



Northeastern  
University

# Lecture 2: Fundamentals of Programming - 2

Prof. Chen-Hsiang (Jones) Yu, Ph.D.  
College of Engineering

Materials are edited by Prof. Jones Yu from Prof. Charlie Wiseman's materials.

# Outline

---

- Introduction to Computation and Programming
- Variables, I/O, Types and Strings
- Control Flow and Conditions
- Methods
- Arrays
- File I/O

# Outline

---

- Introduction to Computation and Programming
- Variables, I/O, Types and Strings
- Control Flow and Conditions
- Methods
- Arrays
- File I/O

# Variables, I/O, Types and Strings

# Variables

---

- Every variable has a *value* that is stored in a particular memory location.
- Each variable has a *name* that the programmer uses to access and modify that variable's value.
- Each variable holds *exactly one value*.
- Over time, as a program executes, *the value of a variable can change*.

# Variable Names

---

- In Java, variable names:
  - » Must **start with either a letter** (uppercase or lowercase) or **an underscore**
  - » Must contain only **letters**, **digits**, and **underscores**
  - » Are **case sensitive**
- Examples: `count`, `x`, `user_input2`, `hit_points`
- Invalid names: `42`, `5x`, `#yolo`, `file.cpp`, `a-b`

# Variable Declarations

---

- Every variable must be *declared* before you can use it.
- To declare a variable, give it a specific *type*:
  - » **int**: integer (whole number), positive or negative
  - » **double**: numbers with a fractional component
  - » **boolean**: boolean value (true or false)
  - » **char**: a single character
- Syntax: **TYPE NAME**;
  - » **int** count;
  - » **int** num\_vals;
  - » **double** average;
  - » **char** first\_initial;

# Variable Initialization

---

- You can *initialize* a variable when you declare it or you can do so afterwards.
- Syntax **after declaration**: NAME = VALUE;
- Syntax **during declaration**: TYPE NAME = VALUE;
- Examples
  - » count = 0;
  - » ultimate\_answer = 42;
  - » **int** num\_vals = 10;
  - » **double** pi = 3.14159;



# Data Types - Numbers

---

## ■ **int**

- » Integer, whole numbers
- » Examples: 0, 15, -100464, 420712003, -1
- » Range:  $-2^{31}$  (-2147483648) to  $2^{31}-1$  (2147483647)
- » 4 bytes of memory

## ■ **double**

- » Numbers with a fractional component (15 digit precision)
- » Examples: 11.23, -959.75, 0.5, -1.0
- » Range:  $\sim 10^{-308}$  to  $\sim 10^{308}$ , positive or negative
- » 8 bytes of memory

# Data Types - Alphanumeric

---

- **char**

- » Single character or symbol
- » Always put in **single quotes**
- » Examples: 'a', 'C', '3', '.', '\$'
- » **2 bytes** of memory

- **String**

- » A sequence of characters and/or symbols
- » Always put in **double quotes**
- » Examples: "Hello World", "475!", "a", "\$"

# Data Types - Boolean

---

- **boolean**

- » Boolean valued
- » Only values: **true**, **false**
- » At least one byte of memory

# Program Output

---

- `System.out.println()` is used to output the current value of a variable.
- Don't put the name of the variable in quotes
  - » This is one of the most common mistakes made by new programmers.
- Any values in quotes are printed out literally.
- Any values not in quotes are assumed to be variable names.

## Program Output (cont.)

---

- You can mix **literal text** in quotes with **variable names** with **the plus sign**.
- If you don't want a new line automatically added to the end of your output message you can use `System.out.print()` instead.
  - » Otherwise it works the same as `System.out.println()`

```
package edu.northeastern.csye6200;

import java.util.Scanner;

public class MyClass {

    public static void main(String[] args) {

        //TODO: Write your codes here
        System.out.print("Hello World");
        ...

    }

}
```

Example: [MyClass.java](#)

# Program Input

---

- We also need a way to get user input into our programs while they are running.
- Java doesn't (easily) allow reading directly from `System.in`
- Instead, you use a `Scanner` object that `handles reading the input` and ensures that the type of data you read matches what you want.

# Input with a Scanner

---

- First, you have to declare and initialize the Scanner object

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

- Then you call different methods on the Scanner object to read different types of values from the keyboard

» Read an **int**: `variable = input.nextInt();`

» Read a **double**:

`variable = input.nextDouble();`



# Exercise

---

- Write a Java program that reads **exactly three integers** from the user, calculates the average of the three numbers, and prints out the average.

# Answer

---

```
import java.util.Scanner;

public class ClassExamples {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        int a, b, c;
        double average;

        System.out.println("Enter three integers:");
        a = input.nextInt();
        b = input.nextInt();
        c = input.nextInt();

        average = (a + b + c) / 3.0;

        System.out.println("The average is " + average);

    }

}
```

The ".0" after 3 is necessary to get a **double** result!

# Format String

---

- The format string contains **literals** (items you want outputted verbatim), **converters**, and **flags**.
  - » A **converter** looks to the arguments to fill in a value
    - Starts with a **%** and ends with **a single character code**
  - » A **flag** modifies a converter with options
    - Goes between the **%** and **the converter code**
- Each time you use **a converter**, you must supply a corresponding **argument** (other than newline).

# Some Converters, Flags

---

Converter	Flag	Description
d		An integer
f		A float (includes double)
e		A float in scientific notation.
n		New line
	+	Includes the sign (positive or negative)
	,	Includes grouping characters
	.3	Three places after the decimal.

Many more options exist:

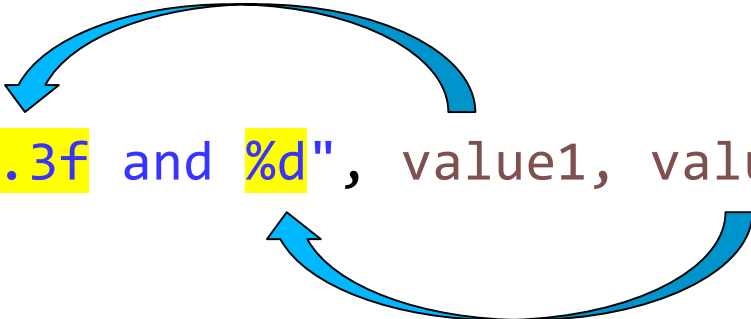
<https://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html#syntax>

# Example of Using String Formatting

---

```
double value1 = 3.1415;  
int value2 = 13;
```

```
System.out.printf("The result is %.3f and %d", value1, value2);
```

A diagram consisting of two blue curved arrows. The top arrow starts from the variable 'value1' in the code above and points to the '%.3f' format specifier in the code below. The bottom arrow starts from the variable 'value2' in the code above and points to the '%d' format specifier in the code below. The format specifiers are highlighted in yellow in the original image.

## Exercise

---

- Write a program that asks the user for a decimal value – output that value with exactly three decimal places, rounding as necessary.

Enter a value: 3.14159

Rounded: 3.142

# Answer

---

```
public static void main(String[] args) {  
    Scanner input = new Scanner(System.in);  
    System.out.print("Enter a value: ");  
    double value = input.nextDouble();  
    System.out.printf("Rounded: %.3f%n", value);  
}
```

# Flow Control and Conditions



# Control Flow

---

- Control flow is the order in which program statements are executed.
- So far, all of our programs have been executed straight-through from the first statement to the last.
- In general, you will need **more complex control flow**.
- For example, to choose between two (or more) possibilities.

# if-else

---

■ Generic form:

```
if(BOOLEAN EXPRESSION)
{
    YES/TRUE STATEMENTS
}
else
{
    NO/FALSE STATEMENTS
}
```

■ Example:

```
if(user == "Harry Potter")
{
    System.out.println("'ello Harry!");
}
else
{
    System.out.println("'ello Hermione!");
}
```

# Multiple `else if` Statements

---

```
if(x > 10)
{
    System.out.println("x is greater than 10");
}
else if(x > 5)
{
    System.out.println("x is between 6 and 10 inclusive");
}
else if(x > 0)
{
    System.out.println("x is between 1 and 5 inclusive");
}
else
{
    System.out.println("x is less than 1");
}
```

# while Loops

---

- **while** loops are used to repeat a set of Java statements *while* some condition is *true*.
- Example:

```
int iteration = 1;
while (iteration <= 100) {
    System.out.println("I will not expose the ignorance of the faculty.");
    iteration = iteration + 1;
}
```

# do-while Loops

---

- A **while** loop body **might be executed zero times** if the condition is never true
- If you need to always execute the body **at least once**, use a **do-while** loop
- Example:

```
int input_value;  
do {  
    System.out.print("Enter 1 to print this message again:");  
    input_value = input.nextInt();  
} while (input_value == 1);
```

# while Loops

---

- **While** loops are often used to repeat a task a fixed number of times, which leads to a similar structure based on **a counter variable**.

```
int count;  
count = 1;  
while (count <= 8) {  
    System.out.println(count + " squared is " + count*count);  
    count++;  
}
```

counter variable  
initialization

counter variable  
boolean expression

counter variable  
update

# for loops

---

- **for** loops are specialized loops based on that counter structure

counter variable  
initialization

counter variable  
boolean expression

counter variable  
update

```
int count;  
for (count = 1; count <= 8; count++) {  
    System.out.println(count + " squared is " + count * count);  
}
```

## Exercise

---

- Write a program that uses a **do-while** loop to read integer values from the user until a value between 1 and 100 (inclusive) is entered.



# Answer

---

```
int input_value;

do {
    System.out.print("Enter a number between 1 and 100 (inclusive): ");
    input_value = input.nextInt();
} while (input_value < 1 || input_value > 100);
```