LECTURE 1

INTRODUCTION TO JAVA

Akademia Finansów i Biznesu Vistula
Studia globalnych możliwości

mgr inż. Dominik Bielecki
email: d.bielecki@vistula.edu.pl

HISTORY OF JAVA - PRELUDE

Before we can fully appreciate the unique aspects of Java, it is necessary to understand the forces that drove to its creation, the programming philosophy that it embodies, and key concepts of its design. As we advance through lectures and exercises, you will see that many aspects of Java are either a direct or indirect results of the historical forces that impacted the language. Thus, it is appropriate to begin our analysis of Java by exploring how Java relates to the larger programming universe.

HISTORY OF JAVA (1)

Java was created by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems in 1991. This language was initially called “Oak” but was renamed “Java” in 1995. Somewhat surprisingly, the original impulse for Java was not the Internet! Instead, the primary motivation was the need for a platform independent language that could be used to create software to be embedded in various consumer electronic devices, such as toasters, microwave ovens, and remote controls. As you can probably guess, many different types of CPUs are used as controllers. The trouble was that (at the time) most computer languages were designed to be compiled into machine code that was targeted for a specific type of CPU. For example, consider the C++ language.

HISTORY OF JAVA (2)

Although it was possible to compile a program in C++ for almost any type of CPU, this required a full C++ compiler targeted for that CPU. The problem, however, is that compilers are expensive and time consuming to create. In an attempt to find a better solution, Gosling and the others worked on a portable, crossplatform language that could produce code that would run on a variety of CPUs under differing environments. This effort ultimately led to the creation of Java. **Question**: What does **WORE** stand for?

HISTORY OF JAVA VS HISTORY OF INTERNET

About the time that the details of Java were being worked out, a second, and ultimately more important, factor emerged that would play a crucial role in the future of Java. This second force was, of course, the World Wide Web. Had the Web not taken shape at about the same time that Java was being implemented, Java might have remained a useful but unnoticed language for programming consumer electronics. However, with the emergence of the Web, Java was propelled to the forefront of computer language design, because the Web, too, demanded portable programs.

While the quest for a way to create efficient, portable (platform independent) programs is nearly as old as the discipline of programming itself, it had taken a back seat to other, more pressing problems. However, with the advent of the Internet and the Web, the old problem of portability returned. After all, the Internet consisted of a diverse, distributed universe populated with many types of computers, operating systems, and CPUs.

HISTORY OF JAVA VS HISTORY OF INTERNET

What was once an irritating but low priority problem had become a high-profile necessity. By 1993 it became obvious to members of the Java design team that the problems of portability frequently encountered when creating code for embedded controllers are also found when attempting to create code for the Internet. This realization caused the focus of Java to switch from consumer electronics to Internet programming. So, although it was the desire for an architecture neutral programming language that provided the initial spark, it was the Internet that ultimately led to Java's largescale success.

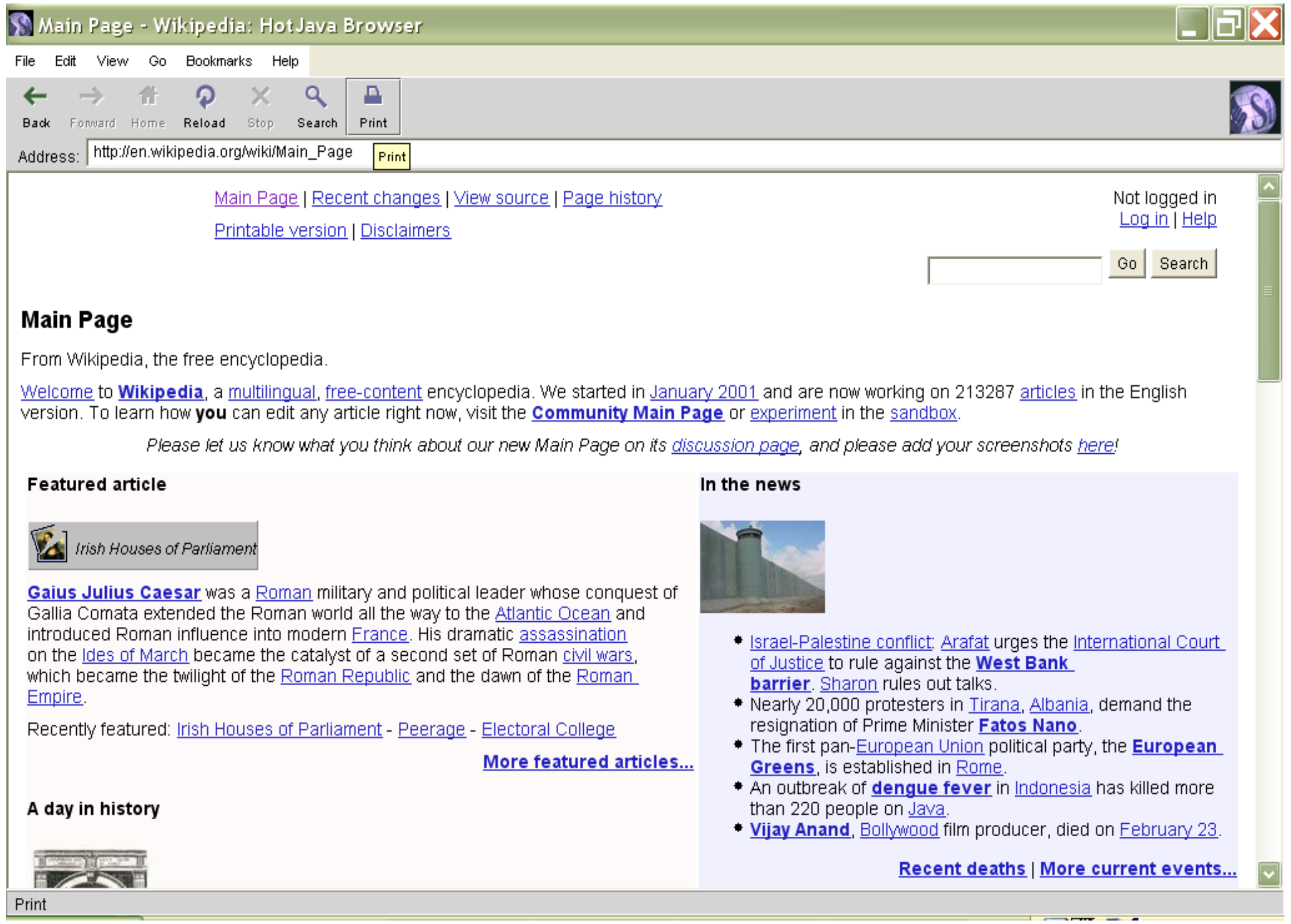
HISTORY OF JAVA AND ITS ANCESTORS

The two languages that form Java’s closest ancestors are C and C++. As you may know, C and C++ are among the most important computer languages ever invented and are still in widespread use today. From C, Java inherits its syntax. Java’s object model is adapted from C++. Java’s relationship to C and C++ is important for a number of reasons. First, at the time of Java’s creation, many programmers were familiar with the C/C++ syntax. Because Java uses a similar syntax, it was relatively easy for a C/C++ programmer to learn Java.

Second, Java’s designers did not “reinvent the wheel.” Instead, they further refined an already highly successful programming paradigm. The modern age of programming began with C. It moved to C++ and then to Java. By inheriting and building on that rich heritage, Java provides a powerful, logically consistent programming environment that takes the best of the past and adds new features related to the online environment and advances in the art of programming.

HOW JAVA IMPACTED THE INTERNET

The Internet helped catapult Java to the forefront of programming, and Java, in turn, had a profound effect on the Internet. First, the creation of Java simplified Internet programming in general, acting as a catalyst that drew legions of programmers to the Web. Second, Java innovated a new type of networked program called the **applet** that changed the way the online world thought about content. Finally, and perhaps most importantly, Java addressed some of the thorniest issues associated with the Internet: **portability and security.**


Main Page - Wikipedia: HotJava Browser
File Edit View Go Bookmarks Help
Back Forward Home Reload Stop Search Print
Address: http://en.wikipedia.org/wiki/Main_Page
Print
Main Page | Recent changes | View source | Page history
Printable version | Disclaimers
Not logged in
Log in | Help
Go Search
Print

Main Page

From Wikipedia, the free encyclopedia.

Welcome to **Wikipedia**, a multilingual, free-content encyclopedia. We started in January 2001 and are now working on 213287 articles in the English version. To learn how **you** can edit any article right now, visit the **Community Main Page** or experiment in the sandbox.

Please let us know what you think about our new Main Page on its discussion page, and please add your screenshots here!

Featured article

Irish Houses of Parliament

Gaius Julius Caesar was a Roman military and political leader whose conquest of Gallia Comata extended the Roman world all the way to the Atlantic Ocean and introduced Roman influence into modern France. His dramatic assassination on the Ides of March became the catalyst of a second set of Roman civil wars, which became the twilight of the Roman Republic and the dawn of the Roman Empire.

Recently featured: Irish Houses of Parliament - Peerage - Electoral College

More featured articles...

A day in history

In the news

- Israel-Palestine conflict: Arafat urges the International Court of Justice to rule against the **West Bank barrier**. Sharon rules out talks.
- Nearly 20,000 protesters in Tirana, Albania, demand the resignation of Prime Minister **Fatos Nano**.
- The first pan-European Union political party, the **European Greens**, is established in Rome.
- An outbreak of **dengue fever** in Indonesia has killed more than 220 people on Java.
- **Vijay Anand**, Bollywood film producer, died on February 23.

Recent deaths | **More current events...**

HOW JAVA IMPACTED THE INTERNET

From the start, Java simplified web-based programming in a number of ways. Arguably the most important is found in its ability to create portable, cross platform programs. Of nearly equal importance is Java’s support for networking. Its library of ready-to-use functionality enabled programmers to easily write programs that accessed or made use of the Internet. It also provided mechanisms that enabled programs to be readily delivered over the Internet. It is important to know that Java’s support for networking was a key factor in its rapid rise.

HOW JAVA IMPACTED THE INTERNET - FEW WORDS ABOUT JAVA APPLETS (1)

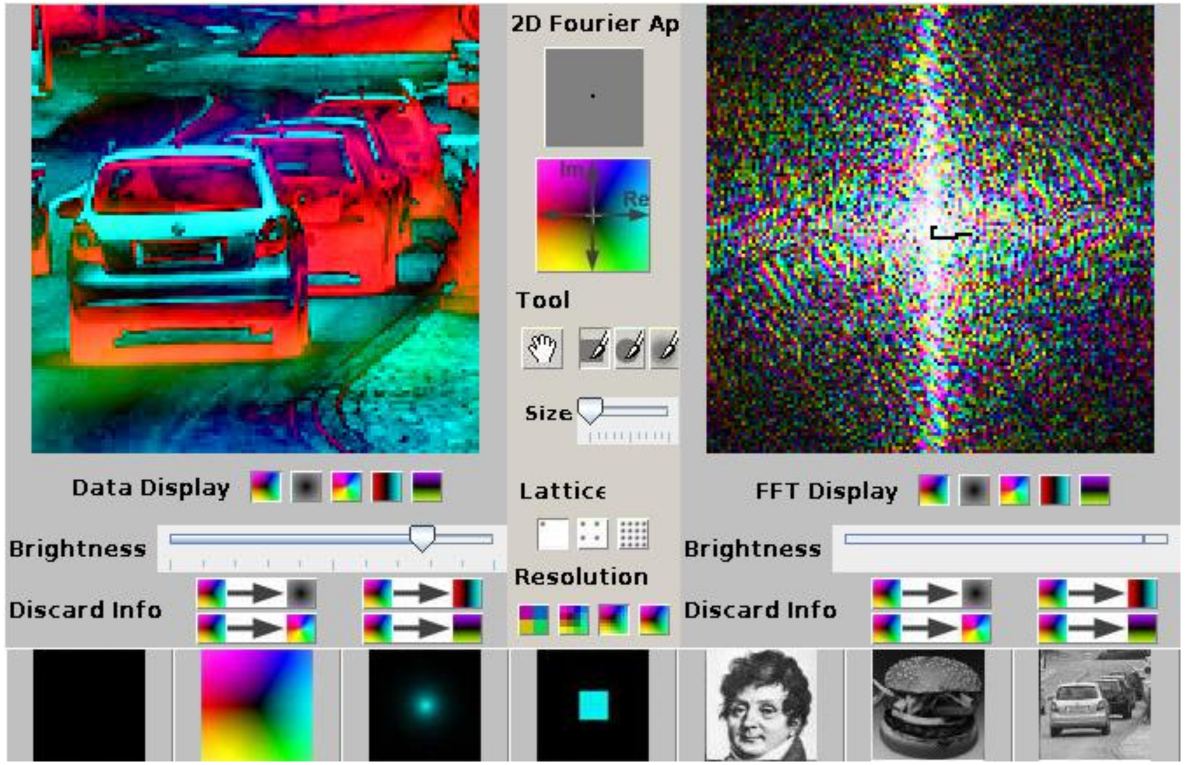
At the time of Java's creation, one of its most exciting features was the applet. An applet is a special kind of Java program that is designed to be transmitted over the Internet and automatically executed inside a Java-compatible web browser. If the user clicks a link that contains an applet, the applet will download and run in the browser automatically. Applets were intended to be small programs, typically used to display data provided by the server, handle user input, or provide simple functions, such as a loan calculator. The key feature of applets is that they execute locally, rather than on the server. In essence, the applet allowed some functionality to be moved from the server to the client.

HOW JAVA IMPACTED THE INTERNET - FEW WORDS ABOUT JAVA APPLETS (2)

The creation of the applet was important because, at the time, it expanded the universe of objects that could move about freely in cyberspace. In general, there are two very broad categories of objects that are transmitted between the server and the client: passive information and dynamic active programs. For example, when you read your email, you are viewing passive data. Even when you download a program, the program's code is still only passive data until you execute it. By contrast, the applet is a dynamic, self-executing program. Such a program is an active agent on the client computer, yet it is delivered by the server.

HOW JAVA IMPACTED THE INTERNET - FEW WORDS ABOUT JAVA APPLETS (3)

In the early days of Java, applets were a crucial part of Java programming. They illustrated the power and benefits of Java, added an exciting dimension to web pages, and enabled programmers to explore the full extent of what was possible with Java. Although it is likely that there are still applets in use today, **over time they became less important**, JDK 9 began their phaseout process. Finally, applet support was removed by JDK 11.



INTERNET AND JAVA SOLUTION FOR SECURITY PROBLEM

As desirable as dynamic, networked programs are, they also present serious problems in the areas of security and portability. Obviously, a program that downloads and executes on the client computer must be prevented from doing harm. It must also be able to run in a variety of different environments and under different operating systems. Java addressed these problems in an effective and elegant way.

As you are likely aware, every time that you download a program, you are taking a risk because the code you are downloading might contain a virus, Trojan horse, or other harmful code. At the core of the problem is the fact that malicious code can cause damage because it has gained unauthorized access to system resources. For example, a virus program might gather private information, such as credit card numbers, bank account balances, and passwords, by searching the contents of your computer's local file system. In order for Java to enable programs to be safely downloaded and executed on the client computer, it was necessary to prevent them from launching such an attack.

Java achieved this protection by enabling you to confine an application to the Java execution environment and prevent it from accessing other parts of the computer. The ability to download an application with a high level of confidence that no harm will be done contributed significantly to Java's early success.

INTERNET AND JAVA SOLUTION FOR PORTABILITY PROBLEM

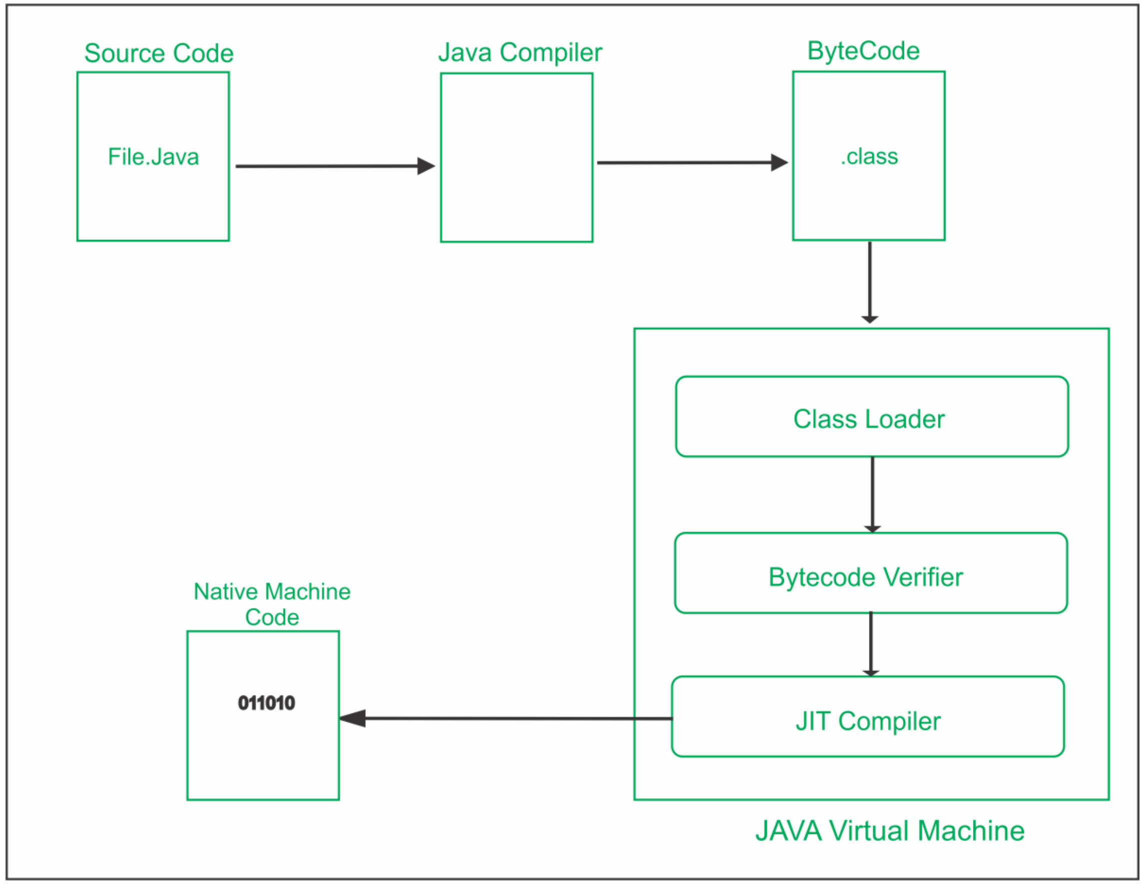
Portability is a major aspect of the Internet because there are many different types of computers and operating systems connected to it. If a Java program were to be run on virtually any computer connected to the Internet, there needed to be some way to enable that program to execute on different types of systems. In other words, a mechanism that allows the same application to be downloaded and executed by a wide variety of CPUs, operating systems, and browsers is required. It is not practical to have different versions of the same application for different computers. The same application code must work in all computers. Therefore, some means of generating portable code was needed. As you will soon see, the same mechanism that helps ensure security also helps create portability.

JAVA KEY TO SUCCESS: THE BYTECODE (1)

The key that allowed Java to address both the security and the portability problems just described is that the output of a Java compiler is not executable code. Rather, it is bytecode. Bytecode is a highly optimized set of instructions designed to be executed by what is called the Java Virtual Machine (JVM), which is part of the Java Runtime Environment (JRE). In essence, the original JVM was designed as an interpreter for bytecode. This may come as a bit of a surprise because many modern languages are designed to be compiled into CPU specific, executable code due to performance concerns. However, the fact that a Java program is executed by the JVM helps solve the major problems associated with web based programs.

JAVA KEY TO SUCCESS: THE BYTECODE (2)

Translating a Java program into bytecode makes it much easier to run a program in a wide variety of environments because only the JRE (which includes the JVM) needs to be implemented for each platform. Once a JRE exists for a given system, any Java program can run on it. Remember, although the details of the JRE will differ from platform to platform, all JREs understand the same Java bytecode. If a Java program were compiled to native code, then different versions of the same program would have to exist for each type of CPU connected to the Internet. This is, of course, not a feasible solution. Thus, the execution of bytecode by the JVM is the easiest way to create truly portable programs.



JAVA KEY TO SUCCESS: THE BYTECODE (3)

The fact that a Java program is executed by the JVM also helps to make it secure. Because the JVM is in control, it manages program execution. Thus, it is possible for the JVM to create a restricted execution environment, called the sandbox, that contains the program, preventing unrestricted access to the machine. Safety is also enhanced by certain restrictions that exist in the Java language.

When a program is interpreted, it generally runs slower than the same program would run if compiled to executable code. However, with Java, the differential between the two is not so great. Because bytecode has been highly optimized, the use of bytecode enables the JVM to execute programs much faster than we might expect.

JAVA KEY TO SUCCESS: THE BYTECODE (4)

Although Java was designed as an interpreted language, there is nothing about Java that prevents on-the-fly compilation of bytecode into native code in order to boost performance. For this reason, the HotSpot JVM was introduced not long after Java’s initial release. HotSpot includes a just-in-time (JIT) compiler for bytecode. When a JIT compiler is part of the JVM, selected portions of bytecode are compiled into executable code in real time on a piece-by-piece demand basis. That is, a JIT compiler compiles code as it is needed during execution. Furthermore, not all sequences of bytecode are compiled—only those that will benefit from compilation. The remaining code is simply interpreted. However, the just-in-time approach still yields a significant performance boost. Even when dynamic compilation is applied to bytecode, the portability and safety features still apply because the JVM is still in charge of the execution environment. One other point: Beginning with JDK 9, some Java environments will also support an ahead of-time compiler that can be used to compile bytecode into native code prior to execution by the JVM, rather than on-the-fly. Ahead-of-time compilation is a specialized feature and it does not replace Java’s traditional approach just described. Because of the highly sophisticated nature of Ahead-of-time compilation, it is not something that we will use when learning Java during our classes.

JAVA APPLETS

It has been three decades since Java’s original release. Over those years, many changes have taken place. At the time of Java’s creation, the Internet was a new and exciting innovation; web browsers were undergoing rapid development and refinement; the modern form of the smartphone had not yet been invented; and the near omnipresent use of computers was still a few years off. As we would expect, Java has also changed and so, too, has the way that Java is used. Perhaps nothing illustrates the ongoing evolution of Java better than the applet.

As explained previously, in the early years of Java, applets were a crucial part of Java programming. They not only added excitement to a web page, they were a highly visible part of Java, which added to its charisma. However, applets rely on a Java browser plug-in. Thus, for an applet to work, it must be supported by the browser. Recently, support for the Java browser plugin has been vanishing. Simply put, without browser support, applets are not practicable. Because of this, beginning with JDK 9, the phaseout of applets was begun, with support for applets being deprecated. In the language of Java, deprecated means that a feature is still available but flagged as obsolete. Thus, a deprecated feature should not be used for new code. The phaseout became complete with the release of JDK 11 because support for applets was removed.

A FASTER RELEASE SCHEDULE

Another major change has recently occurred in Java, but it does not involve changes to the language or the runtime environment. Rather, it relates to the way that Java releases are scheduled. In the past, major Java releases were typically separated by two or more years. However, subsequent to the release of JDK 9, the time between major Java releases has been decreased. Today, it is anticipated that a major release will occur on a strict time-based schedule, with the expected time between major releases being just six months.

Each major release, now called a feature release, will include those features ready at the time of the release. This increased release cadence enables new features and enhancements to be available to Java programmers in a timely fashion. Furthermore, it allows Java to respond quickly to the demands of an everchanging programming environment. Simply put, the faster release schedule promises to be a very positive development for Java programmers.

THE JAVA IMPORTANT WORDS

Word	Meaning
Simple	Java has a brief, compact set of features that makes it easy to learn and use.
Secure	Java provides a secure means of creating Internet applications.
Portable	Java programs can exectue in any environment for which there is a Java run-time system.
Object oriented	Java embodies the modern object-oriented programming philosophy.
Robust	Java encourages error-free programing by being strictly typed and performing run-time checks.
Multithreaded	Java provides integrated suport for multithreaded programming.
Architecture-neutral	Java is not tied to a specific machine or operating system architecture.
Interpreted	Java supports cross-platform code through the use of Java bytecode.
High performance	The Java bytecode is higly optimized for speed of execution.
Distributed	Java was designed with distributed environment of the Internet in mind.
Dynamic	Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve access to objects at run time.

OBJECT-ORIENTED PROGRAMMING (1)

At the center of Java is object-oriented programming (OOP). The object oriented methodology is inseparable from Java, and all Java programs are, to at least some extent, object-oriented. Because of OOP's importance to Java, it is useful to understand in a general way OOP's basic principles before you write even a simple Java program. Later during our lectures, you will see how to put these concepts into practice.

OOP is a powerful way to approach the job of programming. Programming methodologies have changed dramatically since the invention of the computer, primarily to accommodate the increasing complexity of programs. For example, when computers were first invented, programming was done by toggling in the binary machine instructions using the computer's front panel. As long as programs were just a few hundred instructions long, this approach worked. As programs grew, assembly language was invented so that a programmer could deal with larger, increasingly complex programs, using symbolic representations of the machine instructions. As programs continued to grow, high-level languages were introduced that gave the programmer more tools with which to handle complexity. The first widespread language was, of course, FORTRAN. Although FORTRAN was a very impressive first step, it was hardly a language that encouraged clear, easy-to-understand programs.

OBJECT-ORIENTED PROGRAMMING (2)

The 1960s gave birth to structured programming. This is the method encouraged by languages such as C and Pascal. The use of structured languages made it possible to write moderately complex programs fairly easily. Structured languages are characterized by their support for standalone subroutines, local variables, rich control constructs, and their lack of reliance upon the GOTO. Although structured languages are a powerful tool, even they reach their limit when a project becomes too large.

At each milestone in the development of programming, techniques and tools were created to allow the programmer to deal with increasingly greater complexity. Each step of the way, the new approach took the best elements of the previous methods and moved forward. Prior to the invention of OOP, many projects were nearing (or exceeding) the point where the structured approach no longer works. Object-oriented methods were created to help programmers break through these barriers.

OBJECT-ORIENTED PROGRAMMING (3)

Object-oriented programming took the best ideas of structured programming and combined them with several new concepts. The result was a different way of organizing a program. In the most general sense, a program can be organized in one of two ways: around its code (what is happening) or around its data (what is being affected). Using only structured programming techniques, programs are typically organized around code. This approach can be thought of as “code acting on data.”

Object-oriented programs work the other way around. They are organized around data, with the key principle being “data controlling access to code.” In an object-oriented language, you define the data and the routines that are permitted to act on that data. Thus, a data type defines precisely what sort of operations can be applied to that data. To support the principles of object-oriented programming, all OOP languages, including Java, have three traits in common: **encapsulation, polymorphism, and inheritance.** I will now describe each of them.

ENCAPSULATION (1)

Encapsulation is a programming mechanism that binds together code and the data it manipulates, and that keeps both safe from outside interference and misuse. In an object-oriented language, code and data can be bound together in such a way that a self-contained black box is created. Within the box are all necessary data and code. When code and data are linked together in this fashion, an object is created. In other words, an object is the device that supports encapsulation.

Within an object, code, data, or both may be private to that object or public. Private code or data is known to and accessible by only another part of the object. That is, private code or data cannot be accessed by a piece of the program that exists outside the object. When code or data is public, other parts of your program can access it even though it is defined within an object. Typically, the public parts of an object are used to provide a controlled interface to the private elements of the object.

ENCAPSULATION (2)

Java’s basic unit of encapsulation is the **class**. Although the class will be examined in detail later during our lectures, the following brief discussion will be helpful now. A class defines the form of an object. It specifies both the data and the code that will operate on that data. Java uses a class specification to construct objects. Objects are instances of a class. Thus, a class is essentially a set of plans that specify how to build an object.

The code and data that constitute a class are called members of the class. Specifically, the data defined by the class are referred to as member variables or instance variables. The code that operates on that data is referred to as member methods or just methods.

POLYMORPHISM (1)

Polymorphism (from Greek, meaning "many forms") is the quality that allows one interface to access a general class of actions. The specific action is determined by the exact nature of the situation. A simple example of polymorphism is found in the steering wheel of an automobile. The steering wheel (i.e., the interface) is the same no matter what type of actual steering mechanism is used. That is, the steering wheel works the same whether your car has manual steering, power steering, or rack-and-pinion steering. Therefore, once you know how to operate the steering wheel, you can drive any type of car.

The same principle can also apply to programming. For example, consider a stack (which is a first-in, last-out list). You might have a program that requires three different types of stacks. One stack is used for integer values, one for floating-point values, and one for characters. In this case, the algorithm that implements each stack is the same, even though the data being stored differs. In a non-object-oriented language, you would be required to create three different sets of stack routines, with each set using different names. However, because of polymorphism, in Java you can create one general set of stack routines that works for all three specific situations. This way, once you know how to use one stack, you can use them all.

POLYMORPHISM (2)

More generally, the concept of polymorphism is often expressed by the phrase “one interface, multiple methods.” This means that it is possible to design a generic interface to a group of related activities. Polymorphism helps reduce complexity by allowing the same interface to be used to specify a general class of action. It is the compiler’s job to select the specific action (i.e., method) as it applies to each situation. You, the programmer, don’t need to do this selection manually. You need only remember and utilize the general interface.

INHERITANCE (1)

Inheritance is the process by which one object can acquire the properties of another object. This is important because it supports the concept of hierarchical classification. If you think about it, most knowledge is made manageable by hierarchical (i.e., top-down) classifications. For example, a Red Delicious apple is part of the classification apple, which in turn is part of the fruit class, which is under the larger class food. That is, the food class possesses certain qualities (edible, nutritious, etc.) which also, logically, apply to its subclass, fruit. In addition to these qualities, the fruit class has specific characteristics (juicy, sweet, etc.) that distinguish it from other food. The apple class defines those qualities specific to an apple (grows on trees, not tropical, etc.). A Red Delicious apple would, in turn, inherit all the qualities of all preceding classes, and would define only those qualities that make it unique.

INHERITANCE (2)

Without the use of hierarchies, each object would have to explicitly define all of its characteristics. Using inheritance, an object need only define those qualities that make it unique within its class. It can inherit its general attributes from its **parent**. Thus, it is the inheritance mechanism that makes it possible for one object to be a specific instance of a more general case.

JAVA VS. C#

A few years after the creation of Java, Microsoft developed the C# language. This is important because C# is closely related to Java. In fact, many of C#'s features directly parallel Java. Both Java and C# share the same general C++style syntax, support distributed programming, and utilize a similar object model. There are, of course, differences between Java and C#, but the overall "look and feel" of these languages is very similar. This means that if you already know C#, then learning Java will be especially easy. Conversely, if C# is in your future, then your knowledge of Java will come in handy.

SUMMARY

In computing, few technologies have had the impact of Java. Its creation in the early days of the Web helped shape the modern form of the Internet, including both the client and server sides. Its innovative features advanced the art and science of programming, setting a new standard in computer language design. The forward-thinking culture that grew up around Java ensured it would remain vibrant and alive, adapting to the often rapid and varied changes in the computing landscape. Simply put: not only is Java one of the world's most important computer languages, it is a force that revolutionized programming and, in the process, changed the world.

Although Java is a language often associated with Internet programming, it is by no means limited in that regard. Java is a powerful, full-featured, general-purpose programming language. Java is an excellent language to learn.

THANK YOU

Więcej na:

www.vistula.edu.pl

Akademia Finansów i Biznesu Vistula
ul. Stokłosy 3
02-787 Warszawa
(obok stacji metro Stokłosy)