Java Training

Day 3: Conditionals, Control-Flow, Scope, more Primitives (and their boxed types)

Download today’s slides:
go/java+espresso-training/day3

“Hello World!”
2 number Adder challenge

Using what you’ve learned so far, turn HelloWorld into a two parameter integer adder so that:

```
> java HelloWorld 24 3
```

Will print:

```
The answer is: 27
```

The parts you will need:

1. The `args` parameter to `main()` and access to indexes of that array using the syntax : `args[0]`
2. The `Integer` box type and the class method `valueOf()`
3. The `println()` class method

Now change the program so the name of the class and application is `Adder` rather than `HelloWorld`

Control Flow: `if` and `switch`

Like C and C++, Java offers `if` and `switch` for conditional branching

Assume an `int` called `count` and an instance object `obj` that has: `lessThanTwo()`, `equalToFour()` and `everythingElse()` in it's public API

```
if (count < 2) {
    obj.lessThanTwo();
} else if (count == 4) {
    obj.equalToFour();
} else {
    obj.everythingElse();
}
```

```
switch (count) {
    case 0:
    case 1:
        obj.lessThanTwo();
        break;
    case 4:
        obj.equalToFour();
        break;
    default:
        obj.everythingElse();
        break;
}
```

There's also a ternary operator (concise if-else):

```
int val = (a > b) ? 5 : 7;
```

The boolean Type and Relation Operators

The `if` statement operates on the primitive type `boolean` which can only take the values: `true` or `false`

```
if (true) {
     obj.always();
} else {
     obj.never();
}
```

```
if (false) {
     obj.never();
} else {
     obj.always();
}
```

Relational expressions such as: `(count < 2)` produce a `boolean` value.

The numeric relational operators

==	equal to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

Back to the ‘Adder’ application

```
public class Adder {
    public static void main(String[] args){
        int x = Integer.valueOf(args[0]);
        int y = Integer.valueOf(args[1]);
        int sum = x + y;
        System.out.println("The answer is: " + sum)
    }
}
```

Your Adder solution from Wednesday may look something like this

It works, but (as you may have noticed when giving too few arguments) the application is ***brittle***

In general terms, how might a conditional expression be used to reduce the brittleness of the Adder app?

The first fix for ‘Adder’

```
...
public static void main(String[] args){
…
```

The `args` variable passed into `main()` is an instance object of the class `Array` which has some useful methods, one of which is: `length`

You can send a `length` message to an instance of `Array` and it will return an int which is the number of value in the `Array` instance:

```
int n = args.length
```

Which can be used in a conditional: `if (args.length < ???) { ...`

Fix #1: Use the `if` conditional and `args.length` to display an error message if too few command-line arguments are passed into the application

Adder fix #1, One Solution

```
public class Adder {
    public static void main(String[] args){
        if (args.length > 1) {
            int x = Integer.valueOf(args[0]);
            int y = Integer.valueOf(args[1]);
            int sum = x + y;
            System.out.println("The answer is: " + sum);
        } else {
            System.out.println("Error: too few arguments");
        }
    }
}
```

This is one possible solution. Did anyone do the conditional differently?

Logical Boolean Operators

It's often useful to combine boolean expressions, for example if you want to test if an `int` variable `count` is both greater than `5` AND less than `10`

You could nest if statements:

```
if (count > 5){
    if (count < 10){
        obj.doSomething();
    }
}
```

A more concise solution is to use the logical AND operator: &&

```
if ((count > 5) && (count < 10)){
    obj.doSomething();
}
```

(operator precedence makes parens optional)

A is `true` and **B** is `false`

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

‘Adder’ fix #2

The Adder application still has an issue with arguments. If a user expects to be able to add *more* than two numbers, the Adder will exhibit unexpected behavior

```
> java Adder 1 5 4
```

Will output:

```
6
```

instead of the expected value:

```
10
```

Fix #2: Modify the checking of `args.length` so that the user is sees appropriate errors for both too few and too many command-line arguments

Looping in Java (doing things more than once)

The two most common loops are `while` and `for`
Both operate using a `boolean` condition (just like `if`)

Assuming an `int` called `count`

```
while (count < 10) {
    obj.doStuff();
}
```

This will continue to loop until `count` is equal-to or greater-than `10`

If `doStuff()` never alters the value of `count`, this will loop forever

doStuff() need to increment count:
`count = count + 1` or `count++`

or just change the value to more than 9

For loops include both initialization and incrementing within the loop syntax. Assume `i` is an `int`

```
for (i=0; i<6; i++){
    obj.doStuff();
}
```

This will execute `doStuff()` 6 times and doesn't depend on `doStuff()` changing the loop condition to prevent infinite looping.

‘Repeater’ Problem

Create a new application ‘Repeater’ that takes a single numeric argument *N* and repeats a text message *N* times

For example

```
> java Repeater 3
```

Will output:

```
Repeating
Repeating
Repeating
```

Repeater, One Possible Solution

```
public class Repeater {
      public static void main(String[] args){
              int n = Integer.valueOf(args[0]);
              int i;

              for (i=0; i< n; i++){
                      System.out.println("Repeater");
              }
      }
}
```

This is one possible solution. Did anyone do the conditional differently?

There is solution that does without the second `int` variable `i`