# COMP231

## Advanced Programming

Chapter 2 Elementary Programming

Compiled By: Dr. Majdi Mafarja
Fall Semester 2017/2018

*animation*

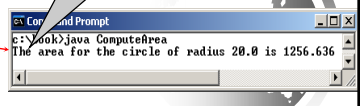
## Trace a Program Execution

```
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



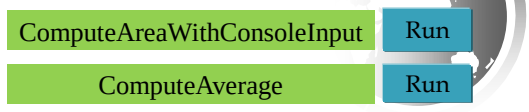
# Reading Input from the Console

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();
```








# 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.

# 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);
```




# 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;`

# Named Constants

```
final datatype CONSTANTNAME = VALUE;

final double PI = 3.14159;
final int SIZE = 3;
```




# Naming Conventions

- Choose meaningful and descriptive names.
- Variables and method names:
  - Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name. For example, the variables `radius` and `area`, and the method `computeArea`.

# Naming Conventions, cont.

- Class names:
  - Capitalize the first letter of each word in the name. For example, the class name `ComputeArea`.
- Constants:
  - Capitalize all letters in constants, and use underscores to connect words. For example, the constant `PI and` MAX_VALUE





# 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 |

# Reading Numbers from the Keyboard

```
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. |





# Numeric Operators

| Name | Meaning | Example | Result |
|---|---|---|---|
| + | Addition | 34 + 1 | 35 |
| - | Subtraction | 34.0 – 0.1 | 33.9 |
| * | Multiplication | 300 * 30 | 9000 |
| / | Division | 1.0 / 2.0 | 0.5 |
| % | Remainder | 20 % 3 | 2 |

# Integer Division

+, -, *, /, and %

5 / 2 yields an integer 2.

5.0 / 2 yields a double value 2.5

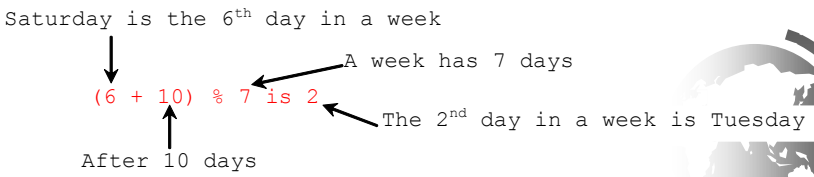
5 % 2 yields 1 (the remainder of the division)





# 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:




# 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
```





# 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

# 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

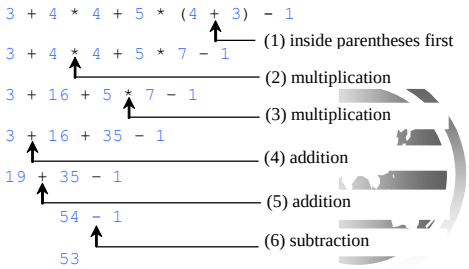
```
(3+4*x)/5 – 10*(y-5)*(a+b+c)/x + 9*(4/x + (9+x)/y)
```





# 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.



# 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





# 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--;

# Numeric Type Conversion

Consider the following statements:

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

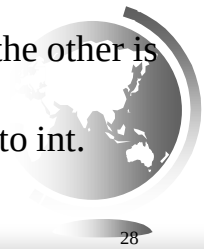




# Conversion Rules

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

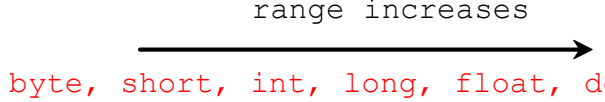
```
double d = 3;
```
(type widening)

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;




byte, short, int, long, float, double





# 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)**.

# 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)