Programming Primer for Object-Oriented and Procedural Languages

by: $h@v3-uR-$n@+cH
(4.2.2001, Ishikiri, Japan)

PREFACE

I released a very brief guidebook about how to learn Java titled "The Simple and Effective Way to Learn Java" (only a 13k pdf) in the Gnutella network recently, and it was brought to my attention by a number of individuals that the material contained in the tutorials was not suitable for the uninitiated beginner, the person who doesn't even know what a variable is or what "heap" means.

Lots of folks want to learn Java, Eiffel, C++ or some other OO (Object Oriented) language but they simply can't start in the middle, and 99% of the tutorials and information on the web - or even in books you can buy! - are written for people who have at least some expierience with programming in C++, Visual Basic, or Pascal v7.1.

So, without further delay, here is a primer for the absolute beginner. It walks through the most basic of computer science concepts and works the true beginner up to the point where he can leave this primer and move on down the list of tutorials and reference resources I have included at the bottom and eventually become a real programmer.

Be aware of a few things before you move on to the meat of this primer:

1 - This is intended for the absolute beginner and will cover only the most basic of CS (Computer Science) terms and concepts. In other words, you will NOT be a programmer by the time you finish reading this.

2 - The OO language I am personally most familiar with is Java, and as the greatest current amateur educational demand is Java (hey, it's free), this primer includes mostly Java-related references and resources at the end.

3 - All of the concepts covered here are common to all programming languages, both procedural languages and OO languages.

4 - If you are an absolute beginner with computer programming and you want to learn, but you're getting discouraged by all of the free educational resources on the web that are aimed at educating the folks who are already educated, then this is for you, regardless of what your target language is.

5 - This primer is just that, a primer. It is for people who want to learn, but just don't know enough yet to understand all of those free educational programming resources on the web.

6 - Most importantly, this is 100% Public Domain literature. If you paid ANY money for ANYTHING contained herein, you got ripped off. Nobody can charge any money for this ever, and nobody can ever copyright it, either...and if they try I'm going to do something quite nasty to them.

7 - I'm a normal guy...I make mistakes. If I did something stupid and misspelled a few words, mistyped a few links, whatever, don't send me hate-mail! I'm appologizing in advance for my deficiencies.

NOTE ABOUT THE ABSOLUTE BASICS

This primer is centered on Object Oriented technologies, of course, but before you can understand OOP (Object Oriented Programming) or procedural programming you must understand some very basic programming concepts that future tutorials, educational websites and books will gloss over, assuming you already know them (even the books that claim they are for the "absolute beginner" gloss over 90% of the truly basic material).

This might be boring, but please bear with me. It is very neccessary that you understand these concepts before you go any farther.

WHAT MAKES A COMPUTER WORK?

This is a very basic, common question. I like to think of a computer as a zombie, dedicated only to doing whatever I, the user tell it to. That is only true to a certain extent, as the computer does not always do what I tell it to do (sometimes it makes an error or is not turned on), it has limitations (it cannot make me dinner), and it doesn't really do exactly what I tell it to do, it does what I tell it to do based upon a set of rules someone else already gave the computer.

So when you use a computer, you are using a computer only in the same sense that a general uses a private, via someone else or something else. That someone or something else is a protocol or a mediator, for computers this is a program.

So, basically, programs are the most basic instructions for the computer. When I type a letter, the computer prints it on the screen. Why? Because the computer assigned a number to each key on the keyboard because a certain very basic program told it to according to a certain set of rules. The computer identifies the keys in a certain way based upon another set of rules, and I am running a program that gives the computer instructions to explicitly print and record the keystrokes I make in a certain order.

In other words, the computer functions based upon pre-ordered rules which in turn are based upon other pre-ordered rules.

The rules and instructions stack up on each other until the desired function is achieved.

This may be an oversimplification, but for most purposes, this way of thinking will suffice.

COMPUTER LANGUAGES

If we want to set new rules for the computer on our own (read as: "write a program") we have to communicate our rules and instructions to the computer in a very explicit and direct way. Humans speak, think and read in complicated and flexibly-structured languages or images; unfortunately, computers can only think in 1's and 0's (the old binary number system you learned a long time ago in elementary school). This makes it nearly impossible for a person to communicate anything to a computer in the computer's native language (called "machine language").

We have developed other languages that are a mix of psuedo-English (well, usually English), and machine-definable concepts. These psuedo-English languages are computer languages. The grammar (syntax) of each specific language is not of great importance, per se; it is how the compiler translates the psuedo-English into machine language that really matters. Some languages are more powerful than others, not usually because of the specific syntax of each language, but because of the difference between compilers.

I keep saying "compilers" but maybe you don't even know what that is just yet. A compiler is a program that can act as a translator between you and your psuedo-English and the binary machine language the computer understands.

So think about a compiler as a real-world translator. Some translators know more idioms than others, and therefore can express an idea more quickly and colorfully than others. Some translators specialize in business terms and language, some in scientific or colloquial grammar. Each translator has his own advantages and disadvantages over the other translators.

This is where the difference between computer languages really appears. Some compilers are better for some things than others are. Some simply can't do things that others can. Some languages were written in a very short-sighted manner and even the syntax cannot support certain concepts that the compiler could implement in

the future if only it were rewritten... and this is exactly why we have different versions of different computer languages. The syntax basically stays the same, but new functionality has been added (or in some cases taken away) from the compiler (well, really the API gets changed, but don't worry about that right now. Just think "compiler"!).

So, compilers are translators between you and your computer. They specifically translate rules and instructions you want to give the computer.

Computer's think in binary numbers, only.

Sets of rules and instructions that computers follow are called programs (as if you didn't know that already...).

Again, this is an oversimplification, but it will suffice 90% of the time.

Examples of computer languages:
C, C++, Eiffel, FORTRAN, BASIC, Pascal, COBOL, Java, Modula-2, PL/I, Ada, Logo

Examples of simple syntax variation by computer language (very simple addition example):

C := A + B; (Pascal)
C = A + B (FORTRAN)
C = A + B; (PL/I)
C = A + B (BASIC)
ADD A,B GIVING C (COBOL)
C = A + B; (C)
C := A + B; (Ada)
C := A + B; (Modula-2)
MAKE "C : A + :B (Logo)

WHAT YOU SHOULD KNOW BEFORE YOU MOVE ON

OK, basically I have either just given you a few things to chew on if you are completely, 100% new to computer programming or I have reiterated the very basic concepts that you already know about computers and programming.

I am not going to write an Intro to Programming 101 course in this little primer, but with the basic knowledge and understanding that you have from the first two sections, you can either figure out the things I am about to write, or go look them up topic-by-topic on the web somewhere as I go.

(Don't worry, this will be very easy to understand and slow-going. Also, each topic and keyword will be specific enough to look up on the web at any reference point I place at the end of this primer.)

Key Concepts

- Six steps in problem solving (in computer science) are: Analyze the problem, Develope an algorithm, Write code for the program, Run the program, Test the results against answers manually arrived at (either with pencil and paper, or checked against a known program), Document the program.

[NOTE: These steps are valid for both OOP and procedural programming. The two types of programming part ways, however, when it comes to the specifics of actually writing the code. More on that a little later...]

- Top-down design is a process of dividing tasks into subtasks until each subtask can be readily accomplished. In other words, you take a problem you need the program to solve, and you divide the overall problem into a series of mini-problems. This is done in real-life, too.

Problem: My room is messy. Remedy: Clean my room. OK, so how do I do that? First, I need to get individual

things organized. Next, I need to make sure the things are actually clean. Then vacuum, dust, etc.

Each one of these steps can be subdivided a number of times until one, specific instruction exists for each step in the cleaning process. In OOP practices, subdivision generally takes the form of creating objects to perform each micro-task and creating a main object to control the whole process. The procedural process is similar, but not as modular or portable. The main routine or process controls a series of sub-processes or sub-programs. The basic idea is the same, but the implementation is very different in procedural programming and OOP.

- Stepwise refinement refers to refinement of tasks into subtasks.

- A structure chart is a graphic representation of the relationship between modules (or objects).

[Again, the idea of a structure chart is similar in both procedural programming and OOP. The difference comes from the actual implementation of the ideas.]

- Software systems are large, interconnected systems of programs that work together to perform a series of related tasks. (Oracle's business suite, for example...)

- Software engineering is the process of devloping and maintaining large software systems.

- The software system lifecycle consists of: Analysis, Design, Coding, Testing/Verification, Maintenance, Obsolescence.

- You cannot tell a computer to do anything more than mathematically process information. Everything you ever see a computer do - even in the most abstract-looking video-game - is merely a complicated set of math problems. Usually, the problems have been subdivided a gozillion times so the computer can process them. (Remember stepwise refinement and top-down design above? They are very similar to what your 8th grade algebra teacher wanted you to do when she said, "SHOW YOUR STEPS!" which is something I never did, and so I had some trouble adapting to programming...) The basic idea here is, computers can only do very simple mathematical computations... but they can do them repeatedly, 100% accurately, and REALLY fast.

- Computers need to know what variables (just like good old "x" and "y" in algebra class) you intend to use and what their values are before you tell the computer to use them.

Key Terms

You, as a programmer NEED to know these following terms before you move on to any real tutorials:

[WARNING: A short glossary follows. I recommend either printing this out or writing down the terms you don't understand so you can study them in a timely manner...]

Algorithm : A finite sequence of practical (remember stepwise and top-down) instructions or procedures that will solve the specific problem at hand (or perform the specific desired task).

ANSI : American National Standards Institute. ANSI-standard programming languages conform to ANSI recommendations in order to eliminate subtle or extreme variations that could cause problems in transporting programs from one type of computer system to another. Unfortunately, this doesn't exactly work out the way they planned. ANSI is generally accepted (along with ASCII and Unicode) as a text-character representation standard. In ANSI, ASCII, and Unicode, characters are represented by on a table as a number (remember, computers can only think in binary numbers...). Which number is assigned to which charater is generally established by one of these three systems.

Applet : A teeny-tiny application - usually written in Java - that runs in a browser window.

Application : A program designed for a specific use. (Examples: Notepad, MS Word, Calculator, Diablo II

runtime system, etc.)

Application Programming Interface (API) : An application through which you to program other programs. An API is "the language" as far as you are concerned. Whenever you see "API specifications..." just think "the specific language specifications..."

Argument : A value that is passed to, and used in an operation or function. In the military there is an established system of facing movements (soldiers in formation turning to face one way or the other upon verbal or flagged commands). Each "Face" command carries a specific argument with it (no, not a verbal argument of disagreement... this is different...). "Right Face!" is a command to face which way? Right! I can't just say "Face!"; it doesn't make much sense. But "Right Face!" has some meaning, made specific by the argument "Right". In computer terms, "Face" is the keyword or command word, and "Right" is the argument passed to the process when it is called.

ARPAnet : The US Advanced Research Project Agency network, which was the first large multi-site computer network.

Array : A list of variables (remember "x" and "y" from algebra class...), all with the same name and data type (but usually numbered in a specific series within the array itself). You can think of this like an algebraic series of numbers. {1, 2, 3, 4...} is a series of numbers. It also is an array that contains ALL positive integers. In computer terms, the series {(0) 1, (1) 3, (2) 5, (3) 7, (4) 11...} is an array of all prime numbers with a locator (or internal name within the array) that is in parenthesis. So if the array name was PRIMES, the number 5 would be "called" as PRIMES(2), or PRIMES.2, or PRIMES[2] or whatever, depending upon the specific method forced upon the programmer by the syntax of that specific language.

[NOTE: An array is not the same as a mathematical set. It is merely a useful analogy. You will learn the differences later. They are related concepts.]

ASCII : American Standard Code for Information Interchange. The same deal as ANSI in the previous definition.

Binary Digits : Counting by 1 and 0. Remember, we count in "base-10" normally. We could also count in base-2 if we wanted. Base-2 is called "binary" and base-10 is called "decimal". Easy...

Boolean expression : Don't ask me where the word "boolean" came from, cuz I don't have a clue. But I do know what a boolean expression is! It contains a very simple "yes" or "no" value. Just like 1 or 0. Usually represented by "true" or "false". (3 + 4 = 67) = "False". This is a boolean statement. Boolean operators can also be "and" or "if"... whatever. All that stuff you learned in logic class. The simple logic stuff is all boolean. Well, just think of it that way for right now. "true", "false", "either"..."or", "and", "if"..."else", etc.

Client : A workstation or personal computer that gets and uses information from other computers. You computer is a client right now, you downloaded this pdf file from somewhere. That somewhere was acting as a server at that time.

Comment : A little note instered into the actual psuedo-English code of the program. The compiler totally ignores this message when it translates the psuedo-English (the "source code") into machine language. It is there for the benefit of other programmers or the programmer who write the code. You can remind yourself of just what the devil you thought you were doing when you wrote the thing. Psuedo-English becomes very difficult to understand as programs get longer, and when you go back to edit a long program you are writing, the comments you remembered to insert into your code can save your day by reminding you what a particularly confusing instruction does or how it works. Remembering to insert proper comments into your code is one of the most basic and most important things you can ever learn to do as a programmer.

Floating Point : Mathematical operation based on a number expressed as a whole or a fraction, such as 1.5 or 3.421.

GUI : Graphical User Interface. This lets you use a mouse, click on pictures, etc. instead of writing in

commands on your keyboard on a boring green screen all day long. Windows, MacOS, and Linux are examples of operating systems that use GUIs. DOS, UNIX, and VMS are examples of operating systems that use command-line interfaces. Individual applications have the same differences. Just think: "The difference between a text game and a graphics game", and it will all make sense.

Hexadecimal : Remember the old base-10 and base-2 examples? Decimal and Binary? Well, Hexadecimal is just the same idea, but this one is base-16. Why have so many different number bases? Well, in certain mathematical models, different number systems provide different mathematical advantages in different situations. Some scientists prefer base-8 (called octal, I think).

IDE : Integrated Developement Environment. This is either an extension of the API or an independent program that makes programming a whole lot easier. Basically, it is an application that provides a software design and developement environment that is a lot more user-friendly than just typing up some code in Notepad and compiling it from the MS-DOS prompt on a Windoze machine...

Java : An interpreted high-level computer language that is fully object-oriented. "Interpreted" means the compiler just compiles the psuedo-English code into a middle-of-the-road stage instead of actually going all the way to machine language. The reason for this is that different computer platforms (combinations of operating systems and hardware... for instance Wintel machines versus Macs) speak different machine languages (but they're all binary). So Java compiles to a universal Java-only language, then, later, when the program is going to actually run, the Java system reads the bytecode (the middle-of-the-road thingy) and does its thing. This lets you write a program on a Mac and run it on a Wintel machine, or on a Wintel machine and run it on a Linux machine. All you need is the Java package for that specific platoform, and you're good to go. That means Java is interpreted and portable.

Object : A self-contained set of processes (those little instructions) and the variables the processes will do something to (remember, the computer needs to have all the variables defined before it will be able to do any processing...). Right now, just think about objects as little tiny programs that are designed to behave like a real-life object. You can make a table object and make it exist inside of a room object. Each object is just a little program that behaves like the real-world counterpart, or some abstract idea in your imagination. The cool thing about objects is they are portable between programs, so you can build a larger program out of pre-written little programs... kinda.

OS : Operating System. Good examples: Windows, Linux, UNIX, DOS, MacOS, etc...

Process : Like a simple math process or function. A process is about the simplest instruction you can give a computer. Since every program is a complicated combination of processes and simple functions, you can think about processes, functions, and procedures as the building blocks of programs. In OOP concepts, procedures and processes are combined with their variables into little bundles called objects, so the whole thing is portable. In procedural programming, nothing is really bundled, each program can be viewed as a huge and massively complex object. (So, you should be thinking that the more simple and specific an object is, the more portable it is...)

Subroutine : A smaller routine (kinda like an object, but not really...) or mini program that the larger program can call to do a specific job for the overall program. Subroutines and subdivision of program tasks is what gave rise to OOP from procedural programming practices.

Two's complement : [NOTE: Don't really worry with this one just yet. Just remember to come back and re-read this definition when you encounter a specific need to know this. This won't really matter in day-to-day programming. I ganked this definition from http://nhse.npac.syr.edu/hpccgloss/hpccgloss.html. Sorry, I just didn't feel like getting into this one!] A system used in some computers to represent negative numbers in binary. Each bit of the number is inverted (zeros are replaced with ones and vice versa), as for ones complement, but then one (000...0001) is added (ignoring overflow). This avoids the two representations for zero found in ones complement by using all ones to represent -1. (..., 000...00011 = +3, 000...00010 = +2, 000...00001 = +1, 000...00000 = 0, 111...11111 = -1, 111...11110 = -2, 111...11101 = -3, ...) This representation simplifies the logic required for addition and subtraction, at the expense of a little extra complexity for negation.

Variable : A container used to temporarily hold data in the program. You can store numbers, names, and property values in variables. In other words, just like in algebra class, accept instead of just numbers, you can store strings of text, references to other programs (or objects, more specifically), words, characters, etc.

WWW : [NOTE: I ganked this definition from http://nhse.npac.syr.edu/hpccgloss/hpccgloss.html. Sorry, but again, I really didn't feel like defining this one throroughly...] The collection of all the resources (HTML documents, images, and other files, as well as CGI interface programs) accessible on the Internet mainly via HTTP but also via older protocols and mechanisms, such as FTP or Gopher, which are supported by most Web browsers. The emergence of Web browsers has made access to these resources achievable to a broad base of users beyond the more technically savvy traditional users of the Internet who relied on less user-friendly access tools than currently available browsers.

WYSIWYG : What You See Is What You Get. Just a display method in which nothing is hidden, all real results are displayed. A good example would be the visual HTML editors such as DreamWeaver or WebExpress where you can edit a webpage and format it the same way you would do it in MS Word or something similar. It is also a good way to get an annoying girl to shut up and stop complaining...

ZIP : An open standard for compression and decompression used widely for PC download archives. If you haven't seen .zip on Gnutella or the web yet, you haven't been using a computer!

Z : My favorite letter... I don't know why... it just is...

OK, NOW WHAT?

Well, this is as far as I can help you in this little primer. You should know the absolute basics now, and if you are like 99% of the people who have downloaded this from me or wrote me with regards to "The Simple and Effective Way to Learn Java", then you are just waiting to get on with learning Java!

This is where the C, C++, Pascal, Eiffel, and Java people will all part ways. Here are some URLs to get you going and places to learn from. I can only provide a tutorial pattern for Java, as that is the only language I am really familiar with and only programming community I'm really familiar with.

So for the Java people, I would recommend:

1 - **http://www.quiver.freeserve.co.uk/** When you arrive at this website, you will see a number of links on the left of the page. Go find the one near the bottom that says "tutorials". Click on it. Then choose the tutorial that reads "Object Oriented blah blah blah". It is excellent for people with no clue what OOP is all about. There are a few examples in Modula-2, C++ and Java. If you don't know anything about any of these languages, it's ok. Just follow along and read through the complete tutorial. The code (psuedo-English) for these simple examples is mostly simple enough to be understood by anyone. Just pay particular attention to the definitions and general concepts. You can worry about the code later.
2 - **http://javaranch.com/** This is a really funny website dedicated to Java and the Java community. It has lots of good beginner ("greenhorn", hyuk, hyuk...) material and things. It also has a really great message board system that is very supportive. I would definitely recommend registering there and reading all the stuff on the boards that you can understand. Make sure to visit the "campfire" section and read through the tutorial stories and things. They are very educational.
3 - **http://java.sun.com/** This may sound kinda crazy, but just right into the actual Java tutorial provided by Sun (Sun actually designed and produced Java...) and be sure to download and install the Java 2 SDK from that website (or whatever the latest version of Java is when you happen to read this. As of this writing it was Java 1.3.0 in the Java 2 SDK.).
4 - **http://java.ittoolbox.com/** Once you have the basics down, this is a great Java resource.

For the other people who want to just learn lots in general about other languages or whatever, I would recommend:

1 - **http://www.techweb.com/encyclopedia/** If you're interested in tech stuff in general, definitely check this site out. This is good for ANYONE to go visit!
2 - **http://www.zdnet.com/** For your tech news, game news, whatever news, free downloads, product reviews, etc...
3 - **http://www.ittoolbox.com/** Has lots of different specialized sections and things. Very informational, very good. Special sections for the most major languages (notably C/C++, Eiffel and Java)

These are some general links and sites I recommend you take a gander at:

http://nhse.npac.syr.edu/hpccgloss/hpccgloss.html [glossary of Programming Terms...]
http://www.techweb.com/encyclopedia/ [TechWeb tech encyclopaedia]
http://www.ics.uci.edu/~eppstein/junkyard/ [interesting geometry place "Geometry Junkyard"]
http://www.cs.unb.ca/~alopez-o/math-faq/index.html [mathematics FAQs]
http://members.aol.com/jeff570/mathword.html [earliest uses of mathematical terms site]
http://members.aol.com/jeff570/mathsym.html [earliest uses of mathematical symbols]
http://www.research.att.com/~njas/sequences/index.html [integer sequences encycolpaedia]
http://archives.math.utk.edu/materials.html [sites dedicated to teaching mathematics]
http://www.math.com/ [Site dedicated to teaching math, mostly the basics... which is what you need when you do programming...]
http://www.bearshare.net/ [BearShare forums, etc. for the segment of the Gnutella file-sharing community that uses BearShare. Cool stuff, and a great Gnutella app.]

If you have any comments, questions, suggestions, bitches, gripes, free money to give me, naked women who want me, etc. please feel free to drop me a line at:

shaveyoursnatch@math.com

This has been a production of $h@v3-uR-$n@+cH. This work is hereby released by him to the Public Domain (not that you would give a damn if he (c)'ed it anyway...) on 4.2.2001 from Ishikiri, Japan.

[DISCLAIMER: The views expressed here are merely the opinion of the author ($h@v3-uR-$n@+cH) and do not necessarily represent the official views or opinions of the rest of the Bald Beaver loving population.]

[NOTE: All URLs were validated as of 4.2.2001.]

[Additional – you can find “The Simple and Effective Way to Learn Java” by $h@v3-uR-$n@+cH on the Gnutella network in pdf format as well.]

[This primer will be mirrored at **http://www.fuka.250x.com/comp/progprime.htm** from 4.3.2001.]