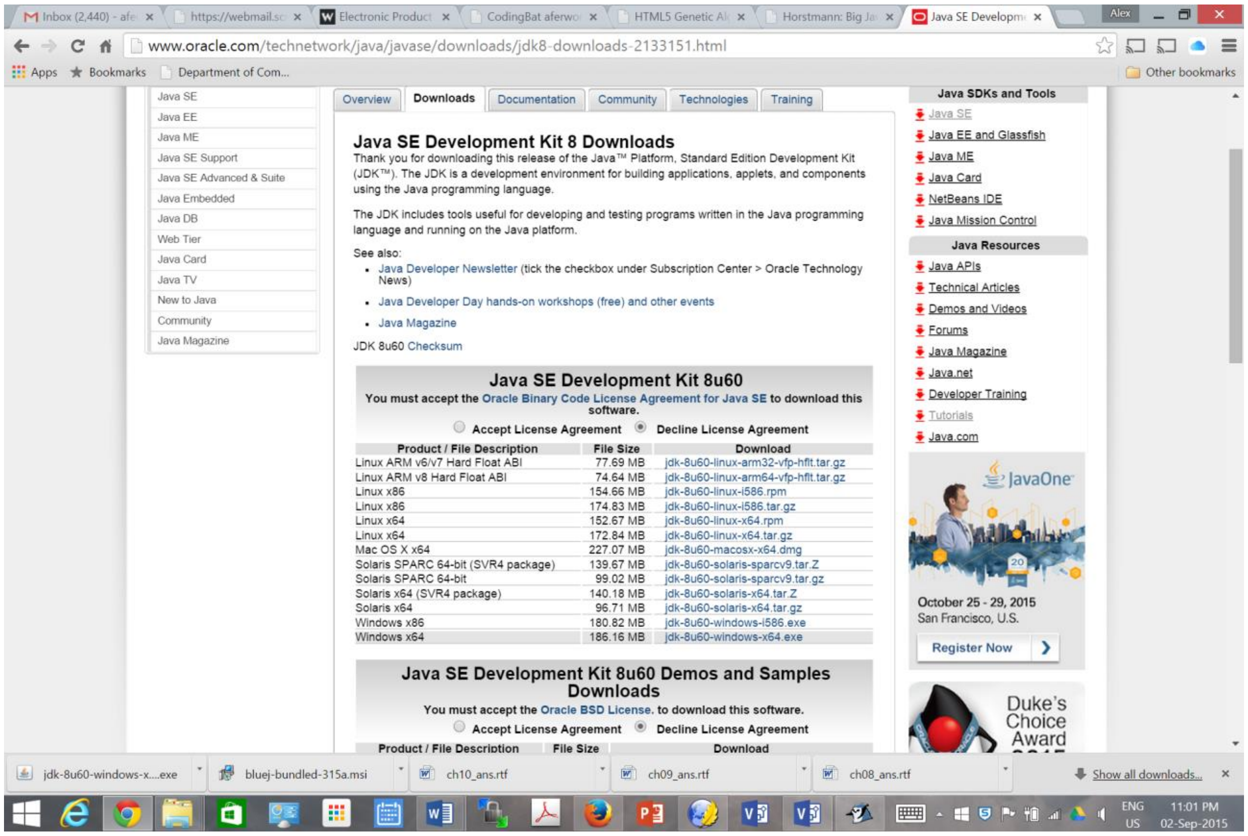
Why, you might wonder, use the intermediate Java bytecode at all? Why not just distribute the original Java program and let each person compile it into the machine language of whatever computer they want to run it on? There are many reasons. First of all, a compiler has to understand Java, a complex high-level language. The compiler is itself a complex program. A Java bytecode interpreter, on the other hand, is a fairly small, simple program. This makes it easy to write a bytecode interpreter for a new type of computer; once that is done, that computer can run any compiled Java program. It would be much harder to write a Java compiler for the same computer.

Furthermore, Java programs are meant to be downloaded over a network. This leads to obvious security concerns: you don't want to download and run a program that will damage your computer or your files. The bytecode interpreter acts as a buffer between you and the program you download. You are really running the interpreter, which runs the downloaded program indirectly. The interpreter can protect you from potentially dangerous actions on the part of that program.

I should note that there is no necessary connection between Java and Java bytecode. A program written in Java could certainly be compiled into the machine language of a real computer. And programs written in other languages could be compiled into Java bytecode. However, it is the combination of Java and Java bytecode that is platform-independent, secure, and network-compatible while allowing you to program in a modern high-level object-oriented language.

8 Before you can use Java you need to install the JDK

This section is for all those people who own their own computer that does not have Java installed on it already you will have to install it. Oracle Corporation took over from Sun Microsystems when they bought the company and now maintains the download site where you can download the appropriate version for your computer. You will need to install something called the "Java Development Kit" (JDK). As the JDK tends to be updated relatively frequently, this site moves around a bit. The easiest way to find it is to Google it with the search string: "Java Development Kit download" and a screen something like this will show up.



Java SE Development Kit 8 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

See also:

- Java Developer Newsletter (tick the checkbox under Subscription Center > Oracle Technology News)
- Java Developer Day hands-on workshops (free) and other events
- Java Magazine

JDK 8u60 Checksum

Java SE Development Kit 8u60

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement [✓ Decline License Agreement]

Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	77.69 MB	jdk-8u60-linux-arm32-vfp-hflt.tar.gz
Linux ARM v8 Hard Float ABI	74.64 MB	jdk-8u60-linux-arm64-vfp-hflt.tar.gz
Linux x86	154.66 MB	jdk-8u60-linux-i586.rpm
Linux x86	174.83 MB	jdk-8u60-linux-i586.tar.gz
Linux x64	152.67 MB	jdk-8u60-linux-x64.rpm
Linux x64	172.84 MB	jdk-8u60-linux-x64.tar.gz
Mac OS X x64	227.07 MB	jdk-8u60-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.67 MB	jdk-8u60-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.02 MB	jdk-8u60-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	140.18 MB	jdk-8u60-solaris-x64.tar.Z
Solaris x64	96.71 MB	jdk-8u60-solaris-x64.tar.gz
Windows x86	180.82 MB	jdk-8u60-windows-i586.exe
Windows x64	186.16 MB	jdk-8u60-windows-x64.exe

Java SE Development Kit 8u60 Demos and Samples Downloads

You must accept the Oracle BSD License. to download this software.

Accept License Agreement [✓ Decline License Agreement]

Product / File Description	File Size	Download

Select the version of the JDK that matches your computer.

9 Installing an Integrated Development Environment (IDE)

You can use Java from a command line or you can use a number of IDEs that are freely available. One such IDE is called BlueJ. If you visit the site: www.bluej.org you can download and install it relatively easily.

10 Getting on with Writing Java

In the following sections we will walk through a simple Java program and discuss how it works and how to make it work.

10.1 Applications vs. Applets

Java programs can be written and executed in two ways:

- Stand-alone **application** from a command line of an operating system.
- **Applet** that runs under control of a Java-capable browser.

Programming an "application" in Java is significantly different from programming an "applet." Applets are designed to be downloaded and executed on-line under the control of a browser. Hence, their functionality is restricted in an attempt to prevent downloaded applets from damaging your computer or your data. No such restrictions apply to the functionality of a Java application.

All Java programs consist of one or more class definitions (wait until we talk about object orientation). In this course, we will refer to the primary class definition for a Java application as the controlling class.

A stand-alone Java application requires a method (Hey, there is that object stuff again) named **main** in its controlling class. An Applet does not require a main method nor is one allowed.

10.2 The programming cycle in a nutshell

The cycle for writing code and working with it is the same on all computers and takes the following form:

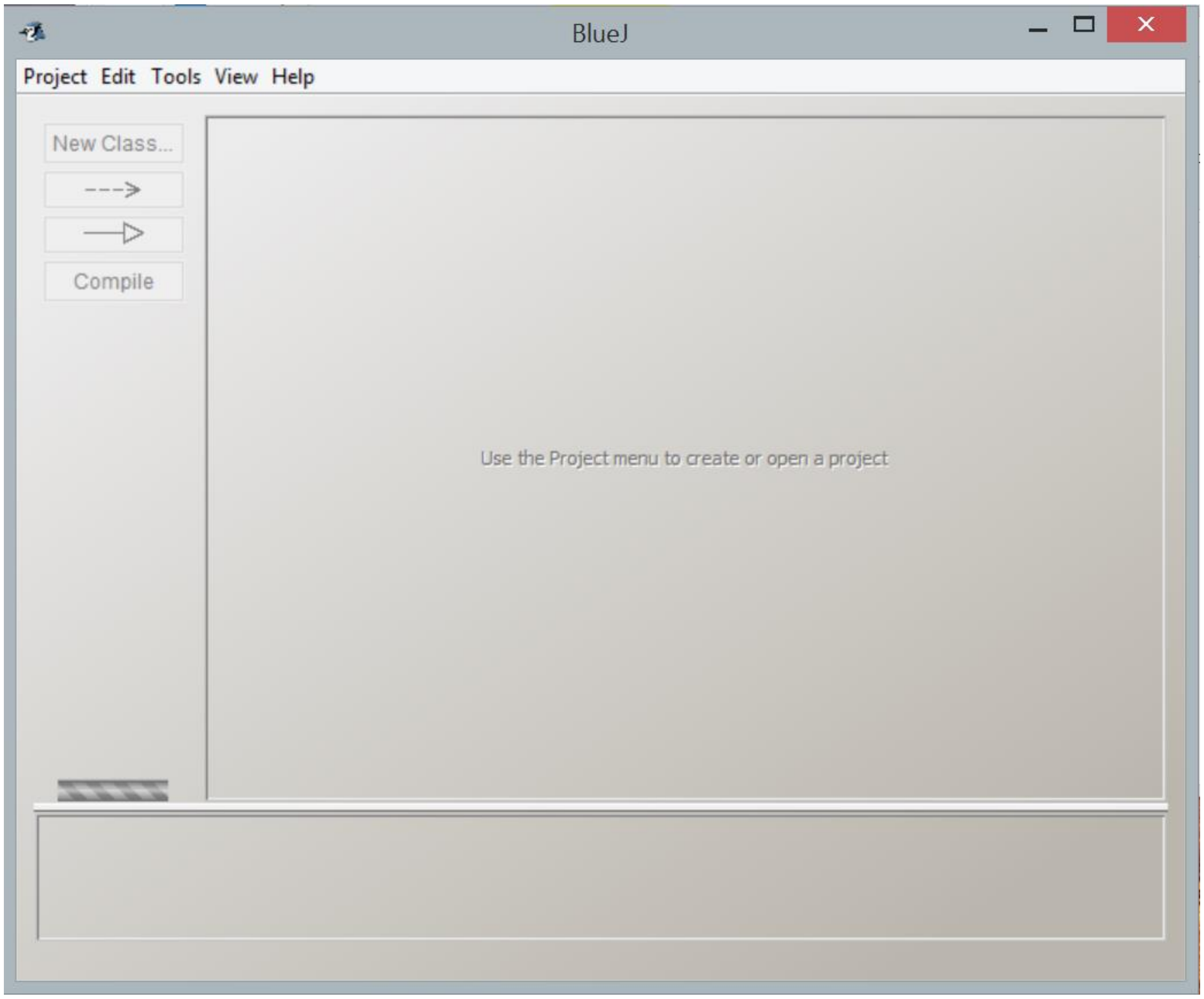
- Understand the problem
- Develop an algorithm
- Write the code
- Enter/modify the code on to the system using an editor
- Compile the code
- Fix compilations errors and keep going back to step 4 until the code compiles
- Run the compiled code
- If there are execution errors go back to an appropriate earlier step to fix the problem(s)

10.2.1 Your First Java Program

In 1978, Brian Kernighan and Dennis Ritchie wrote, in their book *The C Programming Language*, “the first program to write is the same for all languages.” They were referring, of course, to the “Hello World” program. The Java implementation of this program is the subject of the next few sections.

The first thing you need to do is somehow create a file with your program on the computer you are using. If you are using an IDE like blueJ just double click on its icon to get started.

A window should open on your desktop that looks very much like this:

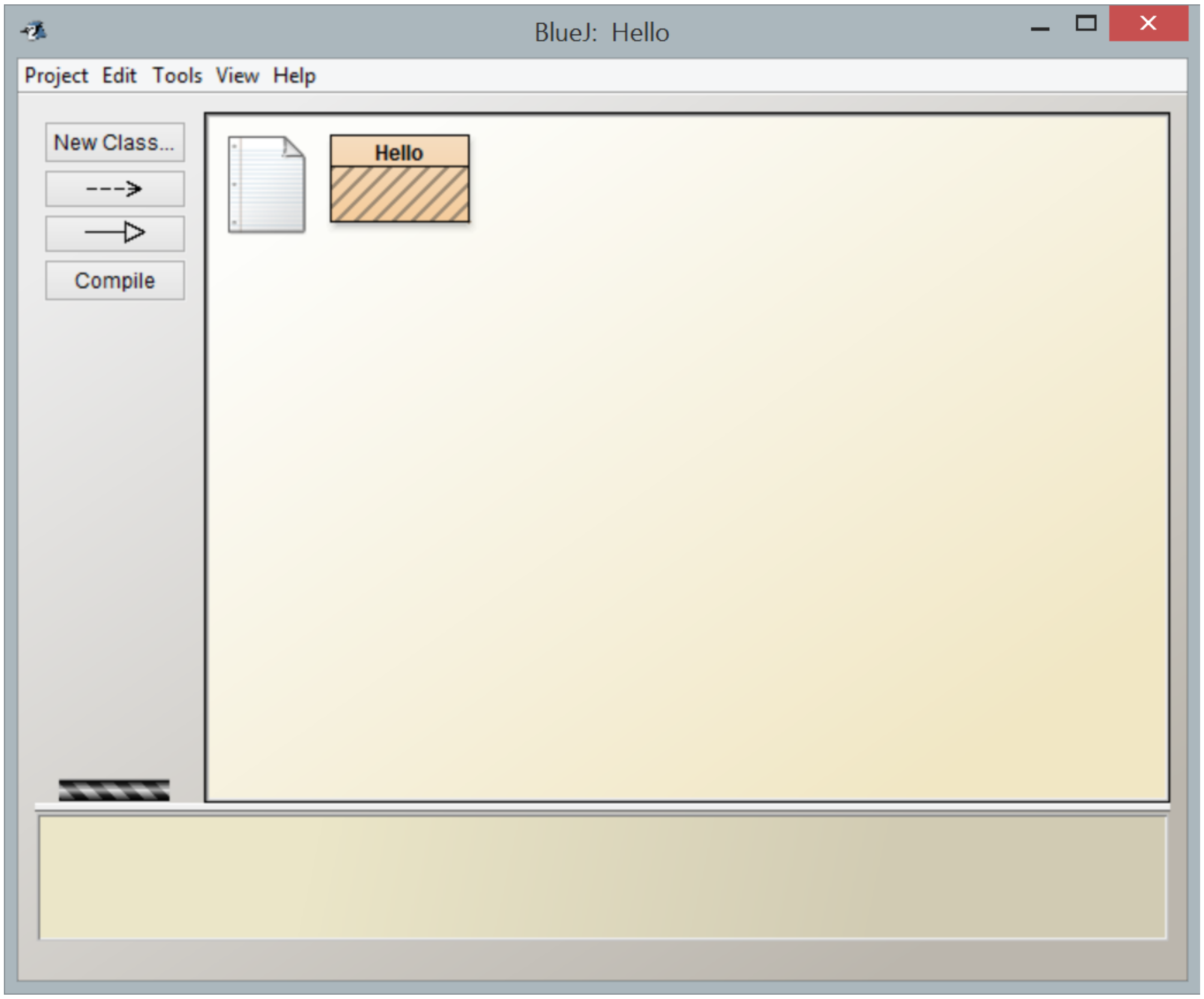


Use the Project menu to create or open a project

From the “Project” menu, select “New Project…”

In this example, I will call our project “Hello”. Projects in BlueJ provide a convenient place to store related Java code.

In order for us to write our program we need to create something called a “Class”. Press the “New Class” button on the window and call your new Class “Hello”. You will end up with a with a window that looks like this:



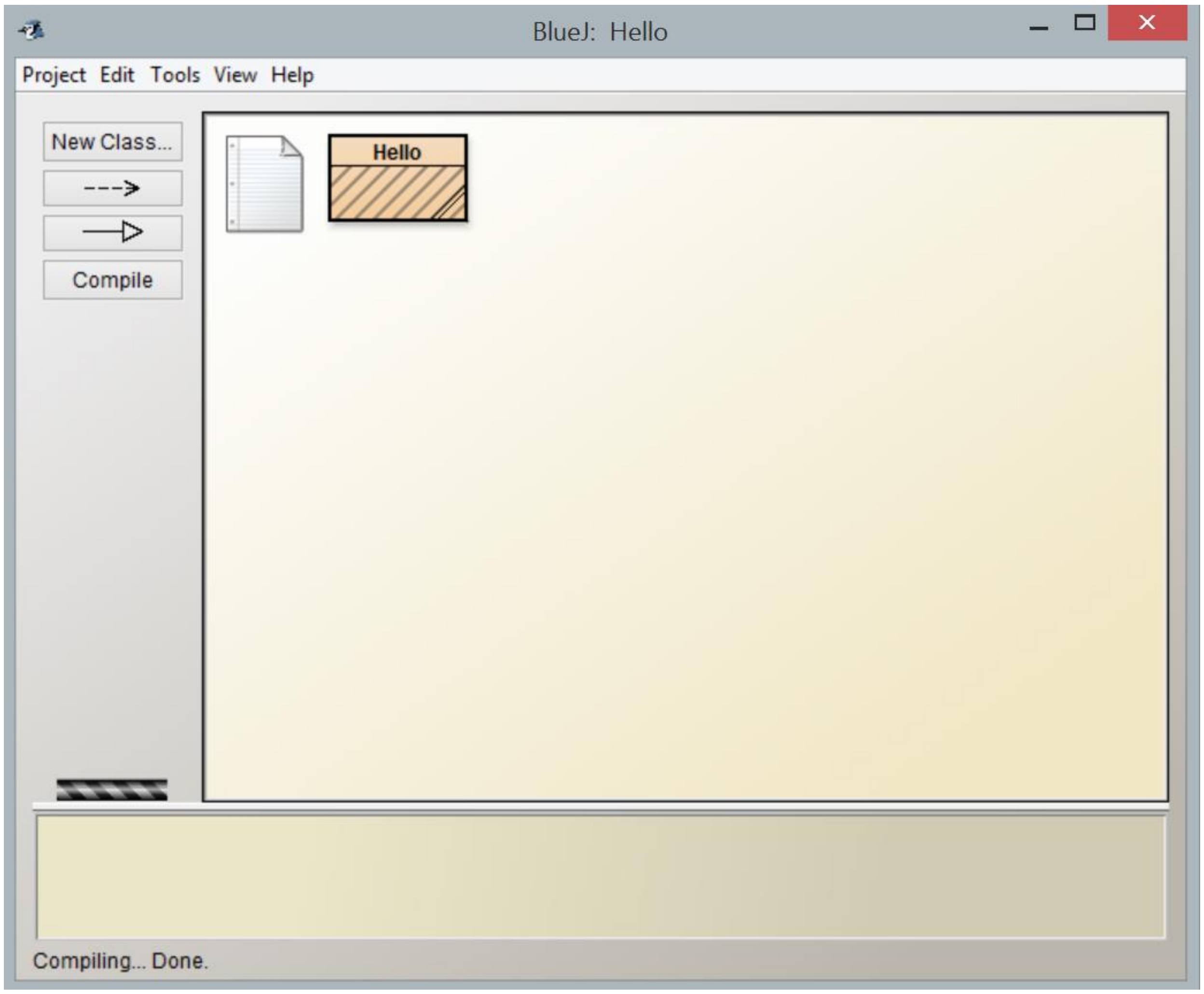
Double click on the Hello box. This will let you start editing your Java program. You will notice there is an awful lot of Java already in the file. BlueJ tries to be your friend by populating a file with common Java stuff. At the moment, it is just confusing so delete everything in your Hello file and replace it with the Java program below.

```
//Example: Hello Java!
public class Hello
{
     public static void main(String[] args)
     {
          System.out.println(“Hello Java!”);
     }
}
```

Once you have pasted the program inside, close the window.

Compiled Java programs are stored in "bytecode" form in a file with an extension of .**class** where the name of the file is the same as the name of the controlling class in the program. In our example above the controlling class (and the only class) is the Hello class.

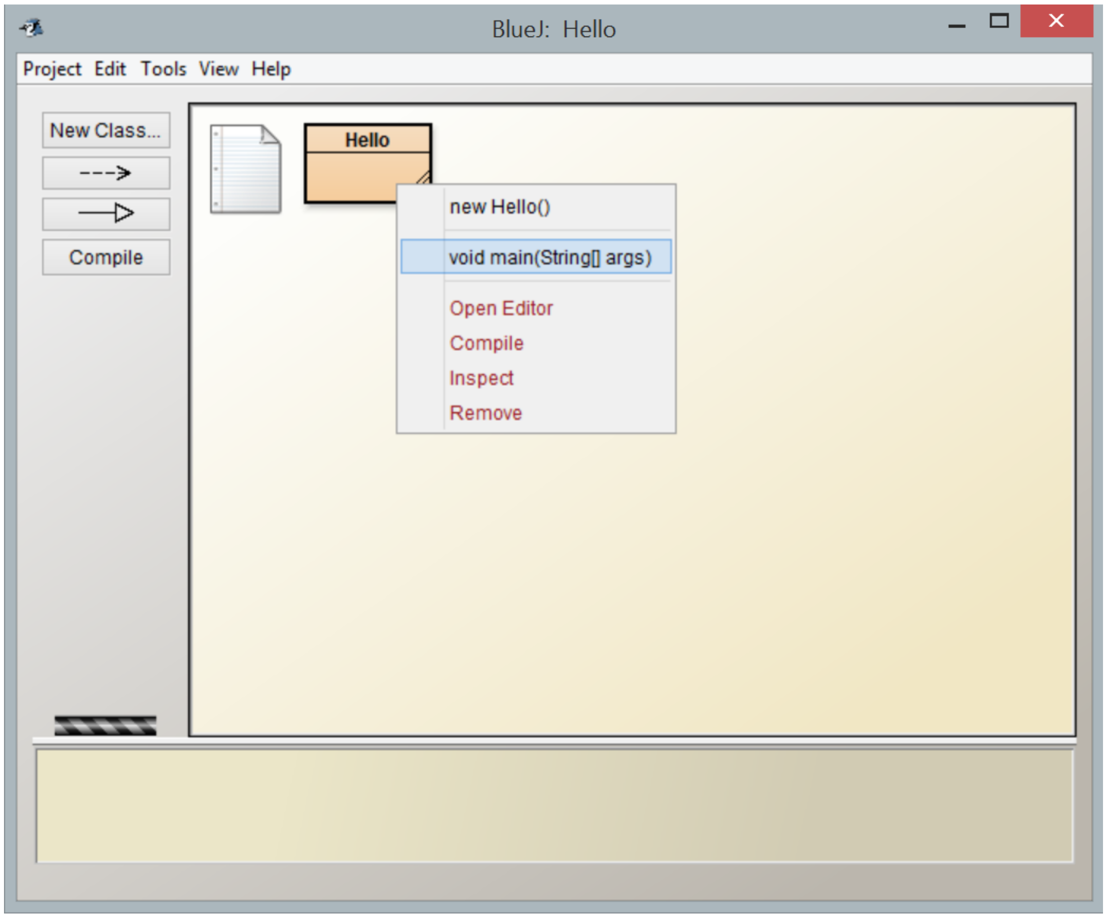
In order to generate the “bytecode” you must “compile” your application. This is done by pressing the “compile” button. If everything goes well (and it rarely does) you will end up with a window that looks like this:



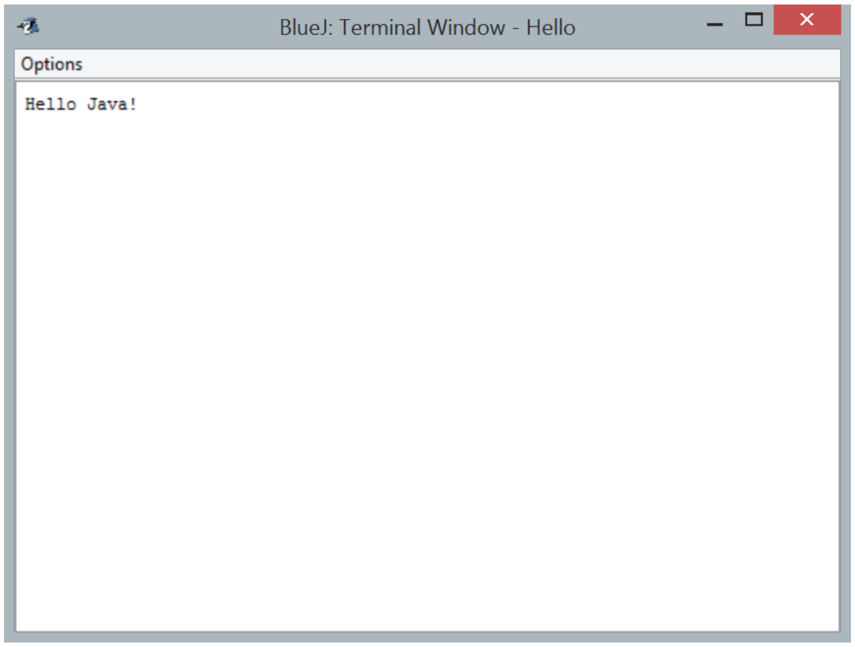
Compiling... Done.

When a Java application is started, the Java interpreter finds and invokes the main method in the class whose name matches the name of the class (in this case, Hello).

To run our Hello program, right click on the Hello box and invoke main() as shown below,



Try it and you will see the output:



```
Hello Java!
```

10.2.2 Understanding what happens in Hello.java

The line,

```
    //Example: Hello Java!
```

is a comment. Comments are a way of describing to yourself and others what the program does. Sometimes the comment only describes what a small part of the program does. This is a really useful facility in most programming languages as it makes it much easier to fixthe code if something goes wrong later on.

Java supports several styles of comments:

```
        /*
            Java/C/C++ style multi-line comment
        */
```

and

```
        // Java/C++ style single-line comment
```

The lines,

```
    public class Hello
    {
        public static void main(String[] args)
        {
```

are a required part of the template. Every program and every applet—every piece of Java code—you write is part of a class. The first line says that we're writing a class named Hello that is a public class, which means that it can be used by anyone.

Every standalone Java program requires a main() method. This is where the Java interpreter begins running a Java program. The second line of the example declares this main() method. It says that the method is public, that it has no return value (i.e., its return value is non-existent or "void"), and that it is passed an array of strings as its argument. The name of the array is args. The line also says that main() is a static method. This is all a bunch of mumbo-jumbo right now but as you learn Java you will see what they mean.

You might as well memorize;

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

Every standalone Java program you will ever write will contain a line that looks exactly like this one. (actually, you can change “args” but people don’t).

The line,

```
System.out.println(“Hello Java!”);
```

is the line that does something we can notice. The System.out.println() method sends a line of output to “standard output”, which is usually the screen.

The lined up { and } represent “blocks” of code. A block of code forms the body of things like classes and methods. Method blocks are always defined within class blocks. As you can see here, we can “nest” blocks within one another.

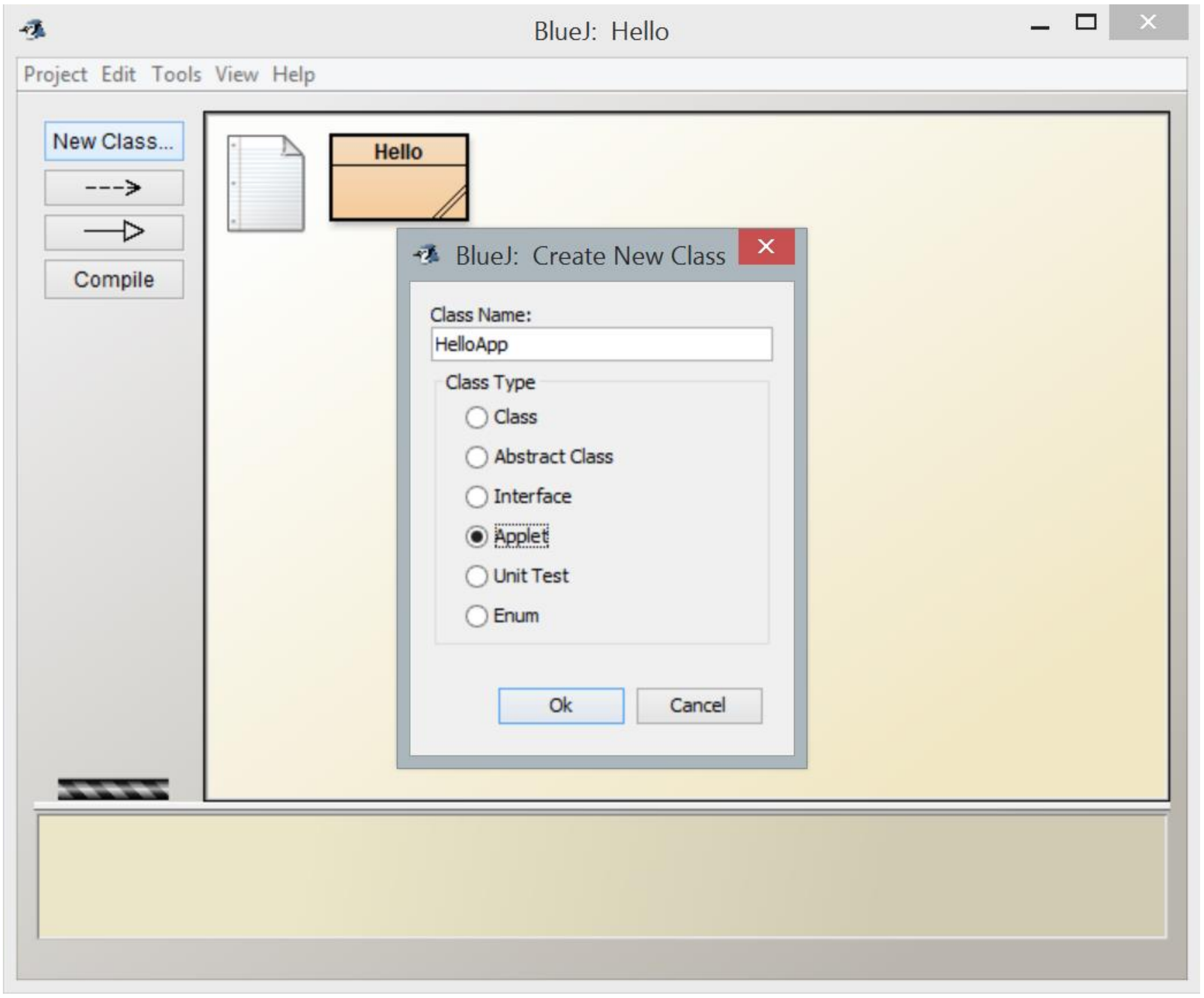
10.2.3 Your First Java Applet

Now that we have seen a simple Java program (application) let us expand our capabilities by making the application into an applet. Recall that an applet—as the name implies—is a kind of mini-application, designed to run by a Java enabled World Wide Web Browser. The Java Development Kit (JDK) also offers an applet viewer which will allow you to execute an applet in it. BlueJ has a built-in applet viewer as well so we will use it.

Remember, an applet must use the resources of its viewer to run while a Java application has all the resources it needs to run internally (stand-alone).

The first thing you need to do is the same as what you needed to do to create an application.

Create a new class as shown in the figure below.



Class Name: HelloApp

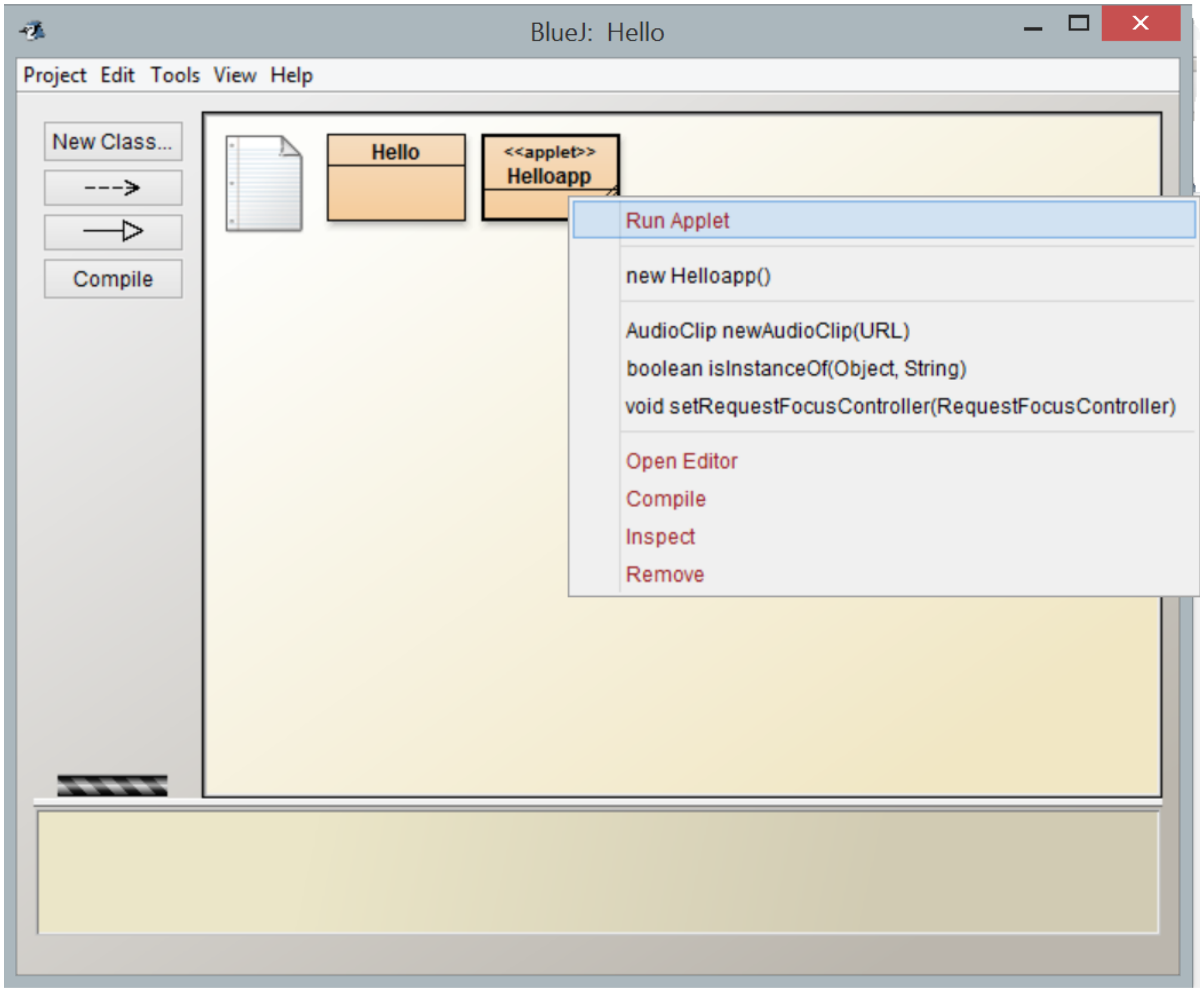
Class Type

Class [✓ Applet] Abstract Class Interface Unit Test Enum

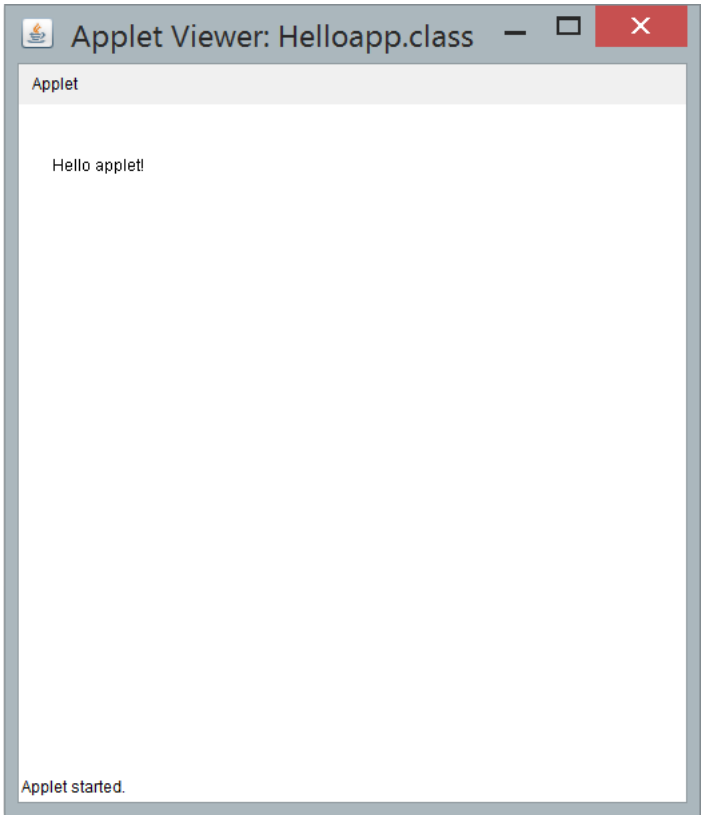
Type in the following Java applet.

```
//Example: Hello (applet version)

import java.applet.*;     // The applet class
import java.awt.*;        // The abstract window class

public class Helloapp extends Applet
{
     public void paint(Graphics g)
     {
          g.drawString("Hello applet!",25,50);
     }
}
```

In order to generate the “bytecode” you must compile your applet. Again, press the compile button in the window. If it compiles without syntax errors, close the window. To run the applet, right click on the Helloapp box and select “Run Applet”.


BlueJ: Hello
Project Edit Tools View Help
New Class...
Compile
Hello
<<applet>>
Helloapp
Run Applet
new Helloapp()
AudioClip newAudioClip(URL)
boolean isInstanceOf(Object, String)
void setRequestFocusController(RequestFocusController)
Open Editor
Compile
Inspect
Remove

This will give you several options on where to run the applet, we will select the applet viewer built into BlueJ.



```
Hello applet!
```

```
Applet started.
```

10.2.4 Understanding what happens in Helloapp.java

Like an application, everything is a class. In the case of an applet we extend the class Applet (which we import (meaning it lives somewhere in the system and we want to get at it without worrying about where)). We do this with the line;

```
import java.applet.*;     // The applet class
```

To do graphics we need access to graphical methods. They live in awt which we also import.

```
import java.awt.*;        // The abstract window class
```

Note there is no main method.

This example introduces the paint() method, which is invoked by the applet viewer or web browser when the applet needs to be drawn. This method will perform graphical output like drawing text or lines or displaying images for the applet.

What happens is that declare a Graphics object called g. We do this in the line;

```
public void paint(Graphics g)
```

We invoke the drawString method which is part of our object g to put the text on the screen.

```
g.drawString("Hello Java!",25,50);
```

We will deal more with applets later in the course.

10.3 Notes about Java

- In Java, all functions (methods) and variables must be defined inside a class definition. There can be no freestanding functions or global variables.
- The overall skeleton of any Java program is a class definition. To make it easier to keep track of the files that make up an application, the name of the controlling class should be the same as the name of the source file that contains it.
- Files containing source code in Java have an extension of java.
- The controlling class definition for an application must contain the main method.
- The file produced by the compiler containing the controlling class has the same name as the controlling class, and has an extension of class.
- The Java compiler produces a separate file for every class definition contained in an application or applet, even if two or more class definitions are contained in the same source file.
- Thus, the compilation of a large application can produce many different class files. A capability known as a jar file can be used to consolidate those class files into a single file for more compact storage, distribution, and transmission.
- Java knows about upper and lower case and treats them differently. “fred” is not the same as “Fred” or “frEd”.
- The controlling class for a Java application must contain a static method named main. When you run the application using the Java interpreter, you specify the name of the class file that you want to run. The interpreter then invokes the main method defined in the class file having that name. This is possible because a class method can be invoked without a requirement to instantiated an object of the class. We do not have a main in an applet
- The main method defined in that class definition controls the flow of the program much as the main function in a C program controls the flow of a stand-alone C program.
- A Java applet is very similar to a Java application except it requires an associated html file in order to be invoked inside a Java-enable browser or applet viewer.