Boolean Expressions and if-else Statements

David Greenstein
Monta Vista High School

Boolean Values

- Most programming languages express a boolean value as either **0/1** or **true/false**.

- Java uses the reserved words **true** and **false**.
  - C uses the integer values 0 and 1.

- Java has a **boolean** primitive type to store these values.
  - C used to use **int**, but in the 1990's added **bool**.

```
boolean done = false;
```

Boolean Expressions

- Boolean expressions evaluate to either **true** or **false**.

- Examples:

Does the die have six sides?	`numSides == 6`
Is answer1 or answer2 true?	`answer1 || answer2`
Is it not goodInput?	`! goodInput`
Are these two objects equal?	`object1.equals(object2)`

Boolean Expressions

- Boolean expressions are written with

  - **boolean variables**
  - **relational and logic operators**
  - **methods that return a boolean value**

```
numSides == 6
answer1 || answer2
! goodInput
object1.equals(object2)
```

Boolean Variables

- Java uses the **boolean** primitive to declare boolean variables or return types.

```
boolean a;
boolean [] b;

public boolean isPositive(int value) {…}
```

Relational Operators

- There are **six** relational operators

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

Relational Operators (cont)

- Relational operators are best used by primitive numbers and characters.

```
count > 6
firstNum <= secondNum
letter == 'Y'
```

- **Avoid** using **==** and **!=** for **double** numbers. There is always a chance of a **rounding error**.

- **Avoid** using **==** and **!=** for **objects**. These compare the address of the object, not the objects' contents!

Relational Operators (cont)

- To compare objects contents, always use their **equals** method.

```
String s1 = new String("Hello");
String s2 = new String("Goodbye");
boolean areEqual = s1.equals(s2);
```

- **Do** use **==** to see if two variables point to the same object. (same address)

```
String s1 = new String("Morning");
String s2 = s1;
…
boolean areEqual = (s1 == s2);
```

Logical Operators

- There are **three** logical operators

```
!       NOT
||      Inclusive OR
&&      AND
```

Truth Table

A	B	!A	A || B	A && B
true	true	false	true	true
true	false	false	true	false
false	true	true	true	false
false	false	true	false	false

De Morgan's Laws

`!`, `||`, and `&&` obey the rules of formal logic.

```
!(p || q) = !p && !q
!(p && q) = !p || !q
```

Example:

```
! (a < 10 || a > 20)
```

is equivalent to

```
a >= 10 && a <= 20
```

Short-Circuit Evaluation

(*cond1* && *cond2*)

if *cond1* is **false**, then *cond2* is not evaluated.

The result is **false** regardless of *cond2*.

(*cond1* || *cond2*)

if *cond1* is **true**, then *cond2* is not evaluated.

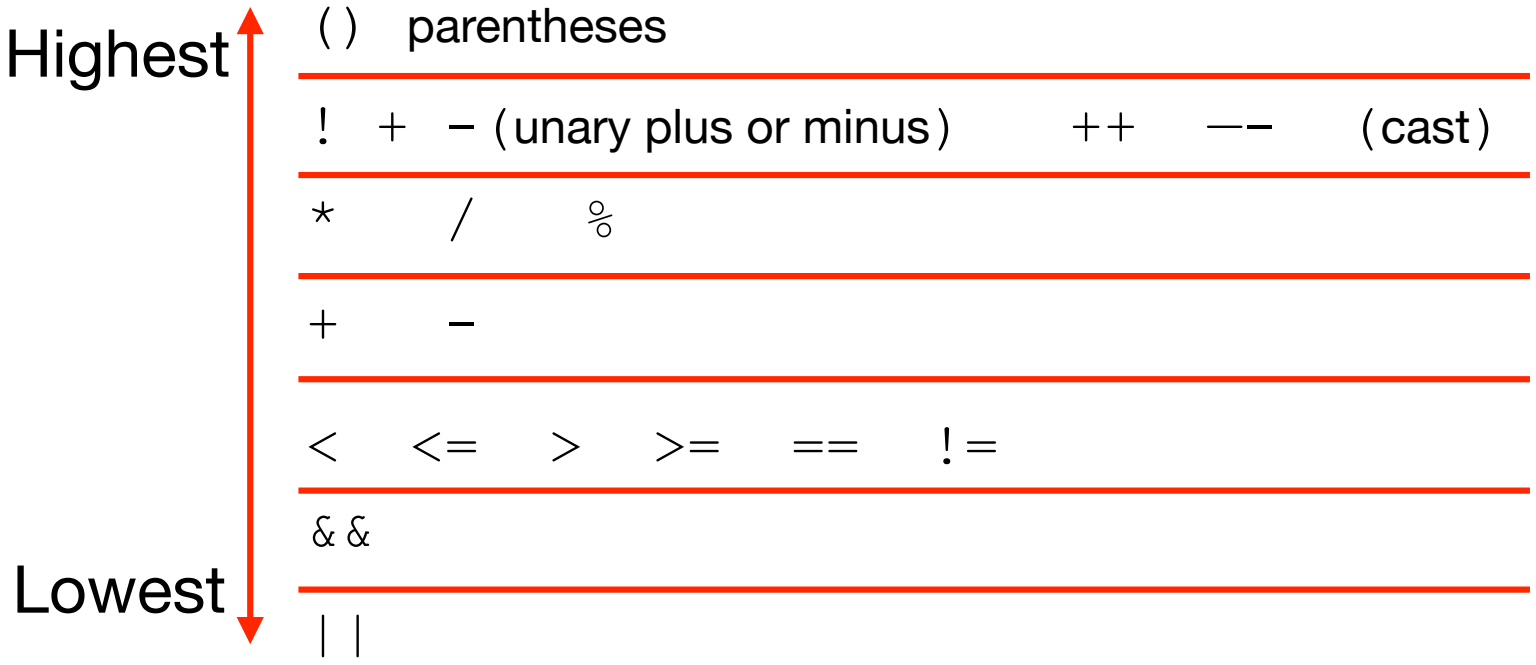
The result is **true** regardless of *cond2*.

Example:

```
( x >= 0 && Math.sqrt(x) > 2.3)
```

No error.
This will only evaluate if x >= 0.

Operator Precedence



Example:

```
if  ( ( ( year % 4 ) == 0 ) && ( month == 2 ) ) ...
if  ( year % 4 == 0 && month == 2 ) ...
```

Easier to read

if-else Statement

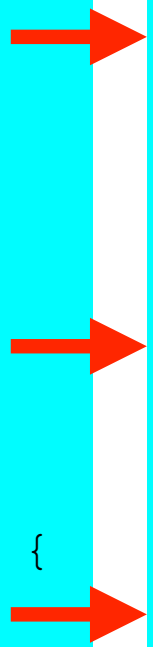
- `if` and `if-else` statements

```
if (points > 10) {
   playHand();
   movePiece();
}
// else do nothing
```

```
if (deposit > 2500) {
   setInterest(0.025);
}
else {
   setInterest(0.014);
}
```

Nested if-else Statements

```
if (initial == 'D') {
    name = "David";
}
else {
    if (initial == 'R') {
        name = "Rafi";
    }
    else {
        if (initial == 'T') {
            name = "Tyrone";
        }
        else {
            if (initial == 'V') {
                name = "Vivian";
            }
            else { // default
                name = "John";
            }
        }
    }
}
```



```
if (initial == 'D') {
    name = "David";
}
else if (initial == 'R') {
    name = "Rafi";
}
else if (initial == 'T') {
    name = "Tyrone";
}
else if (initial == 'V') {
    name = "Vivian";
}
else { // default
    name = "John";
}
```

Common if-else Errors

Extra semicolon

```
if (condition);
{
   statements
}
```

Missing braces

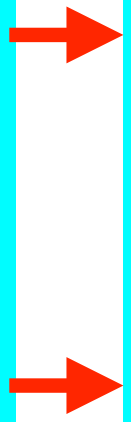
```
if (condition)
   statement1;
   statement2;
```

Ambiguous else

```
if (condition1)
   if (condition2)
      statement1;
else
   statement2;
```

Switch Statement

```
if (initial == 'D') {
    name = "David";
}
else if (initial == 'R') {
    name = "Rafi";
}
else if (initial == 'T') {
    name = "Tyrone";
}
else if (initial == 'V') {
    name = "Vivian";
}
else { // default
    name = "John";
}
```



```
switch (initial) {
    case 'D': name = "David";
              break;
    case 'R': name = "Rafi";
              break;
    case 'T': name = "Tyrone";
              break;
    case 'V': name = "Vivian";
              break;
    default: name = "John";
              break;
}
```

`switch` parameter can take:

`byte`, `short`, `char`, `int`, `String`,

enumerated types,

wrapper classes `Byte`, `Short`, `Character`, `Integer`

Switch Cascading case

```
switch (what) {
   case 1: System.out.println("One for the money");
   case 2: System.out.println("Two for the show");
            break;
   case 3: System.out.println("Three to get ready");
   default: System.out.println("Four to go");
}
```

what = 1

Prints: One for the money
Two for the show

what = 2

Prints: Two for the show

Switch Cascading case

```
switch (what) {
   case 1: System.out.println("One for the money");
   case 2: System.out.println("Two for the show");
            break;
   case 3: System.out.println("Three to get ready");
   default: System.out.println("Four to go");
}
```

When ***default*** is not last:

```
switch (what) {
   default: System.out.println("Four to go");
   case 1: System.out.println("One for the money");
   case 2: System.out.println("Two for the show");
            break;
   case 3: System.out.println("Three to get ready");
}
```

what = 1

Prints:
```
One for the money
Two for the show
```

what = 5

Prints:
```
Four to go
One for the money
Two for the show
```

Enumerated Data Type

```
private enum StudentClass { FRESHMAN, SOPHOMORE,
                            JUNIOR, SENIOR };
private enum Month { JAN, FEB, MAR, APR, MAY, JUN,
                     JUL, AUG, SEP, OCT, NOV, DEC };
```

- Used when an object's attribute or state can be one of a small set of values
- enum values are treated like **constants** with **textual symbols**
- enum variable **do not** represent characters, strings, or numbers
- enum is an Object and has a `toString()` method
- Style: enum values should always be UPPERCASE

Enumerated Data Type (cont)

- Use == or != to compare enum variables

```
private enum StudentClass { FRESHMAN, SOPHOMORE,
                            JUNIOR, SENIOR };
 …
StudentClass currentClass = StudentClass.JUNIOR;
 …
if (currentClass == StudentClass.JUNIOR) …
```

- enum's can be used in a switch statement

```
switch (currentClass) {
   case FRESHMAN: …
   case SOPHOMORE: …
   case JUNIOR: …
   case SENIOR: …
}
```

Questions?