Lecture 12 Supplement

Java Classes and Objects Continued

Dr. Martin O'Connor
CA166
www.computing.dcu.ie/~moconnor

Table of Contents

Does Java Pass by Value or Pass by Reference

- Every beginner on the road to programming (and unfortunately some seasoned professionals) struggles with this question.

- When an argument (that is a variable, object, array, etc) is passed to a method, is the argument passed by value or by reference?

- To pass by value, means the method is given a copy of the value stored by the argument.

- To pass by reference means the method is given the reference (the address in memory) of the argument.

Does Java Pass by Value or Pass by Reference

- **Java passes everything *by value*.**

- When we say everything, we mean everything – objects, arrays (which are objects in Java), primitive types (int, float, char, byte, etc) are all passed by value

- When a method that receives parameters is invoked (or called), Java makes a copy of the values of the arguments and the method processes those copies. The method does not process the original values of the parameters. That is why we use the term *pass by value*

Does Java Pass by Value or Pass by Reference

- Suppose we have a method called incrementNumber that takes one parameter (an int) and increments it by 2

```
public static void incrementNumber (int var)  {
  var = var + 2;
}
```

- What is the value of myNum printed out to the screen?

```
public static void main (String[] args)
{
    int myNum = 3;

    incrementNumber(myNum);

    System.out.println("The value of MyNum is : " + myNum);
}
```

Does Java Pass by Value or Pass by Reference

- The program will print out “3”, not “5”
- Why does it print out a 3 when we clearly added a 2 to myNum in the incrementNumber method?
- It prints out a “3” because Java passes arguments by value – which means that when the call to “incrementNumber” is made, Java will **create a copy** of the “myNum” variable and **that copy is what gets passed** to the incrementNumber method – and **not** the original myNum variable stored in the “main” method.
- Thus, whatever happens to “myNum” inside the incrementNumber method does not affect the “myNum” variable inside the main method.

Does Java Pass by Value or Pass by Reference

- If we wanted the myNum variable inside main to be incremented by 2 (using the method incrementNumber), then incrementNumber should return the result of its computation, as shown below

```
class TestIncrement2 {

    public static void main (String[] args)
    {
        int myNum = 3;

        myNum = incrementNumber(myNum);

        System.out.println("The value of MyNum is : " + myNum);
    }

    public static int incrementNumber (int var)  {
      return (var + 2);
    }
}
```

- Output to screen: `The value of MyNum is : 5`

Does Java Pass by Value or Pass by Reference

- **Are objects passed by reference in Java?**

- No, everything is passed by value in Java. So, objects are **not** passed by reference in Java

- **HOWEVER**, when objects are passed, a copy of the reference (or address in memory) is made and that copy is passed to the method (so the copy of the reference is still pointing to the original object in memory)

- **Remember that a variable of type CLASS stores a reference to the object (the address in memory of the object) and not the object itself.**

Does Java Pass by Value or Pass by Reference

- An example of passing an object by value

```
// Declare and instantiate an object by passing in a name and age
PersonClass variable1 = new PersonClass("Mary", 32);

// Declare an object but do not instantiate it
// (therefore variable2 will by default point to null)
PersonClass variable2;

// variable2 now references the same object as variable1
// Technically variable2 contains the same address (reference)
// as variable1
variable2 = variable1;

/* What happens as a result of this statement? */
variable2.set("Jack", 22);

System.out.println(variable1);
```

- (Let's just assume that System.out.println method above will print the name and age of a PersonClass object)
- What is printed to the screen?

Does Java Pass by Value or Pass by Reference

- The output is: `Jack 22`
- Remember, a copy of the address in variable1 is made and placed in variable2
- Thus, the setter method *variable2.set("Jack", 22)* changes the data of the object it is pointing at
- But variable1 is also pointing at the exact same object in memory as variable2. Thus, the data of the object pointed to by variable1 is changed

Does Java Pass by Value or Pass by Reference

- One more example – What is printed to the screen?

```
//create an object by passing in a name and age:
PersonClass variable1 = new PersonClass("Mary", 32);

PersonClass variable2;

//Both variable2 and variable1 now reference the same object
variable2 = variable1;


PersonClass variable3 = new PersonClass("Andre", 45);

// variable1 now points to variable3
variable1 = variable3;

//WHAT IS OUTPUT BY THIS?
System.out.println(variable2);
System.out.println(variable1);
```

Does Java Pass by Value or Pass by Reference

- The output is:

```
Mary 32
Andre 45
```

Comparing Object References

The equality operator (==) compares object references.

Example:

If *d1* and *d2* are two *SimpleDate* object references, then

```
( d1 == d2 )
```

evaluates to *true* only if *d1* and *d2* point to the same object, that is, the same memory location.

*** The equality operator does not compare the data (*month*, *day*, and *year*) in those objects.

Comparing Object Data

To compare object data, use the *equals* method

Return type	Method name and argument list
`boolean`	`equals( Object obj )` returns *true* if the data of the object *obj* is equal to the data in the object used to call the method; *false* otherwise

Example:

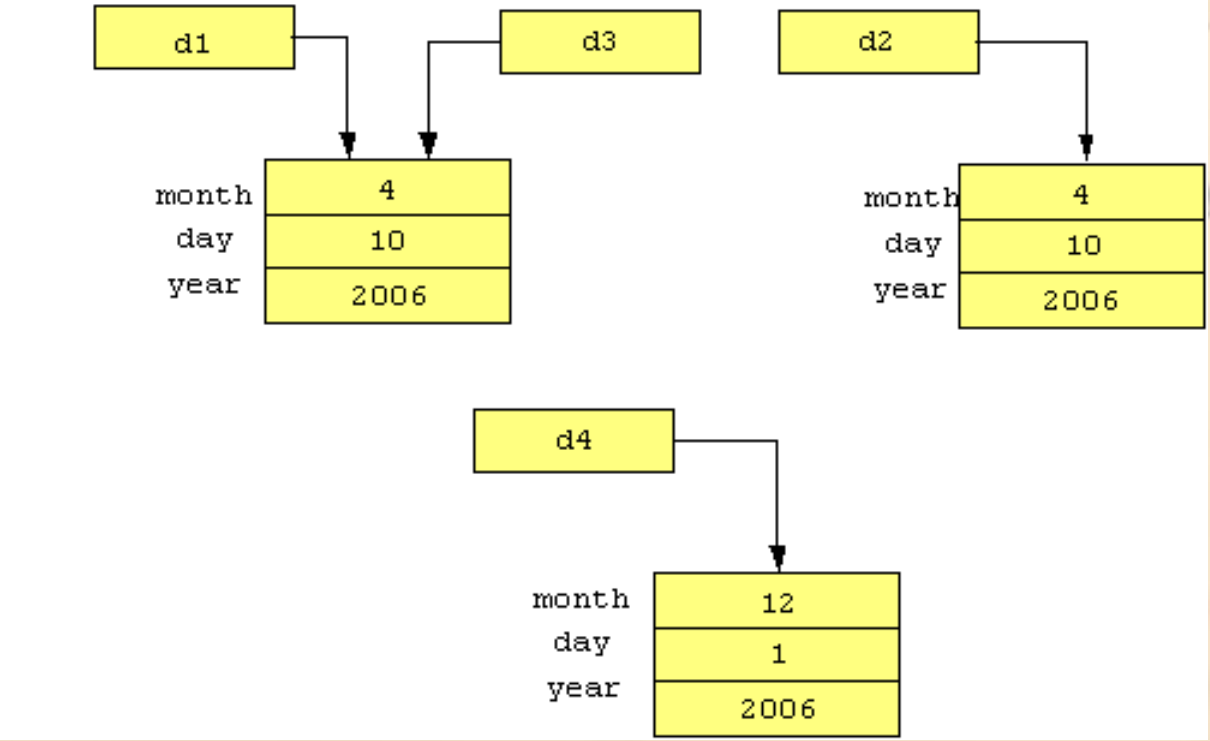
If *d1* and *d2* are *SimpleDate* object references,

```
d1.equals( d2 )
```

returns *true* if the *month*, *day*, and *year* of *d1* equals the *month*, *day*, and *year* of *d2*.

Comparing Objects

Comparing SimpleDate Objects



Common Error Trap

Do not use the equality operators (`==`, `!=`) to compare object data; instead, use the *equals* method.

Comparing *Strings*

- *Strings* are objects.
- Thus, to compare two *Strings*, use the *equals* method.
- Example: *s1* and *s2* are *Strings*

```
s1.equals( s2 )
```

  returns *true* only if each character in *s1* matches the corresponding character in *s2*
- Other methods of the *String* class also can be used for comparing *Strings*:

  equalsIgnoreCase

  compareTo

The *equalsIgnoreCase* Method

Return type	Method name and argument list
`boolean`	`equalsIgnoreCase( String str )` compares the value of two *Strings*, treating upper and lower case characters as equal. Returns *true* if the *Strings* are equal; *false* otherwise.

Example:

```
String s1 = "Exit", s2 = "exit";
if ( s1.equalsIgnoreCase( s2 ) )
   System.exit( 0 );
```

The *compareTo* Method

Return type	Method name and argument list
`int`	`compareTo( String str )` compares the value of the two *Strings*. If the *String* object is less than the *String* argument, *str*, a negative integer is returned. If the *String* object is greater than the *String* argument, a positive integer is returned; if the two *Strings* are equal, 0 is returned.

A character with a lower Unicode numeric value is considered less than a character with a higher Unicode numeric value. That is, *a* is less than *b* and *A* is less than *a*. (Lookup Google for Unicode values.)

The Conditional Operator (?:)

The conditional operator (?:) contributes one of two values to an expression based on the value of the condition.

- Some uses are
  - handling invalid input;
  - outputting similar messages.

Syntax:

```
( condition ? trueExp : falseExp )
```

If *condition* is *true*, *trueExp* is used in the expression.

If *condition* is *false*, *falseExp* is used in the expression.

Equivalent Code

The following statement stores the absolute value of the integer *a* into the integer *absValue*.

```
int absValue = ( a > 0 ? a : -a );
```

The equivalent statements using *if/else* are:

```
int absValue;
if ( a > 0 )
  absValue = a;
else
  absValue = -a;
```