# Java

*Introduction*

# History of Java

- Java was originally developed by Sun Microsystems starting in 1991
    - James Gosling
    - Patrick Naughton
    - Chris Warth
    - Ed Frank
    - Mike Sheridan
- This language was initially called *Oak*
- Renamed *Java* in 1995

# What is Java

- A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language -- **Sun Microsystems**

- **Object-Oriented**
  - No free functions
  - All code belong to some class
  - Classes are in turn arranged in a hierarchy or package structure
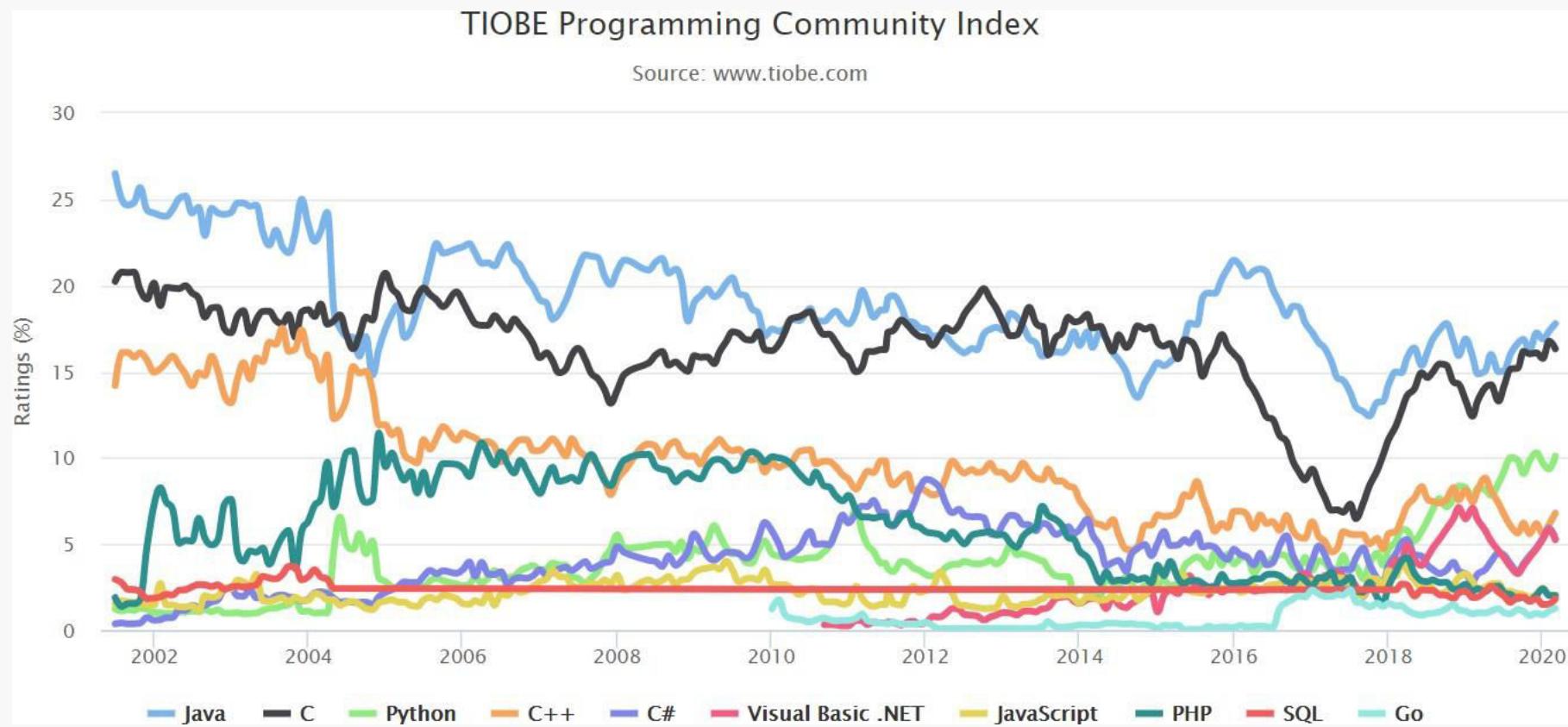
# What is Java

- **Distributed**
  - Fully supports IPv4, with structures to support IPv6
  - Includes support for Applets: small programs embedded in HTML documents

- **Interpreted**
  - The program are compiled into Java Virtual Machine (JVM) code called bytecode
  - Each bytecode instruction is translated into machine code at the time of execution
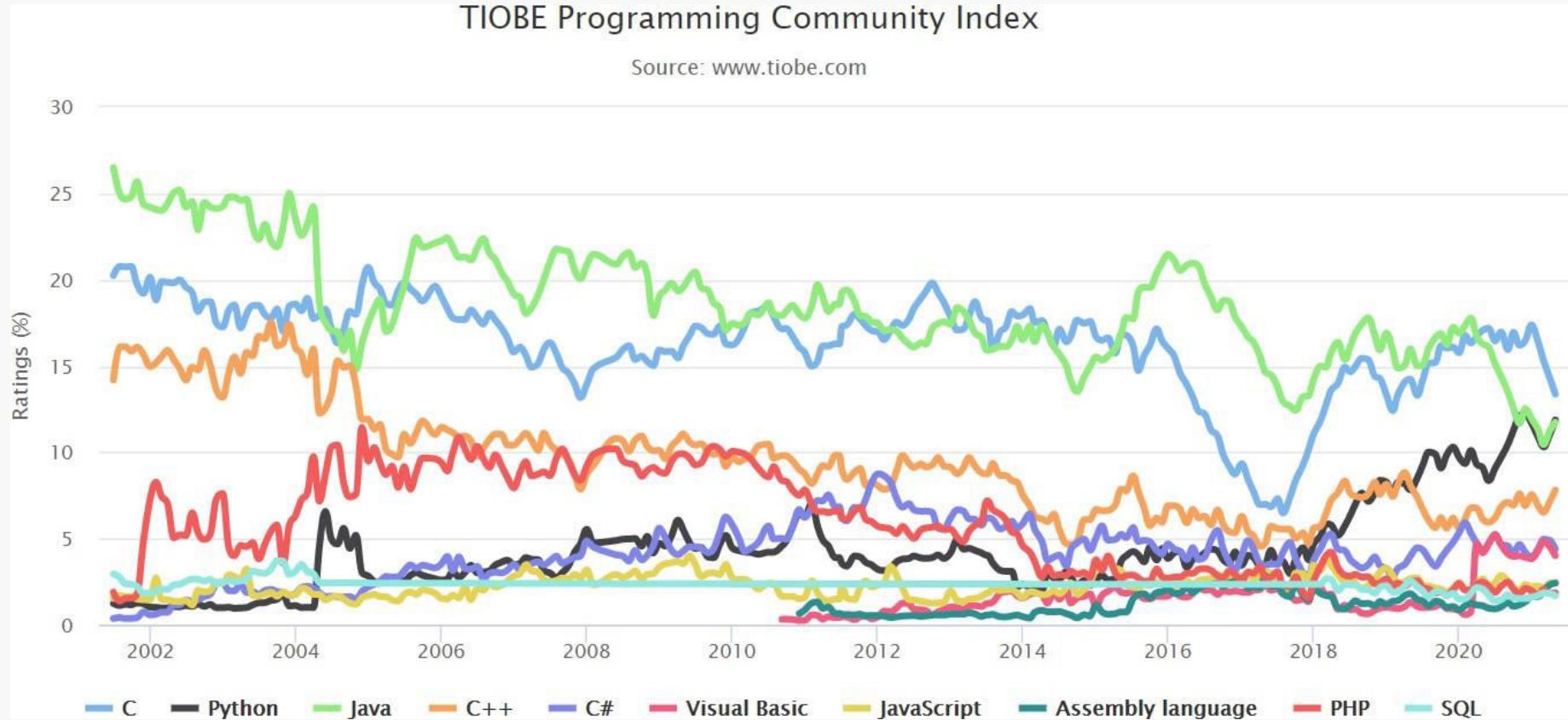
# What is Java

- **Robust**
  - Java is simple – no pointers/stack concerns
  - Exception handling – try/catch/finally series allows for simplified error recovery
  - Strongly typed language – many errors caught during compilation
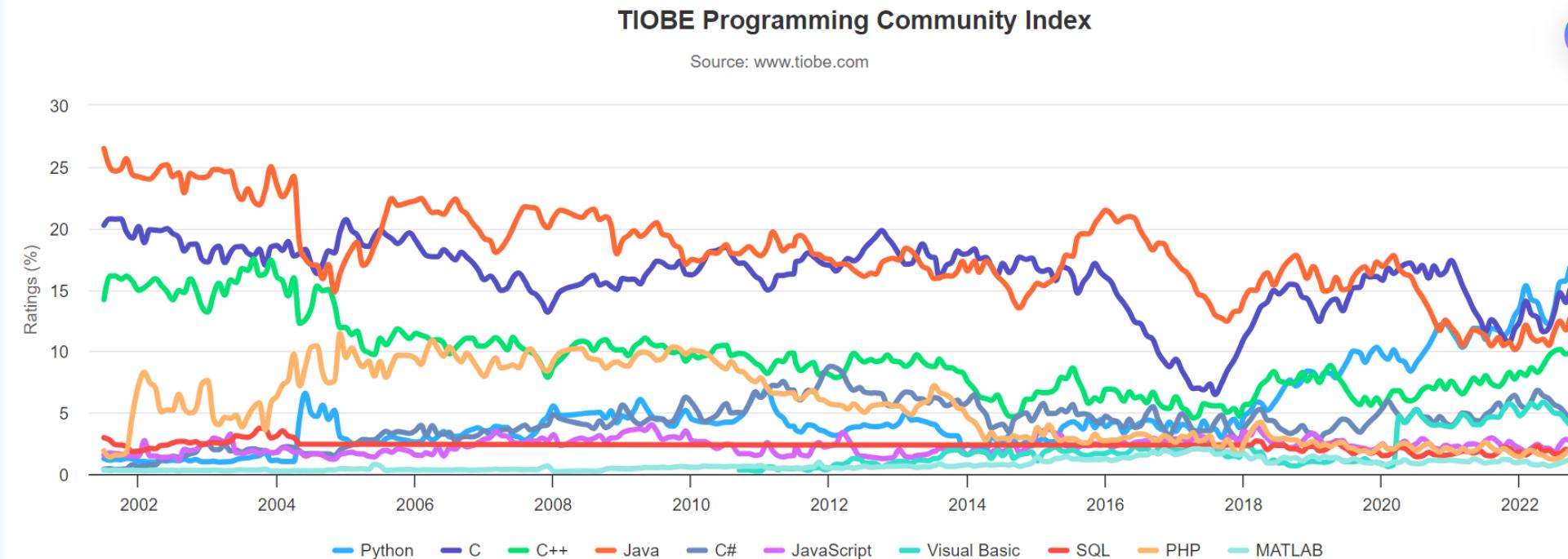
# Java – The Most Popular (2020)



TIOBE Programming Community Index
Source: www.tiobe.com

Legend: Java, C, Python, C++, C#, Visual Basic .NET, JavaScript, PHP, SQL, Go

# Java – Top Three (2021)



TIOBE Programming Community Index
Source: www.tiobe.com

# Java – Top Three (2022)



**TIOBE Programming Community Index**

Source: www.tiobe.com

Legend: Python, C, C++, Java, C#, JavaScript, Visual Basic, SQL, PHP, MATLAB

# Very Long Term History

To see the bigger picture, please find below the positions of the top 10 programming languages of many years back. Please note that these are *average* positions for a period of 12 months.
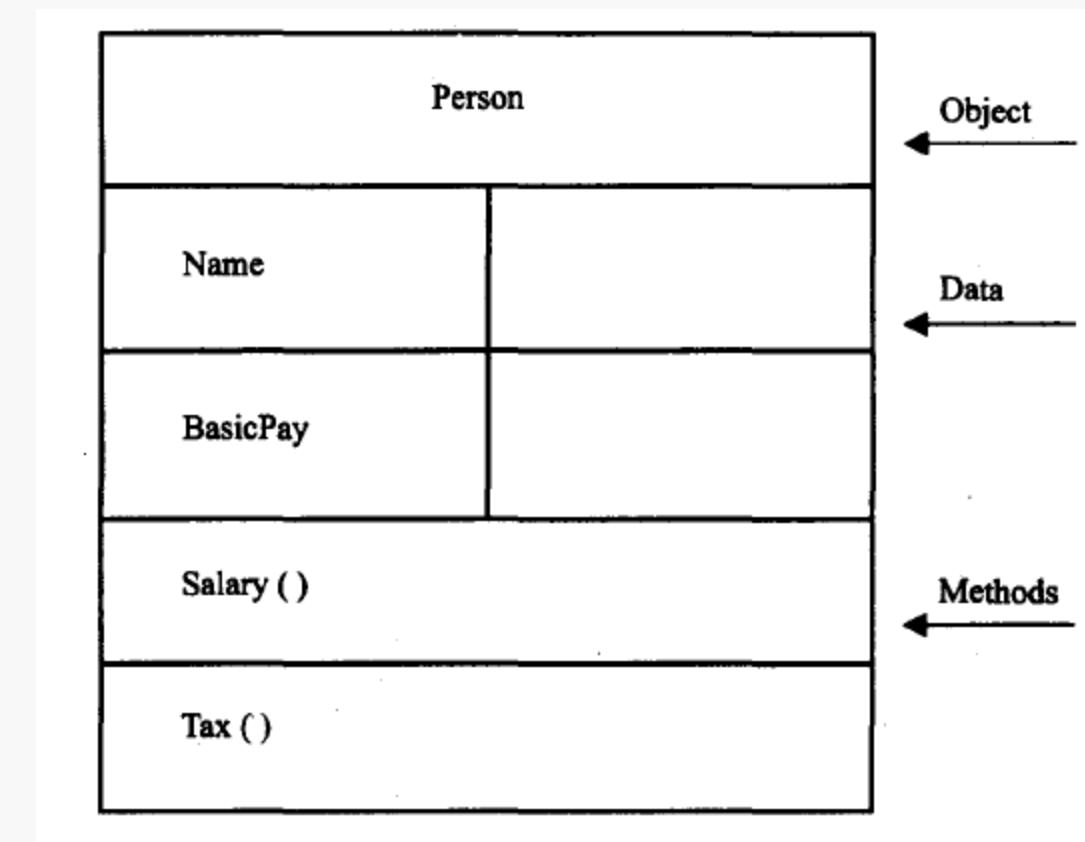
| Programming Language | 2021 | 2016 | 2011 | 2006 | 2001 | 1996 | 1991 | 1986 |
|---|---|---|---|---|---|---|---|---|
| C | 1 | 2 | 2 | 2 | 1 | 1 | 1 | 1 |
| Java | 2 | 1 | 1 | 1 | 3 | 15 | - | - |
| Python | 3 | 5 | 6 | 8 | 25 | 24 | - | - |
| C++ | 4 | 3 | 3 | 3 | 2 | 2 | 2 | 6 |
| C# | 5 | 4 | 5 | 7 | 13 | - | - | - |
| Visual Basic | 6 | 13 | - | - | - | - | - | - |
| JavaScript | 7 | 7 | 10 | 9 | 9 | 20 | - | - |
| PHP | 8 | 6 | 4 | 4 | 10 | - | - | - |

# Java Editions

- Java 2 Platform, Standard Edition (J2SE)
  – Used for developing **Desktop based application** and networking applications

- Java 2 Platform, Enterprise Edition (J2EE)
  – Used for developing **large-scale, distributed networking applications and Web-based applications**

- Java 2 Platform, Micro Edition (J2ME)
  – Used for developing applications for **small memory-constrained devices, such as cell phones, pagers and PDAs**

# OOP Paradigm

- Class
- Object
- Methods
- Data

| Person | |
|---|---|
| Name | |
| BasicPay | |
| Salary ( ) | |
| Tax ( ) | |

Object ←

Data ←

Methods ←

# OOP Pillars

- **Abstraction**

To Simplify reality and focus on the data and processes relevant to the application being built

- **Encapsulation**

Data and the programs that manipulate those data are bound together and their complexity is hidden.

- **Inheritance**
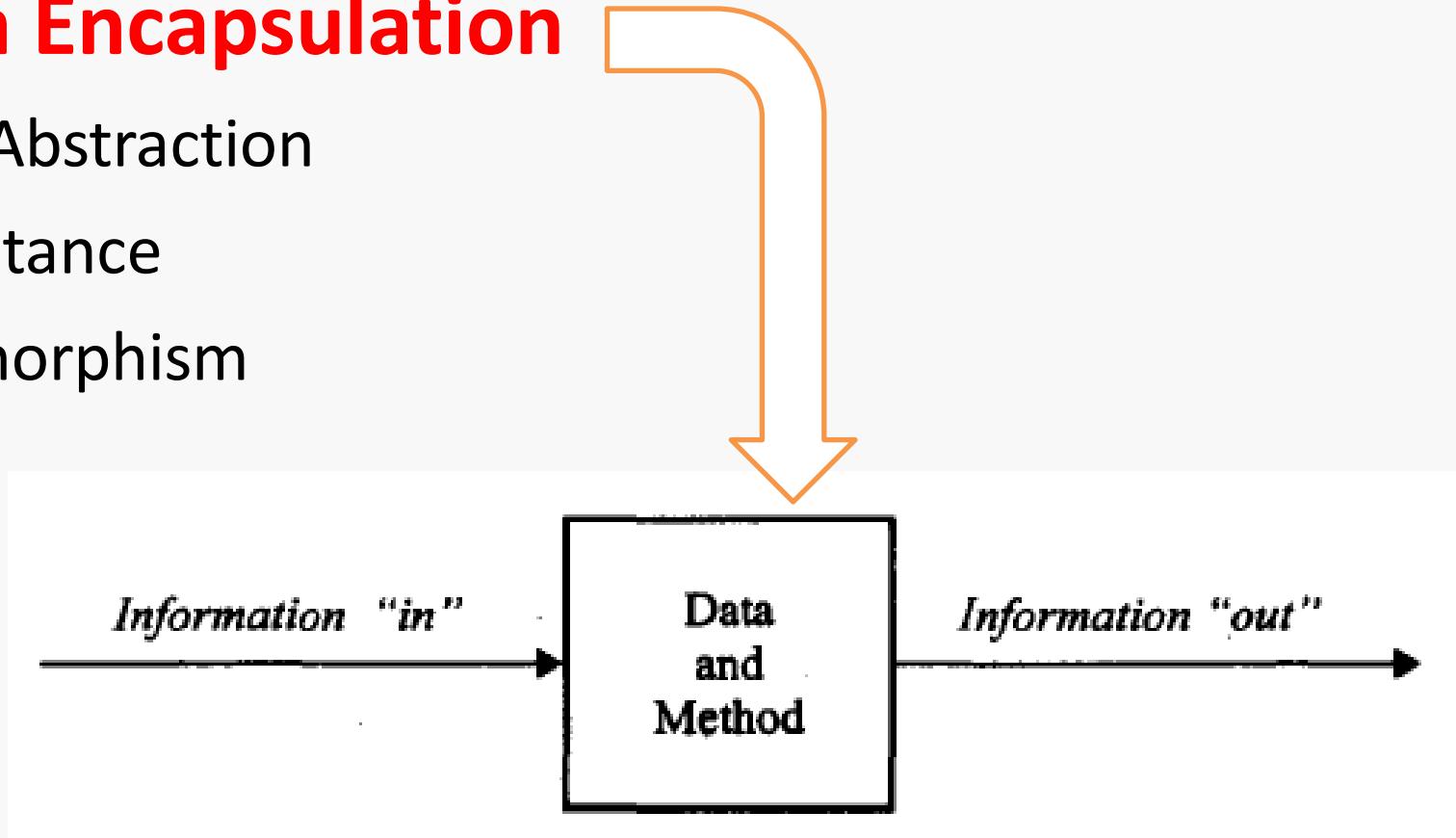
A class can derive methods and properties from another class, resulting in a hierarchy of classes.

- **Polymorphism**

Different subclasses of the same superclass can implement their shared interface differently.

# OOP Pillars

- **Data Encapsulation**
- Data Abstraction
- Inheritance
- Polymorphism

Information "in" → Data and Method → Information "out"

# OOP Pillars

- Data Encapsulation
- **Data Abstraction**
- Inheritance
- Polymorphism

**Access Modifiers**

Public

Protected

Private

**Process of Hiding data**
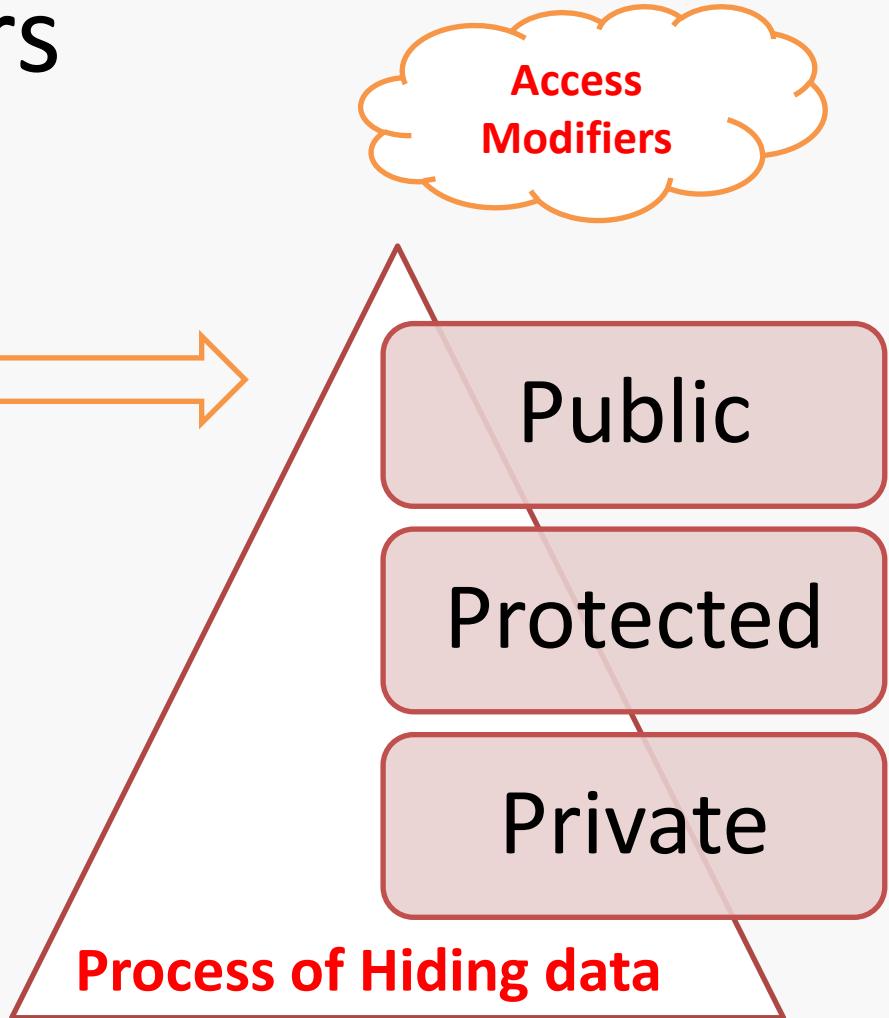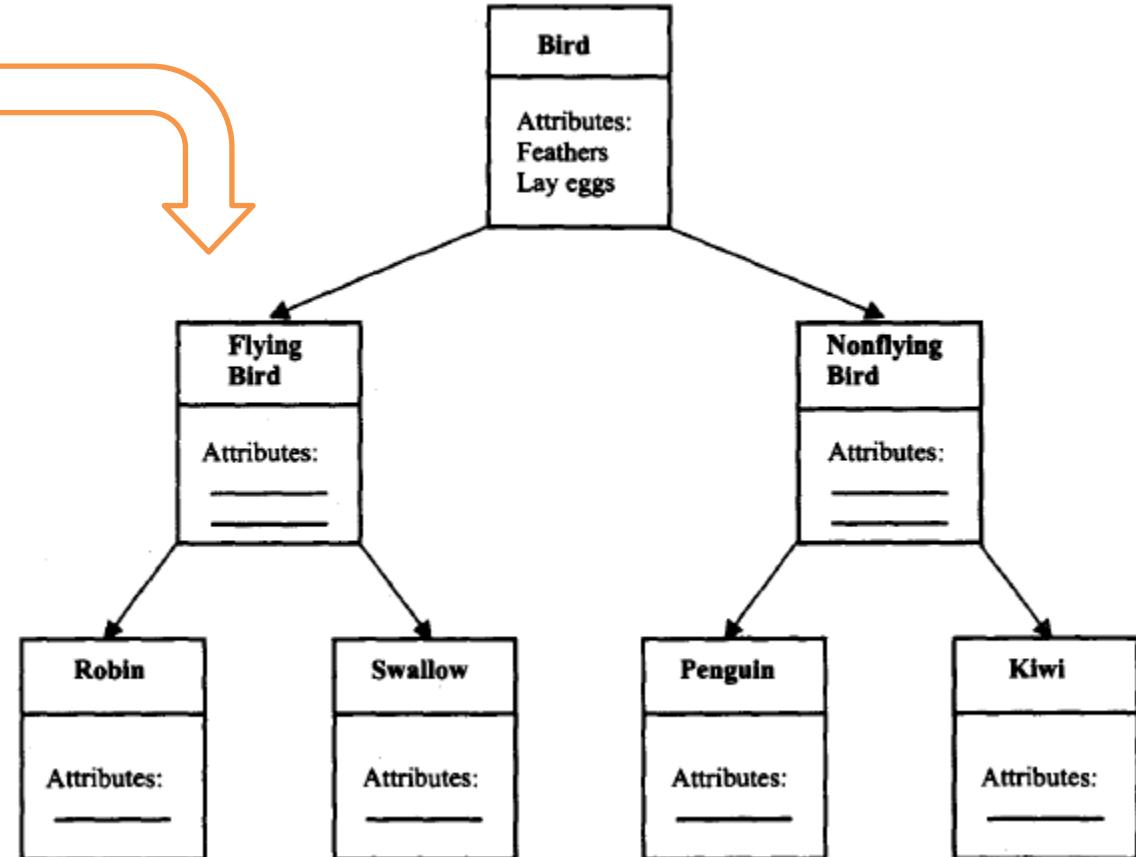
# OOP Pillars

- Data Encapsulation
- Data Abstraction
- **Inheritance**
- Polymorphism

# OOP Pillars

- Data Encapsulation
- Data Abstraction
- Inheritance
- **Polymorphism**
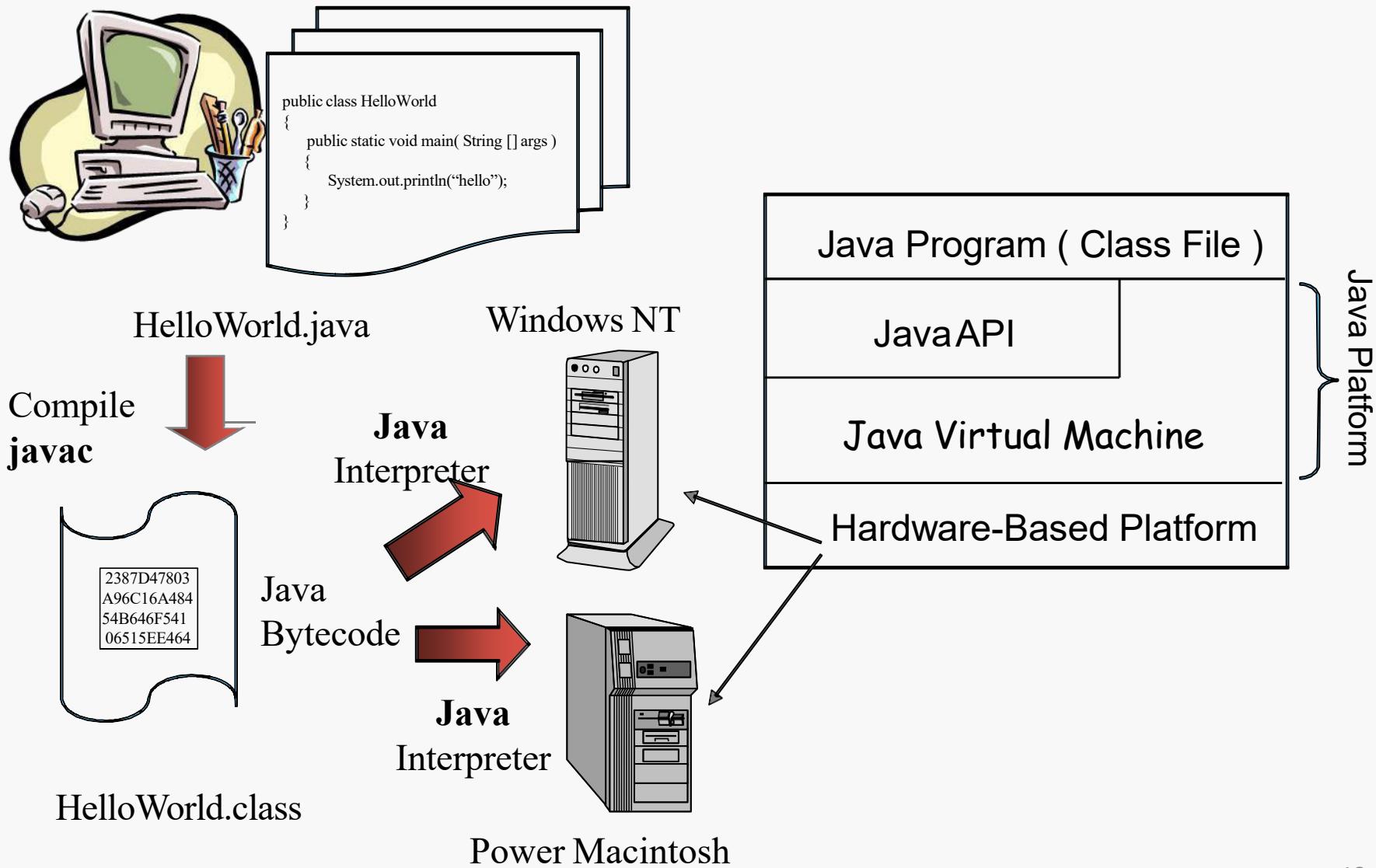
```
              ┌──────────────┐
              │    Shape     │
              ├──────────────┤
              │   Draw ( )   │
              └──────────────┘
```

| Circle Object | Box Object | Triangle Object |
|---|---|---|
| Draw (circle) | Draw (box) | Draw (triangle) |
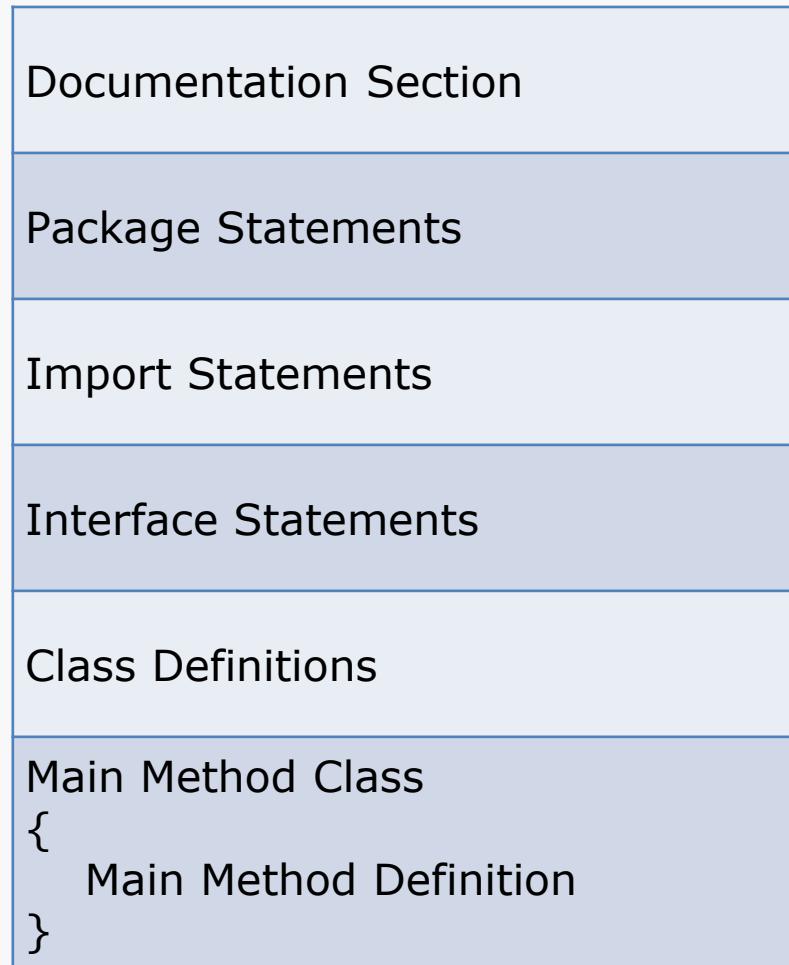
# Java Execution Procedure



```
public class HelloWorld
{
    public static void main( String [] args )
    {
        System.out.println("hello");
    }
}
```

HelloWorld.java

Compile
**javac**

```
2387D47803
A96C16A484
54B646F541
06515EE464
```

Java
Bytecode

HelloWorld.class

**Java**
Interpreter

Windows NT

**Java**
Interpreter

Power Macintosh

| Java Program ( Class File ) |
| Java API |
| Java Virtual Machine |
| Hardware-Based Platform |

Java Platform

# Java Program Structure

| |
|---|
| Documentation Section |
| Package Statements |
| Import Statements |
| Interface Statements |
| Class Definitions |
| Main Method Class<br>{<br>    Main Method Definition<br>} |

**19**

# HelloWorld.java

Case Sensitive

Class name same as Source file name

Comments /* (enter)

```java
/*
*This is a simple Java program.
*Call this file "Example.java".
*/
class HelloWorld
    // Your program begins with a call to main().
    public static void main(String args[]) {

    int numOfPoints = 10;
    String studentName = "Alice";

    System.out.println("This is a simple Java
program.");
    }
}
```

Class starts with Caps so does Source file name

One main() method among all class files

Variable name are Camel Case

Important: Indent, { }, ";"

20

# Java Development Environment

- Edit
  - Create/edit the source code
- Compile
  - Compile the source code
- Load
  - Load the compiled code
- Verify
  - Check against security restrictions
- Execute
  - Execute the compiled

# Phase 1: Creating a Program

- Any text editor or Java IDE (Integrated Development Environment) can be used to develop Java programs

- Java source-code file names must end with the *.java* extension

- Some popular Java IDEs are
  - NetBeans
  - Eclipse
  - IntelliJ

# Phase 2: Compiling a Java Program

- *javac HelloWorld.java*
    - Searches the file in the current directory
    - Compiles  the  source  file
    - Transforms the Java source code into bytecodes
    - Places the bytecodes in a file named *HelloWorld*.class

# Bytecodes *

- They are **not** machine language binary code
- They are independent of any particular microprocessor or hardware platform
- They are platform-independent instructions
- Another entity (**interpreter**) is required to **convert the bytecodes into machine codes** that the underlying microprocessor understands
- This is the job of the **JVM** (Java Virtual Machine)

# JVM (Java Virtual Machine) *

- It is a part of the JDK and the foundation of the Java platform

- It can be installed separately or with JDK

- A **virtual machine (VM)** is a software application that simulates a computer, but hides the underlying operating system and hardware from the programs that interact with the VM

- It is the JVM that makes Java a **portable language.**

# JVM (contd..)*

- The same bytecodes can be executed on any platform containing a compatible JVM
- The JVM is invoked by the java command
  - *java HelloWorld.java*
- It searches the class Welcome in the current directory and executes the main method of class Welcome
- It issues an error if it cannot find the class Welcome or if class Welcome does not contain a method called main with proper signature

# Phase 3: Loading a Program *

- One of the components of the JVM is the class loader

- The class loader takes the **.class** files **containing the programs bytecodes** and transfers them to RAM

- The class loader also loads any of the .class files provided by Java that our program uses
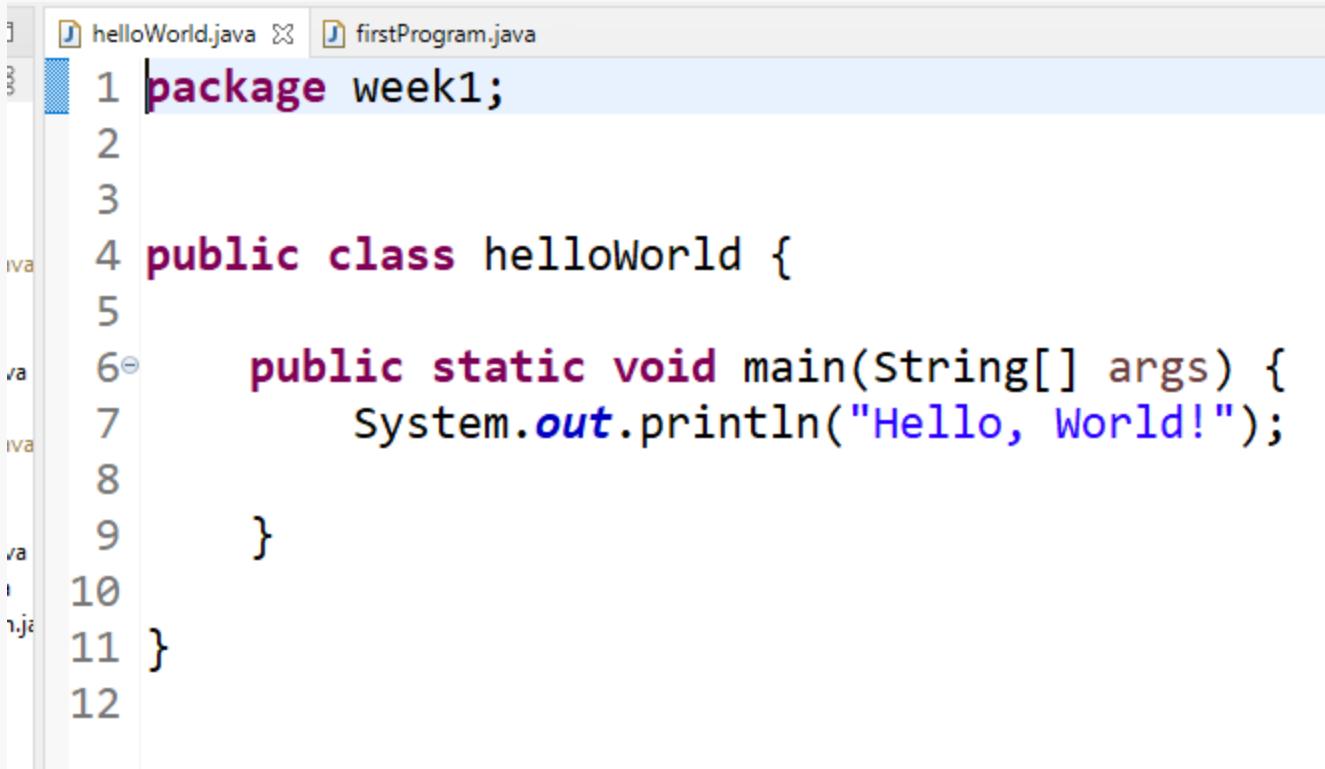
# Phase 4: Bytecode Verification *

- Another component of the JVM is the **bytecode verifier.**

- Its job is to ensure that bytecodes **are valid and do not violate Java's security** restrictions

- This feature helps to **prevent** Java programs arriving over the network from **damaging our system.**

# Phase 5: Execution

- Now the actual execution of the program begins

- Bytecodes are converted to machine language suitable for the underlying OS and hardware

- Java programs actually go through **two compilation phases:**

  - Source code -> Bytecodes

  - Bytecodes -> Machine language

# Editing a Java Program

```java
package week1;


public class helloWorld {

    public static void main(String[] args) {
        System.out.println("Hello, World!");

    }

}
```

# Examining HelloWorld.java

- A Java source file can contain multiple classes, but only **one class can be a public class.**

- Typically Java classes are grouped into **packages** (similar to namespaces in C++)

- A **public** class is accessible across packages.

- The source file name must **match the name** of the public class defined in the file with the .java extension

# Examining HelloWorld.java

- In Java, there is no provision to declare a class, and then define the member functions outside the class.

- Body of every member function of a class (**called method in Java**) must be written when the method is declared.

- Java methods **can be written in any order** in the source file.

- A method **defined earlier in the source file can call a method defined later**.

# Examining HelloWorld.java

- ***public static void main(String[] args)***
  - **main** is the starting point of every Java application
  - **public** is used to make the method accessible by all
  - **static** is used to make main a static method of class HelloWorld. Static methods can be called without using any object; just using the class name. JVM call main using the **ClassName.methodName** (Ex. *HelloWorld.main*) notation
  - **void** means main does not return anything
  - **String args[ ]** represents an array of String objects that holds the **command line** arguments passed to the application.

# Examining HelloWorld.java

- ***System.out.println()***
  - Used to print a line of text followed by a new line *(ln)*
  - **System** is a class in java.lang package
  - **out** is a public static member of class System
  - **out** represents the standard output stream
  - **println** is a public method of the class of which out is an object

# Examining HelloWorld.java

- **System.out.print()** is similar to **System.out.println()**, but does not print a new line automatically

- **System.out.printf()** is used to print formatted output like printf() in C

- In  Java, characters enclosed by double quotes ("") represents a String object, where String is a class of the Java API

- We can use the plus operator **(+) to concatenate** multiple String objects and create a new String object

# READING INPUT FROM KEYBOARD

int a;

Scanner sc = new Scanner(System.in);

 a = sc.nextInt();

**1) int nextInt()**

**It is used to read an integer value from the keyboard.**

**2) float nextFloat()**

**It is used to read a float value from the keyboard.**

**3) long nextLong()**

**It is used to read a long value from the keyboard.**

**import java.util.Scanner;**

**4) String next()**

**It is used to read string value from the keyboard.**

# Scanner example

```java
import java.util.Scanner;

class test

{       public static void main(String args[])

        {       int a,b,c;

                Scanner sc = new Scanner(System.in);

              System.out.println("Enter a first number");

               a = sc.nextInt();

               System.out.println("Enter a second number");

               b = sc.nextInt();

               c = a+ b;

               System.out.println("sum is :"+c);

        }

 }
```

# Scanner example-2

```java
import java.util.Scanner;
class GetInputFromUser{
  public static void main(String args[])  {
          int a;
          float b;
          String s;
          Scanner in = new Scanner(System.in);
          System.out.println("Enter a string");
          s = in.nextLine();
          System.out.println("You entered string "+s);
          System.out.println("Enter an integer");
          a = in.nextInt();
          System.out.println("You entered integer "+a);
          System.out.println("Enter a float");
          b = in.nextFloat();
          System.out.println("You entered float "+b);
  }
}
```

```java
System.out.println("Enter character: ");
char charInput = sc.next().charAt(0);
```

# Data Types and Variables

## ❑ Java Keywords

| abstract | continue | for | new | switch |
|----------|----------|-----|-----|--------|
| assert | default | goto | package | synchronized |
| boolean | do | if | private | this |
| break | double | implements | protected | throw |
| byte | else | import | public | throws |
| case | enum | instanceof | return | transient |
| catch | extends | int | short | try |
| char | final | interface | static | void |
| class | finally | long | strictfp | volatile |
| const | float | native | super | while |

## ❑ Comments

- // for single line, /*  and */ for multiline

# Data Types and Variables

## Primitive Data Types

### Integers:

▸ **Java does not support unsigned integers**

| Name | Width in bits | Range | |
|---|---|---|---|
| long | 64 | -9,223,372,036,854,775,808 | 9,223,372,036,854,775,807 |
| int | 32 | -2,147,483,648 | 2,147,483,647 |
| short | 16 | -32,768 | 32,767 |
| byte | 8 | -128 | 127 |

### Floating-Point:

| Name | Width in bits | Approximate Range | |
|---|---|---|---|
| double | 64 | 4.9e-324 | 1.8e+308 |
| float | 32 | 1.4e-045 | 3.4e+038 |

### Character: char 16 (Unicode)

### Boolean: boolean 8 (true, false)

# Data Types and Variables

## Identifiers

- **Used for naming variables, methods, classes, interfaces and packages**

- **Sequence of uppercase and lowercase letters, numbers, underscore and dollar-sign characters**

- **Keywords cannot be used**

- **Must not begin with a number**

- **Case sensitive**

- **Conventions:**

  - **Variables and Methods: Mixed Case**

    - **first letter of each word except the first word in uppercase, others in lowercase**

  - **Interfaces and Classes: Camel Case**

    - **first letter of each word in uppercase, others in lowercase**

  - **Packages: Lower Case**

**Question:**

```java
class BoolTest
{
public static void main(String args[])
{
    boolean b;
    b = false;
    System.out.println("b is " + b);
    b = true;
    System.out.println("b is " + b);
    if(b)
        System.out.println("This is executed.");
    b = false;
    if(b)
        System.out.println("This is not executed.");
    System.out.println("10 > 9 is " + (10 > 9));
    }
}
```

```
b is false
b is true
This is executed.
10 > 9 is true
```

# Data Types and Variables

## Scope and Lifetime of Variables

**Example:**

```
void test()
{
  int x = 10;
  if (x == 10)
      {
            int y = 20;          // y = 20
            x = y * 2;           // x = 40
            System.out.println("x = " + x);
            System.out.println("y = " + y);
      }
  y = 100;                       // cannot find symbol
  x = x + 15;
  System.out.println("x = " + x);
}
```

# Operators

## ☐ Arithmetic and Relational Operators

| Operator | Result |
|----------|--------|
| + | Addition |
| - | Subtraction (also unary minus) |
| * | Multiplication |
| / | Division |
| % | Modulus |
| ++ | Increment |
| -- | Decrement |
| += | Addition assignment |
| -= | Subtraction assignment |
| *= | Multiplication assignment |
| /= | Division assignment |
| %= | Modulus assignment |

| Operator | Result |
|----------|--------|
| == | Equal to |
| != | Not equal to |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |

**44**

# *Operator precedence Groups*

| Operator Category | Operators | Associativity |
|---|---|---|
| Unary operators | + - + + - - ! | R→L |
| Arithmetic operators | * / % | L→R |
| Arithmetic Operators | + - | L→R |
| Shift Operators | << >> | L→R |
| Relational operators | < <= > >= | L→R |
| Equality operators | == != | L→R |
| Bitwise AND | & | L→R |
| Bitwise XOR | ^ | L→R |
| Bitwise OR | \| | L→R |
| Logical AND | && | L→R |
| Logical OR | \|\| | L→R |
| Conditional expression | ? : | R→L |
| Assignment operator | = += -= *= /= %= | R→L |

# Control Statements

## Selection statements

### The *if* statement

Syntax: Same as in C and C++

- *if* statements can be nested
- *if-else-if* ladder can be used

### The *switch* statement

Syntax: Same as in C and C++

- *switch* statements can be nested

**Note:**

From JDK 7 onwards, strings can be used as case labels in 'switch' statement

# ❑Iteration statements

## ❑The *while* statement

**Syntax:**    **Same as in C and C++**

## ❑The *do-while* statement

**Syntax:**    **Same as in C and C++**

## ❑The *for* statement

**Syntax:**    **Same as in C and C++**

# Control Statements

## Jump (transfer of control) statements

- Basic form of the *break* statement

Syntax:      Same as in C and C++

  – Used with switch statement and loops
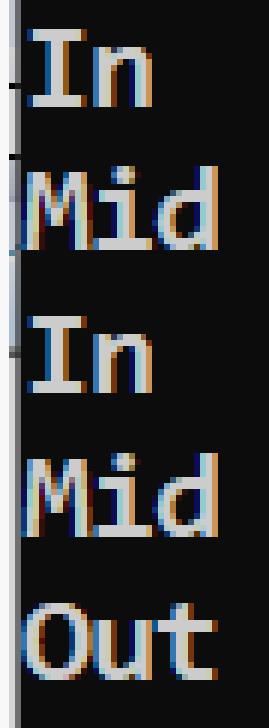

- Labeled *break* statement

Syntax:       *break*  label;

48

# Control Statements

```java
class break_example
{
  public static void main(String args[])
  {
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if ( j == 1 )
                break;
            System.out.println ("In");
        }

        System.out.println ("Mid");
    }

    System.out.println ("Out");

  }
}
```

```
In
Mid
In
Mid
Out
```

49

# Data Types and Variables

## Type Conversion and Casting

- **Automatic Conversion: Conditions**
  - Two types are compatible
    - Numeric types (integer and floating-point) are compatible with each other
  - Destination type is larger than source type
- **Examples:**
  - byte to int – possible
  - Any numeric type to char/boolean – not possible
  - char and boolean – not compatible with each other
- **Automatic type conversion is done when a literal integer constant is stored into variables of type byte, short, long and char**

# Type conversion example:

1. **int  a = 2;**

2. **float  b = 2.5f;**

3. **double  c = 11.11;**

4. **a = c;**  **// Error**

5. **b = c;**  **// Error**

6. **c = a;**  **// OK**

7. **c = b;**  **// OK**

BitwiseAdd.java
DataTypeConversion.java
DataTypeCon2.java

# Data Types and Variables

## Type Conversion and Casting (Continued ...)

- Syntax:  (target-type) value
- Example:  int a; byte b; b = (byte) a;

- **Truncation in case of floating-point values**
  - Example:  1.23 assigned to integer $\rightarrow$ 1 (0.23 truncated)

- **Examples:**

  ```
  byte b;
  int i = 257;
  double d = 323.142;
  b = (byte) i;        // 1 (remainder of special type of division)
  i = (int) d;              // 323 (0.142 lost)
  b = (byte) d;        // 67 (0.142 lost and reduced to modulo)
  ```

# Arrays

## One-Dimensional Arrays

- **Syntax:** **type var-name[];**
  - ▸ **Only declaration, memory not allocated**
  - ▸ **Allocate memory using new operator**
    
    **array-var = new type[size]**
- **Example 1:**
    
    **int one[];**
    
    **one = new int [5];**
- **Example 2:**
    
    **int one[] = new int [12];**
- **Example 3:**
    
    **int one[] = {1, 2, 3, 4, 5};**

# Arrays

## Two-Dimensional Arrays

- **Syntax: type var-name[][];**
  - ▸ **Only declaration, memory not allocated**
  - ▸ **Allocate memory using new operator**

    array-var = new type[size][size]

- **Example 1:**

  int two[][];

  two = new int [3][2];

- **Example 2:**

  int two [][] = new int [3][2];

- **Example 3:**

  int two[][] = {{1, 2}, {3, 4}, {5, 6}};

**55**

## Array Example:

```java
class ArrayDemo
{
    public static void main( String []args)
    {
        int A[], size;
        Scanner S = new Scanner(System.in);
        System.out.println("Enter the size of array:");
        size = S.nextInt();

        A = new int[size];

        System.out.println("Enter elements: ");
        for( int i = 0 ; i < size ; i++ )
            A[i] = S.nextInt();

        System.out.println("Array elements are: ");
        for( int i = 0 ; i < A.length ; i++ )
            System.out.println( A[i] );
    }
}
```

**OUTPUT**

```
Enter the size of array:
4
Enter elements:
11 22 33 44
Array elements are:
11
22
33
44
```

## 2-D Array example:

```
1   class Array2DTest
2   {
3       public static void main( String []args )
4       {
5           int M[][] = { { 11 , 22 , 33 } ,
6                         { 44 , 55 , 66 }
7                       };
8           for( int i = 0 ; i < M.length ; i++)
9           {
10              for( int j = 0 ; j < M[i].length ; j++ )
11                  System.out.printf("%d\t", M[i][j] );
12              System.out.println();
13          }
14      }
15  }
```

```
11          22          33
44          55          66
```

## 2-D Array example-2:

```
1   class Array2DTest
2   {
3       public static void main( String []args )
4       {
5           int M[][] = { { 11 , 22 , 33 } ,
6                         { 44 } ,
7                         { 55 , 66 }
8                       };
9           for( int i = 0 ; i < M.length ; i++)
10          {
11              for( int j = 0 ; j < M[i].length ; j++ )
12                  System.out.printf("%d\t", M[i][j] );
13              System.out.println();
14          }
15      }
16  }
```

```
11              22              33
44
55              66
```

# Java version, IDE, and Textbook

- **JDK:** https://www.oracle.com/java/technologies/downloads/

- **IDE-Eclipse:** https://www.eclipse.org/downloads/

- **Books**

  - Schildt H, Java: The Complete Reference, (10e), Tata McGraw-Hill Education Group, 2017.

  - Balagurusamy E, Programming with Java, (5e), Tata McGraw Hill, 2017.

  - Daniel Liang Y, Introduction to Java Programming, (10e), Pearson Education, 2018.

  - Horstmann CS, Big Java: Early Objects, (5e), Wiley's Interactive Edition, 2015.