

# **Building Java Programs**

## **Chapter 4**

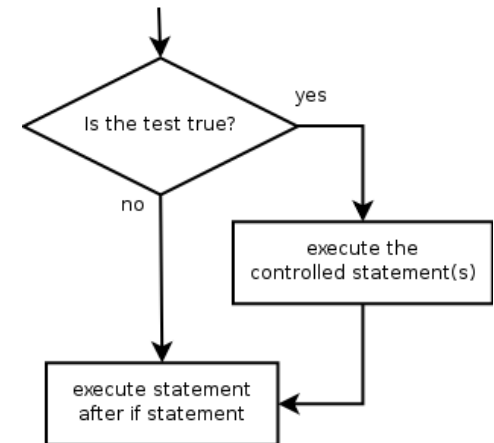
### Conditional Execution

Copyright (c) Pearson 2013.  
All rights reserved.

# The `if` statement

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

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



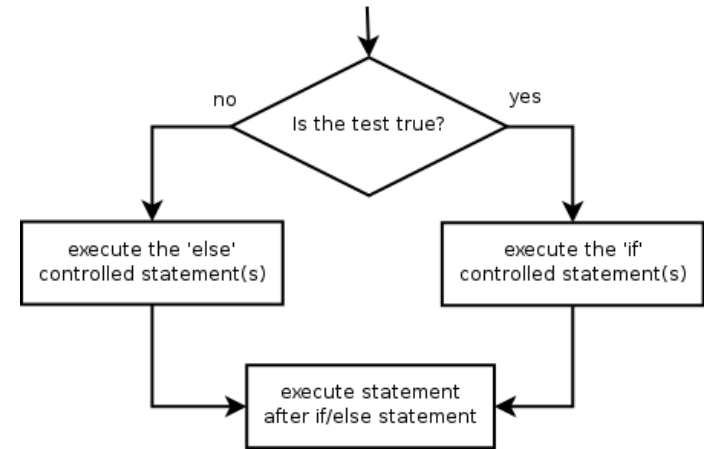
- Example:

```
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, seen in Ch. 5.

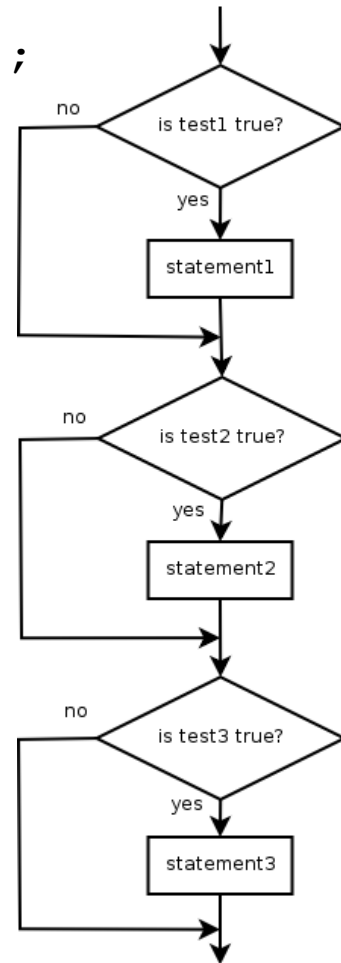
- Tests use *relational operators*:

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

# Misuse of if

- What's wrong with the following code?

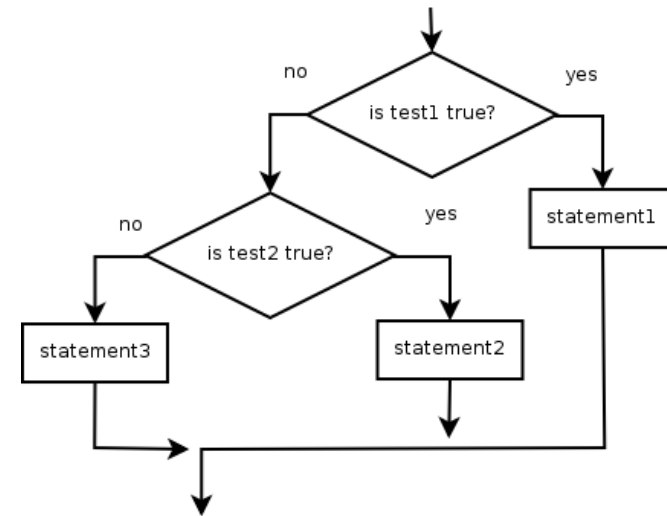
```
Scanner console = new Scanner(System.in);
System.out.print("What percentage did you earn? ");
int percent = console.nextInt();
if (percent >= 90) {
    System.out.println("You got an A!");
}
if (percent >= 80) {
    System.out.println("You got a B!");
}
if (percent >= 70) {
    System.out.println("You 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");  
}
```

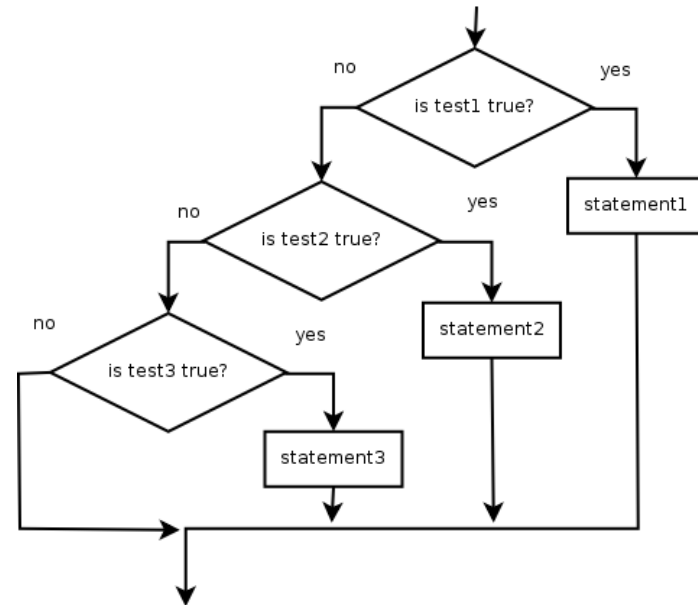
# Nested if/else/if

- 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 ( $\text{GPA} \geq 3.8$ ) 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`

# 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

```
// 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.

```
// 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");
    }
}
}
```

# 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.

```
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  $q$ :

<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

# 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
```

```
error!
```

(assume that x is 15)

- 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`

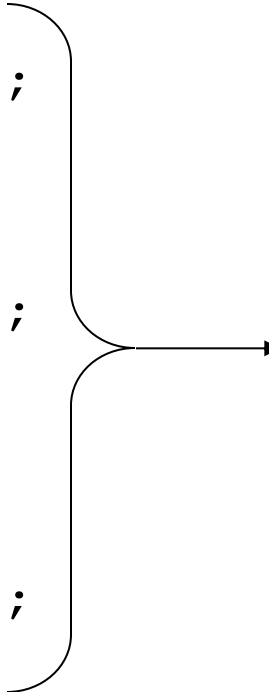
- **Exercise:** Write a program that prompts for information about a person and uses it to decide whether to date them.



# 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

*// 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  
}
```

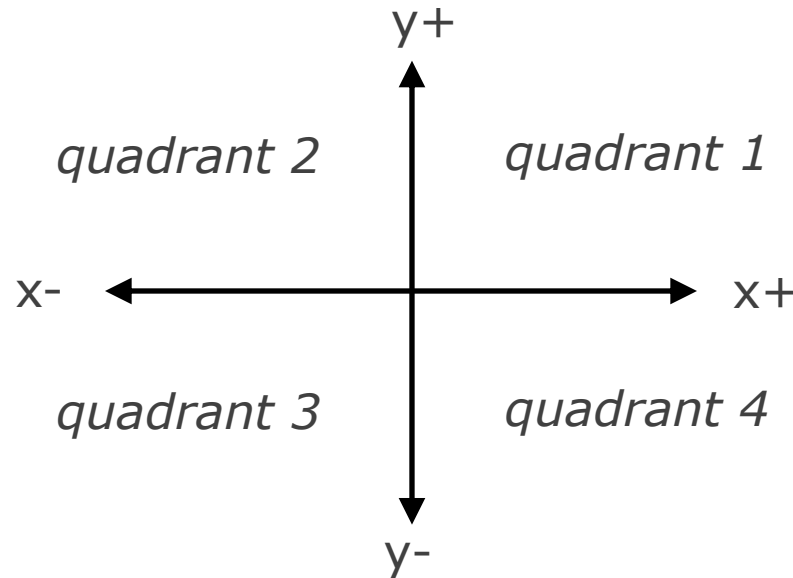
- The following also does not compile:

```
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;  
    }  
}
```

# **Cumulative algorithms**

# Adding many numbers

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

// 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

```
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.



# Cumulative product

- This cumulative idea can be used with other operators:

```
int product = 1;
for (int i = 1; i <= 20; i++) {
    product = product * 2;
}
System.out.println("2 ^ 20 = " + product);
```

- How would we make the base and exponent adjustable?

# 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);
```

# Cumulative sum question

- Modify the `Receipt` program from Ch. 2.
  - Prompt for how many people, and each person's dinner cost.
  - Use static methods to structure the solution.
- Example log of execution:

```
How many people ate? 4  
Person #1: How much did your dinner cost? 20.00  
Person #2: How much did your dinner cost? 15  
Person #3: How much did your dinner cost? 30.0  
Person #4: How much did your dinner cost? 10.00
```

```
Subtotal: $75.0
```

```
Tax: $6.0
```

```
Tip: $11.25
```

```
Total: $92.25
```

# Cumulative sum answer

```
// This program enhances our Receipt program using a cumulative sum.  
import java.util.*;
```

```
public class Receipt2 {  
    public static void main(String[] args) {  
        Scanner console = new Scanner(System.in);  
        double subtotal = meals(console);  
        results(subtotal);  
    }
```

```
// Prompts for number of people and returns total meal subtotal.
```

```
public static double meals(Scanner console) {  
    System.out.print("How many people ate? ");  
    int people = console.nextInt();  
    double subtotal = 0.0;           // cumulative sum  
  
    for (int i = 1; i <= people; i++) {  
        System.out.print("Person #" + i +  
                           ": How much did your dinner cost? ");  
        double personCost = console.nextDouble();  
        subtotal = subtotal + personCost; // add to sum  
    }  
    return subtotal;  
}  
...
```

# Cumulative answer, cont'd.

...

**// Calculates total owed, assuming 8% tax and 15% tip**

```
public static void results(double subtotal) {  
    double tax = subtotal * .08;  
    double tip = subtotal * .15;  
    double total = subtotal + tax + tip;  
  
    System.out.println("Subtotal: $" + subtotal);  
    System.out.println("Tax: $" + tax);  
    System.out.println("Tip: $" + tip);  
    System.out.println("Total: $" + total);  
}  
}
```

# 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:

**// 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.";
```

<i>index</i>	0	1	2	3	4	5
<i>value</i>	'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 `chars` 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++) {  
    char c = major.charAt(i);  
    System.out.println(c);  
}  
// output:  
// C  
// S  
// E
```

# Comparing char values

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

```
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

- `%Wd` integer, **W** characters wide, right-aligned
- `%-Wd` integer, **W** characters wide, *left*-aligned
- `%Wf` real number, **W** characters wide, right-aligned
- ...

```
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

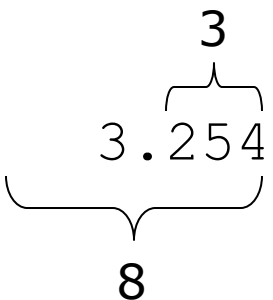
# printf precision

- `%Df`      real number, rounded to **D** digits after decimal
- `%W.Df`    real number, **W** chars wide, **D** digits after decimal
- `%-W.Df`    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:

```
your GPA is 3.3  
more precisely: 3.254
```



# printf question

- Modify our `Receipt` program to better format its output.
  - Display results in the format below, with \$ and 2 digits after .
- Example log of execution:

How many people ate? 4

Person #1: How much did your dinner cost? 20.00

Person #2: How much did your dinner cost? 15

Person #3: How much did your dinner cost? 25.0

Person #4: How much did your dinner cost? 10.00

Subtotal: \$70.00

Tax: \$5.60

Tip: \$10.50

Total: \$86.10



# printf answer (partial)

...

**// Calculates total owed, assuming 8% tax and 15% tip**

```
public static void results(double subtotal) {  
    double tax = subtotal * .08;  
    double tip = subtotal * .15;  
    double total = subtotal + tax + tip;  
  
    // System.out.println("Subtotal: $" + subtotal);  
    // System.out.println("Tax: $" + tax);  
    // System.out.println("Tip: $" + tip);  
    // System.out.println("Total: $" + total);  
  
    System.out.printf("Subtotal: $%.2f\n", subtotal);  
    System.out.printf("Tax: $%.2f\n", tax);  
    System.out.printf("Tip: $%.2f\n", tip);  
    System.out.printf("Total: $%.2f\n", total);  
}  
}
```

# Comparing strings

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

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

```
String name = console.next();  
if (name.startsWith("Prof")) {  
    System.out.println("When are your office hours?");  
} else if (name.equalsIgnoreCase("STUART")) {  
    System.out.println("Let's talk about meta!");  
}
```