Java How to Program

Early Objects

TENTH EDITION

Paul Deitel & Harvey Deitel

Chapter 2 Introduction to Java Application Input, Output and Operators

// Welcome1.java // Text-Printing program public class Welcome1 { // main method begins execution of Java Application public static void main(String[] args) { System.out.println("Welcome to Java Programming!"); } // End main method } // End class Welcome1

```
Welcome to Java Programming!
```

Comments, Spaces and Empty Lines

// Welcome.java

// Text-Printing program

- line 3: is empty line
- line 6:indentation and spaces // main method begins execution of Java Application

// End main method

// End class Welcome1

Java compiler or when you start to run your program, java ignore all of these comments and spaces ad empty line.

But a lot of the intuation behind that like:

- User Readability
- When you access the program to updates
- Hint what is the purpose of this program
- Documented for the future

// This is end-of-line comment

/* This is

multiplelines

comments

*/

Declaring Class

Public class Welcome1{

} // End of the class

Every Java program should contain at least one class

- public, is access modifier we will go in deeper in this but not now
- class, is the keyword to start your declartion of a class
- Welcome1, is the class name and start with capital letter and the file name should be as same as class name with .java as extenation.
- The braces is the start and end of the class body

  So java class is start with class keyword followed by the name of the class which start with capital letter and should you have the file with same name followed by .java

Java is case sensitive so Welcome1 is a class name and welcome1 is another class.

Declaring Method

public static void main(String[] args){

} // End of the main method

Every program must be called the main method and should be at most one main method to start your run of the application which hint from which point should the application start, otherwise no thing will be run.

- public, is access modifier we will go in deeper in this but not now.
- static, also will be defined later.
- void, means this method will not return anything.

Every method you create is indicated to return something or be void which indicates that not information will be returned from this method.

- main, is the method name which indicate that java will start from here.
- String[] args, later when we talked about Array.
- {} The body of the method

Perform output with system.out.println

System.out.println("Welcome to Java Programming!");

- System.out, is the standard output object and its defined in every java application no need to import.
- println, is a method responsible for displaying a line of text in your window, beside of move the cursor to beginning of next line the window.
- Welcome to Java Programming! ==> Is the output of this line and without double quotation.
  - **double quotation.** hint that it will print a string of characters.
  - **;** Semicolon indicate the end of the statement.

Compile the application

Now it's time to compile and run your first application in java.

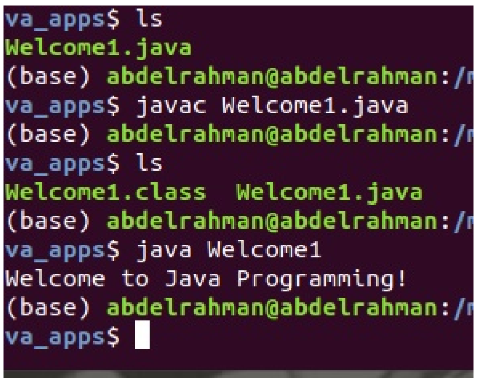
When you click on the button run, java will make this for you.

But if you using terminal like me, you will write **javac Welcome1.java**, javac for compile your java Welcome.java file, if there is no error in your code, java will compile this file to corresponding **bytescode** with the same name but another extention .class indicates that your program have no error.

For each time you edit your .java file you need to compile again.

If you have multiple file ***.java** will compile all files with .java extension for you.

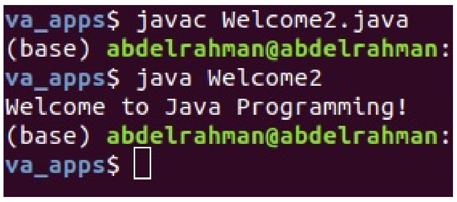
Then you need to write **java Welcome1** and this will run the bytecode file and your application is executed.



```
va_apps$ ls
Welcome1.java
(base) abdelrahman@abdelrahman:/r
va_apps$ javac Welcome1.java
(base) abdelrahman@abdelrahman:/r
va_apps$ ls
Welcome1.class  Welcome1.java
(base) abdelrahman@abdelrahman:/r
va_apps$ java Welcome1
Welcome to Java Programming!
(base) abdelrahman@abdelrahman:/r
va_apps$ 
```

Welcome2

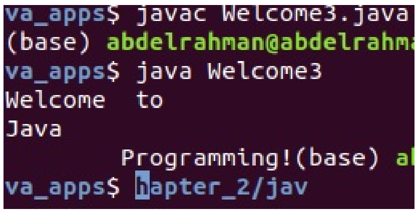
- println
- print



```
va_apps$ javac Welcome2.java
(base) abdelrahman@abdelrahman:
va_apps$ java Welcome2
Welcome to Java Programming!
(base) abdelrahman@abdelrahman:
va_apps$
```

Welcome3

- \n, \t, \r, \\, \"



```
va_apps$ javac Welcome3.java
(base) abdelrahman@abdelrahma
va_apps$ java Welcome3
Welcome  to
Java
        Programming!(base) al
va_apps$ hapter_2/jav
```

Welcome4

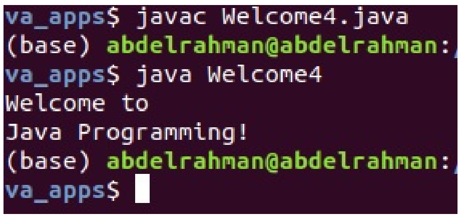
printf, f for **formatted"", it display a formatted data.

- System.out.printf("%s%n%s%n", "Welcome to", "Java Programming!");

Its have a comma separated list.

Before the first comma(first argument) is the format string specifier, or fixed text

This specifier indicates to the data type of the data coming after the first argument in their order in the specifier, which means first %s will be a placeholder(place for string), then after first comma will noticed that string **"Welcome to"**, then %n is indicate to new line instead of \n with println or print, then %s is another placeholder for another string which is "Java Programming!", you will see that number of argument after first argument will equal to number of placeholder in your specifier(fixed text) which before first comma.

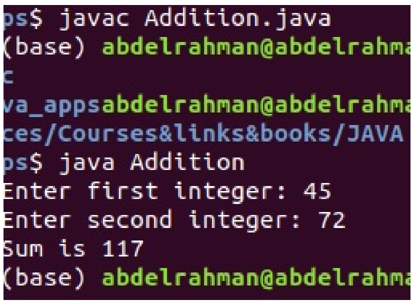


```
va_apps$ javac Welcome4.java
(base) abdelrahman@abdelrahman:
va_apps$ java Welcome4
Welcome to
Java Programming!
(base) abdelrahman@abdelrahman:
va_apps$
```

Another Application: Adding Integer

Use Scanner object to read to integer compute the summation then display the result.

// Addition.java // Addition program that inputs two numbers and displays their sum. import java.util.Scanner; public class Addition { // Main method ebgins execution of the program public static void main(String[] args) { // Create scanner to obtain input from the command window Scanner input = new Scanner(System.in); int number1; // first number to add int number2; // second number to add int sum = 0; // sum of number1 and number2 System.out.print("Enter first integer: "); // Prompt the user number1 = input.nextInt(); // read first number System.out.print("Enter second integer: "); // Prompt the user number2 = input.nextInt(); // read second number sum = number1 + number2; // add numbers, then stor total in sum System.out.printf("Sum is %d%n", sum); } // End of main method } // End of class Addition



```
ps$ javac Addition.java
(base) abdelrahman@abdelrahma
c
va_appsabdelrahman@abdelrahma
ces/Courses&links&books/JAVA
ps$ java Addition
Enter first integer: 45
Enter second integer: 72
Sum is 117
(base) abdelrahman@abdelrahma
```

Import declaration

import java.util.Scanner;

Do not **"reinvent the wheel"**, A great strength of OOP, and Java is one of those OOP languages, when it comes to writing some classes or other ones write some classes or Java developer they selfes provide you with predefined classes that you can use, in this case, no need to build your own class, its available for you, and for that, we import the Scanner class from library java which inside the package util, the package contain some of the classes related to each other.

These classes library are called **Java API** (Java Application Programming Interface).

Note !

All these import declarations should declare before your the class that you write or use.

Declaring Class

Talked about above

Declare and Create a Scanner

Scanner input = new Scanner(System.in);

A variable is a location in computer memory where value is stored in for the user to use later in the program.

Each variable should be have **type** and **name**.

- Type, which value should be this variable store, is it number or text, or file and others?
- name, the thing that you deal with when you tend to use this variable. (like access the value it hold).

In this case of Scanner, the **type** is Scanner, the **name** is input, and for that, we use input.nextInt(), which refers to scanner object but what we deal with now is the variable name to refer to this Scanner.

The word **new** is to create a new Scanner with **name** input, which will be used to read characters from the user.

The **System.in** enable the application to read bytes of data, then the **Scanner** object translate these bytes to corresponding **type** of the variable you need to read in, like in case of **integer type**

Declare Variables to store integer

```
int number1;
int number2;
int sum = 0;
```

These variables store the value of type integer which like [-1, 0, 1] not [-1.4, 5.3], which another type of **primitive type**, which will be talked about later.

Choose meaningful variables name help your program be self-documented.

When your name of the variable is more than one word to be meaningful, a convenient way is to separate each word by _ for example **sum two integers**.

Prompting the User for input

System.out.print("Enter first integer: ");

A good form of code when you tend the user to input some values which your program tends to use is to hint the user and prompt him/ her, to enter data with the type that should be taken from.

Obtaining an int as input from the user

number1 = input.nextInt();

nextInt() the method that used from the **input object** from the Scanner class, is to obtain an integer from the user to enter and wait for the user to enter the value, in case of user input another **type** than integer it will case **Run Time Error**.

The **=** sign is called **Assignment Operator** which work with **Two Operand**, the **nextInt()**, and the **number1**, and how it works from right to left, take the value first from the user, then save this values to right operand which in this case is **number1**.

The = is also called from those binary operator like [+, -, /, %] and others.

Using Variables in a calculation.

int sum = 0;

First, when you do not tend to read the value from the user in the variable, it's very useful to initialize these variables when you create like **int sum = 0**, unlike the **int number1;** and **int number1;**, which also be useful to initialize also with 0 and when you tend to read in it will remove the old value and assign the new value.

sum = number1 + number2;

As we talked about **binary operator** the **+** which is happening between **two operands** which are **number1** and **number2** then as we talked the **=** operator which also happened between two operands, which are the total of these number after summation happened and from right to left to store in the **sum** variable.

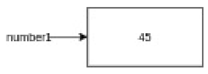
Displaying result using printf

Talked about above

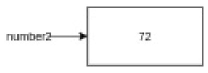
- %d, to display integer value.

Memory Concepts

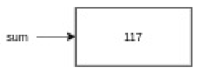
When a user first reads **number1**, it seems to something like that in memory.



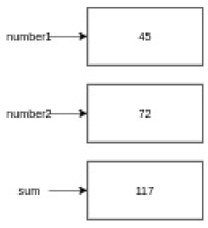
When a user second reads **number2**, it seems to something like that in memory.



When a user sum **number1** and **number2** in the variable **sum**, it seems to something like that in memory.



The memory now looks like



Arithmetic

Most Arithmetic operation done between **Two Operands** and called binary operators like:

- **Addition**, +
- **Subtraction**, -
- **Division**, /
- **Multiplication**, *
- **Remainder**, %

All these operators are written in calculation in one line like not like math, and also some of them are different than the math like **Multiplication operation**

Example of not like math:

10

—

2

In Java it **10/2** on same line.

Integer Division

17 / 5 = 3 because you have integer number divided by integer number, since of like these case we use **casting** , like (double) 17 / 3, and because we have cast one integer, java promote the other one which is 3 to also double because integer is smaller than double in bytes, and this happened because we can not divide two different types.

Precedence of operator

*, /, % is Evaluated first

+, - is Evaluated second

= is Evaluated last

Example :

y = a*x*x + b*x + c let a=2, x=5, b=3, c=7

---

y = 2 *5* 5 + 3 * 5 + 7 (leftmost multiplication)

2 * 5 is 10, then

y = 10 *5 + 3* 5 + 7 (leftmost multiplication)

10 * 5 is 50, then

y = 50 + 3 * 5 + 7 (multiplication before addition as we mentioned above who should evaluate first)

3*5 is 15, then

y = 50 + 15 +7 (leftmost addition)

50 + 15 is 65

y = 65 + 7 (leftmost addition)

65 + 7 is 72

y = 72 (last step is to place 72 in the varible y which take a place in memory)

Decision Making: Equality and Relational Operator

The **Condition** is some thing that we can evalute as true or false, like when I ask you **have you bring breakfast**, you answer **yes** or **no**, in this case you decide what you will make based on the answer.

Like this question or others like is your weight greater than 65 you also answer yes or no, from that we have the **Relation Operator** associated with your Decision Making, and instead of this setence just we ask you in Java like **weight > 65**.

So in java to ask something or select something, we use **if** in this manner.

if(weight > 65) { Do something }

if(weight == 65) { Do something }

if you will write after the condition true just one thing, no need to braces, because java take first statement after the condition if there is no braces, otherwise use braces, and for simplicity use braces in all your condiation.

So we have **Equality Operator**

==, !=, Equal Equal to evaluate something and != is the otherwise, one = is called assignment operator.

And we have **Relational Operator**

- y > x, y is grater than x
- y >= x, y is grater than or equal x
- y < x, y is less than x
- y <= x, y is less than or equal x

// Comparison.java // Compare integers using if statement, and relational operators // and equality operator import java.util.Scanner; public class Comparison { // Main Method public static void main(String [] args) { // Create the Scanner object Scanner input = new Scanner(System.in); int number1; int number2; System.out.print("Enter First integer: "); number1 = input.nextInt(); System.out.print("Enter Second integer: "); number2 = input.nextInt(); if(number1 == number2) System.out.printf("%d == %d%n", number1, number2); if(number1 != number2) System.out.printf("%d != %d%n", number1, number2); if(number1 > number2) System.out.printf("%d > %d%n", number1, number2); if(number1 >= number2) System.out.printf("%d >= %d%n", number1, number2); if(number1 < number2) System.out.printf("%d < %d%n", number1, number2); if(number1 <= number2) System.out.printf("%d <= %d%n", number1, number2); } // End of Main method } // End of Comparison Class

Logical error and empty space

if(number1 == number2); System.out.printf("%d == %d%n", number1, number2);

if(number1 == number2) ; System.out.printf("%d == %d%n", number1, number2);

In all cases java will print number1 == number2, because of semicolon.

- frist because the **;** is the end of statement so if is end its work by ;
- second because if execute one statement which in this case empty statement with just **;**

Now

*, /, % is Evaluated first

+, - is Evaluated second

<, <=, >, >= is Evaluated third

==, != is Evaluated forth

= is Evaluated last

---------------------------------------تم بحمد الله   -----------------------------------