0-language-and-features

Contents

Java

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let programmers write once, run anywhere (`WORA`), meaning that compiled Java code can run on all platforms that support Java without the need to recompile. Java applications are typically compiled to `bytecode` that can run on any Java virtual machine (`JVM`) regardless of the underlying computer architecture. The syntax of Java is similar to C and C++, but has fewer low-level facilities than either of them.

The History

Java was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems, Inc. in 1991. It took 18 months to develop the first working version. This language was initially called "Oak," but was renamed "Java" in 1995. Between the initial implementation of Oak in the fall of 1992 and the public announcement of Java in the spring of 1995

Instead, the primary motivation was the need for a platform-independent (that is, architecture-neutral) language that could be used to create software to be embedded in various consumer electronic devices, such as microwave ovens and remote controls. As you can probably guess, many different types of CPUs are used as controllers. The trouble with C and C++ (and most other languages) is that they are designed to be compiled for a specific target.

Changing the world

Applets - An applet is a special kind of Java program that is designed to be transmitted over the Internet and automatically executed by a Java-compatible web browser. Furthermore, an applet is downloaded on demand, without further interaction with the user. If the user clicks a link that contains an applet, the applet will be automatically downloaded and run in the browser. Applets are intended to be small programs. They are typically used to display data provided by the server, handle user input, or provide simple functions, such as a loan calculator, that execute locally, rather than on the server. In essence, the applet allows some functionality to be moved from the server to the client.

Servlets - Java on the Server Side As useful as applets can be, they are just one half of the client/server equation. Not long after the initial release of Java, it became obvious that Java would also be useful on the server side. The result was the servlet. A servlet is a small program that executes on the server. Just as applets dynamically extend the functionality of a web browser, servlets dynamically extend the functionality of a web server. Thus, with the advent of the servlet, Java spanned both sides of the client/server connection.

Security - Java achieved this protection by confining an applet to the Java execution environment and not allowing it access to other parts of the computer. (You will see how this is accomplished shortly.) The ability to download applets with confidence that no harm will be done and that no security will be breached may have been the single most innovative aspect of Java

Portability - 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 systems.

Byte code magic

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 the Java run-time system, which is called the Java Virtual Machine (JVM). In essence, the original JVM was designed as an interpreter for bytecode. This may come as a bit of a surprise since many modern languages are designed to be compiled into executable code because of performance concerns. However, the fact that a Java program is executed by the JVM helps solve the major problems associated with web-based programs. Here is why.

Translating a Java program into bytecode makes it much easier to run a program in a wide variety of environments because only the JVM needs to be implemented for each platform. Once the run-time package exists for a given system, any Java program can run on it. Remember, although the details of the JVM will differ from platform to platform, all 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.

The fact that a Java program is executed by the JVM also helps to make it secure.

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 technology was introduced not long after Java's initial release. HotSpot provides 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. It is important to understand that it is not practical to compile an entire Java program into executable code all at once, because Java performs various run-time checks that can be done only at run time. Instead, a JIT compiler compiles code as it is needed, during execution.

Core concepts

- Secure
- Simple
- Portable
- Object-oriented
- Robust
- Multithreaded
- Architecture-neutral
- Interpreted
- High performance
- Distributed
- Dynamic

Java Versions

Here is a brief synopses of the major features in each version of java, the release of java 5 is what would mark the language's powers in the new age of programming

- Java 1.0 (1996) - Initial release of Java. Introduced basic language features, AWT (Abstract Window Toolkit), Applets, JVM, Garbage Collection, and multithreading support.
- Java 1.1 (1997) - Significant update with event handling model (delegation event model), inner classes, JavaBeans, JDBC (Java Database Connectivity), RMI (Remote Method Invocation), and deprecation of some methods from 1.0.
- Java 1.2 (Java 2) (1998) - Major update marking the beginning of "modern Java". Introduced Swing GUI toolkit, Collections Framework, JIT Compiler integration into JVM, Java Plug-in, and Security Enhancements.
- Java 1.3 (2000) - Performance improvements, HotSpot JVM became the default, and added JavaSound API. Minor updates to collections and networking.

- Java 1.4 (2002) - Introduced assertions, regular expressions, NIO (New Input/Output), Java Web Start, logging API, and XML parsing. Focused on better core language support and more robust I/O.
- Java 1.5 (Java 5) (2004) - Introduced Generics, enhanced for loop, autoboxing/unboxing, varargs, annotations, enumerations, and the concurrency utilities package. Renamed as Java 5.0 to reflect major updates.
- Java 1.6 (Java 6) (2006) - Focused on performance improvements, added Scripting API, Compiler API, Web Services support (JAX-WS), and enhancements to JDBC 4.0.
- Java 1.7 (Java 7) (2011) - Introduced try-with-resources for better exception handling, diamond operator (<>), NIO.2 (enhanced file I/O), Fork/Join framework, switch statements with strings, and improvements to the JVM (invokedynamic bytecode).

With the very important release of Java J2SE 5, it was revolutionary. Unlike most of the previous Java upgrades, which offered important, but measured improvements, J2SE 5 fundamentally expanded the scope, power, and range of the language.

The importance of these new features is reflected in the use of the version number "5." The next version number for Java would normally have been 1.5. However, the new features were so significant that a shift from 1.4 to 1.5 just didn't seem to express the magnitude of the change. Instead, Sun elected to increase the version number to 5 as a way of emphasizing that a major event was taking place. Thus, it was named J2SE 5, and the developer's kit was called JDK 5. However, in order to maintain consistency, Sun decided to use 1.5 as its internal version number, which is also referred to as the developer version number. The "5" in J2SE 5 is called the product version number.

This marks a significant milestone in the development of the language and it's runtime, from then on the major version will be kept as a Java SE 5,6,7 etc. But the development version number format of 1._release_ would be kept up until the release of Java 8 as Java SE 1.8

The Java SE 7 was the first major release of Java since Sun Microsystems was acquired by Oracle. Java SE 7 made several additions to the Java API library. Two of the most important were the enhancements to the NIO Framework and the addition of the Fork/Join Framework. NIO (which originally stood for New I/O) was added to Java in version 1.4. However, the changes added by Java SE 7 fundamentally expanded its capabilities. So significant were the changes, that the term NIO.2 is often used.

Java Features

Java Beans

Set of conventions for creating reusable software components in Java. They are used to encapsulate many objects into a single object (the bean)

- Properties: JavaBeans use properties to store and retrieve data. These properties are accessed through getter and setter methods. For example:
- Default Constructor: JavaBeans must have a no-argument (default) constructor, which allows for easy instantiation without needing to provide parameters.
- Serializable: JavaBeans should implement the Serializable interface, which allows the bean to be easily serialized and deserialized. This is useful for storing or transmitting bean instances.
- Getter and Setter Methods: JavaBeans follow a naming convention for accessor methods. Getter methods retrieve property values and start with get followed by the property name (e.g., getName),

AWT (Abstract Window Toolkit)

AWT is the original Java GUI toolkit introduced in JDK 1.0. It provides a set of native, platform-dependent GUI components.

- Components: Includes basic components such as buttons, checkboxes, labels, text fields, and more.
- Rendering: AWT components are rendered by the native OS, which means they look like native components on different platforms.

Swing (Swing beyond AWT)

Swing is a more advanced GUI toolkit introduced in JDK 1.2 as part of Java 2. It builds on AWT but provides a richer set of components.

- Components: Includes enhanced components like JButton, JLabel, JTable, JTree, JList, and many more, offering greater functionality and customization.
- Rendering: Swing components are rendered by Java and do not rely on the native OS. This provides a consistent look and feel across different platforms.

JDBC (Java Database Connectivity)

Is a Java API that allows Java applications to interact with relational databases. It provides a standard interface for connecting to databases, executing SQL queries, and retrieving results

- Database Connectivity: JDBC enables Java applications to connect to a wide variety of relational databases (e.g., MySQL, PostgreSQL, Oracle) using a database driver.
- SQL Execution: Through JDBC, you can execute SQL statements, such as queries (SELECT), updates (INSERT, UPDATE, DELETE), and more complex commands.
- Result Handling: JDBC allows you to process the results returned by SQL queries, including navigating through results, extracting data, and handling errors.
- Transaction Management: JDBC provides support for transaction management, allowing you to commit or rollback transactions to ensure data integrity.

RMI (Remote Method Invocation)

Is a Java API that allows a Java program to invoke methods on an object located on a different Java Virtual Machine (JVM), whether it's on the same machine or across a network, can communicate over the Internet

- Remote Communication: Allows objects to communicate over a network, enabling the development of distributed applications.
- Object Serialization: Transmits object states between JVMs using Java's built-in serialization mechanism.
- Stubs and Skeletons: Clients use stubs to interact with remote objects, while the server side uses skeletons (deprecated in recent versions) to handle communication.

The Java Sound API

Provides a set of classes and interfaces for handling audio in Java applications

- It allows developers to work with audio data for tasks such as playback, recording, and processing of audio.
- Provides access to audio system resources and manages audio data. It includes classes for audio mixing, playback, and recording.
- javax.sound.sampled.AudioSystem: A central class that provides methods for obtaining and managing audio line objects, including mixers, data lines, and audio input/output lines.

Java Web Start

Was a technology developed by Sun Microsystems (later Oracle) that allowed Java applications to be launched and run over a network from a web browser. It was introduced in Java 1.4 and was part of the Java Runtime Environment (JRE).

- Deployment: Simplified the deployment of Java applications by enabling users to launch Java programs directly from a web browser or desktop shortcut without needing to install them manually.
- Java Network Launch Protocol (JNLP): Used JNLP files to describe how to launch and manage the application. The JNLP file is an XML file that includes information such as the application's main class, required libraries, and resources.

How Java Web Start Worked

- Launching: When a user clicked on a link to a Java Web Start application, the JNLP file was downloaded and read by the Java Runtime Environment (JRE).
- Downloading: Java Web Start downloaded the application's JAR files and other resources specified in the JNLP file.
- Caching: The application was cached locally on the user's machine to improve performance on subsequent launches.
- Running: The JRE launched the application in its own process, providing a user interface as specified by the application.

Java Logging API

Also known as the java.util.logging package. This API provides a standardized mechanism and the actual concrete implementation for logging information in Java applications

Relationship:

- Java Logging API: Provides a concrete implementation of logging within the Java standard library. It's a built-in logging framework with its own set of classes and methods for logging messages.
- SLF4J: Acts as a facade or abstraction layer. It provides a common API that allows you to switch between different logging implementations without changing your logging code.

Usage:

- Java Logging API: Directly used by developers to create loggers and manage log messages.
- SLF4J: Used by developers to write logging code in a generic way, which can then be routed to different logging frameworks, including the Java Logging API, Log4j, Logback, etc.

The original I/O

API in Java, introduced in Java 1.0, is part of the java.io package. This API provides classes and interfaces for performing input and output operations, primarily with files, streams, and data

1. Byte Streams:

- InputStream: Base class for reading byte data.
- OutputStream: Base class for writing byte data.

2. Character Streams:

- Reader: Base class for reading character data.
- Writer: Base class for writing character data.

3. Data Streams:

- DataInputStream: Allows reading of primitive data types.
- DataOutputStream: Allows writing of primitive data types.

4. Object Streams:

- ObjectInputStream: Allows reading of serialized objects.
- ObjectOutputStream: Allows writing of serialized objects.

5. File I/O:

- File: Represents file and directory pathnames, enabling file and directory manipulation.

The Java NIO

Introduced in Java 1.4 and found in the java.nio package, includes several key components designed for more efficient and flexible I/O operations

1. Buffers: Containers for data that allow for efficient reading and writing. Examples include ByteBuffer, CharBuffer, and IntBuffer.
2. Channels: Represent connections to I/O devices and provide a way to read from and write to buffers. Examples include FileChannel, SocketChannel, and DatagramChannel.
3. Selectors: Allow for non-blocking I/O operations by enabling a single thread to manage multiple channels. The Selector class is used for multiplexing I/O operations.
4. Charsets: Support for character encoding and decoding. Key classes include Charset, CharsetEncoder, and CharsetDecoder.

TODO: finish this

Incoming HTTP Request Over a Socket Client Request:

A client (e.g., a web browser or another application) sends an HTTP request to a server. This request is transmitted over the network using the HTTP protocol. Socket Communication:

The HTTP request is sent over a `TCP/IP` connection. This involves creating a socket connection between the client and the server. On the server side, a socket is an endpoint for communication. The web server listens on a specific port (e.g., port 80 for HTTP or 443 for HTTPS) for incoming connections. Web Server Listening:

The web server (e.g., Apache HTTP Server, Nginx) listens for incoming connections on its configured port. When a request arrives, the server accepts the socket connection and reads the incoming HTTP request data from the socket. 2. Request Handling by the Web Server Parsing the Request:

The web server parses the HTTP request to extract the request method (GET, POST, etc.), URL, headers, and body (if present). Routing the Request:

The server determines how to handle the request based on the URL and method. This may involve serving static content (like HTML files) or forwarding the request to an application server for dynamic content. Forwarding to Application Server (if needed):

For dynamic content or web services, the web server forwards the request to a `Java EE` application server. This can be done using various methods, including proxying or direct communication. 3. Application Server Processing Receiving the Request:

The application server (e.g., Apache Tomcat, JBoss EAP) receives the request from the web server. This server is capable of handling `Java EE` components. Request Dispatching:

The application server dispatches the request to the appropriate component based on the configuration. For example, it might route the request to a `servlet` or a web service. 4. Web Service Invocation Service Locator:

If the request is for a web service, the application server locates the appropriate service endpoint based on the request URL and method. Processing the Request:

SOAP Web Services (`JAX-WS`): `WSDL`: The service is described using `WSDL` (Web Services Description Language). The server uses the `WSDL` to understand the request structure and determine how to process it. Service Endpoint: The application server uses `JAX-WS` to handle the SOAP request. It invokes the appropriate service method based on the SOAP message. `RESTful` Web Services (`JAX-RS`): Annotations: `RESTful` services use annotations to map HTTP methods to Java methods. The server uses `JAX-RS` to find the appropriate resource method. Processing: The application server processes the request, possibly involving business logic or database interactions, and prepares a response. Business Logic Execution:

The web service might interact with other components like `EJBs` (Enterprise `JavaBeans`) or data access layers to perform the necessary business logic. 5. Generating and Sending the Response Response Preparation:

The web service generates a response based on the request. For SOAP services, this involves creating a SOAP message. For `RESTful` services, it might be `JSON` or XML. Returning the Response:

The response is sent back to the web server, which then forwards it to the client. The response includes the HTTP status code, headers, and body content. Socket Closure:

Once the response is sent, the socket connection is closed if it is not kept alive for further requests.

Release Cycle

There are several reasons for the Six-Month Release Cycle Increased Agility - The rapid pace of technological advancement and evolving programming paradigms necessitate a more agile approach to language updates. A six-month cycle allows for quicker iterations on features and improvements.

- Feature-Driven Releases: By allowing smaller, more frequent updates, Java can incorporate new features, enhancements, and fixes without waiting for a major version release. This means developers can access improvements more frequently.
- Focus on Quality: The smaller scope of each release helps maintain quality. Instead of waiting for a large number of features to accumulate, each release can focus on a specific set of enhancements or bug fixes.
- Community Engagement: Regular releases encourage community involvement. Developers can provide feedback on new features more frequently, and their input can be integrated into upcoming releases.
- Predictability: A consistent release schedule helps organizations plan their upgrade paths and incorporate new features into their development processes.

Future of the Six-Month Cycle As for whether this cycle will continue indefinitely, several points are worth noting

- Sustainability: The six-month cycle has been well-received and has proven sustainable over several years. There is no official indication from Oracle or the `OpenJDK` community that they plan to change this schedule in the near future.
- Continuous Improvement: The Java community regularly evaluates the release process and adapts as needed. If the community or the demands of modern software development change significantly, adjustments to the release cadence might be considered.
- Feature Proposals: The community continues to propose new features through the `JEP` (JDK Enhancement-Proposal) process. This iterative improvement mechanism fits well with a six-month release model.