西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

Object Oriented Programming

Chapter 1 Introduction

Dr. Helei Cui

25 Feb 2025

Slides partially adapted from lecture notes by Cay Horstmann

Introduction to this course

About this course

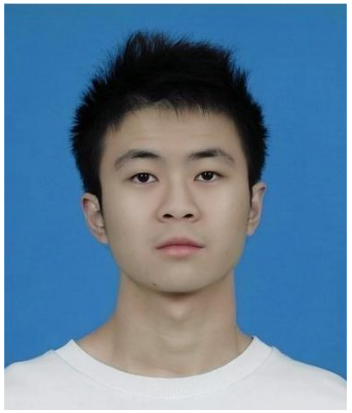
- Course code: U10M12004
- Course title: **Object Oriented Programming**
- Hours and credits: 40 hours/2.5 credits
- Prerequisite courses: C Programming Language
- Course offered by: School of Computer Science
- Starting semester: Spring
- Course category: Discipline Elementary Course
- **Schedule: 19:00-20:40, Tuesday & Thursday, Week 2-11**
  - 10 minutes break at 19:45
- **Course webpage:**
  - https://harrycui.github.io/courses/oop-cs-2025.html

Teaching group

- **Instructors:**
  - Chapter 1~4: Dr. Helei Cui 崔禾磊
  - Chapter 5~9: Dr. Muhammad Umar Farooq Qaisar
  - Lab session: Dr. Yaxing Chen 陈亚兴
- **Teaching assistant:**
  - Mr. Hao Zeng 曾豪 (PhD student)

Muhammad Umar Farooq Qaisar
muhammad@nwpu.edu.cn

Yaxing Chen
yxchen@nwpu.edu.cn



Hao Zeng
hao_zeng@mail.nwpu.edu.cn

About me

- Dr. Helei Cui（崔禾磊）
  - **Professor in CS**
  - PhD @ City University of Hong Kong
  - MSc @ The Chinese University of Hong Kong
  - BEng @ Northwestern Polytechnical University
- Research Interests
  - Trustworthy Crowd Computing
  - Privacy-Preserving Computing
  - Edge Intelligence
  - Decentralized Cloud Storage
  - …
  - More details @ https://harrycui.github.io/

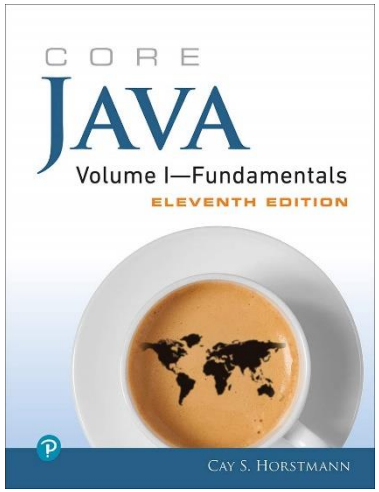
Quick question 1

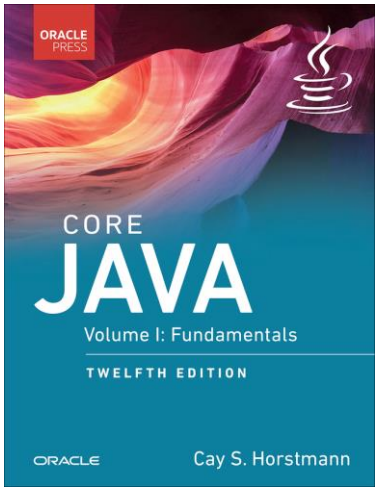
Which programming language have you learned before?

A. C/C++
B. Java
C. Python, Go, Swift, etc.
D. None

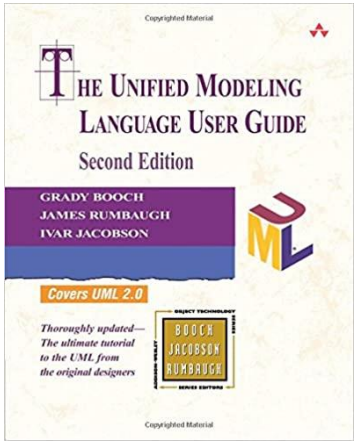
Textbooks

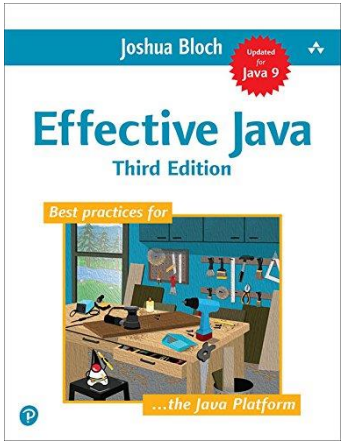
- Textbook:
  - Cay Horstmann, *Core Java Volume I - Fundamentals 11th Edition*, Pearson, 2018.





- Reference books:
  - Grady Booch, James Rumbaugh, and Ivar Jacobson, *The Unified Modeling Language User Guide 2nd Edition*, Addison-Wesley Professional, 2017.
  - Joshua Bloch, *Effective Java 3rd Edition*, Addison-Wesley Professional, 2017.





Tools

- Hardware:
  - PC (Windows or Linux)
  - Mac (macOS)

- Software:
  - Eclipse IDE
  - Notepad
  - Sublime
  - VS Code
  - …

Assessment (tentative)

- Attendance (10%)
  - Randomly check
- Midterm Quiz (20%)
  - Multiple choice questions and others
- Assignment (20%)
  - Two tasks, covering UML design and programming
- Final Exam (50%)
  - No less than 60 grades

Chapters

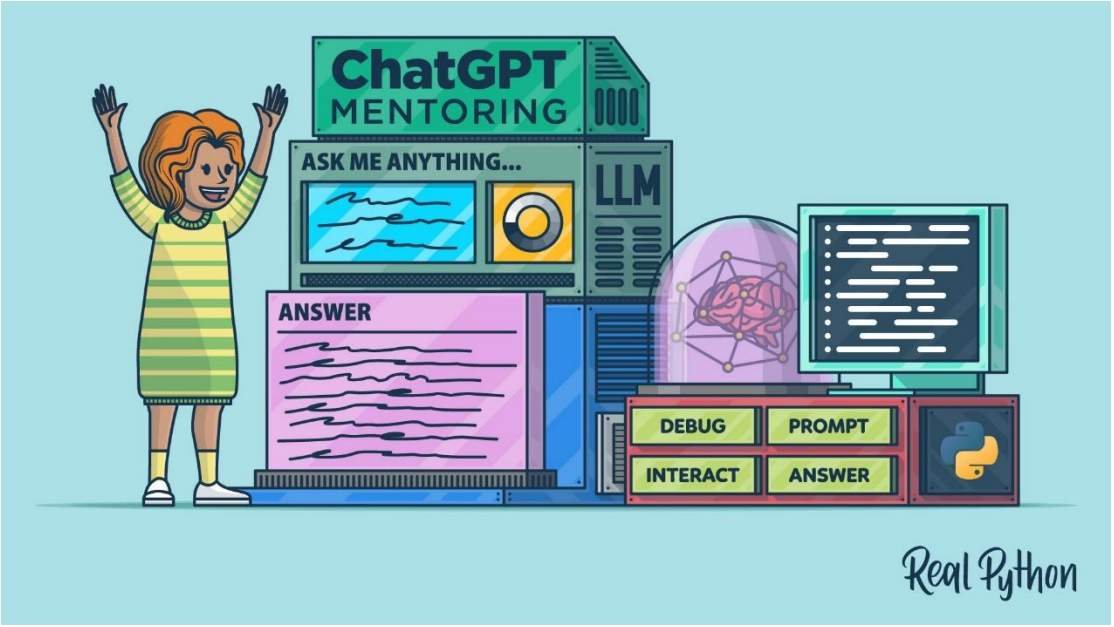
1. Introduction (4 hours)
2. The Java Programming Environment (2 hours)
3. Fundamental Programming Structures in Java (8 hours)
4. Object and Classes (6 hours)
5. Inheritance (6 hours)
6. Interfaces, Lambda Expressions, and Inner Classes (5 hours)
7. Exceptions (3 hours)
8. Collections (4 hours)
9. I/O (2 hours)

Intended learning outcomes

1. Learn Java programming language, including types, operators, program control, and several useful classes.
2. Develop problem-solving skills through practice and understanding of the divide-and-conquer and top-down approaches.
3. Learn the principles of OOP in Java with the usage of classes, inheritance, polymorphism, interfaces, containers, and with the goal of understanding code reuse and building scalable programs.
4. Use UML tools to visualize a system design.

Suggestions

- **Coding style is extremely important.**
- **Try to code it by yourself.**
- **Google is your “best” teacher.**
- **Enjoy coding.**



A first look at Object Oriented Programming (OOP)

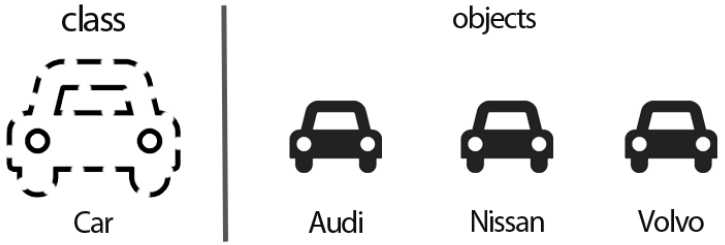
What is OOP?

- OOP allows programmers to think of software development as if they are working with real-life entities.
  - In your everyday life, people have the knowledge and can-do various works/tasks.
  - In OOP, objects have fields to store knowledge/state/data and can-do various methods.
- OOP is a **programming paradigm** based on the concept of "**objects**", which can contain data and code:
  - **data**, in the form of fields (a.k.a. *attributes* or *properties*);
  - **code**, in the form of procedures (a.k.a. *methods*).

OOP helps programmers create complex programs by grouping together related data and methods.

Some basic terminologies

- **Object**
  - Objects are instances of a class.
- **Class**
  - Classes are templates for objects.



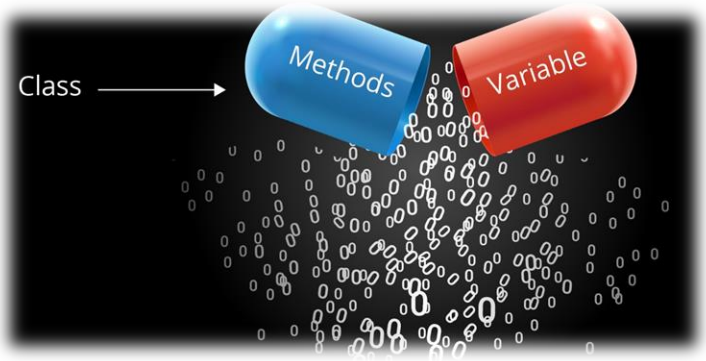
- **Method**
  - Can modify a class state that would apply across all the instances of the class.
- **Instance**
  - **Recall that *"An object is an instance of a class".***
  - Let's think about it in these terms:
    - A blueprint for a car design is the class description, all the cars manufactured from that blueprint are *objects* of that class.
    - Your car that has been made from that blueprint is an *instance* of that class.

Four main principles

1. Encapsulation
2. Abstraction
3. Inheritance
4. Polymorphism

Encapsulation

- **Bundles data with methods** that can operate on that data within a class.
  - Essentially, it is the idea of hiding data within a class, preventing anything outside that class from directly interacting with it.



Keeps the programmer in control of access to data and prevents the program from ending up in any strange or unwanted states.

Abstraction

- **Only shows essential details** and keeps everything else hidden.

Important to driver

1. How the steering wheel steers the car?
2. How much gas your car has?

Car

Not important to driver

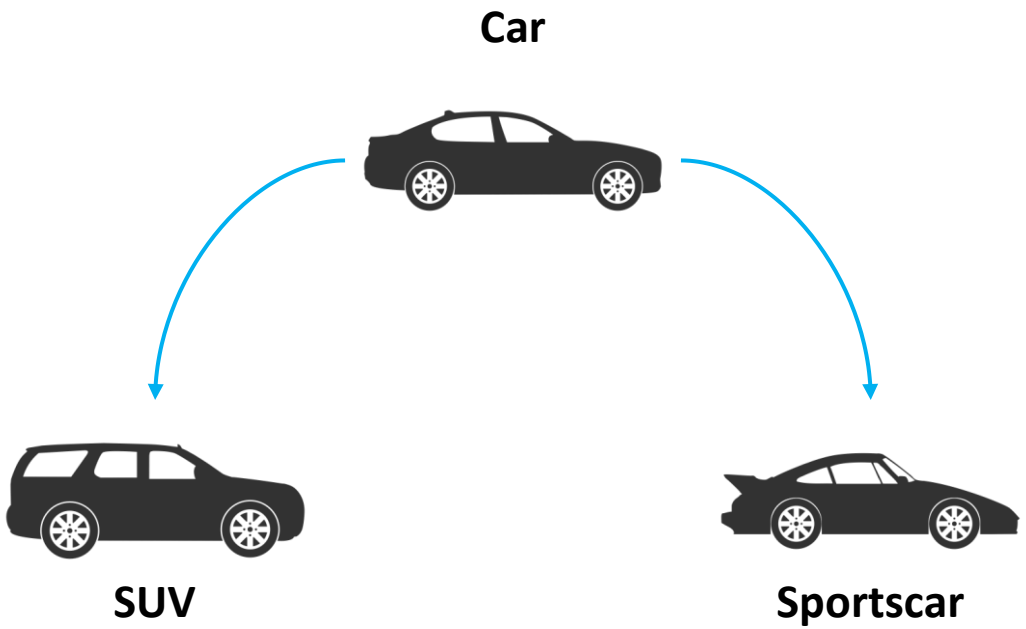
1. How the gas reacts to the engine?
2. How the engine makes your car move?

- Users of your classes should not worry about the inner details of those classes.
- The interface is exposed for communication, while the implementation should be hidden.

Abstraction allows the program to be worked on incrementally and prevents it from becoming entangled and complex.

Inheritance

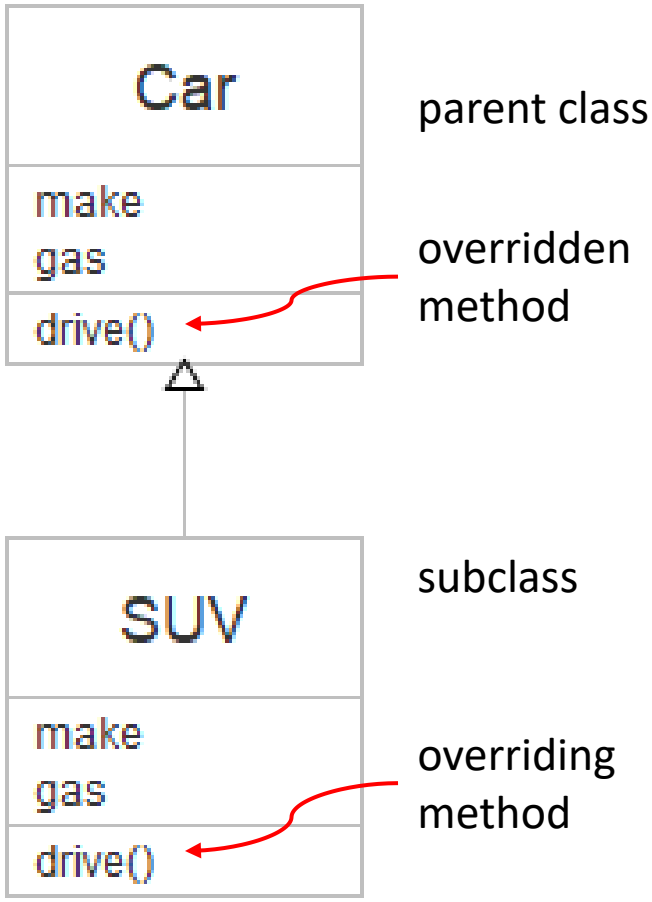
- **Allows classes to derive from other classes.**



With inheritance, reusability is a major advantage. You can reuse the fields and methods of the existing class.

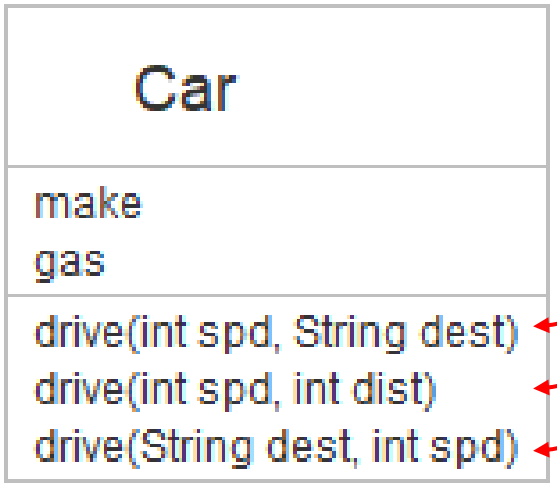
Polymorphism

- **Describes methods that can take on many forms.**
  - **Dynamic** polymorphism: (a.k.a. *method overriding*)
    - Occurs during the **runtime** of the program.
    - The methods share the same name but have different implementation.
    - The implementation of the subclass that the object is an instance of overrides that of the superclass.



Polymorphism

- **Static** polymorphism: (aka *method overloading*)
  - Occurs during the **compile-time**.
  - Multiple methods with the same name but different arguments are defined in the same class.



Ways to differentiate methods of the same name:

```
myCar.drive(45, "NPU");
myCar.drive(45, 100);
myCar.drive("Home", 50);
```

Be sure that you are calling the correct form of the method.

Object-oriented vs Procedural

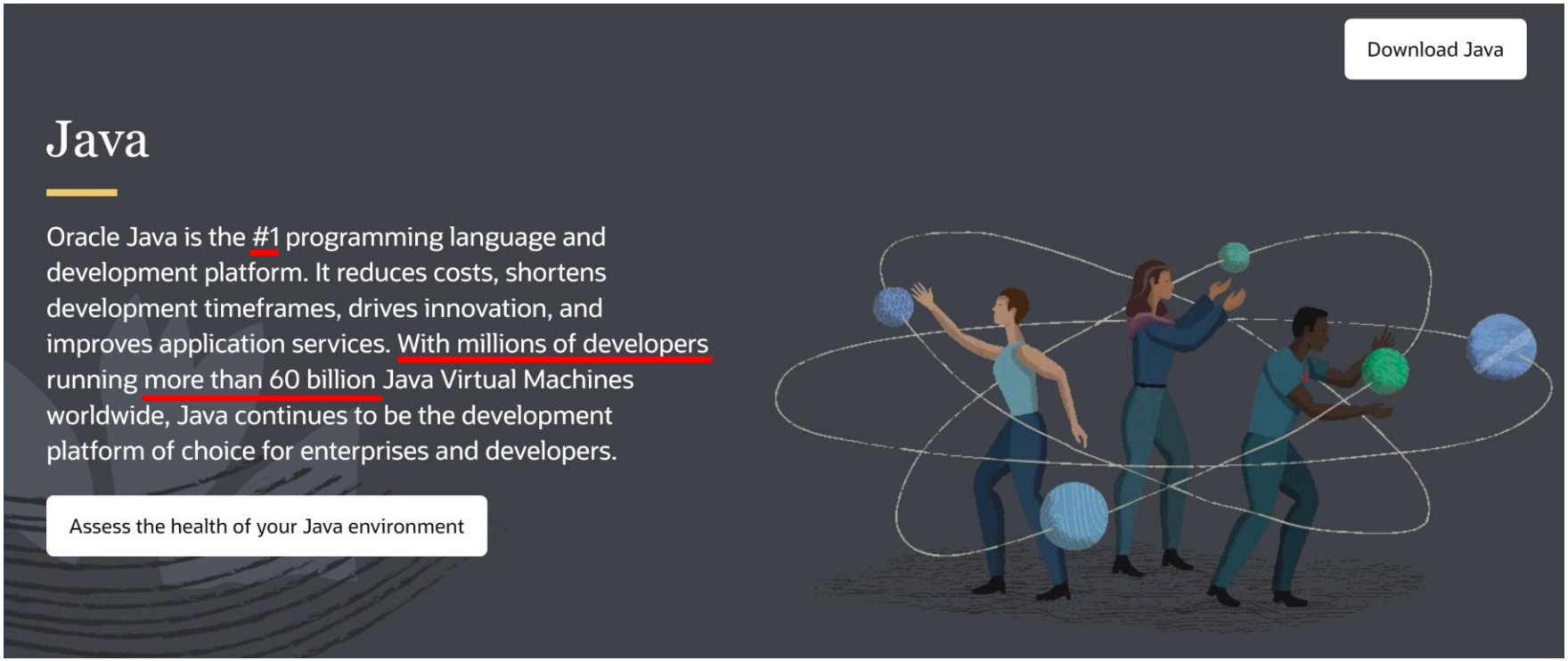
Paradigm	Description	Pros	Cons	Examples
Object-oriented	Treats data fields as *objects* manipulated through predefined methods only	1. Much easier to scale for future needs and development. 2. Good for larger more complex applications. 3. More dynamic and fluid in terms of the architecture and overall design. 4. Maintainable.	1. Can easily become very complicated in terms of design and architecture. 2. Takes much longer to develop initially. 3. More difficult to learn than Procedural.	**Java**, C++, Kotlin, Go, Python, etc.
Procedural	Derived from structured programming, based on the concept of *modular programming* or the *procedure call*	1. Quick to develop and implement. 2. Easy to learn. 3. Simple architecture and overall structure. 4. Good for quick and simple applications.	1. Difficult to scale for future needs. 2. Usually is very flat in terms of design and structure. 3. Not good for larger applications that will likely change over time. 4. Maintaining can be very challenging.	**C**, C++, PHP, Python, etc.

Understand the design decisions that shaped Java

What is Java?

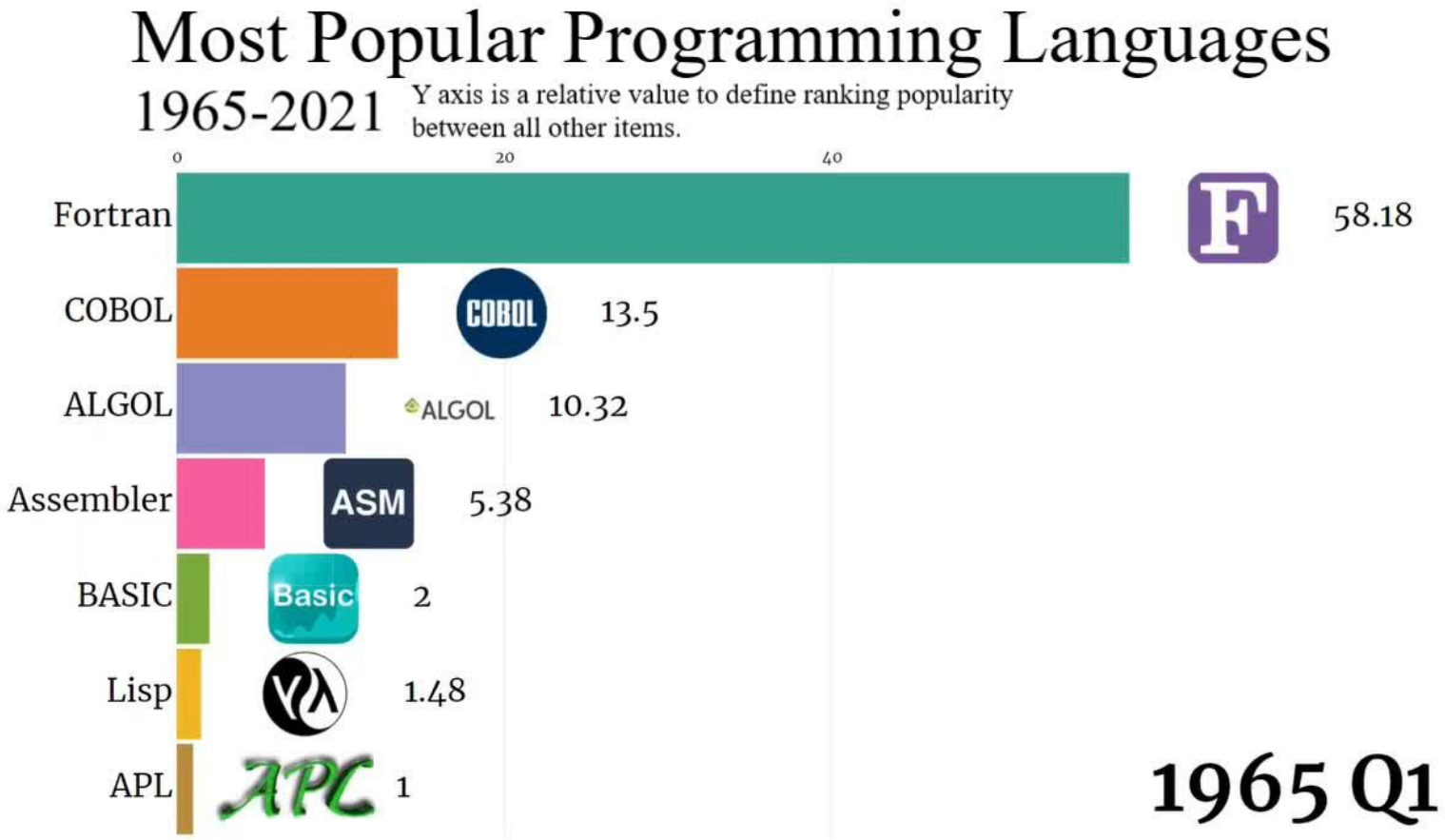
- Java is a **class-based**, **object-oriented** programming language that is designed to have as few implementation dependencies as possible.
- It is intended to let application developers **write once, run anywhere** (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.

Java is Popular



https://go.java/?intcmp=gojava-banner-java-com

A short video (2 min)



The "White Paper" buzzwords

1. Simple
2. Object-Oriented (OO)
3. Distributed
4. Robust
5. Secure
6. Architecture-Neutral
7. Portable
8. Interpreted
9. High-Performance
10. Multithreaded
11. Dynamic

The authors of Java wrote an influential white paper that explains their design goals and accomplishments.

“Simple”

- Java has an English-like syntax, which makes it the perfect language for beginners.
  - Compared with C++, the syntax of Java is much easier and more comprehensible.
  - There is no need for header files, pointer, structures, etc.

- Java is relatively small.
  - Initially designed for small machines;
  - The size of the basic interpreter and class support is about 40KB;
  - The basic standard libraries and thread support add another 175KB.

"Object-Oriented"

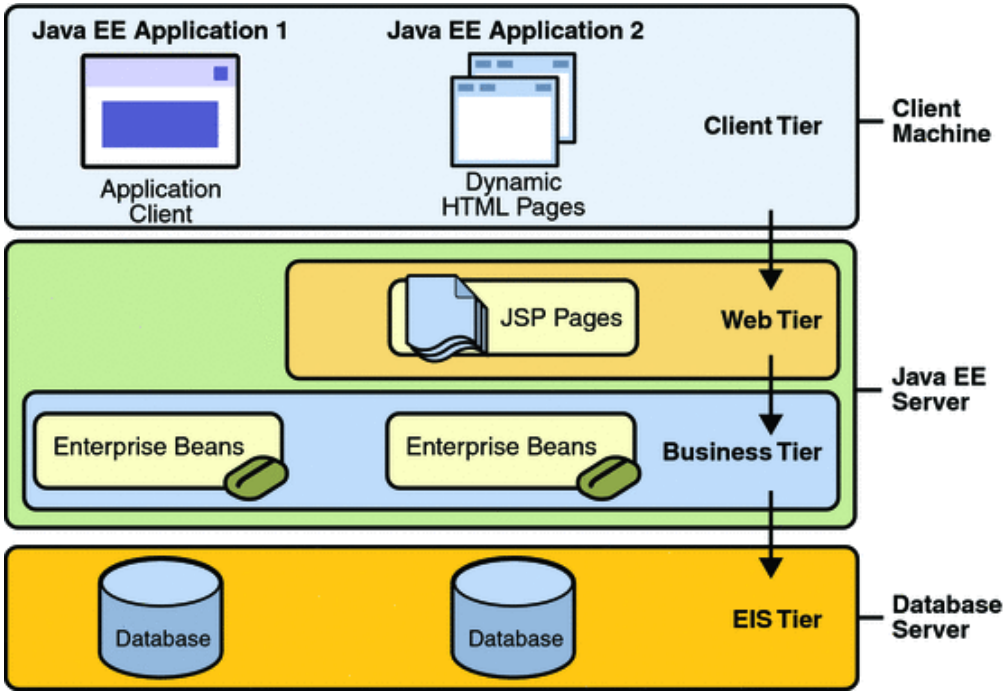
- Simply stated, object-oriented design is a programming technique that focuses on the data-objects-and on the interfaces to those objects.

- **Object orientation was pretty well established when Java was developed.**

- The object-oriented features of Java are comparable to those of C++.
  - The major difference between Java and C++ lies in multiple inheritance, which Java has replaced with a simpler concept of interfaces.

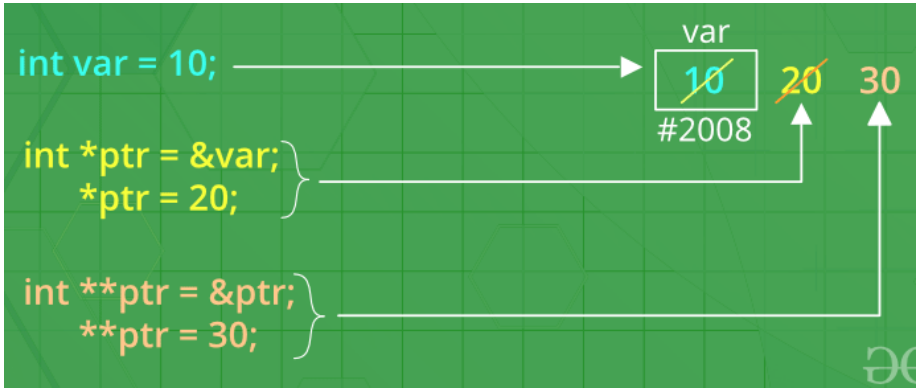
"Distributed"

- Java has an extensive library of routines for coping with TCP/IP protocols like HTTP and FTP.
  - Java applications can open and access objects across the Net via URLs with the same ease as when accessing a local file system.



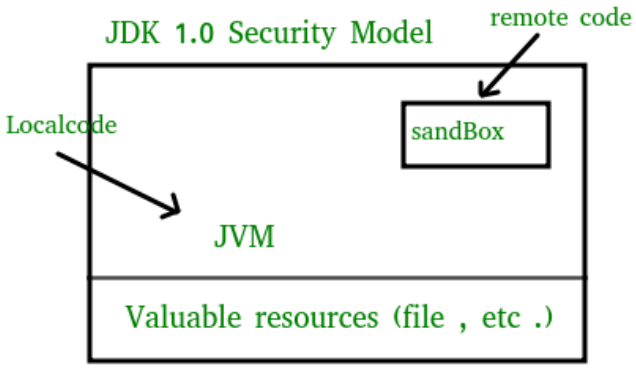
“Robust”

- Java is intended for writing programs that must be reliable in a variety of ways.
  - Java detects many problems that in other languages would show up only at runtime.
  - Java puts a lot of emphasis on early checking for possible problems, later dynamic (runtime) checking, and eliminating situations that are error-prone. . . .
- The single biggest difference between Java and C/C++ is:
  - Java has a pointer model that eliminates the possibility of overwriting memory and corrupting data.



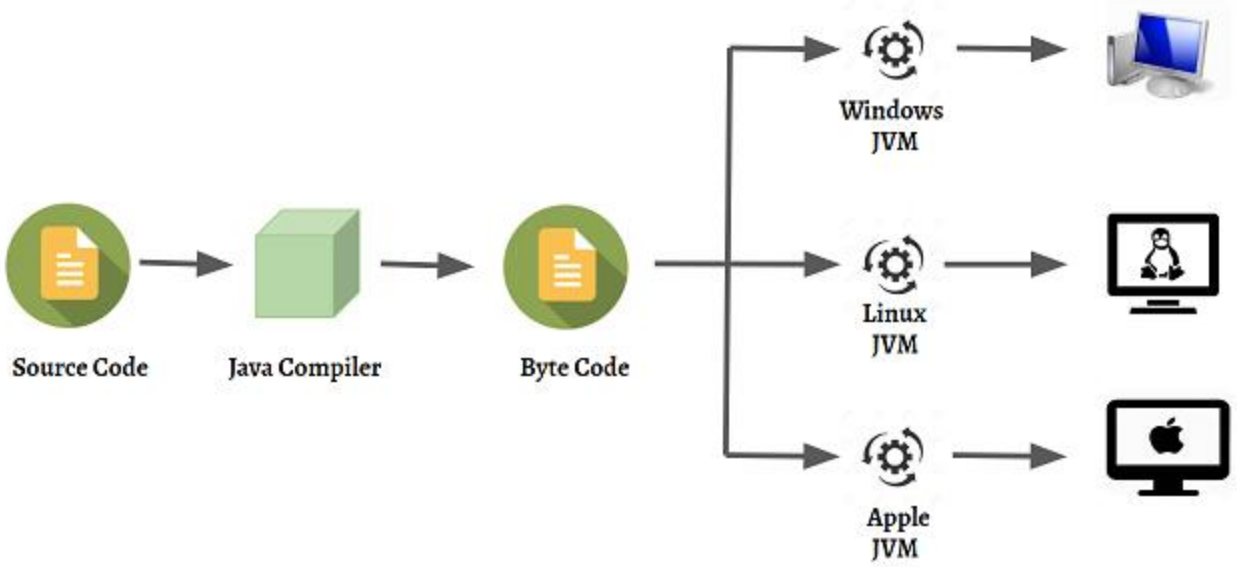
“Secure”

- Java was designed to make some kinds of attacks impossible:
  - Overrunning the runtime stack - a common attack of worms/viruses
  - Corrupting memory outside its own process space
  - Reading or writing files without permission
- The Java security model is based on a customizable “**sandbox**” in which Java programs can run safely, without potential risk to systems or users.
  - Nothing bad could happen because Java code, no matter where it came from, could never escape from the sandbox.



“Architecture-Neutral”

- The compiler generates an architecture-neutral object file format.
  - The Java compiler does this by generating bytecode instructions which have nothing to do with a particular computer architecture.
  - Rather, they are designed to be both easy to interpret on any machine and easy to translate into native machine code on the fly.



Quick question 2

Which OS are you using?

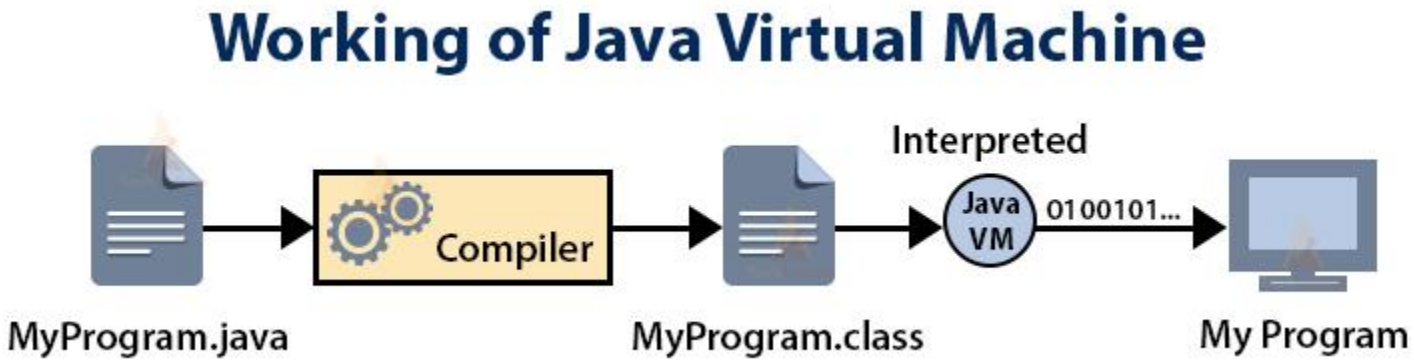
A. Windows
B. macOS
C. Linux
D. Other

"Portable"

- Unlike C and C++, there are no "implementation-dependent" aspects of the specification. The sizes of the primitive data types are specified, as is the behavior of arithmetic on them.
  - For example, an `int` in Java is always a 32-bit integer. In C/C++, `int` can mean a 16-bit integer, a 32-bit integer, or any other size that the compiler vendor likes. The only restriction is that the `int` type must have at least as many bytes as a `short int` and cannot have more bytes than a `long int`.
  - Having a fixed size for number types eliminates a major porting headache. Binary data is stored and transmitted in a fixed format, eliminating confusion about byte ordering. Strings are saved in a standard Unicode format.

“Interpreted”

- The Java interpreter can execute Java bytecodes directly on any machine to which the interpreter has been ported.
- Since linking is a more incremental and lightweight process, the development process can be much more rapid and exploratory.

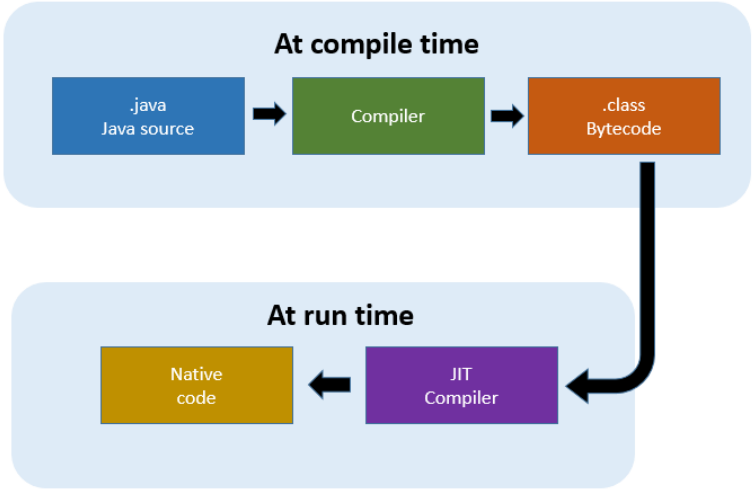


Compiling vs Interpreting (6 min)

https://www.youtube.com/watch?v=JNMy969SjyU

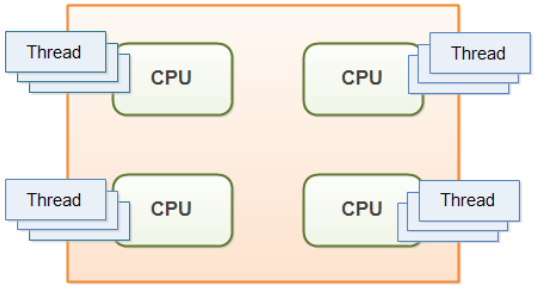
“High-Performance”

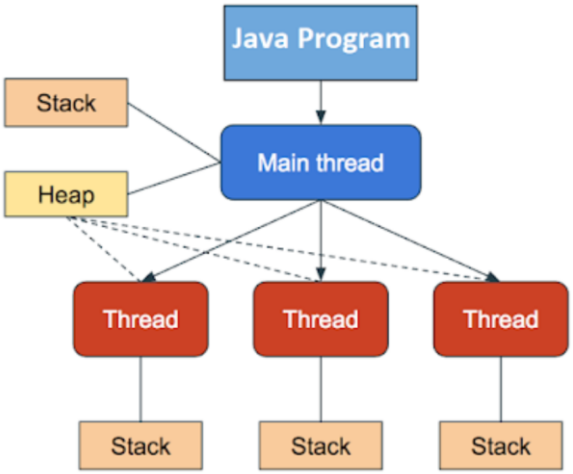
- While the performance of interpreted bytecodes is usually more than adequate, there are situations where higher performance is required. The bytecodes can be translated on the fly (at runtime) into machine code for the particular CPU the application is running on.
- Today, however, **the just-in-time compilers** have become so good that they are competitive with traditional compilers and, in some cases, even outperform them *because they have more information available*.
  - For example, a just-in-time compiler can monitor which code is executed frequently and optimize just that code for speed.



“Multithreaded”

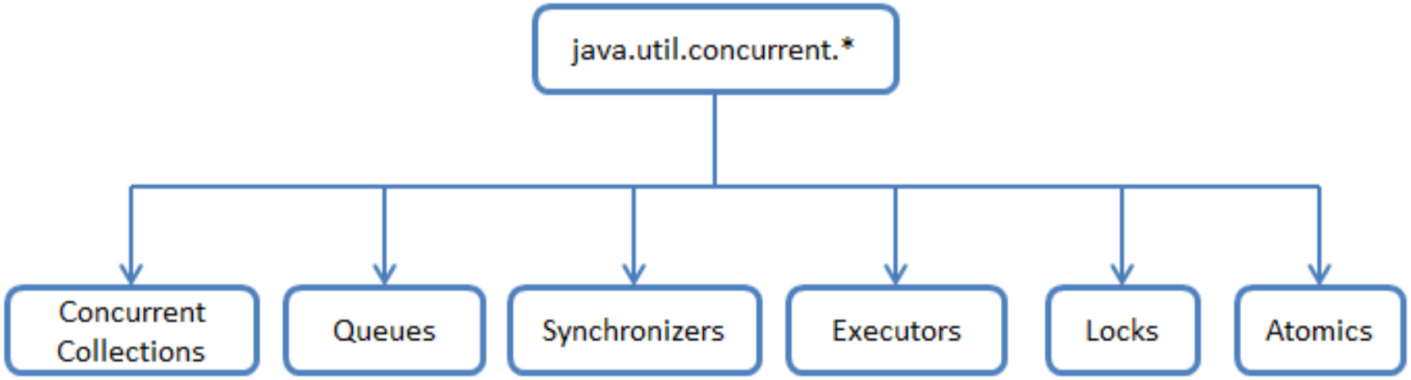
- A thread is an independent path of execution within a program, executing concurrently.
- Multithreaded means handling multiple tasks simultaneously or executing multiple portions (functions) of the same program in parallel.
- The code of java is divided into smaller parts and Java executes them in a sequential and timely manner.
- Advantages:
  - Maximizing utilization of resources.
  - Doesn’t occupy memory for each thread. It shares a common memory area.
  - No need to wait for the application to finish one task before beginning another one.
  - Decreased cost of maintenance and time-saving.
  - Improves the performance of complex applications.





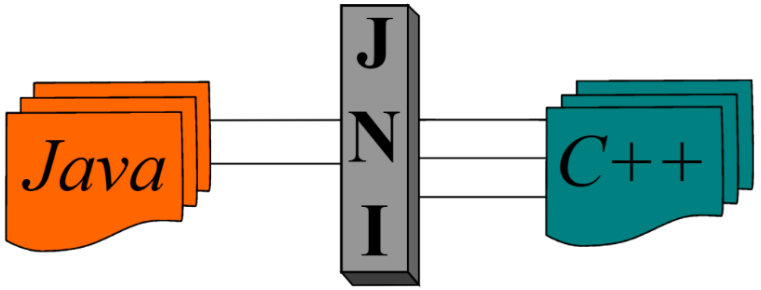
“Multithreaded”

- Java was well ahead of its time. It was the first mainstream language to support concurrent programming.
  - At the time, multicore processors were not widely deployed, but web programming had just started, and processors spent a lot of time waiting for a response from the server. Concurrent programming was needed to ensure the user interface didn’t freeze.
  - Concurrent programming is never easy, but Java has done a very good job making it manageable.



“Dynamic (and Extensible)”

- With the help of OOPs, we can add classes and add new methods to classes, creating new classes through subclasses.
  - This makes it easier for us to expand our own classes and even modify them.
- Java gives the facility of dynamically linking new class libraries, methods, and objects.
  - It is highly dynamic as it can adapt to its evolving environment.
- Java even supports functions written in other languages such as C and C++ to be written in Java programs.
  - These functions are called “native methods”. These methods are dynamically linked at runtime.



Why is Java popular?

1. **Java is user-friendly**
   - English-like syntax
2. **Java for everything**
   - Can be used for developing Web apps, Android apps, etc.
   - Can be used in Data Science applications, Machine Learning applications, and even IoT.
3. **Java has rich API**
4. **A robust community backs Java**
5. **Java has excellent documentation**
6. **Java has a suite of powerful development tools**

Become familiar with the history of Java

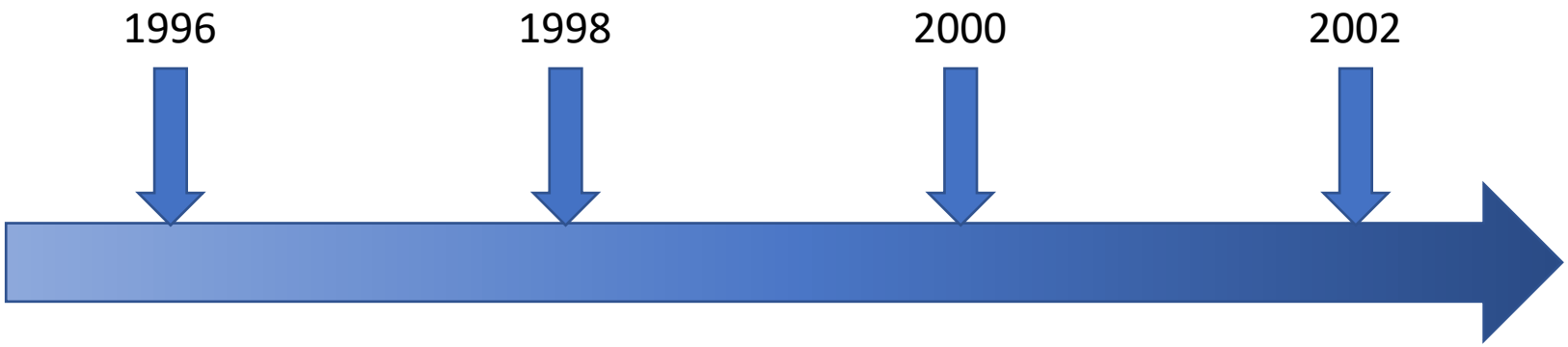
Before Java 1.0

- 1991
  - James Gosling worked on "Project Green", a system for consumer devices.
  - He designed a programming language, originally called "**Oak**".
  - That name was trademarked, so it was renamed to "**Java**".
- 1992
  - The first project was released, a TV switchbox called "*7".
  - Nobody cared, and the project was renamed "First Person, Inc."
- 1994
  - Still, nobody cared, and Gosling realized that they could build a "really cool browser…architecture-neutral, real-time, reliable, secure."
- 1995
  - The HotJava browser was released.
- 1996
  - Java 1.0 was released!

History of Java (2 min)

https://www.youtube.com/watch?v=DcQPtlFlgzY&t=72s

Java versions



JDK 1.0

Very first version was released on 23 Jan 1996. The principal stable variant, JDK 1.0.2, is called Java 1. JDK 1.1 was released on 19 Feb 1997.

J2SE 1.2

"Playground" was the codename which was given to this form and was released on 8 Dec 1998. Its real expansion included: *strictfp* keyword, the Swing graphical API.

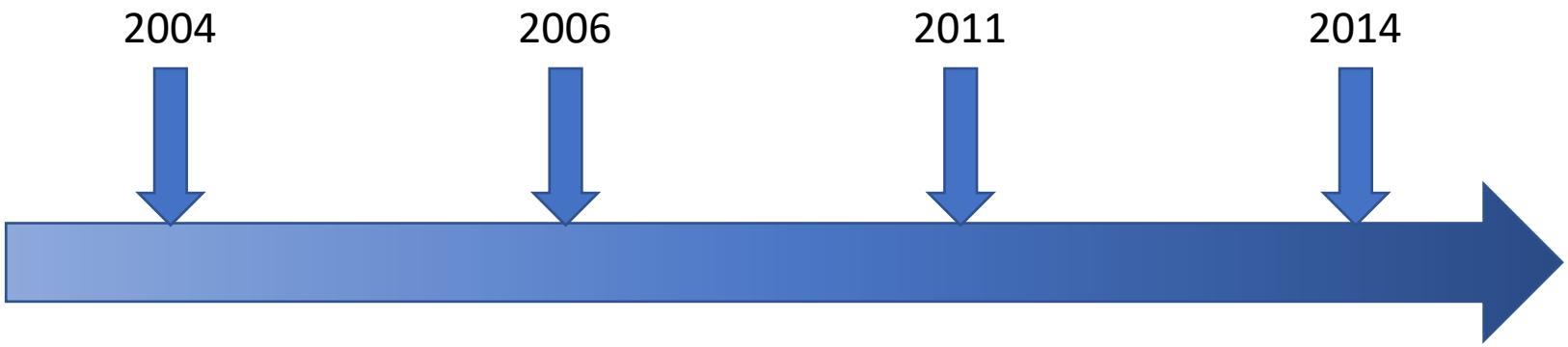
J2SE 1.3

Was given a codename "Kestrel" and was released date 8 May 2000 and contains additions like HotSpot, JVM included, Java Naming and Directory Interface.

J2SE 1.4

Was given the codename "Merlin" and was released on date 6 Feb 2002 and contains additions like Library improvements, Regular expressions modelled after Perl regular expressions.

Java versions



J2SE 5.0
Was given the codename "Tiger" and was released on 30 Sep 2004 originally numbered as 1.5 which is still used as its internal version. Added several new language features such as *for-each* loop.

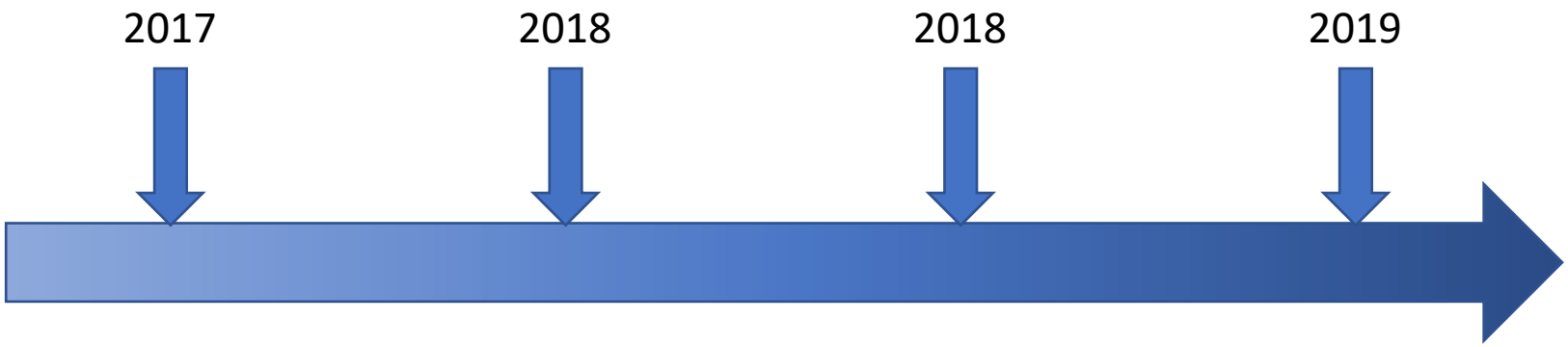
Java SE 6
Was given the codename "Mustang" and was released on date 11 Dec 2006. Packaged with a database supervisor and encourages the utilization of scripting.

Java SE 7
Was given the codename "Dolphin" and was released on date 7 Jul 2011. Added small language changes including strings in switch. The JVM was extended with support for dynamic languages.

Java SE 8 (LTS)
Was released on date 18 Mar 2014. Language level support for lambda expressions and default methods and a new date and time API inspired by Joda Time.

Long-term support

Java versions



Java SE 9

Was released on 21 Sep 2017. Project Jigsaw: designing and implementing a standard, module system for the Java SE platform, and to apply that system to the platform itself and the JDK.

Java SE 10

Was released on 20 Mar 2018 contains additions like Additional Unicode language-tag extensions, Rot certificates, Thread-local handshakes, Heap allocation on alternative memory devices.

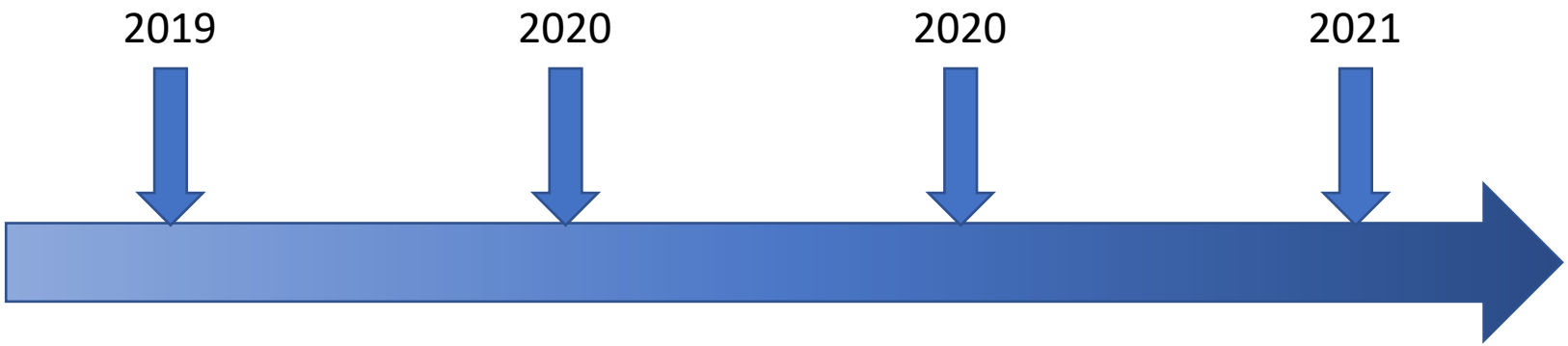
Java SE 11 (LTS)

Was released on 25 Sep 2018 contains additions like Dynamic class-file constants, Epsilon: a no-op garbage collector. Local-variable syntax for lambda parameters. Low-overhead heap profiling.

Java SE 12

Was released on 19 Mar 2019 contains additions like Shenandoah: A Low-Pause-Time Garbage Collector(Experimental), Microbenchmark Suite, Switch Expressions (Preview), JVM Constants API.

Java versions



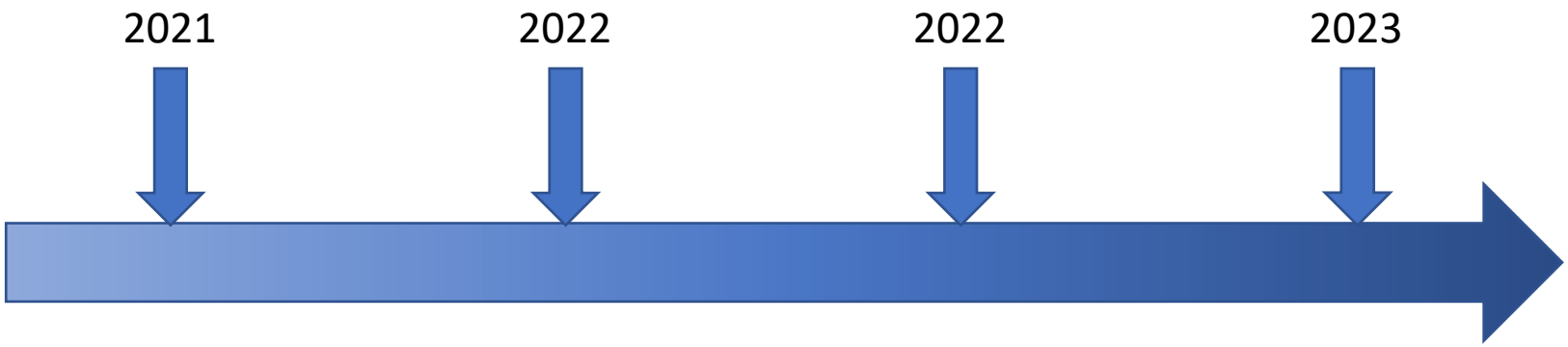
Java SE 13
Was released on 17 Sep 2019. It includes the following new features like Dynamic CDS Archives, ZGC: uncommit unused memory, reimplement the legacy socket API.

Java SE 14
Was released on 17 Mar 2020. It includes the following new features like pattern matching for instanceof, packaging tool, JFR event streaming, non-volatile mapped byte buffers.

Java SE 15
Was released on 15 Sep 2020. It adds support for multi-line string literals. The Shenandoah and Z garbage collectors(latter sometimes abbr. ZGC) are now ready for use in production.

Java SE 16
Was released on 16 Mar 2021. It removes Ahead-of-Time compilation options, enables C++ 14 language features and the source code of Java is migrated to GitHub.

Java versions



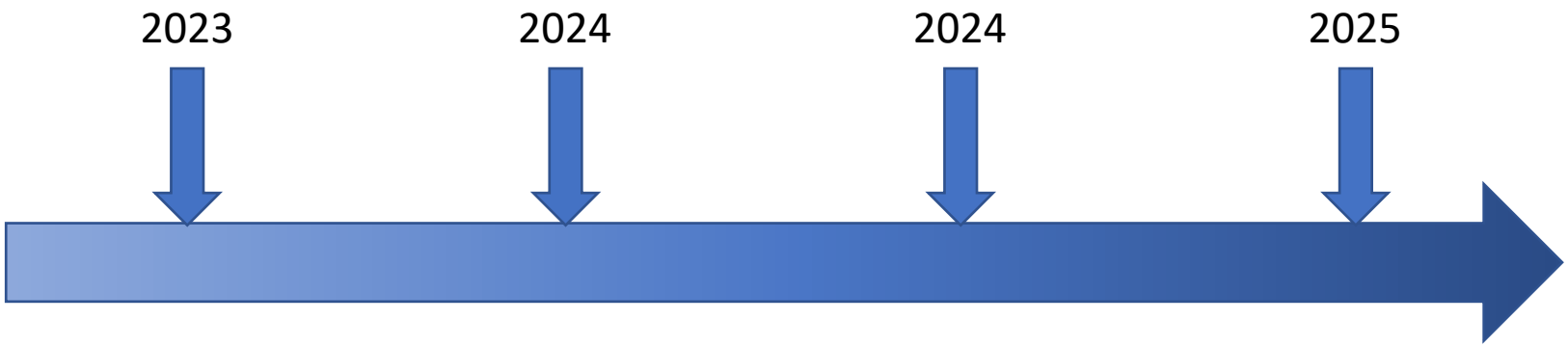
Java SE 17 (LTS)
Was released on 14 Sep 2021. It contains features like always-strict floating point semantics, a uniform API for pseudo random number generators, and much more.

Java SE 18
Was released on 22 Mar 2022. Set UTF-8 as the default encoding. A new command jwebserver is provided for testing,education, demonstration and other needs.

Java SE 19
Was released on 20 Sep 2022. The record pattern extends pattern matching and expresses more complex data queries. Gradually introduce virtual threads.

Java SE 20
Was released on Mar 2023. Seven JDK Enhancement Proposals (JEPs) updates and improvements are provided. Most of the updates are to improve features introduced in earlier versions.

Java versions



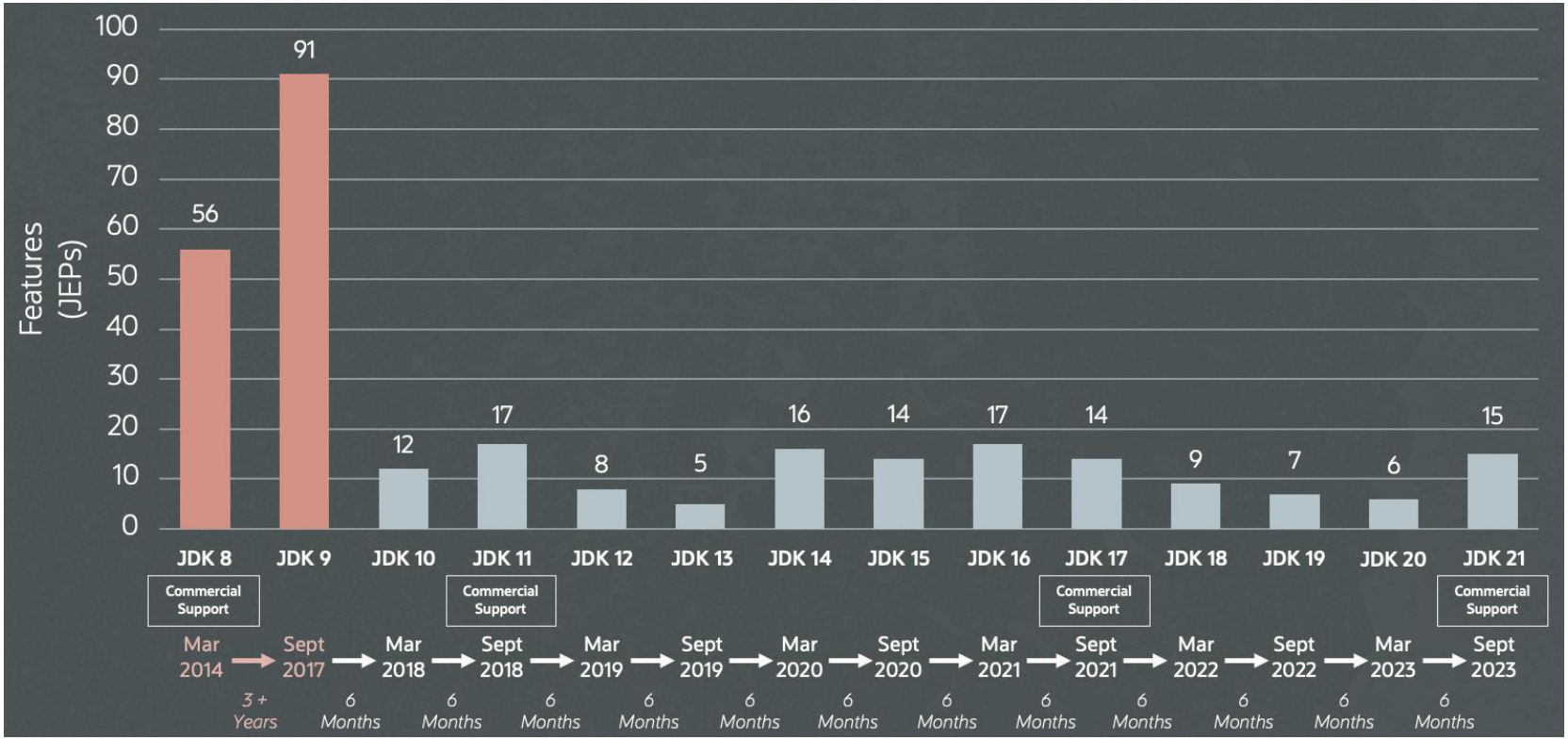
Java SE 21 (LTS)
Was released on 19 Sep 2023. Thousands of updates for performance, stability, and security, plus many new features. 15 enhancements have separate JEPs, covering six previews and one incubator feature.

Java SE 22
Was released on 19 Mar 2024. It delivers 12 enhancements that are significant enough to warrant their own JEPs, including seven preview features and one incubator feature.

Java SE 23
Was released on 17 Sep 2024. It delivers 12 JEPs updates and improvements. The Oracle JDK now includes the Oracle GraalVM JIT compiler as one of its available JIT compilers.

Java SE 24
Will be released in Mar 2025.
...

Java's six-month release cadence



https://blogs.oracle.com/java/post/the-arrival-of-java-21

Currently supported releases

Version	Initial Release	Current Release	Version Info	End of Life
21	2023-09-19	21.0.6 2025-01-21	Documentation Configurations Risk Matrix JSR 396	2031-09-30
17	2021-09-14	17.0.1 2021-10-19	Open JDK Project Page JSR 392	2029-09-30
11	2018-09-25	11.0.13 2021-10-19	Release Notes Documentation Certified Configurations Risk Matrix Open JDK Project Page JSR 384	2026-09-30
8	2014-03-18	8u311 2021-10-19	Release Notes Documentation Certified Configurations Risk Matrix JSR 337	2030-12-31

More information on https://www.java.com/releases/

Let’s see common misconceptions about Java

Misconceptions about Java

- "Java is an extension of HTML or XML."
  - Java is a programming language.
  - HTML is a way to describe the structure of a web page.
  - XML is a way to describe data.
- Java and HTML have nothing in common except that there are HTML extensions for placing Java applets on a web page.
- You can process XML data with any programming language, but the Java API contains excellent support for XML processing. In addition, many important XML tools are implemented in Java.

Misconceptions about Java

- "Java is an easy programming language to learn."
  - No programming language as powerful as Java is easy.
  - You always have to distinguish between how easy it is to write toy programs and how hard it is to do serious work.

https://www.google.com/url?sa=i&url=http%3A%2F%2Fusedsportscars.co.uk%2Ftoy-vw-lupo-vs-real-vw-lupo%2F&psig=AOvVaw340J2dNlWxx6JAEYOC4l1T&ust=1615608090046000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCOi_wKrvqe8CFQAAAAAdAAAAABAJ

Misconceptions about Java

- "Java will become a universal programming language for all platforms."
  - This is possible in theory. But in practice, there are domains where other languages are entrenched.
    - **Objective C** and its successor, **Swift**, are not going to be replaced on iOS devices.
    - Anything that happens in a browser is controlled by **JavaScript**.
    - Windows programs are written in **C++** or **C#**.
    - **Java** has the edge in server-side programming and in cross-platform client applications.

Misconceptions about Java

- "Java is just another programming language."
  - The success of a programming language is determined mostly by the utility of its surrounding system:
    - *Are there useful, convenient, and standard libraries for the features that you need to implement?*
    - *Are there tool vendors that build great programming and debugging environments?*
    - *Do the language and the toolset integrate with the rest of the computing infrastructure?*
  - **Java is successful** because its libraries let you easily do things such as **networking, web applications, and concurrency**.
    - The fact that Java reduces pointer errors is a bonus, so programmers seem to be more productive with Java.

Misconceptions about Java

- "Java is proprietary and should be avoided."
  - When Java was first created, **Sun** gave free licenses to distributors and end users.
    - Source code for the virtual machine and the libraries has always been freely available, but only for inspection, not for modification and redistribution.
    - Java was "closed source but playing nice."
  - In 2007, **Sun** announced that future versions of Java would be available under the General Public License (GPL), the same open-source license that is used by Linux.
    - **Oracle** has committed to keeping Java open source.
    - Everyone is given a patent grant to use and modify Java, subject to the GPL, but only on desktop and server platforms.
    - If you want to use Java in embedded systems, you need a different license and will likely need to pay royalties.
    - **However, these patents will expire within the next decade, and at that point Java will be entirely free.**

Misconceptions about Java

- "Java is interpreted, so it is too slow for serious applications."
  - In the early days of Java, the language was interpreted.
  - Nowadays, the Java virtual machine uses a **just-in-time compiler**.
    - The "hot spots" of your code will run just as fast in Java as they would in C++, and in some cases even faster.

Misconceptions about Java

- "All Java programs run inside a web page."
  - All Java applets run inside a web browser.
    - That is the definition of an applet-a Java program running inside a browser.

Java applets were deprecated since Java 9 in 2017 and removed from Java SE 11 (18.9), released in September 2018.

  - But most Java programs are stand-alone applications that run outside of a web browser.
    - In fact, many Java programs run on web servers and produce the code for web pages.

Misconceptions about Java

- "Java programs are a major security risk."
  - In the early days of Java, there were some well-publicized reports of failures in the Java security system.
    - The technical failures that they found have all been quickly corrected.
  - Later, there were more serious exploits, to which Sun, and later Oracle, responded too slowly.
    - Browser manufacturers reacted, and perhaps overreacted, by deactivating Java by default.
  - **Even 20 years after its creation, Java is far safer than any other commonly available execution platform.**

Misconceptions about Java

- "JavaScript is a simpler version of Java."
  - JavaScript, a scripting language that can be used inside web pages, was invented by Netscape and originally called LiveScript.

JAVA	Java Script
• Compiled	• Interpreted
• Mainly used for back-end	• Mainly used for front-end
• Executed in JVM or in the browser	• Executed in the browser
• Allows better security	• Needs more effort to enhance security
• Static type checking	• Dynamic type checking
• The syntax is similar to C++	• The syntax is similar to C
• Requires Java Development Kit (JDK)	• Can be written in any text editor
• For various apps	• Mainly for web apps

Misconceptions about Java

- "With Java, I can replace my desktop computer with a cheap 'Internet appliance'."
  - When Java was first released, some people bet big that this was going to happen.
  - Companies produced prototypes of Java-powered network computers, but users were not ready to give up a powerful and convenient desktop for a limited machine with no local storage.
  - Nowadays, of course, the world has changed, and for a large majority of end users, the platform that matters is a mobile phone or tablet.
    - The majority of these devices are controlled by the Android platform, which is a derivative of Java.
    - Learning Java programming will help you with Android programming as well.

Recap

- What is Object Oriented Programming (OOP)?
  - OOP is a programming paradigm based on the concept of "**objects**", which can contain data and code:
    - **data**, in the form of fields (a.k.a. *attributes* or *properties*);
    - **code**, in the form of procedures (a.k.a. *methods*).

- Four main principles of OOP:
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism

- Java!