CMSC 341

Introduction to Java

Based on tutorial by Rebecca Hasti at

http://pages.cs.wisc.edu/~hasti/cs368/JavaTutorial/

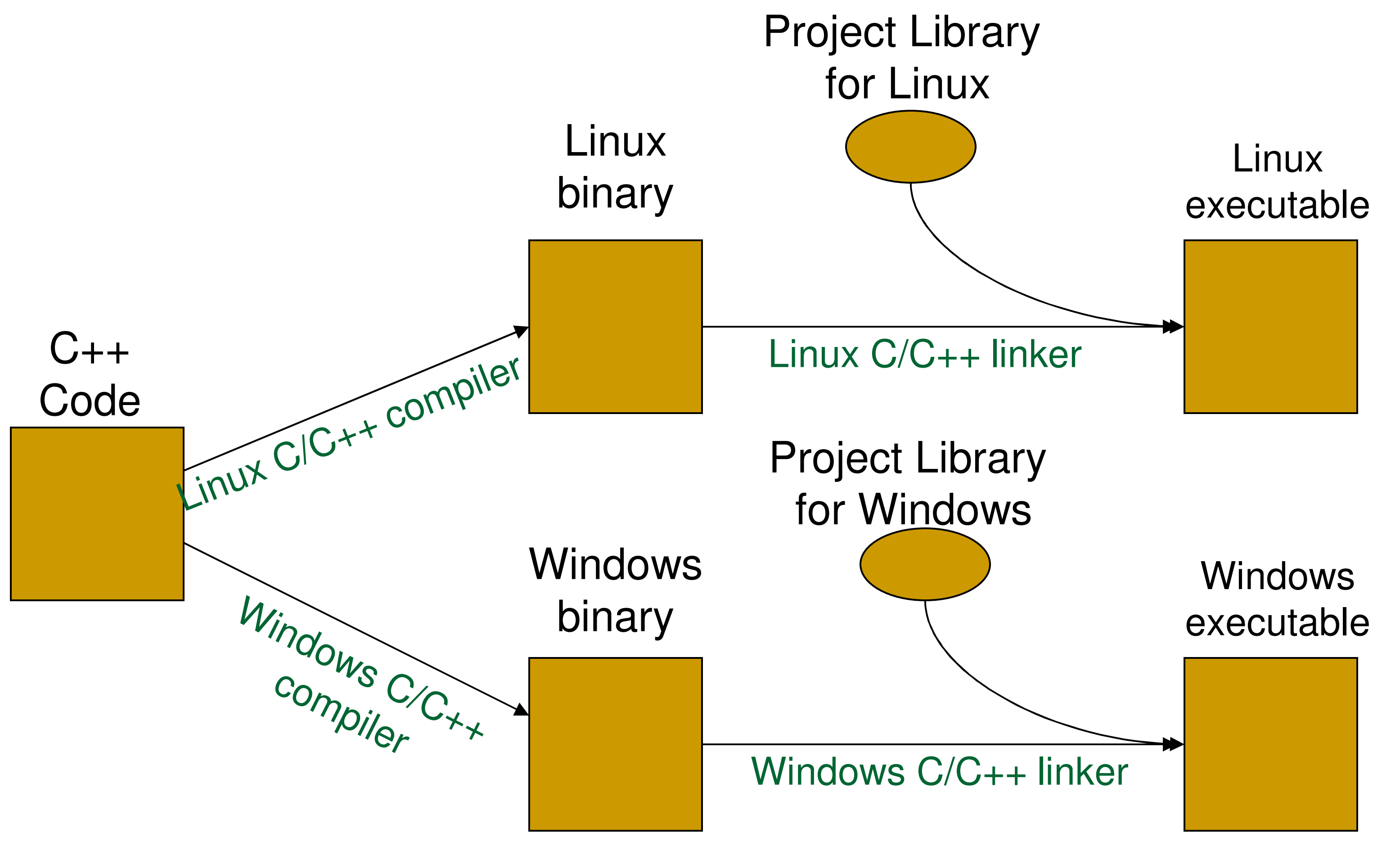
Important Java Concepts and Terminology

- JRE is the Java Runtime Environment and it creates a virtual machine within your computer known as the JVM (Java Virtual Machine). JRE is specific to your platform and is the environment in which Java byte code is run.
- JDK (formerly SDK) is the Java Development Kit.

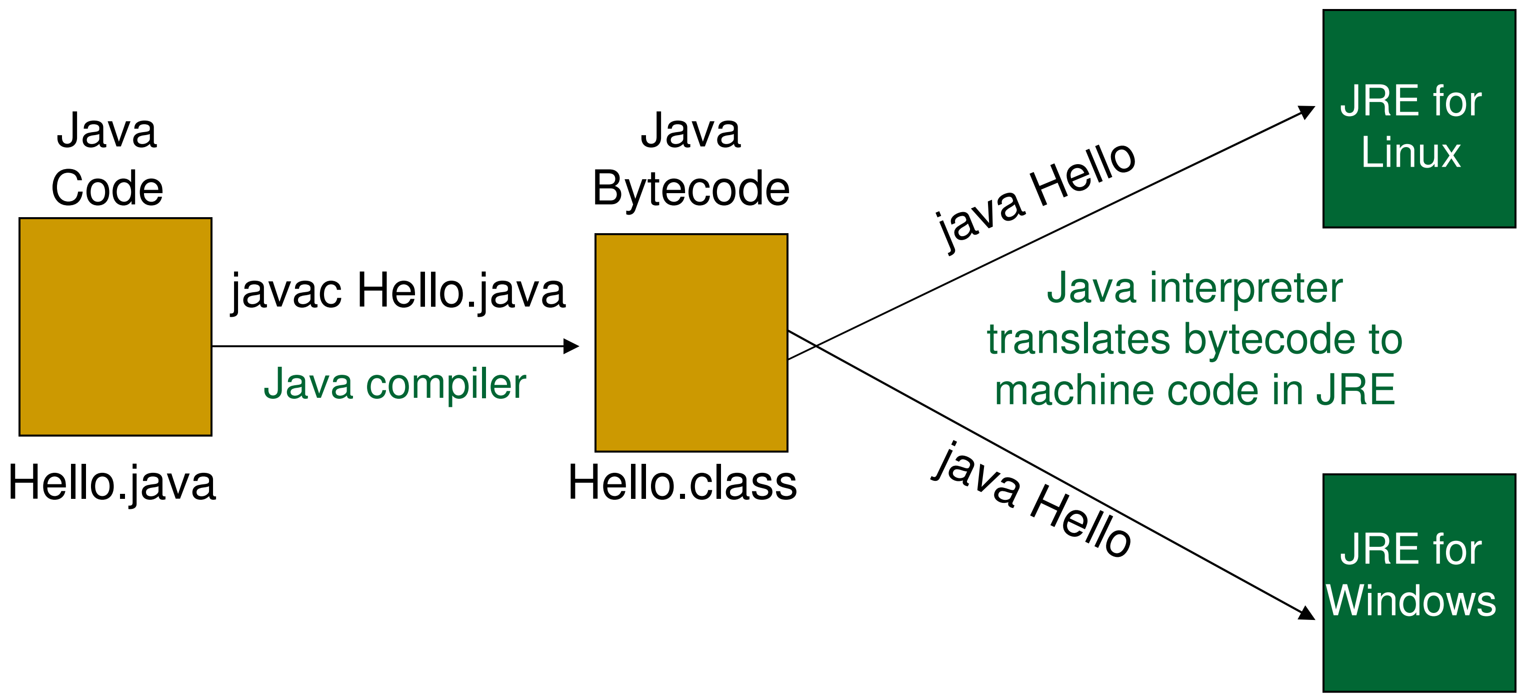
  JDK = JRE + development tools
- J2SE is the Java 2 Platform Standard Edition, which you will be using in this course to build stand alone applications.
- To learn more about JDK, JRE, etc., visit

  http://java.sun.com/javase/technologies/index.jsp

Running and Compiling C/C++



Running and Compiling Java



JRE contains class libraries which are loaded at runtime.

Important Java Concepts

- Everything in Java must be inside a class.
- Every file may only contain one public class.
- The name of the file must be the name of the class appended to the java extension.
- Thus, *Hello.java* must contain one public class named *Hello*.

Methods in Java

The *main* method has a specific signature.

- Example: “Hello world!” Program in Java

```
public class Hello
{
  public static void main(String args[])
  {
    System.out.println(“Hello world!”);
  }
}
```

⇦ Notice no semi-colon at the end!

Methods in Java (cont.)

- All methods must be defined inside a class.
- Format for defining a method:

```
[modifiers] return_type method_name([param_type param]*)
{
        statements;
}
```

- For ***main***, modifiers must be ***public static***, return type must be ***void***, and the parameter represents an array of type String, ***String []***. This parameter represents the command line arguments when the program is executed. The number of command line arguments in the Hello program can be determined from *args.length*.

Static Method Invocation

- Methods are invoked using the dot notation.
- Methods are either static or instance methods.
- Static methods do not have access to instance data.
- Static methods are typically invoked using the class name as follows:

```
Math.random();
```

Instance Method Invocation

- Create an instance of the class and have object invoke method.
- *System.out* is an object in the System class that is tied to standard output. We invoke the println() and print() methods on the object to write to standard output.

```
System.out.println("Hello world!");
```

Instance Method Invocation (cont.)

- The *println* and *print* methods have been overloaded so that they can convert all 8 of Java’s primitive types to a *String*.
- The + sign is overloaded to work as a concatenation operator in Java if either operand is a *String*.

```
int x =15, y = 16;
System.out.println(x + y + “/” + x + y);
```

Instance Method Invocation (cont.)

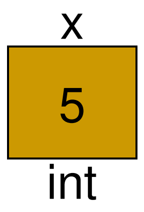
- To invoke an instance method, you must first create an instance of the class (an object) and then use the object to invoke the method.

```
StringBuffer phrase;
phrase = new StringBuffer(“Java is fun”);
phrase.replace(8,11, “cool”);
System.out.println(phrase);
```

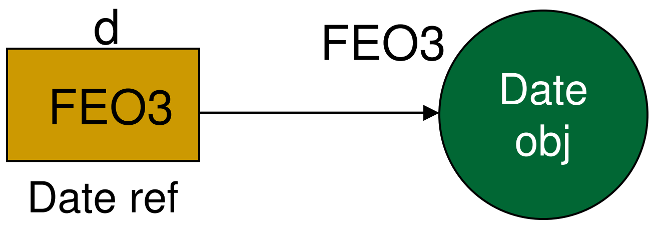
Data Types

- There are two types of data types in Java – primitives and references.
- Primitives are data types that store data.
- References, like pointers and references in C++, store the address of an object, which is encapsulated data.

```
int x = 5;
```



```
Date d = new Date();
```



Primitive Data Types

- Java has 8 primitive data types which always allocate the same amount of memory in JVM.
  - Integral
    - byte – 8 bits
    - short – 16 bits
    - int – 32 bits – default for integer literals
    - long – 64 bits

```
int x = 5;
short y = 03;
long z = 0x23453252L;
```

Primitive Data Types (cont.)

- 8 primitive data types (cont.)
  - Floating point
    - double - 64 bits - default for literal decimal value

```
double d = 234.43;
double db = 123.5E+306;
```

    - float - 32 bits - literal value must contain a F or f to avoid compiler errors

```
float f = 32.5f;
```

Primitive Data Types (cont.)

- 8 primitive data types (cont.)
  - Logical
    - boolean - 1 bit

      ```
      boolean a = true;
      boolean b = 5 < 3;
      ```
  - Textual
    - char- 16 bit - Unicode

      ```
      char c = ‘A’;
      char d = ‘\u04a5’
      char e = ‘\t’;
      char f = 96;
      ```

Reference Data Types

- Reference data types contain an address and function like pointers without the complex syntax.
- In the following code, the second line does not call a copy constructor, but rather you will have two references pointing to the same object.

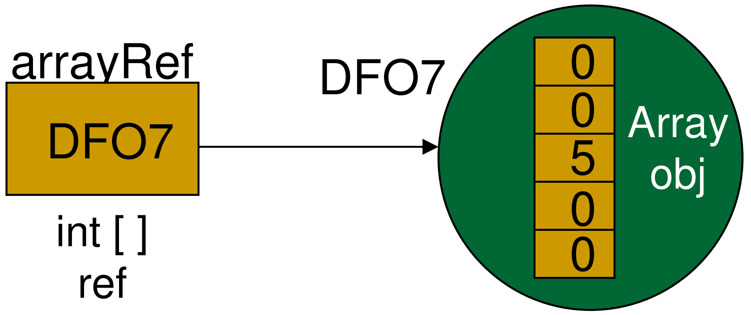
```
Date d = new Date();
Date e = d;
```

- Java tackled the problem of memory leaks in C++ by
  - not allowing the programmer to have direct access to the memory (i.e. no more pointer arithmetic),
  - checking array bounds at runtime, and
  - having a garbage collector in the JVM that periodically reallocates memory that is not referenced.

Arrays

- Arrays in Java are objects. The first line of code creates a reference for an array object.
- The second line creates the array object.

```
int [] arrayRef;
arrayRef = new int[5];
arrayRef[2] = 5;
```



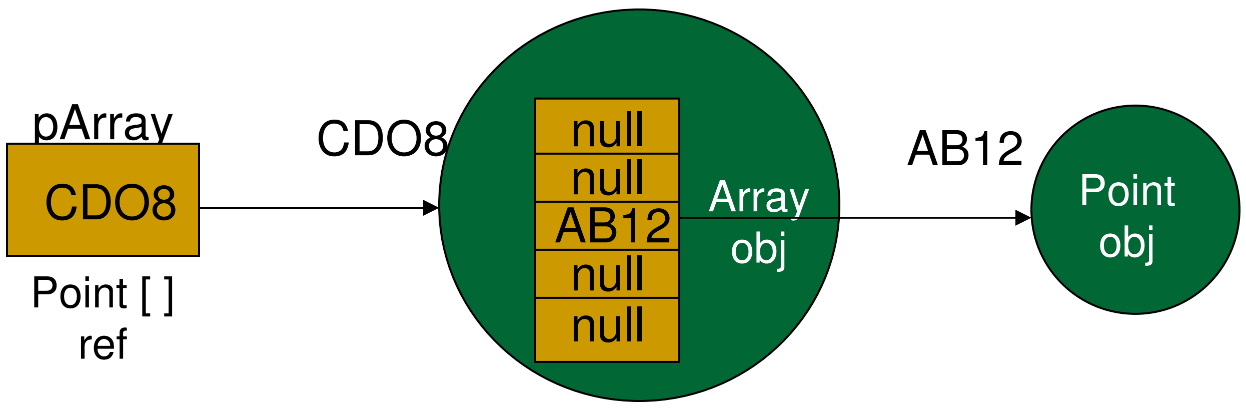
Arrays (cont.)

- All primitive data in an array is initialized to its zero value.
  - boolean - false
  - char – '\u0'
  - byte, short, int, long, float, double – 0
- All references are initialized to null.
- All arrays have a length property that gives you the number of elements in the array.
  - *args.length* is determined at runtime

Arrays (cont.)

- An array of objects is an array of object references until the objects are initialized.

```
Point pArray [] = new Point[5];
pArray[2] = new Point();
```



Arrays (cont.)

- Arrays may also be initialized when they are declared using the following syntax.

```
int intArray[]={1,2,3,4,5};
Point pArray[]={ new Point(1,2),
                 new Point(3,4),
                 new Point(5,6) };
```

Arrays (cont.)

- Because arrays are objects and the name of an array is its reference, arrays in Java can grow or shrink upon reassignment.
- Also, the location of the square brackets can change.

```
int [] aArr = new int[5];
int bArr [] = new int[3];
bArr = aArr; // now both are pointing
             // to same array and have
             // length of 5
```

Arrays (cont.)

- The System class provides an *arraycopy* method that performs a shallow copy of one array to another. Use *System.arraycopy* to copy an array of primitive data, not for an array of references.

number of elements

```
System.arraycopy(srcArray, 4, destArray, 3, 2);
```

Index in source

Index in target

Arrays (cont.)

- The declaration of array is carried through a comma separated list. The following declares two integer arrays.

```
int [] a, b;
```

Multidimensional Arrays

- The following declares a two-dimensional array, a reference.

```
int [][] twodim;
```

- The following creates the array with twenty elements initialized to 0.

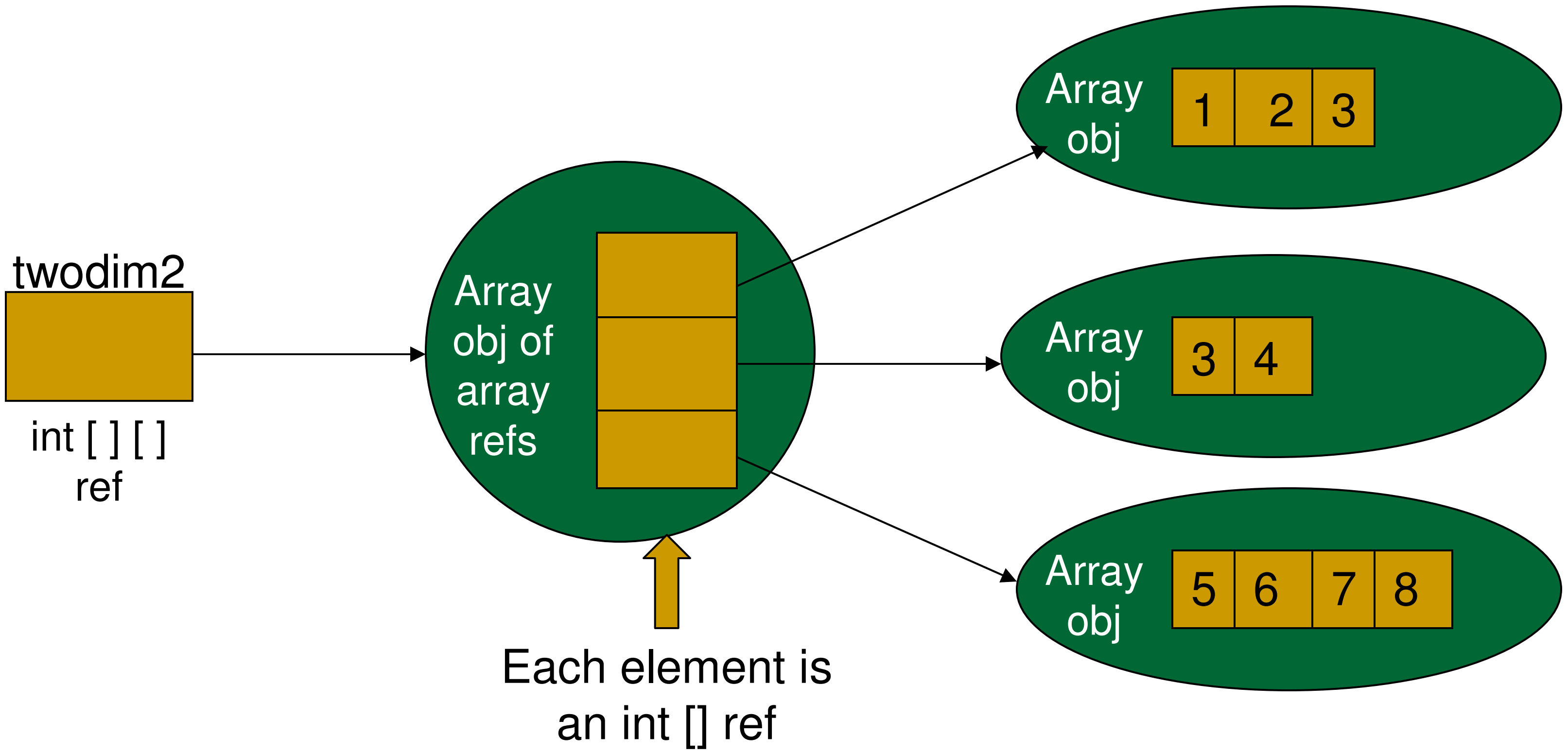
```
twodim = new int [4][5];
```

- The following does both at the same time. Notice the array is not rectangular.

```
int [][] twodim2 = {{1,2,3}, {3,4}, {5,6,7,8}};
```

Multidimensional Arrays (cont.)

- A pictorial rendition of twodim2.


twodim2
int [ ] [ ] ref
Array obj of array refs
Array obj
1 2 3
Array obj
3 4
Array obj
5 6 7 8
Each element is an int [] ref