CMSC131

Introduction to your Introduction to Java

Why Java?

- It’s a popular language in both industry and introductory programming courses.
- It makes use of programming structures and techniques that can be transferred to using a variety of other languages.
- Development on different platforms without platform-specific differences for what we will be doing is more straight-forward.

Phrases you might hear…

- Garbage collection
  - …when we talk about requesting memory to hold information for our program to use…

- Cross-platform executables
  - …the modules we build and even full applications can be used under different operating systems due to the Java virtual machine…

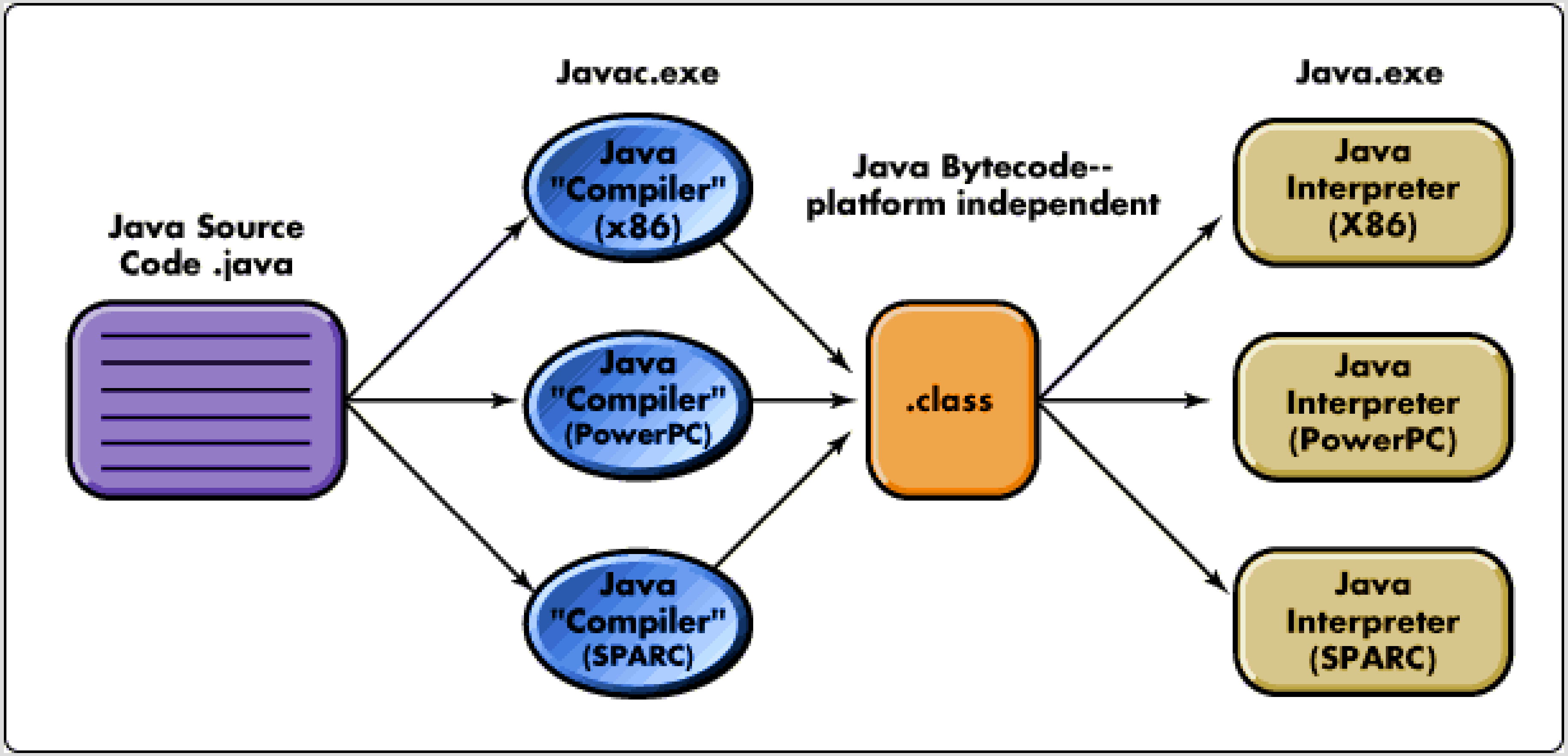
The JVM

- The Java compiler does not compile to the machine code of an actual physical machine architecture, but rather to “bytecode” which is run on a Java Virtual Machine.

- If you compile a JVM for a new operating system, then (in theory) all of your existing Java programs could run there too.

- There are times when (for efficiency) Java bytecode is actually compiled to a specific architecture’s machine code (no longer portable).

Java to Bytecode to Execution



http://support.novell.com/techcenter/articles/img/ana1997070102.gif

How many bytes in a kilobyte?

1. 8
2. 128
3. 1000
4. 1024
5. none of the above

0 of 5

45

Countdown Timer On Slide

HelloWorld2.java example

```
public class HelloWorld2 {
  public static void main(String[] args) {
     System.out.println("Hello World!");
     myMethod("Hi", "There");
  }
  public static void myMethod(String first, String second)
  {
    System.out.println(first+" "+second);
    System.out.println(second+" "+first);
  }
}
```

Methods (operations that can be performed)

```
public static void main(String[] args)
public static void myMethod(String first, String second)
```

HelloWorld2.java example

```
public class HelloWorld2 {
  public static void main(String[] args) {
     System.out.println("Hello World!");
     myMethod("Hi", "There");
  }
  public static void myMethod(String first, String second)
  {
    System.out.println(first+" "+second);
    System.out.println(second+" "+first);
  }
}
```

Operands/Parameters (data the method uses)

`String[] args`

`String first, String second`

HelloWorld2.java example

```
public class HelloWorld2 {
  public static void main(String[] args) {
     System.out.println("Hello World!");
     myMethod("Hi", "There");
  }
  public static void myMethod(String first, String second)
  {
    System.out.println(first+" "+second);
    System.out.println(second+" "+first);
  }
}
```

Output Commands (in this case to the console)

```
System.out.println("Hello World!");
System.out.println(first+" "+second);
System.out.println(second+" "+first);
```

Output

- We will see two basic output options for sending text to the console:

  System.out.println

  System.out.print

- The difference between the two is that the first one adds an “end of line” character at the end of whatever it has sent to the console.

- You can request the contents of variables to be displayed (which a variable may or may not support) or you can ask for a literal string to be displayed (contained in quotation marks in your output command).

HelloWorld2.java example

```
public class HelloWorld2 {
  public static void main(String[] args) {
     System.out.println("Hello World!");
     myMethod("Hi", "There");
  }
  public static void myMethod(String first, String second)
  {
    System.out.println(first+" "+second);
    System.out.println(second+" "+first);
  }
}
```

Method Call

```
myMethod("Hi", "There");
```

Method Signatures

```
public static void myMethod(String first, String second)
```

public: we will discuss “visibility” later

static: we will discuss this later as well

void: the “type” of the value the method will return (in this case it does not return a value so we say the type is “void”)

myMethod: method name

String first, String second: defining memory spaces and local names for values that will be passed into the method

DoSomeMath.java example

```
public class DoSomeMath {
  public static void main(String[] args) {
  int x;
  int y;    ← Variable Declarations
     x = 17;
     y = 23;   ← Commands/Instructions
     printSum(x,y);
     printProd(x,y);
     printQuot(x,y);
     printQuot(y,x);
  }
  public static void printSum(int first, int second){
     System.out.println(first+second);
  }
  public static void printProd(int alpha, int beta){
     System.out.println(alpha*beta);
  }
  public static void printQuot(int alpha, int second){
     System.out.println(alpha/second);
  }
}
```

Variables

In the DoSomeMath example, there were some local variables…

declared

```
int x;
int y;
```

assigned to

```
x = 17;
y = 23;
```

and used

```
printSum(x,y);
```

In short, variables have a data type (such as int) and refer to space within the computer's memory where their values (such as 17) are stored.

You can assign values to them and read those values back out from them.

AddOne.java example

```
public class AddOne {
  public static void main(String[] args) {
  int x;
     x = 17;
     System.out.println(x);
     printPlus1(x);
     System.out.println(x);
  }
  public static void printPlus1(int val){
     val = val + 1;
     //NOTE: Due to how the int data type works,
     //   x back in the calling method doesn't change!
     System.out.println(val);
  }
}
```

Comments

- There are times that you will want to leave notes for yourself and other programmers within the code without it being executed.
- This is where comments come in!
- There are two types of comments in Java:

```
/* comment goes until the “close” */
/* comment goes until the “close”
      which might not be on the same line
      as where the “open” appears  */
// comment goes from “start” to the end of that line
```

Some Primitive Types

- Integer values (and how many bytes they get)
  - byte (1), short (2), int (4), long (8)
- Real numbers (and how many bytes they get)
  - float (4), double (8)
- Individual Unicode characters
  - char (2)
- Boolean truth values
  - boolean (1)

Range of Values

- The range of values depends on several things, one of which is how much memory is available for storing the value and another is how Java interprets the 0s and 1s stored in that memory.
- A **byte** is one byte in size and can store a value between **-128** and **+127** but **boolean** is also one byte in size yet it can only store two values; **true** or **false**.
- An **int** is four bytes in size and can store a value between **-2,147,483,648** and **+2,147,483,647** but a **float** is also four bytes in size yet it can store numbers between (roughly) **-3.4×10^{38}** and **3.4×10^{38}** (though only with 7 digits of precision).

If a short integer holds values from -32,768 to 32,767, what happens if it is holding 32,767 and you add one?

1. The program crashes.
2. The program automatically allocates a larger integer to hold larger values.
3. The value becomes 32,767.
4. The value becomes -32,768.
5. The value becomes 0.

0 of 5

45

Countdown Timer On Slide

float -vs- double

By default, real number values are assumed to be "double" values.

float val = 17.5; won't work… it needs to be **float val = 17.5F;**

Some Math Operations

- Addition: **+**
- Subtraction: **-** (also used as unary “negative”)
- Multiplication: *****
- Division: ***/***
  - Integer division discards remainder

    17 / 5 yields **3**

- Modulus: **%**
  - Returns what the remainder would be if the two operands were “divided”

    17 % 5 yields **2**

Numeric Comparison Operators

• Less than:	**<**
• Greater than:	**>**
• Less than or equal to:	**<=**
• Greater than or equal to:	**>=**
• Is equal to:	**==**
• Is not equal to:	**!=**

NOTE = is for assignment, == is for comparison

Comps.java example

```
public class Comps {
  public static void main(String[] args)
  {
  boolean flag;
     flag = (7<14);
     System.out.println(flag);
     flag = (17<14);
     System.out.println(flag);
     flag = (17=14); //THIS WON'T COMPILE
  }
}
```

The **String** type

- The **String** data type is not a primitive type.
- It is a **Class** and when we declare one to use, it is called an **Object** (we will discuss these ideas more later).
- As we saw earlier, you can concatenate two **String** objects using the **+** operator.
- Note that the **+** operator can also be used for math operations. The left operand "decides" which form of **+** is being used.

"S" is a **String** of length 1

'S' is a **char**

Some Special Character Values

single quote:	\‘
double quote:	\”
new line (crlf):	\n
tab:	\t
single backslash:	\\

SomeStrings.java example

```
public class SomeStrings {
  public static void main(String[] args) {
     String name = "Sam";
     char letter = 'A';
     int number1 = 14;
     float number2 = 17.5F;

     System.out.println("Hi\nThere\tClass\\\\\n\n");

     System.out.println(name+letter+number1+number2);
     System.out.println(name+letter+(number1+number2));
  }
}
```

String Comparison

- **String** objects are compared using **Methods** rather than **Operators**.

  string1.equals(string2) will return a boolean value

  string1.compareTo(string2) will return an integer value using the following rules based on the lexicographical order of the strings:

  - If the result is less than 0 string1 precedes string2
  - If the result is equal to 0 string1 matches string2
  - If the result is greater than 0 string1 succeeds string2

NOTE the negative value doesn't have to be -1 and the positive value doesn't have to be +1

Input using the Scanner class

- The **Scanner** class is included as part of the utility libraries available in Java 5.0 (or later) but we need to **import** it to make use of it.

```
import java.util.Scanner;
```

- It allows us to obtain data from an input source (such as a keyboard) and also convert that data into the format of different Java types.

Creating and Using

- We will need to declare and create a Scanner object:

```
Scanner myScanner = new Scanner(System.in);
```

- We can then use any of Scanner's methods to read and convert data, such as...
  - **`nextBoolean()`**
  - **`nextFloat()`**
  - **`nextInt()`**
  - **`next()`**

    ```
    //reads/returns characters until next whitespace
    ```
  - **`nextLine()`**

    ```
    //reads/returns characters until the end of the input line
    ```

SimpleInput.java example

```
import java.util.Scanner;

public class SimpleInput {
  public static void main(String[] args) {
     Scanner myScanner = new Scanner(System.in);
     int i;
     float f;
     String s;

        i = myScanner.nextInt();
        System.out.println("The integer was a " + i);

        s = myScanner.next();
        System.out.println("The next \'word\' was " + s);

        s = myScanner.nextLine();
        System.out.println("Rest of the line was " + s);
  }
}
```