Type safety and strong versus weak typing

Those whose first language is Python, Ruby, Perl, or Matlab may find Java wordy or cumbersome because every variable has to be declared (with its type) before it is used, including parameters. Even the return type must be given for a function. Java is more *strongly typed* than these others, and these others are more *weakly typed* than Java. The terms *strongly* and *weakly typed* are not well defined, and we will discuss them later in this document. But first let's discuss the notion of *type safety* and give a little history.

Type safety

Java was designed to be *type safe*. By that we mean that a program cannot access any part of memory inappropriately. It had to be type safe, since Java programs, called applets could be run from a webpage on any computer, by anyone, anywhere. If a language is not type safe, not only can bugs more easily creep into the programs but security can be a big issue.

One of the most prevalent bugs that has been exploited by hackers is the *buffer overflow* or *buffer overrun*, in which one can store a value in element n+1 of an array (or buffer) when only n elements are allocated for the array. By doing this, knowing the layout of memory, one can overwrite a known piece of data or even executable code. Google "wiki: buffer overrun" to learn more.

Ensuring type safety: syntactic type checking

Java ensures type safety by defining syntactic type rules. Each variable has to be declared with a type before it is used, and the type of each expression (and sub-expression) can be determined from the syntax of a program, that is, at compile-time. Types and type rules are part of the syntax of the language. Just by looking at a program and its structure, without executing it, one can tell whether the program is type correct. For example, try to halve a string using "bcd"/2 in a program and the program won't compile. It is syntactically incorrect. Further, all index references like b[i] and s.charAt(i) are checked at runtime to be sure that index I is in bounds.

Generally, the sooner an error is detected, the better. Detecting an error at compile-time —when a program is being translated into the machine language for execution— is better than detecting it after the program is compiled and while it is being executed. The larger a program, or the team of people writing, developing, an debugging it, the more important it is to find errors as early as possible.

The first worm: what happens when buffer overflows and other non type-safe holes exist

In November 1988, Robert Morris, a graduate student in CS at Cornell, released a *worm*. The worm got into one computer, then used that computer to get into more, and on it went. In a matter of minutes, the worm infected some 6,000 major UNIX machines in a deadly way, one tenth of the estimated 60,000 that were on the internet at the time. It brought down the internet. Of course, the internet is not what it is today. There were no browsers, no websites! The internet was use mainly for communication by email and such.

The buffer overflow was one of the 4-5 faults Morris exploited in writing his worm.

Morris wasn't malicious. He was just experimenting, wondering how to estimate the size of the internet, and he thought his worm could do it unobtrusively. But he miscalculated, and it brought down the internet.

If this wasn't the first worm, it was the first to received nationwide attention. Morris became the first person convicted under the new Computer Fraud and Abuse Act. He served no jail time but did community service and paid a fine. He went on to become a professor at MIT, tenured in 2006.

Juris Hartmanis (first chair of CS at Cornell and father of the field of "computational complexity") and David Gries were on an *ad hoc* commission convened by Cornell to investigate the issue. Read a summary of that report here: cs.cornell.edu/courses/cs2110/2017fa/online/files/CornellInvestMorris.pdf. You can also read more about Morris and the worm in these two places:

https://en.wikipedia.org/wiki/Robert_Tappan_Morris
Morris's appeal of conviction: https://scholar.google.com/scholar_case?case=551386241451639668

We tell this story to give you a sense of history and to caution you: make sure what you do is both legal and ethical. Think about how things you do might affect others around you.

For this reason, we call Java a *strongly typed* language.

Python and Matlab do not have types for variables. Your program can store a double value in a variable m; later, it can store a string, or an array, or anything else. They do little type checking at compile-time. The type of an expression is not a syntactic property, as it is in Java. Therefore, we say that these languages are *weakly typed* compared to Java.

But Python does try for *type safety* by performing type checks as runtime.. The term *duck typing* has been used for the type checking done by Python at runtime: "If it walks like a duck and it quacks like a duck, then it must be a duck."

Some people call Java's type-checking *static type checking* while Python's is *dynamic type checking*. We would prefer the terms are *syntactic type checking* and *semantic type checking*.

The following website talks about Python and type safety,

https://beam.apache.org/documentation/sdks/python-type-safety/

It says that, "the deferred nature of runner execution, developer productivity can easily become bottle-necked by time spent investigating type-related errors." This is what they mean: Suppose some error occurs at runtime that is related to a variable's value being used in an appropriate way —like dividing a string value by 2. An error message appears. The programmer has to find the source of the error —which may be far from the point of *detection* of the error— and this can take a great deal of time. But if type-checking was done at compile-time, the point of detection of the error might have been obvious without even running the program.

The above-mentioned website talks about allowing a programmer to provide "type hints" to help find such errors earlier. We don't go into detail on this but just want you to know that the issue of types, type checking, and how to maintain type safety is still an interesting issue, with many different avenues to approach it.

Language that are not type safe

The language C is not type safe. It was initially developed in order to have a language in which to write the operating system UNIX —just as a research tool at Bell Labs. As such, it *had* to allow the ability to look at and change specific machine locations. It could not be type safe.

Finally, any assembly language, which is a symbolic representation of a machine language, is inherently not type safe. Essentially, anything can be stored in any memory location, and the contents of a memory location can be interpreted in any way one wants.

Exploiting a buffer overflow bug in your software to lock out a competitor

This little episode happened in 1999. Read about it here:

www.cnn.com/TECH/computing/9908/20/aolbug.idg/index.html

AOL's Instant Messenger (AIM) service was in competition with Microsoft's new MSN Messenger Service. There was a buffer overflow error in AIM. The buffer was 256 bytes. When an AOL client logged onto Instant Messenger, the client actually sent back 256 + 24 bytes —an overflow. But when a Microsoft Messaging client logged in, it sent only 256 bytes. So the AOL server could identify Microsoft clients and block them.

The webpage listed above says that Robert Graham, chief technical officer of Network ICE, an independent intrusion detection and security company, uncovered this buffer overflow bug and how AOL was using it.