

# Lecture 4: Conditionals

Building Java Programs: A Back to Basics Approach  
by Stuart Reges and Marty Stepp

Copyright (c) Pearson 2013.  
All rights reserved.



**Type boolean**

# Type boolean

- **boolean**: A logical type whose values are `true` and `false`.
  - It is legal to:
    - create a `boolean` variable
    - pass a `boolean` value as a parameter
    - return a `boolean` value from methods
    - call a method that returns a `boolean` and use it as a test

```
int age = 18;  
String name = "Mr. Smith";  
boolean minor = (age < 21);  
boolean lovesAPCS = true;
```

# Using boolean

- Why is type `boolean` useful?
  - Can capture a complex logical test result and use it later
  - Can write a method that does a complex test and returns it
  - Makes code more readable
  - Can pass around the result of a logical test (as param/return)

```
int age = 21, height = 88;  
double salary = 100000;
```

```
boolean goodAge      = age >= 12 && age < 29; //true  
boolean goodHeight   = height >= 78 && height < 84; //false  
boolean rich          = salary >= 100000.0; //true
```

NOTE: `&&` is the “and” operator. See slide 13.

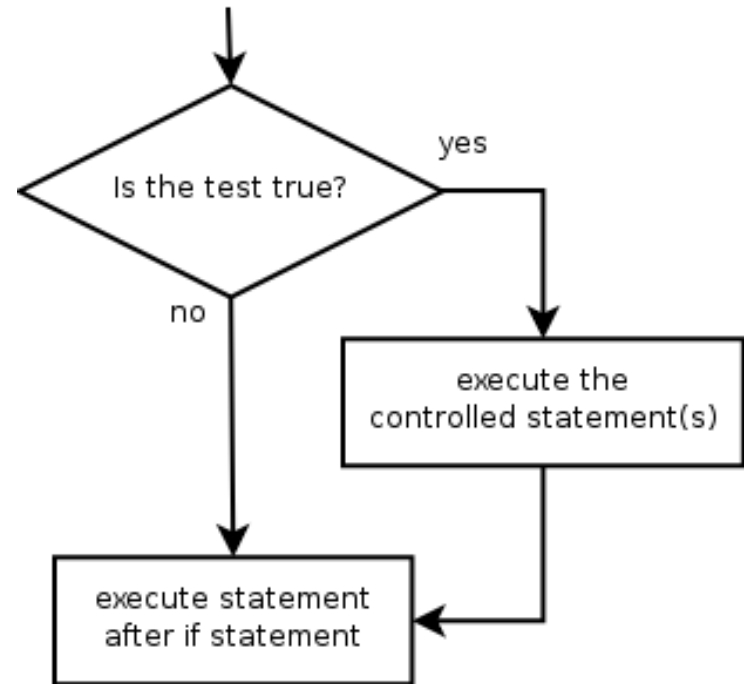
# The `if` statement

*Executes a block of statements only if a test is true*

```
if (test) {  
    statement;  
    ...  
    statement;  
}
```

- Example:

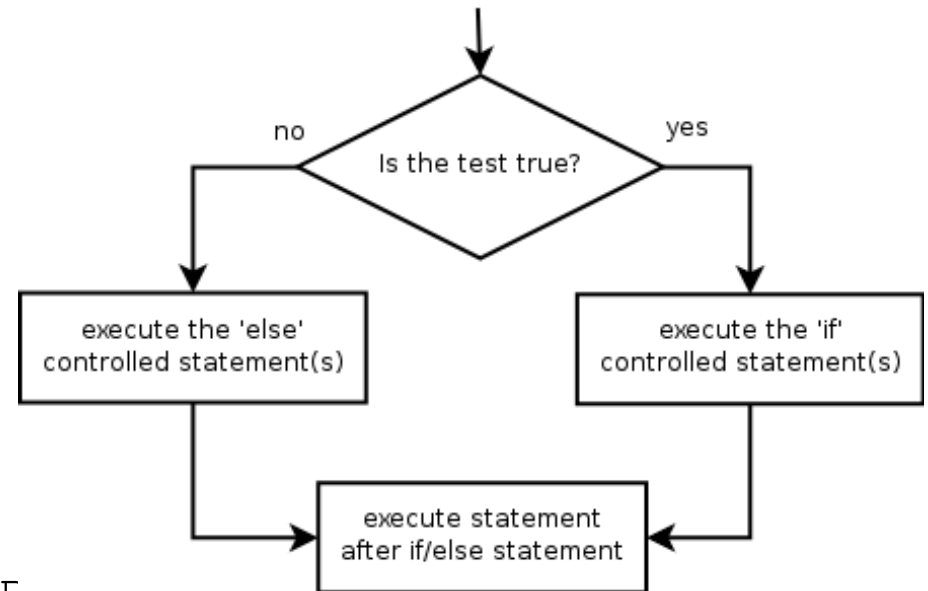
```
double gpa = console.nextDouble();  
if (gpa >= 2.0) {  
    System.out.println("Application accepted.");  
}
```



# The if/else statement

*Executes one block if a test is true, another if false*

```
if (test) {  
    statement(s);  
} else {  
    statement(s);  
}
```



- **Example:**

```
double gpa = console.nextDouble();  
if (gpa >= 2.0) {  
    System.out.println("Welcome to Mars University!");  
} else {  
    System.out.println("Application denied.");  
}
```

# Relational expressions

- Tests use *relational operators*:

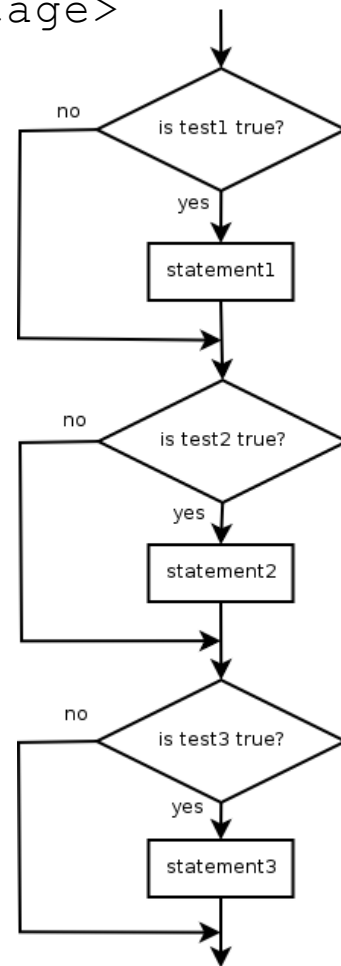
Operator	Meaning	Example	Value
==	equals	1 + 1 == 2	true
!=	does not equal	3.2 != 2.5	true
<	less than	10 < 5	false
>	greater than	10 > 5	true
<=	less than or equal to	126 <= 100	false
>=	greater than or equal to	5.0 >= 5.0	true

# Misuse of if

- What's wrong with the following code?

```
int percent = <Code to ask user to enter a percentage>
```

```
if (percent >= 90) {  
    System.out.println("You got an A!");  
}  
if (percent >= 80) {  
    System.out.println("You got a B!");  
}  
if (percent >= 70) {  
    System.out.println("You got a C!");  
}  
if (percent >= 60) {  
    System.out.println("You got a D!");  
}  
if (percent < 60) {  
    System.out.println("You got an F!");  
}  
...
```





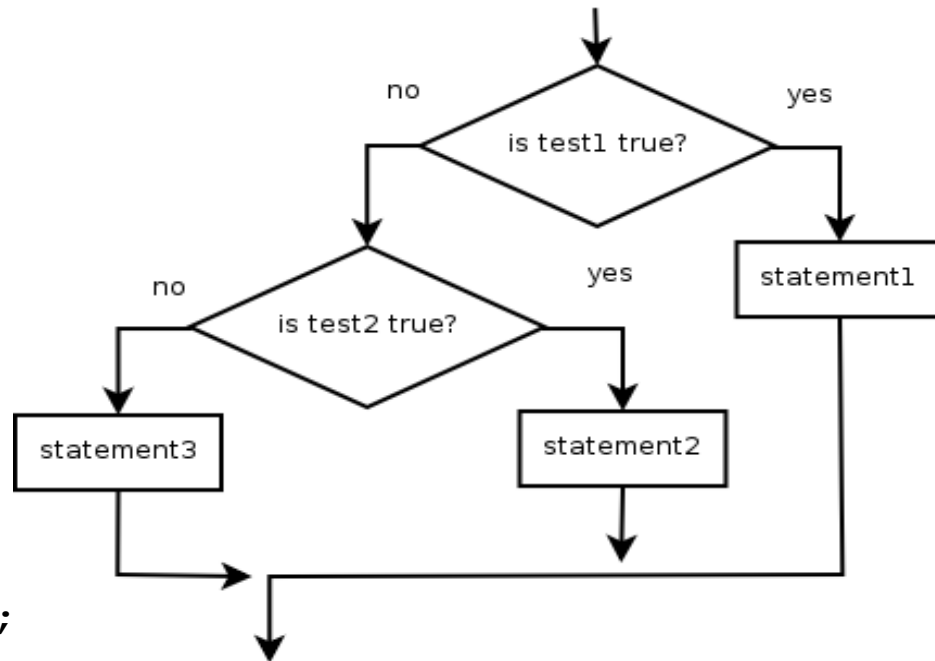
# Nested if/else

*Chooses between outcomes using many tests*

```
if (test) {  
    statement(s);  
} else if (test) {  
    statement(s);  
} else {  
    statement(s);  
}
```

- Example:

```
if (x > 0) {  
    System.out.println("Positive");  
} else if (x < 0) {  
    System.out.println("Negative");  
} else {  
    System.out.println("Zero");  
}
```



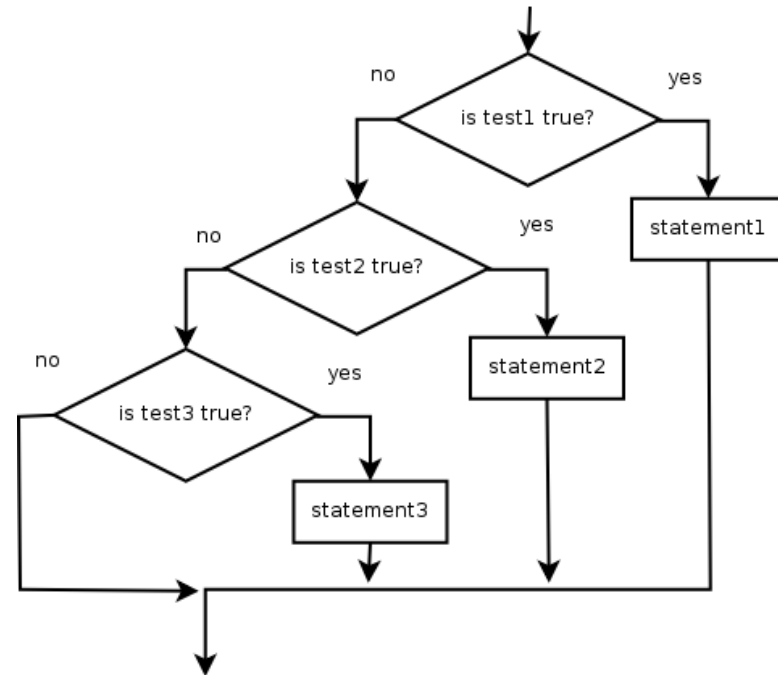
# Nested if/else/if

- If it ends with `else`, exactly one path must be taken.
- If it ends with `if`, the code might not execute any path.

```
if (test) {  
    statement(s);  
} else if (test) {  
    statement(s);  
} else if (test) {  
    statement(s);  
}
```

- Example:

```
if (place == 1) {  
    System.out.println("Gold medal!");  
} else if (place == 2) {  
    System.out.println("Silver medal!");  
} else if (place == 3) {  
    System.out.println("Bronze medal.");  
}
```



# Nested if structures

- exactly 1 path (*mutually exclusive*)

```
if (test) {  
    statement(s);  
} else if (test) {  
    statement(s);  
} else {  
    statement(s);  
}
```

- 0 or 1 path (*mutually exclusive*)

```
if (test) {  
    statement(s);  
} else if (test) {  
    statement(s);  
} else if (test) {  
    statement(s);  
}
```

- 0, 1, or many paths (*independent tests; not exclusive*)

```
if (test) {  
    statement(s);  
}  
if (test) {  
    statement(s);  
}  
if (test) {  
    statement(s);  
}
```

# Which nested if/else?

- **(1) if/if/if    (2) nested if/else    (3) nested if/else/if**
  - Whether a user is lower, middle, or upper-class based on income.
    - **(2)**    nested `if / else if / else`
  - Whether you made the dean's list ( $\text{GPA} \geq 3.8$ ) or honor roll (3.5-3.8).
    - **(3)**    nested `if / else if`
  - Whether a number is divisible by 2, 3, and/or 5.
    - **(1)**    sequential `if / if / if`
  - Computing a grade of A, B, C, D, or F based on a percentage.
    - **(2)**    nested `if / else if / else if / else if / else`

# Logical operators

- Tests can be combined using *logical operators*.

Operator	Description	Example	Result
&&	and	(2 == 3) && (-1 < 5)	false
	or	(2 == 3)    (-1 < 5)	true
!	not	!(2 == 3)	true

- "Truth tables" for each, used with logical values  $p$  and  $q$ .

<b>p</b>	<b>q</b>	<b>p &amp;&amp; q</b>	<b>p    q</b>
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

<b>p</b>	<b>!p</b>
true	false
false	true

# Using boolean

```
boolean goodAge      = age >= 12 && age < 29;  
boolean goodHeight   = height >= 78 && height < 84;  
boolean rich         = salary >= 100000.0;  
  
if ((goodAge && goodHeight) || rich) {  
    System.out.println("Okay, let's go out!");  
} else {  
    System.out.println("It's not you, it's me...");  
}
```

# Using boolean

```
boolean minor      = (age < 21);  
boolean isProf     = name.contains("Prof");  
boolean lovesAPCS  = true;  
  
// allow only APCS-loving students over 21  
if (minor || isProf || !lovesAPCS) {  
    System.out.println("Can't enter the club!");  
}
```

# Evaluating logic expressions

- Relational operators have lower precedence than math.

```
5 * 7 >= 3 + 5 * (7 - 1)
5 * 7 >= 3 + 5 * 6
35      >= 3 + 30
35      >= 33
true
```

- Relational operators cannot be "chained" as in algebra.

```
2 <= x <= 10
true    <= 10          (assume that x is 15)
error!
```

- Instead, combine multiple tests with `&&` or `||`

```
2 <= x && x <= 10
true    && false
false
```



# Evaluating logic expressions

- AND is evaluated before OR.

```
int x = 2;
```

```
int y = 4;
```

```
int z = 5;
```

```
z > 2 || x > 3 && y < 3 ;
```

```
// true if evaluate && before ||
```

```
// false if evaluate || before &&
```

```
// the correct answer is true: &&
```

```
// must be evaluated before ||
```

# Logical questions

- What is the result of each of the following expressions?

```
int x = 42;
```

```
int y = 17;
```

```
int z = 25;
```

```
- y < x && y <= z
```

```
- x % 2 == y % 2 || x % 2 == z % 2
```

```
- x <= y + z && x >= y + z
```

```
- !(x < y && x < z)
```

```
- (x + y) % 2 == 0 || !((z - y) % 2 == 0)
```

- **Answers:** true, false, true, true, false

# if/else with return

// Returns the larger of the two given integers.

```
public static int max(int a, int b) {  
    if (a > b) {  
        return a;  
    } else {  
        return b;  
    }  
}
```

- Methods can return different values using `if/else`
  - Whichever path the code enters, it will return that value.
  - Returning a value causes a method to immediately exit.
  - All paths through the code must reach a `return` statement.

# All paths must return

```
public static int max(int a, int b) {  
    if (a > b) {  
        return a;  
    }  
    // Error: not all paths return a value  
}
```

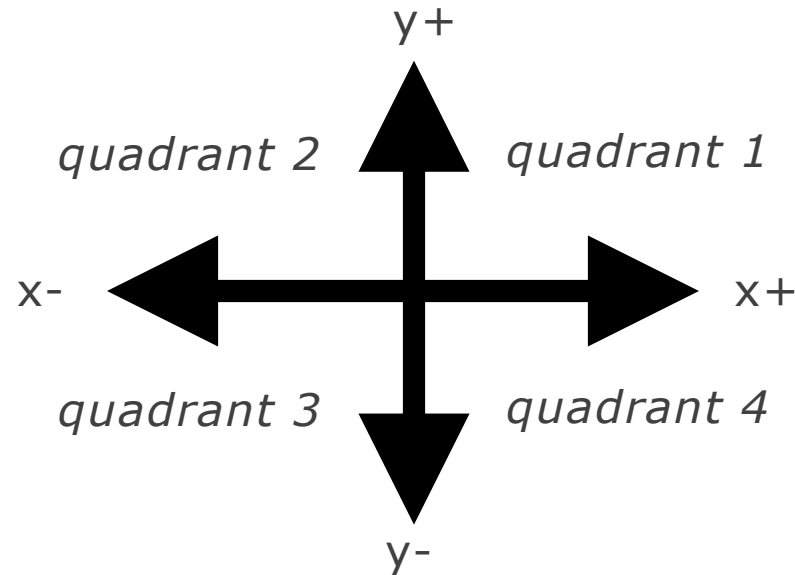
- The following also does not compile:

```
public static int max(int a, int b) {  
    if (a > b) {  
        return a;  
    } else if (b >= a) {  
        return b;  
    }  
}
```

- The compiler thinks `if/else/if` code might skip all paths, even though mathematically it must choose one or the other.

# if/else, return question

- Write a method `quadrant` that accepts a pair of real numbers  $x$  and  $y$  and returns the quadrant for that point:



- Example: `quadrant(-4.2, 17.3)` returns 2
  - If the point falls directly on either axis, return 0.

# if/else, return answer

```
public static int quadrant(double x, double y) {  
    if (x > 0 && y > 0) {  
        return 1;  
    } else if (x < 0 && y > 0) {  
        return 2;  
    } else if (x < 0 && y < 0) {  
        return 3;  
    } else if (x > 0 && y < 0) {  
        return 4;  
    } else {           // at least one coordinate equals 0  
        return 0;  
    }  
}
```

# Lab 1: BMI

**Create a folder called BMI for these labs.**

Formula for body mass index (BMI):

$$BMI = \frac{weight}{height^2} \times 703$$

BMI	Weight class
below 18.5	underweight
[18.5 – 25)	normal
[25.0 – 30)	overweight
30.0 and up	obese

- Write a program that produces output like the following:

```
Height (in inches) 70.0
Weight (in pounds) 194.25
BMI = 27.868928571428572
Overweight
```

# BMI

Your program must include two methods: 1) the method `bmi` which takes two double parameters `height` and `weight` and returns the `bmi` and 2) the method `weightClass` which takes two double parameters `height` and `weight` and returns a string classifying the weight class. **The `weightClass` method must call the `bmi` method!**

```
public static double bmi(double height, double
    weight)
```

```
{...}
```

```
public static String weightClass(double height,
    double weight)
```

```
{...}
```



# Lab 2: Day Of the Week

Write a program that outputs the day of the week for a given date.

**Create a new folder in CS50 called DayOfWeek.**

Given the month,  $m$ , day,  $d$  and year  $y$ , the day of the week (Sunday = 0, Monday = 1, ..., Saturday = 6)  $\mathcal{D}$  is given by:

$$\begin{aligned}y_0 &= y - (14 - m)/12 \\x_0 &= y_0 + y_0/4 - y_0/100 + y_0/400 \\m_0 &= m + 12 \times ((14 - m)/12) - 2 \\\mathcal{D} &= (d + x_0 + 31 \times m_0/12) \bmod 7\end{aligned}$$

---

Your program needs one method:

```
public static String dayOfWeek(int m, int d, int y){  
      
}
```

# Day Of the Week

Write the main method so that the output is similar to the following:

Output:

Date: 9-25-2018

Day of the week: Tuesday

Use conditionals!