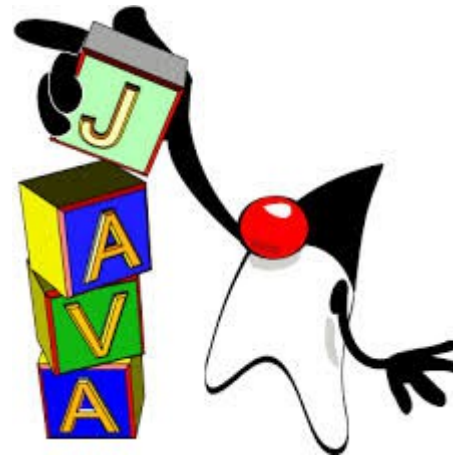# Advanced Programming
Introduction

Course Description

- The Goal
- The Motivation
- Lectures and Assignments
- Programming Platform
- Resources
- Evaluation

Lab: *problems, projects, essays* → easy

Exam: *written test* → hard

What exactly is "Java"?

- Programming Language
- Programming **Platform**
- 1995
- Sun Microsystems / **Oracle** (2010)
- James Gosling
- Duke

Why Java?

Where Is Java Used?

- **Enterprise** applications (banking, commerce, etc.)
  - Large, complex, distributed, scalable, secure, etc.
  - Web: Gmail, Amazon, LinkedIn, Netflix, EBay, FB, etc.
- **Android** applications: most of them
- **Desktop** applications
  - IDES: Netbeans, Eclipse, IntelliJ, PyCharm, etc.
  - Application servers: GlassFish, Tomcat, etc.
- **IoT** applications
- **Huge** ecosystem of **libraries** and **frameworks**
  - Apache Foundation Projects, Hadoop, Spark, Spring, etc.
- Minecraft :)

Java Programming Language

- **Simplicity**

  "as simple as possible, but not simpler"

- **Robustness**: ~~pointers~~, automatic memory management, garbage collection, strong typing

- Completely **object-oriented**

- **Secure** class loading and verification

- **Architecture Neutrality**

- **Portability**

- **Performance**

WORA
Write once, run anywhere

Java Platforms

- **Java SE (Standard Edition)**

  Desktop applications, applets, Java Web Start, JavaFX

- **Java EE (Enterprise Edition)**

  Complex, distributed, large scale, applications; server-side components, Web Services, etc.

- **Java ME (Micro Edition)**

  Programming embedded systems, mobile devices, TVs, GPSs, etc.

- **Java Card**

Compiled and Interpreted

- **Interpreted languages**
  - simplicity, portability
  - low execution speed

- **Compiled languages**
  - high execution speed
  - no portability

- **Java: compiled *and* interpreted**

  The Java compiler doesn't generate "machine code" (native hardware instructions). Rather, it generates **bytecodes**: a high-level, machine-independent code for a hypothetical machine that is implemented by the Java interpreter and run-time system.

Static vs. Dynamic Types

- **Statically typed** programming languages verify and enforce the constraints of data types at **compile-time**.

```
int test(int a) {
    return (a > 0 ? a + 1 : "a" - 1);
}
//Java
```
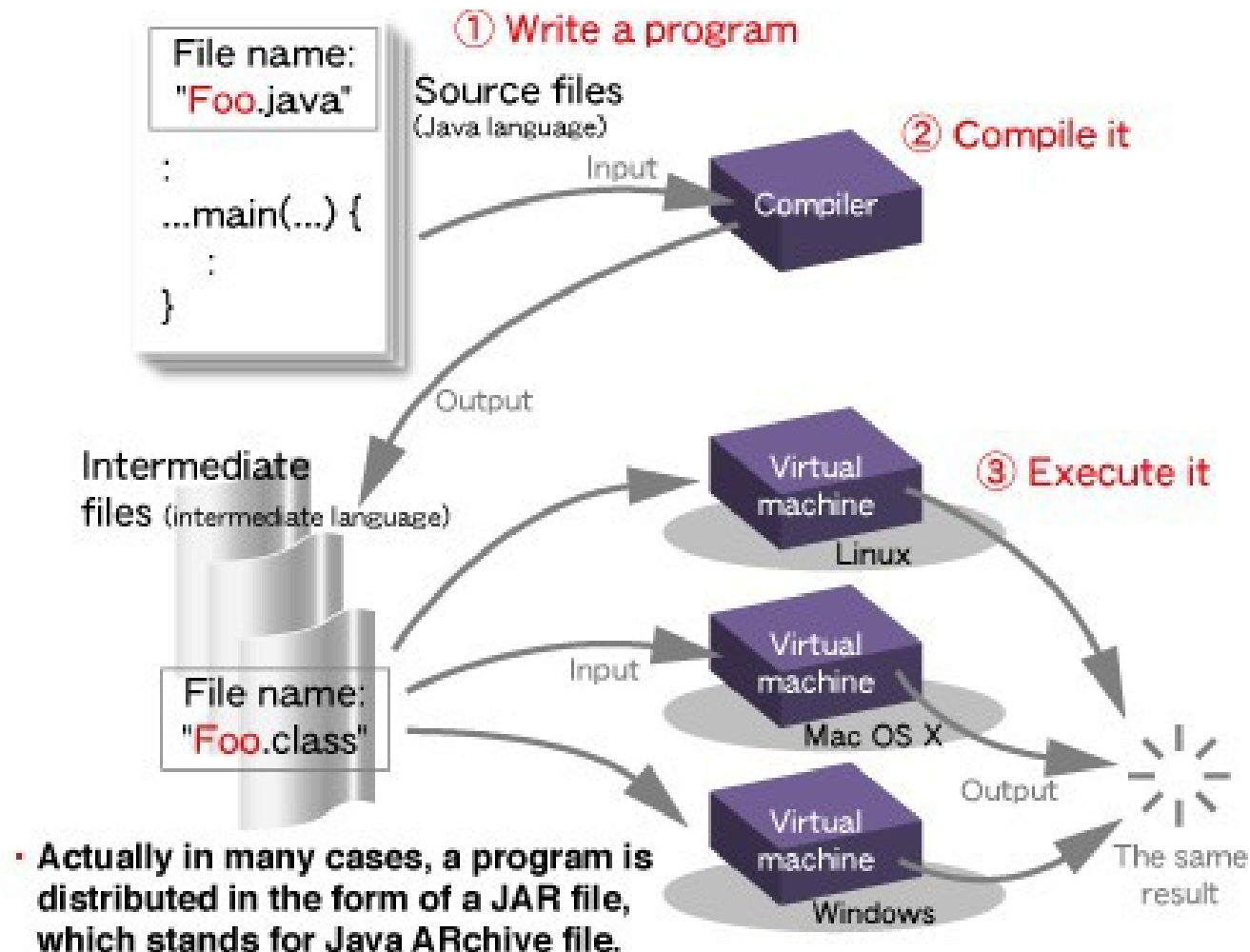
Compile-time error

- **Dynamically typed** programming languages do type checking at **run-time**.

```
def test(a):
    return a + 1 if a > 0 else "a" - 1

#Python
```

Run-time error

Java Virtual Machine (JVM)

You are not alone...

① Write a program

File name:
"Foo.java"

Source files
(Java language)

:
...main(...) {
    :
}

Input

② Compile it

Compiler

Output

Intermediate files (intermediate language)

File name:
"Foo.class"

Input

Virtual machine
Linux

Virtual machine
Mac OS X

③ Execute it

Output

The same result

Virtual machine
Windows

· Actually in many cases, a program is distributed in the form of a JAR file, which stands for Java ARchive file.

JVM Languages

- Java

- Groovy   :dynamic, scripting

- Scala   :functional

- Kotlin   :static, less verbose, Android

- Clojure   :functional, Lisp dialect

- JRuby   :port for Ruby

- Jython   :port for Python

- etc.

The First Program

```
public class HelloWorld {

  public static void main(String args[]) {
    System.out.println("Hello world!");
  }
}
```
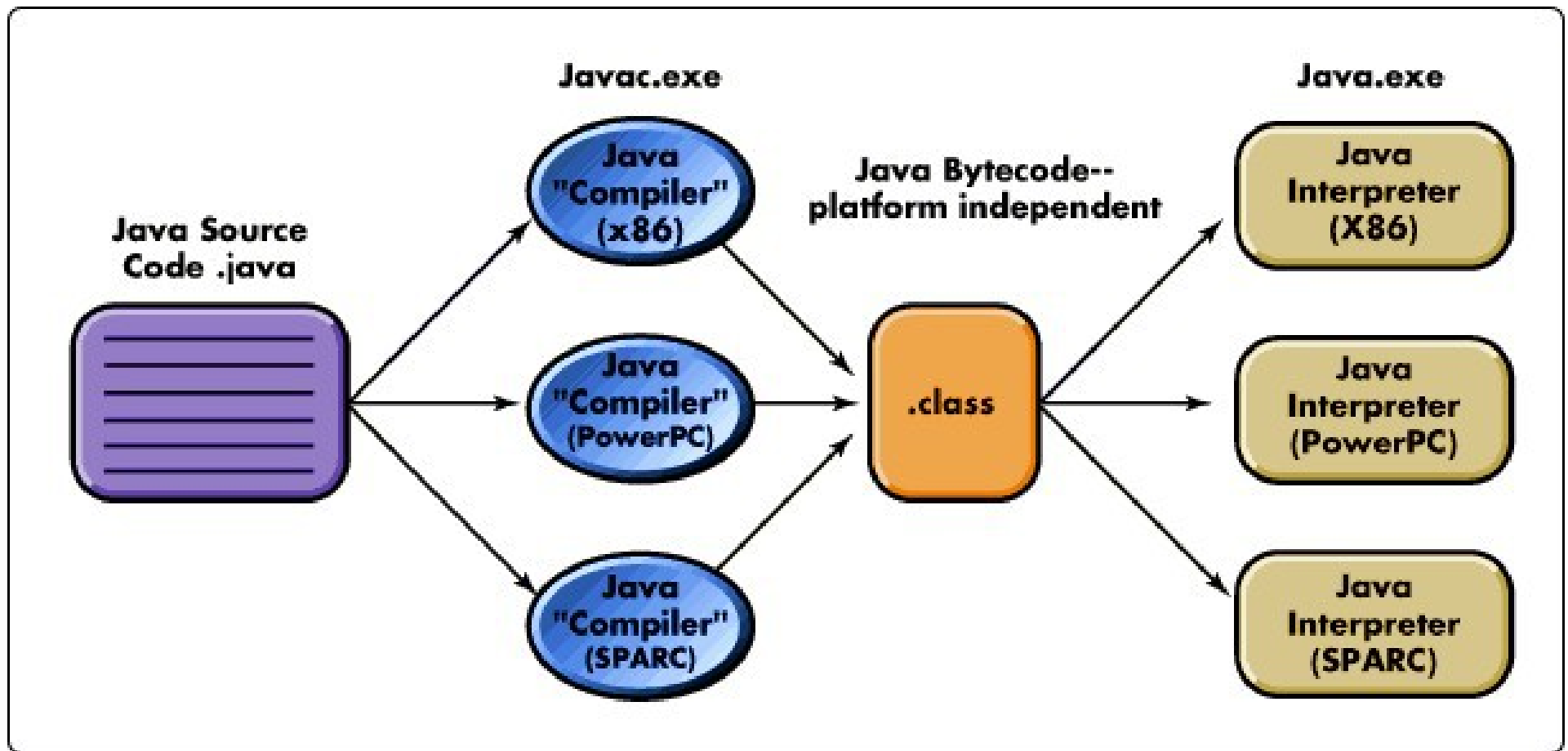
- *Source*: `HelloWorld.java`
- *Compile*

        `javac HelloWorld.java → HelloWorld.class`
- *Run*

        `java HelloWorld`

java, javac

javap

javap -c HelloWorld

```
Compiled from "HelloWorld.java"
public class HelloWorld extends java.lang.Object{
HelloWorld();
  Code:
   0: aload_0
   1: invokespecial #1;
      //Method java/lang/Object."<init>":()V
   4: return

public static void main(java.lang.String[]);
  Code:
   0: getstatic  #2;
      //Field java/lang/System.out:Ljava/io/PrintStream;
   3: ldc #3;
      //String Hello world!
   5: invokevirtual  #4;
      //Method java/io/PrintStream.println:(Ljava/lang/String;)V
   8: return
}
```

aload = load a reference onto the stack from a local variable #index
invokespecial = invoke instance method on object objectref and puts the result on the stack
getstatic = get a static field value of a class, where the field is identified by field reference in the constant pool index
ldc = push a constant #index from a constant pool
...

Obfuscation

JDK, JRE

- **JDK** = Java Development Kit
  - All the tools you need for Java **development**
  - Includes a JRE
  - Does not include an IDE
- **JRE** = Java Runtime Environment
  - All that is required to **run** a Java program
- Both include the **JVM**
- Current version: **11** (2018)
- Oracle JDK vs Open JDK

Integrated Development Environment (IDE)

Code Assistance: Smart code completion, managing imports, refactoring, generating code, templates, hints, navigation, documentation, debugging, profiling, etc.

- **NetBeans IDE**
  - Apache Software, Oracle

- **Eclipse IDE**
  - Eclipse Foundation, IBM

- **IntelliJ IDEA**
  - JetBrains

- JDeveloper

- Oracle Developer Studio, etc.

Examples (Netbeans)
Write:
    `sout`
*Press **TAB** and you get:*
    `System.out.println("");`
Easy to write the code.
Easy to **read** its meaning.

*Suggestions: **Ctrl+Space***
*Rename: **Ctrl+R***
...

UNICODE

- Character encoding system.

- It supports most of the written languages.

- Each character is represented using **2 bytes**

- **65536** symbols, **\uxxxx** $(\backslash u03B1 \rightarrow \alpha)$

- ASCII compatible

- Structured in **blocks**: Basic Latin, Greek, Arabic, Gothic, Currency, Mathematical, Arrows, Musical, etc.

- public class приветмир{}

- System.out.println(" 好世界 ");

Java Basic Syntax

- ## Similar to C++

- ## Keywords (50) (C++:93, C#: 79, Python: 33, Go:25, SmallTalk:6)

- ## Literals: "Hello World", 'J', 'a', 'v', 'a', 10, 010, 0xA, 0b11, 12.3,

  12.3d, 12.3f, 12e3, 123L, true, false, null, 0722_123_456

- ## Separators: ( ) { } [ ] ; , .

- ## Operators

  (char)65 + "nna" + "has" + (8 >> 2) + " apples"

  http://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html

Comments

```java
/* To change this template file, choose Tools | Templates
   and open the template in the editor. */
/**
 * Main class of the application
 * @author Duke
 */
public class HelloWorld {
    /**
     The execution of the application starts here.
     @param args the command line arguments
     */
    public static void main(String args[]) {
        // TODO code application logic here
        System.out.println("Hello World!"); // Done!
    }
}
```

javadoc – a tool for generating API documentation in HTML format from doc comments in source code

Data Types

Primitive types

- arithmetic: `byte (1), short (2), int (4), long (8)`

- floating point: `float (4), double (8)`

- character: `char (2)`

- logical: `boolean (?)`

Reference types

classes, interfaces, annotations, enumerations

~~pointer, struct, union~~

Variables

A variable's name can be any legal identifier — an unlimited-length sequence of Unicode letters and digits, beginning with a letter, the dollar sign "$", or the underscore character "_". Subsequent characters may be letters, digits, dollar signs, or underscore characters

Declaration [+ Initialization]

byte a;

int value = 100;

final double PI = 3.14;

boolean isFebruary = true;

long numberOfElements = 12345678L;

String myFavouriteDrink = "water";

Java naming conventions

Variables (cont.)

```
class Example {
  int a; //class member

  public void someMethod(int b) { //method argument
    a = b;
    int c = 10;//local to a method
    for(int d=0; d < 10; d++) {
      //local to a block of code
      c --;
    }
    try {
      a = b/c;
    } catch(ArithmeticException e){
      //exception handler argument
      System.err.println(e.getMessage());
    }
  }
}
```

Control Flow Statements

- Decision-making

  if-else, switch-case

- Looping

  for, while, do-while

- Exception handling

  try-catch-finally, throw

- Branching

  break, continue, return, ~~goto~~, *label:*

Arrays

- ## Declaration

    ```
    int[] a; byte b[];
    ```

    100 elements of type char

- ## Instantiation

    ```
    a = new int[10]; char c[] = new char[100];
    ```

- ## Initialization

    ```
    String colors[] = {"Red", "Yellow"};

    someMethod( new String[] {"Red", "Yellow"} );
    ```

- ## The size of an array

    ```
    a.length and not a.length()
    ```

Multi-dimensional Arrays

- ## Arrays of arrays

```
int[][] m2d = new int[10][20];

int[][][] m3d = new int[10][20][30];
```

- ## Copying arrays

```
System.arrayCopy

int a[]; int b[]; …   What about a = b?;
```

- ## Utility methods for arrays

```
java.util.Arrays

   - binarySearch, equals, fill, ...
```

Strings

- <u>char[]</u>

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

- <u>String</u> Immutable Object

  ```
  String s = "abc";   String s = "a" + "b" + "c";

  String s = new String("abc");

  String s = new String(data);
  ```

- <u>StringBuilder, StringBuffer</u>

  ```
  StringBuilder sb = new StringBuilder("a");

  sb.append("b").append("c");
  ```

Equality Testing

- <u>Arrays</u>

```
int a[] = {1, 2};
int b[] = {1, 2};
a == b / a.equals(b) / Arrays.equals(a,b)
```

- <u>Strings</u>

```
String s1 = new String("abc");
String s2 = new String("abc");
s1 == s2 / s1.equals(s2) / s1.compareTo(s2)
"abc" == "abc" 
```

Example of Using Chars and Strings

```java
/** Generates random words, using a given set of characters. */
public class Example {

  public static void main(String args[]) {
      Example app = new Example();
      int nbWords = 10; //how many words to generate
      final int alphabetSize = 26; //how many characters has the alphabet
      char[] latin = new char[alphabetSize]; //create the alphabet array
      for (int i = 0; i < latin.length; i++) {
          latin[i] = (char) ('a' + i); //a b c d ...
      }
      String words[] = app.generate(nbWords, latin);
    }

   public String[] generate(int n, char[] alphabet) {
      String[] words = new String[n];
      for (int i = 0; i < n; i++) {
          StringBuilder sb = new StringBuilder();
          while (true) {
              int pos = (int) (Math.random() * (alphabet.length + 1)) - 1;
              if (pos < 0) break;
              sb.append(alphabet[pos]);
          }
          words[i] = sb.toString();
      }
      return words;
  }
}
```

Command Line Arguments

```java
public class Main {
  public static void main (String args[]) {
    if (args.length < 3) {
      System.out.println("Not enough arguments!");
      System.exit(-1);
    }
    String str = args[0];
    int a = Integer.parseInt(args[1]);
    double x = Double.parseDouble(args[2]);
  }
}
```

`java Main "Hello World" 2016 1.8`

Bibliography

- ***The Java Tutorials***

  http://docs.oracle.com/javase/tutorial/

- ***The Java Language Specification,*** James Gosling, Bill Joy, Guy Steele, Gilad Bracha

- ***The Java Virtual Machine Specification***

  Tim Lindholm, Frank Yellin

- **Curs practic de Java**, C. Frăsinaru

- http://profs.info.uaic.ro/~acf/java