

# Boolean Expressions and if-else Statements

David Greenstein  
Monta Vista High School

# Boolean Values

1 of 24

- 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

2 of 24

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

A graphic where the word 'FALSE' is written in a large, black, serif font. The word 'true' is written in a smaller, orange, lowercase, sans-serif font, overlapping the bottom of the 'FALSE' text.

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

3 of 24

- Boolean expressions are written with

**FALSE**  
**true**

- **boolean variables**

- **relational and logic operators**

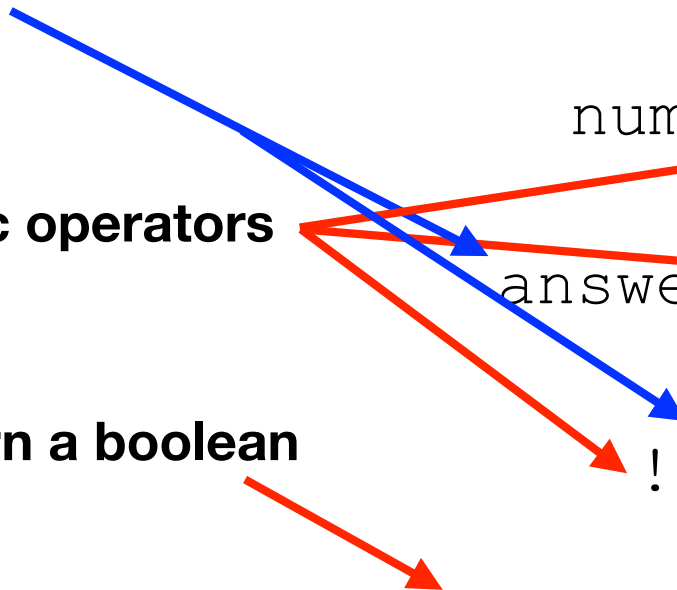
- **methods that return a boolean value**

`numSides == 6`

`answer1 || answer2`

`! goodInput`

`object1.equals(object2)`



# Boolean Variables

4 of 24

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

```
boolean a;  
boolean [] b;  
  
public boolean isPositive(int value) {...}
```

# Relational Operators

5 of 24

- 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)

6 of 24

- 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**.

```
double d1 = Math.sqrt(Math.sqrt(17));
double d2 = Math.pow(Math.pow(d1, 2), 2);
// d2 ≠ 17
```

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

# Relational Operators (cont)

7 of 24

- 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

8 of 24

- There are **three** logical operators

!            NOT  
||           Inclusive OR  
&&          AND

**Truth Table**

<b>A</b>	<b>B</b>	<b>!A</b>	<b>A    B</b>	<b>A &amp;&amp; B</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

9 of 24

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

$$\!(p \ || \ q) = \!p \ \&\& \ \!q$$

$$\!(p \ \&\& \ q) = \!p \ || \ \!q$$

Example:

$$\!(a < 10 \ || \ a > 20)$$

is equivalent to

$$a \geq 10 \ \&\& \ a \leq 20$$

# Short-Circuit Evaluation

10 of 24

*(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*.

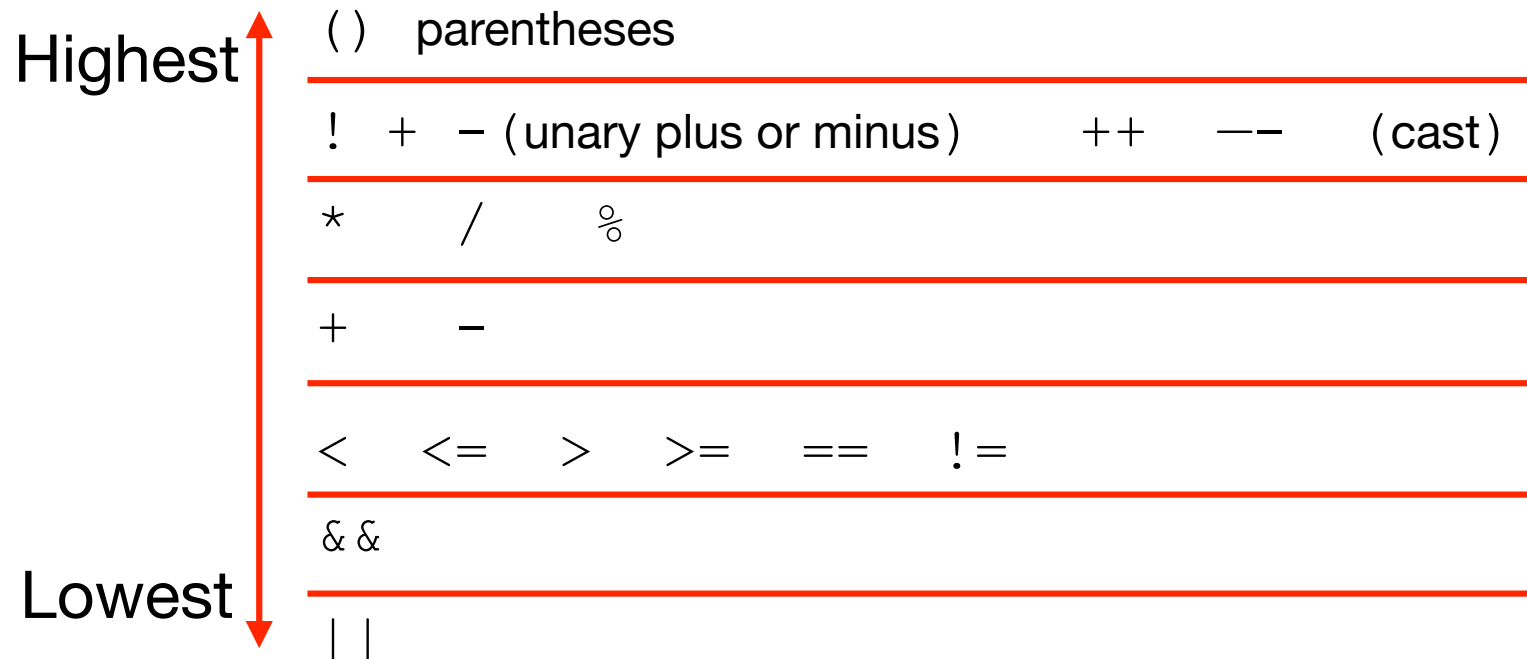
Example:

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

No error.  
This will only evaluate if  $x \geq 0$ .

# Operator Precedence

11 of 24



## Example:

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

Easier to read

# if-else Statement

12 of 24

- `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

13 of 24

```
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

14 of 24

## Extra semicolon

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

## Missing braces

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

## Ambiguous else

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

# Switch Statement

15 of 24

```
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 expressions that result in:

byte, short, char, int, String,

enumerated types,

wrapper classes Byte, Short, Character, Integer

# Switch Fall-through case

16 of 24

```
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 Fall-through case

17 of 24

```
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

# Switch multiple case

18 of 24

```
switch (dayOfWeek) {  
    case MONDAY, TUESDAY, WEDNESDAY  
        THURSDAY, FRIDAY: name = "Weekday";  
        break;  
    case SATURDAY, SUNDAY: name = "Weekend";  
        break;  
    default: name = "unknown";  
}
```

case can list each value  
separated by a comma.

# Switch Lambda Arrow

19 of 24

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

switch using the lambda arrow does not fall through. There is no need for break.

# Switch Lambda Arrow

20 of 24

Each arrow executes a block of code, so you need braces for more lines.

```
String n = "";  
switch (number) {  
    case 1 -> {  
        n = "one";  
        callMethod(n);  
    }  
    case 2 -> {  
        n = "two";  
        callMethod(n);  
    }  
    default -> {  
        n = "five";  
        callMethod(n);  
    }  
}
```

# Switch case vs. arrow

21 of 24

```
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");
}
```

**VS.**

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

# Switch yield statement

22 of 24

```
int value = switch (greeting) {  
    case "hi": System.out.println("Nice");  
                yield 1;  
    case "hello": System.out.println("More formal");  
                yield 2;  
    default: System.out.println("Try again");  
                yield 0;  
}
```

`yield` takes the place of `break` and returns a value.

If the arrow case has a value then it returns that value.

```
int value = switch (greeting) {  
    case "hi" -> 1;  
    case "hello" -> 2;  
    default -> 0;  
}
```

# Enumerated Data Type

23 of 24

```
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)

24 of 24

- 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: ...  
}
```