









# Lesson 1

# Programming Fundamentals Using Java









**Absolute Java**

**Walter Savitch**




# Topics

-  **Introduction to Java**
-  **Structure of a Java Program**
-  **System.out.println()**
-  **Arithmetic Operators**
-  **Data Types, Variables**
-  **Assignment Statement**
-  **Simple Java Programs**
-  **Comments**




# Topics (cont'd.)

-  **Implicit Conversion**
-  **Explicit Conversion**
-  **Precedence of Arithmetic Operators**
-  **Compiling a Java Program**
-  **Executing a Java Program**
-  **String Class**
-  **Escape Sequences**
-  **Variable naming conventions**

# Java is an object-oriented Programming (OOP) language

-  The world around us is made up of objects, such as people, automobiles, buildings, streets, adding machines, papers, and so forth.
-  Each of these objects has the ability to perform certain actions, and each of these actions has some effect on some of the other objects in the world.
-  OOP is a programming methodology that views a program as similarly consisting of objects that interact with each other by means of action




# Java is an object-oriented Programming (OOP) language

-  The objects are called objects.
-  The actions that an object can take are called methods.
-  Objects of the same kind are said to have the same *type* or, more often, are said to be in the same class.

# Java is an object-oriented Programming (OOP) language

- For example, in an airport simulation program, all the simulated airplanes might belong to the same class, probably called the **Airplane** class.
- All objects within a class have the same methods, such as taking off, flying to a specific location, landing, and so forth. However, all simulated airplanes are not identical.
- They can have different characteristics, which are indicated in the program by associating different data (that is, some different information) with each particular airplane object. For example, the data associated with an airplane object might be two numbers for its speed and altitude.

# Two kinds of Java programs

-  Java applets and applications are two kinds of common Java programs.
-  An application, or application program, is just a regular program . They run on your computer .
-  Whereas, an applet is meant to be run from a Web browser or applet viewer, and so can be sent to another location on the Internet and run there.

# Example of Java program

```
import java.util.Scanner; //Gets the Scanner class from package(library)
                          // java.util
```

```
public class Example
```

```
{
```

```
    public static void main(String[ ] args)
```

```
{
```

```
    int a,b;
```

```
    Scanner input=new Scanner(System.in); //Object for taking input
```

```
    System.out.println("Enter a number:");
```

```
    a=input.nextInt(); //Reads whole number from keyboard
```

```
    System.out.println("Enter a number:");
```

```
    b=input.nextInt(); //Reads whole number from keyboard
```

```
    System.out.println("sum=" + (a+b));
```

```
}
```

```
}
```



# Sample Output:

Enter a number:

10

Enter a number:

5

sum=15

# Variable

**Variable:** It is a location in the computer memory which can store data and is given a symbolic name for easy reference. It's value can change during program execution.

The syntax for declaring a variable:

**<data type> <variable name> [=<initial value>], ... ;**

**Example:**

```
int a,b=10,c;
```

**Note: A variable should be declared before its usage.**

# Data types

The primitive data types available in Java are:

<b>Data Type</b>	<b>Kind of value</b>	<b>Memory Used</b>	<b>Range of Values</b>
<b>byte</b>	Integer	1 byte	-128 to 127
<b>short</b>	Integer	2 bytes	-32768 to 32767
<b>int</b>	Integer	4 bytes	-2,147,483,648 to 2,147,483,647
<b>long</b>	Integer	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

# Data types cont.

Data Type	Kind of value	Memory Used	Range of Values
<b>float</b>	Floating Point	4 bytes	$\pm 3.4 \times 10^{+38}$ to $\pm 1.4 \times 10^{-45}$
<b>double</b>	Floating Point	8 bytes	$+1.7 \times 10^{+308}$ to $\pm 4.9 \times 10^{-324}$
<b>char</b>	Single Character (Unicode)	2 bytes	All Unicode values from 0 to 65,355
<b>boolean</b>		1 bit	true or false

# Keyword/Reserved word

- A keyword is a reserved word that has a predefined meaning and purpose in the language.
- It cannot be used as an identifier by the user in his program. Eg. **public**.
- Some predefined words, such as **System** and **println**, are not keywords.
- These predefined words are not part of the core Java language and you are allowed to redefine them. Although these words are not keywords, they are defined in libraries required by the Java language standard.

# Output using println()

- 🍌 System.out.println statement is used to display a value on the screen.
- 🍌 It takes a line feed after displaying the value.
- 🍌 Java programs work by having things called objects perform actions. The actions performed by an object are called methods.
- 🍌 **System.out** is an **object** used for sending output to the screen; **println** is the **method** (that is, the action) that this object performs.

# Output using println()



## Syntax:

```
System.out.println(value/variable/expression);
```

## Example:

```
System.out.println("Enter a number:");
```

# Input using System.in

-  We use **class Scanner**, which Java supplies, to accept keyboard input.
-  We need to import the definition of the Scanner class from the package `java.util` by using the following statement:  
**`import java.util.Scanner;`**



# Input using System.in

An object of Scanner class has to be created using the statement:





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

For assigning the input value to a variable , use the following syntax:



```
variable =input.nextInt();
```

**//nextInt() is for reading an integer value.**

# Assignment Statement

-  The equal sign, causes the value on right to be assigned to the variable on the left.
-  The statement which assign values to variables are called assignment statements.
-  **Syntax:**  
Variable=value/variable/expression;
-  **Example:**  
a=10;


# operator + as Concatenation operator

-  plus sign is an operator to concatenate (connect) two strings.
-  If one of the two operands to + is a string, Java will convert the other operand, to a string.

## Example

```
System.out.println("sum=" + (a+b));
```

# Assignment Compatibilities

 You can assign a value of any type on the following list to a variable of any type that appears further down on the list:

byte → short → int → long → float → double

# Errors

There are three types of errors encountered in any programming language:

- 🍌 **Syntax Errors:** The errors which are traced by the compiler during compilation, due to wrong grammar for the language used in the program, are called syntax errors.

**For example:** `System.out.println(a) // Missing Semicolon`

- 🍌 **Run-Time Errors:** The errors encountered during execution of the program, due to unexpected calculation or input or output, are called run-time errors.

**For Example:** `a=n/0; //Division by zero`

- 🍌 **Logical Errors:** These errors are encountered when the program does not give the desired output, due to wrong logic of the program.

**For Example:** `remainder= b+c; // Wrong operator is used`

# Arithmetic Operators

Operator	Usage
+	Used for addition
-	Used for subtraction
*	Used for multiplication
/	Used for division
%	This operator is called the <b><i>remainder</i></b> or the <b><i>modulus operator</i></b> . It is used to find the remainder when one integer value/variable is divided by another value/variable.


# Example

```
int a=10,b=3;
```

```
System.out.println("Sum=" + (a+b));
```

```
System.out.println("Difference=" + (a-b));
```

# Parentheses and Precedence Rules

 If you want to specify exactly what subexpressions are combined with each operator, you can fully parenthesize an expression. When adding parentheses, Java follows precedence rules that determine how the operators, such as + and \*, are enclosed in parentheses. These precedence rules are similar to rules used in algebra.



# Precedence Rules in Arithmetic Operators

**Highest Precedence → Lowest Precedence**


**Unary Operators( +, -, !, ++, and --)**

**→ Multiplication/Division( \*,/,%)**

**→ Addition/Subtraction (+,-)**

**The expression is evaluated from left to right in case of operators with same precedence.**

# Automatic Type Conversion / Implicit Conversion

 When two operands of different data types are encountered in the same expression, the variable of lower data type is automatically converted to the data type of variable with higher data type, and then the expression is calculated.



# Automatic Type Conversion / Implicit Conversion

## For Example:

```
int a=98;
```

```
float b=5;
```

```
System.out.println(a/3.0);
```

//converts a temporarily to float type, since 3.0 is of float type

```
System.out.println(a/b);
```

//converts a temporarily to float type, since b is of float type,


//and gives the result 19.6



Of all the types used in the expression , it is the one that appears rightmost in the following list:

byte → short → int → long → float → double

# Type Casting / Explicit Conversion

 Type casting refers to the data type conversions specified by the programmer, as opposed to the automatic type conversions. This can be done when the compiler does not do the conversions automatically. Type casting can be done to higher or lower data type.

## Example

```
System.out.println((float)12/5);
```

// displays 2.4 , since 12 is converted to float type

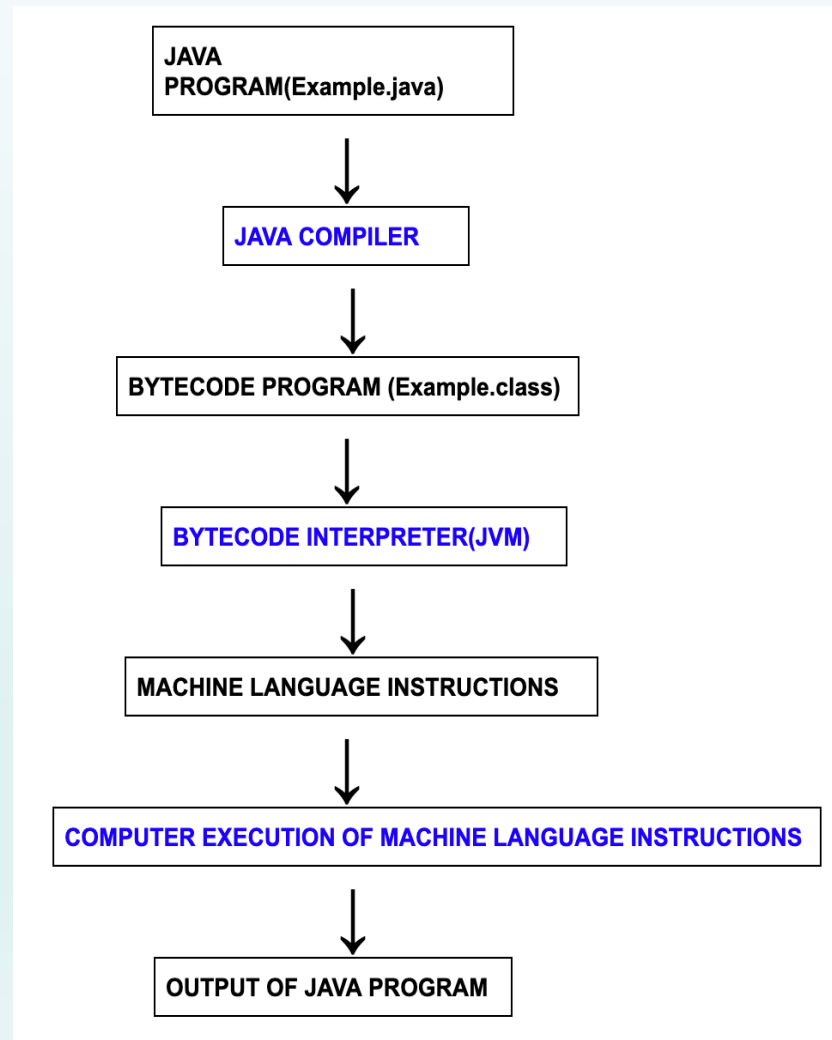
# Java Bytecode

- The Java compiler does not translate the program into the machine language for your particular computer.
- It translates into bytecode.
- Bytecode is not the machine language for any particular computer. Instead it is a machine language for a hypothetical computer known as virtual machine.
- Translating a program written in bytecode into a machine language program for an actual computer is quite easy.
- The program that does this translation is a kind of interpreter called the **Java Virtual Machine or JVM**. The JVM translates and runs the bytecode.



## 2 Steps for running a java program

- a. **Compile** - Converts the java program into bytecode.
- b. **Run** - The bytecode is converted into machine language instructions of the computer and executed.

# Running a java program



# IDE for Java

-  We can use any IDE (Integrated development environment) like BlueJ, Eclipse, JGrasp and NetBeans for writing, compiling and running a Java program.
-  Java program can also be written using simple text editors like Notepad (in windows) and TextEdit(in Mac). To then compile the java program we can use the free Java system distributed by Sun Microsystems for Windows, Linux, and Solaris.



# Class Loader

- 🍌 A Java program is divided into smaller parts called classes, and normally each class definition is in a separate file and is compiled separately.
- 🍌 In order to run your program, the byte-code for these various classes needs to be connected together.
- 🍌 The connecting is done by a program known as the class loader. It is typically done automatically, so you normally need not be concerned with it.

# Compiling a Java Program or Class

- Before you can compile a Java program, each class definition used in the program (and written by you, the programmer) should be in a separate file. Moreover, the name of the file should be the same as the name of the class, except that the file name has `.java` added to the end.
- If you are using an IDE (Integrated Development Environment), there will be a simple command to compile your Java program from the editor.

# Compiling a Java Program or Class



- 🟡 To compile your Java program or class with a one-line command for the Java system distributed by Oracle (usually called “the SDK” or “the JDK”) use the following syntax :

```
javac  example.java
```

**Note** - Example can be replaced by any program name

- 🟡 You should be in the same directory (folder) as the file **example.java** when you give this **javac** command.

# Compiling a Java Program or Class

-  When you compile a Java class, the resulting byte-code for that class is placed in a file of the same name, except that the ending is changed from **.java** to **.class**.
-  So, when you compile a class named **example** in the file **example.java**, the resulting byte-code is stored in a file named **example.class**.

# Running a Java Program

- If you are using an IDE, you will have a menu command that can be used to run a Java program. When you run a Java application program, only run the class that contains a main method.
- To run a Java program with a one-line command given to the operating system, then (in most cases) you can run a Java program by giving the command `java` followed by the name of the class containing the main method. For example, for the program `example`, you would give the following one-line command:  
**`java example`**
- Note that there is no extension used. We only use the name of the class.

# String Class

- 🍌 Strings of characters, such as "Enter the amount:", are treated slightly differently than values of the primitive types.
- 🍌 Java has no primitive type for strings.
- 🍌 However, Java supplies a **class** called **String** that can be used to create and process strings of characters.
- 🍌 The **java.lang.String** class provides a lot of methods to work on string.
- 🍌 The class String is a predefined class that is automatically made available to you when you are programming in Java.




# String Class cont.

- 🍌 Objects of type String are strings of characters that are written within double quotes.
- 🍌 By the help of methods in String class, you can perform operations on string such as trimming, concatenating, converting, comparing, replacing strings etc.
- 🍌 The following declares greeting to be the name for a String variable:

String greeting;


String greeting = "Hello!";

# String Class cont.

-  A string can have any number of characters. For example, "Hello" has five characters.
-  A string can even have zero characters. Such a string is called the **empty string** and is written as a pair of adjacent double quotes, like so: "".
-  You can use + to join, or concatenate, strings together.




# String Methods

 **Length():** returns the number of characters in a String object.


Example:

```
String greeting = "Hello";  
System.out.println(greeting.length( ));
```

**//Displays 5**

 Positions in a string begin with 0, not with 1. In the string "Hi Mom", 'H' is in position 0, 'i' is in position 1, the blank character is in position 2, and so forth.

# int indexOf(s)

 **int indexOf(s):** returns the index of a substring **s** given as its argument. It returns -1, if the substring is not present.

## **Example:**

```
String phrase = "Java is fun.";
```

```
System.out.println(phrase.indexOf("fun"));
```

```
//Displays 8
```

# int indexOf(A\_String, Start)


- 🟡 **int indexOf(A\_String, Start):** Returns the index (position) of the first occurrence of the string **A\_String** in the calling object string that occurs at or after position **Start**. Positions are counted 0, 1, 2, etc. Returns -1 if **A\_String** is not found.

## Example:

String name = "Mary, Mary quite contrary";  
name.indexOf("Mary", 1) returns 6.

- 🟡 The same value is returned if 1 is replaced by any number up to and including 6.
- 🟡 name.indexOf("Mary", 0) returns 0.
- 🟡 name.indexOf("Mary", 8) returns -1.
- 🟡 **Example Code: Example6.java**


# int lastIndexOf(A\_String)

 **int lastIndexOf(A\_String):** Returns the index (position) of the last occurrence of the string A\_String in the calling object string. Positions are counted 0, 1, 2, etc. Returns -1, if A\_String is not found.

## Example

String name = "Mary, Mary, Mary quite so";

 greeting.indexOf("Mary") returns 0, and

 name.lastIndexOf("Mary") returns 12.

# toUpperCase() and toLowerCase()

- **toUpperCase() and toLowerCase():** The java string toUpperCase() method converts this string into uppercase letter and string toLowerCase() method into lowercase letter.

## Example:

```
String s="San Francisco";
```

```
System.out.println(s.toUpperCase()); //SAN FRANCISCO
```

```
System.out.println(s.toLowerCase()); //san francisco
```

```
System.out.println(s); //San Francisco(no change in original)
```


## Output:

```
SAN FRANCISCO
```

```
san francisco
```

```
San Francisco
```

# trim()

 **trim():** The string trim() method eliminates white spaces before and after string.

## Example:

```
String s=" Hello World ";
```

```
System.out.println(s); // Hello World
```

```
System.out.println(s.trim()); //Hello World
```

## Output:

Hello World

Hello World

# startsWith() and endsWith()

 **startsWith() and endsWith():** It returns true if the string starts or ends with specified string.

## Example:

```
String s="Sachin";
```

```
System.out.println(s.startsWith("Sa")); //true
```


```
System.out.println(s.endsWith("n")); //true
```

## Output:

```
true
```

```
true
```

# charAt()

 **charAt():** The string charAt() method returns a character at specified index.

## Example:


```
String s="Sachin";
```

```
System.out.println(s.charAt(0)); //S
```

```
System.out.println(s.charAt(3)); //h
```



# valueOf()

 **valueOf():** The string valueOf() method converts given type such as int, long, float, double, boolean, char and char array into string.

## **Example:**

```
int a=10;
```

```
String s=String.valueOf(a);
```

```
System.out.println(s+10);
```

Output:

```
1010
```

# replace()

- **replace():** The string replace() method replaces all occurrence of first sequence of character with second sequence of character.

## Example:

```
String s1="Java is a programming language. Java is a platform.  
Java is an Island.";
```

```
String replaceString=s1.replace("Java","Kava");
```

```
//replaces all occurrences of "Java" to "Kava"
```

```
System.out.println(replaceString);
```

## Output

Kava is a programming language. Kava is a platform. Kava is an Island.

# substring(Start)

- 🟡 **substring(Start):** Returns the substring of the calling object string starting from **Start** through to the **end** of the calling object. Positions are counted 0, 1, 2, etc. Be sure to notice that the character at position Start is included in the value returned.

## Example:

String sample = "AbcdefG";

System.out.println(sample.substring(2));

## Output

cdefG

# substring(Start, End)

- **substring(Start, End):** Returns the substring of the calling object string starting from position **Start** through, but not including, position **End** of the calling object. Positions are counted 0, 1, 2, etc. Be sure to notice that **the character at position Start is included in the value returned, but the character at position End is not included.**

## Example:


```
String sample = "AbcdefG";
```


```
System.out.println(sample.substring(2, 5));
```

## Output

```
cde
```

# equals()

 **equals():** It The boolean expression `s1.equals(s2)` returns true if the strings `s1` and `s2` have equal values, and returns false otherwise. Notice that the two expressions

 `s1.equals(s2)`  
`s2.equals(s1)`

 are equivalent.

# equalsIgnoreCase()


- 🍌 **equalsIgnoreCase():** This method behaves like equals, except that equalsIgnoreCase considers the uppercase and lowercase versions of the same letter to be the same.
- 🍌 For example, "Hello" and "hello" are not equal because their first characters, 'H' and 'h', are different characters. But the method equalsIgnoreCase would consider them equal.
- 🍌 The following will display Equal:  

```
if("Hello".equalsIgnoreCase("hello"))  
    System.out.println("Equal");
```

# compareTo()


- **compareTo():** The method compareTo tests two strings to determine their **lexicographic order**. In lexicographic ordering, the letters and other characters are ordered according to their Unicode sequence.
- If s1 and s2 are two variables of type String that have been given String values, the method call  
s1.compareTo(s2): compares the lexicographic ordering of the two strings and returns
  - A negative number if s1 comes before s2
  - Zero if the two strings are equal
  - A positive number if s1 comes after s2
- **Example Code: Example7.java**

# char charAt(Position)

 **char charAt(Position):** Returns the character in the calling object string at the Position. Positions are counted 0, 1, 2, etc.

## Example

String greeting = "Hello!";

 greeting.charAt(0) returns 'H', and

 greeting.charAt(1) returns 'e'.



# Naming Constants

- Named constants can be declared in java using the syntax below:
- The values of constants defined this way, cannot be modified in the program.
- Constants must be placed outside of the main method.


**public static final Type variable=Constant;**

**Example:**

```
public static final int BRANCH_COUNT = 10;
```

```
public static final float PI = 3.14;
```

# Escape Characters/Sequences

 A backslash, \, preceding a character tells the compiler that the character following the \ does not have its usual meaning. Such a sequence is called an escape sequence or an escape character. The sequence is typed in as two characters with no space between the symbols. Several escape sequences are defined in Java.



## Escape Characters:

# Escape Characters/Sequences

- Several escape sequences are defined in Java.

```
\ " Double quote.  
\ ' Single quote.  
\ \ Backslash.  
\ n New line. Go to the beginning of the next line.  
\ r Carriage return. Go to the beginning of the current line.  
\ t Tab. Add whitespace up to the next tab stop.
```



- For example, suppose you want to display the following on the screen:

**The word "Java" names a language, not just a drink!**

Use the following code snippet:

```
System.out.println("The word \"Java\" names a language, "  
                        + "not just a drink!");
```





# Java Spelling Conventions

-  In Java, as in all programming languages, identifiers for variables, methods, and other items should always be meaningful names that are suggestive of the identifiers' meanings.
-  Although it is not required by the Java language, the common practice of Java programmers is to start the names of classes with uppercase letters and to start the names of variables, objects, and methods with lowercase letters.

# Naming Conventions for Variables

- Variable names are case-sensitive.
- A variable's name can be any legal identifier — an unlimited-length sequence of Unicode letters and digits, beginning with a letter, the dollar sign "\$", or the underscore character "\_".
- Subsequent characters may be letters, digits, dollar signs, or underscore characters.
- If the name you choose consists of only one word, spell that word in all lowercase letters. If it consists of more than one word, capitalize the first letter of each subsequent word.

# Comments

-  Documentation can be added to programs using two kinds of comments:
-  **Single Line Comments:** They start with `//` (two slashes) and end with the end of line
-  **Multi-Line Comments:** They are enclosed between `/*....*/`.
-  Comments are ignored by the compiler.