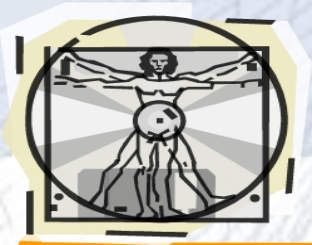
Computer Programming

Elementary Programming

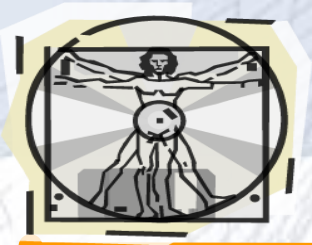
Özgür Koray ŞAHİNGÖZ
Prof.Dr.

Biruni University
Computer Engineering Department



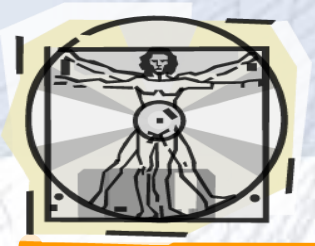
Motivations

In the preceding chapter, you learned how to create, compile, and run a Java program. Starting from this chapter, you will learn how to solve practical problems programmatically. Through these problems, you will learn Java primitive data types and related subjects, such as variables, constants, data types, operators, expressions, and input and output.



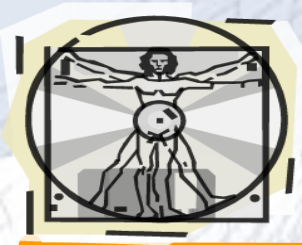
Objectives

- To write Java programs to perform simple computations (§2.2).
- To obtain input from the console using the **Scanner** class (§2.3).
- To use identifiers to name variables, constants, methods, and classes (§2.4).
- To use variables to store data (§§2.5–2.6).
- To program with assignment statements and assignment expressions (§2.6).
- To use constants to store permanent data (§2.7).
- To name classes, methods, variables, and constants by following their naming conventions (§2.8).
- To explore Java numeric primitive data types: **byte**, **short**, **int**, **long**, **float**, and **double** (§2.9.1).
- To read a **byte**, **short**, **int**, **long**, **float**, or **double** value from the keyboard (§2.9.2).
- To perform operations using operators **+**, **-**, *****, /, and **%** (§2.9.3).
- To perform exponent operations using **Math.pow(a, b)** (§2.9.4).



Objectives

- To perform exponent operations using **Math.pow(a, b)** (§2.9.4).
- To write integer literals, floating-point literals, and literals in scientific notation (§2.10).
- To write and evaluate numeric expressions (§2.11).
- To obtain the current system time using **System.currentTimeMillis()** (§2.12).
- To use augmented assignment operators (§2.13).
- To distinguish between postincrement and preincrement and between postdecrement and predecrement (§2.14).
- To cast the value of one type to another type (§2.15).
- To describe the software development process and apply it to develop the loan payment program (§2.16).
- To write a program that converts a large amount of money into smaller units (§2.17).
- To avoid common errors and pitfalls in elementary programming (§2.18).

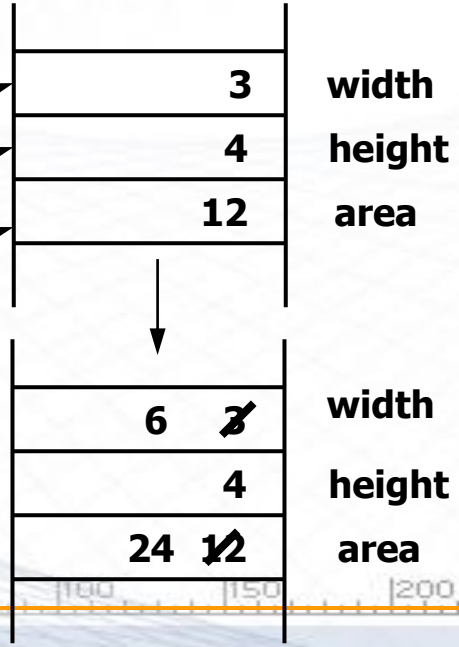


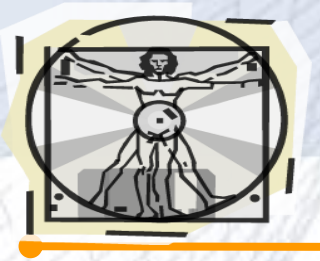
Variables

- **What is a variable?**
  - The name of some **location of memory** used to hold a data value
  - **Different types** of data require **different amounts** of memory. The compiler's job is to reserve sufficient memory
  - Variables need to be declared once
  - Variables are **assigned** values, and these values may be changed later
  - Each variable has a type, and **operations can only be performed between compatible types**

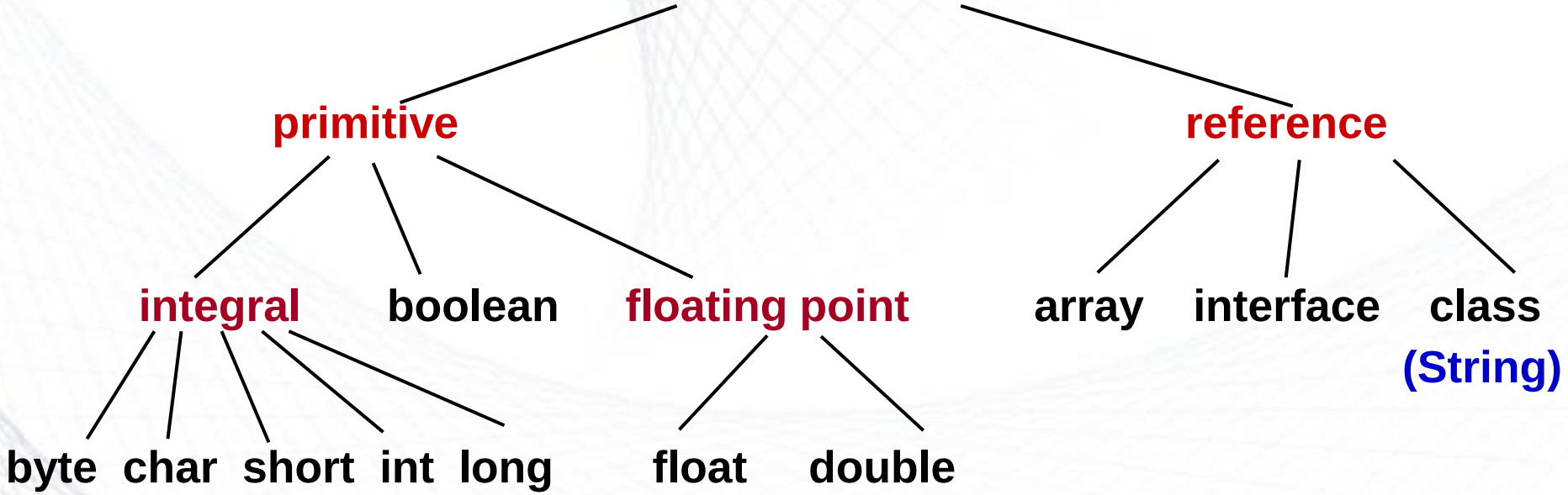
- **Example**

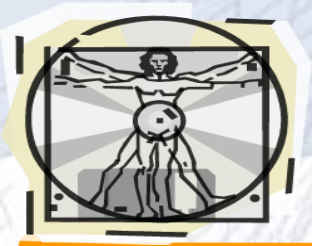
```
int width = 3;
int height = 4;
int area = width * height;

width = 6;
area = width * height;
```





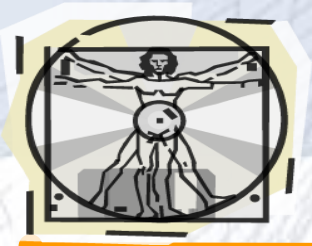
Java Data Types





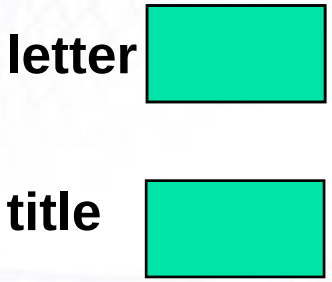
Primitive and Reference Types

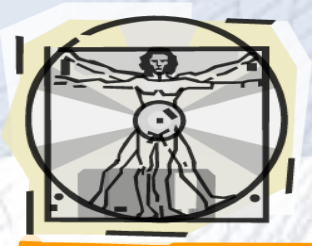
```
char  letter;

String  title;

String  book;

letter = 'J';

title = "Problem Solving";

book = title;
```



Primitive and Reference Types

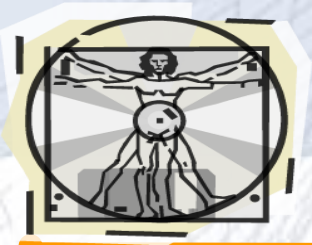
```
char  letter;

String  title;

String  book;

letter = 'J';

title = "Problem Solving";

book = title;
```





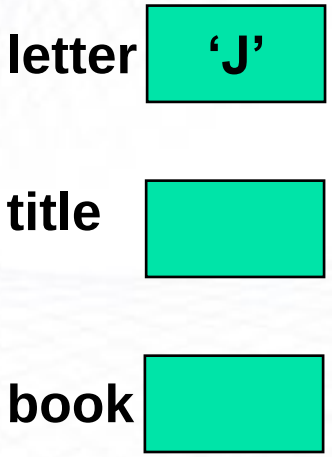
Primitive and Reference Types

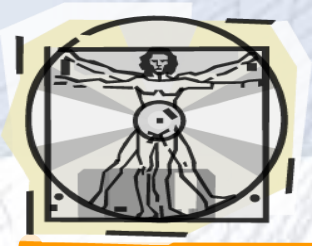
```
char  letter;

String  title;

String  book;

letter = 'J';

title = "Problem Solving";

book = title;
```



Primitive and Reference Types

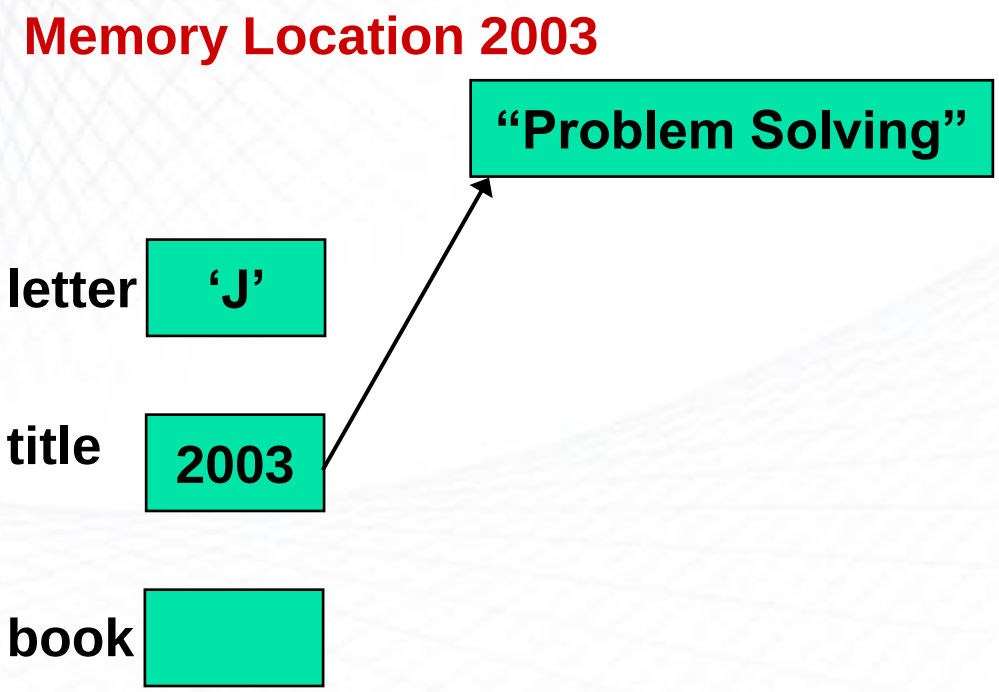
```
char  letter;

String  title;

String  book;

letter = 'J';

title = "Problem Solving";

book = title;
```



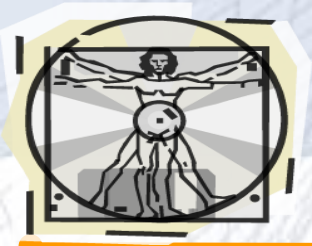


Primitive and Reference Types

```
char  letter;

String  title;

String  book;

letter = 'J';

title = "Problem Solving";

book = title;
```

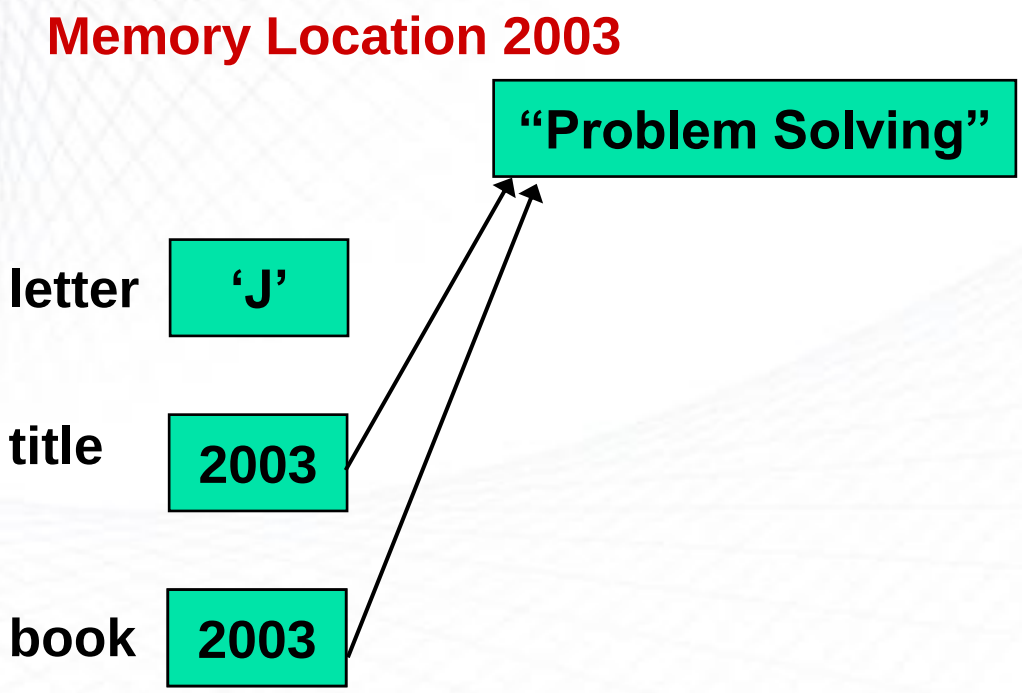


2003 is the address of the "Problem Solving"!!!

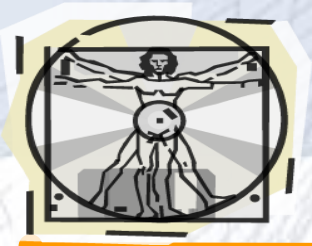


Primitive and Reference Types

```
char  letter;

String  title;

String  book;

letter = 'J';

title = "Problem Solving";

book = title;
```

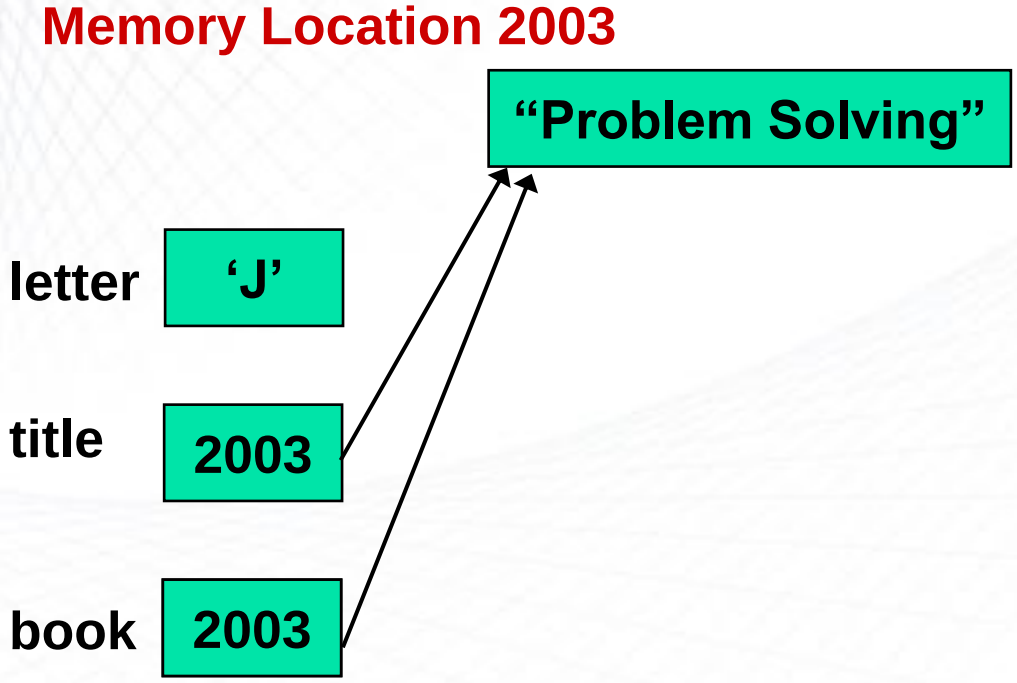


2003 is the address of the "Problem Solving"!!!

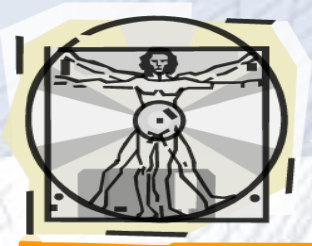


Primitive and Reference Types

```
char  letter;

String  title;

String  book;

letter = 'J';

title = "Problem Solving";

book = title;
```



2003 is the address of the "Problem Solving"!!!



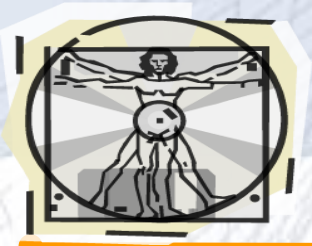
Declaring (Creating) Variables

- To create a variable, you must specify the type and assign it a value:

Syntax

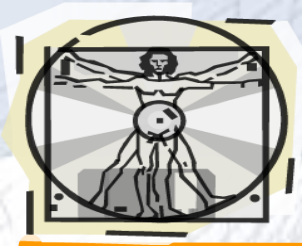
type variable = value;

- Where type is one of Java's types (such as int or String), and variable is the name of the variable (such as x or name). The equal sign is used to assign values to the variable.



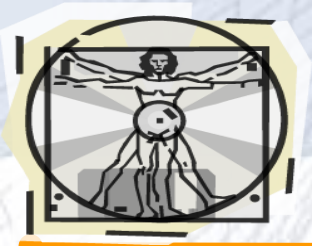
Declaring Variables

```
int x;              // Declare x to be an
                    // integer variable;

double radius;      // Declare radius to
                    // be a double variable;

char a;             // Declare a to be a
                    // character variable;
```



Assignment Statements

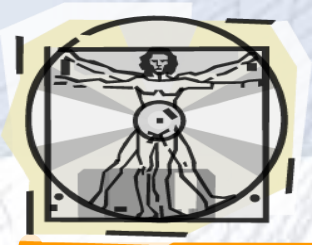
```
x = 1;              // Assign 1 to x;

radius = 1.0;       // Assign 1.0 to radius;

a = 'A';            // Assign 'A' to a;
```



Declaring and Initializing in One Step

- `int x = 1;`

- `double d = 1.4;`



Introducing Programming with an Example

Listing 2.1 Computing the Area of a Circle

This program computes the area of the circle.

ComputeArea

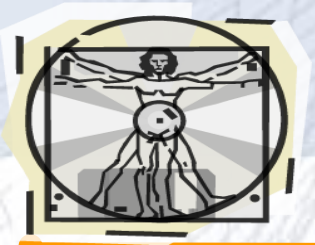
Run

Note: Clicking the green button displays the source code with interactive animation. You can also run the code in a browser. Internet connection is needed for this button.

Note: Clicking the blue button runs the code from Windows. If you cannot run the buttons, see

IMPORTANT NOTE: If you cannot run the buttons, see

.

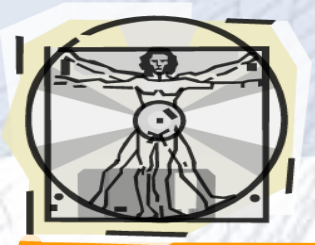


Trace a Program Execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```

allocate memory for radius

radius no value



Trace a Program Execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```

memory

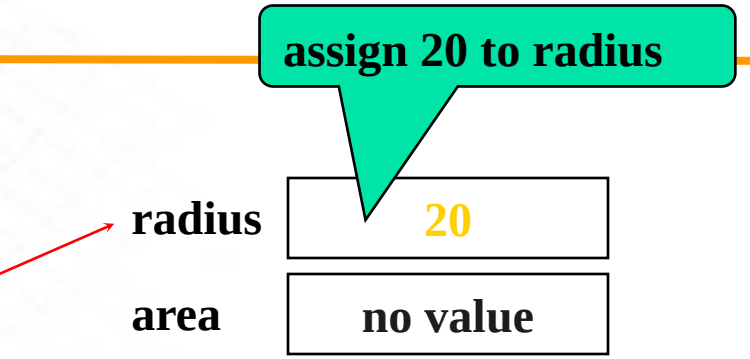
radius **no value**

area **no value**

allocate memory for area

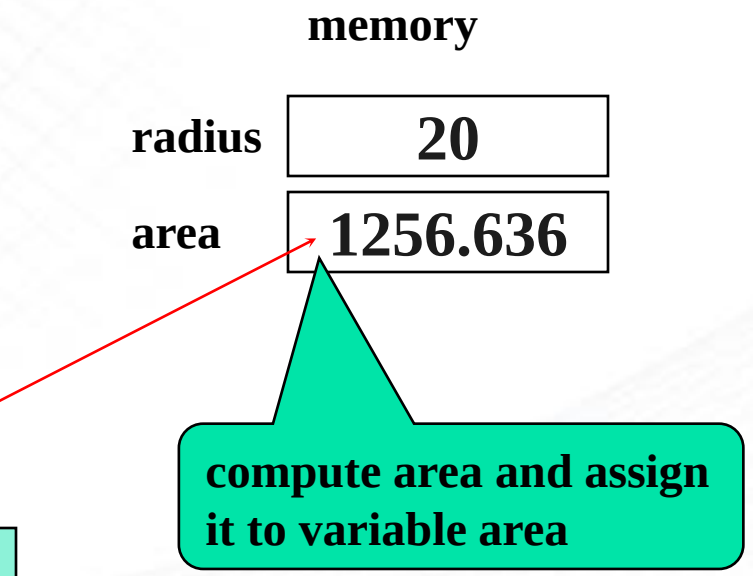
Trace a Program Execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```



Trace a Program Execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```



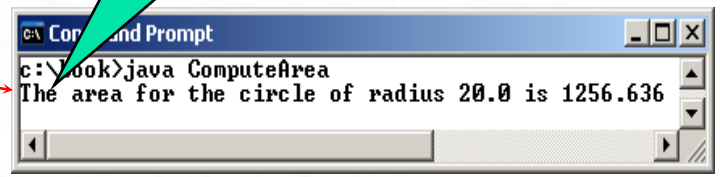
Trace a Program Execution

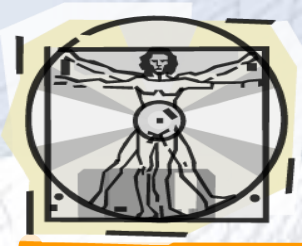
```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```

memory

radius	**20**
area	**1256.636**

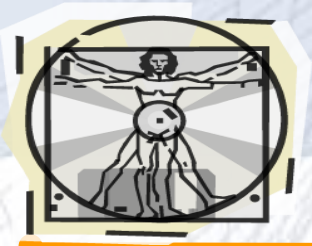
print a message to the console





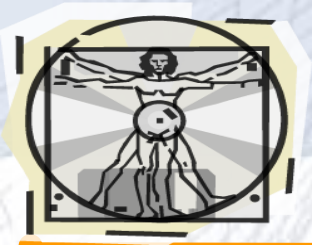
Variables

```
// Compute the first area
radius = 1.0;
area = radius * radius * 3.14159;
System.out.println("The area is " + area + " for radius
  "+radius);

// Compute the second area
radius = 2.0;
area = radius * radius * 3.14159;
System.out.println("The area is " + area + " for radius
  "+radius);
```



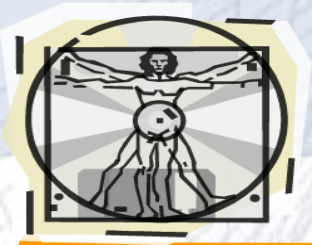
Variable Names - Identifiers

- An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).
- An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
- An identifier cannot be a reserved word. (See Appendix A, “Java Keywords,” for a list of reserved words).
- An identifier cannot be `true`, `false`, or `null`.
- An identifier can be of any length.



Reserved Words

abstract	assert	boolean	break	byte	case
catch	char	class	const*	continue	default
double	do	else	enum	extends	false
final	finally	float	for	goto*	if
implements	import	instanceof	int	interface	long
native	new	null	package	private	protected
public	return	short	static	strictfp	super
switch	synchronized	this	throw	throws	transient
true	try	void	volatile	while	



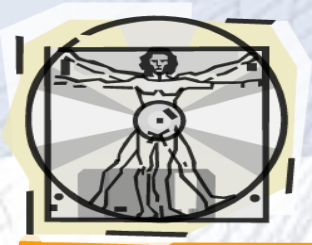
Valid/Invalid Identifiers

Valid:

$$_
R2D2
INT okay. “int” is reserved, but case is different here
_dogma_95_
riteOnThru
SchultzieVonWienerschnitzelIII

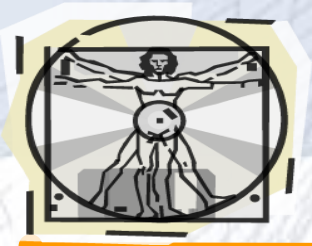
Invalid:

30DayAbs starts with a digit
2 starts with a digit
pork&beans `&’ is illegal
private reserved name
C-3PO `-’ is illegal



Good Variable Names

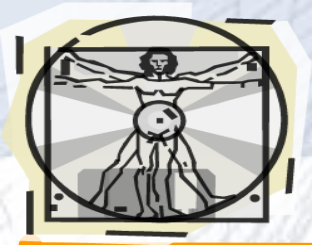
- **Choosing Good Names** →Not all valid variable names are good variable names
- Some guidelines:
  - Do not use `$' (it is reserved for special system names.)
  - Avoid names that are identical other than differences in case (total, Total, and TOTAL).
  - Use meaningful names, but avoid excessive length
    - **crItm** → Too short
    - **theCurrentItemBeingProcessed** → Too long
    - **currentItem** →Just right
- **Camel case** capitalization style
- In Java we use camel case
  - Variables and methods start with lower case
  - **dataList2 myFavoriteMartian showMeTheMoney**
  - Classes start with uppercase
  - **String JOptionPane MyFavoriteClass**



Identifiers

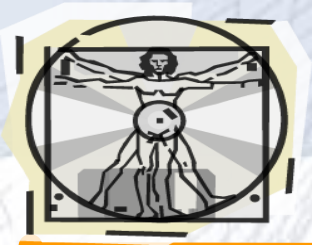
Which of the following can be used in a Java program as identifiers? Check all of the identifiers that are legal

1. sum_of_data
2. AnnualSalary
3. _average
4. 42isThesolution
5. B4
6. ABC
7. for
8. println
9. "hello"
10. first-name



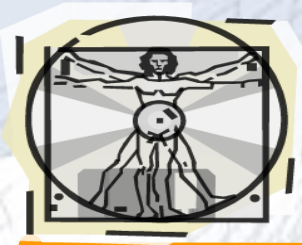
Primitive Data Types

- **Java's basic data types:**
  - **Integer Types:**
    - **byte** 1 byte Range: -128 to +127
    - **short** 2 bytes Range: roughly -32 thousand to +32 thousand
    - **int** 4 bytes Range: roughly -2 billion to +2 billion
    - **long** 8 bytes Range: Huge!
  - **Floating-Point Types** (for real numbers)
    - **float** 4 bytes Roughly 7 digits of precision
    - **double** 8 bytes Roughly 15 digits of precision
  - **Other types**:
    - **boolean** 1 byte {true, false} (Used in logic expressions and conditions)
    - **char** 2 bytes A single (Unicode) character
- String is not a primitive data type (they are objects)

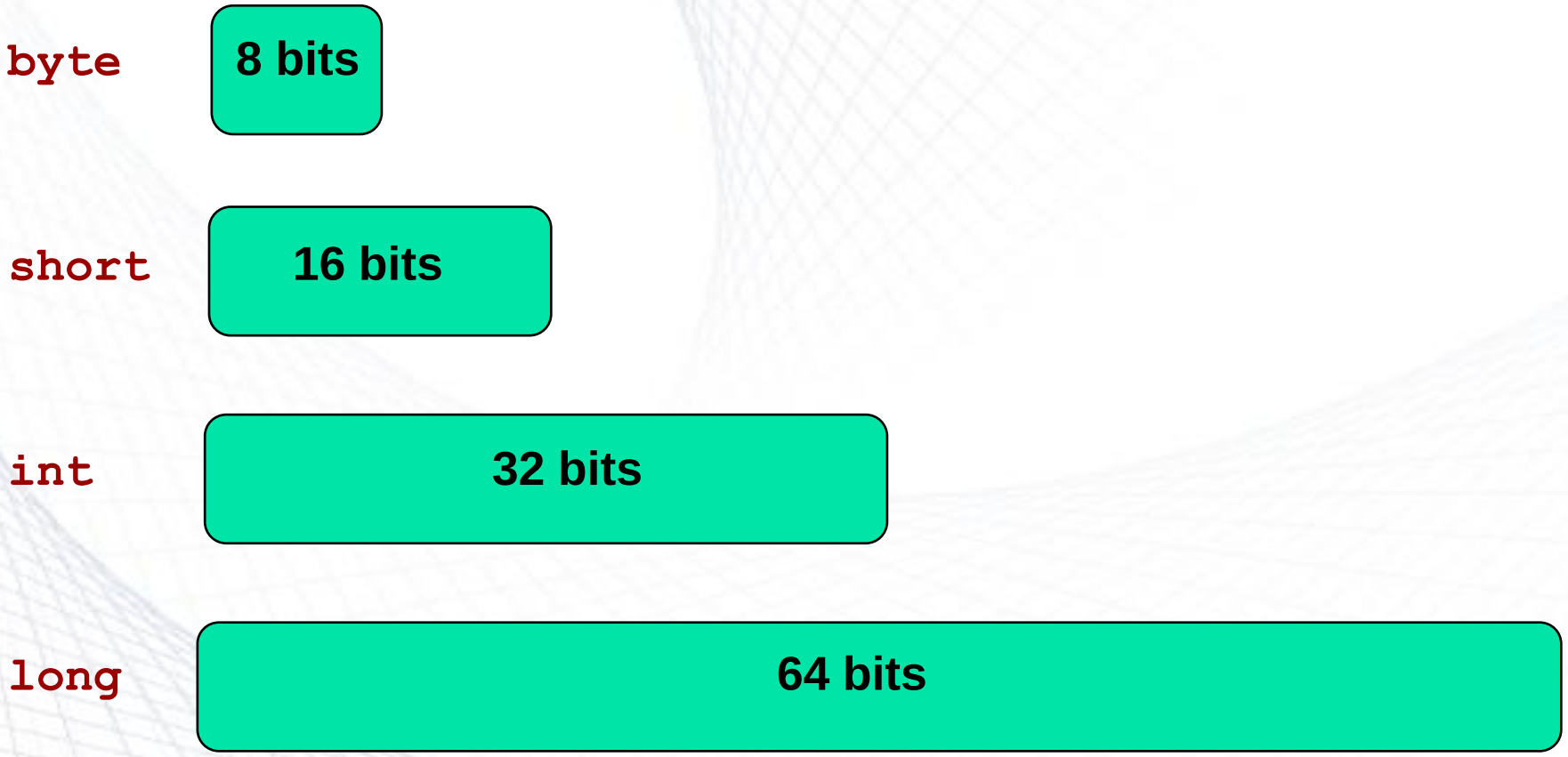


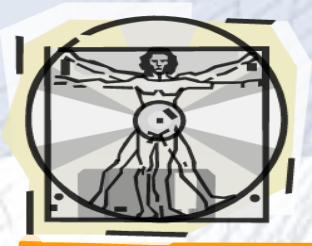
Numerical Data Types

Name	Range	Storage Size
`byte`	-2^7 to $2^7 - 1$ (-128 to 127)	8-bit signed
`short`	-2^{15} to $2^{15} - 1$ (-32768 to 32767)	16-bit signed
`int`	-2^{31} to $2^{31} - 1$ (-2147483648 to 2147483647)	32-bit signed
`long`	-2^{63} to $2^{63} - 1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
`float`	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
`double`	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754



Sizes of Integral Java Types





double vs. float

The double type values are more accurate than the float type values. For example,

```
System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
```

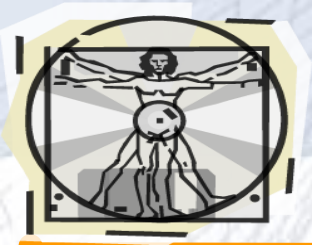
displays `1.0 / 3.0 is 0.3333333333333333`

16 digits

```
System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
```

displays `1.0F / 3.0F is 0.33333334`

7 digits

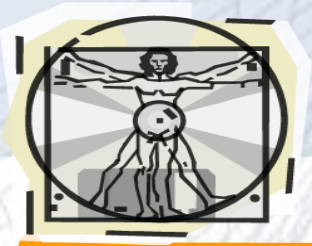


Data Types and Variables

- **Java → Strongly-type language, Strong Type Checking →** Java checks that all expressions involve **compatible** types
  - int x, y; // x and y are integer variables
  - double d; // d is a double variable
  - String s; // s is a string variable
  - boolean b; // b is a boolean variable
  - char c; // c is a character variable

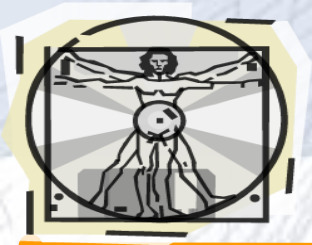
  - x = 7; // legal (assigns the value 7 to x)
  - b = true; // legal (assigns the value true to b)
  - c = '#'; // legal (assigns character # to c)
  - s = "cat" + "bert"; // legal (assigns the value "catbert" to s)
  - d = x – 3; // legal (assigns the integer value 7 – 3 = 4 to double d)

  - b = 5; // illegal! (cannot assign int to boolean)
  - y = x + b; // illegal! (cannot add int and boolean)
  - c = x; // illegal! (cannot assign int to char)



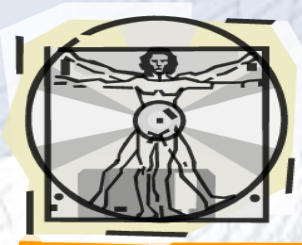
Character Strings

- Every character string is an object in Java, defined by the `String` class
- Every string literal, delimited by double quotation marks, represents a `String` object
- The *string concatenation operator* (+) is used to append one string to the end of another
- It can also be used to append a number to a string
- A string literal cannot be broken across two lines in a program



String Concatenation

- The plus operator (+) is also used for arithmetic addition
- The function that the + operator performs depends on the type of the information on which it operates
- If both operands are strings, or if one is a string and one is a number, it performs string concatenation
- If both operands are numeric, it adds them
- The + operator is evaluated left to right
- Parentheses can be used to force the operation order



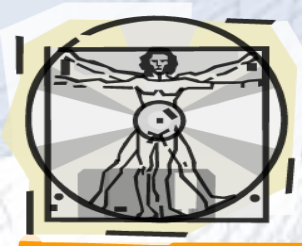
Escape Sequences

- What if we wanted to print a double quote character?
- The following line would confuse the compiler because it would interpret the second quote as the end of the string

```
System.out.println ("I said "Hello" to you.");
```

- An *escape sequence* is a series of characters that represents a special character
- An escape sequence begins with a backslash character (\), which indicates that the character(s) that follow should be treated in a special way

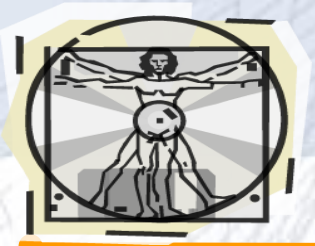
```
System.out.println ("I said \"Hello\" to you.");
```



Escape Sequences

- Some Java escape sequences:

Escape Sequence	Meaning
`\b`	backspace
`\t`	tab
`\n`	newline
`\r`	carriage return
`\"`	double quote
`\'`	single quote
`\\`	backslash



Escape sequences

- **escape sequence**: A special sequence of characters used to represent certain special characters in a string.

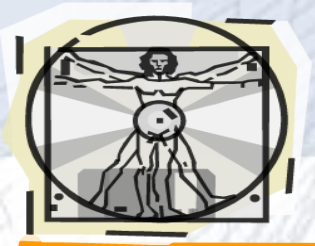
`\t`	tab character
`\n`	new line character
`\"`	quotation mark character
`\\`	backslash character

- Example:

```
System.out.println ("\\hello\nhow\tare \"you\"?\\\\");
```

- Output:

```
\hello
how   are "you"?\\
```



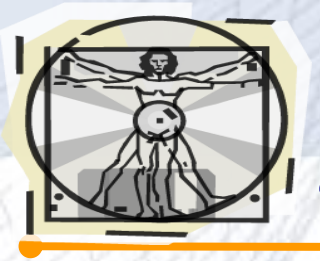
Questions

- What is the output of the following `print` statements?

```
System.out.println ("\ta\tb\tc")
System.out.println ("\\\\")
System.out.println ("'")
System.out.println ("\"\"\"")
System.out.println ("C:\nin\the downward spiral")
```

- Write a `System.out.println` statement to produce this output:

```
/ \ // \\ /// \\\
```



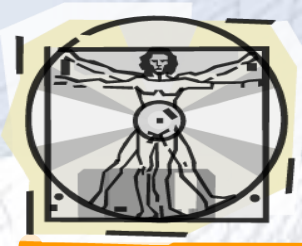
Answers

- Output of each `print` statement:

```
        a       b       c
 \\
 '
 """
 C:
 in      he downward spiral
```

- `System.out.println` statement to produce the line of output:

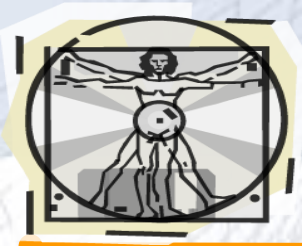
```
System.out.println ("/ \\ // \\\\ /// \\\\\\")
```



Characters

- The *ASCII character set* is older and smaller than Unicode, but is still quite popular
- The ASCII characters are a subset of the Unicode character set, including:

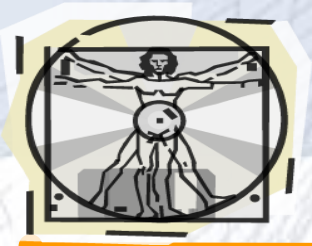
uppercase letters	**A, B, C, …**
lowercase letters	**a, b, c, …**
punctuation	**period, semi-colon, …**
digits	**0, 1, 2, …**
special symbols	**&, |, \, …**
control characters	**carriage return, tab, ...**



Boolean

- A `boolean` value represents a true or false condition
- A boolean also can be used to represent any two states, such as a light bulb being on or off
- The reserved words **`true`** and **`false`** are the only valid values for a boolean type

```
boolean done = false;
```



Character and String Constants

- **char constants**: Single character enclosed in single quotes (‘…’) including:
  - **letters and digits**: ‘A’, ‘B’, ‘C’, …, ‘a’, ‘b’, ‘c’, …, ‘0’, ‘1’, …, ‘9’
  - **punctuation symbols**: ‘*’, ‘#’, ‘@’, ‘$’ (except single quote and backslash ‘\’)
  - **escape sequences**: (see below)
- **String constants**: Zero or more characters enclosed in double quotes (“…”)
  - (same as above, but may not include a double quote or backslash)
- **Escape sequences**: Allows us to include single/double quotes and other special characters:

  `\"` double quote `\n` new-line character (start a new line)

  `\'` single quote `\t` tab character

  `\\` backslash

- **Examples**: `char x = '\''` → (x contains a single quote)

  `"\"Hi there!\""` → `"Hi there!"`

  `"C:\\WINDOWS"` → `C:\WINDOWS`

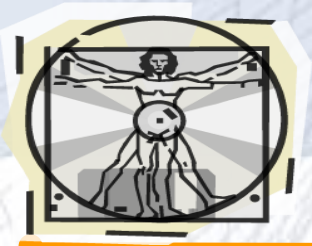
  `System.out.println( "Line 1\nLine 2" )` prints

```
Line 1
Line 2
```

Reading Numbers from the Keyboard

Use of

```
Scanner input = new Scanner(System.in);
```

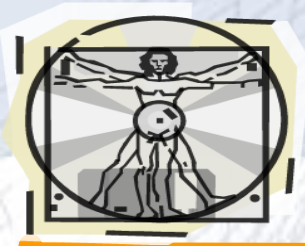


Reading Input from the Console

- 1. Create a Scanner object
  - `Scanner input = new Scanner(System.in);`
- 2. Use the method nextDouble() to obtain to a double value. For example,
  - `System.out.print("Enter a double value: ");`
  - `Scanner input = new Scanner(System.in);`
  - `double d = input.nextDouble();`

ComputeAreaWithConsoleInput **Run**

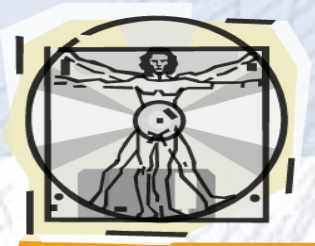
ComputeAverage **Run**



Implicit Import and Explicit Import

```
java.util.* ; // Implicit import

java.util.Scanner; // Explicit Import
```

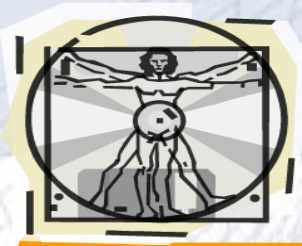
No performance difference



Reading Numbers

```
Scanner input = new Scanner(System.in);
int value = input.nextInt();
```

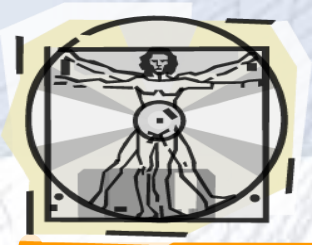
Method	Description
`nextByte()`	reads an integer of the `byte` type.
`nextShort()`	reads an integer of the `short` type.
`nextInt()`	reads an integer of the `int` type.
`nextLong()`	reads an integer of the `long` type.
`nextFloat()`	reads a number of the `float` type.
`nextDouble()`	reads a number of the `double` type.



Boolean

- A `boolean` value represents a true or false condition
- A boolean also can be used to represent any two states, such as a light bulb being on or off
- The reserved words `true` and `false` are the only valid values for a boolean type

```
boolean done = false;
```

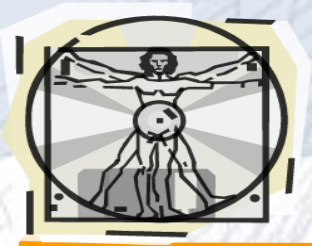


Arithmetic Expressions

- An *expression* is a combination of one or more operands and their operators
- *Arithmetic expressions* compute numeric results and make use of the arithmetic operators:

Addition	+
Subtraction	-
Multiplication	*
Division	/
Remainder	%

- **If either or both operands associated with an arithmetic operator are floating point, the result is a floating point**



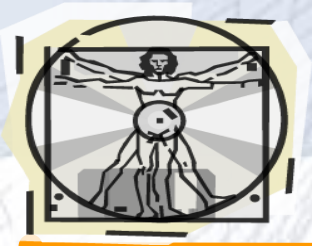
Division and Remainder

- If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded)

```
14 / 3      equals?   4

8 / 12      equals?   0
```

- The remainder operator (%) returns the remainder after dividing the second operand into the first

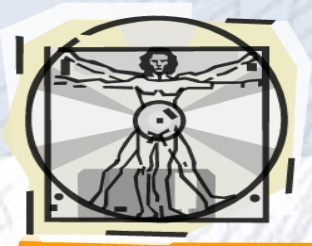
```
14 % 3      equals?   2

8 % 12      equals?   8
```



Quick Check

What are the results of the following expressions?

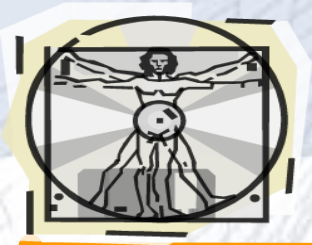
```
    12 / 2
12.0 / 2.0
    10 / 4
10 / 4.0
    4 / 10
4.0 / 10
    12 % 3
    10 % 3
    3 % 10
```



Quick Check

What are the results of the following expressions?

```
   12 / 2   =   6
12.0 / 2.0  =   6.0
   10 / 4   =   2
10 / 4.0    =   2.5
   4 / 10   =   0
4.0 / 10    =   0.4
   12 % 3   =   0
   10 % 3   =   1
   3 % 10   =   0
```

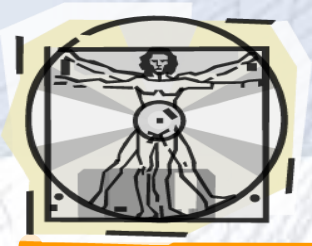


Operator Precedence

- Operators can be combined into complex expressions

```
result  =  total + count / max - offset;
```

- Operators have a well-defined precedence which determines the order in which they are evaluated
- Multiplication, division, and remainder are evaluated prior to addition, subtraction, and string concatenation
- Arithmetic operators with the same precedence are evaluated from left to right
- Parentheses can be used to force the evaluation order



Operator Precedence

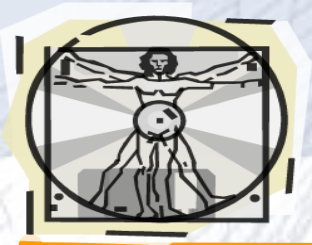
- What is the order of evaluation in the following expressions?

```
a + b + c + d + e
  1   2   3   4
```

```
a + b * c - d / e
    3   1   4   2
```

```
a / (b + c) - d % e
  2    1     4   3
```

```
a / (b * (c + (d - e)))
  4    3    2    1
```



Assignment Revisited

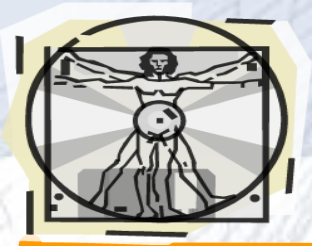
- The assignment operator has a lower precedence than the arithmetic operators

First the expression on the right hand side of the = operator is evaluated

```
answer   =   sum / 4 + MAX * lowest;
```

4 1 3 2

Then the result is stored in the variable on the left hand side



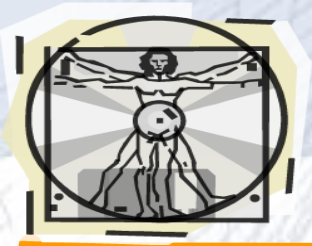
Assignment Revisited

- The right and left hand sides of an assignment statement can contain the same variable

First, one is added to the original value of count

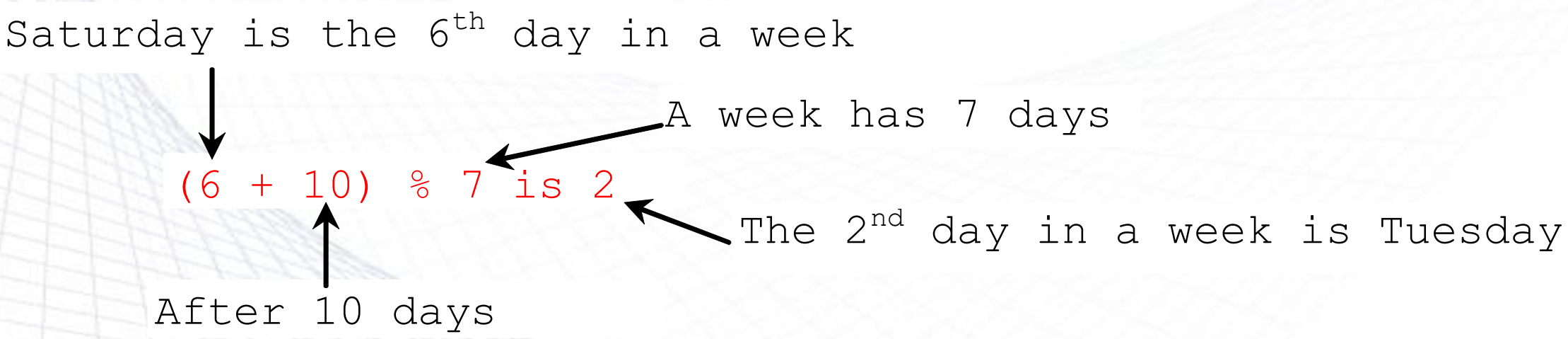
```
count  =  count + 1;
```

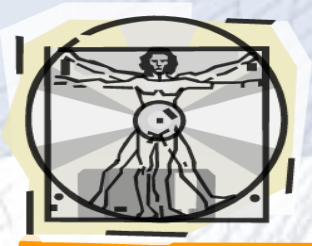
Then the result is stored back into count (overwriting the original value)



Remainder Operator

Remainder is very useful in programming. For example, an even number % 2 is always 0 and an odd number % 2 is always 1. So you can use this property to determine whether a number is even or odd. Suppose today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression:

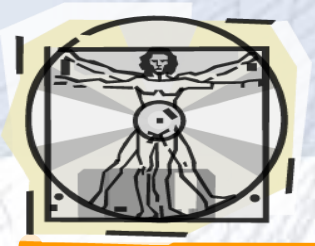




Problem: Reaching the Digit

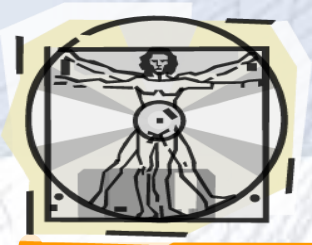
- Write a Java program which reads a two-digit integer values from keyboard and displays each digits as follows.

- Enter the two digit number : **78**
- Digits are : 7 and 8



Solution

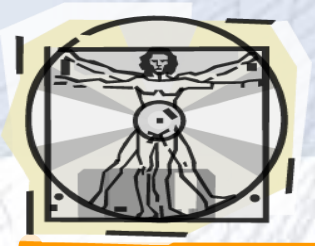
```
public static void main(String[] args) {
 int x1;   int x2;   int x3;
 Scanner input= new Scanner(System.in);

 System.out.print("Enter a two-digit Number  : ");
 x1=input.nextInt();

 x2= x1 / 10 ;
 System.out.println("First Digit is  : " + x2);

 x3=   x1 % 10  ;
 System.out.println("Second  Digit is  : " + x3);
 }
```



Problem: Converting time/seconds

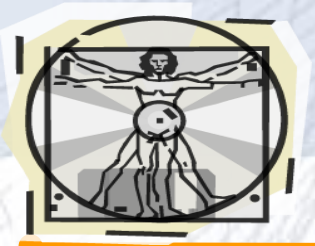
- Read an integer value from Keyboard which contains the number of secons, and convert this value to hours/minutes/seconds manner as follows

- Enter the seconds: **25851**
- 7 hours 10 minutes and 51 seconds



Solution

```
public static void main(String[] args) {
  int x1, hour, minute, second;
  Scanner input= new Scanner(System.in);

  System.out.print("Enter the seconds  : ");
  x1=input.nextInt();

  hour= x1/3600;
  minute = x1%3600/60;
  second = x1%60;

  System.out.println(hour + "hours " + minute + " minutes and " + second +
"seconds");
  }
```



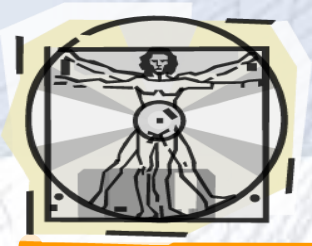
Problem: Displaying Time

Write a program that obtains minutes and remaining seconds from seconds.

DisplayTime **Run**

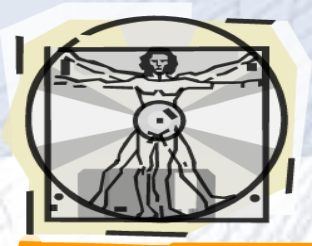
Prededefined Classes (Libraries)

Math Class



Exponent Operations

```
System.out.println(Math.pow(2, 3));
// Displays 8.0
System.out.println(Math.pow(4, 0.5));
// Displays 2.0
System.out.println(Math.pow(2.5, 2));
// Displays 6.25
System.out.println(Math.pow(2.5, -2));
// Displays 0.16
```



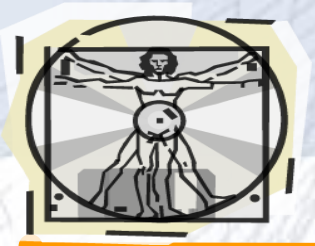
Java's `Math` class

Method name	Description
`Math.abs(`*value*`)`	absolute value
`Math.ceil(`*value*`)`	moves up to ceiling
`Math.floor(`*value*`)`	moves down to floor
`Math.log10(`*value*`)`	logarithm, base 10
`Math.max(`*value1*`, `*value2*`)`	larger of two values
`Math.min(`*value1*`, `*value2*`)`	smaller of two values
`Math.pow(`*base*`, `*exp*`)`	*base* to the *exp* power
`Math.random()`	random `double` between 0 and 1
`Math.rint(`*value*`)`	Round int, nearest whole number
`Math.sqrt(`*value*`)`	square root

`Math.sin(`*value*`)`
`Math.cos(`*value*`)`
`Math.tan(`*value*`)`
	sine/cosine/tangent of an angle in radians

`Math.toDegrees(`*value*`)` `Math.toRadians(`*value*`)`	convert degrees to radians and back

Constant	Description
`Math.E`	2.7182818...
`Math.PI`	3.1415926...



No output?

- Simply calling these methods produces no visible result.

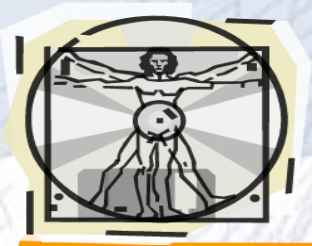
```
Math.pow(3, 4);    // no output
```

- Math method calls use a Java feature called *return values* that cause them to be treated as expressions.
- The program runs the method, computes the answer, and then "replaces" the call with its computed result value.

```
Math.pow(3, 4);    // no output
81.0;              // no output
```

- To see the result, we must print it or store it in a variable.

```
double result = Math.pow(3, 4);
System.out.println(result);      // 81.0
```



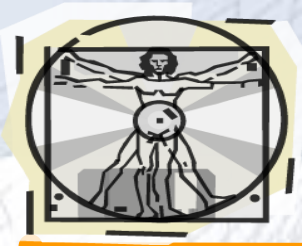
Calling `Math` methods

`Math.`**methodName**`(`**parameters**`)`

- Examples:

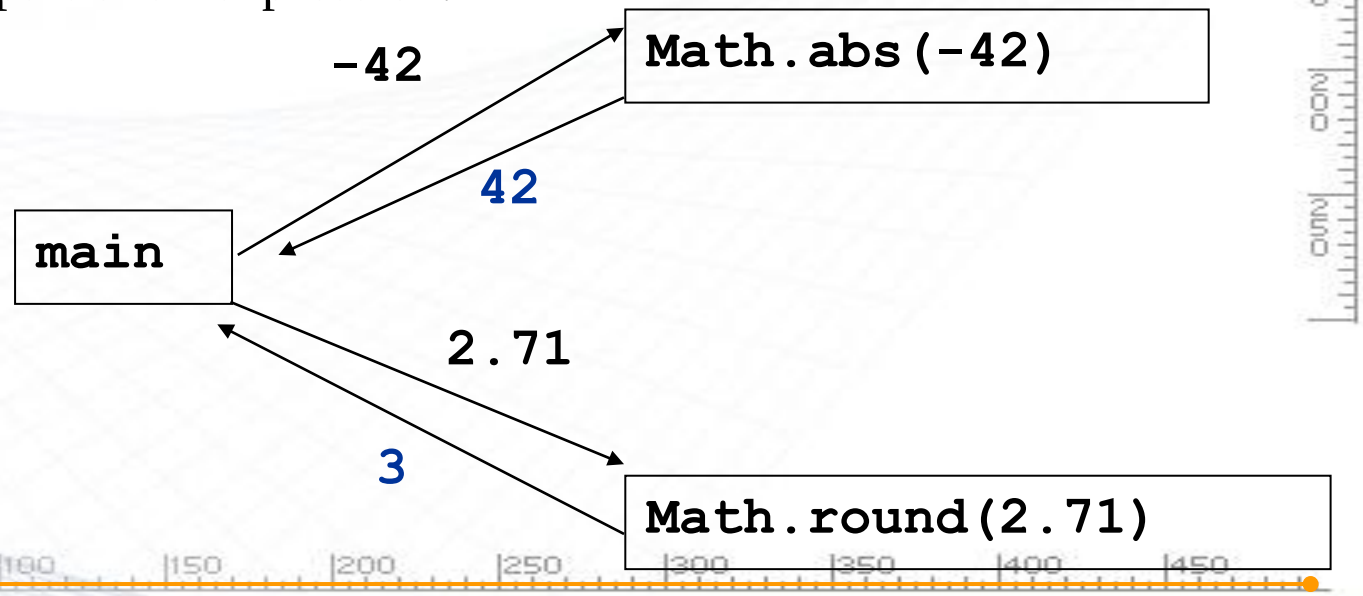
```
double squareRoot = Math.sqrt(121.0);
System.out.println(squareRoot);             // 11.0

int absoluteValue = Math.abs(-50);
System.out.println(absoluteValue);          // 50

System.out.println(Math.min(3, 7) + 2);     // 5
```

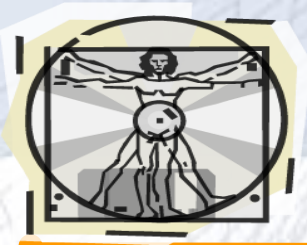
- The `Math` methods do not print to the console.
  - Each method produces ("returns") a numeric result.
  - The results are used as expressions (printed, stored, etc.).



Return

- **return**: To send out a value as the result of a method.
  - The opposite of a parameter:
    - Parameters send information *in* from the caller to the method.
    - Return values send information *out* from a method to its caller.
      - A call to the method can be used as part of an expression.





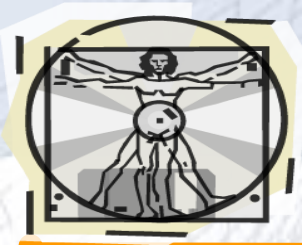
Why return and not print?

- It might seem more useful for the `Math` methods to print their results rather than returning them. Why don't they?
- Answer: Returning is more flexible than printing.
  - We can compute several things before printing:

```
double pow1 = Math.pow(3, 4);
double pow2 = Math.pow(10, 6);
System.out.println("Powers are " + pow1 + " and " + pow2);
```

  - We can combine the results of many computations:

```
double k = 13 * Math.pow(3, 4) + 5 - Math.sqrt(17.8);
```



What is output by the following code?

```
double a = -1.9;
double b = 2.25;
System.out.print( Math.floor(a) +" " + Math.ceil(b) + " " + a);
```

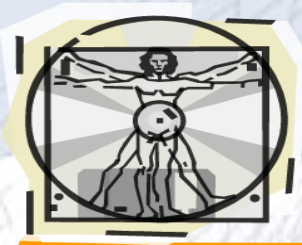
A. 3.0

B. -2.0 3.0 -2.0

C. -1.0 3.0 -1.0

D. -1 3 -1.9

E. -2.0 3.0 -1.9



Math questions

- Evaluate the following expressions:

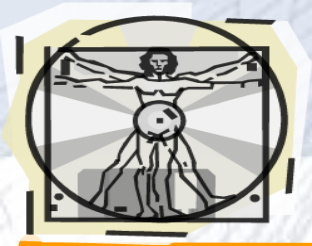
```
Math.abs(-1.23)
Math.pow(3, 2)
Math.pow(10, -2)
Math.sqrt(121.0) - Math.sqrt(256.0)
Math.round(Math.PI) + Math.round(Math.E)
Math.ceil(6.022) + Math.floor(15.9994)
Math.abs(Math.min(-3, -5))
```

- `Math.max` and `Math.min` can be used to bound numbers.

  Consider an `int` variable named `age`.

  What statement would replace negative ages with 0?

  What statement would cap the maximum age to 40?



Quirks of real numbers

- Some `Math` methods return `double` or other non-`int` types.

```
int x = Math.pow(10, 3); // ERROR: incompat. types
```

- Some `double` values print poorly (too many digits).

```
double result = 1.0 / 3.0;
System.out.println(result);     // 0.3333333333333
```

- The computer represents `doubles` in an imprecise way.

```
System.out.println(0.1 + 0.2);
```

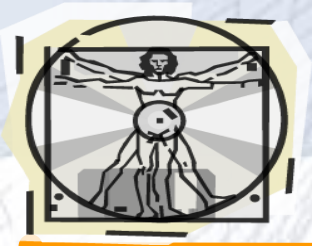
  - Instead of 0.3, the output is `0.30000000000000004`

```java
import java.util.Scanner;

public class Quadratic {
   public static void main(String[] args)   {
      int a, b, c;  // ax^2 + bx + c
      double discriminant, root1, root2;

      Scanner scan = new Scanner(System.in);

      System.out.print("Enter the coefficient of x squared: ");  a = scan.nextInt();

      System.out.print("Enter the coefficient of x: ");          b = scan.nextInt();

      System.out.print("Enter the constant: ");                  c = scan.nextInt();

      discriminant = Math.pow(b, 2) - (4 * a * c);
      root1 = ((-1 * b) + Math.sqrt(discriminant)) / (2 * a);
      root2 = ((-1 * b) - Math.sqrt(discriminant)) / (2 * a);

      System.out.println("Root #1: " + root1);
      System.out.println("Root #2: " + root2);
   }
}
```

Sample Run

```
Enter the coefficient of x squared: 3
Enter the coefficient of x: 8
Enter the constant: 4
Root #1: -0.666666666666666
Root #2: -2.0
```

Arithmetic Expressions

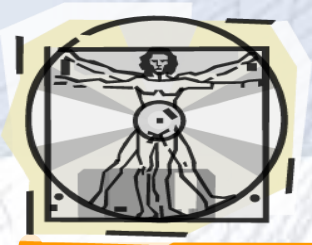


Arithmetic expressions

An arithmetic expression is a combination of variables, constants, and operators.

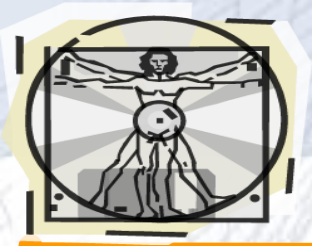
For example,

- a*b-c → a*b-c
- (m+n)(x+y) → (m+n)*(x+y)
- ax^2+bx+c → a*x*x+b*x+c



Example

```
// computing the roots of a quadratic equation:
    double      a,          // coefficient of x squared
                b,          // coefficient of x
                c,          // 3rd term in equation
                x1,         // first root
                x2;         // second root


// read in values for a, b, and c – not shown here …


x1 = (-b + Math.sqrt(Math.pow(b, 2) – (4 * a * c))) / (2 * a);
x2 = (-b - Math.sqrt(Math.pow(b, 2) – (4 * a * c))) / (2 * a);
```

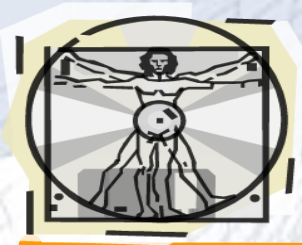


Arithmetic Expressions

$$\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)$$

is translated to

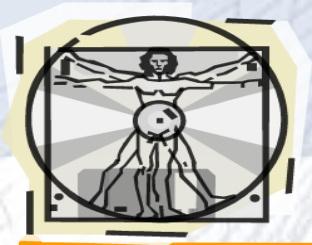
(3+4*x)/5 – 10*(y-5)*(a+b+c)/x + 9*(4/x + (9+x)/y)



How to Evaluate an Expression

- Though Java has its own way to evaluate an expression behind the scene, the result of a Java expression and its corresponding arithmetic expression are the same. Therefore, you can safely apply the arithmetic rule for evaluating a Java expression.

```
3 + 4 * 4 + 5 * (4 + 3) - 1
                (1) inside parentheses first
3 + 4 * 4 + 5 * 7 - 1
            (2) multiplication
3 + 16 + 5 * 7 - 1
         (3) multiplication
3 + 16 + 35 - 1
     (4) addition
19 + 35 - 1
    (5) addition
54 - 1
   (6) subtraction
53
```



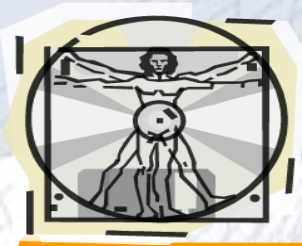
Problem: Converting Temperatures

Write a program that converts a Fahrenheit degree to Celsius using the formula:

$$celsius = (\frac{5}{9})(fahrenheit - 32)$$

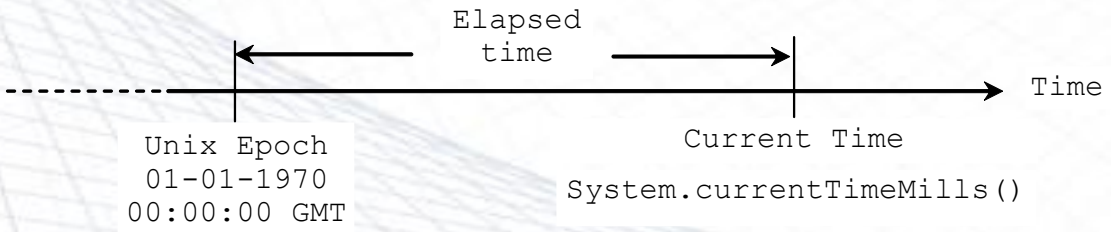
Note: you have to write
celsius = (5.0 / 9) * (fahrenheit – 32)

FahrenheitToCelsius **Run**



Problem: Displaying Current Time

- Write a program that displays current time in GMT in the format hour:minute:second such as 1:45:19.
- The currentTimeMillis method in the System class returns the current time in milliseconds since the midnight, January 1, 1970 GMT. (1970 was the year when the Unix operating system was formally introduced.) You can use this method to obtain the current time, and then compute the current second, minute, and hour as follows.

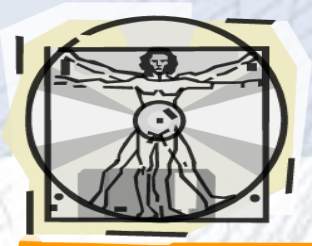


ShowCurrentTime

Run

Data Conversion

Casting



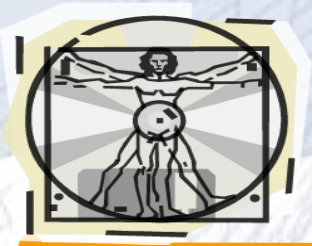
Data Conversion

Widening Conversions

From	To
byte	short, int, long, float, or double
short	int, long, float, or double
char	int, long, float, or double
int	long, float, or double
long	float or double
float	double

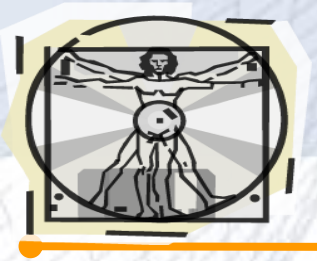
Narrowing Conversions

From	To
byte	char
short	byte or char
char	byte or short
int	byte, short, or char
long	byte, short, char, or int
float	byte, short, char, int, or long
double	byte, short, char, int, long, or float



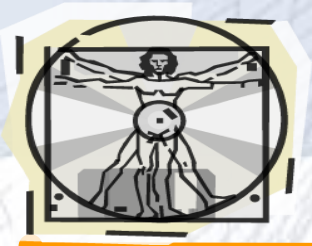
Data Conversions

- Sometimes it is convenient to convert data from one type to another
- For example, we may want to treat an integer as a floating point value during a computation
- Conversions must be handled carefully to avoid losing information
- *Widening conversions* are safest because they tend to go from a small data type to a larger one (such as a `short` to an `int`)
- *Narrowing conversions* can lose information because they tend to go from a large data type to a smaller one (such as an `int` to a `short`)



Data Conversions

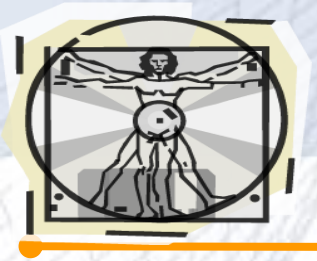
- In Java, data conversions can occur in three ways:
  - assignment conversion
  - arithmetic promotion
  - casting
- *Assignment conversion* occurs when a value of one type is assigned to a variable of another
  - Only widening conversions can happen via assignment
- *Arithmetic promotion* happens automatically when operators in expressions convert their operands



Data Conversions

- *Casting* is the most powerful, and dangerous, technique for conversion
  - Both widening and narrowing conversions can be accomplished by explicitly casting a value
  - To cast, the type is put in parentheses in front of the value being converted
- For example, if `total` and `count` are integers, but we want a floating point result when dividing them, we can cast `total`:

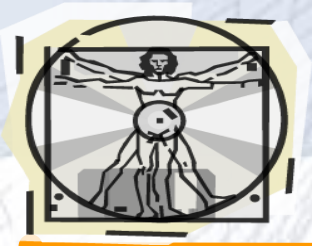
```
result = (float) total / count;
```



Numeric Type Conversion

Consider the following statements:

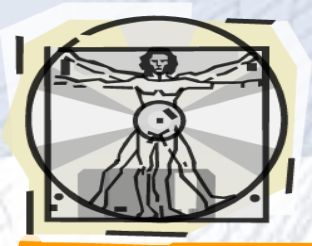
```
byte i = 100;
long k = i * 3 + 4;
double d = i * 3.1 + k / 2;
```



Conversion Rules

When performing a binary operation involving two operands of different types, Java automatically converts the operand based on the following rules:

1. If one of the operands is double, the other is converted into double.
2. Otherwise, if one of the operands is float, the other is converted into float.
3. Otherwise, if one of the operands is long, the other is converted into long.
4. Otherwise, both operands are converted into int.



Type Casting

Implicit casting

```
double d = 3;
```
(type widening)

Explicit casting

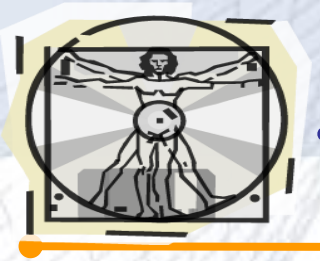
```
int i = (int)3.0;
```
(type narrowing)

```
int i = (int)3.9;
```
(Fraction part is truncated)

What is wrong? int x = 5 / 2.0;

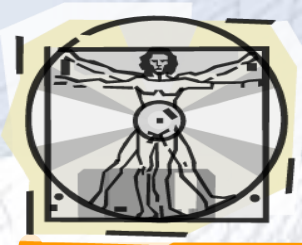
range increases

byte, short, int, long, float, double



Type Casting is Explicit Conversion of Type

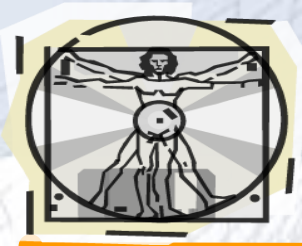
`int(4.8)`	has value	4
`double(5)`	has value	5.0
`double(7/4)`	has value	1.0
`double(7) / double(4)`	has value	1.75



Problem: Keeping Two Digits After Decimal Points

Write a program that displays the sales tax with two digits after the decimal point.

SalesTax Run



Casting in an Augmented Expression

In Java, an augmented expression of the form **x1 op= x2** is implemented as **x1 = (T)(x1 op x2)**, where **T** is the type for **x1**. Therefore, the following code is correct.

```
int sum = 0;
sum += 4.5; // sum becomes 4 after this statement
```

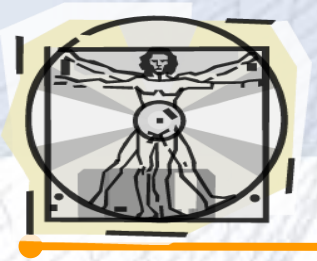
sum += 4.5 is equivalent to **sum = (int)(sum + 4.5)**.

Additional Operators

Augmented Assignment

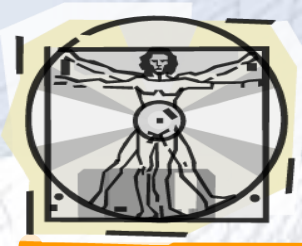
Increment

Decrement



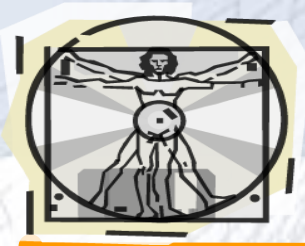
Augmented Assignment Operators

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Division assignment	`i /= 8`	`i = i / 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`



Increment and Decrement Operators

Operator	*Name*	*Description*	*Example (assume i = 1)*
`++var`	preincrement	Increment `var` by `1`, and use the new `var` value in the statement	`int j = ++i;` `// j is 2, i is 2`
`var++`	postincrement	Increment `var` by `1`, but use the original `var` value in the statement	`int j = i++;` `// j is 1, i is 2`
`--var`	predecrement	Decrement `var` by `1`, and use the new `var` value in the statement	`int j = --i;` `// j is 0, i is 0`
`var--`	postdecrement	Decrement `var` by `1`, and use the original `var` value in the statement	`int j = i--;` `// j is 1, i is 0`



Increment and Decrement Operators, cont.

```
int i = 10;
int newNum = 10 * i++;
```

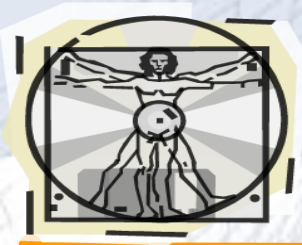
Same effect as

```
int newNum = 10 * i;
i = i + 1;
```

```
int i = 10;
int newNum = 10 * (++i);
```

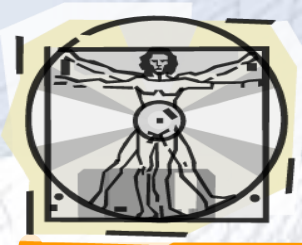
Same effect as

```
i = i + 1;
int newNum = 10 * i;
```



Increment and Decrement Operators, cont.

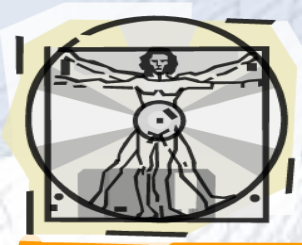
- Using increment and decrement operators makes expressions short, but it also makes them complex and difficult to read. Avoid using these operators in expressions that modify multiple variables, or the same variable for multiple times such as this: int k = ++i + i.



Assignment Expressions and Assignment Statements

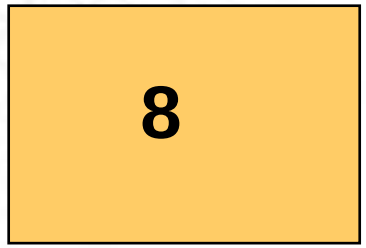
Prior to Java 2, all the expressions can be used as statements. Since Java 2, only the following types of expressions can be statements:

```
variable op= expression;       // Where op is +, -, *, /, or %
++variable;
variable++;
--variable;
variable--;
```



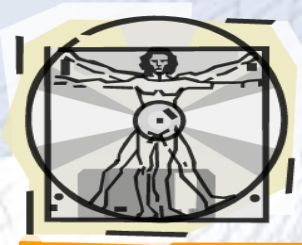
More Java Operators

```
int  age;

age = 8;

age = age + 1;
```



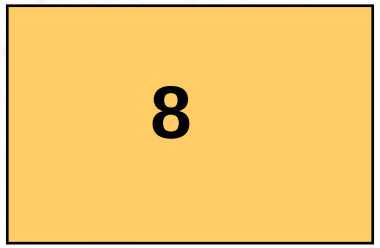
age

age

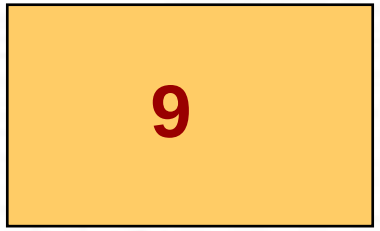


Prefix Form: Increment Operator

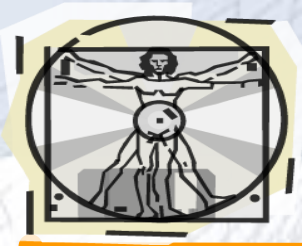
```
int  age;

age = 8;

++age;
```



age

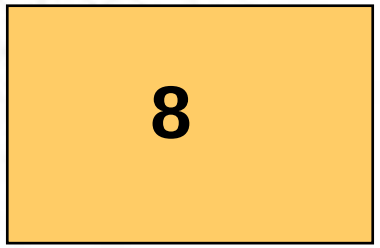


age

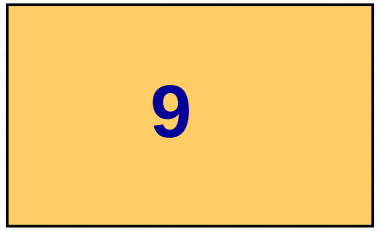


Postfix form: Increment Operator

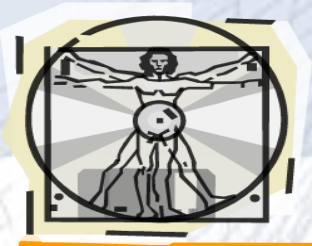
```
int  age;

age = 8;

age++;
```



age

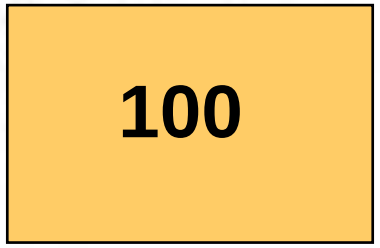


age

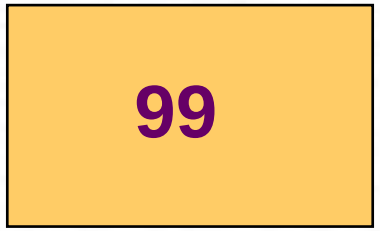


Decrement Operator

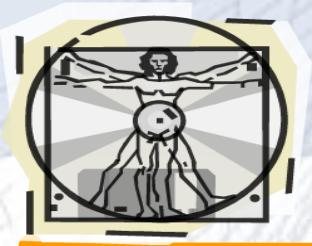
```
int  dogs;

dogs = 100;

dogs--;
```



dogs



dogs

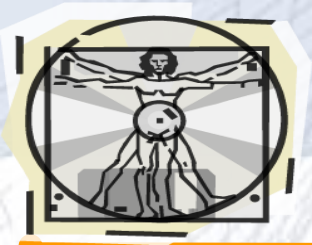


Which form to use?

- When the increment (or decrement) operator is used in a "*stand alone*" statement solely to add one (or subtract one) from a variable's value, it can be used in either prefix or postfix form.


```
dogs--;
```

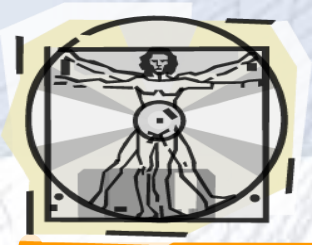
```
--dogs;
```



BUT…

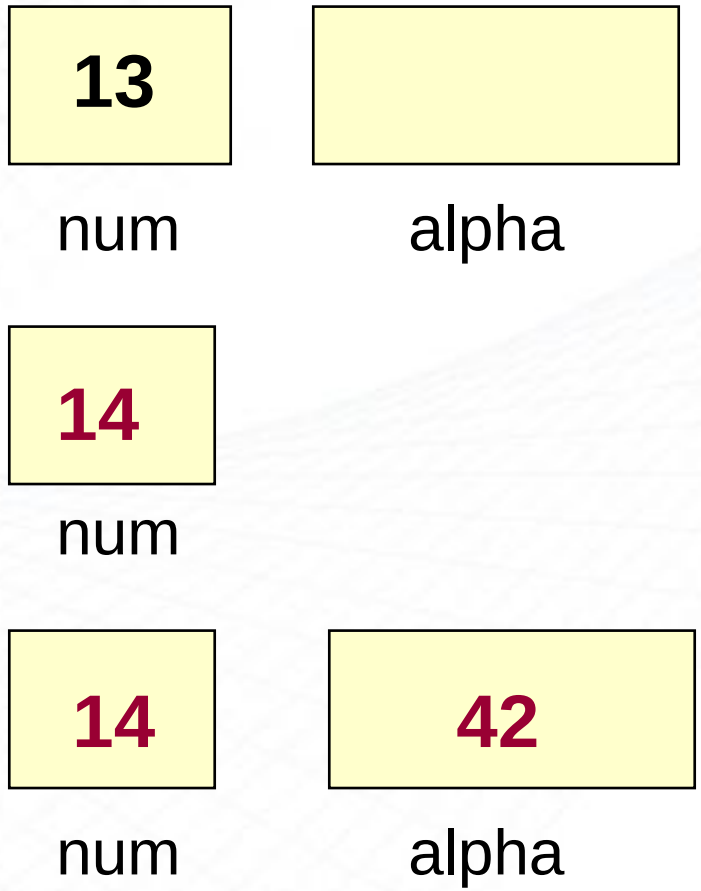
- **When the increment (or decrement) operator is used in a statement with other operators, the prefix and postfix forms can yield *different* results.**

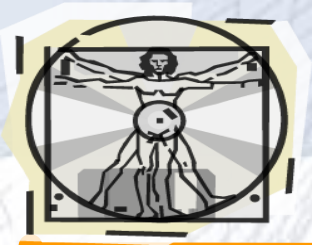
LET'S SEE HOW. . .



“First increment, then use”

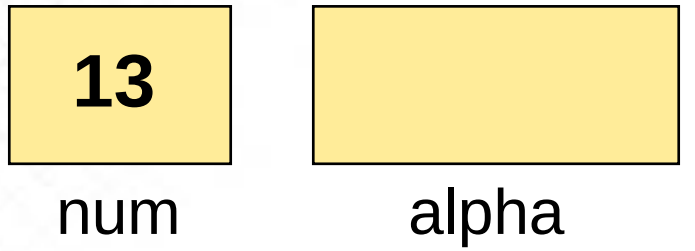
```
int  alpha;
int  num;

num = 13;


alpha = ++num * 3;
```

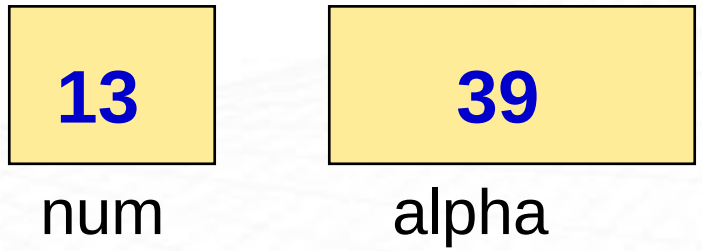


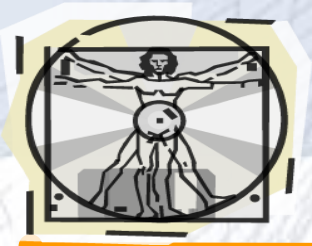


“Use, then increment”

```
int  alpha;
int  num;

num = 13;


alpha = num++   * 3;
```





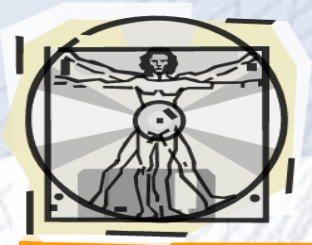


Problem: Computing Loan Payments

- This program lets the user enter the interest rate, number of years, and loan amount, and computes monthly payment and total payment.

$$monthlyPayment = \frac{loanAmount \times monthlyInterestRate}{1 - \frac{1}{(1 + monthlyInterestRate)^{numberOfYears \times 12}}}$$

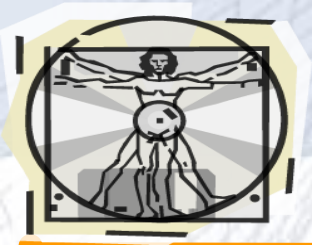
ComputeLoan Run



Problem: Monetary Units

- This program lets the user enter the amount in decimal representing dollars and cents and output a report listing the monetary equivalent in single dollars, quarters, dimes, nickels, and pennies. Your program should report maximum number of dollars, then the maximum number of quarters, and so on, in this order.

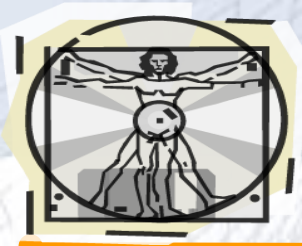
ComputeChange Run



Common Errors and Pitfalls

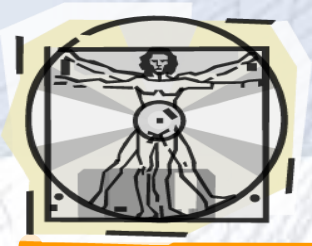
- Common Error 1: Undeclared/Uninitialized Variables and Unused Variables
- Common Error 2: Integer Overflow
- Common Error 3: Round-off Errors
- Common Error 4: Unintended Integer Division
- Common Error 5: Redundant Input Objects

- Common Pitfall 1: Redundant Input Objects



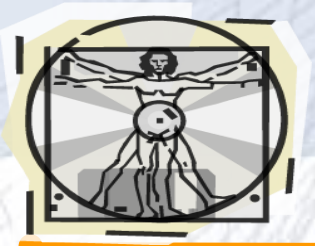
Common Error 1: Undeclared/Uninitialized Variables and Unused Variables

```
double interestRate = 0.05;
double interest = interestrate * 45;
```



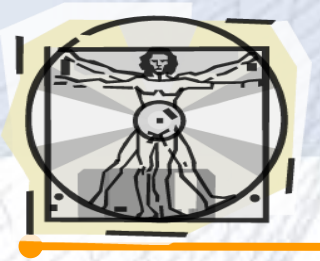
Common Error 2: Integer Overflow

```
int value = 2147483647 + 1;
// value will actually be -2147483648
```



Common Error 3: Round-off Errors

```
System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);

System.out.println(1.0 - 0.9);
```



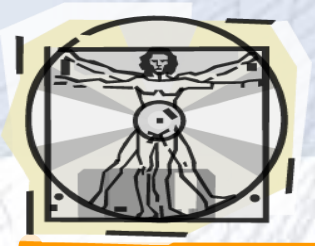
Common Error 4: Unintended Integer Division

```
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2;
System.out.println(average);
```

(a)

```
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2.0;
System.out.println(average);
```

(b)



Common Pitfall 1: Redundant Input Objects

```
Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input.nextInt();

Scanner input1 = new Scanner(System.in);
System.out.print("Enter a double value: ");
double v2 = input1.nextDouble();
```