Introduction to Java

Lecture 1, 2024

University American College Skopje
School of Computer Science and Information Technology
Course: Programming Languages
Prepared by: Dejan Mitov

Why Fundamentals Matter in the Age of AI: The Unchanging Core in a Fast-Evolving World!"

The future is bright, but the foundation is now. What's your take?

Santiago
@svpino

Subscribe

Software Engineering is changing right before our eyes.

Today's software is fundamentally defined as a set of rules implemented through code. This is all we've known for many decades.

The future, however, looks very different.

Modern software combines code, data, and models. This is a huge shift that requires a completely new approach to how we build software.

AI is not killing software engineering; it's transforming it.

I don't want to stand still while the AI/ML revolution sweeps the world under my feet. I've been trying to adapt, learn, and do things differently.

3 recommendations I'm currently following:

1. Learning how to bend computers to our will is—and will continue to be—a fundamental skill. In 2024, this involves learning how to write code aided by AI.

2. The ability to build scalable software solutions is still one of the most valuable skills in the world. Understanding how to go from "it works on my computer" to "a million people are using this" is a competitive advantage very few humans have.

3. Few things are more valuable than our ability to think objectively and critically about novel problems. Don't outsource your basic understanding of how software works to an AI model. Learn the fundamentals.

What else can we do to stay competitive for the next 10 years?

6:53 PM · Sep 20, 2024 · **108K** Views

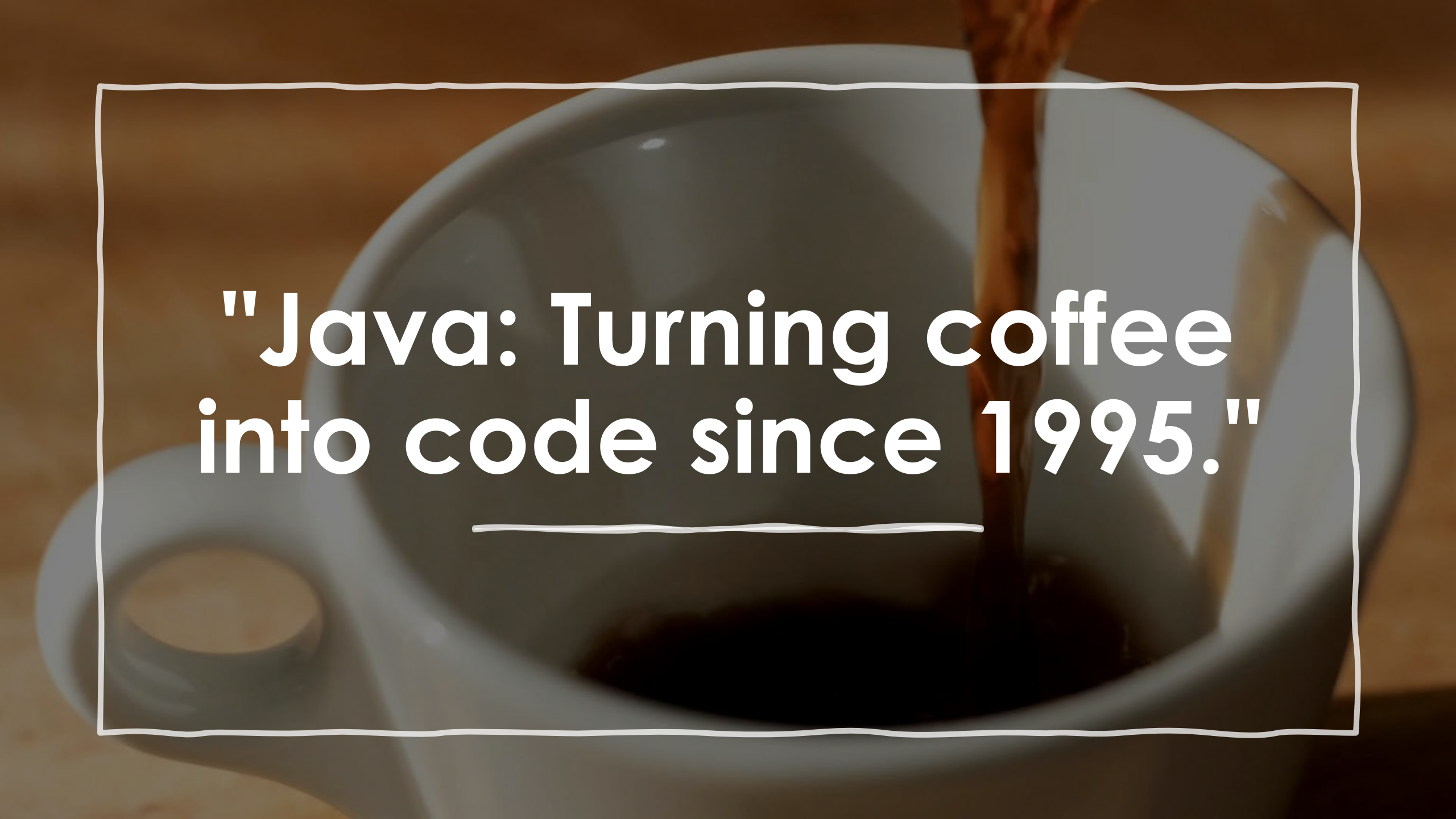
50 165 1K 765

Hey! I'm Santiago.

I'm a machine learning engineer with over two decades of experience building and scaling enterprise software and machine learning systems.

I love neural networks. I love to make them work at scale.

From 2009 to 2023, I built products for Disney, Boston Dynamics, IBM, Dell, G4S, Anheuser-Busch, and NextEra Energy, among other clients. I learned about trade-offs and how to create products that work.


"Java: Turning coffee
into code since 1995."

What is Java?

A versatile programming language with the following features:

- Cross-platform compatibility
- Object-oriented design
- Network-centric architecture
- Multimedia support

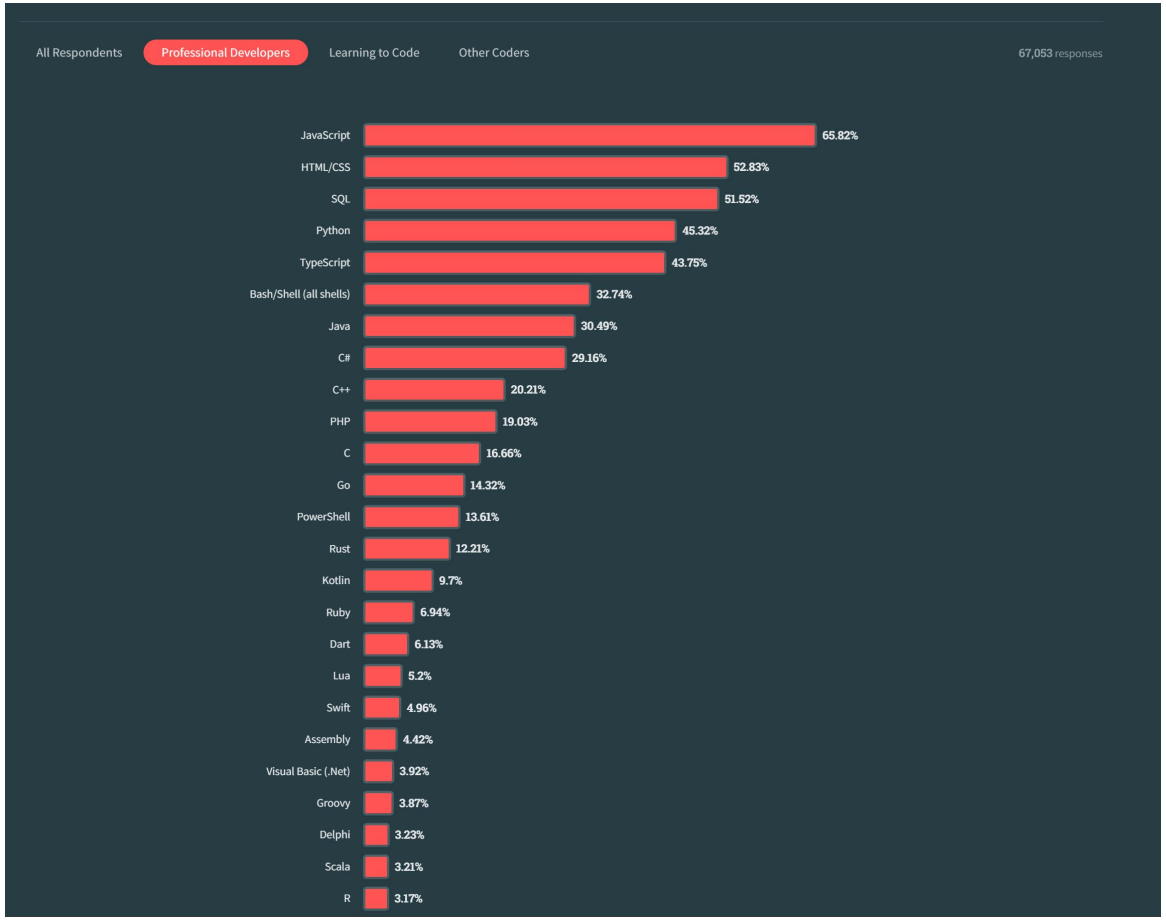
Java's Unique Characteristics:

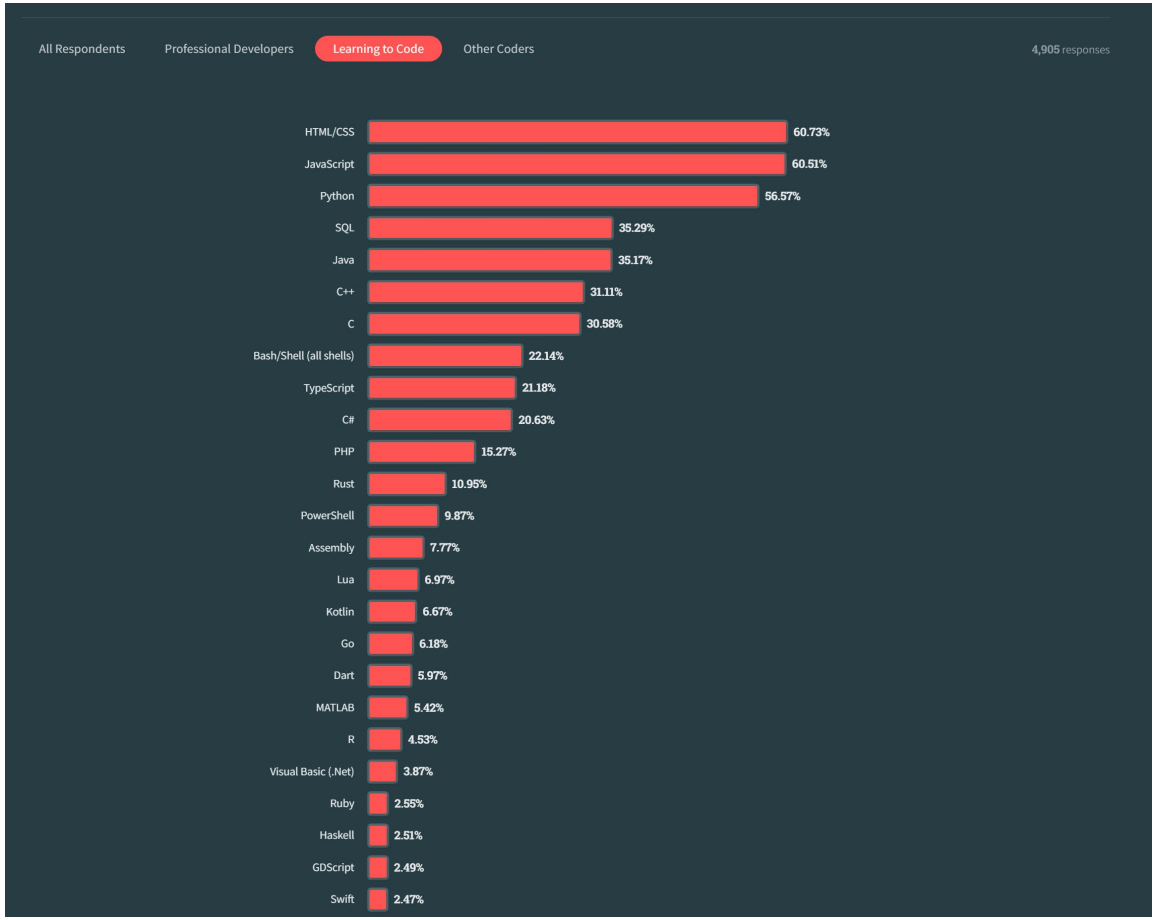
- Self-contained programming environment
- Java applications can run independently, without reliance on external platforms or software

Brief History of Java

- **Inception**: Created by James Gosling at Sun Microsystems in 1991.
- **Initial Purpose**: Initially designed for interactive television, but evolved due to its potential for broader applications.
- **Public Release**: First publicly released as Java 1.0 in 1995.
- **Breakthrough**: Gained prominence for its "Write Once, Run Anywhere" philosophy.
- **Enterprise Adoption**: Java 2 Platform, Enterprise Edition (J2EE) launched in 1999, paving the way for enterprise-level applications.
- **Ownership Transitions**: Acquired by Oracle Corporation in 2010 after Oracle bought Sun Microsystems.
- **Current Version**: Continues to evolve with regular updates, enhancing features and performance.

Is Java Outdated? What's the Rationale for Learning It?





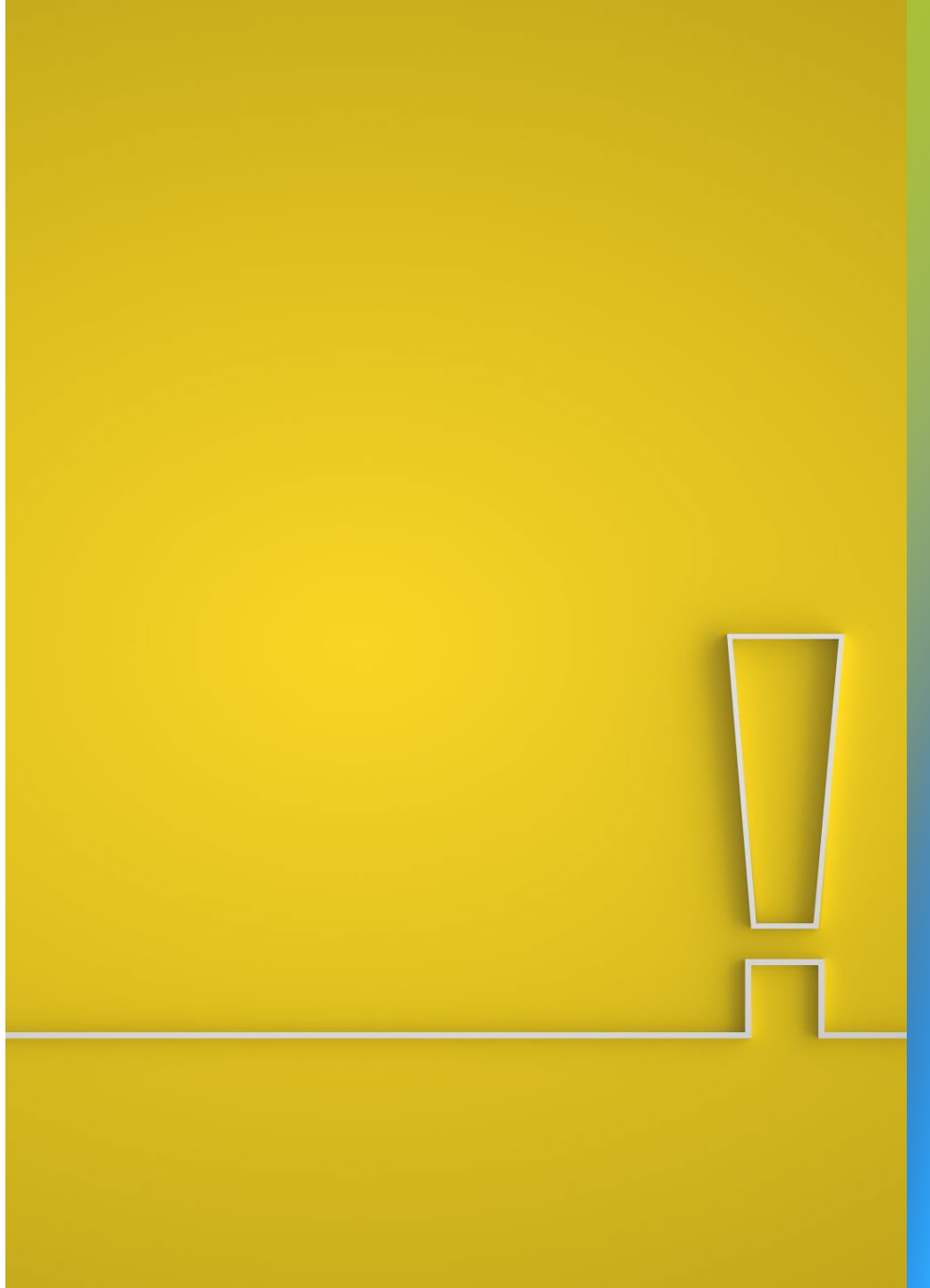
Stack Overflow Developer Survey 2023

As per Sun Microsystems, Java is a programming language characterized by its:

- Ease of Use
- Object-Oriented Nature
- Network-Distributed Capabilities
- Interpretive Execution
- Reliability
- Enhanced Security
- Platform Independence
- Portability Across Systems
- High-Speed Performance
- Support for Multithreading
- Dynamism

Java's Simplicity

- Influenced by C++
  - Retains the use of curly braces { }
  - Incorporates common keywords like if, else, for, while, do, etc.
- Streamlined Design Choices
  - Replaces multiple inheritance with interfaces
  - Omits pointers for simplified coding
- Automated Resource Management
  - Automatic memory allocation and garbage collection



Java's Object-Oriented Nature

Differentiating Factor:

- Unlike procedural languages such as COBOL, FORTRAN, BASIC, Pascal, and Ada, Java is object-oriented.

Procedural vs Object-Oriented:

- Procedural languages focus on procedures as the main operating units.
- Java, following in C++'s footsteps, emphasizes objects.

Role of Objects:

- Objects encapsulate features, including methods that represent executable code.
- Objects act as software agents, equipped with instructions to carry out specific tasks.

Java as a Distributed Programming Language

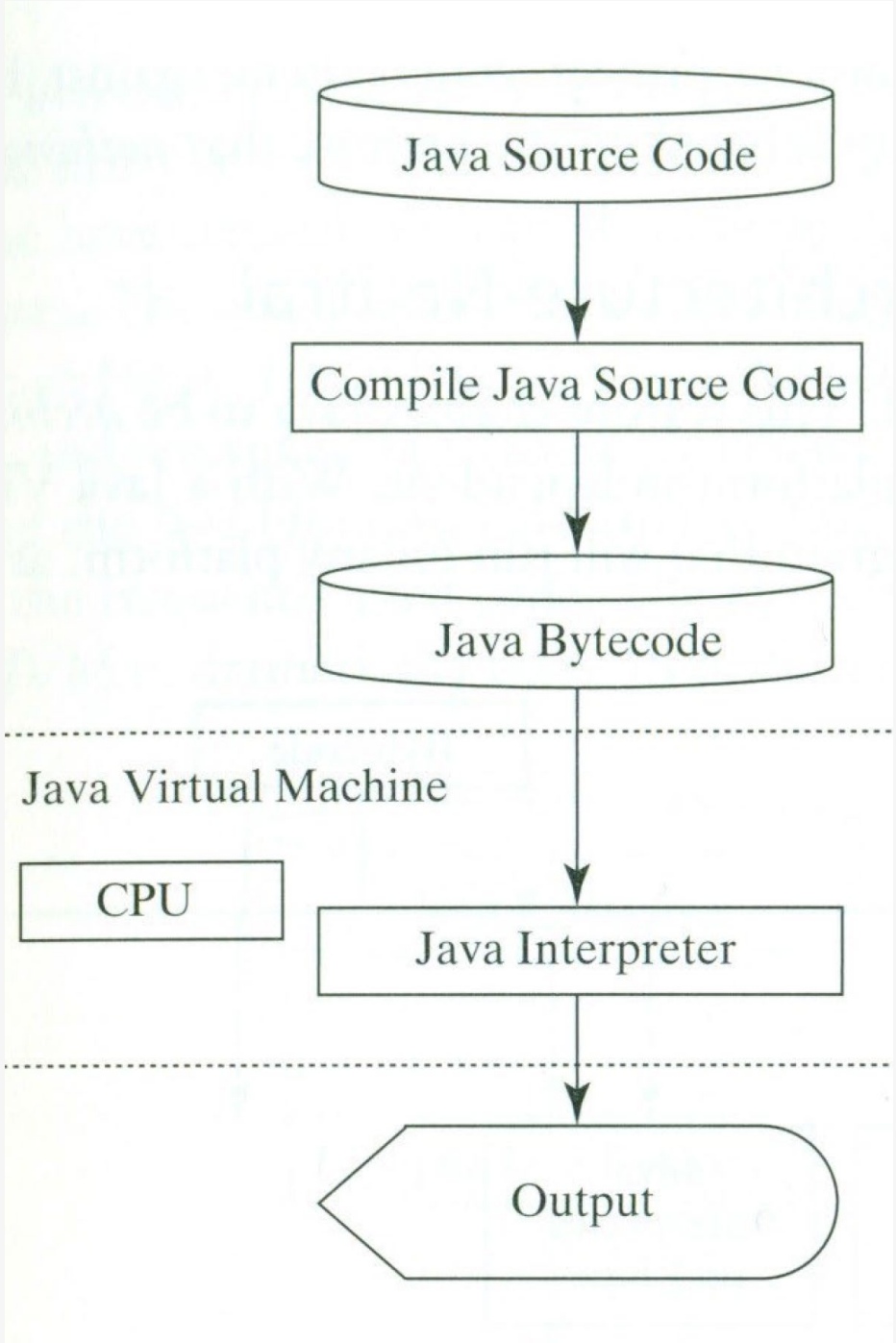
Distributed computing refers to a system where multiple interconnected computers work together to achieve a common task or objective.

- Fundamental Aspect:
  - Java is inherently designed for network-based, distributed computing.
- Remote Method Invocation (RMI):
  - Enables Java objects on different machines to interact, facilitating distributed computing.
- Java EE (Enterprise Edition):
  - Offers robust features for building large-scale, distributed systems.
- Network Libraries:
  - Rich set of APIs for network protocols like HTTP, FTP, and TCP/IP, simplifying network communication.
- Security Features:
  - Built-in security mechanisms ensure safe data transmission across networks.

Java is Interpreted: A Two-Step Execution Process

Compiled languages are translated into machine code in a single step for direct execution by the computer's hardware, while interpreted languages are translated line-by-line and executed on the fly by an interpreter during runtime.

- Initial Compilation:
  - Java source code is first compiled into an intermediate form known as bytecode.
- Cross-Platform Compatibility:
  - This bytecode can be executed on any system equipped with a Java Virtual Machine (JVM), ensuring platform independence.
- Bytecode to Machine Code:
  - The JVM interprets the bytecode into native machine code specific to the host system.
- Internet-Focused Interpretation:
  - The interpretation mechanism is designed to fully support Java's internet-centric features.



Java is Robust: Built for Reliability

Definition of Robustness:

- In Java, being robust is synonymous with being highly reliable.

Strong Typing:

- Java mandates type declaration for each variable, enforcing type safety and reducing runtime errors.

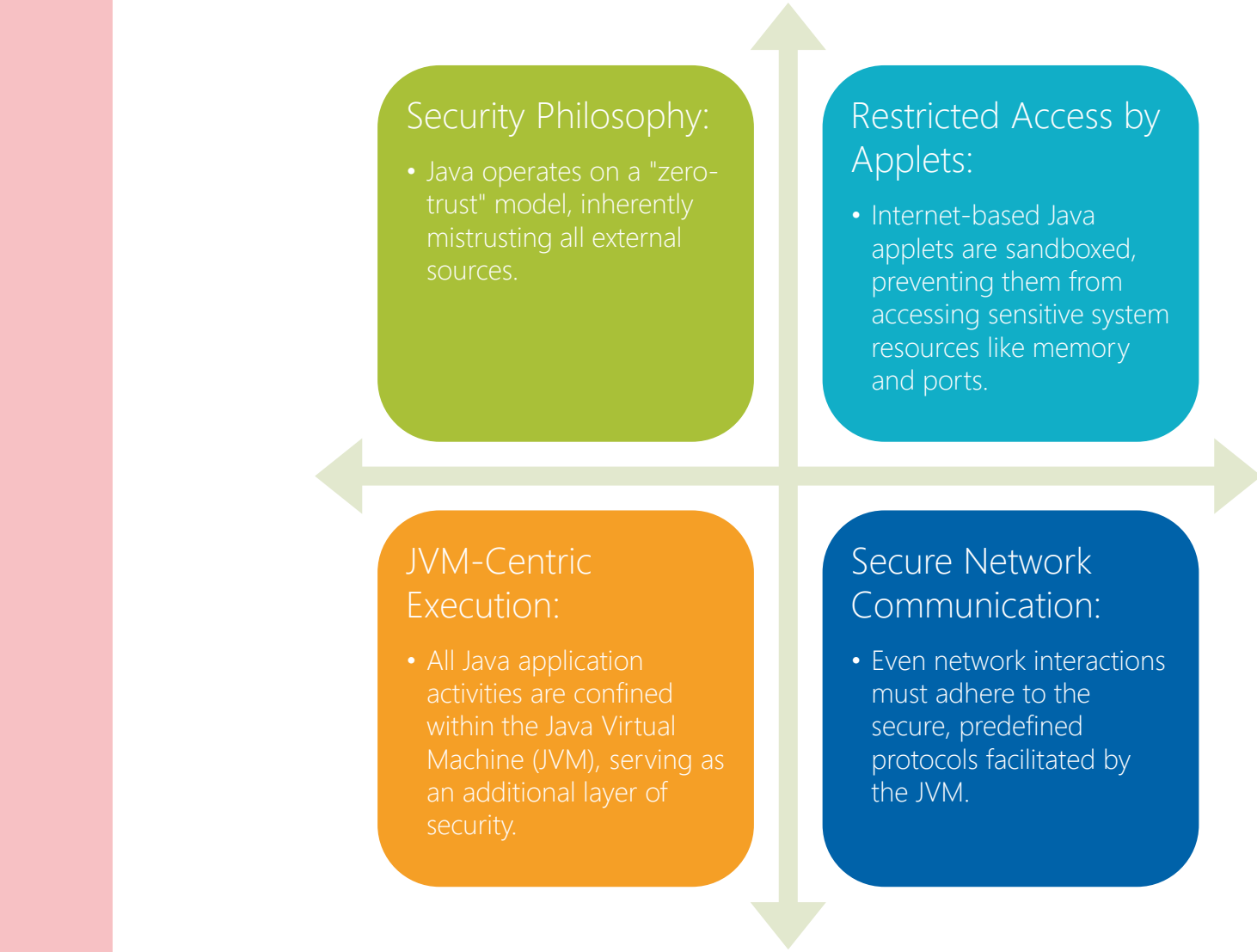
No Pointers:

- The absence of pointers eliminates risks associated with unauthorized memory access.

Exception Handling:

- Java is equipped with robust exception-handling mechanisms, ensuring that exceptional situations like division by zero don't crash the program by default.

Java is Secure: A Trust-Nothing Approach



Java is Architecture-Neutral: Universal Bytecode Across Platforms

- Interpreted Nature:
  - Java's architecture-neutrality is fundamentally tied to its interpreted execution model.
- Multi-Platform JVMs:
  - Versions of the Java Virtual Machine (JVM) are available for a diverse range of operating systems, including but not limited to Windows, UNIX, Mac, OS/2, and various mobile OS.
- Uniform Bytecode:
  - The bytecode generated is identical across all platforms, ensuring seamless portability.
- Cross-Platform Execution:
  - A Java program can execute on any machine equipped with a compatible JVM, regardless of the underlying architecture.

Java's High-Performance: Bridging the Gap with JIT Compilation

- Debatable Performance:
  - From a theoretical standpoint, compiled languages are generally faster than interpreted ones, which raises questions about Java's high-performance claim.
- Advancements in Hardware:
  - Modern, faster CPUs have significantly narrowed the performance gap, making Java nearly as efficient as compiled languages.
- Just-In-Time (JIT) Compilation:
  - A hybrid approach that blends features of both compilation and interpretation.
  - Code is compiled not just line-by-line, but also several lines ahead.
  - Once compiled, the code is stored for future reuse.
  - Despite its compilation aspects, JIT is technically classified under interpretation.

Java is Multithreaded: Concurrency Built-In

Concurrency Support:

- Java allows for the simultaneous execution of multiple threads, enabling more efficient use of CPU resources.

Practical Applications:

- Examples include writing to a file while updating the user interface, which is particularly useful in GUI and network applications.

Multi-Tasking Capabilities:

- Facilitates handling of multiple user interfaces or clients simultaneously, crucial in today's multi-window and multi-user environments.

Inherent to JVM:

- Multithreading support has been an integral part of the Java Virtual Machine (JVM) since its inception.

Java is Dynamic: Built for Evolving Ecosystems

Adaptive Design:

- Java is engineered to be flexible and adapt to changing environments without requiring significant alterations.

Seamless Software Updates:

- Allows for the use of new versions "on the fly," eliminating the need for manual uninstallation and reinstallation of software.

Effortless Version Management:

- Users can simply replace the existing bytecode to upgrade, streamlining the update process.

Java's Object-Oriented Paradigm: Objects as Agents

Object-Centric Design:

- In Java, the majority of elements are treated as objects, although not absolutely everything falls under this category.

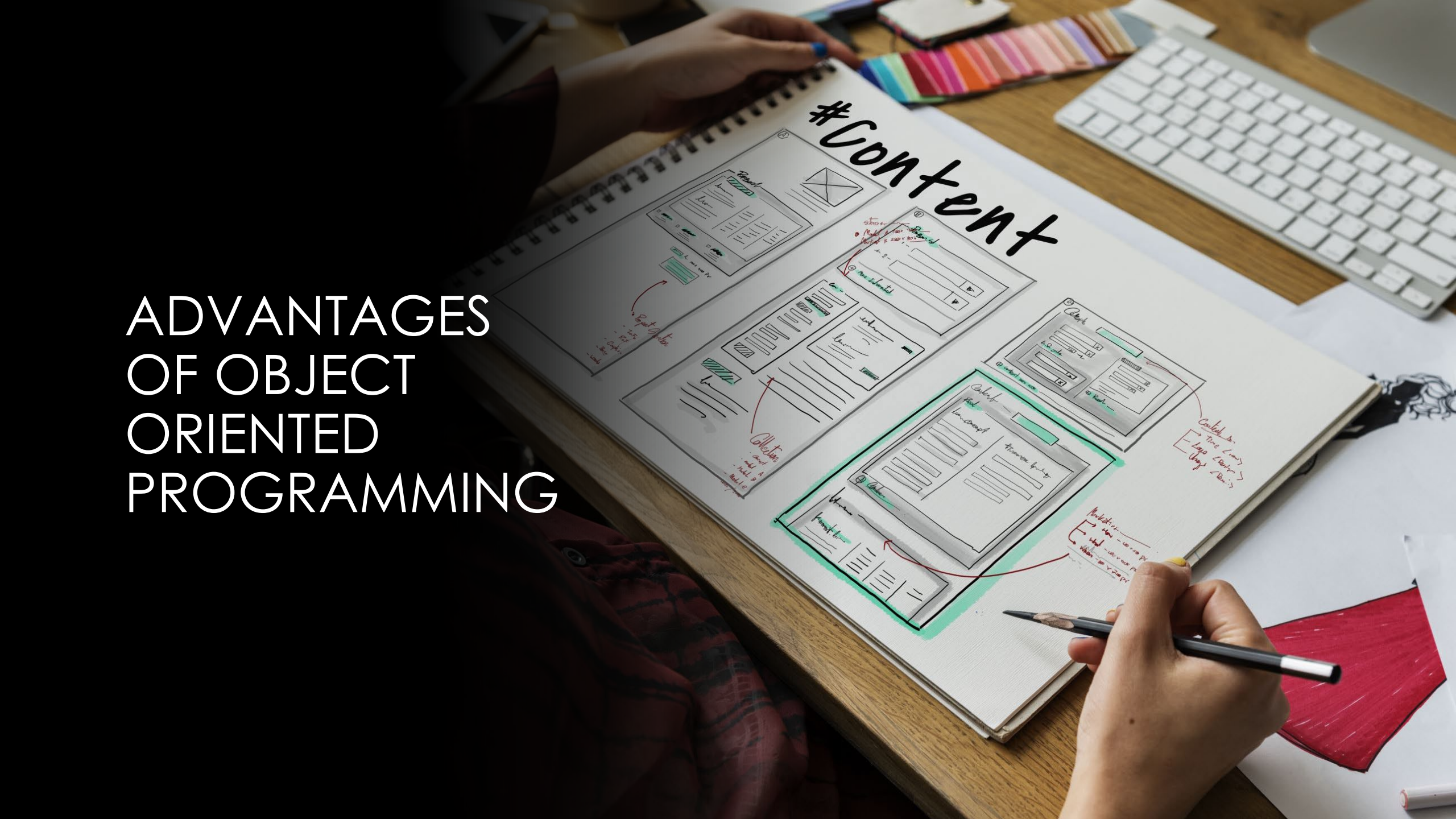
Contrast with Procedural Languages:

- In procedural languages, procedures handle tasks, and the language itself serves as the executing agent.
- Introduction of a new task necessitates the creation of a new procedure.

Object-Oriented Approach:

- Classes are defined with both descriptive and operative code to outline the tasks to be executed.
- Objects, derived from these classes, perform the actual tasks.
- In this paradigm, the language sets the environment, while objects act as agents.
- For new tasks, either a new object is instantiated, or an entirely new class is created to generate the new object.

ADVANTAGES OF OBJECT ORIENTED PROGRAMMING



Encapsulation: The Art of Information Hiding

- Internal Code Concealment:
  - Encapsulation ensures that the inner workings of objects are hidden from external access, shielding the operational code from users.
- User Interaction:
  - Users can utilize the encapsulated features without needing to understand the underlying mechanics.
  - For instance, turning a car key ignites the engine, but the driver doesn't need to know the technical details involved.
- Programming Advantages:
  - Developers can leverage encapsulated elements to streamline their own software, simplifying complex functionalities into easy-to-use interfaces.

Polymorphism: The Power of Flexibility

Context-Sensitive Behavior:

- Polymorphism allows a single instruction to take on different meanings based on the context.
- For example, 2+5=7 in a numerical context, while "two" + "five" equals "twofive" in a string context.

Dynamic Outcomes:

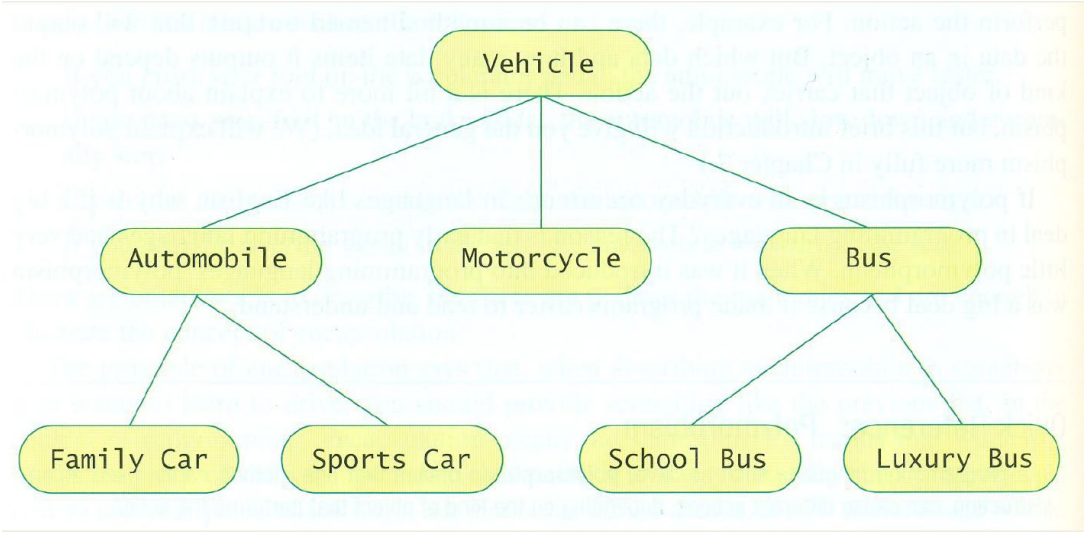
- The result of invoking a method can vary depending on the data provided during the method call.

Types of Polymorphism:

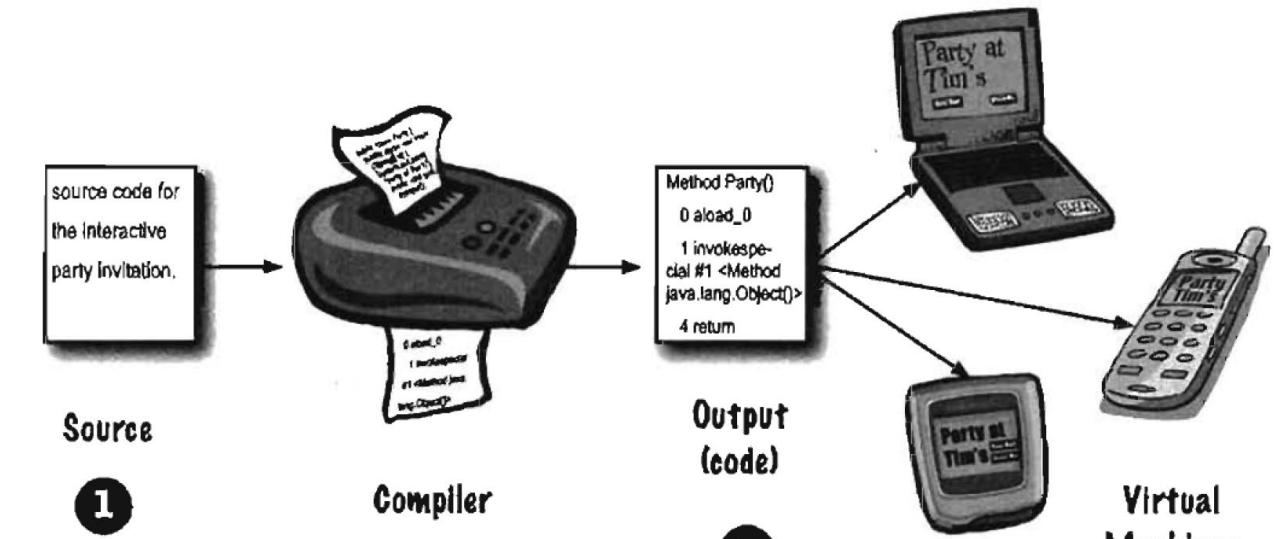
- Compilation-Time Polymorphism (Overloading): Occurs when methods of the same name are defined with different parameters.
- Runtime Polymorphism (Overriding): Occurs when a subclass provides its own implementation of a method that is already provided by its superclass.

Inheritance: Maximizing Code Reusability

- Inherited Functionality:
  - Inheritance allows one class to adopt the attributes and methods of another, serving as a foundation for further development and enhancement.
- Extending Features:
  - While inheriting the core functionalities of a superclass, a subclass can also introduce additional attributes and methods, thus expanding its capabilities.



HOW JAVA OPERATES



Source

1

Create a source document. Use an established protocol (in this case, the Java language).

Compiler

2

Run your document through a source code compiler. The compiler checks for errors and won't let you compile until it's satisfied that everything will run correctly.

Output (code)

3

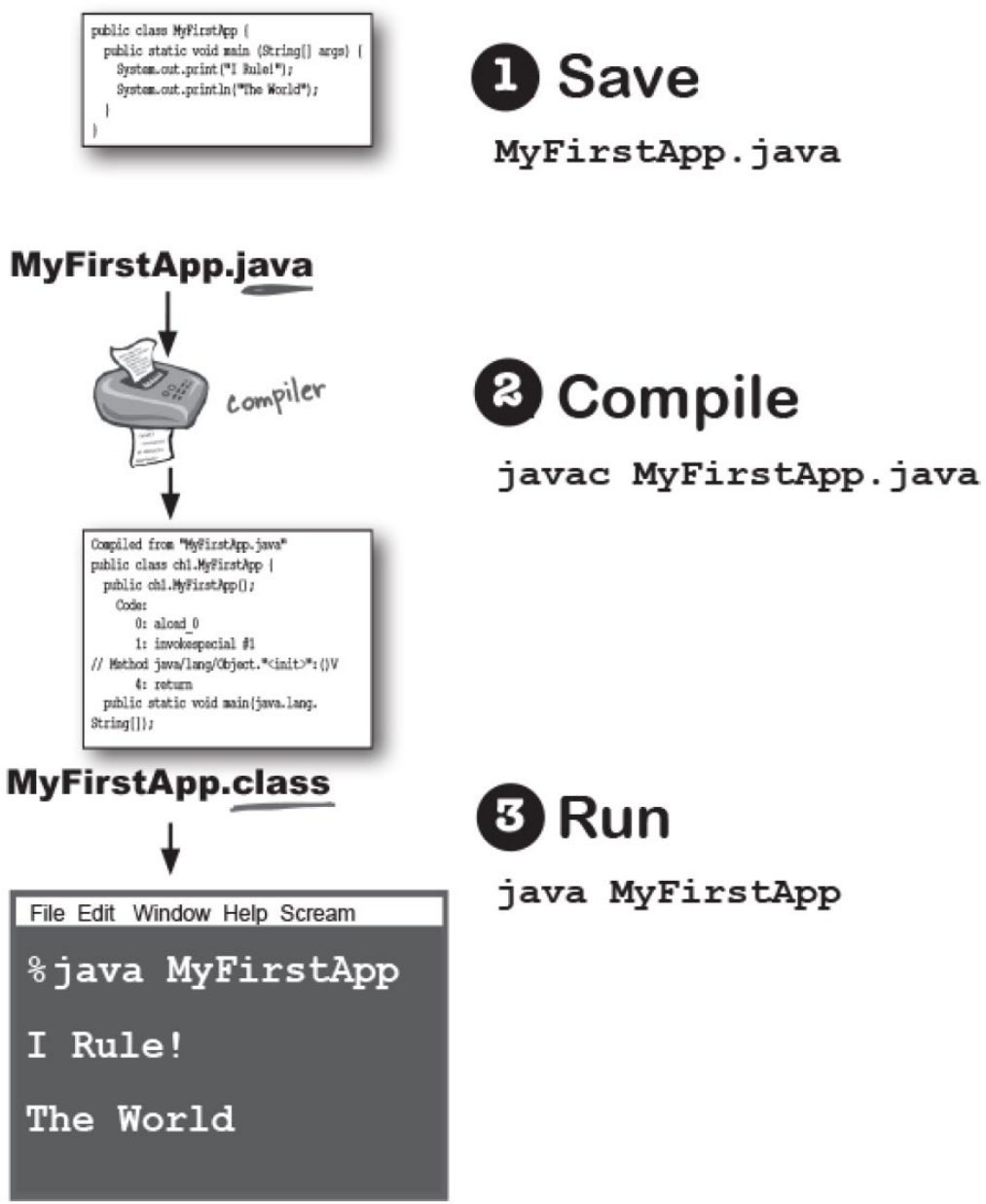
The compiler creates a new document, coded into Java ***bytecode***. Any device capable of running Java will be able to interpret/translate this file into something it can run. The compiled bytecode is platform-independent.

Virtual Machines

4

Your friends don't have a physical Java Machine, but they all have a ***virtual*** Java machine (implemented in software) running inside their electronic gadgets. The virtual machine reads and *runs* the bytecode.

HOW THE PROGRAMS ARE GENERATED AND EXECUTED



The code structure in Java

A source file: CLASS

public class Dog {
…
}

The class has one or more : METHODS

void bark () {
…
}

A method can be defined as a set of STATEMENTS

How easy it is to write Java

```
int size = 27;
String name = "Fido";
Dog myDog = new Dog(name, size);
x = size - 5;
if (x < 15) myDog.bark(8);

while (x > 3) {
  myDog.play();
}

int[] numList = {2, 4, 6, 8};
System.out.print("Hello");
System.out.print("Dog: " + name);
String num = "8";
int z = Integer.parseInt(num);

try {
  readTheFile("myFile.txt");
}
catch (FileNotFoundException ex) {
  System.out.print("File not found.");
}
```

Java: A Fully Integrated Object-Oriented Language

- Method-Class Integration:
  - In Java, methods are intrinsically tied to classes and cannot exist independently, reinforcing its object-oriented nature.
- Absence of Global Methods:
  - The language design prohibits the concept of "global methods," ensuring that all methods must belong to a specific class.

Engineered for Security Through Design Choices

- No Pointers by Default:
  - Java eliminates the use of pointers and pointer arithmetic, reducing the risk of unauthorized memory access.
- Direct Object Generation:
  - Objects in Java are instantiated directly, bypassing the need for pointers.
- Reference-Based Access:
  - Objects are accessed via references that are linked to their corresponding memory locations, adding an extra layer of security.

Anatomy of a Java Class: A Walkthrough

- Basic Structure:

- Execution Process:
  - Running the program involves instructing the Java Virtual Machine (JVM) to load the class and invoke the main() method.
- Execution Lifecycle:
  - The program continues to execute as long as the main() method is running, and it terminates once the main() method concludes.

Understanding Java Instructions: A Closer Look

- Types of Instructions:
  - Java supports various kinds of instructions, including declarations, assignments, and method invocations, among others.
- Code Examples:

MyFirstApp.java

```java
public class MyFirstApp {
    public static void main(String[] args) {
        int x = 3;                  // Declaration & Assignment
        String name = "Dirk";       // Declaration & Assignment
        x = x * 17;                 // Assignment with Arithmetic Operation
        System.out.print("x is " + x); // Method Invocation
        double d = Math.random();   // Method Invocation & Assignment
        // This is a comment       // Comment
    }
}
```

Understanding Branching in Java: Conditional Logic Explained

- Java provides flexible branching mechanisms through if-else statements, allowing for complex decision-making in your code.
- You can combine multiple conditions for more nuanced logic, while certain lines can be designed to execute unconditionally.
- if-else Statements:
  - Used for making decisions based on conditions.
- Compound Conditions:
  - Multiple conditions can be combined using logical operators like &.

```
if (x == 10) {
    System.out.print("x must be 10");
} else {
    System.out.print("x is not 10");
}
```

```
if ((x < 3) & (name.equals("Dirk"))) {
    System.out.println("Gently");
}
```

Understanding Simple Boolean Operators in Java

- < (Less Than)
- > (Greater Than)
- <= (Less Than or Equal To)
- >= (Greater Than or Equal To)
- == (Equal To)
- != (Not Equal To)

Fundamental Data Types in Java: A Comprehensive Overview

Java offers a variety of elementary data types to handle different kinds of numerical, textual, and logical data. In addition to these, Java also provides reference types like `String` for more complex data handling needs.

- Elementary Data Types:
  - `boolean`: For true/false values
  - `byte`: 8-bit integer value
  - `char`: Single Unicode character
  - `short`: 16-bit integer value
  - `int`: 32-bit integer value
  - `long`: 64-bit integer value
  - `float`: Single-precision floating-point number
  - `double`: Double-precision floating-point number
- Reference Types:
  - `String`: Sequence of characters, treated as an object rather than a primitive type

The boolean Data Type in Java: Essentials

The boolean data type is a simple yet powerful construct in Java, designed to store binary logical values.

- Size:
  - Occupies 1 bit of memory.
- Possible Values:
  - Can only hold one of two values: true or false.
- Declaration:
  - Syntax: boolean b;
- Assignment:
  - Examples: b = true; or b = false;

Understanding `byte` and `char` Data Types in Java

The byte data type is primarily used for storing small integer values and supports ASCII character representation.

- `byte`:
  - Size: 1 signed byte
  - Value Range: -128 to 127
  - Character Support: ASCII characters
- `char`:
  - Size: 2 unsigned bytes
  - Value Range: 0 to 65,535
  - Character Support: Unicode characters

Exploring short, int, and long Data Types in Java

The short, int, and long data types are used for storing integer values, with varying sizes and ranges to accommodate different numerical needs. Each type has its own specific range of values it can store, allowing you to choose the most efficient type for your application.

- short:
  - Size: 2 signed bytes
  - Value Range: -32,768 to 32,767
- int:
  - Size: 4 signed bytes
  - Value Range: -2,147,483,648 to 2,147,483,647
- long:
  - Size: 8 signed bytes
  - Value Range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
- *Note: Signed integers can represent both positive and negative values but have a smaller range of positive numbers they can represent. Unsigned integers can only represent positive numbers and zero but can represent a larger range of positive numbers.*

Understanding `float` and `double` Data Types in Java

Both float and double are used for storing floating-point numbers, but they differ in size and precision. float is suitable for applications that can tolerate less precision and a smaller range, while double is used when higher precision and a larger range are required.

- `float`:
  - Size: 4 signed bytes
  - Value Range: $1.40129846432481707 * 10^{-45}$ to $3.40282346638528860 * 10^{38}$ (positive or negative)
- `double`:
  - Size: 8 signed bytes
  - Value Range: $4.94065645841246544 * 10^{-324}$ to $1.79769313486231570 * 10^{308}$ (positive or negative)

Understanding the `String` Data Type in Java

- The `String` data type in Java is an object-oriented representation of text.
- Embedded Type:
  - In Java, strings are treated as an embedded, or built-in, type, which sets it apart from languages like C and C++ where strings are arrays of characters.
- Example:

```
public class MyFirstApp {
    public static void main(String[] args) {
        String s = "This is a string";
    }
}
```

- Character Access:
  - Unlike character arrays, strings in Java do not allow for direct character extraction.
  - However, specific methods are available for accessing and manipulating characters within the string.

Understanding Arithmetic Operators in Java

- `+`: Addition Operator
- `-`: Subtraction Operator
- `*`: Multiplication Operator
- `/`: Division Operator
- `%`: Modulus Operator (Remainder)

- Arithmetic operators in Java allow for basic mathematical operations, enabling you to perform calculations like addition, subtraction, multiplication, division, and finding the remainder.
- These operators are fundamental for any mathematical manipulations in your Java programs.

Understanding Unary Operators in Java

Unary operators in Java are used to modify the value of a single operand. These operators are useful for specifying sign, as well as for incrementing or decrementing variables in a compact manner.

- `+`: Unary Plus Operator (Indicates Positive Value)
- `-`: Unary Minus Operator (Indicates Negative Value)
- `++`: Increment Operator (Increases Value by 1)
- `--`: Decrement Operator (Decreases Value by 1)

Greatness of Java: A Summary

1. If you're looking for the epitome of user-centric design in programming, look no further than Java.
2. Its 'Write Once, Run Anywhere' philosophy is revolutionary, breaking down barriers between platforms.
3. But it's not just about compatibility; Java's object-oriented nature is a masterclass in modular, reusable code. It's like building with Lego blocks, each piece seamlessly integrating into a greater, more efficient whole.
4. Java is a fortress in a world teeming with digital vulnerabilities. It's like having a VIP pass to the safest, most exclusive club in town.
5. Add to that a standard library so rich it's akin to an artist's fully-stocked palette, and you begin to see the unlimited possibilities.
6. But what makes Java truly special is its community. It's not just a language; it's a thriving ecosystem.
7. A universe brimming with innovation, inspiration, and endless resources. Java doesn't just make things possible; it makes them inevitable."


Questions?