Introduction to Java

11 Buzzwords for Java

- Simple
- Object Oriented
- Network Savvy
- Robust
- Secure
- Architecture Neutral
- Portable
- Interpreted
- High performance
- Multithreaded
- Dynamic

Java: Simple

- Syntax is cleaned up version of C++
- No need of header files
- No pointer arithmetic/syntax
- No operator overloading
- …

Java: Object Oriented

- Object Oriented Design: Focus on Data (objects) and interfaces.
- Object oriented features comparable to C++
- More simplified because of different handling of multiple inheritance using ‘interfaces’.

Java: Network Savvy

- Much easier network programming compared to C++
- Simpler remote method invocation mechanism.

Java: Robust

- No pointer model: Eliminates possibility of overwriting memory or corrupting data
- Improved compiler: Detects problems that would otherwise show at runtime in other languages

Java: Secure

- Stack overrun, Memory corruption and Reading/Writing files without permission are much more difficult in Java.

Java: Architecture Neutral and Portable

- Java Virtual Machine
- Sizes of primitive data types are specified

Java: High Performance

- Just in time compilation.
- The bytecodes translated on the fly (at runtime) into machine code for the particular CPU the application is running on.

Java: Multithreading

- C++: Different multi-threading models exist for different operating systems
- (Unix: pthread.h; Windows: <windows.h>)
- Java: Consistent API. Different implementations for JVMs

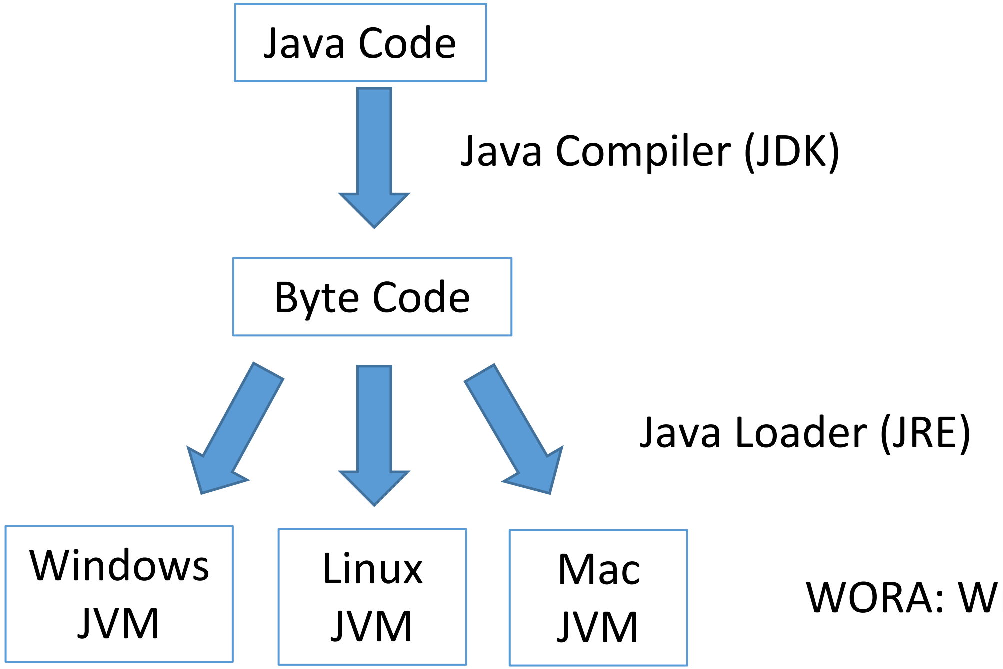
Java: Dynamic

- Support for libraries etc.- adding new methods to libraries without affecting client
- Possible to find runtime type information.

Java Programming Environment

- JDK
- JRE

- .java
- .class
- .jar



Java Code

Java Compiler (JDK)

Byte Code

Java Loader (JRE)

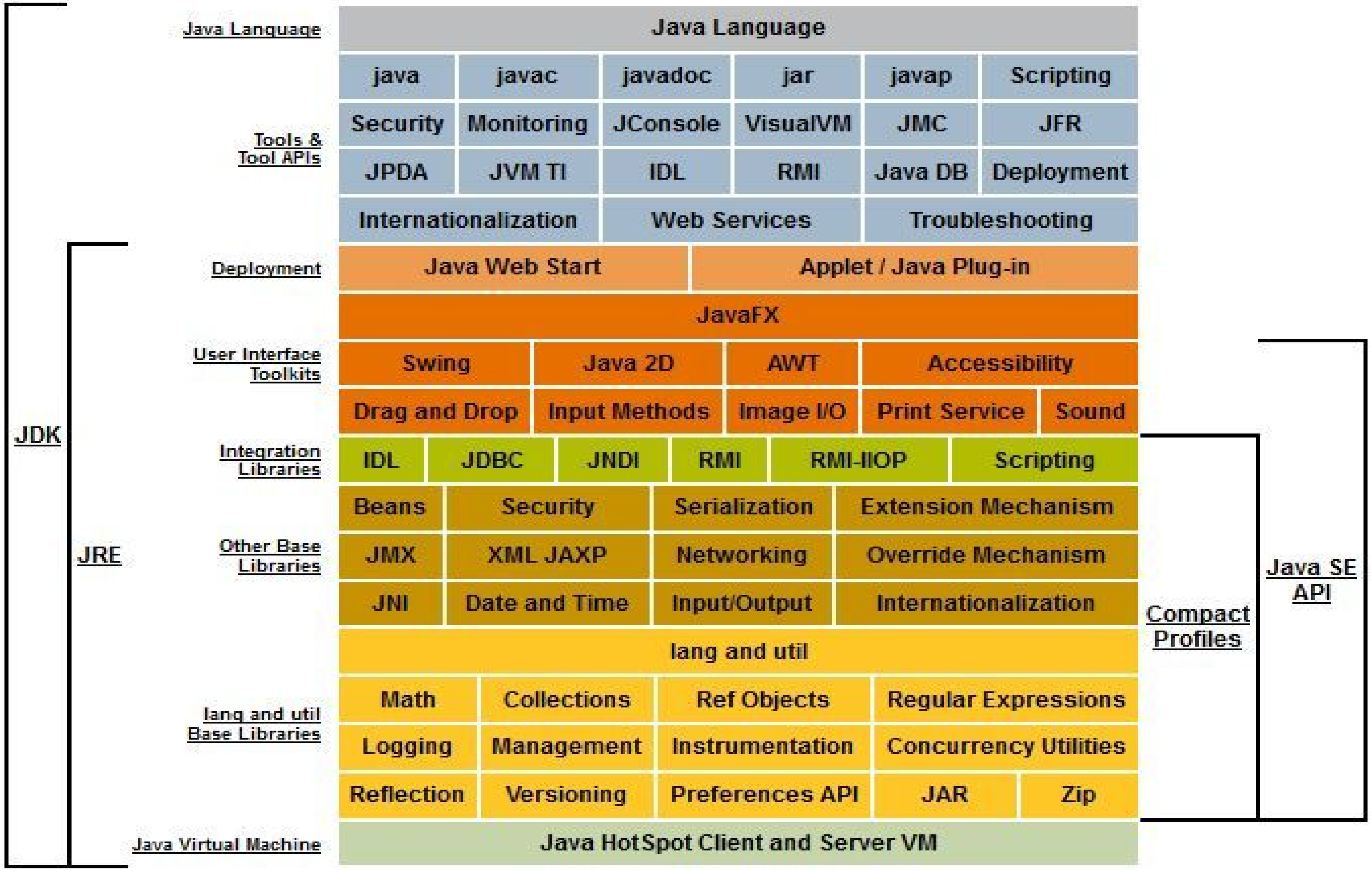
Windows JVM

Linux JVM

Mac JVM

WORA: Write Once Run Everywhere

We will use JDK 1.8.0_45 in this course



JDK

JRE

Java Language

Java Language

Tools & Tool APIs

java | javac | javadoc | jar | javap | Scripting

Security | Monitoring | JConsole | VisualVM | JMC | JFR

JPDA | JVM TI | IDL | RMI | Java DB | Deployment

Internationalization | Web Services | Troubleshooting

Deployment

Java Web Start | Applet / Java Plug-in

User Interface Toolkits

JavaFX

Swing | Java 2D | AWT | Accessibility

Drag and Drop | Input Methods | Image I/O | Print Service | Sound

Integration Libraries

IDL | JDBC | JNDI | RMI | RMI-IIOP | Scripting

Other Base Libraries

Beans | Security | Serialization | Extension Mechanism

JMX | XML JAXP | Networking | Override Mechanism

JNI | Date and Time | Input/Output | Internationalization

lang and util Base Libraries

lang and util

Math | Collections | Ref Objects | Regular Expressions

Logging | Management | Instrumentation | Concurrency Utilities

Reflection | Versioning | Preferences API | JAR | Zip

Java Virtual Machine

Java HotSpot Client and Server VM

Compact Profiles

Java SE API

Src: http://www.oracle.com/technetwork/java/javase/tech/index.html

Java Programming Environment

- IDE: Eclipse, Netbeans etc.
- In this course, we will use Eclipse
  - Version: Mars or Neon
  - Release: 4.5.0
  - Build id: 20150621-1200
- For developing/compiling/running from command line use ‘javac’ and ‘java’ commands

Your First Java Program: Hello World

```
public class MainClass {
    public static void main(String[] args) {
        //comment 1
        /*
         * more comment formats
         */
        System.out.println("Hello World");
    }
}
```

Note:

1. Naming Convention
    a. Class
    b. File
2. Braces
3. Comments
4. Console output

Ignore for now:

1. Class
2. Access Modifier

Java Variable Types

- Java is a strongly typed language
  - Every variable must have a type.
  - Restrictions on types intermixing: string can not added to a int or double.
- Java basic types: Platform independent
  - Integers: int (4 bytes), short(2 bytes), long (8 bytes), byte (1 bytes)
  - Reals: float (4), double (8) .
    - Default is double precision. Use ‘f’ for floats: 1.234f
  - Boolean: boolean (true/false)
  - Character: char (Unicode: UTF 16)

Java Variable Types

- Special support for character strings via the java.lang.String class (java.lang is imported in your program by default)
- Enclosing your character string within double quotes will automatically create a new String object;

String s = "my string object";

- String objects are *immutable*: once created, their values cannot be changed

Java Operators

- Basic: +, -, *, /
- Incremental: ++, --
  - Difference between b = a++ and b = ++a
- Combined: +=, -=, *=, /=
- Remainder (Mod): %
- Relational: <, >, >=, <=, ==, !=
- Logical: ||, &&
- Bitwise: |, &, ^ (XOR)

Sample Program

```
public class MainClass {
    public static void main(String[] args) {
        int a = 10;
        a++;
        float f = 20.0f;
        ++f;
        System.out.printf("int: %d, float: %.3f",a, f);
        // (We almost never use this function 'printf')
    }
}
```

Type Conversion

- Implicit conversion
  - Conversion of value from one type to another without any directive from the programmer. This is possible when
    - The two types are compatible.
    - The destination type is larger than the source type.

    Widening conversion takes place. Example: char to int implicit conversion

    ```
    char c = 'a';
    int k = c;
    long x = c;
    ```

Type Conversion

- Which of these is/are implicit conversion(s)
  1. byte to int
  2. int to byte
  3. int to char

Type Conversion

- Explicit conversion- narrowing conversion
    - Casting

      ```
      double d = 5.6;
      int k = (int)d;
      ```

Type Conversion

Java's widening conversions are

From a byte to a short, an int, a long, a float, or a double
From a short to an int, a long, a float, or a double
From a char to an int, a long, a float, or a double
From an int to a long, a float, or a double
From a long to a float or a double
From a float to a double

Narrow conversions

From a byte to a char
From a short to a byte or a char
From a char to a byte or a short
From an int to a byte, a short, or a char
From a long to a byte, a short, a char, or an int
From a float to a byte, a short, a char, an int, or a long
From a double to a byte, a short, a char, an int, a long, or a float

Answer!

State whether the following statements are correct:

1. float f = 234.56F;
   short s = (short)f;
2. float f = 32.3;
3. float f = (float) 32.3;
4. byte b = 3;
   b = b + 7;

Answer!

State whether the following statements are correct:

1. float f = 234.56F;
   short s = (short)f;
2. float f = 32.3;
3. float f = (float) 32.3;
4. byte b = 3;
   b = b + 7;

Operator Precedence

Operators	Precedence
!, ++ ,-- (unary operators)	First (Highest)
*, /, %	Second
+, -	Third
<< >>	Fourth
<, <=, >=, >	Fifth
==, !=	Sixth
&	Seventh
^	Eighth
|	Ninth
&&	Tenth
||	Eleventh
= (assignment operator)	Last(lowest)

Answer!

```
System.out.println( 3 + 3 * 2 );
System.out.println( 3 * 3 - 2 );
System.out.println( 3 * 3 / 2 );
System.out.println( 1 * 1 + 1 * 1);
System.out.println( 1 + 1 / 1 - 1);
System.out.println( 3 * 3 / 2 + 2);

int x = 1;
System.out.println( x++ + x++ * --x );

x = 1;
System.out.println( x << 1 * 3 >> 1);

x = 0xf;
System.out.println( 0xf & 0x5 | 0xa );
```

Answer!

```
System.out.println( 3 + 3 * 2 );              9
System.out.println( 3 * 3 - 2 );              7
System.out.println( 3 * 3 / 2 );              4
System.out.println( 1 * 1 + 1 * 1);           2
System.out.println( 1 + 1 / 1 - 1);           1
System.out.println( 3 * 3 / 2 + 2);           6

int x = 1;
System.out.println( x++ + x++ * --x );        5(post increment has higher precedence over pre-increment)

x = 1;
System.out.println( x << 1 * 3 >> 1);         4

x = 0xf;
System.out.println( 0xf & 0x5 | 0xa );        15
```

Scope

Block Scope

- Java statements surrounded by a pair of braces
- Define the scope of your variables

```
public static void main(String[] args)
{
   int n;
   . . .
   {
      int k;
      . . .
   } // k is only defined up to here
}
```

```
public static void main(String[] args)
{
   int n;
   . . .
   {
     int k;
     int n; // ERROR--can't redefine n
in inner block
     . . .
   }
}
```

Control Statements: If-else conditions

```java
if (yourSales >= 2 * target)
{
   performance = "Excellent";
}
else if (yourSales >= target)
{
   performance = "Satisfactory";
}
else
{
   System.out.println("You're fired");
}
```

Control Statements: Switch case condition

```
public class Test {
   public static void main(String args[]){
      char grade = 'C';
      switch(grade)
      {
        case 'A' :System.out.println("Excellent!");
                  break;
        case 'B' :
        case 'C' :System.out.println("Well done");
                  break;
        case 'D' :System.out.println("You passed");
        case 'F' :System.out.println("Better try again");
                  break;
        default :System.out.println("Invalid grade");
      }
   }
}
```

Control Statements: while, do-while loops

```
while (balance < goal)
{
   balance += payment;
   double interest=balance*interestRate/100;
   balance+= interest;
   years++;
}
System.out.println(years + " years.");
```

```
do
{
   balance += payment;
   double interest=balance*interestRate/100;
   balance += interest;
   year++;
   // print current balance
   . . .
   // ask if ready to retire and get input
   . . .
}
while (input.equals("N"));
```

Control Statements: for loop

```
for (int i = 10; i > 0; i--)
   {
System.out.println("Counting down . . . " + i);
}
System.out.println("Time up!");
```

Scope of i ?

```
int i;
for (i = 10; i > 0; i--)
   {
System.out.println("Counting down . . . " + i);
}
System.out.println("Time up!");
```

Scope of i ?

Input output from/to console

- Scanner
- Print, println

```
public class InputTest
{
     public static void main(String[] args)
     {
        Scanner in = new Scanner(System.in);
        // get first input
        System.out.print("What is your name? ");
        String name = in.nextLine();

        // get second input
        System.out.print("How old are you? ");
        int age = in.nextInt();

        // display output on console
        System.out.println("Hello, " + name + ". Next year, you'll be "
+ (age + 1));
     }
  }
```

Functions in Java

```
public class MainClass {
	public MainClass() {
	}
	public int Mult(int a, int b) {
		return a*b;
	}
	public static void main(String[] args) {
		MainClass m = new MainClass();
		System.out.println("Function output = " + m.Mult(10, 20));
	}
}
```

- Ignore access modifier for now
- Parameter passing
- return

Strings in Java

- operations on strings

```
public class StringDemo {

   public static void main(String args[]) {
      String palindrome = "Dot saw I was Tod";
      int len = palindrome.length();
      System.out.println( "String Length is : " + len );
   }
}
```

Example

```
String str = “Hello”;

System.out.println(str); // Output?

str = “Hello Students”;

System.out.println(str); // Output?

str.concat(“, Welcome!”);

System.out.println(str); // Output?

System.out.println(str.concat(", Welcome!")); // Output?

str = str.replace(“Hello”, “Hi”);

System.out.println(str); // Output?
```

Arrays

- 1D, 2D array(array of arrays)
- .length
- Array class
- Forloop on arrays

```
public class TestArray {

   public static void main(String[] args) {
      double[] myList = {1.9, 2.9, 3.4, 3.5};

      // Print all the array elements
      for (double element: myList) {
         System.out.println(element);
      }
   }
}
```

StringBuilder

- Like String objects, except that they can be modified.
- Internally treated like variable-length arrays that contain a sequence of characters.
- At any point, the length and content of the sequence can be changed through method invocations
- Strings should always be used unless string builders offer an advantage.
    - eg. if you need to concatenate a large number of strings, appending to a StringBuilder object is more efficient

StringBuilder

```
// creates empty builder, capacity 16
StringBuilder sb = new StringBuilder();
// adds 9 character string at beginning
sb.append("Greetings");
```