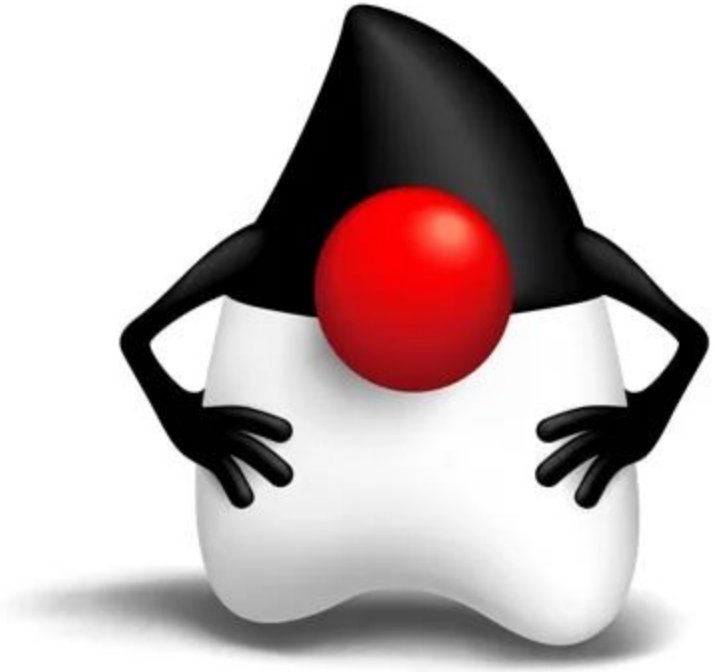
Java

Introduction to Object Oriented Programming

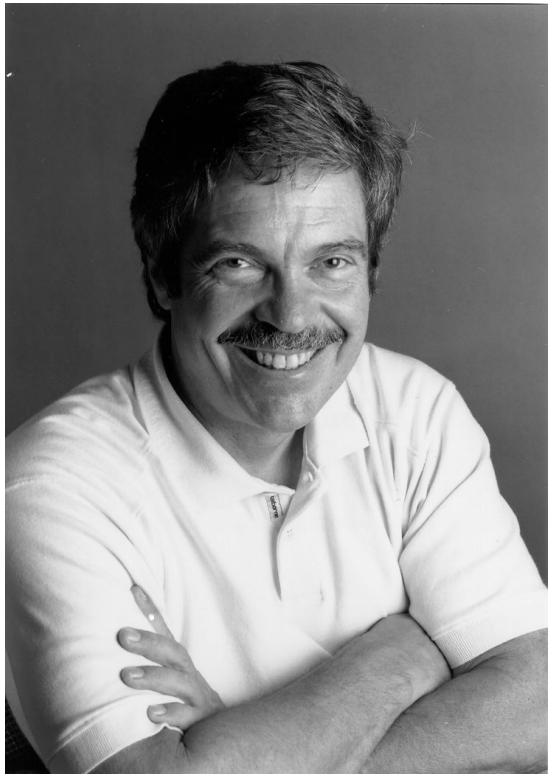
Duke



write once, run anywhere

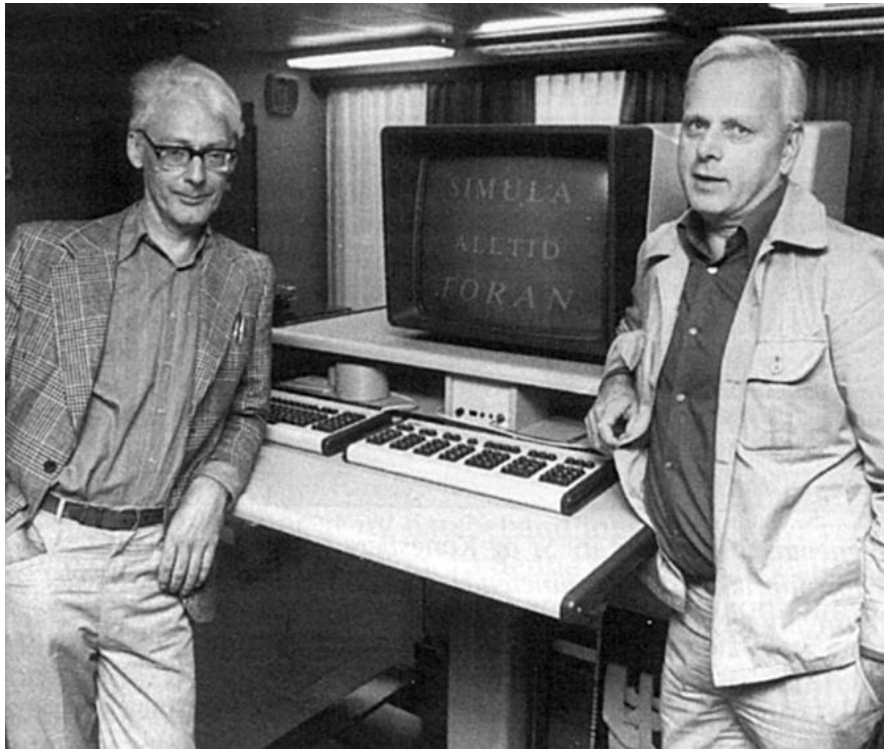
Alan Kay

- Coined the term Object Oriented.
- Key figure behind **Smalltalk**
  - Pure OOP language for educational use
- Emphasized objects as **independent entities** that:
  - Contain both **data** and **behavior**
  - Interact through **message passing** (not shared memory)
- Big idea is **messaging**
  - Focused on *communication* between objects rather than classes / inheritance.
- OOP should model **real world systems**



Simula 67 (Norway, 1967)

- **Created by**
  - Ole-Johan Dahl & Kristen Nygaard
- **Goal**
  - Model real-world systems like ships, traffic, and queues.
- First language to introduce *classes, objects, inheritance*.
- **Main idea**
  - Software objects can represent real-world entities.



Smalltalk (1972-1980s)

- **Created by**
  - Alan Kay, Adele Goldberg, Dan Ingalls @ Xerox PARC
- **Vision**
  - Everything is an object
- First environment with GUI, mouse, etc.
  - Inspired Mac, Windows
- **Trivia**
  - **Alan Kay:** OOP was more about messages between objects than about *classes/inheritance*.
  - Influenced Python, Ruby, Objective C.
- **Fun story**
  - Xerox PARC demoed their GUI + OOP to Steve Jobs → inspired Apple Lisa/Mac.

C with Classes (1979-1983)

- **Created by**
  - Bjarne Stroustrup @ Bell Labs
- **Info**
  - Added classes, inheritance, polymorphism to C.
  - Renamed to **C++** in 1983.

Oak (1991-1995

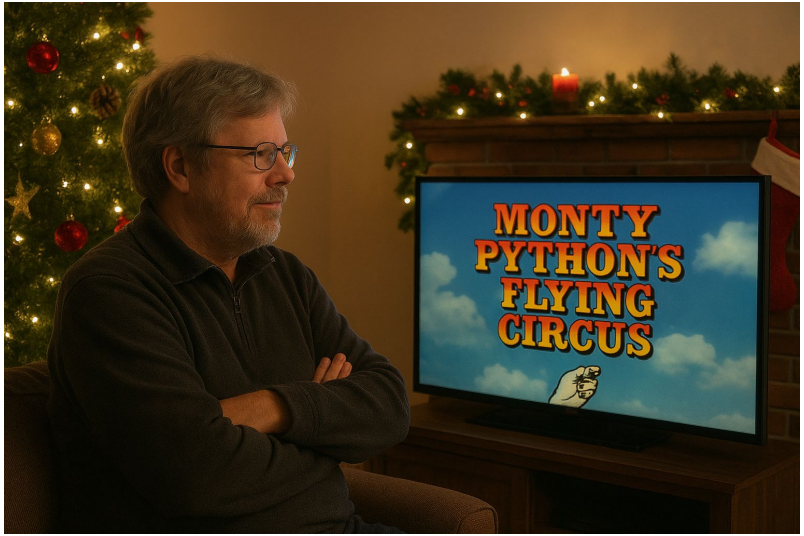
- **The Green Project (1991)**
  - James Gosling, Mike Sheridan, Patrick Naughton @ *Sun Microsystems*
- **Goal**
  - Build software for consumer electronics (set-top boxes, PDAs, interactive TVs)
- **Oak**
  - *Gosling* designed a new programming language for this project.
  - Simple, OO, Multiplatform, Memory safe.

Java (1994 - ..)

- **Problem**
  - Oak name was trademarked by Oak Technologies.
- **Solution**
  - Named Java, after Java coffee.
  - Aim was to select a short, snappy and fun name.
- **Life**
  - Officially released in May 1995.
  - Java Applets
  - Write once, run anywhere
- **Killer apps**
  - Netscape browser with embedded JVM to run applets.
  - Called the “Language of the internet”

Python (1989-1991)

- Created by
  - Guido van Rossum
-



Compiled vs. Interpreted

- Source code → **Compiler** → Machine code
- Program runs directly on hardware.
- **Pros:**
  - High performance & efficiency
  - Optimizations at compile-time
- **Cons:**
  - Longer dev cycle (compilation time)
  - Platform-dependent binaries
- **Ex:**
  - C, C++, Rust, etc.

- Source code → **Interpreter** → Executes line by line
- No compilation step
- **Pros:**
  - Easy to test & debug
  - Cross-platform
- **Cons:**
  - Slower execution
  - Less optimization
- **Ex:**
  - Python, Javascript, Ruby, etc.

VM: Virtual Machine (link)

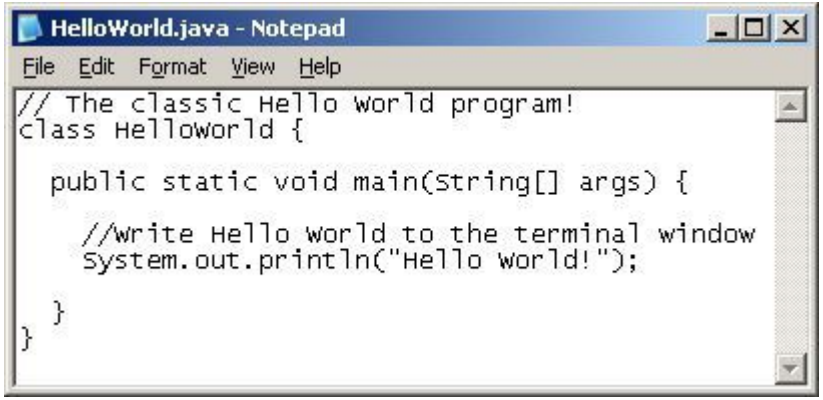
- Software-based emulation of a computer system.
  - Provides an **abstraction layer** between hardware and software.
  - Runs programs as if they were on a **real physical machine**.
- Examples
  - JVM (Java family)
  - CLR (C# family)
  - Python VM

JVM

- Java code is not compiled into **machine code**
  - Compiled into **bytecode**
- This bytecode is specific to JVM.
  - JVM can run it!
- If we have JVM implementation, we can run the code.
  - That means multiplatform!

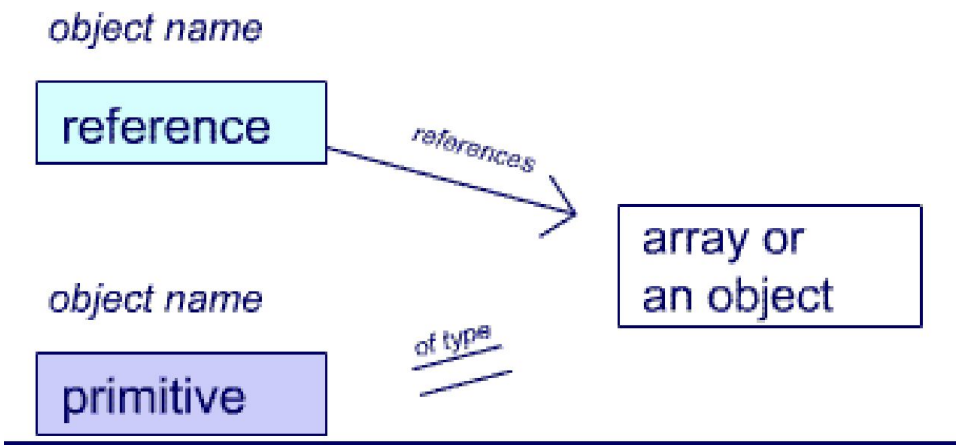
Your First Java Program

- Multiple IDE's
  - Eclipse, JetBrains, IntelliJ
  - Can use VS Code as well.
- Save the file as HelloWorld.java
- Compile it using **javac**
  - Becomes *HelloWorld.class*
- Run by using **java**
  - **java HelloWorld**
- IDE does it automatically.



Variable

- Reference
  - Points to an instance of a *class* or *array*
- Primitive
  - byte, short, int, float, char, boolean etc.




```
typename identifier = initial_value;
```

Or its value can be set after its declaration as:

```
typename identifier;
...
identifier = initial_value;
...
```

Scope

- A variable is accessible within its scope.
- for(int i=0;i<10;i++)
- a variable declared in the function
- a variable declared outside the function
- lets see some examples

Primitive Data Types

- 3 types
- Numeric
  - Integer or decimal
- Boolean
- Character

Primitive Data Types

Category	Type Name	Size	Format / Range of Values
Integers	`byte`	1 byte integer	2's complement `-128` to `127`
	`short`	2 byte integer	2's complement -2^{15} to $2^{15}-1$
	`int`	4 byte integer	2's complement -2^{31} to $2^{31}-1$
	`long`	8 byte integer	2's complement -2^{63} to $2^{63}-1$
Decimal Numbers	`float`	4 byte floating point number	IEEE 754
	`double`	8 byte real number	IEEE 754
Characters	`char`	2 byte Unicode character	Unicode `0` to Unicode $2^{16}-1$
Booleans	`boolean`	-	`true` or `false`

Literals

- Actual representation of a *number*, a *character*, a *boolean* or a *string*.

Literals

Type	Format	Examples
`int`	A sequence of digits `0-9` (`0-9,A,B,C,D,E,F` characters for hexadecimal and `0-7` characters for octal)	`123` `0123` (octal-base 8) `0x123` (hexadecimal - base 16)
`long`	A sequence of digits followed by a `l` or `L` letter.	`123L`

`double`	A sequence of digits with one decimal point symbol `.` and/or `e` (or `E`) letter. The literal may be followed by `d` or `D` letter.	`123D`
		`1.23`
		`1.23D`
		`1.2E-3`
		`12.3E4D`

`float`	>A sequence of digits with one decimal point symbol `.` and/or `e` (or `E`) letter. The literal must be followed by `f` or `F` letter.	`123F`
		`1.23F`
		`1.2E-3F`
		`1.2E3F`

`boolean`	`true` or `false`	`true` `false`
`char`	A character in single quotes	`'a'`
`String`	A sequence of characters in double quotes	`"abc"`

Arithmetic operators

Expression	Function
`++A`	Pre increment: This expression is equivalent to `A=A+1`, and the result is equal to `A+1`.
`--A`	Pre decrement: This expression is equivalent to `A=A-1`, and the result is equal to `A-1`.
`A++`	Post increment: The result is equal to A, but after the expression evaluated value of `A` is set to `A+1`.
`A--`	Post decrement: The result is equal to A, but after the expression evaluated value of `A` is set to `A-1`.
`+A`	Promotion: If `A` is of type `byte` or `short`, the result will be `A` of type `int`.
`-A`	Negation: Result is negative of `A`.
`A+B`	The result is the sum of `A` and `B`.
`A-B`	The result is `B` subtracted from `A`.
`A*B`	The result is `A` multiplied by `B`.
`A/B`	The result is `A` divided by `B`. If both `A` and `B` are integers, integer division is performed (i.e., The result is the integer part of `A/B`).
`A%B`	The result is remainder from the integer division `A/B`.

```java
public static void main(String[] args) {


    int a = 5;
    int b = ++a;

    System.out.println(b);
    System.out.println(a);
    //a = 6, b = 6


    int c = 10;
    int d = c++;

    System.out.println("c=" + c + " d=" + d);
    //d= 10, c=11
}
```

Comparison operators

- Compares the values of two operands and returns a **boolean**
  - This is what is in the *if* statements

Expression	Function
A==B	Equal to: If the value of A is equal to the value of B then the result will be true otherwise, the result will be false.
A!=B	Not equal to: If the value of A is not equal to the value of B then the result will be true, otherwise the result will be false.
A<B	Less than: If the value of A is less than the value of B then the result will be true, otherwise the result will be false.
A<=B	Less than or equal to: If the value of A is less than or equal to the value of B then the result will be true, otherwise the result will be false.
A>B	Greater than: If the value of A is greater than the value of B then the result will be true, otherwise the result will be false.
A>=B	Greater than or equal to: If the value of A is greater than or equal to the value of B then the result will be true otherwise the result will be false.

Logical operators

- NOT A
- A and B
- A or B
- A xor B

- Truth tables for AND, OR, XOR, NAND

Expression	Function
`!A`	NOT: if `A` is `true` then the result will be `false`; if `A` is `false` then the result will be `true`.
`A&B`	AND: If both `A` and `B` are `true` then the result will be `true`, otherwise the result will be `false`.
`A&&B`	Conditional AND: If `A` is `true`, then the result will be the value of `B`, otherwise the result will be `false` (`B` will not be evaluated!).
`A|B`	OR: If both `A` and `B` are `false` then the result will be `false`, otherwise the result will be `true`.
`A||B`	Conditional OR: If `A` is `false` then the result will be the value of `B`, otherwise the result will be `true` (`B` will not be evaluated!).
`A^B`	XOR: The result will be `true` if only one of `A` or `B` is `true`. If both `A` and `B` are `true` or both `A` and `B` are `false` then the result will be `false`.

Ternary operation

- A shorthand if

```
public static void main(String[] args) {

    int number = 10;
    String result = (number % 2 == 0) ? "even" : "odd";
    System.out.println(result);

}
```

Assignment operators

- **=** assigns the *right* operand to the *left* operand.
- A=B
  - means that A now holds B's value
- The *left* operand must be a variable.

Expression	Equivalent Expression
A+=B	A=A+B
A-=B	A=A-B
A*=B	A=A*B
A/=B	A=A/B
A%=B	A=A%B
A&=B	A=A&B
A|=B	A=A|B
A^=B	A=A^B
A<<=B	A=A<<B
A>>=B	A=A>>B
A>>>=B	A=A>>>B

Reference Data Types

- We will see them later.

Arrays

- Can contain data of same type
  - Unlike python
- Need to declare the length beforehand.

```
int[] j
int k[]

int[] j = new int[10];
int k[];
k = new int[5];
```

```
char[] c = new char[3];
c[0] = 'a';
c[1] = 'b';
c[2] = c[1];
```

```
char[] c = {'a', 'b', 'b'};
```

Strings

```
String s = new String("abc"); // Create using constructor
String t = "abc"; // shortcut initialization
s = "abccd"; // shortcut assignment
```

```
String s;
s = "abc";
s += 1;
s += '0';
int j=5;
s = s+j;
```

```
int i = 5;
String s = Integer.toString(i);
i = String.valueOf(s);
```