Object Oriented Programming – Java 1

Lecture 2

Basic Elements of Java Programming

Dr. Obuhuma James

Description

This topic kickstarts a series of lectures on the Java programming language that will be used to demonstrate object-oriented programming concepts. In this topic, an introduction to Java programming is covered. Part of the aspects covered include the basic structure of a Java program, data types, variables, constants, expressions, and operators, among other foundational elements of computer programming.

Learning Outcomes

By the end of this topic, you will be able to:

- Discuss the general concept of object-oriented programming.
- Describe the syntax of a Java program with respect to object-orientation.
- Write and execute Java programs that perform basic foundational programming tasks.

Overview of Object-Oriented Programming

A computer program is a set of instructions that tell a computer what to do [1]. The program is written in a specific computer programming language where these languages can be broadly categorized as machine languages, low-level languages, and high-level languages, depending on the level at which they are used. Machine languages operate at the lowest level possible, whereby they control the digital circuitry in binary format. Above this level, is assembly language, which is a low-level programming language expressed in form of instructions. At the very top are high-level programming languages that are expressed in human-friendly formats, closer to spoken languages [2], and mostly platform independent. This course will be based on the Java programming language, that is one of the commonly used high-level programming languages.

As earlier covered in the previous lecture, Java is an object-oriented programming language that is founded on classes and objects. Just like other object-oriented languages, Java is frequently used for creation of computer simulation applications and graphical user interfaces (GUIs) [1]. Some other factors that make object-oriented programming languages different from languages in other paradigms is the support for inheritance, polymorphism and encapsulation. This will be covered and demonstrated using the Java programming in subsequent topics in this course.

In object-oriented programming, a class describes objects with related properties. It is composed of a class name, attributes/properties, and methods signifying operations to be performed on it. Thus, a class must have a definition and an instance. All Java programs must be made up of at least one class. Objects are specific, concrete instances of a class used to represent real-world things. Classes are represented using compartmented rectangles, where the three compartments represent the class name, its attributes/properties, and operations. This is as shown in Figure 1, while Figure 2 shows an example representation of a class called Student.

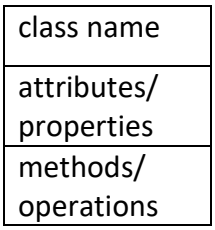

Figure 1. Structure of a Class

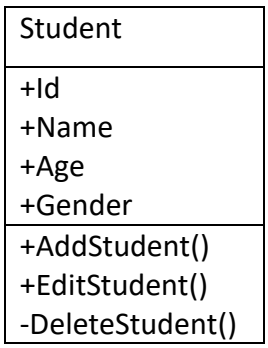

Figure 2. Example of a Class Representation

Each attribute and method of a class is mapped to an access specifier that defines the circumstances under which it can be accessed and used. This can either be public, private, or protected. This is signified by the plus (+), minus (-), and hash (#) signs respectively, where

- Public makes the field or method usable in any other class beyond the class that it resides in [1].
- Private access provides the highest level of security such that no other classes can access the field's values, and only methods of the same class are allowed to set, get, or otherwise use private variables [1]. Most methods are public, not private. Private data/public method arrangement provides a means for you to control outside access to data [1].
- Protected provides an intermediate level of security between public and private access [1]. A protected data field or method can be used within their own class or in any classes extended from that class but cannot be used by outside classes.

Similarly, class definitions fall under the either of the three options. These are formally referred to as access specifiers/modifiers.

The Java Programming Language

Java is a general-purpose object-oriented programming language developed by Sun Microsystems. It is a secure, architecture neutral language. Java does not execute instructions directly on the computer, it instead runs on a hypothetical computer known as a Java Virtual Machine (JVM) [1]. The main Java program types are applets and applications. Applications in this case include stand-alone programs, console applications, and windowed applications.

This course will use console applications for demonstration of concepts. The following is the main syntax of a Java program.

```
package packagename;
        //libraries to be imported in the program
public class classname{

        //additional methods placed here

        //main method
        public static void main(String[] args){
                //statements to be executed
        }

        //additional methods placed here

}
```

The package defines a container that can hold one or more classes for the same application. Every Java program code must be enclosed within a class with the class having a name that should be similar to the name of the Java program file. The mandatory requirement of enclosing the code in a class is one of the outstanding features of object-oriented programming. Within a class, one or more methods are defined. Each method tries to take care of an operation/task or set of operations/tasks being performed by the program. One critical requirement is for a given program to have one method called main(). The rest of the methods can be given any name that conforms to identifier naming conventions.

Display Output on the Console

The first step in learning a programming language is to be able to create, save, compile and run a program that displays the statement “Hello World” on the console. Thus, the source code showed in Figure 3 creates a program with class name Hello, with only the main() method and a statement that allows display of the phrase “Hello World”.

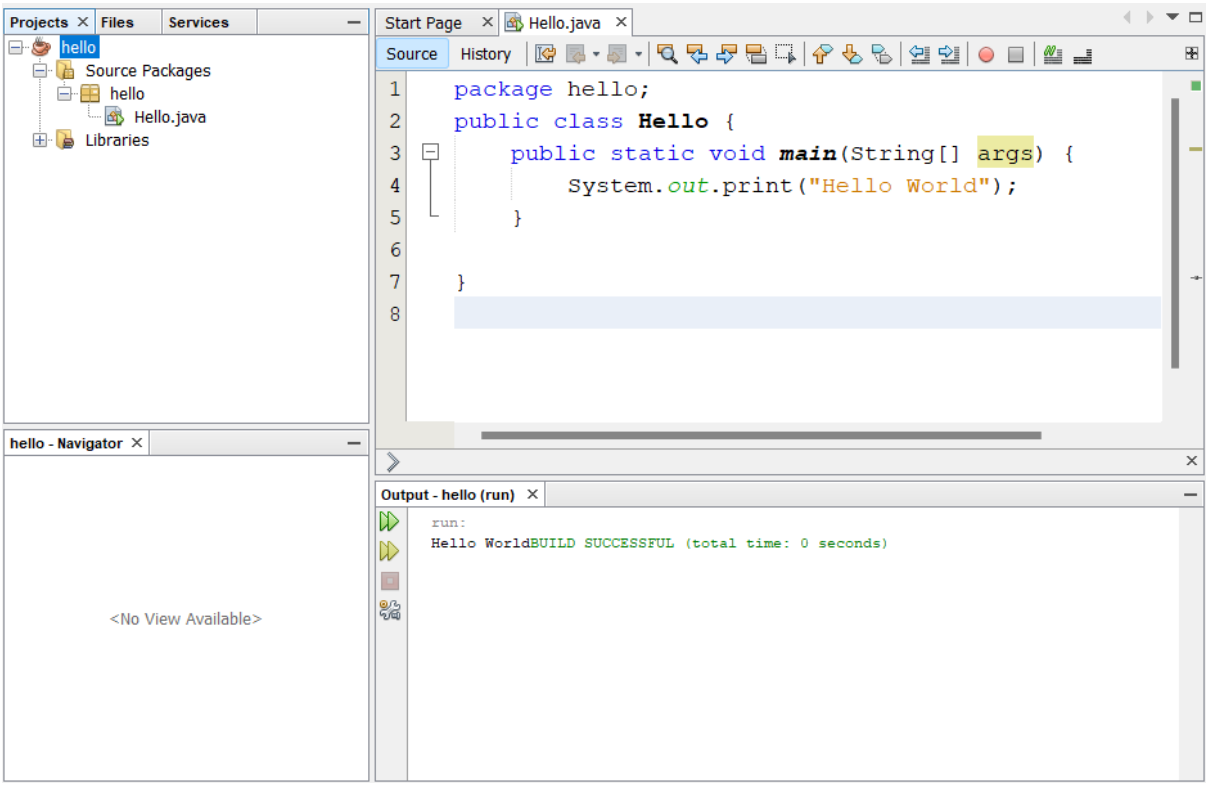

Figure 3. Hello World Program

The program should be saved with a name similar to the class name with a “.java” extension i.e. in this case, the name will be “Hello.java” as shown on the left menu in Figure 1. The class name and file name are case sensitive. Interpretation of the code is as follows:

- public static void main(String[] args) – represents the definition of the method header, where,
  - public – is an access specifier
  - static – is a reserved keyword that means that the method in question is accessible and usable even without objects of the class.
  - void – is the return type, in this case it means the method returns nothing.
  - main() – represents the name of the method.

- System.out.print("Hello World") – represents the statement that facilitates display of the phrase "Hello World" on the console, as an output, where,
    - System.out – is a standard output object [2].
    - print() – is a method in the System.out class whose role is to write on the console. The println() method may be used in cases where the cursor is expected to move to the next line after writing the statement on the console.
    - Hello World – represents a literal string that is printed as is. This is the case since that phrase has been put in double quotation marks.

Addition of any other lines of code using the System.out.print() option will result in the display of the desired content on the console.

Variables

Variables are discrete information-holding units that can be associated with specific locations in computer memory [1]. They serve as containers for data in the computer memory. Variables are a central concept to all high-level programming languages.

Defining Variable

Defining a variable requires a variable name to be determined first, which in return acts as an identifier for that variable. The following are the rules and conventions for naming a variable in Java [2], most of which are similar to other programming languages:

1. A Java identifier consists of letters, digits, the underscore character (_), and the dollar sign ($).
2. The identifier must begin with a letter, underscore, or the dollar sign.

In addition to a variable name, the set of allowable data for the variable must be known. This is referred to as the data type. A data type defines a specific category of information that a variable can contain. They define the kind of values that can be assigned to variables or to constants. According to [2], a data type is a set of values together with a set of operations that can be performed on the values. Usage of data types in variable and constant declaration varies from one programming language to the other. Hence, one of the categorization of programming languages is based on how data types are incorporated in variable declaration, such that:

1. Strongly Typed Programming Languages – require data types to be determined upfront and included in the declaration of variables [1, 2]. This category is sometimes referred to as static typing since the data type for any given variable does not change after declaration. Examples of programming languages under this category include Java, C, C++, among others.
2. Loosely Typed Programming Languages – do not require upfront determination and inclusion of data types in declaration of variables. This category is sometimes referred to as dynamic typing since the data type for any given variable can change after declaration. Examples of programming languages under this category include PHP, JavaScript, among others.

It is worth noting that the amount of storage location to be reserved for a variable in a computer varies based on the data type assigned to the variable. Thus, for loosely typed languages, the compiler automatically determines the data type based on the data supplied to the variable. Similarly, operations to be performed on any given variable is determined by its data type. Hence, whether the data type is explicitly or implicitly defined, the amount of storage and type of operations vary from one data type to the other. Java supports several primitive data types. These are as outlined in Table 1 [1].

Table 1. Characteristics of Primitive Data Types [1]

Data Type	**Description**
byte	Byte-length integer
short	Short integer
int	Integer
long	Long integer
float	Single-precision floating point
double	Double-precision floating point
char	Single character
boolean	A true or false Boolean value

Having known the various data types in Java, there are two main approaches that could be used to declare variables in Java:

1. Declaration of variables without initialisation

Syntax

datatype variablename;

Example

String name; //declares a variable called name of type String
int age; //declares a variable called age of type integer

2. Declaration of variables together with initialization of values to the variables

Syntax

datatype variablename = value;

Example

String name = "John Doe"; //declares a variable called name of type String and assigns John Doe to it.
int age = 10; //declares a variable called age of type integer and assigns value 10 to it.

In cases where variables are declared without initialization, as outlined in option 1, the initialization or assigning of values to the variable happens later in the program.

Using Variables

Once a variable has been declared, it can now be used in any section(s) of the program coming after it. This means that a variable cannot be referenced before it has been declared. Part of this usage could include incorporation in expressions, or in System.out.print() statements, among other uses.

The following example demonstrates how variable values could be displayed on the console.

int age = 10; //declares a variable called age then initializes it to 10.
System.out.print(age); //display variable contents on the console.

Notice the fact that to display contents of a variable requires passing the variable name to the System.out.print() object and method. Be sure not to enclose the variable name in quotation marks. Note that, enclosing a variable name in quotation marks makes it to become a literal string as opposed to being a variable.

Java just like other programming languages also permits concatenation of strings, variables, or both at any given point in the program code. In this case, a plus (+) sign is used to implement concatenation.

Syntax

"literal string" + "literal string";
variablename + "literal string";

Example

```
int age = 10;    //declares a variable called age then initializes it to 10.
System.out.print("Your age is " + age);    //concatenate a literal string "Your age is " with a
                                            variable called age for display on the console.
```

In this example, the output will be *"Your age is 10"* such that upon execution of the code, variable age will be replaced by its content, 10, which will be concatenated with the literal string "Your age is".

Constants

In programming, a constant is a variable whose value does not change during program execution [2]. For instance, π, the radius of the earth, the speed of light, among others. Just like for the case of variables, constants must be declared before they are used. Declaration of constants is however different from that of variables, such that, the final keyword precedes the data type. Constants should be initialized at the point of declaration [2].

Syntax

final datatype constantname = value;

Example

final float pi = 3.142; //declares a constant called PI then initializes it to 3.142.

Using Constants

Once a constant has been declared, it can be used in the code that follows afterwards in a similar manner that a variable is used.

Expressions and Operators

Variables and data become more useful when used in an expression. An expression is a literal value or variable that be evaluated by the Java compiler to produce a result. Expressions are

created using operands and operators, where operands are the values to be worked on while operators are symbols that signify the operation to be performed on the operands. Thus, operators are a critical part of expressions. They can be broadly grouped into two types, namely, binary, or unary. Binary operators are signified by the presence of an operand before and after an operator while unary operators require only a single operand before or after the operator.

Operators can be further grouped into four main categories, namely, arithmetic, assignment, comparison, and logical operators. The categorization depends on the nature of operations performed by various operators forming the category as follows:

1. Arithmetic operators – perform mathematical operations outlined in Table 2.
2. Assignment operators – map or assign values to variables as outlined in Table 3.
3. Comparison operators – perform comparison of operands and returns a Boolean value as outlined in Table 4.
4. Logical operators – perform Boolean operations on Boolean operands as outlined in Table 5.

Table 2. Binary Arithmetic Operators

Operation	**Symbol**	**Description**
Addition	+	Adds values for two operands
Subtraction	-	Subtracts the value of the right operand from that in the left operand
Multiplication	*	Multiplies values for two operands
Division	/	Divides the value of the left operand by that of the right operand
Modulus	%	Returns the remainder of dividing the value of the left operand by that of the right operand

Table 3. Unary Arithmetic Operators

Operation	**Symbol**	**Description**
Increment	++	Increments the value of an operand by 1
Decrement	--	Decrements the value of an operand by 1

Table 4. Assignment and Compound Assignment Operators

Operation	Symbol	Description
Assignment	=	Assigns the value on the right-hand side of the operator to the variable on the left-hand side
Compound addition assignment	+=	Adds the value on the right-hand side to that on the left-hand side of the operator then assigns the new values to the variable on the left-hand side
Compound subtraction assignment	-=	Subtracts the value on the right-hand side from that on the left-hand side of the operator then assigns the new values to the variable on the left-hand side
Compound multiplication assignment	*=	Multiplies the value on the right-hand side with that on the left-hand side of the operator then assigns the new values to the variable on the left-hand side
Compound division assignment	/=	Divides the value on the left-hand side by that on the right-hand side of the operator then assigns the new values to the variable on the left-hand side
Compound modulus assignment	%=	Divides the value on the left-hand side by that on the right-hand side of the operator then assigns the remainder to the variable on the left-hand side

Table 5. Comparison Operators

Operation	Symbol	Description
Equal	==	Returns TRUE if the values on the left-hand side are equal to those on the right-hand side
Strict equal	===	Returns TRUE if the values on the left-hand side are equal and of the same data type to those on the right-hand side
Not equal	!=	Returns TRUE if the values on the left-hand side are not equal to those on the right-hand side
Strict not equal	!==	Returns TRUE if the values on the left-hand side are not equal or not of the same data type to those on the right-hand side
Greater than	>	Returns TRUE if the values on the left-hand side are greater than those on the right-hand side

Less than	<	Returns TRUE if the values on the left-hand side are less than those on the right-hand side
Greater than or equal to	>=	Returns TRUE if the values on the left-hand side are greater than or equal to those on the right-hand side
Less than or equal to	<=	Returns TRUE if the values on the left-hand side are less than or equal to those on the right-hand side

Table 6. Logical Operators

Operation	**Symbol**	**Description**
Logical AND	&&	Returns TRUE if both outcomes for the left-hand side and the right-hand side are TRUE, otherwise it returns FALSE.
Logical OR	||	Returns TRUE if either outcome for the left-hand side or the right-hand side is TRUE, otherwise it returns FALSE.
Logical Exclusive Or	^	Returns TRUE if only one outcome for either the left-hand side or the right-hand side is TRUE, otherwise it returns FALSE.
Logical NOT	!	Returns TRUE if an expression is FALSE and returns FALSE if an expression is TRUE.

Keyboard Input

Java uses the Scanner library class to facilitate reading of keyboard input. The System.in class is used at this point. The following steps are involved

1. Import the Scanner class using “import java.util.Scanner”
2. Create of an object of the Scanner class using the System.in class

   Syntax

   Scanner objectname = new Scanner(System.in);
3. Use the object of the Scanner class with special methods to read keyboard data.

   Syntax

   objectname.methodname();

The methods in this case determine the expected type of data to be received from the keyboard. These are

a) next() and nextLine() – for Strings data types
b) nextInt() – for integers data types
c) nextFloat() – for float data types
d) nextDouble() – for double data types

Example

Consider the program in Figure 2 that prompts a user to input his/her name and age. The program then displays the name and age on the console.

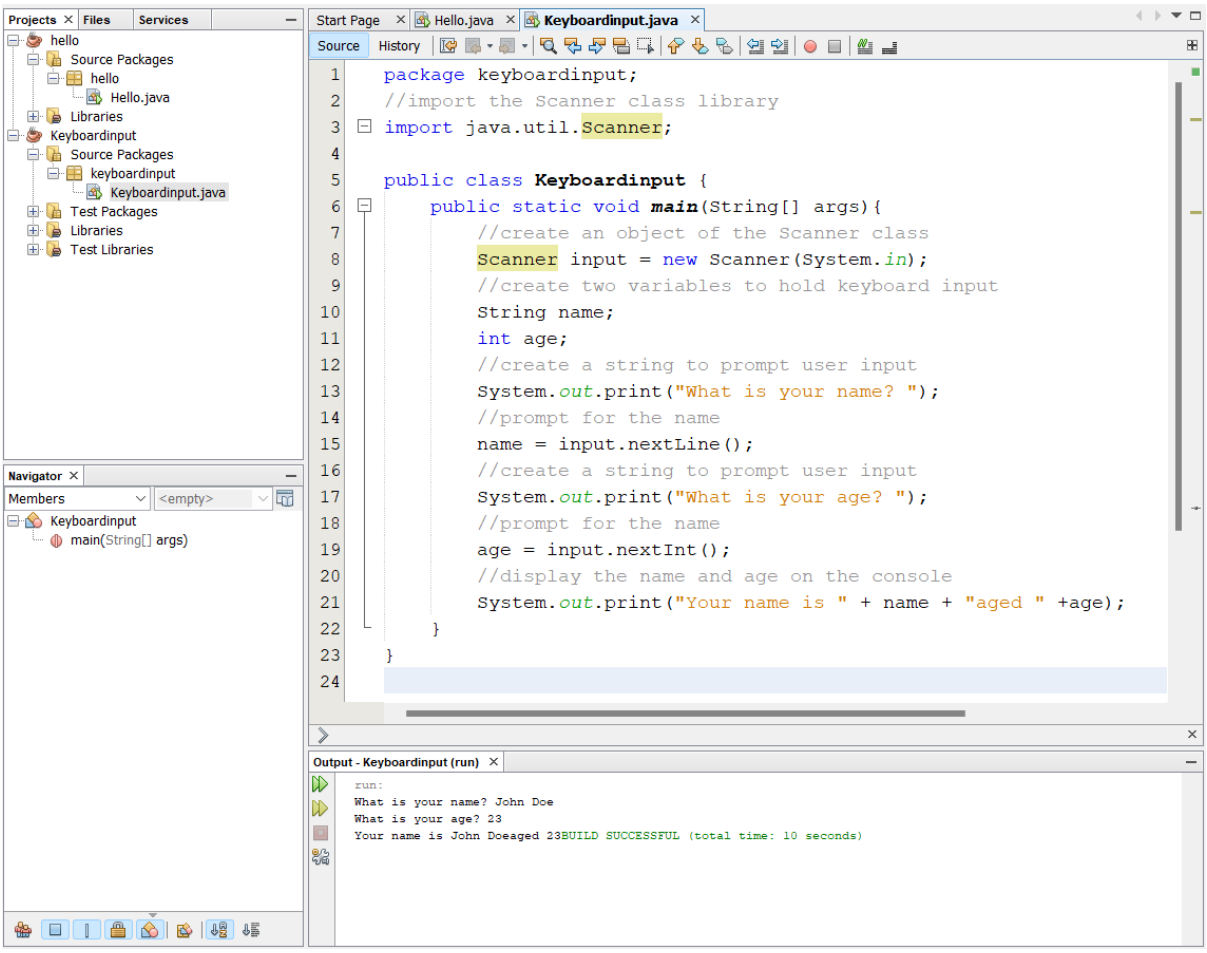

Figure 4. Keyboard Input Example

The output shows that the program stops to prompt for the name of the user and the age. The program proceeds to display the name and age keyed in by the user, in this case John Doe and 23.

Summary

The topic has explored the first set of basic elements of Java programming. These includes definition and usage of variables and constants; the different data types used in Java; the various types of expressions and operators; and finally, the process of facilitating keyboard input through the Scanner class. It is clearly evident that the concept of definition and usage of variables, constants, data types, expressions and operators is more or less similar to what happens in other programming languages, especially strongly types programming languages. However, keyboard input in Java uses a completely different approach compared to other programming languages. Java involves slightly longer steps to accomplish the process compared to other languages.

Check Points

1. Describe the three types of access specifiers as used in Java.
2. Using any example, differentiate the use of print() and println() methods.
3. Using an example, differentiate between variables and constants as applied in Java.
4. Discuss the different types of operators as used in Java.
5. Describe the different primitive data types used in Java.
6. Using an example, describe the process of reading and displaying keyboard inputs in Java.

Core Textbooks

1. Joyce Farrell, Java Programming, 7th Edition. Course Technology, Cengage Learning, 2014, ISBN-13 978-1-285-08195-3.
2. Malik, Davender S. JavaTM Programming: From Problem Analysis to Program Design, International Edition, 5th Edition, Cengage Learning.

Other Resources

3. Sebester, R. W., Concepts of Programming Languages, 12th Edition, Pearson, 2018, ISBN 0-321-49362-1.
4. Daniel Liang, Y. "Introduction to Java Programming, Comprehensive." (2011).
5. Malik, Davender S. JavaTM Programming: From Problem Analysis to Program Design, International Edition, 4th Edition, Cengage Learning, 2011.
6. Shelly, Gary B., et al. Java programming: comprehensive concepts and techniques. Cengage Learning, 2012.

References

[1] Farrell, J., Java Programming, 7th Edition. Course Technology, Cengage Learning, 2014, ISBN-13 978-1-285-08195-3.

[2] Malik, D. S., JavaTM Programming: From Problem Analysis to Program Design, International Edition, 5th Edition, Cengage Learning.

[3] Sebester, R. W., Concepts of Programming Languages, 12th Edition, Pearson, 2018, ISBN 0-321-49362-1.