Java

CSCI-4448 - Boese

University of Colorado **Boulder**

Java History

- Problem
  - C/C++ are not portable
  - C/C++ are not platform independent
  - Emergence of WWW required portable
- James Gosling – Sun Microsystems

1995 Hot Java
1996 JDK 1.0
1997 JDK 1.1
1998 J2SE 1.2
2000 J2SE 1.3
2002 J2SE 1.4
2004 J2SE 5.0
2006 Java SE 6
2011 Java SE 7
2014 Java SE 8
*2016 Java SE 9**

Java vs C++

> *"If Java is a family sedan with seatbelts, 6 airbags and antiskid brakes, then C is the dragster racing car with no safety features. So, if you like to live life dangerously with adventure and risk - go for C! But for a safe family ride, Java is better."*
> –Ravi Reddy

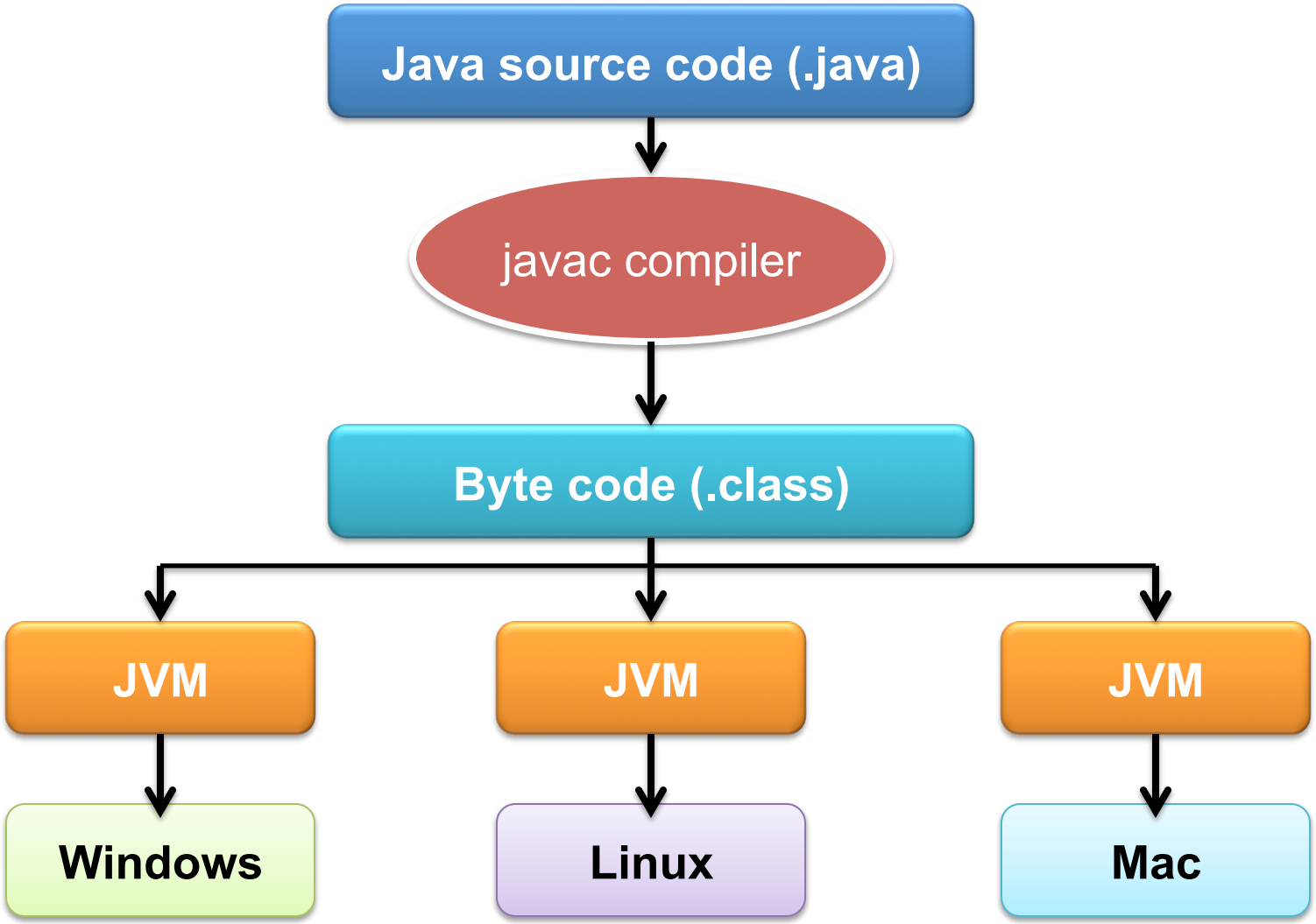
Java

- Interpreted
- *Write once, run anywhere*
- The biggest potential stumbling block is speed
  - Interpreted Java can run in the range of 5-20 times slower than C. Sometimes it is comparable. But: nothing prevents the Java language from being compiled and there are just-in-time (JIT) compilers that offer significant speed-ups.

Java Editions

- J2SE – Java 2 Standard Edition
  - Client-side standalone applications/applets
- J2ME – Java 2 Micro Edition
  - Apps for mobile devices/cell phones
- J2EE – Java 2 Enterprise Edition
  - Server-side apps (web)
    - *Java Servlets*
    - *Java ServerPages*
- J2FX – Java 2 Flash Player alternative

Java Program Execution



Java

Java: even a simple program is not simple.

```
public class Program1
{
 public static void main(String[] args)
 {
    System.out.println("Hello World");
 }
}
```

Some Similarities between C++ and Java

- Simple (primitive) types
  - `int, double, char`
- Control Structures
  - `if else, switch, while, for`
- Arithmetic expressions
- Both have a string type
  - C++ → string
  - Java → String
- Arrays
- Both have classes
- Both have a "main" function

Some Differences between C++ and Java

- Java has **automatic garbage collection**.
  C++ does not.
- C++ has **operator overloading**.
  Java does not.
- C++ says "**function**".
  Java says "**method**".

Every compilation unit in Java is a ***class***.

A ***program*** is a class
with a method named **`main`**:

```
public class Program1
{
 public static void main(String[] arg)
 {
      …
```

- In Java,
  every ***method*** is a member of some class.

  You cannot have a freestanding (global) function in Java.

  You can fake a "no classes" program in Java by making all methods static.

  But don't do it!

A Sample Java Class

```
public class PetRecord
{
    private String name;
    private int age = 0;          //in years
    public PetRecord(String initName, int initAge)
    {
        name = initName;
        if ((initAge < 0))
            System.out.println("Error");
        else
            age = initAge;
    }
    public void writeOutput()
    {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age + " years");
    }
}
```

C++ and Java divide a program into pieces (for separate compilation) in different ways.

Java vs C++

- C++: Traditionally has
  - interface (header) file,
  - implementation file(s),
  - application (driver) file.

  C++: Can confine a program to a single file if you want.

- Java: A compilation unit is always a class definition.
  - Every class is in a separate file *(except for some special cases).*
  - No header files.
  - Normally, you don't have *one-file* programs in Java.

Exception handling in Java

Option 1: throw it onwards

```
public class TextFileOutputDemo
{
   public static void main(String[] arg)
                              throws IOException
   {
      PrintWriter outputStream =
         new PrintWriter(…));
      outputStream.println("To file");
       …
```

Exception handling in Java

Option 2: Catch it

```
public class TextFileOutputDemo
{
 public static void main(String[] arg)
 {
  PrintWriter outputStream = null;
    try
    {
      outputStream = new PrintWriter(
        new FileOutputStream("out.txt"));
    }
    catch(FileNotFoundException e)
    {…}
  outputStream.println("To file");
   …
```

Need to Know

What you need to know

- ArrayList
- Class
  - *Instance variables*
  - *Static variables*
  - *Constants*
  - *Constructors*
  - *Getters/setters*
  - *toString method*
  - *Static vars/methods*
  - *Visibility modifiers*
  - *this*
  - *Overloading*
- Inheritance
  - *Abstract class*
  - *Protected*
  - *Java's Object class*
  - *Super*
  - *overriding*
- Interfaces
- instanceof
- Enum (basic)

```
enum Season { WINTER,
SPRING, SUMMER, FALL }
```

- What you don't need
  - ~~Inner classes~~