1 Boolean expressions

- Evaluate to either true or false
- Can be composed of several boolean expressions connected with operators
- Operators themselves return booleans
- Basic boolean literals: true or false
- Equals: ==
- Not-equals: !=
- Conditional AND: &&
- Conditional OR: ||
- Not: !

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(true &amp;&amp; false)</td>
<td>false</td>
</tr>
<tr>
<td>(true</td>
<td></td>
</tr>
<tr>
<td>(true</td>
<td></td>
</tr>
<tr>
<td>(3 == 5) != (3 != 5)</td>
<td>true</td>
</tr>
</tbody>
</table>

Person jack = new Person("Jack", 21, "Area 51");
Person jack2 = new Person("Jack", 21, "Area 51");

jack == jack2 false
jack.age == jack2.age true

2 Short-circuit evaluation

- Stop evaluation of expression if result is known - only applies to Conditional-AND and Conditional-OR

AND Example: (5 == (3 + 9)) && true
OR Example: (5 == (2 + 3)) || false

- In both examples, result is known before evaluation of second operand
- For Conditional-AND, if the first operand is false, then the second is not evaluated
- For Conditional-OR, if the first operand is true, then the second is not evaluated
• Usually the result is not different with short-circuit evaluation, but sometimes it is (without it, this example would cause an ArithmeticException to be thrown due to the divide by zero):

\[((1+3) == 4) || ((2 / 0) > 17)\]

3 Selection statements

• The if statement:

```java
if (condition) {
    code
}
```

• Executes `code` only if `condition` evaluates to true
• If `code` is only one statement, `{ }` are not required
• Another form of if is if-else:

```java
if (condition) {
    codeTrue
} else {
    codeFalse
}
```

• Execute `codeTrue` only if `condition` evaluates to true, otherwise executes `codeFalse`

Example

```java
Person steve = new Person("Steve", 17, "SF");

if (steve.age >= 18) {
    System.out.println("You can vote!");
} else if (steve.age > 15) {
    System.out.println("At least you can drive");
} else {
    System.out.println("Wait a few years, punk");
}

// output: "At least you can drive"
```

• The switch statement

```java
switch (var) {
    case value1:
        code1
    case value2:
        code2
    ...
    default:
        defaultcode
}
```

Example
char letter = 'b';
switch (letter) {
    case 'a':
    case 'A':
        System.out.print("Got an A");
        break;
    case 'b':
    case 'B':
        System.out.print("Got a B");
        break;
    default:
        System.out.print("Got something else");
}
// output: "Got a B"

4 Announcements

• Readings: Schaum’s Ch 4-6, 8.1-8.6; Programming Into Java Ch 1-2
• Register your account
• Get cardkey access at the Soda front desk
• Use the newsgroup!
• Attend office hours: M-Th 1-3:30, 329 Soda

5 Iterative Control

• The while statement:

while (condition) {
    code
}

• Executes code repeatedly as long as condition is true
• May never execute code at all – when could that happen?

• The do while statement:

do {
    code
} while (condition);

• Behaves the same as while except condition is evaluated after code is executed
• Always will execute code at least once

• The for statement:
for(init; condition; update) {
    code
}

is functionally equivalent to:

init
while (condition) {
    code
    update
}

For (minor) the differences, I refer you to Programming Into Java 6.4.2

Example

/* print the numbers 1 - 10 */
/* with a for loop */
for (int i = 1; i <= 10; i++) {
    System.out.println(i);
}

/* with a while loop */
int i = 1;
while (i <= 10) {
    System.out.println(i);
    i += 1; // equivalent to i++
}

/* with a do while loop */
int i = 1;
do {
    System.out.println(i);
    i = i + 1; // equivalent to i++ or i += 1
} while (i <= 10);

6 Peer-instruction question

What does a call to compute() print?

static void compute() {
    for (int i = 0; (i < 10) || !valid(i-10); i++) {
        System.out.println("moose");
    }
}
static boolean valid(int z) {
    if (z < 3) {
        System.out.println("moose");
        return false;
    } else {
        return true;
    }
}
White: It will print moose 16 times and terminate
Blue: It will print moose forever and not terminate
Green: It will print moose 13 times and terminate
Pink: It will print moose 23 times and terminate
Yellow: It will print moose 10 times and terminate

White is correct, because it will iterate through the loop in compute 10 times without ever calling valid (due to short-circuit evaluation), and then will iterate 3 more times due to valid returning false, but for those 3 iterations, it will print from both methods, adding up to 16.

7 Arrays

• An array is a set of elements of one type
• T [] is an array of elements of type T
• Arrays are reference types and must be instantiated using new
• All arrays have an immutable length field that contains the number of elements in the array
• Elements can be directly accessed using [x] where x is the index to be accessed
• Arrays are indexed from 0
• Arrays cannot be resized after instantiation
• The exception ArrayIndexOutOfBoundsException is thrown if an invalid index is accessed

Example

/* create an array of size 5 of int */
int [] elements = new int[5];

/* all indices are automatically initialized to 0 because it is of type int*/

elements[0] = 12; /* store 10 in the first index */
elements[4] = 57; /* store 57 in the last index */

/* create an array of size 3 of Person */
Person [] people = new Person[3];

/* all indices are automatically initialized to null because it is a reference type */

people[0] = new Person(); /* store a new Person reference in people[0] */

Arrays are easy to iterate over:

/* make an array of size max of factorials */
static int [] makeFactorials(int max) {
    int [] factorial = new int[max];
    factorial[0] = 1;

    /* store the factorial in each index-1 */
for (int i = 1; i < factorial.length; i++) {
    factorial[i] = (i+1) * factorial[i-1];
}

return factorial;
}

8 Things to look up

- The ? : operator – return (steve.age >= 18) ? "Vote" : "Don’t Vote";
- Multidimensional arrays – int [][] box = new int[3][3];
- Comparison operators – < <= == > >= !=
- Bitwise operators – << >> >>> & ^ |
- Array initializers – int [] zipcodes = {94704, 94705};