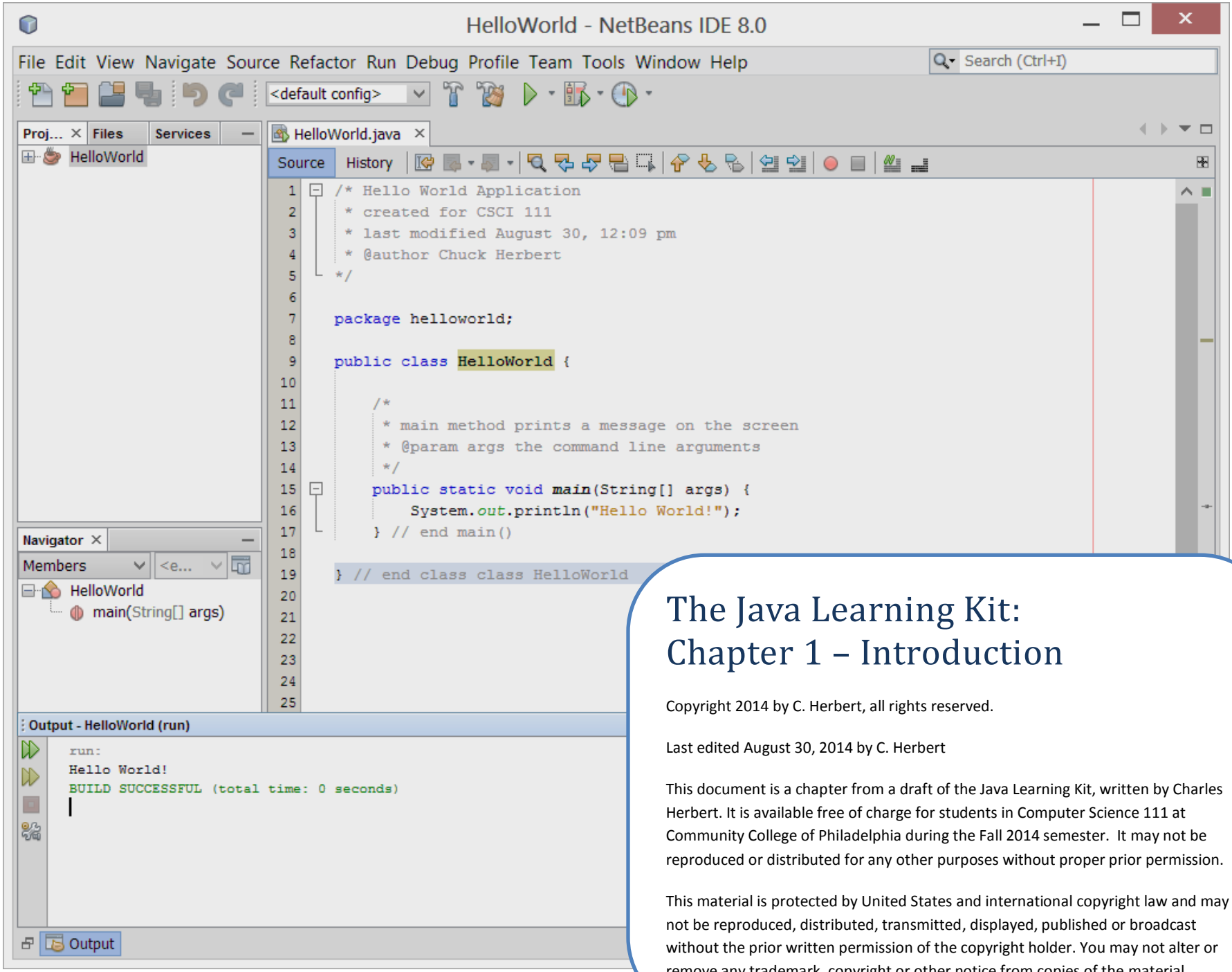


The Java Learning Kit: Chapter 1 – Introduction

Last edited August 30, 2014 by C. Herbert

This document is a chapter from a draft of the Java Learning Kit, written by Charles Herbert. It is available free of charge for students in Computer Science 111 at Community College of Philadelphia during the Fall 2014 semester. It may not be reproduced or distributed for any other purposes without proper prior permission.

The Java Learning Kit: Chapter 1 – Introduction

Chapter 1 is an introduction to computer programming in the Java programming language

Lesson 1.1 – Computing and Computer Science

Lesson 1.2 – Algorithms and Objects

Lesson 1.3 – Programming Languages

Lesson 1.4 – The Java Programming Language

Lab 1 – Getting Started with NetBeans and Java

The labs include instructions for using *Java 8* with the *NetBeans* integrated development environment. Instructions for downloading and installing the NetBeans IDE with Java 8 is provided in the *Java Learning Kit Appendix A*.

Chapter 1 Learning Outcomes

Upon completion of this chapter students should be able to:

- describe the primary components a modern digital electronic computer system;
- describe the field of computer science and common specializations within the field;
- briefly describe the notion of an algorithm and its importance in computer science;
- briefly describe the notion of an object in object-oriented programming;
- describe what compilers and interpreters do, and how they compare to one another;
- describe several major programming languages and the role of each in the history of programming languages;
- Describe how Java code is compiled and executed, and the role of byte code, the Java Virtual Machine and the Java Runtime Environment in that process;
- use the one of several common IDEs to initiate a new Java application project;
- create and run a Java application with console output;
- insert and modify comments in Java code;
- copy a java project, zip the project, and send the project via email as an attachment.

Lesson 1.1

An Overview of Computing and Computer Science

Generally speaking, a computer is any device that can store and process information. Most modern computers include a *processing unit*, often called *a central processing unit* or *CPU*; *internal memory*; *input units*; *output units*; and *external storage devices*, which could be considered a special category of input and output units. The various parts of the computer system are connected by one or more communication channels. Each communication channel is a set of wires called a *bus*. Usually, three separate busses carry data, instructions, and control signals throughout the computer. Figure 1-1 shows a block diagram of such a system.

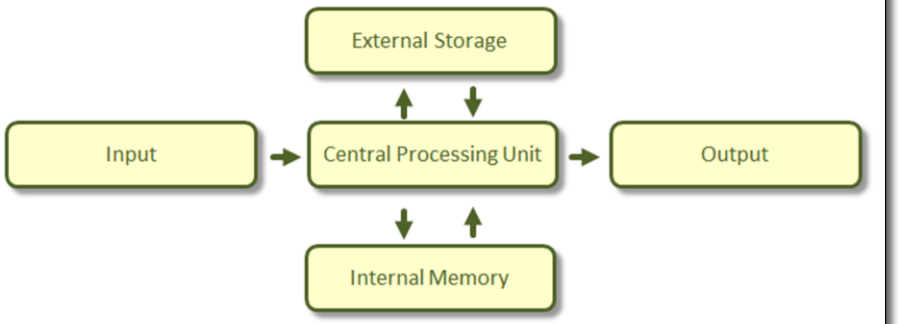


Figure 1. A simple block diagram of the major components in a computer system.

A **Central Processing Unit** can be thought of as the heart or the brain of a computer system. It usually has one or more **Arithmetic Logic Units (ALU)** that process data by performing binary arithmetic, **registers** to hold data temporarily during processing, and a **control unit** to control the flow of information throughout the computer. The control unit orchestrates the timing and coordination of all of the computer system's components, both within and outside of the CPU.

Internal memory in a modern computer is chip-based electromagnetic memory, with no mechanical moving parts. It is usually much smaller and faster than external storage. The internal memory could be on the same chip as the CPU or in separate chips, depending on the sophistication and size of the system.

There are generally two distinct types of internal memory – **Random Access Memory (RAM)** in which randomly selected memory cells can be written to, erased, and reused; and **Read Only Memory (ROM)** which has programs or data permanently burned into a chip that can read but not re-written.

External storage includes larger and slower memory devices, such as hard disk drives. They are usually electromechanical, which means they have moving parts that slow things down, such as spinning disks. Some newer technologies used for external storage, such as Flash ROM, use no moving parts, but are still slower than internal memory because of the technology used and because of the distance to the CPU. Flash ROM is a slower, less expensive non-volatile electromagnetic version of RAM technology with a limited number of access cycles. Commonly used USB flash ROM storage devices are convenient, but should not be relied upon for long-term storage of important data because the chips eventually wear out and because they can be damaged by environmental factors, such as magnetic fields or static electricity.

Input units bring data into the computer from the outside world. **Output units** send data from the computer to the outside world. Together, they are referred to by the term input/output, or **I/O**.

Peripheral equipment, or just **peripherals,** are devices outside of the main part of the computer that communicate with the CPU through I/O controllers. Most people consider external storage devices to be included in the term peripherals, along with I/O and communication equipment, such as keyboards, screens, and network adapters. The field of **human–computer interaction (HCI)** addresses how people interact with computers including

hardware, software, and computing practices. HCI experts help design I/O devices that people will use to communicate with computer systems.

Data — including both data to be processed and instruction sets (programs) for the CPU — are usually moved from slow external storage into fast internal memory when needed, then moved back again for long-term storage. There is an additional hierarchy of memory, Within the computer system itself, from smaller, faster (and more expensive) caches of memory located close to or within the CPU to larger, slower (and less expensive) memory located farther away from the CPU.

The actual organization of a computer varies from system to system, but the general diagram above still captures the essential organization of digital electronic computers. Today, the circuitry for a complete computer system can actually be placed on a single chip.

The organization of computer systems and the transfer and control of data within a computer system are studied in greater detail a Computer Organization or Computer Organization and Architecture course – the equivalent of an anatomy and physiology course for computers. Such courses are often required in Computer Science, Computer Engineering and related degree programs.

What is computer science?

Computer Science is the scientific study of computation, theoretically and in practice, as the basis for modern information systems. **Computation** includes numerical calculation and processes related to calculation.

This may sound a bit boring, but remember, Information isn't just numbers; it can also be text, graphics, sound, video – anything that can have meaning to people. Modern computer systems manipulate, communicate, and store information in its many forms. Modern personal computer systems, tablets, and cell phones are the convergence of many devices dealing with information – typewriters, calculators, telephones, cameras, radios, televisions, sound and video recording equipment, and so on.

Modern computer science is really a convergence of many fields of study dealing with information in its many forms. Yet at heart, computers are always about numerical processing, since, all information in its many forms is represented numerically inside a computer. It's all done with numbers. Computer Science is the study of how it's done. Computer scientists study the theoretical ideas underlying modern computer systems and the practical application of those ideas to make sure that computer systems do what they are supposed to do correctly and in an efficient manner. They also search for new and better ways to build and use computers.

Programming is just one important part of computer science. Today those computer scientists who develop new software and maintain existing software are often known as **software engineers**, who apply time-tested engineering concepts and practices to the modern discipline of computer programming.

What do computer scientists and software engineers actually do?

The layering of complex systems around simpler systems is one of the key aspects of computer science and modern computer systems. Some computer scientists and software engineers work directly with algorithms for the fundamental processes found in the innermost layers of all computer systems, but most work with the outer layers, closer to what people see when they use computers. Let's look at an example of this layering, and then see how it relates to what computer scientists and software engineers actually do, followed by a look at specializations in computer science.

A typical task for which we would use a computer system is word processing – creating, editing and printing documents using a computer system, most often using a keyboard and screen. A word processing program manages all of this so that you don't need to worry about how it's actually done.

The word processing program asks the operating system to tell it what keys the user is pressing, and translates this into instructions telling the

operating system what to store in memory, what to display on the screen, and so on. An **operating system** is like a master program always running on a computer and coordinating what the computer does. It also provides the interface between a user and the system.

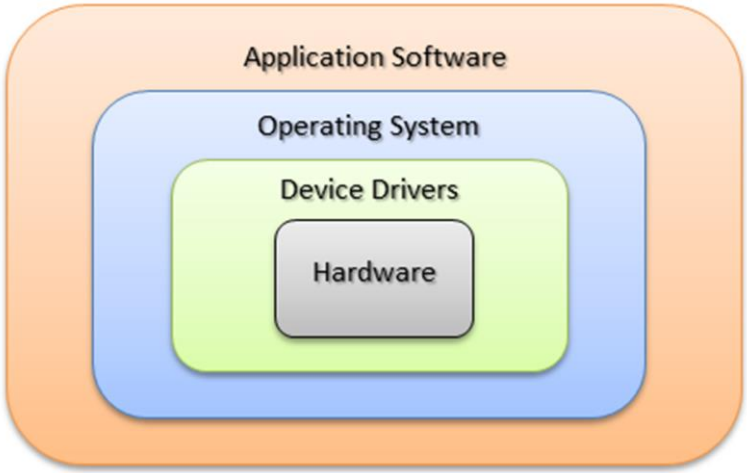


Figure 4. A computer system is composed of layers of specialized software.

The operating system works with short specialized programs called **system utilities** to complete specific tasks. **Device drivers** are a type of utility software that allows an operating system to communicate with the circuitry controlling specific hardware– such as a video control unit, a memory management unit or a disk controller. Together, the application software, the operating system, and the device drivers carry out binary digital arithmetic manipulating billions of bits of data each second to complete all of the tasks that are part of word processing.

Hardware engineers design the hardware, systems programmers develop the operating system and utility software, and applications programmers design the software that most people see and use directly. All contribute to the development of systems for tasks such as word processing.

Those who start out studying computer science may end up as pure computer scientists expanding the theoretical foundations of computing, but most will specialize in other areas, particularly software development, and the majority of software developers are application developers.

Most application developers specialize in areas that overlap with other disciplines. Business programmers, for example, usually need to know something about business, most often accounting, or they need to work with specialists who have such knowledge. Game programmers often work with graphic artists and audio engineers. They sometimes work with physics programmers or use a physics engine written by physics programmers to create realistic movement and interactions between physical objects in games that mimic the real world. The expertise needed to design and create software is as varied as the software itself.

The terms *computer engineering* is related to computer science. **Computer engineering** usually refers to the study and development of computer hardware and involves the study of physics, computer science, and engineering, especially electrical engineering. Hardware engineering and software engineering are distinct fields, but there is a great deal of overlap between the two, so the term computer engineering often includes both. Theoretical computer science provides a foundation for both hardware and software engineering, but is most closely linked to software engineering, especially focusing on the development of efficient algorithms and data structures for information processing. The study of advanced mathematics is important in both software engineering and hardware engineering.

In summary, some computer scientists are theoretical computer scientists – the philosophers of the computer world. Others focus on low-level systems engineering and programming, but most are application developers working on the outer layers of software that carry out common processes in our everyday world – video games, word processing programs, business software, and so on. They often have additional knowledge in other fields or work with people who have such knowledge.

Professional Organizations

The **Computer Science Accreditation Board** (CSAB) is the primary agency for accrediting Computer Science degree programs in the United States. They recognize several areas of specialization in computer science education, listed on the following page. (See their website at http://www.csab.org/)

The CSAB is sponsored by two major computer science professional organizations – the **Association for Computing Machinery (ACM)** and the **Institute of Electrical and Electronic Engineers (IEEE)**. Professional organizations are made up of members with qualifications in a specific area – usually a college degree in the field or professional licensing, such as the American Bar Association for lawyers and the American Medical Association for physicians.

According to their Website, "*The ACM, the world's largest educational and scientific computing society, delivers resources that advance computing as a science and a profession. ACM provides the computing field's premier Digital Library and serves its members and the computing profession with leading-edge publications, conferences, and career resources.*" Their Website has listings for professional jobs in computer science and education. They have a large repository of professional literature about research and development in the field of computing.

The IEEE Website says:" *The IEEE is the world's largest professional association dedicated to advancing technological innovation and excellence for the benefit of humanity. IEEE and its members inspire a global community through IEEE's highly cited publications, conferences, technology standards, and professional and educational activities."* The IEEE focuses on Electrical Engineering and related fields, including data communications and computing. They publish standards for these areas that are often the basis for ISO standards. Their website features job listing for professional engineers.

See http://www.acm.org/ and http://www.ieee.org/ for more information about the ACM and for more information ,about the IEEE Both the ACM and the IEEE have student memberships.

Computer Science Specializations

- **algorithms and data structures**
  specializing in the spatial and temporal efficiency of software and the relationship between data formats and processing to manipulate data
- **artificial intelligence (AI)**
  developing machines with intelligence, including decision making, perception, planning, and machine learning similar to human mental abilities
- **computer architecture**
  the logical design and organization of computers and their components
- **computer graphics**
  creating and improving computer based imaging methods and technology
- **computer networking and data communication**
  improving the efficient transport and security of data between systems
- **database systems**
  mathematically-based efforts to improve the way people and enterprises store and access data
- **distributed computation**
  computing across multiple platforms
- **human-computer interaction (HCI)**
  how people use computer and how computers communicate with people
- **information assurance and security**
  keeping data safe from harm and from loss or change due to technical factors
- **numerical methods**
  the applied mathematics end of computing
- **operating systems**
  focus on software to coordinate and control computer systems, and to provide a bridge between the hardware and the user
- **parallel computation**
  the execution of algorithms using multiple coordinated processors
- **programming methods**
  research into better ways to develop software
- **software engineering**
  an engineering approach to designing and building software
- **theory of computation**
  theoretical study of the limits and nature of computing

The CSAB recognizes each of the above areas as sub-fields within computer science. For career information see: http://www.csab.org/career.html

ACM special interest groups (SIG) bring together computer scientists in these and other specializations, such as bioinformatics, computers and society, or embedded systems. See: http://www.acm.org/sigs

Lesson 1.2 Algorithms and Objects

At the heart of everything computer scientists study, at the core of computing, we find the concept of the algorithm.

Simply put, an **algorithm** is a step-by-step process. Early in the Ninth Century of the Christian Era, a scholar named Muhammad ibn Musa al-Khwarizmi (Arabic: محمد بن موسى الخوارزمي) came to live and work in the City of Baghdad. Al-Khwarizmi[1] means from Khwarezm (Arabic: خوارزم, Russian: Хорезм, Uzbek: Xorazm), in what is now Uzbekistan, which was his ancestral homeland. He was one of many scholars brought to Baghdad by the Caliph to form one of the world's first modern universities. Al-Khwarizmi wrote several important books on mathematics, geography, and astronomy, but perhaps his most important work was his second book, *al-Kitab al-mukhtasar fi hisab al-jabr wa'l-muqabala* (حساب في المختصر الكتاب والمقابلة الجبر) (*A Compendium on Calculation by Completion and Balancing)*, in which he presented a method for solving math problems by balancing equations. His method came to be known as *algebra*. (Al-jabr = algebra.) This was only one part of his systematic approach to solving math and science problems, known, in various languages as the method of al-Khwarizmi, and from which we get the Latinized term *algorithm*[2].

Figure 2. A Soviet stamp from the 1980's marking al-Khwarizmi's 1200th birthday.

According to the French writer and historian Andre Allard[3], almost all Western ideas about mathematics, and hence much of modern science, technology, and engineering, were derived in part from seven critical texts written during the 12th Century and widely circulated through Europe in the centuries that followed. All seven of those books quote extensively from translations of al-Khwarizmi's work. He stands with Euclid and a few others as one of the ancient founders of modern Science, Technology, Engineering and Mathematics – the STEM disciplines.

A **computer program** is a step-by-step set of instructions telling a computer how to perform a specific task. As such, every computer program is an algorithm. Students in a good computer programming course are consciously learning about the logical structure of algorithms and techniques for organizing algorithms which have their roots in the work of al-Khwarizmi more than a thousand years ago.

Of course, data and algorithms have evolved since the days of al-Khwarizmi, especially since the advent of computers in the Twentieth Century. Computer scientists developed the notion of objects and object-oriented programming to help manage the growing complexity of data and algorithms, Anything that can be represented by data in the computer's memory and manipulated by algorithms implemented as computer programs can be organized as an object. Objects can be things in the physical world or even just abstract ideas. An airplane, for example, is a physical object that can be manipulated by a computer. A student's grade point average is an example of an object that is not a physical object.

[1] For more about Al-Khwārizmī and his work see: http://www-groups.dcs.st-andrews.ac.uk/history/Mathematicians/Al-Khwarizmi.html

[2] From Donald Knuth's Website at Stanford University: http://www-cs-faculty.stanford.edu/~uno/graphics.html

[3] Allard, Andre (1992) *Le calcul indien (Algorismus) / Muhammad ibn Musa al-Khwarizmi ; histoire des textes*, Paris: Libraire scientifique et technique Albert Blanchard.

The data that represent an object are organized into a set of properties. Each **property** is a unit of data that describes the object in some way.

The programs that manipulate the properties of an object are known as the object's **methods**. In a modern computer system, an **object** is collection of properties and the methods that are used to manipulate those properties. This modern approach to computer programming is known as **object-oriented programming**, or OOP for short.

Objects with the same methods and properties are **instances** of the same **class** of objects. A class definition is a like blueprint for new instances of the object. Figure 3 shows a Universal Modeling Language (UML) diagram for a BankAccount class of objects. Each instance of the will have the same properties, but have its own values in those properties.

Java is an object-oriented programming language. You will learn much more about objects and object-oriented programming as you study Java.

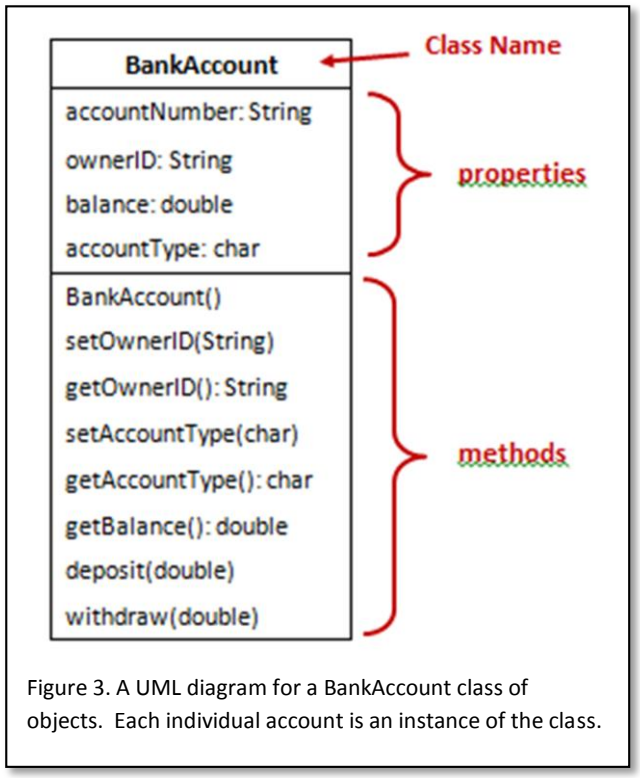


Figure 3. A UML diagram for a BankAccount class of objects. Each individual account is an instance of the class.

Lesson 1.3 Programming Languages

All of a computer system's data and instructions are processed in a CPU's arithmetic logic units that perform binary arithmetic. Usually, the data and instructions come into a processor in in separate streams – a data stream and an instruction stream. Some of the instructions in the instruction stream tell the processor which math operations to perform, while others direct the movement of data. Still others control the logic of programming – the branching and looping present in algorithms.

At the level of the CPU, a computer's instructions are in the form of binary numbers, just like the data itself. Programs written in languages like Java must somehow be converted into these ones and zeroes.

Machine Code

The set of binary numbers that the CPU understands as its instruction set is called the computer's **machine code**. Each CPU, or each family of CPUs, such as the *Intel x86* family, has its own machine code. So, there are as just many different machine codes as there are families of processing units. Eventually, everything that a computer does must be translated into its machine code.

Assembly Language

When a new processor is first developed, it can only be programmed in machine code. Systems programmers use machine code to build an **assembler**, which is a program that translates assembly language into machine code. An **assembly language** is made up of very primitive instructions, just like machine code, but they can be written using numbers in bases other than base two and **mnemonics**, which are short words that sound like what they represent and are easy to remember, such as ADD for addition or SUB for subtraction. They can also use symbolic names instead of numbers to refer to memory locations.

Figure 5 shows the main body of a program written in x86 assembly language, the language for the Intel family of processors used in most personal computers, along with the code's translation into binary x86 machine code and its hexadecimal (base 16) equivalent. Hexadecimal numbers are often used in printouts of machine code to make the code easier read. The translation from binary to hexadecimal is simple, because $16 = 2^4$. Each hexadecimal digit represents four binary digits, as shown in the chart below. The complete assembly language source code file is shown in

Figure 5. Part of an *Intel x86* family assembly language program, with the assembled machine code and its hexadecimal (base 16) equivalent. Programmers often use base 16 to display machine code make the numbers easier to read, and because the translation from base 2 to base16 is quick and easy.

Assembly Language			Machine Code	Hexadecimal Equivalent
	mov	ebx,mval	10001011 01011101 00001000	8B 5D 08
	mov	ecx,arraySize	10001011 01001101 00010000	8B 4D 10
	mov	edi,arrayPtr	10001011 01111101 00001100	8B 7D 0C
L1:	mov	eax,ebx	10001011 11000011	8B C3
	imul	eax, DWORD PTR[edi]	00001111 10101111 00000111	0F AF 07
	mov	DWORD PTR[edi],eax	10001001 00001100	89 07
	add	edi,TYPE DWORD	10000011 11000111 00000100	83 C7 04
	loop	L1	11100010 11110100	E2 F4

Figure 6. Hexadecimal and binary numbering are often covered in courses in *Computer Math and Logic* or *Discrete Mathematics*.

Binary Equivalents of Hexadecimal Digits

1111 = F	**1011 = B**	**0111 = 7**	**0011 = 3**
1110 = E	**1010 = A**	**0110 = 6**	**0010 = 2**
1101 = D	**1001 = 9**	**0101 = 5**	**0001 = 1**
1100 = C	**1000 = 8**	**0100 = 4**	**0000 = 0**

High Level Languages

Creating software can still be difficult and time consuming in assembly language. Eventually, computer scientists and software engineers build translators that can handle **high-level languages**, which are programming languages that are closer to human languages and easier for people to understand. *Java*, *JavaScript*, *Visual BASIC*, *C++*, and *Python* are all examples of modern high-level computer programming languages. By contrast, machine code and assembly language are called **low-level languages**.

Compilers and Interpreters

There are two types of translators that convert high-level languages into machine code: compilers and interpreters. A **compiler** translates an entire program into machine code and stores the result. The programmer ends up with two stored copies of the program. The first, in the original high-level programming language, is called the **source code**. The second stored copy of the program, which is the same program after translation into a particular machine code, is called the **object code**.

Even after translation into machine code, a program may still need to be linked with subroutines from the operating system so that it will run on a particular platform. This step is sometimes called *linking and loading* or *making* an executable program. Sometime linking and loading happens when we try to run object code, and sometimes the compiler makes and stores an executable program as another step in the process of compiling. In either case, with a compiler, there are at least two stored copies of the program, the original source code and the object code, and often a third copy called an executable program.

```
title MultArray Procedure      (AsmMultArray.asm)

.586
.model flat,C

AsmMultArray PROTO,
   srchVal:DWORD, arrayPtr:PTR DWORD, arraySize:DWORD

.code
;-----------------------------------------------

   AsmMultArray PROC USES edi,
   mval:DWORD, arrayPtr:PTR DWORD, arraySize:DWORD

; Multiplies each element of an array by mval.

;-----------------------------------------------

      mov   ebx,mval          ; multiplier
      mov   ecx,arraySize        ; number of items
      mov   edi,arrayPtr      ; pointer to array

L1:   mov   eax,ebx        ; get multiplier
      imul  eax, DWORD PTR[edi] ; multiply by array val
      mov   DWORD PTR[edi],eax  ; store in the array
      add   edi,TYPE DWORD
      loop  L1

      ret

AsmMultArray ENDP

END
```

Figure 6. An *x86* assembly language source code file. The *.586* directive after the title indicates the code was written for an older Pentium 32-bit processor. eax, ebx and edi are the names of processor registers. This code performs scalar multiplication – multiplying each item in an array by the same value.

An interpreter is much simpler than a compiler. An **interpreter** translates a program one instruction at a time as the program runs. It feeds each translated instruction to the CPU to be executed before translating the next instruction. The only stored copy of the program is the original source code. Often scripting languages, such as *JavaScript* or *Visual BASIC for Applications (VBA)*, work this way. **Scripting languages** are simplified high-level languages that allow someone to program in a particular environment. JavaScript can be added to the HTML codes for Web pages to provide them with some primitive data processing capability. VBA allows someone to program features in Microsoft Office products, such as Word or Excel.

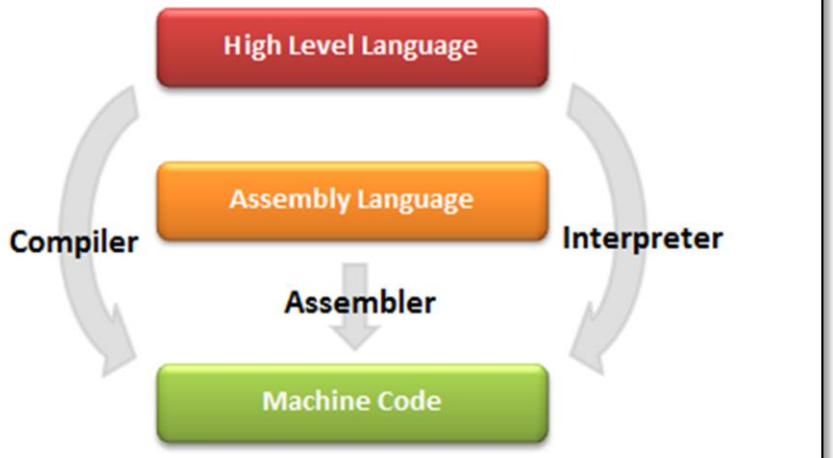


Figure 7. All programs must be translated into machine code. Compilers, interpreters and assemblers perform this translation. Java adds an intermediate step for portability.

Interpreters have also been used for teaching languages. Serious computer programming languages such as *Java* and *C#* that are used by professional software developers are sometimes referred to as **production languages**. **Teaching languages** are languages that are not generally used in production environments, but are instead used to teach someone the logic of computer programming or the processes used in creating computer software before attempting to teach them to use production languages. The *Pascal* programming language was an example of a teaching language.

The Java Advantage

What about Java? Java is both compiled and interpreted. Portability was a design goal for the Java language, as was high performance. These goals are sometimes in conflict because of the differences from one platform to another. As a solution, The Java developers came up with the **Java Virtual Machine (JVM),** which is basically an interpreter that can handle a simplified intermediate language called **byte code.** In a two-step process, a Java compiler translates the Java language into byte code. This can be stored and moved from one platform to another. Each machine that needs to run Java is equipped with a **Java Runtime Environment** that has everything needed to run byte code on that machine, including a JVM that translates and runs the byte code as machine code for the host machine.

Both Apple and Microsoft were highly critical of this approach, but it proved successful, largely due to the Internet. Today, almost all Web browsers are Java-enabled – meaning that they can run Java byte code – and most personal computers, tablets and smart phones are enabled with a Java Runtime Environment or its equivalent, which is free for all systems[4]. According to Oracle, there were over 1.1 billion PCs and over 3 billion smartphones running Java at the end of 2013[5]. There are now more Java-enabled devices in the world than there are people. This portability for a powerful object-oriented language is what IBM calls "*The Java advantage".*

[4] The JRE download page is online at: http://www.oracle.com/technetwork/Java/Javase/downloads/Java-se-jre-7-download-432155.html

[5] See Oracle's *Lean About Java Technology* page online at: http://www.Java.com/en/about/ According to the site, "*31 times more Java phones ship every year than Apple and Android combined*."

Let's take a look at several other important and popular high-level languages that were developed over the years.

FORTRAN

FORTRAN, one of the first high level programming languages, was released in 1957 by IBM for use on their popular 701 computer. It was proposed by John Backus, a mathematician who specialized in writing programs to perform complex calculations. The name FORTRAN comes from the two words "formula translator", and as the name implies, it was designed for use by scientists and engineers with special instructions for math functions.

Versions of FORTRAN were soon created for IBM's other computers, including the 1400 series, and then for most of the more popular computers from other vendors. This made FORTRAN the first portable programming language – one that is widely used across different computing platforms. The FORTRAN code had to be compiled for each specific platform, but the source code could be used on any computer with a FORTRAN compiler. IBM developed several versions, culminating in FORTRAN IV in 1962. The American National Standards Institute (ANSI) issued a definition for a standard FORTRAN language, called *ANSI FORTRAN 66,* in 1966 followed by the *ANSI FORTRAN 77* standard a decade later. In 1991, the ISO released an international standard for Fortran, with a parallel ANSI standard, which broadened the language and changed the name from all caps to simply *Fortran*. That version is often called *Fortran 90*.

Before FORTRAN almost all software was created using assembly language and machine code. Once FORTRAN appeared, people began to use it for much more than science and engineering. The increasing use of FORTRAN to process commercial business data led to problems for financial accountants and auditors. A bank auditor, for example, needs to be able to "follow the money" from one account to another. This was nearly impossible with FORTRAN, unless the auditor was also a trained computer programmer. In addition, even though FORTRAN contained functions for higher mathematics, it was not well suited for programming business and financial transactions.

A list of free FORTRAN tutorials is available online at: http://www.fortran.com/fortran/market.html . This webpage is maintained by the *Fortran Company*, which provides Fortran products and professional consulting. Today Fortran is a rarely used programming language, but some applications based on old legacy code still use it.

Algol

Soon after the development of FORTRAN, John Backus was also instrumental in the development of **Algol**, which was a language designed by an international group of mathematicians and programmers for use in describing and discussing algorithms. Some of the more important features of modern programming languages that first appeared in Algol include:

- begin-end pairs to mark blocks of code and limit the scope of variables
- recursion
- a logical IF…THEN…ELSE… structure
- two different ways to pass parameters
- Implementation of data abstraction by allowing programmers to implement their own data types.

Algol was never commercially successful as a production language, nor was it intended to be. Burroughs was the only large computer company that provided an Algol compiler, and many of its features were additions to the Algol standard. Algol did not have the built-in specific tools for scientific or business programming like FORTRAN or COBOL, and it lacked a standard definition of input and output routines for file handling, screen displays, and printing. Instead, it was the language used internationally by research computer scientists to publish algorithms, and created a foundation for more advanced programming languages, including most modern

Who develops and regulates technology standards?

Technology standards are important in the modern world so that things like computers, (or telephones, or even plumbing systems) are compatible with one another. They are also important to maintain public health and safety. Who is responsible for technology standards? A combination of governmental and private agencies.

- **Government agencies** are part of the government or are backed by the government and have the force of law behind them. Their standards are often called *regulations* rather than standards. Sometimes they adopt standards from other agencies and give them the force of law. For example, The **Federal Communications Commission (FCC)** is a federal government agency in the U.S. responsible for regulating electronic communications. They can regulate the nature and amount of electromagnetic interference a computer generates, so that it won't interfere with cell phones. The *Occupational Health and Safety Administration* (OSHA) can regulate how computers and related equipment affect workers in business and industry. There are many such agencies.
- **Standards agencies** are often independent organizations, but sometime they are supported by the government or are part of the government, depending on the country. The **International Organization for Standardization, also named ISO,** is an umbrella group responsible for international standards in a wide variety of areas, especially with regard to technology. (The name ISO comes from the Greek for *equa*l, and is not an acronym*)* It is composed of national standards agencies from more than 160 countries around the world. The **American National Standards Institute (ANSI)** is the national standards agency in the United States. It is a private, non-profit voluntary organization composed of committees of experts in different fields. Some national standards agencies are government agencies, such as the Standardization Administration of the People's Republic of China (SAC), the Chinese equivalent of ANSI. The ISO has released standards for many programming languages.

  For more information on ISO and its members, see: http://www.iso.org/iso/home/about/iso_members.htm
- **Professional societies** such as the IEEE and the ACM mentioned earlier in this chapter often release standards and guidelines. They are made up of professionals in the field. The IEEE has played an important role in developing standards for electronics and computing, such as the IEEE standards for floating point numbers.
- **Trade associations** are made up of companies who share a common interest. The **Electronics Industries Alliance (EIA)**, formerly known as the *Electronics Industries Association,* is an organization of computer and electronic equipment manufacturers. They often adopt standards so that their equipment and practices are compatible with one another. The **Computer Technology Industry Association (CompTIA)** publishes standards for certifying computer professionals, especially in networking and security.
- Sometimes **a group of industry experts** from industry, academia, and the government get together to address areas of common interest. This happened in Philadelphia in 1960 when CODASYL, the Conference on Data Systems Languages, was formed to discuss "*the feasibility of establishing a common language for programming of computers in business data processing applications*." * Their work gave rise to the COBOL language.
- Proprietary languages are owned by **corporations**, who control their standards. Technically, Java belonged to Sun Microsystems, but was designated as Free Open Source Software, and the current mechanism for proposing changes to the java standard is the Java Community Process (see http://www.jcp.org). The *.NET* langauges – (*C#.NET*, *VB.NET*, etc.) are owned by Microsoft.

---

* *The History of Cobol,* edited by William M. Klein, Oct 4, 2010 available online at: http://home.comcast.net/~wmklein/DOX/History.pdf

programming languages. *C*, *C++*, Java, *Python* and most other languages used widely today can trace their origins back to ideas implemented in Algol. In fact, even early languages, such as *COBOL* and *BASIC*, adopted features from Algol. Several early versions of Algol were defined, including *Algol 58* (1958), *Algol 60* (1960).

COBOL

The solution to the problem of using FORTRAN in the business world was solved with the introduction of the **COBOL** language in 1960. Like the name FORTRAN, COBOL is an acronym that comes from the words *COmmon Business-Oriented Language*. COBOL was developed by a team of people working for the United States Navy under the direction of Grace Hopper, who went on to become an admiral (O-7) before she retired after a 40-year naval career. In April of 1959, The *Conference on Data Systems Languages* (CODASYL) held their first meeting at the University of Pennsylvania in Philadelphia. The conference, made up of representatives of major computer companies and several government agencies, was formed by the Defense Department to develop a standardized business programming language that could be used across all computing platforms. The first COBOL compilers were released in 1960. The first ANSI version of COBOL was *COBOL 68*, followed by *COBOL 74* and *COBOL 85*, which was adopted as an ISO standard.

COBOL has functions and instructions that are more suited to commercial data processing than FORTRAN, and is a wordier language, which makes it easier for financial auditors to understand without extensive training. It was better suited for business recordkeeping and accounting than FORTRAN or almost any other language, which is what the language was designed to do.

It was estimated that by the year 2000 there were more lines of code written in COBOL than in any other computer programming language, and that more than 80 percent of the world's business software was written in COBOL. The use of COBOL declined at the end of the 20th Century, coinciding with the rise of electronic spreadsheets, specialized accounting software, and relational database management systems. In 1990, the US Department of Labor estimated there were 500,000 professional COBOL programmers in the US. By 2010, that number had dropped to less than 75,000. Today, only a small percentage of new programmers are learning COBOL.

Functional Programming: LISP and Lambda Calculus

Another programming language that appeared around 1960 was LISP, developed at MIT by John McCarthy with some help from Marvin Minsky. The name LISP comes from "List Processing". **LISP** is a functional language based on Lambda Calculus. As the name implies, the processing of lists, especially linked lists, is an important feature of the language.

Princeton mathematician Alonzo Church developed **Lambda Calculus** in the 1930s as a formal system of logic for discussing mathematical functions. It is important in computer science, especially as the basis for functional programming languages. Lambda Calculus is all about substituting one equivalent item for another. In standard algebra, for example, if we discover that $Y = X+3$, then we could substitute $X+3$ in place of Y to help solve for X in $2X+Y=12$. It becomes $2X+(X+3) = 12$ which reduce to $3X+3=12$, then $3X=9$, then $X=3$. Earlier in this chapter we discussed the Ninth Century work of al-Khwarizmi, from whom algorithms derive their name. He was one of the first to formalize a language for this kind of substitution to solve calculation problems. Professor Church developed lambda calculus as a theoretical model for computation, originally used to explore which functions can and cannot be computed using algebra or any other system of calculation. It has become a cornerstone of theoretical computer science.

LISP has been an important language, associated with research into artificial intelligence and functional programming, but otherwise rarely used as a production language for business, scientific, or systems programming. Today the two most common versions of LISP are *Scheme* and *Common*

LISP. Scheme has been used in introductory computer science courses, but its use has declined in recent years. *Haskell*, *ML,* and *CAML* are other functional programming languages sometimes used in upper-level computer science courses and computer science research.

BASIC

COBOL, like FORTRAN, takes a while to master. For College students, this often meant that a semester or more had to be spent learning programming before anything useful could be done with a computer. At the same time, computers were becoming smaller, less-expensive, and more accessible to the public. Personal computers were still some years away, but by the mid-1960's many colleges and universities had computers that students could use on campus. In 1965, in order to make programming accessible to students on the new "mini-computers" that had begun to appear, two professors at Dartmouth College in Hanover, New Hampshire, John Kemeny and Thomas Kurtz, invented the BASIC programming language. **BASIC (Beginner's All-Purpose Symbolic Instruction Code** was an interpreter-based language rather than a compiled language like FORTRAN or COBOL. It was designed to be easy to learn and easy to use.

BASIC caught on quickly, and when personal computers began to appear in the 1970's almost every machine was equipped with a BASIC interpreter, so, at the time, more people learned BASIC than any other language.

Over the years there have been many versions of BASIC, including GW-BASIC, Quick BASIC, True BASIC, and Visual Basic, just to name a few. Today the most common version of Basic is Microsoft's *Visual Basic.NET (VB.NET)*, which appears to be declining in popularity and professional use. The newest version of Visual Basic is part of Microsoft's *Visual Studio 2013*.

Pascal

BASIC and FORTRAN both had a GOTO command, which is sometimes referred to as an *unconditional branching* command. Each line in a BASIC or FORTRAN program was numbered, and at any point in the program the GOTO instruction could suddenly re-direct the flow of control to a line number in another part of the program. The command was intended to let users set up branching and looping, but it was so flexible to use that programmers often ended up with poorly designed logic that jumped repeatedly back and forth throughout the code, leading to what was referred to as *spaghetti code*. People other than the original programmer often had to spend hours trying to figure out how such a program worked.

In response to this problem, Swiss computer scientist Niklaus Wirth released the **Pascal** programming language in 1970. He named the language after the 17th Century French Mathematician and Philosopher, Blaise Pascal, one of the first people to ever build a working mechanical calculator.

Wirth's Pascal was widely used as a teaching language and for prototyping - designing algorithms in one language for implementation in another.

in the late 1970s and 1980s. It was based on concepts from Algol, but had built-in commands for looping and branching that forced the user to write programs according to good principles of structured logical design. In Pascal, it became natural for programmers to code with a logical flow of instructions and almost impossible for them to end up with spaghetti code as they did in BASIC. Pascal was a simple interpreter-based language that was easy to learn and use. Pascal code could also be compiled for production of commercial software. It did see some professional use, particularly by Apple from 1978 into the mid 1980's, but was never on a par with FORTRAN, COBOL, or newly emerging object-oriented languages.

Niklaus Wirth went on to develop **Modula** and **Modula 2**, languages designed around the concept of modular programming, and **Oberon**, a true object-oriented language.

The C Programming Language

The **C programming language** was developed at Bell Labs by Dennis Ritchie for use as a systems programming language. Although it is a high-level language that incorporates many of the Algol features, it is still close to assembly language, which means it is close to machine code. This means that it has a fairly simple compiler, and uses a small amount of memory compared to more complex languages. This provides C with two advantages: first, it is relatively easy for a team of good assembly language programmers to build a C compiler quickly, and second, the language can run in a small amount of memory, such as on small embedded processors.

C was used to write the original *Unix* operating system, and has been closely linked with Unix and Linux systems. It is still used for quickly creating device drivers and other system software for new computers and related devices.

C is also an important language on which many more advanced languages are based, such as C++, Java, and Python. There is a direct connection from Algol through C to many modern languages.

In 1978, *The C Programming Language,* a book by Brian Kernighan and Dennis Ritchie, appeared on the market.[6] It was the definitive book on C programing with a very simple straightforward style teaching experienced programmers how to use the language. It became one of the most widely read and influential programming books of all time. Many C compilers were written specifications based on *K&R*, as the book was called.

In 1989 ANSI adopted a standard for the C programming language, which became the ISO standard the following year. *ANSI C*, *ISO C*, *C 89* and *C 90* are all fundamentally the same language. ISO revisions of the C standard appeared in 1994, 1999 and 2011.

Simula and Smalltalk

In the 1960s two Norwegian computer scientists, Kristen Nygaard and Ole-Johan Dahl, developed a language for programming discrete event simulations, which they called **Simula**. A discrete event simulation simulates a system in which distinct events changes the state of a system. It differs from a continuous real-time simulation in which variables are constantly changing. For example, a game of American football could be viewed as a discrete event simulation if we only see the results of each play and not the plays themselves. The state of the system could include which team has the ball, on what yard line, what down it is, how far to go for a first down, and the score. Each play could be considered a discrete event that changes the state of the system. The defense picks a formation, the offence picks a play, and the simulation shows the result. Many games and real world processes can be viewed in this way. Most discrete event simulations in business and industry are much more complicated, such as modeling a manufacturing process or a transportation system.

Nygaard and Dahl needed a programming language to simulate discrete systems and interactions between multiple systems. They came up with an Algol-based language that had objects with properties, along with distinct methods to manipulate those properties – the basis for modern object oriented programming. Objects communicated with each other by passing messages back and forth to each other. The language was defined by the *International Federation for Information Processing* in 1967 as *Simula 67*.

[6] Kernighan, Brian W.; Ritchie, Dennis M. (February 1978). *The C Programming Language (1st edition.)*. Englewood Cliffs, NJ: Prentice Hall. ISBN 0-13-110163-3.

Simula was a language for a very specific purpose, but the ideas in Simula inspired others. At the *Xerox Palo Alto Research Center* in California, computer scientists led by Alan Kay developed the *Smalltalk* programing language, which was highly influenced by the innovations in Simula. **Smalltalk**, released in 1980, is a true object-oriented programming language in which every bit of data and every method must be part of an object. Objects communicate with each other only through message passing between objects. The properties of each object are fully encapsulated, meaning that no object can see the properties of another object except through messages from that object.

After several years of research, Smalltalk-80 was released in 1980. *ANSI Smalltalk* (1998), *Apple Smalltalk*, *Squeak*, and *VisualWorks* are just a few of the many versions of Smalltalk. Smalltalk is rarely used for modern production programming compared to languages like Java, and C++.

C++

The **C++** programming language is a revision of C influenced by Simula and Samlltalk. (The name C++, comes from the fact that the ++ operator indicates iteration in the C language.) C++ is an object-oriented language in the sense that it can be used to implement objects and message passing, but it does not require that programmers use objects and message passing. In fact, the C language is a subset of C++ and many C programs will work with a C++ compiler with little or no change.

C++ was developed at Bell labs by Dr. Bjarne Stroustrup, a Danish-born computer scientist with a PhD. in Computer Science from Cambridge University. He merged the object features from Smalltalk into the C programming language to create a language that was practical and versatile like C, but that could support object-oriented development. Dr. Stroustrup viewed C++ as a good tool for developing large software systems. *C++* was released in 1983. ISO standards for C++ were released in 1998 and revised in 2003, 2007 and 2011.

Today C++ is among the most widely used programming languages, and has spawned several spinoffs, such as *C#,* and *Objective C*. Many game development systems use a programming language based on C++. Much of the Java language was based on C++.

Perl

Perl was developed by Larry Wall in 1987 as a language for extracting data and creating reports on Unix computers. The language was well suited to preforming similar tasks for Web and Network programming and increased in popularity with the rise of the Internet. Perl is also well-suited for tying together applications and for writing very short (often online) scripts for Unix and Linux systems administrators. Perl scripts are sometimes used by systems administrators when installing software.

Wall originally named the language *Pearl* after “the pearl of great price”, mentioned in the Bible’s book of Mathew. He changed it to *Perl* when he found that an older obscure language named Pearl existed. Today some people say the Perl stands for *Program Extraction and Reporting Language*.

Perl 5 has been around since 1994. The most recent version, released in May 2014, is Perl 5.20. Perl 6 has been developed as a different language related to Perl 5. Both are freely available software.

For more about Perl 5, see: http://www.perl.org

For more about Perl 6, see: http://perl6.org

C# *(C Sharp)*

C# was developed as a proprietary part of Microsoft’s *.NET Framework*. Anders Hejlsberg was the lead developer. The C# programming language was released by Microsoft following their failed attempt to implement the *J++* language, a version of the Java programming language.

From its inception, Java was intended to be a freely available language for use on any computer system. Sun Microsystems (and later Oracle) allowed others companies to develop their own Java compilers and software development systems, provided they agreed to make their versions of Java conform to Sun's standards for Java compatibility. Microsoft developed a version of Java under such a license, but began to add proprietary features to their *"J++"* language, which Sun believed to be in violation of the agreement. Following a lawsuit filed against Microsoft by Sun, *J++* development was halted. The last available version was discontinued in 2001.

Microsoft subsequently released *J#*, which was also discontinued and does not work with any version of the.NET framework beyond *.NET Studio 2005*.

The Sun-Microsoft lawsuit resulted in the release of internal Microsoft documents revealing what they referred to as an "*embrace-extend-exterminate"* corporate strategy. Microsoft planned to adopt industry standards, co-opt the standards by extending them to include Microsoft-only features linked to their operating systems, then attempt to drive their competitors from the market by not letting other companies use the new features[7].

C# was released by Microsoft during their legal battle with Sun. Despite its name, many people believe it is closer to Java than to C++, although over the years the two languages have developed in different directions. In the article "*Why Microsoft's C# Isn't*" in a January 2002 edition of *CNET News*, James Gosling was asked about his reaction to C#. He replied:

> "*The trite answer is, 'Imitation is the sincerest form of flattery…But the other answer is, 'You guys (at Microsoft) still don't get it, because it's sort of Java with reliability, productivity and security deleted… They had this problem in their design rules that they had to support C and C++, which means you have to have a memory model where you can access everything at all times. It's the existence of those loopholes that is the source of security, reliability and productivity problems for developers."* [8]

C# has replaced VB.NET as the most popular programming language for the.NET environment. Microsoft C# is a proprietary closed source language, meaning the source code is not available. The ISO released a C# standard in 2003, and a revision to the standard in 2006. To date, a few open source C# compilers exist, but none are full implementations of C#. Microsoft has extended its version of C# to include feature not in the ISO standard.

Objective C and Swift

Objective C was originally developed by Brad Cox and Tom Love at Productivity Products International (later renamed StepStone) in the early 1980's– roughly the same time the C++ language was released. Like C++, Objective C was intended to add Smalltalk-like objects to C programming.

The language was not widely used until it was adopted by Apple as the language of choice for iPhone and iPad programming. The NeXT computer company (founded by Steve Jobs) acquired the rights to Objective C from StepStone in 1995. In 1996 Apple bought the Next company and Jobs returned to Apple. The Objective C language became available for the Mac OS X operating system and later became the preferred language for the iOS operating system used on iPhones and iPads.

[7] See the ZDNets Newsletter, Nov 9, 1998, available online at: http://www.zdnet.com/news/intel-exec-ms-wanted-to-extend-embrace-and-extinguish-competition/100925

[8] available on the Web at: http://news.cnet.com/2100-1082-817522.html

C is a subset of Objective C, just as it is with C++. The object-oriented nature of the language is actually closer to the original Smalltalk than to C++. Objective C syntax is different from the C++/Java syntax. For example, methods are not *called* from other methods in Objective C, they are invoked by message passing. There are many other differences between the two languages and how they are used, but they are best understood by experienced programmers. Basic logic and control – branching, looping, and so on – is similar to the Algol-derived syntax in C, C++, and Java.

The primary difference between Objective C and Java is in their target systems: Java is a cross platform language for programming devices on large heterogeneous networks, while Objective C is used almost exclusively for programming OS X and iOS devices, especially iPhones and iPads.

At Apple's 2014 Word Wide Developer's Conference, the company introduced a new language called *Swift*, and announced that Objective C would be replaced by Swift. Apple Vice President Craig Federighi referred to Swift as "*Objective C without the C*".[9]

Python

The **Python** programming language was developed by Guido von Rossum, a Dutch-born computer scientist who now lives in the United States. He began developing Python as a scripting language to replace *ABC,* an earlier teaching and prototyping language he had helped develop. Von Rossum said that he choose the name Python because he is a big fan of *Monty Python's Flying Circus* and he was in an irreverent mood at the time.

Von Rossum succinctly described Python in an article titled: *"What is Python? Executive Summary"* on the Web at https://www.python.org/doc/essays/blurb :

> *"Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed."*

Von Rossum designed a language that was easy to read, east to write and easy to debug, with exception handling built into the Python interpreter. Exceptions are objects that are created detailing what happens when a computer program doesn't run properly. Exception handling is a major topic in Java programming which you will learn more about later.

Python programs are easy to create and run, but Python programs are slower than programs written in major production languages like Java because they are interpreted. In a 1997 comparison of Java and Python, Von Rossum wrote that

> *"Python programs are generally expected to run slower than Java programs, but they also take much less time to develop. Python programs are typically 3-5 times shorter than equivalent Java programs."* [10]

[9] See *Venture Beat News*, June 5, 2014. Online at: http://venturebeat.com/2014/06/02/apple-introduces-a-new-programming-language-swift-objective-c-without-the-c/

[10] *Comparing Python to Other Languages,* a blog entry available on the Web at: https://www.python.org/doc/essays/comparisons

Remember, interpreted programs must be translated line by line each time the program runs, making them slower than compiled programs whose already-translated object code or executable version of a program is ready to go when the program runs. Java adds the JVM step for portability, but it is still significantly faster than interpreted software.

Von Rossum suggests that java and Python complement each other well, with Python suited to prototyping software designs that can be more fully developed as components of large systems using Java. He also suggests that Python is well-suited for a scripting language used for quick, short applications in place of other scripting languages like JavaScript, and as a "glue language" to create short pieces of software that "tie together" components of large systems.

Python is free open source software. For more about Python see the Python Website at: Python.org .

JavaScript

JavaScript was developed by Netscape and announced by NetScape and Sun Microsystems in 1995 as an:

> *"open standard object scripting language… JavaScript is an easy-to-use object scripting language designed for creating live online applications … JavaScript is designed for use by HTML page authors …it can be used by people with little or no programming experience to quickly construct applications."* [11]

JavaScript is implemented as an interpreted language built into Web browsers. Short quick programs add interactivity and other feature to Web pages through HTML code that includes JavaScript. All version of Microsoft Internet Explorer since 1996 include support for JavaScript, as do the Chrome, Firefox, Safari, and Opera browsers.

Bredan Eich, one of the cofounders of the Mozilla project, developed the original version of JavaScript for use in the Netscape Navigator Web browser – viewed by many as the "grandfather" of most modern Web browsers.

ECMA International (formerly the European Computer Manufacturers Association) developed a standard for Web scripting languages based on JavaScript. Today JavaScript, JScript, ActionScript and several other Web and network scripting languages are all considered implementations of the ECMAScript Standard, which was adopt by the ISO in 1997 and updated in 1999, 2009 and 2011.

JavaScript and other ECMAScript languages have a structured syntax similar to C (which is similar to Java and can be traced back to Algol) for branching, looping and simple arithmetic. JavaScript can be called object-based, but it does not require or support true object-oriented programming like Java. In practice, JavaScript software is usually limited to simple programs that add features such as interactivity and the ability to perform calculations in HTML web pages. As its name indicates, it can be thought of as Java-light, although a number of its feature differ from those in Java.

To learn more about JavaScript, see the W3 Schools Java Tutorial, on the Web at: http://www.w3schools.com/js

(w3schools.com has introductory tutorials for many other scripting languages used for Web development.)

[11] The original 1995 press release from NetScape and Sun is available on the Web at https://web.archive.org/web/20070916144913/http://wp.netscape.com/newsref/pr/newsrelease67.html

The Website www.javascriptsource.com has an extensive "cut and paste" repository of freely available JavaScript programs that can be used in Web page development.

PHP

PHP is a freely available server-side scripting language primarily used for applications involving HTML Web pages. The language was originally developed as *Personal Home Page Tools (PHP Tools) and Personal Home Page Forms Interpreter (PHP FI)* by Rasmus Lerdorf, a Canadian Computer Software engineer from Greenland who had contributed to the development of the Apache server software. He has said he created the PHP interpreter for his own use to process HTML Web forms and communicate with databases. The source code for the language was freely released to the public in 1995[12].

in 1997, two students from Tel Aviv University, Andi Gutmans and Zeev Suraski, began working with Lerdorf to turn PHP into a more useful programming language. PHP 3.0, the first version of PHP similar to the current language, was released in 1998 and gained some popularity as a scripting language for Web development as the Internet began to grow.

PHP is now maintained by The PHP Group as a general-purpose scripting language that is still used primarily for Web development. According to the group, the name PHP now stands for "PHP Hypertext Preprocessor". The most recent version of PHP is PHP 5.

There are some issues that have prevented PHP from being more widely used as a general purpose scripting language. There is no ISO standard for PHP and the licensing agreement, though free and open-source, has some restrictions on the use of the name PHP or names derived from PHP[13]. PHP does not fully support ISO's Unicode standard for the encoding of character data, as most languages do.

In 2013 Google reported that its Safe Browsing Service had found malware in versions of the PHP source code being distributed to the public. The PHP group locked down its site, moved to new servers, and applied for a new security certificate.

Despite these issues, several million Web developers continue to use PHP as a scripting language for Web development.

Ruby

Ruby was developed by Yukihiro Matsumoto in Japan as a true object-oriented scripting language with functional programming capabilities. He said the name came from a colleague's birthstone[14]. The first version of Ruby 1.0 was released to the public in 1996. It became very popular for Web scripting in Japan, but was rarely used elsewhere before 2000, when the first English language Ruby documentation became available.

The language declined a bit in popularity in 2003 when Ruby 1.8 was released. Ruby 1.8 was incompatible with older versions of the language, and a lot of Ruby applications had to be rewritten. Interest in Ruby began to increase significantly in 2005 when David Heinemeier Hansson, a Danish professional programmer and race car driver, used Ruby to develop the *Rails* framework for Web scripting that was especially well suited to mapping data from relational database tables to HTML Web pages. Apple included it with its OS X operating system in 2007. Since then, the language and the *Ruby on Rails* framework have steadily been increasing in popularity internationally.

[12] history refereence

[13] PHP license refeerence

[14] *An Interview with the Creator of Ruby, from the LInuxDevCenter Website, online at: http://www.linuxdevcenter.com/pub/a/linux/2001/11/29/ruby.html*

The ISO issued a standard for the Ruby language in 2012. *Ruby* and *Ruby on Rails* are both freely available. The most recent version of Ruby, Ruby 2.1 was released by The Ruby Community late in 2013.

There are several versions of Ruby is use today, such as *JRuby* which utilizes the Java Virtual Machine for cross-platform programming, *MacRuby* for Apple's OS X operating system, *IronRuby* for the Microsoft .NET environment, and *mruby* – a light version of Ruby for Web applications.

For more about Ruby see the official Ruby Website at:
https://www.ruby-lang.org

Lesson 1.4 The Java Programming Language

The **Java** programming language was developed by a team at *Sun Microsystems* led by chief developer James Gosling. (Sun was acquired by *Oracle Corporation* in 2010.) Sun started the development of Java in the early 1990s as a language for multimedia programming, but as the language developed, the internet blossomed – in particular the World Wide Web portion of the Internet. Quoting from Gosling's early description of Java[15]:

> "*The design requirements of the Java programming language are driven by the nature of the computing environments in which software must be deployed.*
>
> *…To live in the world of electronic commerce and distribution, Java technology must enable the development of secure, high performance, and highly robust applications on multiple platforms in heterogeneous, distributed networks.*"

A **heterogeneous network** is a network that connects computers with different hardware and different operating systems. A **distributed network** is a network of computer equipment that connects many machines with CPU's of their own, rather than connecting many devices to one CPU. The Internet is a heterogeneous, distributed network; in fact, it the most heterogeneous and most distributed network of all time. As the Internet grew, so did the use of Java, which was perfectly suited for Web-based programming.

Sun was interested in developing a language in which applications could be created and distributed across many different computers, not just for one computer or one family of computers. Java's built-in networking and security, distributed approach to computing, and multimedia capabilities fit in very well with the rise of the World Wide Web, and Java quickly became the language of choice for the development and delivery of software using the Internet. Today, more devices connected to the internet use Java than all other programming languages combined. There are more Java Runtime Environments installed on computers, tablets and smartphones – and embedded in kitchen appliances, automobiles, industrial equipment and other systems – than all Android, Windows and iOS systems combined[16].

The syntax of the Java language was based on C++, the most popular programming language at the time Java was introduced, so it would be familiar to programmers. Therefore, it can be traced back to Algol through C++ and C. Unlike C++, it requires all programs to be methods in objects, so it is a true object-oriented language. It has features limiting the use of low level system calls that can introduce security hazards in languages such as C and C++.

[15] *Design Goals of the Java TM Programming Language* from *The Java Language Environment* by James Gosling and henry McGilton, May, 1996 Available online at: *http://www.oracle.com/technetwork/Java/intro-141325.html*

[16] *Learn About Java Technology,* an oracle publication, online at http://www.Java.com/en/about

We will learn more about the nature and history of Java while we learn to use Java as a tool for the development of software applications. Java is a very sophisticated software development tool, which may seem like overkill for the introductory topics a new programmer needs to learn, but Java is often chosen for programming courses because of its market dominance, its usefulness for large software development projects, and the fact that the language and techniques of object-oriented programming in Java are easily transferable to many other languages.

Lab 1 – Getting Started with NetBeans and Java

In this step-by-step exercise we will learn to create a simple Java program using the *NetBeans IDE*. This lesson is similar to the *NetBeans IDE Java QuickStart Tutorial* which can be found online at:

https://NetBeans.org/kb/docs/Java/quickstart.html

This exercise also includes information about:

- comments in Java source code,
- the elements of the source code in a new Java project,
- how to copy Java projects,
- how to submit Java projects as homework assignments in *Computer Science 111*.

You should complete the previous sections of chapter 1 before starting this exercise. The exercise uses the ***Java SE Development Kit 8 update 11*** with ***NetBeans 8.0****, as* installed in the labs and classrooms at CCP. Appendix A contains instructions for downloading and installing the software at home.

Hello World!

We will create a simple program that displays the message "*Hello World!*" – traditionally the first program students learn to write in a new programming language. *Hello World* appeared in *A Tutorial for the B Programming language* written by Brian Kernighan at Bell labs in 1972. In 1978, his *Hello World* program became famous as the first program in *The C Programming Language.* Ever since then, it has been a custom for programmers to try the *Hello World* program as the first program with a new compiler, or when learning a new programing language.

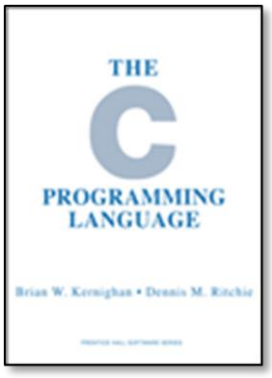


Figure 11. The book ***The C Programming Language*** by Kernighan and Ritchie featuring the first widely published *Hello Word!* program.

The *Hello World* program is available in roughly 300 different programing languages online at several Websites, including:

- http://www.mycplus.com/featured-articles/hello-world-programs-in-300-programming-languages/
- http://rosettacode.org/wiki/Hello_world/Text.

Even though *Hello World* is among the simplest of programs, we must create it inside a NetBeans project. So, first we will open a new NetBeans project. The organization of the resources and format of the NetBeans IDE depends on the type of project. We will create a simple Java console application, which means our *"Hello World!"* message will be displayed in a simple system console window on the screen. The **system console** is a combination of the system's standard input and output devices – usually the keyboard and display screen.

Figure 12 shows the console output as it appears when we run the program in NetBeans. The output – in this case simply "*Hello World*" – appears between the **run** message and the **Build Successful** message.

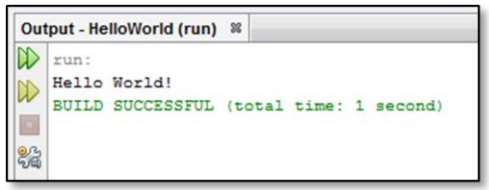
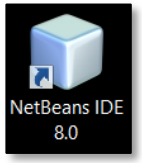


Figure 12. Hello World! console output in a NetBeans Console Window.

The steps below will show you how to create a Java *Hello World* program with console output using NetBeans. Our reference for Java-related terminology will be the **Java Language Specification (JLS).**[17]

Creating a New Java Project – Step-by-Step

STEP 1.



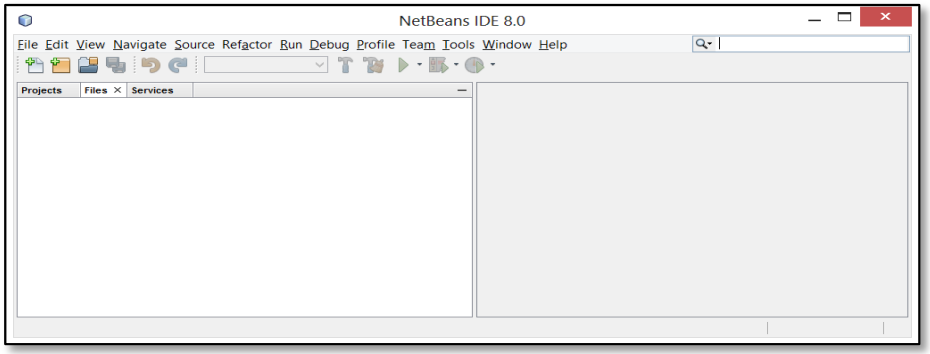
Start your computer and run the NetBeans software by using the NetBeans desktop icon or finding NetBeans on your system. NetBeans will take a moment to start, then you should see either a blank Netbeans IDE or the IDE with the *MyNetBeans* startup screen, as shown in the next column

The startup screen has three tabs:

- **Learn and Discover** – links to samples, tutorials demos online documentation, and the NetBeans Community Corner.
- **My NetBeans** – links to recent NetBeans projects and NetBeans plugins for other languages.
- **What's New** – links to NetBeans News, new tutorials, and blogs.

NetBeans IDE with startup screen

Blank NetBeans IDE



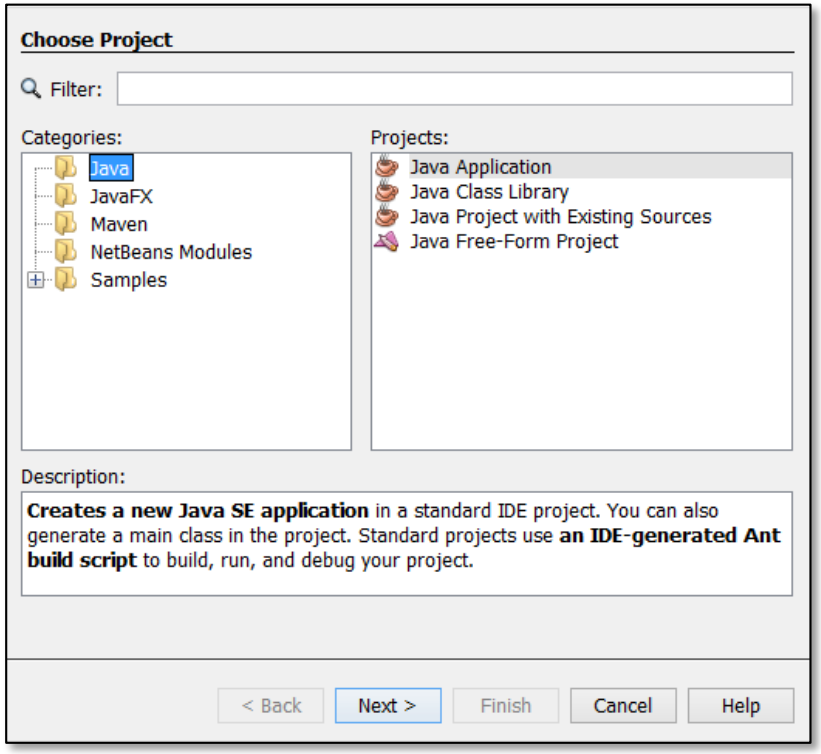
If you wish, you can deselect the checkbox in the upper-right corner of the startup screen to stop showing the screen when the IDE starts. You can bring back the startup screen by choosing *Reset Windows* from the Netbeans *Windows* drop-down menu.

[17] The *Java Language Specification*, (*JLS)* is online at: http://docs.oracle.com/Javase/specs

STEP 2.

Select *New Project* from the *File* menu.

According to the NetBeans documentation, "*A project is a group of source files and the settings with which you build, run, and debug those source files. In the IDE, all Java development has to take place within a project.*"

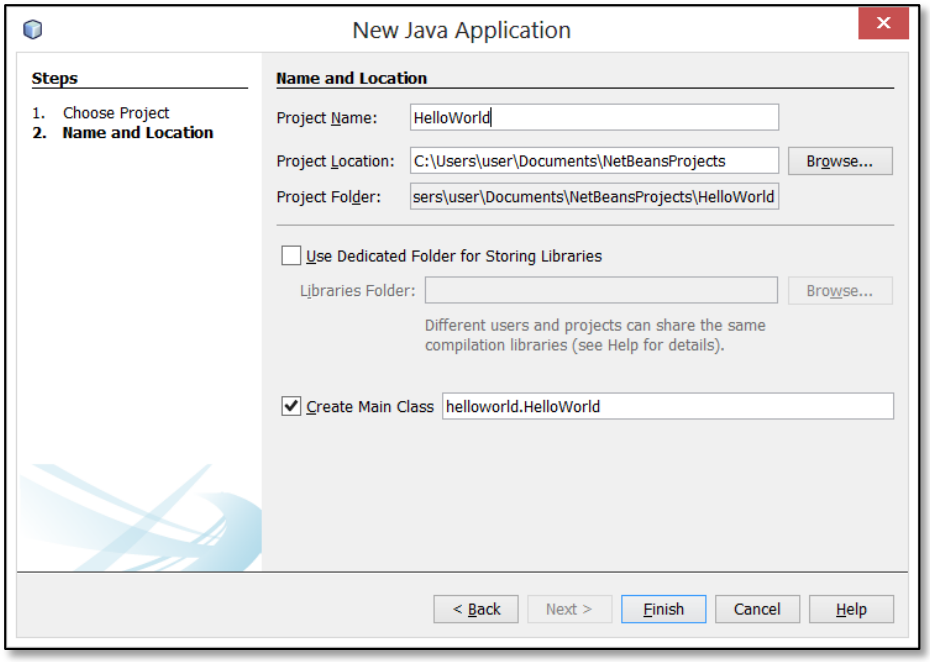


The figure above shows the *New Project Wizard*. Notice the *Categories* section on the left and the *Projects* section on the right. They are used to tell NetBeans the type of project for which it should configure the IDE.

Select *Java* in the Categories section, then *Java Application* in the Projects section. to configure the IDE for a standard Java application. Almost all programming assignments in CSCI 111 will be standard Java applications.

Click the [Next] button to continue.

A *New Java Application* window will appear as shown below. It has text boxes to enter the *project's name* and *storage location*. **Type *HelloWorld* in the Project Name field. It should be in CamelCase. (See the box about CamelCase names on the next page.)**



Leave the project location and folder as the default. On a Windows system, this will create a new folder for the project in the *Netbeans* folder in Documents (or MyDocuments). It also has options to use a dedicated folder for sharing Libraries and to create the application's main class. We will not use a dedicated folder for libraries in this project. We also should **leave the**

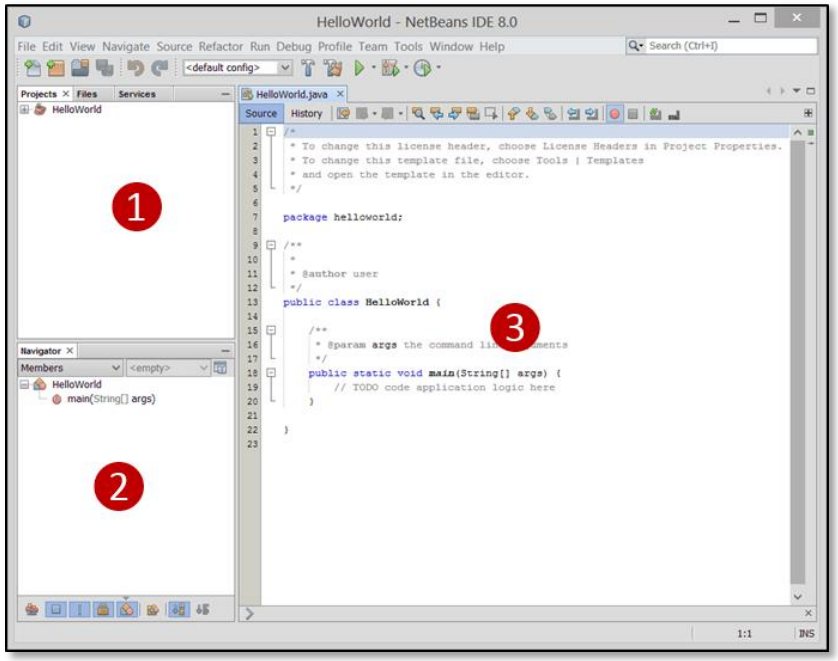
main class as the default, which is based on the project name. Most Java applications require the project, folder and main class to have the same name. *HelloWorld* will only have one class.

STEP 3.

Click [Finish] to continue.

NetBeans will take a moment to create and open a project configured for editing source code as a Java application. There are three windows open in the NetBeans interface as shown below:

1. the project window,
2. the navigator window,
3. the editor window.



HelloWorld will be a simple application with only a single source code class, and that class will have only one method. Before continuing, we will look at what's in the source code.

CamelCase Names in Java

The names of classes, objects, methods, and other items in Java generally follow the **camelCase** naming convention, used in many programming languages.

CamelCase is the practice of writing compound names without using blank spaces, but capitalizing the first letter of each name that forms the compound name, as in "*CamelCase*." If the first letter of the compound name is upper case, then we have upper CamelCase. If the first letter is lower, then we have lower camelCase.

Java class names are usually in upper CamelCase, while the names of instances of a class, properties, methods, and variables are usually lower camelCase. This is not a requirement for Java programming that will be enforced by a java compiler, but is instead a Java programming convention. **Programming conventions** are rules that most programmers follow to make the work of one programmer compatible with the work of another, and easier to understand. It is good practice to follow known programming conventions.

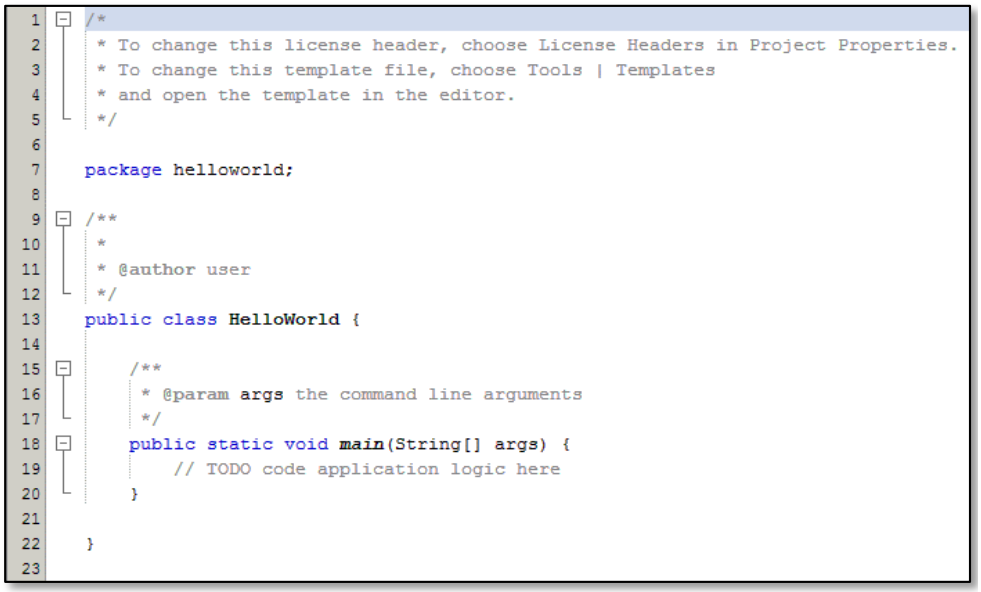
Sometimes there is a slight variation in the naming convention to aid readability and make the meaning of a name easier to understand, such as a property named *studentID* where the I and D are both capitalized.

Oracle, Microsoft, Apple and many other companies support the use of support the use of CamelCase names in software development. Quoting from Oracle's *Java Language Basics* tutorial "*If the name you choose consists of only one word, spell that word in all lowercase letters. If it consists of more than one word, capitalize the first letter of each subsequent word.*" (See *Naming* on the *Variables* page, online at http://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html)

Description of the Source Code in a New Java Project

The default source code for the new Java Project created by the Java Application Wizard in NetBeans is shown below. Take a moment to look it over. Notice that each line in the Editor is conveniently numbered. This is a reference feature of the editor, not part of the Java code.

Default source code in the NetBeans IDE with a new blank Java Application



```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6
7  package helloworld;
8
9  /**
10  *
11  * @author user
12  */
13 public class HelloWorld {
14
15     /**
16      * @param args the command line arguments
17      */
18     public static void main(String[] args) {
19         // TODO code application logic here
20     }
21
22 }
23
```

Some of the items in the source code are comments. **Comments** are notes for programmers reading the source code, but they do not affect the way the program runs. In fact, comments are stripped out of the code as it is compiled. Comments are important. They appear as documentation, helping to make our source code easier to read and understand – for both ourselves and other programmers. Comments can be included in Java source code as in-line comments or block comments.

An **In-line comment** begin with a *double slash*, like this:

```
//This is an in-line comment.
```

Everything after the double slash is a comment, ignored by the compiler, but everything before the double slash is part of the code, not part of the comment. All of Line 17 in the source code in Figure on the left is an in-line comment.

A **block comment** begins with slash-star and end with a star-slash, like this:

```
/*  This is a block comment.  It is three lines long.
 *  In this comment, the star-slash is on the third line,
 *  lined up with the beginning slash-star.
 */
```

Star refers to the *asterisk* key, on the same key as the number 8 on a QWERTY keyboard. Everything typed between the slash-star marking the beginning of the comment and the star-slash marking the end of the comment – no matter how many lines – is part of the block comment and will be ignored by the compiler.

Many programmers line up the stars in the slash-star at the beginning of a block comment and the star-slash at the end. They may also place a star on each line to make it clear that this is part of a comment. The NetBeans Source Code Editor will automatically add a star the beginning of each line in a block comment. Lines 1 through 4 in Figure 18 show this.

Notice that in this case there are no words on the first line of the block comment – just the symbols marking the beginning of the comment. Again, this is just programming style, not a Java compiler rule. It is really up to you how you format your block comments, as long as they are clear and easy to read. You should probably ask your instructor in a class or check with your employer on the job to see what they would like you to do.

There are three block comments and a single one line comment in the code that the Java Application Project Wizard created to help get us started.

Lines 1 through 5 are a comment about editing License headers and the *Java Application Project Wizard Template*. (Don't even think about trying this until you are a more experienced NetBeans and user.)

Lines 7 is the package directive telling the compiler which Java package will contain this software. Each CSCI 111 program will be in its own packages. This name is generated by the *Java Application Project Wizard Template.* Later we will learn about establishing our own packages.

Lines 9 through 12 are a comment identifying the author of the code. We will expand on this later in this exercise.

Lines 13 is the class header. It identifies the beginning of the HelloWorld class. The class is a public class, meaning that is can be accessed by software from outside the package. At least one class in each package must be a public class, so that the software can be run from the operating system.

Lines 15 through 17 identify any parameters for the method following on line 18. A **Parameter** is a value that is passed to a method when it runs. For example, if a method finds the sine of X degrees, where X could be any integer, then the value 45 might be pass ed to the method as a parameter to ask the method to find the sine of 45 degrees.

A method can have more than one parameter. Each parameters passed to a method is also called an **argument** of the method, and the list of parameters at the beginning of the method is called the **argument list**. The comment on lines 15 through 17 is often used to describe the arguments in the argument list. It is also a good idea to use a comment before a method to describe what the method is intended to do.

This method does not use any parameters, although a String array named *args* is allowed for by the presence of **String[] args** in the method header.

Line 18 the header for the *main* method – the only method in our class. The name of the method is **`main`**, with a lowercase *m*. It is a Java naming convention to start method names with a lowercase letter.

Every application must have at least one class with a main method. This is the place where our application will start. When a java application is executed, the main method is the first method that will run. In the case of our *Hello World* application, it is the only method that will run. This will be true for our first few programing exercises in CSCI 111.

The keywords **`public`**, **`static`** and **`void`** in the method header are important.

public means the method can be run from outside the class.

static tells the compiler that the method is associated with the class itself, not with an instance of the class. Only static methods can be run without a reference to a specific instance of an object. We will learn more about this later in the semester. For now, our methods will be static methods. **void** means that method does not return a value, as, for example, a square root method might. We will leave these keywords as is. Every Java application's main method must be a **`public static void`** method.

You should also notice that blocks of code in Java begin and end with **braces** `{ block of code }`. (Braces are also often called brackets. The Java Language Specification uses the term *braces*.) This includes classes, methods, and other units of code. The NetBeans wizard places the beginning brace on the same line as the class or method header, then places the ending brace on a line by itself after the last line of the block. The blocks of code should be indented to make the blocks easier to identify. The NetBeans Source Code Editor will indent blocks of code automatically.

Java source code with standard use of begin-end braces

```
public class HelloWorld {

    /*
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        System.out.println("Hello World!");
    } // end main()

}// end class HelloWorld
```

The figure below shows you a commonly used alternate method for lining up the braces in a block of code. This method lines up the beginning and ending braces for each method, and includes an in-line comment identifying the block of code's ending brace. **No matter how you line up the braces, it is always a good idea to put identifying comments on ending braces.**

Java source code with an alternate use of begin-end braces

```
public class HelloWorld {

    /*
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    } // end main()

}// end class HelloWorld
```

You should ask your instructor in a course or your employer on a job how to line up braces for blocks of code in a particular programing environment.

Java code must run correctly and efficiently, but the readability of the source code is also very important. Comments and programming style should make the source code easier to read and understand. Sometimes a small touch can make a difference – in this example, line 18 is blank. The separation between the two lines with ending braces makes the code easier to read in a subtle way.

Next we will finish our exercise by modifying the code created by the Java Application Wizard to create our *Hello World* program.

Hello World – Step-by-Step

STEP 4.

Delete the comment above the package directive on lines 1-5. The package directive is now line 2. Comments have been added to the closing with text cut and pasted from class and method headers.

```
1
2   package helloworld;
3
4   /**
5    *
6    * @author user
7    */
8   public class HelloWorld {
9
10      /**
11       * @param args the command line arguments
12       */
13      public static void main(String[] args) {
14          // TODO code application logic here
15      }
16
17  }
```

STEP 5.

Add a new block comment at the top of code to include the following identifying information:

- Add a line to identify the project – **`Hello World Application`**
- Ad a line to identify the purpose or context for the project – **`created for CSCI 111.`** Your instructor may also want you to include the semester and section number.
  Add a line indicating when the code was last edited – **`last modified [date and approximate time]`**.
- Identify the programmer (author) of the code - **`@author [your name]`**.
  `@author` is a JavaDoc tag, which we will learn about later in the semester. For now, be sure to include the tag in front of your name.

```
1  /* Hello World Application
2   * created for CSCI 111
3   * last modified August 30, 12:09 pm
4   * @author Chuck Herbert
5   */
6
7  package helloworld;
8
9  /*
10  *
11  * @author user
12  */
13 public class HelloWorld {
14
15     /**
16      * @param args the command line arguments
17      */
18     public static void main(String[] args) {
19         // TODO code application logic here
20     }
21
22 }
```

STEP 6.

Delete the comment on lines 9 through 12. It is no longer needed because of the information in the code's opening comment.

```
1  /* Hello World Application
2   * created for CSCI 111
3   * last modified August 30, 12:09 pm
4   * @author Chuck Herbert
5   */
6
7  package helloworld;
8
9  public class HelloWorld {
10
11     /**
12      * @param args the command line arguments
13      */
14     public static void main(String[] args) {
15         // TODO code application logic here
16     }
17
18 }
```

STEP 7.

Add aline to the block comment that on line 12 to briefly describe what the method does – "`main method prints a message on the screen`".

The comment already has a Javadoc parameter tag (**`@param`**) identifying the method's parameters. We will not use these parameters, but they could be used by a more experienced java user, so we will leave the **`@param as is.`**

It is also always a good idea to include a brief comment at the beginning of a method to state what the method does (or at least what the programmer thinks it does.). That is what we are doing in this step, resulting in the code shown on the next page.

```
1  /* Hello World Application
2   * created for CSCI 111
3   * last modified August 30, 12:09 pm
4   * @author Chuck Herbert
5   */
6
7  package helloworld;
8
9  public class HelloWorld {
10
11     /*
12      * main method prints a message on the screen
13      * @param args the command line arguments
14      */
15     public static void main(String[] args) {
16         // TODO code application logic here
17     }
18
19 }
```

STEP 8.

Replace the TO DO comment on line 16 with the instruction:

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

This is the actual Java code to be executed. The TODO comment is only a placeholder put there by the NetBeans wizard.

```
11     /*
12      * main method prints a message on the screen
13      * @param args the command line arguments
14      */
15     public static void main(String[] args) {
16         System.out.println("Hello World!");
17     }
```

Notice that the new line ends in a semicolon. This is necessary in Java. Each individual instruction in a block of code ends in a semicolon in Java.

The **`System.out.println`** part of the command tells Java to print the line on the standard system console output unit.

- **`System`** tells Java to run a command from the package of system commands in its library.
- **`out`** tells Java to use an output stream – to send something to the console output.
- **`println`** tells Java to print the message then start a new line. **`print`** by itself, without the **`ln`**, would tell Java to print the message then wait on the line for whatever comes next. **`println`** tells Java to behave as if [Enter] is pressed after the message, whereas **`print`** is similar to typing something without pressing [Enter] at the end of the line. Here we use **`println.`**
- **The parentheses** enclose the argument of the command – what is to be printed.
- **The quotes** are needed for string literals – this means we are telling Java to literally print the exact string of characters between the opening and closing quotes – **`Hello World!`**

STEP 9.

Add an in-line comment after the ending brace on line 17 – `// end main().` This makes it clear that the brace is for the end of the main method. It doesn't seem like a big deal for our short program, but in longer programs with many blocks of code it adds to readability and could save hours trying to debug code with a missing brace.

NetBeans Notes – Resizing IDE Windows

The windows in the NetBeans interface can be re-sized by clicking and dragging the boundaries between the windows, either horizontally or vertically.

STEP 10.

Add an in-line comment after the ending brace on line 19 –
`// end class HelloWorld.`

Make sure to leave a blank lline between the end main brace and the end class brace. This it easier to see the structure of the code when reading the code.

```
1  /* Hello World Application
2   * created for CSCI 111
3   * last modified August 30, 12:09 pm
4   * @author Chuck Herbert
5   */
6
7  package helloworld;
8
9  public class HelloWorld {
10
11     /*
12      * main method prints a message on the screen
13      * @param args the command line arguments
14      */
15     public static void main(String[] args) {
16         System.out.println("Hello World!");
17     } // end main()
18
19 } // end class class HelloWorld
```

NetBeans Notes – Copy and Paste

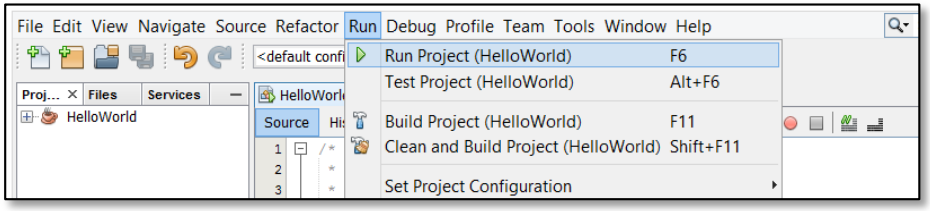
Here's a tip to improve your efficiency and cut down on typographical errors: use *copy and paste* to copy method or class header information to the comment marking the block's ending brace. Generally, cut, copy and paste are a programmer's friends.

```
Public class HelloWorld
    {
      . . .
    } // end class HelloWorld
```

Now we are finished editing the program and it is ready to run. It should resemble the code in the column on the left.

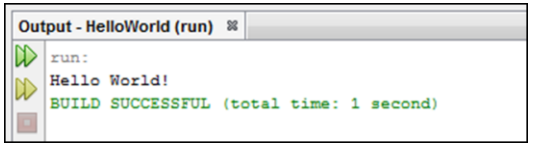
STEP 11.

Run the application by selecting *Run Project* from the *Run* menu.



The **Run Project** command is at the top of the menu. You could also use **Run File**, but only if the cursor is in the Source Code Editor window before opening the menu. Pressing the F6 keys also runs a java application in Netbeans.

It will take a moment for Java to compile the source code before it runs the application. The image on the right shows us the console output as it appears when the program runs from inside NetBeans.



Hello World! console output in a NetBeans Console Window.

The output from the program appears in a window below the Source Code Editor, with the actual output from the application between the *run* message *and the Build Successful* message.

Source code and other project elements are automatically saved when we run a project in NetBeans. We do not need to save before running, although it is a good idea to save any code that you have worked on for a while without either running or saving the project.

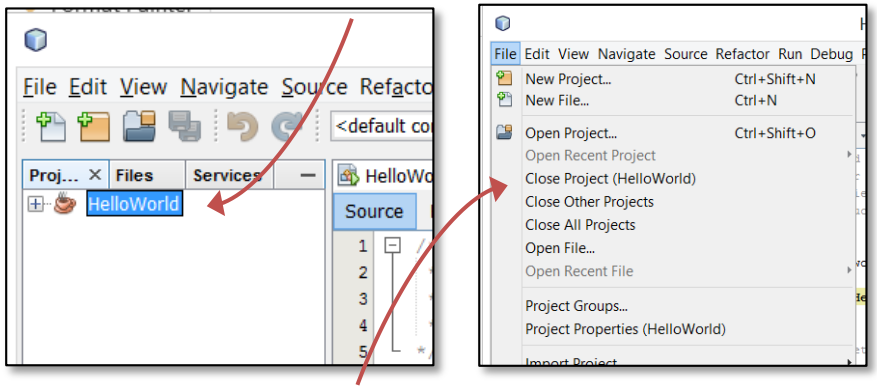
Closing NetBeans Properly

Projects remain open in NetBeans until they are purposely closed. If you exit NetBeans without closing a project, then normally the same project will be open when you restart NetBeans. You must specifically close a project when you are finished working on it.

> **Warning! – CCP System Reset**
> The systems in the labs and classrooms at CCP are often set to delete student-created files from the hard drives when the systems are re-booted. Please remember to copy your work from the hard drive before turning off the machines, re-booting the machines, or leaving the room.

STEP 12.

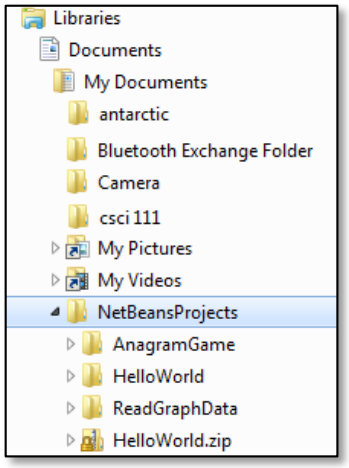
Select the *HelloWorld* project in the Project window,



then select *Close Project* on the NetBeans *File Menu*, to close the project.

After closing the project, the NetBeans interface should be blank. When you are ready to exit NetBeans, select **Exit** from the bottom of the File menu. When you re-open NetBeans, you can open your project by selecting **Open Project** form the NetBeans File Menu, or by selecting **Open Recent Project.**

Each NetBeans project is stored in its own folder, within the ***NetBeansProjects*** folder on the host computer. If you installed *NetBeans 8* on a Windows system as directed in *Appendix 1*, then the *NetBeansProjects* folder will be in the *Documents* folder (*My Documents* on some systems) as shown on the right. On other systems, or if you are not sure where the project folder is, then you should search for the ***NetBeansProjects*** folder and check its contents.



To open an existing project in NetBeans, first start NetBeans, then use the **Open Project** item on the NetBeans file menu.

To copy a NetBeansProject from one computer to another, copy the individual project folder from the *NetBeansProjects* folder on one system to the *NetBeansProjects* folder on another systemYou could also zip/unzip the project folder to make moving it or storing a copy of it more convenient.

If you have several copies of a project with the same name, be careful to keep track of which one was most recently edited, especially if you are working on more than one system.

To submit a CSCI 111 programming assignment to your instructor:

- **create a zipped copy of the NetBeans project folder**.
- **send the zipped copy to your instructor.** Submit the zipped folder by using the upload link in the assignment in Canvas, as shown in Figure 24. If you have any trouble with this, send it as an attachment to a Canvas message or a regular email message.

Chapter 1 Summary

Lesson 1.1 provided a brief overview of Computing and Computer Science.

Lesson 1.2 discussed the nature of algorithms and introduced the concept of an object in modern computer programing.

Lesson 1.3 described the historical evolution of some of the more prominent computer programming languages.

Lesson 1.4 introduced The Java Programming Language.

Lab 1 showed you how to create a simple Java program using Netbeans.

Key Terms in Chapter 1

Warning! – Flash ROM Failures

USB flash ROM drives are convenient, but they can suddenly go bad without warning. They are good for moving data from one computer to another, but you should not rely on them as a backup copy of your data, or as the only copy of your data.

USB flash ROM technology is good for a limited number of write operations. The cheapest chips are good for about 10,000 write operations. If you are working directly from the chip, you could have as many as 100 write operations in an hour. Someone in the habit of working directly on the USB device could have the chip go bad in a few weeks of use. (More robust industrial grade flash ROM chips last for up to 1 million write operations, but are expensive – $50 to $75 per gigabyte.)

Instead of working directly on a flash ROM drive, you can copy your work to a computer's hard drive, do some work on your program, then copy your work back to the USB drive when you need to move it to another machine.

If a file or folder is not too large, you can send it to yourself as an email attachment, then open the email message on another computer and save a copy of the attachment.

Chapter 1 - Questions

1. What are the primary components of most computers? How do they communicate with each other?
2. What is inside a central processing unit and what is the function of each of these things?
3. What are the differences between internal memory and external storage? What are two distinct types of internal memory?
4. What electromagnetic technology is commonly used for external storage? How does it compare with internal RAM technology?
5. What are peripherals? How do they communicate with the CPU?
6. In which courses do students study the organization of modern computer systems and the transfer and control of data in greater detail?
7. What do computer scientists study?
8. For whom are algorithms named? Where and when did he do his most important work? What method for solving math problems did he present in his work? Why is his work important in the history of math and computer science? How are algorithms related to computer programs?
9. Why did computer scientists introduce object-oriented programming? What can be represented by an object in object-oriented programming? How are data and programs organized in object-oriented programming?
10. What is the difference between a computer engineer and a software engineer? How does their work relate to computer science and other fields?
11. What organization is responsible for accrediting computer science degree programs? What organizations sponsor the accrediting agency, and what do they do? How can it be beneficial for a computer professional to belong to one of these agencies? How can we find out more about their student memberships?
12. What are some of the more common specializations for computer scientists? What does the ACM have to bring together computer scientists in specialized fields within computer science?
13. What is the difference between a high-level language and a low-level language? How does assembly language fit into all of this? How many different assembly languages are there?
14. How do compilers and interpreters compare to one another? What is the process for translating Java into machine code for specific systems? What design goals for the Java language led to the development of this method? How did other major companies react to the way Java is implemented? What indications do we have of the success of Java's method of implementation?
15. What are some of the major programming languages that have been important in the history of programming languages, and how do they relate to one another?
16. What early programming language that was not widely used commercially created a foundation for most modern programming languages What are some of the more important features that first appeared in this language?
17. What language was based on Lambda Calculus, and what are some of its modern derivatives? What field of computer science has been most associated with this language? Why is lambda calculus important in computer science?
18. What is a teaching language? What is a scripting language? How is a teaching language or a scritping language used for prototyping software?
19. Why are Simula and Smalltalk important in the history of computer programming? Who developed these languages?

    How is Java related to C and C++?
20. How did Guido von Rossum suggest Python and Java can be used together?

Chapter 1 - Exercises

1. Modify the *HelloWorld* project from the exercise is this chapter to print the message "Hello Dr. Kernighan!" on a line after it prints "Hello World!" Remember to update the comments in the code.

2. The bulleted list on page 31 tells us:

   > *"**println** tells Java to print the message then start a new line. **print** by itself, without the **ln**, would tell Java to print the message then wait on the line for whatever comes next. **println** tells Java to behave as if [Enter] is pressed after the message, whereas **print** is similar to typing something without pressing [Enter] at the end of the line..."*

   So the following two lines will print **`Hello. Goodbye.`** on the same line of console output:

   ```
   System.out.print("Hello. ");
   System.out.println("Goodbye.");
   ```

   Create a new Java project similar to the HelloWorld project uses print and println to print the text from a business card for you. The output should look something like this:

   ```
   Chuck Herbert
   Computer Science major
   Community College of Philadelphia
   Philadelphia, PA  19130
   cherbert@ccp.edu
   ```

   It should include the following items as String literals in the printing commands, not variables:

   first name, last name, major, school, city, state, zip, and *email address*.

   No two items from this list above may be printed by the same command, so for *example your first name should be printed using a **Print** instruction, then* your last name printed using a println instruction.

   Blank spaces and the comma between city and state should be included.

   Don't forget to include the proper comments in your code.

3. Create a diagram that shows the lineage of Java from the earliest programming languages. It should look like a family tree, showing which languages led to which other languages, that, in turn, eventually lead to Java. For example, our tree would show that both the *C* language and *Simula* influenced the development of C++.

4. Which of the people mentioned in this chapter do you think is most important or most interesting? Research the person and write a short biography of the person, including what the person did in addition to what's mentioned in the chapter. Cite your sources. You must use at least two sources of information other than Wikipedia. You can read the Wikipedia article, but should not cite it as source since its information is not fully verified as correct. Original sources are the best.

5. Identify one of the programming languages in this chapter and look up the language on the site Webopedia.com. Webopedia is good source of information about technical terms related to computers and computer science. Starting with Webopedia and moving on to other sources, see what you can find out about the language write a short description, including what it primarily used for. List several sources of more information, such as language specifications, sources of compilers and interpreters (especially if they are free), and tutorials, and sample code.

6. There are a number of online forums and blogs discussing programming languages. Log into one of these and create or join a conversation addressing the question "What is best programming language is to learn

and why?" Consider the difference between the easiest language to start with and the best language to know professionally. See if the comments posted by others seem well-founded or not. Do other people describe what their opinions are based on? Do there seem to be biases for or against certain languages, operating systems, or hardware? Which of the posts seem helpful or not helpful, and why? Check the site every day for a week, and write a summary of what you observed and learned related to the items mentioned above.

7. Two websites that show "Hello world!" source code in different programming languages are listed on page 23 of this chapter. Using one of these sites, find the code for "Hello world!" in three different languages – one that seems very easy, one that seems similar to Java, and one that seems complicated. Write a short report, which includes the code, telling what language each is in and something about the language, such as what the language is primarily used for. Look up the code for Java on the site and see if the Java code it shows is complete enough to run as a program by itself. If not, what missing? What does this tell you about the completeness of the code in other languages?

8. A list of some specializations in computer science can be found on page 6 along with a link to the ACM's special interest group page - http://www.acm.org/sigs . Information about the ACM and the IEEE can be found on page 5, including references to their main websites. What specialization in computer science most interests you, even if you have a major other than computing? What resources, including special interest groups, do the ACM or IEEE have related to your area of interest? What special interest groups might you be interested in joining? What courses other than computer courses might be useful for your area of interest?

9. The U.S. Department of Labor's Bureau of Labor Statistics maintains the Occupational Outlook Handbook with career information for most job classifications in the U.S. It is online at http://www.bls.gov/ooh/ . (You can get there easily by typing OOH in Google or most other search engines.) They also publish a more current *Occupational Outlook Quarterly*. Find out what they have to say about careers in computer science and closely related fields. What do the jobs pay? What is the work like? What education or background is needed? Are there many openings in that particular area? Write a short report summarizing your findings for several areas related to computing.

10. In addition to NetBeans and Eclipse there are quite a few others IDEs that can be used for for Java software development, such as *JCreator*, *BlueJ*, *JBuilder*, *IntelliJ, DrJava*, *Javelin*, and *Websphere*. See what you can find out on the Web about some of them and write a short report describing your findings.