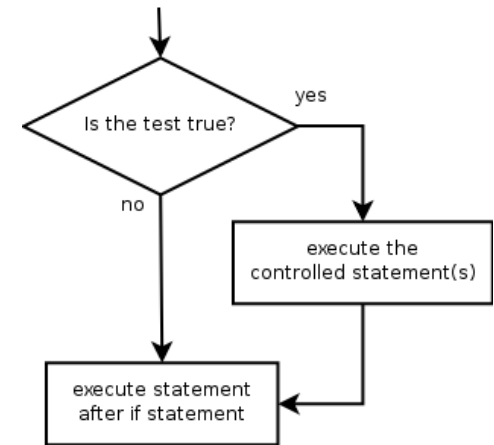# Conditional Execution

The `if` statement

Executes a block of statements only if a test is true

```
if (test) {
    statement;

    ...
    statement;
}
```
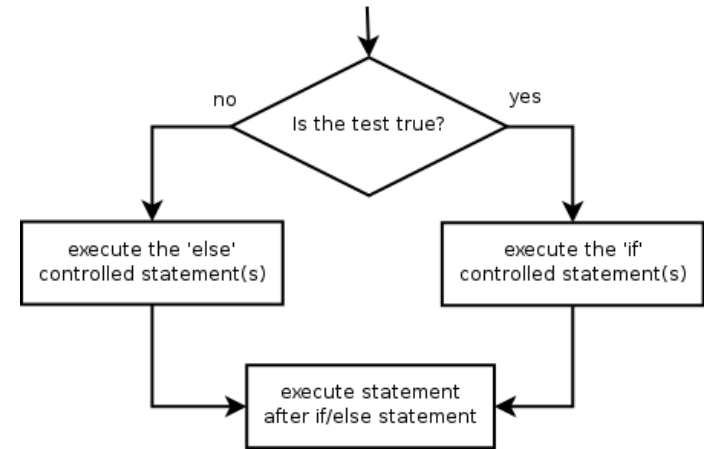

- Example:
```
double gpa = console.nextDouble();
if (gpa >= 2.0) {
    System.out.println("Application accepted.");
}
```

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

- `if` statements and `for` loops both use logical tests.

```
for (int i = 1; i <= 10; i++) { ...
if (i <= 10) { ...
```

   – These are `boolean` 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?

```
Scanner console = new Scanner(System.in);
System.out.print("What percentage did you earn? ");
int percent = console.nextInt();
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!");
}
...
```
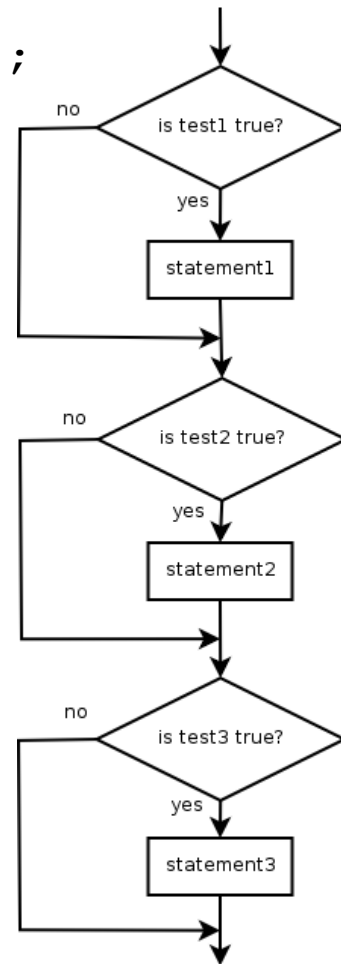
Nested `if/else`

Chooses between outcomes using many tests

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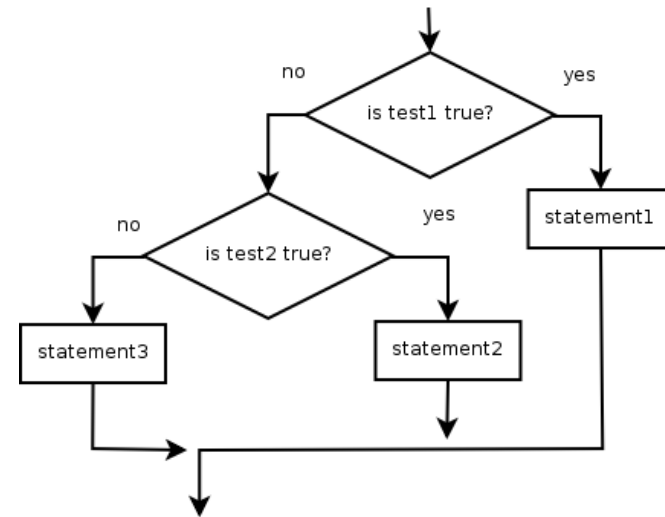

- Example:

```
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.");
}
```
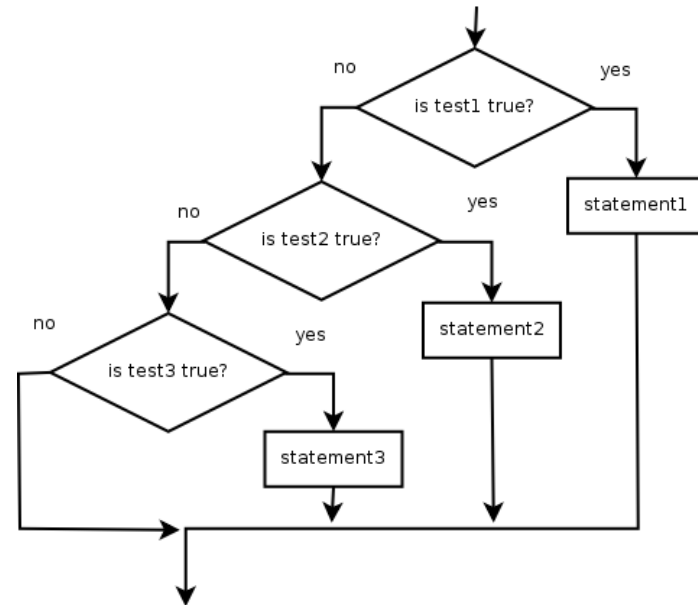
Nested `if` structures

- 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 (GPA ≥ 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`

Nested `if/else` question

Formula for body mass index (BMI):

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

BMI	Weight class
below 18.5	underweight
18.5 - 24.9	normal
25.0 - 29.9	overweight
30.0 and up	obese

- Write a program that produces output like the following:

```
This program reads data for two people and
computes their body mass index (BMI).

Enter next person's information:
height (in inches)? 70.0
weight (in pounds)? 194.25

Enter next person's information:
height (in inches)? 62.5
weight (in pounds)? 130.5

Person 1 BMI = 27.868928571428572
overweight
Person 2 BMI = 23.485824
normal
Difference = 4.3831045714285715
```

Nested `if/else` answer

```java
// This program computes two people's body mass index (BMI) and
// compares them.  The code uses Scanner for input, and parameters/returns.

import java.util.*;  // so that I can use Scanner

public class BMI {
    public static void main(String[] args) {
        introduction();
        Scanner console = new Scanner(System.in);

        double bmi1 = person(console);
        double bmi2 = person(console);

        // report overall results
        report(1, bmi1);
        report(2, bmi2);
        System.out.println("Difference = " + Math.abs(bmi1 - bmi2));

    }

    // prints a welcome message explaining the program
    public static void introduction() {
        System.out.println("This program reads data for two people and");
        System.out.println("computes their body mass index (BMI).");
        System.out.println();
    }
...
```

Nested `if/else,` cont'd.

```java
// reads information for one person, computes their BMI, and returns it
public static double person(Scanner console) {
    System.out.println("Enter next person's information:");
    System.out.print("height (in inches)? ");
    double height = console.nextDouble();

    System.out.print("weight (in pounds)? ");
    double weight = console.nextDouble();
    System.out.println();

    double bodyMass = bmi(height, weight);
    return bodyMass;
}

// Computes/returns a person's BMI based on their height and weight.
public static double bmi(double height, double weight) {
    return (weight * 703 / height / height);
}

// Outputs information about a person's BMI and weight status.
public static void report(int number, double bmi) {
    System.out.println("Person " + number + " BMI = " + bmi);
    if (bmi < 18.5) {
        System.out.println("underweight");
    } else if (bmi < 25) {
        System.out.println("normal");
    } else if (bmi < 30) {
        System.out.println("overweight");
    } else {
        System.out.println("obese");
    }
}
}
```

12

Scanners as parameters

- If many methods need to read input, declare a `Scanner` in `main` and pass it to the other methods as a parameter.

```java
public static void main(String[] args) {
    Scanner console = new Scanner(System.in);
    int sum = readSum3(console);
    System.out.println("The sum is " + sum);
}

// Prompts for 3 numbers and returns their sum.
public static int readSum3(Scanner console) {
    System.out.print("Type 3 numbers: ");
    int num1 = console.nextInt();
    int num2 = console.nextInt();
    int num3 = console.nextInt();
    return num1 + num2 + num3;
}
```

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*:

p	q	p && q	p || q
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

p	!p
true	false
false	true

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
```

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)
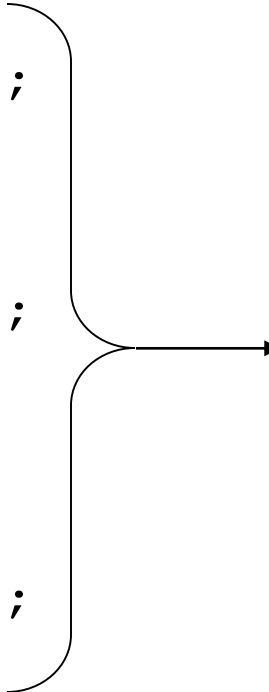
  - Answers: `true, false, true, true, false`

Factoring `if/else` code

- **factoring**: Extracting common/redundant code.
  - Can reduce or eliminate redundancy from `if/else` code.

- Example:

```
if (a == 1) {
    System.out.println(a);
    x = 3;
    b = b + x;
} else if (a == 2) {
    System.out.println(a);
    x = 6;
    y = y + 10;
    b = b + x;
} else {   // a == 3
    System.out.println(a);
    x = 9;
    b = b + x;
}
```

```
System.out.println(a);
x = 3 * a;
if (a == 2) {
    y = y + 10;
}
b = b + x;
```

if/else with return

```java
// 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:


```
                    y+



    quadrant 2           quadrant 1



x-  ◄─────────────────────────►  x+



    quadrant 3           quadrant 4



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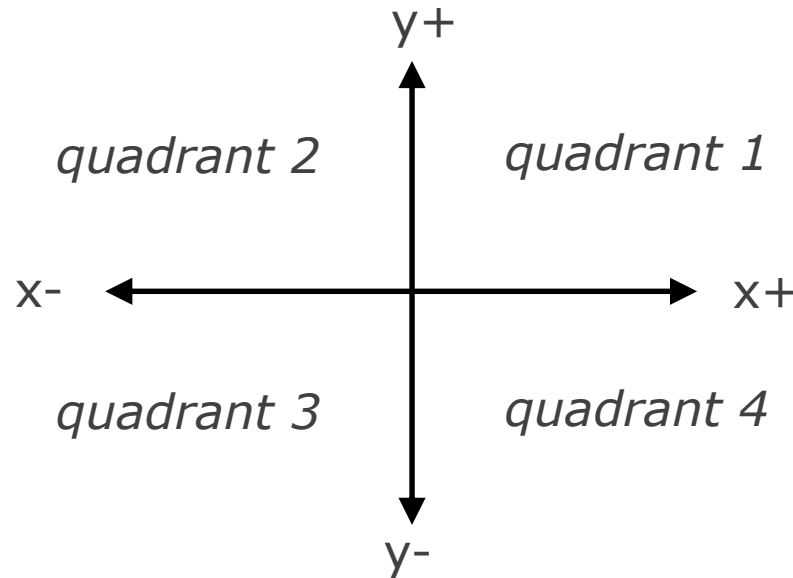
- Example: `quadrant(-4.2, 17.3)` returns `2`
  - If the point falls directly on either axis, return `0`.

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;
    }
}
```

Adding many numbers

- How would you find the sum of all integers from 1-1000?

```java
// This may require a lot of typing
int sum = 1 + 2 + 3 + 4 + ... ;
System.out.println("The sum is " + sum);
```

- What if we want the sum from 1 - 1,000,000?
  Or the sum up to any maximum?
  - How can we generalize the above code?

Cumulative sum loop

```java
int sum = 0;
for (int i = 1; i <= 1000; i++) {
    sum = sum + i;
}
System.out.println("The sum is " + sum);
```

- **cumulative sum**: A variable that keeps a sum in progress and is updated repeatedly until summing is finished.

    – The `sum` in the above code is an attempt at a cumulative sum.

    – Cumulative sum variables must be declared *outside* the loops that update them, so that they will still exist after the loop.

Scanner **and cumul. sum**

- We can do a cumulative sum of user input:

```
Scanner console = new Scanner(System.in);
int sum = 0;
for (int i = 1; i <= 100; i++) {
    System.out.print("Type a number: ");
    sum = sum + console.nextInt();
}
System.out.println("The sum is " + sum);
```

if/else, return **question**

- Write a method `countFactors` that returns the number of factors of an integer.
    - `countFactors(24)` returns `8` because
      1, 2, 3, 4, 6, 8, 12, and 24 are factors of 24.

- Solution:

```java
// Returns how many factors the given number has.
public static int countFactors(int number) {
    int count = 0;
    for (int i = 1; i <= number; i++) {
        if (number % i == 0) {
            count++;   // i is a factor of number
        }
    }
    return count;
}
```

Text Processing

Type `char`

- **`char`** : A primitive type representing single characters.

  - A `String` is stored internally as an array of `char`

```
String s = "Ali G.";
```

index	0	1	2	3	4	5
value	'A'	'l'	'i'	' '	'G'	'.'

  - It is legal to have variables, parameters, returns of type `char`
    - surrounded with apostrophes: `'a'` or `'4'` or `'\n'` or `'\''`

```
char letter = 'P';
System.out.println(letter);              // P
System.out.println(letter + " Diddy");   // P Diddy
```

The `charAt` method

- The `char`s in a `String` can be accessed using the `charAt` method.
  - accepts an `int` index parameter and returns the `char` at that index

```
String food = "cookie";
char firstLetter = food.charAt(0);   // 'c'

System.out.println(firstLetter + " is for " + food);
```

- You can use a `for` loop to print or examine each character.

```
String major = "CSE";
for (int i = 0; i < major.length(); i++) {    // output:
    char c = major.charAt(i);                 // C
    System.out.println(c);                    // S
}                                             // E
```

Comparing `char` values

- You can compare `char`s with `==`, `!=`, and other operators:

```java
String word = console.next();
char last = word.charAt(word.length() - 1);
if (last == 's') {
    System.out.println(word + " is plural.");
}

// prints the alphabet
for (char c = 'a'; c <= 'z'; c++) {
    System.out.print(c);
}
```

char vs. int

- Each `char` is mapped to an integer value internally
  - Called an **ASCII value**

  `'A'` is 65   `'B'` is 66   `' '` is 32
  `'a'` is 97   `'b'` is 98   `'*'` is 42

  - Mixing `char` and `int` causes automatic conversion to `int`.
  `'a' + 10` is 107,   `'A' + 'A'` is 130

  - To convert an `int` into the equivalent `char`, type-cast it.
  `(char) ('a' + 2)` is `'c'`

char vs. String

- "h" is a `String`, but 'h' is a `char` (they are different)

- A `String` is an object; it contains methods.

```
String s = "h";
s = s.toUpperCase();          // "H"
int len = s.length();         //  1
char first = s.charAt(0);     // 'H'
```

- A `char` is primitive; you can't call methods on it.

```
char c = 'h';
c = c.toUpperCase();              // ERROR
s = s.charAt(0).toUpperCase();    // ERROR
```

  - What is `s + 1` ? What is `c + 1` ?
  - What is `s + s` ? What is `c + c` ?

Formatting text with `printf`

```
System.out.printf("format string", parameters);
```

- A format string can contain *placeholders* to insert parameters:
  - `%d`   integer
  - `%f`   real number
  - `%s`   string
    - these placeholders are used instead of + concatenation

  - Example:
    ```
    int x = 3;
    int y = -17;
    System.out.printf("x is %d and y is %d!\n", x, y);
                      // x is 3 and y is -17!
    ```
    - `printf` does not drop to the next line unless you write `\n`

printf width

- %**W**d   integer, **W** characters wide, right-aligned
- %–**W**d   integer, **W** characters wide, *left*-aligned
- %**W**f   real number, **W** characters wide, right-aligned
- ...

```java
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 10; j++) {
        System.out.printf("%4d", (i * j));
    }
    System.out.println();   // to end the line
}
```

Output:
```
   1   2   3   4   5   6   7   8   9  10
   2   4   6   8  10  12  14  16  18  20
   3   6   9  12  15  18  21  24  27  30
```
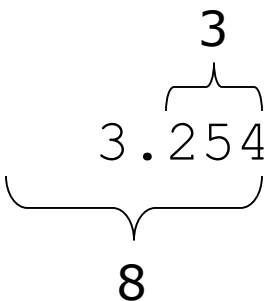
38

printf precision

- %.**D**f   real number, rounded to **D** digits after decimal
- %**W**.**D**f   real number, **W** chars wide, **D** digits after decimal
- %-**W**.**D**f   real number, **W** wide (left-align), **D** after decimal

```
double gpa = 3.253764;
System.out.printf("your GPA is %.1f\n", gpa);
System.out.printf("more precisely: %8.3f\n", gpa);
```

Output:
```
                              3
your GPA is 3.3
more precisely:    3.254
                   8
```

Comparing strings

- Relational operators such as < and == fail on objects.

```java
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name == "Barney") {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

– This code will compile, but it will not print the song.

– `==` compares objects by *references* (seen later), so it often gives `false` even when two `String`s have the same letters.

The `equals` method

- Objects are compared using a method named `equals`.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name.equals("Barney")) {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

- – Technically this is a method that returns a value of type `boolean`, the type used in logical tests.

`String` test methods

Method	Description
`equals(`**str**`)`	whether two strings contain the same characters
`equalsIgnoreCase(`**str**`)`	whether two strings contain the same characters, ignoring upper vs. lower case
`startsWith(`**str**`)`	whether one contains other's characters at start
`endsWith(`**str**`)`	whether one contains other's characters at end
`contains(`**str**`)`	whether the given string is found within this one