

**FEDERAL UNIVERSITY OF TECHNOLOG, MINNA,  
NIGER STATE, NIGERIA**



**CENTRE FOR OPEN DISTANCE AND  
e-LEARNING (CODEL)**

**B.TECH. COMPUTER SCIENCE  
PROGRAMME**

**COURSE TITLE  
OBJECT ORIENTED PROGRAMMING II**

**COURSE CODE  
CPT 221**

**COURSE CODE**

**CPT 221**

**COURSE UNIT**

**3**

**Course Coordinator**

Bashir Mohammed (Ph.D.)  
Department of Computer Science  
Federal University of Technology (FUT) Minna  
Minna, Niger State, Nigeria.

## Course Development Team

---

### CPT 221: Object Oriented Programming II

Subject Matter Experts	Enesi Femi AMINU Department of Computer Science FUT Minna, Nigeria.
Course Coordinator	Bashir Mohammed (Ph.D.) Department of Computer Science FUT Minna, Niger State, Nigeria.
Instructional System Designers	Oluwole Caleb FALODE (Ph.D.) Bushrah Temitope OJOYE (Mrs.) Centre for Open Distance & e-Learning, FUT Minna, Nigeria
ODL Experts	Amosa Isiaka GAMBARI (Ph.D.) Nicholas Ehikioya ESEZOBOR
Language Editors	Chinenye Priscilla UZOCHUKWU (Mrs.)
Centre Director	Abiodun Musa AIBINU (Ph.D.) Centre for Open Distance & e-Learning FUT Minna, Niger State, Nigeria.

### Introduction

**CPT 221 Object Oriented Programming II** is a 3- credit unit course for students studying towards acquiring a Bachelor of Science in any field and particularly, Information Technology. The course is divided into 4 modules and 20 study units. It will first introduce the fundamental of java programming language, which include language general syntax, variables and datatypes, strings and reserved words. Next, is classes, interface, enumeration and objects. This include constructor as a special method, overload and override a method and control access to members. Thereafter, java operators and control statements are discussed. The operators include arithmetic, relational and logical; while both decision and looping statements or constructs would be discussed as well. Finally, the students would be introduced to some java exceptional handling such as java throwable's inheritance hierarchy, catch exception with try/catch/finally blocks, using throws and throw keyword.

The course guide therefore gives you an overview of what CPT 221 is all about, the textbooks and other materials to be referenced, what you expect to know in each unit, and how to work through the course material.

### Recommended Study Time

This course is a 3-credit unit course having 20 study units. You are therefore enjoined to spend at least 3 hours in studying the content of each study unit.

### What You Will Learn in This Course

The overall aim of this course, CPT 221 is to introduce you to basic concepts of Java as object oriented programming. At the end of this course you would have learnt the:

- i. basic concept of object oriented programming
- ii. various building blocks and keywords of java programs
- iii. syntax and semantics of java programming language
- iv. logic of using Java in simple problem solving.

### Course Aims

This course aims to introduce students to understand the fundamentals of object oriented programming in general; and specifically, Java language. It is expected that the knowledge will enable the reader to effectively how to write or develop simple computer program.

### Course Objectives

It is important to note that each unit has specific objectives. Students should study them carefully before proceeding to subsequent units. Therefore, it may be useful to refer to these objectives in the course of your study of the unit to assess your progress. You should always look at the unit objectives after completing a unit. In this way, you can be sure that you have done what is required of you by the end of the unit.

However, below are overall objectives of this course. On completing this course, you should be able to:

- (i). explain object oriented programming languages

- (ii). Describe the characteristics of Java programming language
- (iii). Understand why Java is object oriented language
- (iv). Understand the general syntax of Java constructs
- (v). Identify Java's datatypes and keywords
- (vi). Understand how variables, classes are being declared
- (vii). Establish the links between objects and classes
- (viii). Describe the various operators in Java programs
- (ix). Understand the various constructs of selection statement
- (x). Understand the various constructs of looping statement
- (xi). Describe the basic difference between the constructs
- (xii). Understand some of the various types of Java Exception handling
- (xiii). Discuss how catch exception work with try
- (xiv). Write a simple program for problem solving

### Working Through This Course

In order to have a thorough understanding of the course units, you will need to read and understand the contents, practice the steps by designing and implementing a mini system for your department, and be committed to learning and implementing your knowledge.

This course is designed to cover approximately sixteen weeks, and it will require your devoted attention. You should do the exercises in the Tutor-Marked Assignments and submit to your tutors.

### Course Materials

The major components of the course are:

1. Course Guide
2. Study Units
3. Text Books
4. Assignment File
5. Presentation Schedule

### Study Units

There are Eighteen (18) Study Units and Five (5) Modules in this course. They are:

<b>Module One</b>	<b>Introduction to Object Oriented Programming (OOP)</b>	
	Unit 1	Basic Computer Program Concepts
	Unit 2	Overview of OOP; The OOP principles
	Unit 3	Introduction to Java Programming Language
	Unit 4	Java Environments
<b>Module Two</b>	<b>Java Fundamentals</b>	

	Unit 1	Introduction to Data in Programming; Data structure and Java DataTypes
	Unit 2	Identifiers; Reserved Words; Comments in Java
	Unit 3	Java Operators
	Unit 4	Understanding Classes, Objects and Methods
	Unit 5	Constructors; Interface and Enum
<b>Module Three</b>	<b>Control Statements: Java Selection Statements</b>	
	Unit 1	If statement
	Unit 2	If – else statement
	Unit 3	Switch statement
<b>Module Four</b>	<b>Control Statements: Java Looping Statements</b>	
	Unit 1	For loop
	Unit 2	While loop
	Unit 3	Do-while loop
	Unit 4	Java Break statement
<b>Module Five</b>	<b>Exception Handling</b>	
	Unit 1	Java throwable's inheritance hierarchy
	Unit 2	Try, catch, throw and throws in Java

### Recommended Texts

Kingsley Sage (2019) *Concise Guide to Object-Oriented Programming - An Accessible Approach Using Java*.

Edward Sciore (2019) *Java Program Design: Principles, Polymorphism, and Patterns*

Eckel, B. (2003). *Thinking in JAVA*. Prentice Hall Professional.

Baesens, B., Backiel, A., & Vanden Broucke, S. (2015). *Beginning Java programming: the object-oriented approach*. John Wiley & Sons.

Halim, S., Halim, F., Skiena, S. S., & Revilla, M. A. (2013). *Competitive programming 3*. Lulu Independent Publish.

Schildt, H., & Coward, D. (2014). *Java: the complete reference* (p. 1312). New York: McGraw-Hill Education.

Goncalves, A. (2009). *Beginning Java EE 6 Platform with GlassFish 3: from novice to professional*. Apress.



## Assignment File

The assignment file will be given to you in due course. In this file, you will find all the details of the work you must submit to your tutor for marking. The marks you obtain for these assignments will count towards the final mark for the course. Altogether, there are tutor marked assignments for this course.

## Presentation Schedule

The presentation schedule included in this course guide provides you with important dates for completion of each tutor marked assignment. You should therefore endeavor to meet the deadlines.

## Assessment

There are two aspects to the assessment of this course. First, there are tutor marked assignments; and second, the written examination. Therefore, you are expected to take note of the facts, information and problem solving gathered during the course. The tutor marked assignments must be submitted to your tutor for formal assessment, in accordance to the deadline given. The work submitted will count for 40% of your total course mark.

At the end of the course, you will need to sit for a final written examination. This examination will account for 60% of your total score. You will be required to submit some assignments by uploading them to CPT 111 page on the u-learn portal.

## Tutor Marked Assignments (TMAs)

There are TMAs in this course. You need to submit all the TMAs. The best 10 will therefore be counted. When you have completed each assignment, send them to your tutor as soon as possible and make certain that it gets to your tutor on or before the stipulated deadline. If for any reason you cannot complete your assignment on time, contact your tutor before the assignment is due to discuss the possibility of extension. Extension will not be granted after the deadline, unless on extraordinary cases.

## Final Examination and Grading

The final examination for CPT 111 will last for a period of 2 hours and has a value of 60% of the total course grade. The examination will consist of questions which reflect the self-assessment questions and tutor marked assignments that you have previously encountered. Furthermore, all areas of the course will be examined. It would be better to use the time between finishing the last unit and sitting for the examination, to revise the entire course. You might find it useful to review your TMAs and comment on them before the examination. The final examination covers information from all parts of the course.

## The following are Practical Strategies for Working through this Course

1. Read the course guide thoroughly
2. Organize a study schedule. Refer to the course overview for more details. Note the time you are expected to spend on each unit and how the assignment relates to the units. Important details, e.g. details of your tutorials and the date of the first day of the semester are available. You need to gather together all this information in one place such as a diary,



a wall chart calendar or an organizer. Whatever method you choose, you should decide on and write in your own dates for working on each unit.

3. Once you have created your own study schedule, do everything you can to stick to it. The major reason that students fail is that they get behind with their course works. If you get into difficulties with your schedule, please let your tutor know before it is too late for help.
4. Turn to Unit 1 and read the introduction and the objectives for the unit.
5. Assemble the study materials. Information about what you need for a unit is given in the table of content at the beginning of each unit. You will almost always need both the study unit you are working on and one of the materials recommended for further readings, on your desk at the same time.
6. Work through the unit, the content of the unit itself has been arranged to provide a sequence for you to follow. As you work through the unit, you will be encouraged to read from your set books
7. Keep in mind that you will learn a lot by doing all your assignments carefully. They have been designed to help you meet the objectives of the course and will help you pass the examination.
8. Review the objectives of each study unit to confirm that you have achieved them.
  - a. If you are not certain about any of the objectives, review the study material and consult your tutor.
9. When you are confident that you have achieved a unit's objectives, you can start on the next unit. Proceed unit by unit through the course and try to pace your study so that you can keep yourself on schedule.
10. When you have submitted an assignment to your tutor for marking, do not wait for its return before starting on the next unit. Keep to your schedule. When the assignment is returned, pay particular attention to your tutor's comments, both on the tutor marked assignment form and also written on the assignment. Consult you tutor as soon as possible if you have any questions or problems.
11. After completing the last unit, review the course and prepare yourself for the final examination. Check that you have achieved the unit objectives (listed at the beginning of each unit) and the course objectives (listed in this course guide).

## Tutors and Tutorials

There are few hours of tutorial provided in support of this course. You will be notified of the dates, time and location together with the name and phone number of your tutor as soon as you are allocated a tutorial group. Your tutor will mark and comment on your assignments, keep a close watch on your progress and on any difficulties you might encounter and provide assistance to you during the course. You must mail your tutor marked assignment to your tutor well before the due date. At least two working days are required for this purpose. They will be marked by your tutor and returned to you as soon as possible.

Do not hesitate to contact your tutor by telephone, e-mail or discussion board if you need help. The following might be circumstances in which you would find help necessary: contact your tutor if:

1. You do not understand any part of the study units or the assigned readings.
2. You have difficulty with the self test or exercise.
3. You have questions or problems with an assignment, with your tutor's comments on an assignment or with the grading of an assignment.

You should endeavour to attend the tutorials. This is the only opportunity to have face to face contact with your tutor and ask questions which are answered instantly. You can raise any problem encountered in the course of your study. To gain the maximum benefit from the course tutorials, have some questions handy before attending them. You will learn a lot from participating actively in discussions.

GOOD LUCK!

## Table of Content

---

<b>Course Development Team.....</b>	<b>iii</b>
<b>Study Guide.....</b>	<b>iv</b>
<b>Table of Content.....</b>	<b>ix</b>
<b>Module 1: Introduction to Object Oriented Programming (OOP).....</b>	<b>11</b>
Unit 1: Basic Computer Program Concepts.....	12
Unit 2: Overview of OOP; The OOP principles .....	17
Unit 3: Introduction to Java Programming Language .....	22
Unit 4: Java Environments .....	26
<b>Module 2: Java Fundamentals .....</b>	<b>32</b>
Unit 1: Introduction to Data in Programming; Data structure and Java DataTypes .....	33
Unit 2: Identifiers; Reserved Words; Comments in Java .....	37
Unit 3: Java Operators .....	41
Unit 4: Understanding Classes, Objects and Methods .....	52
Unit 5: Constructors; Interface and Enum .....	59
<b>Module 3: Control Statements: Java Selection Statements .....</b>	<b>64</b>
Unit 1: If statement .....	65
Unit 2: if else statement.....	69
Unit 3: switch statement.....	73
<b>Module 4: Control Statements: Java Looping Statements .....</b>	<b>77</b>
Unit 1: for loop.....	78
Unit 2: while loop.....	82
Unit 3: do while loop.....	86
Unit 4: java break statement .....	89
<b>Module 5: Exception Handling.....</b>	<b>91</b>
Unit 1: Java throwable's inheritance hierarchy .....	93
Unit 2: Try, catch, throw and throws in Java .....	98

# Module 1

---

## Introduction to Object Oriented Programming (OOP)

- Unit 1: Basic Computer Program Concepts
- Unit 2: Overview of OOP, The three OOP Principles
- Unit 3: Introduction to Java Programming Language & General Syntax; properties of Java
- Unit 4: Java Environments

# Unit 1

---

## Basic Concepts

### Contents

- 1.0 Introduction
- 2.0 Learning Outcomes
- 3.0 Learning Contents
  - 3.1 Definition of Computer Program Concepts
  - 3.2 Programming Languages
  - 3.3 Programming Paradigms
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment (TMA)
- 7.0 References/Further Reading

## 1.0 Introduction

---

With the advent of computer programming languages, human activities have become less laborious and time consuming. Before we discuss the concepts of Object Oriented Programming (OOP) as the newest paradigm of programming languages, there are some essential basic computer programs we must understand. For example, the different among the terms computer program, programming, programmer, programming languages and programming paradigms as these provide a special background to clearly understand the concept of OOP.

## 2.0 Learning Outcomes

---

At the end of studying this unit, you should be able to:

- i. Understand the basic computer program terms such as program, programming and programming languages.
- ii. Differentiate between the terms
- iii. Understand the different between the OOP and other programming paradigms

## 3.0 Learning Contents

---

### 3.1 Definition of Computer Program Concepts

---

**A computer program** is a set of instructions that you write to tell a computer what to do. Computer equipment, such as monitors or keyboards, are hardware, and programs are software. A program that manages the computer itself (such as Windows or Linux) is system software. A program that performs a task for a user (such as calculating and producing pay checks, word processing, or playing a game) are application software. The logic behind any computer program, whether it is an application or system program, determines the exact order of instructions needed to produce desired results. All computer programs ultimately are converted to machine language. Machine language, or machine code, is the most basic set of instructions that a computer can execute. Each type of processor has its own set of machine language instructions.

**Programming** is art of writing computer programs. And anyone who writes computer program is called **Programmer**. Computer programs are written with programming language. Today, there are hundreds of computer programming languages a programmer can employ such as Java programming language.

### Self-Assessment Exercise(s) 1

1. Define computer program
2. A program that performs a task for a user (such as calculating) is application software (True/False)
3. Who is a programmer?
4. Computer programs are written with programming language such as Java (True/False)

## 3.2 Programming Language

A programming language is a **computer language** that is used by **programmers (developers) to communicate with computers**. It is a set of instructions written in any specific language (for example, C, C++, Java, Python) to perform a specific task. Programming languages can be broadly classified as low level and high level languages.

Examples of low level languages are machine and assembly languages; while examples of high level languages include Java, Python, C#, C++, FORTRAN, Pascal, Ada, BASIC,

Fortunately, programming has evolved into an easier task because of the development of high-level programming languages. A high-level programming language allows you to use a vocabulary of ordinary terms, such as read, write, or add, instead of the sequences of 1s and 0s that perform these tasks, as in the case of low level language. Using a programming language, programmers write a series of program statements, similar to English sentences, to carry out the tasks they want the program to perform. Program statements are also known as commands because they are orders to the computer, such as “output this word” or “add these two numbers.”

### Self-Assessment Exercise(s) 2

1. Programming language is a set of ..... written in any specific language (Java or Python) to perform a specific task
2. Programming languages can be broadly classified as ..... and .....
3. Mention any three examples of high level languages?
4. Programming has evolved into an easier task because of the development of ..... programming languages
5. Program statements are also known as commands (True or False)

## 3.3 Programming Paradigms

The term programming paradigm refers to a **style of programming**. It does not refer to a specific language, but rather it refers to the way you program. There are lots of programming languages that are well-known but all of them need to follow some strategy when they are implemented. Examples of these paradigms are as follows:

### Imperative Programming Paradigm:

It is one of the oldest programming paradigm. It features close relation to machine architecture. It is based on Von Neumann architecture. It works by changing the program state through assignment statements.

### Procedural Programming Paradigm

This paradigm emphasizes on procedure in terms of underlying machine model. There is no difference in between procedural and imperative approach.

### Object Oriented Programming (OOP) Paradigm

OOP is the most popular programming paradigm because of its unique advantages like the modularity of the code and the ability to directly associate real-world business problems in terms of code. The key characteristics of object-oriented programming include Class, Abstraction, Encapsulation, Inheritance and Polymorphism. The advantages include Data security, Inheritance, Code reusability; and Flexible and abstraction is also present.

### Declarative Programming Paradigm

It is divided as Logic, Functional, Database. In computer science the *declarative programming* is a style of building programs that expresses logic of computation without talking about its control flow.

## 4.0 Conclusion

---

We have learnt so far the difference between computer program and programming language. Programming paradigm refers to a style of programming; and the newest and most popular paradigm is the OOP with lot of advantages.

## 5.0 Summary

---

You have learnt that:

- i. Computer program is a set of instruction; and programming is an art of writing computer programs.
- ii. Java along with other examples are high level programming language because it is English like; therefore, it is easier to learn and understand.
- iii. Programming paradigm is a style of programming and OOP is the most popular paradigm.
- iv. Java is a classic example of OOP

## 6.0 Tutor-Marked Assignment

---

1. What is computer program?
2. Programming languages can be broadly classified as ..... and .....



3. Mention any three examples of high level languages?
4. Programming has evolved into an easier task because of the development of ..... programming languages
5. Mention three advantages of OOP.

## 7.0 References/ Further Readings

---

Scott Sanderson (2016) *Java: Java Programming For Beginners - A Simple Start to Java Programming*.

Alvaro Felix (2016) *JAVA: Easy Java Programming for Beginners, Your Step-By-Step Guide to Learning Java Programming*

# Unit 2

---

## Overview of OOP; The OOP principles

### Contents

- 1.0 Introduction
- 2.0 Learning Outcomes
- 3.0 Learning Contents
  - 3.1 Overview of OOP
  - 3.2 Characteristics of OOP
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment (TMA)
- 7.0 References/Further Reading

### 1.0 Introduction

---

OOP stands for Object Oriented Programming. It is a style of writing computer programs, which consider every real life entity as object. The blueprint of object is referred to as class. This type of programming paradigm is often described as the newest style of programming.

### 2.0 Learning Outcomes

---

At the end of studying this unit, you should be able to:

- i. Define OOP
- ii. List the benefits of OOP
- iii. List the salient characteristics of OOP

## 3.0 Learning Contents

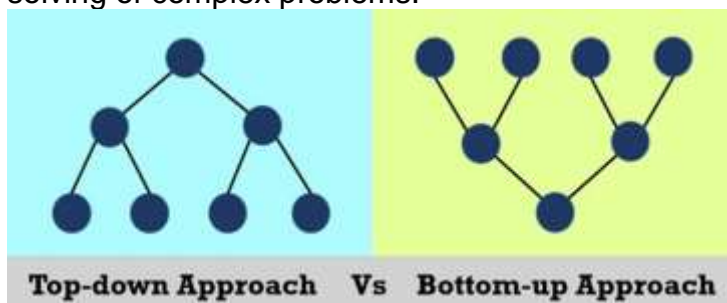
---

### 3.1 Overview of OOP

---

Two popular approaches to writing computer programs are procedural programming and object-oriented programming. Procedural programming is a style of programming in which operations are executed one after another in sequence. In procedural applications, you create names for computer memory locations that can hold values—for example, numbers and text—in electronic form. The named computer memory locations are called variables because they hold values that might vary. For example, a payroll program written for a company might contain a variable named salary. The memory location referenced by the name salary might contain different values (a different value for every employee of the company) at different times. During the execution of the payroll program, each value stored under the name salary might have many operations performed on it. A procedural program defines the variable memory locations and then calls a series of procedures to input, manipulate, and output the values stored in those locations. A single procedural program often contains hundreds of variables and procedure calls.

OOP stands for Object Oriented Programming. Object-oriented programming is an extension of procedural programming in which you take a slightly different approach to writing computer programs. OOP uses bottom up approach. Object-oriented software development is a paradigm that started in the 1980s. It evolved to ease the solving of complex problems.



OOPL: An object-oriented programming language (OOPL) is a high-level programming language based on the object-oriented model. To embark on object-oriented programming, one needs an object-oriented programming language. Many modern programming languages are object-oriented, however some older programming languages, such as Pascal, do offer object-oriented versions. Simula (developed by Kristen Nygaard and Ole-Johan) was the first object-oriented programming language. Others are Java, Python, C++, Visual Basic .NET, Ruby, C#, PHP, Perl, Object Pascal, Objective-C, Dart, Swift, Scala, Common Lisp, and Smalltalk. Benefits of OOPL over others include

1. Develop reusable software modules
2. Reduce production cost
3. Quicken the completion time for software development
4. Reduce maintenance cost

## 3.2 Characteristics of OOP

---

All object-oriented programming languages provide mechanisms that help you implement the object-oriented model. They are being described by some salient principles or characteristics. They are encapsulation, abstraction, inheritance, and polymorphism. Let's take a look at these concepts now.

### Encapsulation

*Encapsulation* is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse. One way to think about encapsulation is as a protective wrapper that prevents the code and data from being arbitrarily accessed by other code defined outside the wrapper. Access to the code and data inside the wrapper is tightly controlled through a well-defined interface.

In Java the basis of encapsulation is the class. A *class* defines the structure and behavior (data and code) that will be shared by a set of objects. Each object of a given class contains the structure and behavior defined by the class, as if it were stamped out by a mold in the shape of the class. For this reason, objects are sometimes referred to as *instances of a class*. Thus, a class is a logical construct; an object has physical reality.

When you create a class, you will specify the code and data that constitute that class. Collectively, these elements are called *members* of the class. Specifically, the data defined by the class are referred to as *member variables* or *instance variables*. The code that operates on that data is referred to as *member methods* or just *methods*.

### Inheritance

Inheritance is the ability of one object to acquire some/all properties of another object. For example, a child inherits the traits of his/her parents. With inheritance, reusability is a major advantage. You can reuse the fields and methods of the existing class. In Java, there are various types of inheritances: single, multiple, multilevel, hierarchical, and hybrid. For example, Apple is a fruit so assume that we have a class Fruit and a subclass of it named Apple. Our Apple acquires the properties of the Fruit class. Other classifications could be grape, pear, and mango, etc. Fruit defines a class of foods that are mature ovaries of a plant, fleshy, contains a large seed within or numerous tiny seeds. Apple the sub-class acquires these properties from Fruit and has some unique properties, which are different from other sub-classes of Fruit such as red, round, depression at the top.

### Polymorphism

*Polymorphism* (from the Greek, meaning "many forms") is a feature that allows one interface to be used for a general class of actions. The specific action is determined by the exact nature of the situation. Consider a stack (which is a last-in, first-out list). You might have a program that requires three types of stacks. One stack is used for integer values, one for floating-point values, and one for characters. The algorithm that implements each stack is the same, even though the data being stored differs. In a non-object-oriented language, you would be required to create three different sets of stack routines, with each set using different names. However, because of polymorphism, in Java you can specify a general set of stack routines that all share the same names.

## Abstraction

Abstraction is an extension of encapsulation. It is the process of selecting data from a larger pool to show only the relevant details to the object. Suppose you want to create a dating application and you are asked to collect all the information about your users. You might receive the following data from your user: Full name, address, phone number, favorite food, favorite movie, hobbies, tax information, social security number, credit score. This amount of data is great however not all of it is required to create a dating profile. You only need to select the information that is pertinent to your dating application from that pool. Data like Full name, favorite food, favorite movie, and hobbies make sense for a dating application. The process of fetching/removing/selecting the user information from a larger pool is referred to as Abstraction. One of the advantages of Abstraction is being able to apply the same information you used for the dating application to other applications with little or no modification.

### Self-Assessment Exercise(s) 1

1. What do you understand by the concept OOPL.
2. List four examples of OOP
3. Mention three benefits of OOPL over procedural programming paradigm
4. State the four major characteristics of OOPL
5. ----- property of OOP is described as an extension of encapsulation

## 4.0 Conclusion

---

OOP languages helps programmer to develop a reusable software module, reduces the production and maintenance costs. A programming language like java is said to be OOP because it supports the four major characteristics.

## 5.0 Summary

---

This unit has described the overview of OOP with some examples. Benefits of OOP were equally mentioned. Lastly, the four salient properties of OOP were also discussed.

## 6.0 Tutor-Marked Assignment

---

1. Briefly describe the four salient characteristics of OOP.
2. State three reasons why OOP is mostly preferred to other programming paradigms.
3. Briefly describe four programming paradigms

## 7.0 References/ Further Readings

---

Schildt, H., & Coward, D. (2014). *Java: the complete reference* (p. 1312). New York: McGraw-Hill Education.

Jose M. Garrido (2003) Object-Oriented Programming (From Problem Solving to JAVA) (Programming Series)

# Unit 3

---

## Introduction to Java Programming Language

### Contents

- 1.0 Introduction
- 2.0 Learning Outcomes
- 3.0 Learning Contents
  - 3.1 Introduction to Java Programming Language
  - 3.2 General Syntax of Java
  - 3.3 Properties of Java
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment (TMA)
- 7.0 References/Further Reading

### 1.0 Introduction

---

Java is a classic example of OOPL. Computer programs created by java are said to be robust. Java programming language is cross platform; that is, architectural independent. This implies the portability nature of java's program. The program can easily be executed across different platforms: windows, Linux or Unix.

## 2.0 Learning Outcomes

---

At the end of studying this unit, you should be able to:

- i. Describe the qualities of java programming language
- ii. Understand how java program is written
- iii. List the salient characteristics of Java

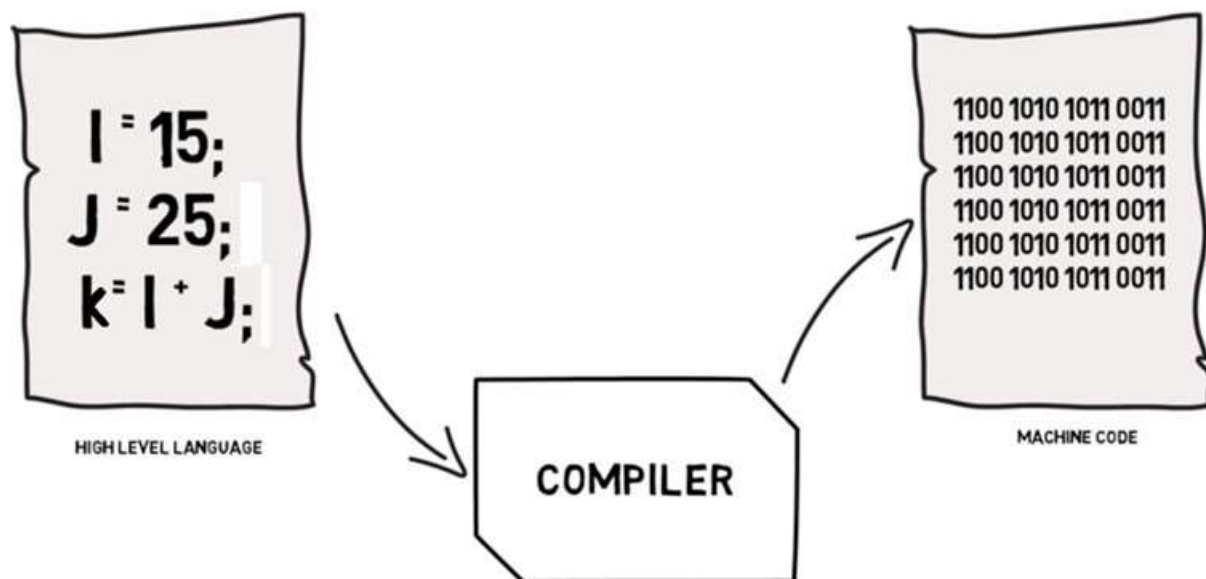
## 3.0 Learning Contents

---

### 3.1 Introduction to Java Programming Language

---

Java is a high-level, object-oriented programming language. It enables programmers to write computer instructions using English based commands, instead of having to write in numeric codes (0's and 1's). Java is popular among professional programmers because it can be used to build visually interesting graphical user interface (GUI) and Web-based applications.



Java also provides an excellent environment for the beginning programmer to quickly build useful programs while learning the basics of structured and object-oriented programming techniques. Java has a set of rules that determine how the instructions are written which is called syntax, rules that determine how the instructions are understood semantics (meaning/interpretation). Basically syntax errors are invalid code the compiler doesn't understand because the rules of writing the codes were not obeyed. On the other hand, semantic errors are valid code the compiler understands, however yield a different result other than what the programmer intends. An example is when you want to add 5 and 4, "5 + 4", you then write "5 - 4" which produces the 1 instead of 9. There is no way for the compiler to detect them.

James Gosling (a Canadian computer scientist), at Sun Labs, around 1992 Invented Java. Java is supported by many operating system, and this is what makes Java cross-platform OR platform independent. This means it programs written in Java can work on any operating system (Windows, Mac OS, or Linux) & device (PCs, phones, and tablet computers).

### 3.2 General Syntax of Java

---

Example of simple Java Programs: How java program is written

#### **First Example: Sum of two numbers**

```
public class AddTwoNumbers {  
  
    public static void main(String[] args) {  
  
        int num1 = 5, num2 = 15, sum;  
        sum = num1 + num2;  
        System.out.println("Sum of these numbers: "+sum);  
    }  
}
```

#### **Output:**

Sum of these numbers: 20

### 3.3 Properties of Java

---

Java possesses various properties. A few of those are:

#### **1 JAVA IS A ROBUST LANGUAGE**

Robust means reliable. Java programming language is developed in a way that puts a lot of emphasis on early checking for possible errors, that's why java compiler is able to detect errors that are not easy to detect in other programming languages. The main features of java that makes it robust are garbage collection, Exception Handling and memory allocation.

#### **2 JAVA IS SECURE**

A lot of security futures are built into java that makes it secure. That is why several security flaws are almost impossible to exploit in Java (examples of such security flaws are stack corruption or buffer overflow).

#### **3 MULTITHREADING**

Java supports multithreading. Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU.

#### **4 PORTABLE**



Java code that is written on one machine can run on another machine. For this reason it is called platform independent. The platform independent byte code can be carried to any platform for execution that makes java code portable and platform independent. With Java, one program version runs on all these platforms. “Write once, run anywhere” (WORA) is a slogan developed by Sun Microsystems to describe the ability of one Java program version to work correctly on multiple platforms.

### Self-Assessment Exercise(s) 1

1. Why Java is OOP
2. How Java program is translated into machine code
3. Identify how class is written in Java: that is, the syntax of class in java
4. Identify how default method is written in Java: that is, the syntax of main method
5. What are the properties of Java programming language?

## 4.0 Conclusion

---

Java programming language is a portable program; which implies it can be executed across different platforms. The default method of java is called main. It has a return datatype void with a keyword public as access specifier.

## 5.0 Summary

---

This unit has described the main characteristics of Java programming language along with the syntax.

## 6.0 Tutor-Marked Assignment

---

1. Follow the java syntax to write the default method of java.
2. Study the syntax of main method of java, what does the keyword *public* and *void* stand for?
3. Briefly explain why compiler is required in java programs
4. Briefly explain three properties of Java programming language.

## 7.0 References/ Further Readings

---

Halim, S., Halim, F., Skiena, S. S., & Revilla, M. A. (2013). *Competitive programming 3*. Lulu Independent Publish.

Schildt, H., & Coward, D. (2014). *Java: the complete reference* (p. 1312). New York: McGraw-Hill Education.

# Unit 4

---

## Java Environments

### Contents

- 1.0 Introduction
- 2.0 Learning Outcomes
- 3.0 Learning Contents
  - 3.1 Introduction to Java Environment
  - 3.2 Java Editions
  - 3.2 Java Code Execution
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment (TMA)
- 7.0 References/Further Reading

### 1.0 Introduction

---

Various environment exists for the successful development, execution and display of a Java program. They can somewhat be classified based on if you are a developer or just a client that needs to use a Java program. Besides, there are different java editions and the program can be developed by different editors.

### 2.0 Learning Outcomes

---

At the end of studying this unit, you should be able to:

- i. Understand various java environments.
- ii. Understand how java codes are executed.
- iii. Describe different editions of java.

## 3.0 Learning Contents

---

### 3.1 Introduction to Java environment

---

Various environment exists for the successful development, execution and display of a Java program. The following technologies form part of java environment.

**JDK:** Java Development Kit, The JDK is a software development environment used for developing Java applications and applets. It includes the Java Runtime Environment (JRE), an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (javadoc) and other tools needed in Java development.

**JRE:** Java Run-time Environment is an environment where Java applications only be ran and not developed (client system). It is part of the Java Development Kit (JDK), and provides the minimum requirements for executing a Java application; it consists of the Java Virtual Machine (JVM), core classes, and supporting files.

**JVM:** Java Virtual Machine, is the environment responsible for the actual execution of a Java program



In summary, whether you are Running (JRE) or developing (JDK) a java program, you defiantly need a JVM. That is why a JVM is contained in both a JDK & JRE

### 3.2 Java Editions

---

The following are the known editions of java programming language.

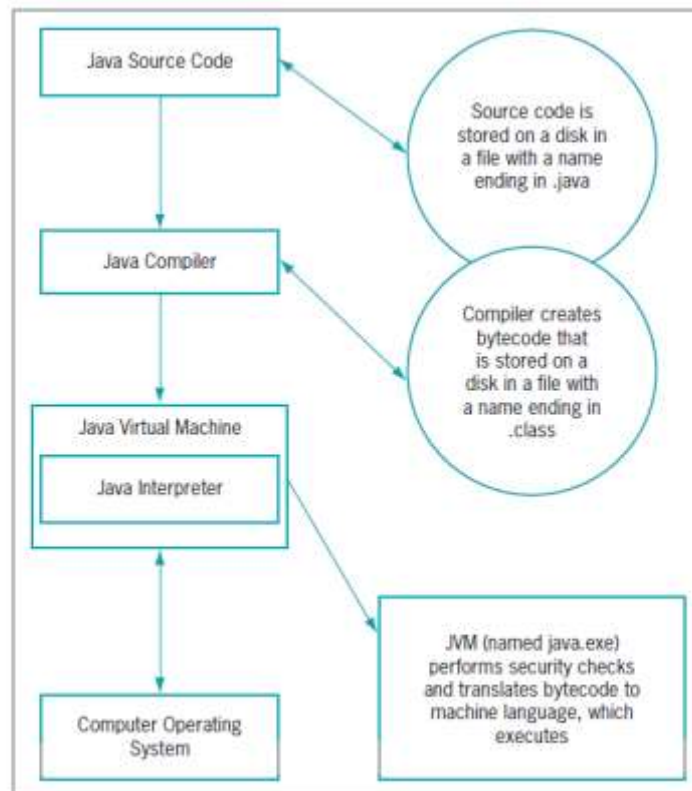
1. Java Platform, Standard Edition (Java SE) is the Java platform for developing client-side applications, which run on desktops, and applets, which run in web browsers
2. Java Platform, Enterprise Edition (Java EE) is the Java platform built on top of Java SE, which is used exclusively to develop enterprise-oriented server applications. Server-

side applications include servlets, which are Java programs that are similar to applets but run on a server rather than a client

3. Java Platform, Micro Edition (Java ME) is also built on top of Java SE. It is the Java platform for developing MIDlets, which are Java programs that run on mobile information devices, and Xlets, which are Java programs that run on embedded devices (eg ATMs, printers, calculators, thermostats .....)

### 3.2 Java Code Execution

Java can be run on a wide variety of computers and devices because it does not execute instructions on a computer directly. Instead, Java runs on a hypothetical computer known as the Java Virtual Machine (JVM). When programmers call the JVM hypothetical, they don't mean it doesn't exist. Instead, they mean it is not a physical entity created from hardware but is composed only of software.

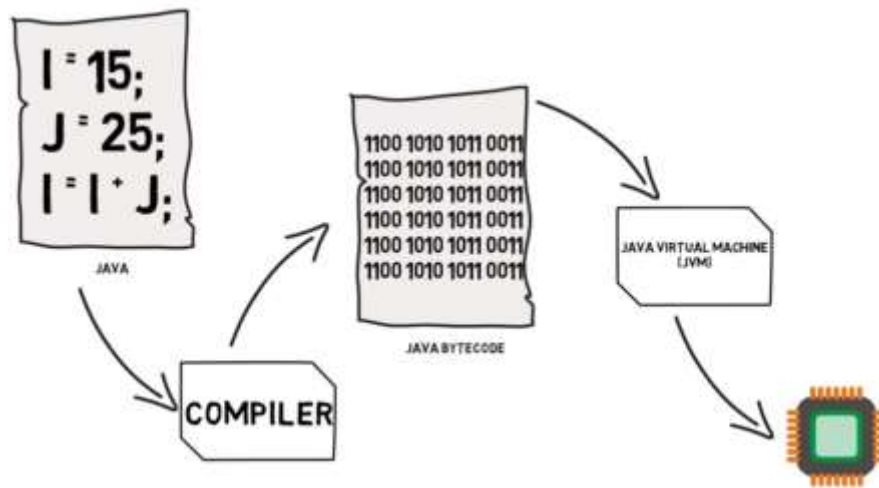


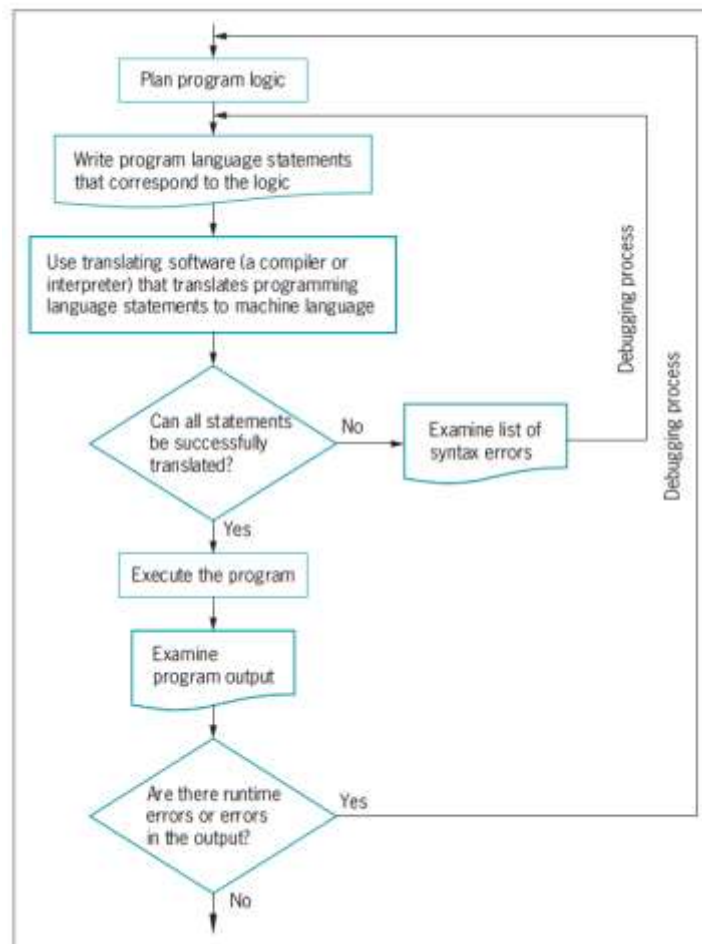
The figure above shows what writing and executing Java program entails. Programming statements written in a high-level programming language are source code. When you want to write and execute a Java program, this is what happens:

1. First construct the source code using a text editor such as Notepad (or a development environment and source code editor such as jGRASP, which you can download from the Web for free). A development environment is a set of tools that help programmers write programs by providing features such as displaying a language's keywords in colour. The statements are saved in a file.
2. Then, the Java compiler converts the source code into a binary program of byte code.

3. A program called the Java interpreter then checks the byte code and communicates with the operating system, executing the byte code instructions line by line within the Java Virtual Machine.

Because the Java program is isolated from the operating system, the Java program also is insulated from the particular hardware on which it is run. Because of this insulation, the JVM provides security against intruders accessing computer hardware through the operating system. Therefore, Java is more secure than other languages.





Java Development Kit, is an environment where Java applications can both be developed and ran (programmers PC)

### Self-Assessment Exercise(s) 1

1. How does java program produces java byte code.
2. At what point does java program requires compiler?
3. What are the available java editions?
4. What are the technologies that form java environment
5. What are the full meaning of JVM, Java EE and JDK

## 4.0 Conclusion

---

Java environment prepare the mind of intending developer the required technologies to be able to develop java program. Besides, developer can easily decide the type of java edition to use based on the type of program intends to create.

## 5.0 Summary

---

This unit has presented the various environments in java; how java program can be executed from source code to java byte code and finally machine code. The unit equally presents various java editions.

## 6.0 Tutor-Marked Assignment

---

1. Briefly describe the following java technologies: JDK, JRE and JVM
2. At what point in java code execution does java bytecode produced?
3. When is java platform micro edition can be used?

## 7.0 References/ Further Readings

---

Kishori Sharan (2017) Beginning Java 9 Fundamentals: Arrays, Objects, Modules, JShell, and Regular Expressions.

Jamie Chan (2016) Learn Java in One Day and Learn It Well

# Module 2

---

## Java Fundamentals

Unit 1:	Introduction to Data in Programming
Unit 2:	Understanding Identifiers; Reserved Words; Comments in Java
Unit3:	Java Operators
Unit4:	Understanding Classes, Objects and Methods
Unit5	Constructors; Interface and Enum



# Unit 1

---

## Introduction to Data in Programming

### Contents

- 1.0 Introduction
- 2.0 Learning Outcomes
- 3.0 Learning Contents
  - 3.1 Overview of Data in Programming
  - 3.2 Data structure
  - 3.3 Java Datatypes
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment (TMA)
- 7.0 References/Further Readings

## 1.0 Introduction

---

The major building blocks of every program include data and instructions. Therefore, the importance of data in writing program is very crucial. Consequently, this unit discusses the data structures in java, and how data are represented and defined with the aid of datatypes.

## 2.0 Learning outcomes

---

At the end of studying this unit, you should be able to:

- i. Understand the building blocks of computer program
- ii. Define data structure
- iii. Understand the different categories of datatypes

## 3.0 Learning Contents

---

### 3.1 Overview of Data in programming

---

As known by now, computers help in providing solutions to problems through the use of programs. A key ingredient for achieving this is the use of data. Programs by design manipulate data (input) in order to get an output (solution). In computer science and computer programming, data is seen and used in different forms. Thus there are different types of data or better put different classifications of data. Data type or simply type is a classification of data which tells the compiler or interpreter how the programmer intends to use the data. Most programming languages support various types of data. A data type provides a set of values from which an expression (i.e. variable, function...) may take its values. This data type defines the operations that can be done on the data, the meaning of the data, and the way values of that type can be stored.

For example, a program to help calculate the area of a circle  $A = \pi r^2$

$\pi = 3.142$  ( $\pi$  can never change, always the same, A CONSTANT)

$r$  = radius of the circle, which would vary, depending on the size of the circle (A VARIABLE)

Thus from this we see 2 forms of data, a constant and a variable.

A variable is a named memory location that can store a value. A variable can hold only one value at a time, but the value it holds can change. For example,  $r$  = radius of the circle, which would vary, depending on the Size of the circle whose radius you might be calculating at the time. Whether a data item is variable or constant, in Java it always has a data type.

### 3.2 Data Structures

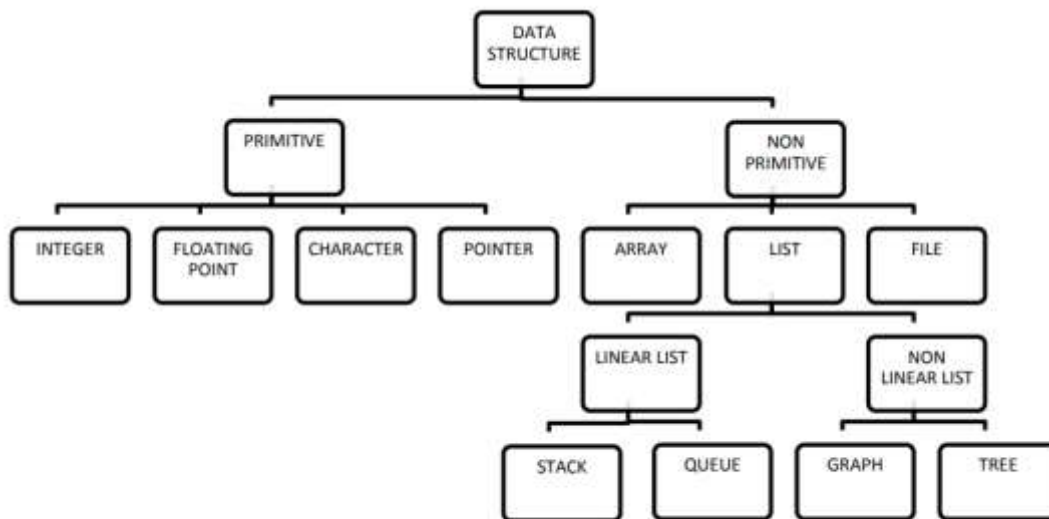
---

Data Structure is a way of organizing and storing data so that operations can be performed efficiently. Accessing, inserting, deleting, finding, and sorting the data are some of the basic operations that one can perform using data structures. Not all data

structures can perform these operations efficiently, that's what led to the development of different data structures. Let's say you are to find a specific book in an unorganized library, that task would take an enormous amount of time. Just like a library organizes their books, we need to organize our data so that operations can be performed efficiently. So does an item's data type describes the type of data that can be stored there, how much memory the item occupies, and what types of operations can be performed on the data.

### 3.3 Java Datatypes

Java comprises of data types, however there are two types of data types in Java, Primitive data types and Non-primitive data types.



## PRIMITIVE TYPES

Java provides for 8 primitive types of data shown below

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

1. The byte data type holds integers from  $-128$  to  $127$
2. The short data type holds integers from  $-32,768$  to  $32,767$
3. Integers data type holds integers from:

- 2,147,483,648 ( $-2^{31}$ ) to 2,147,483,647 ( $2^{31} - 1$ )
4. The long data type holds very large integers, from -9,223,372,036,854,775,808 ( $-2^{63}$ ) to 9,223,372,036,854,775,807
  5. Float data type holds values beyond the scope of this discussion
  6. Double data type holds values beyond the scope of this discussion
  7. A Boolean variable can hold only one of two values true or false
  8. Char data type holds values between 0 and 65,535, it represents symbols in a character set

## 4.0 Conclusion

---

Every user defined data used in java program has to be declared with the aid of datatype. Data with numeric values are expected to be declared with byte, short, long, int. these are for whole number values for example, age. While data such as height may be declared as either float or double. Char is for character, and Boolean is for data that either returns true or false.

## 5.0 Summary

---

You have learnt that:

- i. Data is a major building block of java program.
- ii. Data such as name whose value are string of characters takes string as datatype
- iii. Data such as Age, Number of children take byte, int, short or long as datatype
- iv. Data with alphanumeric values can as well take string as datatypes.

## 6.0 Tutor-Marked Assignment

---

1. What do you understand by data structure
2. State the relationship between data and computer program.
3. Briefly describe the eight primitive datatypes of java.
4. Which type of datatypes would you likely use to declare the following data:
  - a. School fees
  - b. Course of Study
  - c. Sex
  - d. Local Government Area, State of Origin and Nationality
  - e. X, y and z.

## 7.0 References/ Further Readings

---

Kishori Sharan (2017) Beginning Java 9 Fundamentals: Arrays, Objects, Modules, JShell, and Regular Expressions.

Jamie Chan (2016) Learn Java in One Day and Learn It Well

# Unit 2

---

## Understanding Identifiers; Reserved Words; Comments in Java

### Contents

- 1.0 Introduction
- 2.0 Learning Outcomes
- 3.0 Learning Contents
  - 3.1 Identifiers in Java
  - 3.2 Java Reserved Words
  - 3.3 Comments in Java
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment (TMA)
- 7.0 References/Further Readings

## 1.0 Introduction

---

Identifiers are part of the main component of any computer program. They are entities normally used to develop program such as variables. Variables are temporary memory location or placeholder of value that must be declared by using datatype. It has to be declared because identifier (variable) are not part of the reserved words.

## 2.0 Learning Outcomes

---

At the end of studying this unit, you should be able to:

- i. Describe what and how identifiers are used
- ii. Understand the reserved words of java
- iii. Describe the various types of comments in java

## 3.0 Main Content

---

### 3.1 Identifiers in Java

---

An Identifier is a string of alphanumeric characters that begins with an alphabetic character or an underscore character, used to represent a programming element

You can think of an identifier as a name that identifies a programming element (variable, function, constant etc)

Take for example  $\text{Area} = \pi * (\text{Radius})^2$

pie = 3.142

Radius = Radius of the circle

$\text{Area} = \pi * (\text{Radius})^2$

Pie = 3.142 pie is the identifier for the constant 3.142

Radius = the identifier for the value of radius been computed

Area = the identifier for the memory location of the result of the computation

### REQUIREMENTS FOR IDENTIFIERS

1. A Java identifier must begin with a letter of the English alphabet, a non-English letter (such as  $\alpha$  or  $\pi$ ), an underscore, or a dollar sign. A class name cannot begin with a digit
2. A Java identifier can contain only letters, digits, underscores, or dollar signs
3. A Java identifier cannot be a reserved keyword, such as public or class
4. A Java identifier cannot be one of the following values: true, false, or null. These are not Keywords, but they are reserved and cannot be used

## 3.2 Java Reserved Words

---

These are special words with predefined meanings in any given programming language (keywords). Been reserved, they cannot be used other than for the functions they are predefined for. The words in the table below are examples of reserved words.

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

---

### Self-Assessment Exercise(s) 1

1. List twenty reserved words in java
2. Define an identifier and state any three requirements of identifier

## 3.3 Comments in Java

---

While writing large applications that perform many tasks, they may become increasingly complex. This could lead to difficulties in remembering why you did what you did. Documentation helps you remember why you wrote lines of code the way you did. Program comments are non-executable statements that can be added to a program for the purpose of documentation. This is beneficial to both the programmers and to others who might have reasons to read the programs in the future. Comments could include: indicating the author, the date, and the class name or function. The best practice dictates that you also include a brief comment to describe the purpose of each method you create within a class. There are 3 types of comments in Java:

1. Line comments (single line comment): They start with two forward slashes ( // ) and continue to the end of the current line. Line comments do not require an ending symbol of semicolon (;). See example below:

```
Int a = 24;    // a is a variable for age
```

2. Block comments (multi-line comment): start with ( /\* ) and end ( \*/ ). They can extend across as many lines as needed, for example

```
/* This shows  
that these comments  
don't matter */
```

3. Java Line comments and Block comments are called implementation comments while Javadoc comments are called documentation comments. Javadoc comments: are a special case of block comments called documentation comments because they are used to automatically generate nicely formatted program documentation with a program/tool named javadoc. They begin with ( /\*\* ) and end with ( \*/ ).

## 4.0 Conclusion

---

Variables, classes and methods are known as identifiers in java program. They are user defined entities therefore, they must be declared with datatype. Reserved words are known to java compiler as such, they are not identifiers. Comments in computer programs serves as a form of documentation.

## 5.0 Summary

---

You have learnt that:

- i. Variables, classes and methods (functions) are identifiers in java.
- ii. Java reserved words are not part of identifiers and they are known to the java compiler
- iii. Comments are medium of providing documentation to computer programs for readability, easier understanding and reference.

## 6.0 Tutor-Marked Assignment

---

1. What are the requirements of java identifiers?
2. State and explain the three types of comments in java program.
3. State the symbols that differentiate the single line comment from the multi-line comment.
4. Mention any three examples of identifiers in java.

## 7.0 References/ Further Readings

---

Herbert Schildt (2017) Java: A Beginner's Guide

Jamie Chan (2016) Learn Java in One Day and Learn It Well



# Unit 3

---

## Java Operators

### Contents

- 1.0 Introduction
- 2.0 Learning Outcomes
- 3.0 Learning Contents
  - 3.1 Arithmetic Operators
  - 3.2 Assignment Operators
  - 3.3 Relational Operators
  - 3.4 Logical Operators
  - 3.5 Unary Operators
  - 3.6 Bitwise Operators
  - 3.7 Ternary Operators
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment (TMA)
- 7.0 References/Further Readings

## 1.0 Introduction

---

Operators are symbols that perform operations on variables and values. For example, + is an operator used for addition, while \* is also an operator used for multiplication.

## 2.0 Learning Outcomes

---

At the end of studying this unit, you should be able to:

- iv. Describe the various types of java operators
- v. Understand how they are being used
- vi. Describe the different between assignment and equality operators

## 3.0 Main Content

---

### 3.1 Java Arithmetic Operators

---

Arithmetic operators are used to perform arithmetic operations on variables and data. For example,

`a + b;`

Here, the + operator is used to add two variables *a* and *b*. Similarly, there are various other arithmetic operators in Java.

Operator Operation

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Operation (Remainder after division)

Example 1: Arithmetic Operators

---

```
class Main {  
    public static void main(String[] args) {  
  
        // declare variables
```

```
int a = 12, b = 5;

// addition operator
System.out.println("a + b = " + (a + b));

// subtraction operator
System.out.println("a - b = " + (a - b));

// multiplication operator
System.out.println("a * b = " + (a * b));

// division operator
System.out.println("a / b = " + (a / b));

// modulo operator
System.out.println("a % b = " + (a % b));
}
}
```

### **Output**

```
a + b = 17
a - b = 7
a * b = 60
a / b = 2
a % b = 2
```

In the above example, we have used +, -, and \* operators to compute addition, subtraction, and multiplication operations.

### **/ Division Operator**

Note the operation, a / b in our program. The / operator is the division operator.

If we use the division operator with two integers, then the resulting quotient will also be an integer. And, if one of the operands is a floating-point number, we will get the result will also be in floating-point.

In Java,

(9 / 2) is 4

(9.0 / 2) is 4.5

(9 / 2.0) is 4.5

(9.0 / 2.0) is 4.5

### **% Modulo Operator**

The modulo operator % computes the remainder. When a = 7 is divided by b = 4, the remainder is **3**.

**Note:** The % operator is mainly used with integers.

## 3.2 Java Assignment Operators

---

Assignment operators are used in Java to assign values to variables. For example,

```
int age;
```

```
age = 5;
```

Here, = is the assignment operator. It assigns the value on its right to the variable on its left. That is, **5** is assigned to the variable *age*.

Let's see some more assignment operators available in Java.

### **Operator Example Equivalent to**

=        a = b;    a = b;

+=       a += b;    a = a + b;

-=       a -= b;    a = a - b;

\*=       a \*= b;    a = a \* b;

/=       a /= b;    a = a / b;

%=       a %= b;    a = a % b;

### Example 2: Assignment Operators

---

```
class Main {  
    public static void main(String[] args) {
```

```
// create variables
int a = 4;
int var;

// assign value using =
var = a;
System.out.println("var using =: " + var);

// assign value using +=
var += a;
System.out.println("var using +=: " + var);

// assign value using *=
var *= a;
System.out.println("var using *=: " + var);
}
}
```

### Output

```
var using =: 4
var using +=: 8
var using *=: 32
```

### Self-Assessment Exercise(s) 1

1. List all the types of arithmetic operators
2. How is assignment operator different from equality operator

## 3.3 Java Relational Operators

---

Relational operators are used to check the relationship between two operands. For example,

```
// check if a is less than b
a < b;
```

Here, > operator is the relational operator. It checks if *a* is less than *b* or not.

It returns either true or false.

Operator	Description	Example
==	Is Equal To	3 == 5 returns <b>false</b>
!=	Not Equal To	3 != 5 returns <b>true</b>
>	Greater Than	3 > 5 returns <b>false</b>
<	Less Than	3 < 5 returns <b>true</b>
>=	Greater Than or Equal To	3 >= 5 returns <b>false</b>
<=	Less Than or Equal To	3 <= 5 returns <b>true</b>

### Example 3: Relational Operators

---

```

class Main {
    public static void main(String[] args) {

        // create variables
        int a = 7, b = 11;

        // value of a and b
        System.out.println("a is " + a + " and b is " + b);

        // == operator
        System.out.println(a == b); // false

        // != operator
        System.out.println(a != b); // true

        // > operator
        System.out.println(a > b); // false

        // < operator
        System.out.println(a < b); // true

        // >= operator
        System.out.println(a >= b); // false

        // <= operator
    }
}

```

```

    System.out.println(a <= b); // true
}
}

```

**Note:** Relational operators are used in decision making and loops.

### 3.4 Java Logical Operators

---

Logical operators are used to check whether an expression is true or false. They are used in decision making.

Operator	Example	Meaning
&& (Logical AND)	expression1 expression2	&& true only if both <i>expression1</i> and <i>expression2</i> are true
(Logical OR)	expression1 expression2	true if either <i>expression1</i> or <i>expression2</i> is true
! (Logical NOT)	!expression	true if <i>expression</i> is false and vice versa

#### Example 4: Logical Operators

---

```

class Main {
    public static void main(String[] args) {

        // && operator
        System.out.println((5 > 3) && (8 > 5)); // true
        System.out.println((5 > 3) && (8 < 5)); // false

        // || operator
        System.out.println((5 < 3) || (8 > 5)); // true
        System.out.println((5 > 3) || (8 < 5)); // true
        System.out.println((5 < 3) || (8 < 5)); // false

        // ! operator
        System.out.println(!(5 == 3)); // true
        System.out.println(!(5 > 3)); // false
    }
}

```

#### Working of Program

- `(5 > 3) && (8 > 5)` returns true because both `(5 > 3)` and `(8 > 5)` are true.
- `(5 > 3) && (8 < 5)` returns false because the expression `(8 < 5)` is false.
- `(5 < 3) || (8 > 5)` returns true because the expression `(8 > 5)` is true.
- `(5 > 3) && (8 > 5)` returns true because the expression `(5 > 3)` is true.
- `(5 > 3) && (8 < 5)` returns false because both `(5 < 3)` and `(8 < 5)` are false.
- `!(5 == 3)` returns true because `5 == 3` is false.
- `!(5 > 3)` returns false because `5 > 3` is true.

### 3.5 Java Unary Operators

---

Unary operators are used with only one operand. For example, `++` is a unary operator that increases the value of a variable by 1. That is, `++5` will return 6.

Different types of unary operators are:

#### Operator Meaning

- `+`      **Unary plus:** not necessary to use since numbers are positive without using it
- `-`      **Unary minus:** inverts the sign of an expression
- `++`     **Increment operator:** increments value by 1
- `--`     **Decrement operator:** decrements value by 1
- `!`      **Logical complement operator:** inverts the value of a boolean

### Increment and Decrement Operators

---

Java also provides increment and decrement operators: `++` and `--` respectively. `++` increases the value of the operand by 1, while `--` decrease it by 1. For example,

```
int num = 5;
```

```
// increase num by 1
++num;
```

Here, the value of `num` gets increased to 6 from its initial value of 5.

#### Example 5: Increment and Decrement Operators

---

```
class Main {
    public static void main(String[] args) {

        // declare variables
        int a = 12, b = 12;
```



```

int result1, result2;

// original value
System.out.println("Value of a: " + a);

// increment operator
result1 = ++a;
System.out.println("After increment: " + result1);

System.out.println("Value of b: " + b);

// decrement operator
result2 = --b;
System.out.println("After decrement: " + result2);
}
}

```

## Output

```

Value of a: 12
After increment: 13
Value of b: 12
After decrement: 11

```

In the above program, we have used the ++ and -- operator as **prefixes (++a, --b)**. We can also use these operators as **postfix (a++, b++)**.

There is a slight difference when these operators are used as prefix versus when they are used as a postfix.

## 3.6 Java Bitwise Operators

---

Bitwise operators in Java are used to perform operations on individual bits. For example,

Bitwise complement Operation of 35

35 = 00100011 (In Binary)

~ 00100011

—————  
 11011100 = 220 (In decimal)

Here, ~ is a bitwise operator. It inverts the value of each bit (**0** to **1** and **1** to **0**).

The various bitwise operators present in Java are:

### Operator Description

~	Bitwise Complement
<<	Left Shift
>>	Right Shift
>>>	Unsigned Right Shift
&	Bitwise AND
^	Bitwise exclusive OR

Besides these operators, there are other additional operators in Java.

### 3.7 Java instanceof Operators

---

The instanceof operator checks whether an object is an instanceof a particular class. For example,

```
class Main {
    public static void main(String[] args) {

        String str = "Programiz";
        boolean result;

        // checks if str is an instance of
        // the String class
        result = str instanceof String;
        System.out.println("Is str an object of String? " + result);
    }
}
```

### Output

Is str an object of String? true

Here, *str* is an instance of the String class. Hence, the instanceof operator returns true.

## 3.8 Java Ternary Operators

---

The ternary operator (conditional operator) is shorthand for the if-then-else statement. For example,

```
variable = Expression ? expression1 : expression2
```

Here's how it works.

- If the Expression is true, expression1 is assigned to the *variable*.
- If the Expression is false, expression2 is assigned to the *variable*.

Let's see an example of a ternary operator.

```
class Java {  
    public static void main(String[] args) {  
  
        int februaryDays = 29;  
        String result;  
  
        // ternary operator  
        result = (februaryDays == 28) ? "Not a leap year" : "Leap year";  
        System.out.println(result);  
    }  
}
```

### Output

Leap year

In the above example, we have used the ternary operator to check if the year is a leap year or not.

## 4.0 Conclusion

---

This unit has adequately presents the major operators in java. Operators provides mechanism to manipulate two or more operands. In this case, operands are variables. Examples were given for further illustration and better your understanding.

## 5.0 Summary

---

All the major types of java operators have been dealt with by this unit.

## 6.0 Tutor-Marked Assignment

---

1. Which of the java operator can be shorten for if-then-else statement?
2. How does instanceof operator works.
3. How does bitwise operator works. Illustrate your explanation with bitwise compliment operation of 35.
4. Explain how does increment and decrement operators work?
5. Write down the symbols java used for the three logical operators
6. Briefly explain the following operators: (a) assignment operator (b) modulo operator (c) equality operator.

## 7.0 References/ Further Readings

---

Kingsley Sage (2019) Concise Guide to Object-Oriented Programming - An Accessible Approach Using Java.

Jose M. Garrido (2003) Object-Oriented Programming (From Problem Solving to JAVA) (Programming Series).

Edward Sciore (2019) Java Program Design: Principles, Polymorphism, and Patterns

# Unit 4

---

## Understanding Classes, Objects and Methods

### Contents

- 1.0 Introduction
- 2.0 Learning Outcomes
- 3.0 Learning Contents
  - 3.1 Objects and Classes in Java
  - 3.2 Method in Java
  - 3.3 Typical Examples of Object, Class and Method
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment (TMA)
- 7.0 References/Further Readings

## 1.0 Introduction

---

Java is object-oriented. What does that mean? Unlike languages, such as FORTRAN, that focus on giving the computer procedural / imperative “Do this/Do that” commands, object-oriented languages focus on data. Of course, object-oriented programs still tell the computer what to do. A class could be seen as a blueprint (of objects), It exist before any objects are created from. It describes the attributes its objects will posses and what those objects will be able to do. An object is an instance of a class. (is a specific, concrete instance of a class).

## 2.0 Learning Outcomes

---

At the end of studying this unit, you should be able to:

- vii. Describe what class mean in java
- viii. Describe what object mean in java
- ix. Understand the relationship between class and object
- x. Describe the meaning of method in java

## 3.0 Main Content

---

### 3.1 Objects and Classes in Java

---

An object in Java is the physical as well as a logical entity, whereas, a class in Java is a logical entity only. An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

- **Fields**
- **Methods**
- **Constructors**
- **Blocks**
- **Nested class and interface**

A Class is like an object constructor, or a "blueprint" for creating objects.

## Create a Class

---

To create a class, use the keyword `class`:

To create an object, a keyword `new` is used.

## 3.2 Method in Java

---

In Java, a method is like a function which is used to expose the behavior of an object.

A **method** is a block of code which only runs when it is called.

You can pass data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as **functions**.

The following are the rationale behind the use of Method

- Code Reusability
- Code Optimization

## Create a Method

---

A method must be declared within a class. It is defined with the name of the method, followed by parentheses `()`. Java provides some pre-defined methods, such as `System.out.println()`, but you can also create your own methods to perform certain actions:

## Example

---

Create a method inside Main:

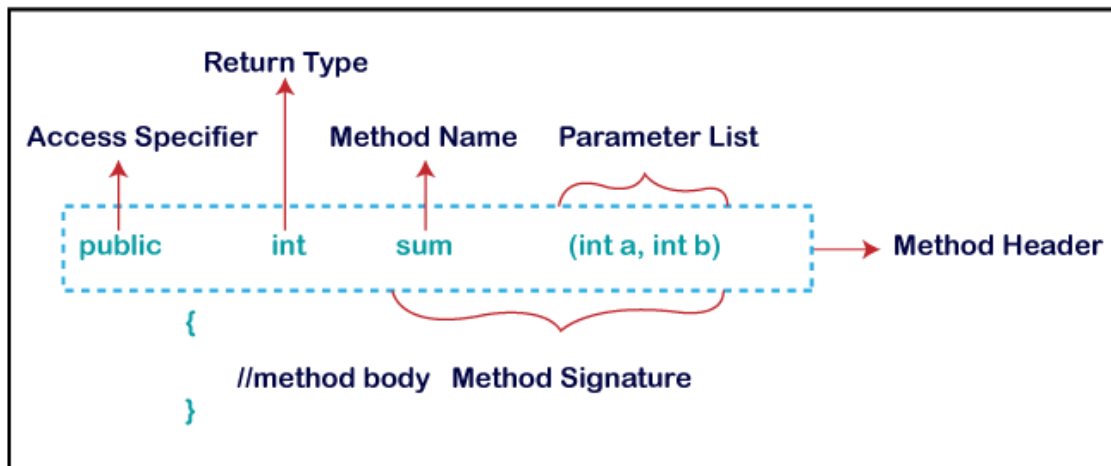
```
public class Main {
    static void myMethod() {
        // code to be executed
    }
}
```

### Example Explained

- `myMethod()` is the name of the method
- `static` means that the method belongs to the Main class and not an object of the Main class.
- `void` means that this method does not have a return value.

Another Clear example of Method:

## Method Declaration



**Method Signature:** Every method has a method signature. It is a part of the method declaration. It includes the **method name** and **parameter list**.

**Access Specifier:** Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **four** types of access specifier:

- **Public:** The method is accessible by all classes when we use public specifier in our application.
- **Private:** When we use a private access specifier, the method is accessible only in the classes in which it is defined.
- **Protected:** When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
- **Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

**Return Type:** Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.

**Method Name:** It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. Suppose, if we are creating a method for subtraction of two numbers, the method name must be **subtraction()**. A method is invoked by its name.

**Parameter List:** It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.

**Method Body:** It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

## Naming a Method

---

While defining a method, remember that the method name must be a **verb** and start with a **lowercase** letter. If the method name has more than two words, the first name must be a verb

followed by adjective or noun. In the multi-word method name, the first letter of each word must be in **uppercase** except the first word. For example:

**Single-word method name:** sum(), area()

**Multi-word method name:** areaOfCircle(), stringComparision()

It is also possible that a method has the same name as another method name in the same class, it is known as **method overloading**.

## Types of Method

---

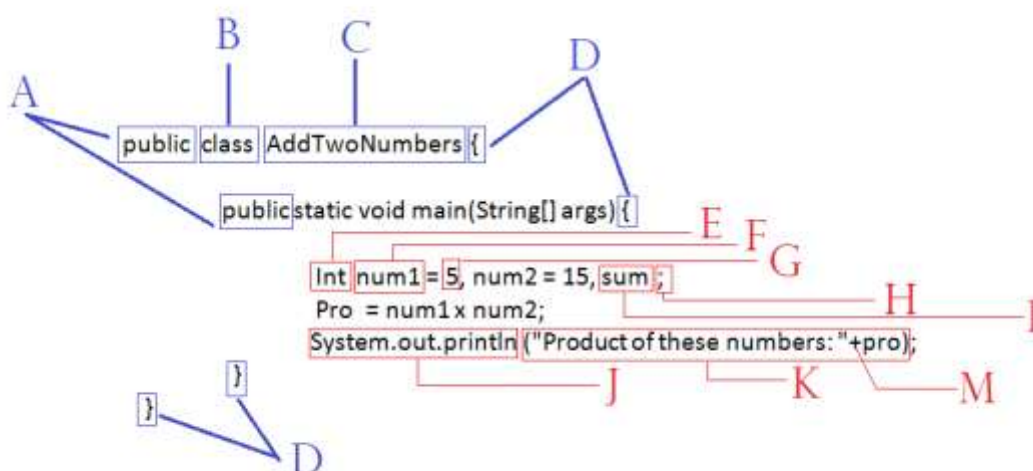
There are two types of methods in Java:

- Predefined Method
- User-defined Method

### 3.3 Typical example of Object, Class and Method

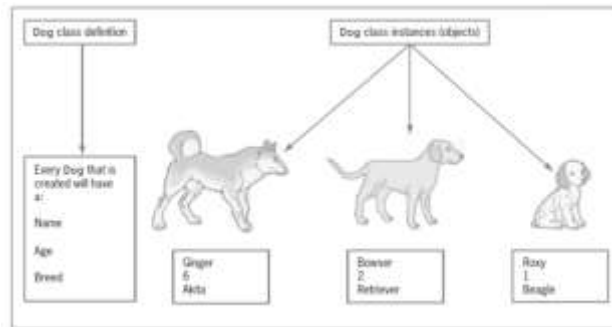
---

#### Second Example: Sum of two numbers



- 
- |           |  |
|-----------|--|
| A: public | The word public means that other Java programs can use the information that follows. |
| B: class  | Stands for class.....  |





- C: AddTwoNumbers      Is the name of the class
- D: {                              Stands for structural open and closing of any code segment
- E: Int                              Stands for Integer, the data type of the declared variable
- F: num1                              Stands for the identifier name of the variable
- G: = 5                              Assigning a constant value of 5 to num1
- H: ;                              Stands for structural ending of any code statement
- I: sum                              Stands for the memory location name that stores the result
- J: System.out.println      Java instruction that executes the “print”/ “Display” function
- K: ("Product of these numbers: ")      “print”/ “Display” exactly what’s in the (“”)
- M: +                              concatenates the sum to the print function

## 4.0 Conclusion

---

Class is a blueprint of object. Every object is defined by some characteristics and functions. The function is called method.

## 5.0 Summary

---

You have learnt that:

- i. In java, object and class are clear representation of each other
- ii. Every class encapsulate identifiers such as variables and methods.

## 6.0 Tutor-Marked Assignment

---

1. What is method overloading?
2. Briefly discuss the four types of access specifiers in java.
3. What are the components of method signature?
4. Enumerate two reasons why method is used in java.
5. What keyword word is used to create a class in java?
6. What keyword is used in java to create an object?
7. State the difference between object and class.

## 7.0 References/ Further Readings

---

Kingsley Sage (2019) Concise Guide to Object-Oriented Programming - An Accessible Approach Using Java.

Jose M. Garrido (2003) Object-Oriented Programming (From Problem Solving to JAVA) (Programming Series).

Edward Sciore (2019) Java Program Design: Principles, Polymorphism, and Patterns

# Unit 5

---

## Constructors, Interface and Enum

### Contents

- 1.0 Introduction
- 2.0 Learning Outcomes
- 3.0 Learning Contents
  - 3.1 Java Constructor
  - 3.2 Java Interface
  - 3.3 Enum in Java
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment (TMA)
- 7.0 References/Further Readings

## 1.0 Introduction

---

Besides, the normal class and object in Java, there are other similar concepts that behave likewise; for example, constructor. A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. Another way to achieve abstraction in Java, is with interfaces. The **Enum in Java** is a data type which contains a fixed set of constants.

## 2.0 Learning Outcomes

---

At the end of studying this unit, you should be able to:

- i. Describe what and how identifiers are used
- ii. Understand the reserved words of java
- iii. Describe the various types of comments in java
- iv. Understand when enum is used

## 3.0 Main Content

---

### 3.1 Java Constructor

---

A constructor in Java is similar to a method that is invoked when an object of the class is created. Unlike Java methods, a constructor has the same name as that of the class and does not have any return type. For example,

```
class Test {
    Test() {
        // constructor body
    }
}
```

Here, `Test()` is a constructor. It has the same name as that of the class and doesn't have a return type.

### Types of Constructor

---

In Java, constructors can be divided into 3 types:

1. No-Arg Constructor
2. Parameterized Constructor
3. Default Constructor

### 3.2 Java Interfaces

---

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods. The interface in Java is a mechanism to achieve *abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Java Interface also **represents the IS-A relationship**.

It cannot be instantiated just like the abstract class.

Since Java 8, we can have **default and static methods** in an interface.

Since Java 9, we can have **private methods** in an interface.

## Why use Java interface?

---

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

## How to declare an interface?

---

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

### Syntax:

---

1. interface <interface\_name>{
- 2.
3. // declare constant fields
4. // declare methods that abstract
5. // by default.
6. }

## 3.3 Enum in Java

---

An enum is a special "class" that represents a group of **constants** (unchangeable variables, like final variables).

To create an enum, use the enum keyword (instead of class or interface), and separate the constants with a comma. Note that they should be in uppercase letters: example

```
enum Level {  
    LOW,  
    MEDIUM,  
    HIGH  
}
```

## Enum inside a Class

---

You can also have an enum inside a class:

```

public class Main {
    enum Level {
        LOW,
        MEDIUM,
        HIGH
    }

    public static void main(String[] args) {
        Level myVar = Level.MEDIUM;
        System.out.println(myVar);
    }
}

```

It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY) , directions (NORTH, SOUTH, EAST, and WEST), season (SPRING, SUMMER, WINTER, and AUTUMN or FALL), colors (RED, YELLOW, BLUE, GREEN, WHITE, and BLACK) etc. According to the Java naming conventions, we should have all constants in capital letters. So, we have enum constants in capital letters.

Java Enums can be thought of as classes which have a fixed set of constants (a variable that does not change). The Java enum constants are static and final implicitly. It is available since JDK 1.5.

Enums are used to create our own data type like classes. The **enum** data type (also known as Enumerated Data Type) is used to define an enum in Java. Unlike C/C++, enum in Java is more *powerful*. Here, we can define an enum either inside the class or outside the class.

## 4.0 Conclusion

---

In this unit, the distinction between constructor, class and method is established. The data type enum is explained. And the keyword interface is also explained.

## 5.0 Summary

---

You have learnt that:

- i. Enums are used to create our own data type like classes.
- ii. Among other reasons, Java interface is used to achieve abstraction
- iii. Constructor has the same name as that of the class and doesn't have a return type.

## 6.0 Tutor-Marked Assignment

---

1. Write a java syntax to enumerate the three types of level as low, medium and high.
2. What is interface in java and show its syntax.
3. State three reasons why interface keyword is used in java.
4. Briefly explain constructor and mention the three types.
5. ----- also represents the IS-A relationship

## 7.0 References/ Further Readings

---

Kingsley Sage (2019) Concise Guide to Object-Oriented Programming - An Accessible Approach Using Java.

Jose M. Garrido (2003) Object-Oriented Programming (From Problem Solving to JAVA) (Programming Series).

Edward Sciore (2019) Java Program Design: Principles, Polymorphism, and Patterns

# Module 3

---

## Control Statements: Java Selection Statements

- Unit 1: If statement
- Unit 2: If else statement
- Unit 3: Nested if statement
- Unit 4: Switch statement



# Unit 1

---

## If Statement

### Contents

- 1.0 Introduction
- 2.0 Learning Outcomes
- 3.0 Learning Contents
  - 3.1 Introduction to if statement
  - 3.2 Syntax of if statement
  - 3.3 Example of if statement
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment (TMA)
- 7.0 References/Further Readings

## 1.0 Introduction

---

In java, the selection statements are also known as decision making statements or branching statements or conditional control statements. The selection statements are used to select a part of the program to be executed based on a condition. Java provides the following selection statements: if statement, if else statement, nested if statement and switch statement.

## 2.0 Learning Outcomes

---

At the end of studying this unit, you should be able to:

- i. Define java selection statements
- ii. Understand the syntax of selection statements
- iii. Use the statements for simple problem solving

## 3.0 Learning Contents

---

### 3.1 Introduction to if statement

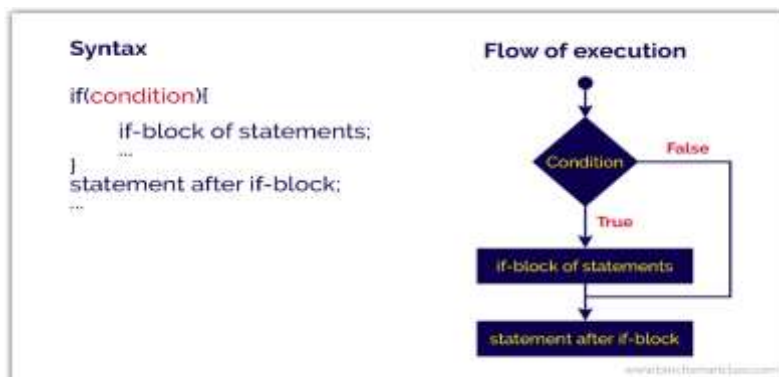
---

In java, we use the *if* statement to test a condition and decide the execution of a block of statements based on that condition result. The *if* statement checks, the given condition then decides the execution of a block of statements. If the condition is True, then the block of statements is executed and if it is False, then the block of statements is ignored.

### 3.2 Syntax of if statement

---

The syntax and execution flow of if the statement is as follows:



Note: statement after if-block; is optional. This implies that without the statement, the program would still execute.

### 3.3 Example of *if* statement

---

Let's look at the following example java code.

#### Java Program

```
import java.util.Scanner;

public class IfStatementTest {

    public static void main(String[] args) {

        Scanner read = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int num = read.nextInt();

        if((num % 5) == 0) {
            System.out.println("We are inside the if-block!");
            System.out.println("Given number is divisible by 5!!");
        }

        System.out.println("We are outside the if-block!!!");

    }

}
```

When we run this code, it produce the following output.

```
Enter any number:10
We are inside the if-block!
Given number is divisible by 5!!
We are outside the if-block!!!
```

In the above execution, assuming the number is 12 which is not divisible by 5, the condition becomes False and the condition is evaluated to False. Then the if statement ignores the execution of its block of statements. When we enter a number which is divisible by 5, then it produces the output as follows.

## 4.0 Conclusion

---

The selection *if* statement only execute if the condition is true. Otherwise, the program will abruptly come to a halt.

## 5.0 Summary

---

In this unit, you have learnt:

- i. How *if* statement works through its syntax.
- ii. How the java operator is used to form the condition.

## 6.0 Tutor-Marked Assignment

---

1. Write the syntax of java if statement
2. Write a simple java program that can find if a given number is a modulo of 5.

## 7.0 References/ Further Readings

---

Premchand S. Nair (2008) Java Programming Fundamentals: Problem Solving Through Object Oriented Analysis and Design

Iuliana Cosmina (2019) Java for Absolute Beginners: Learn to Program the Fundamentals the Java 9+ Way

Scott Sanderson. (2016) Java: Java Programming For Beginners - A Simple Start to Java Programming

# Unit 2

---

## If else statement

### Contents

- 1.0 Introduction
- 2.0 Learning Outcomes
- 3.0 Learning Contents
  - 3.1 Introduction to *if else* statement
  - 3.2 Syntax of *if else* statement
  - 3.3 Example of *if else* statement
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment (TMA)
- 7.0 References/Further Readings

## 1.0 Introduction

---

In java, we use the if-else statement to test a condition and pick the execution of a block of statements out of two blocks based on that condition result. The if-else statement checks the given condition then decides which block of statements to be executed based on the condition result. If the condition is True, then the true block of statements is executed and if it is False, then the false block of statements is executed.

## 2.0 Learning Outcomes

---

At the end of this unit, the student will be able to

- i. Define java selection statements
- ii. Understand the syntax of *if else* statements
- iii. Use the *if else* statements for simple problem solving

## 3.0 Learning Contents

---

### 3.1 Introduction to *if else* statement

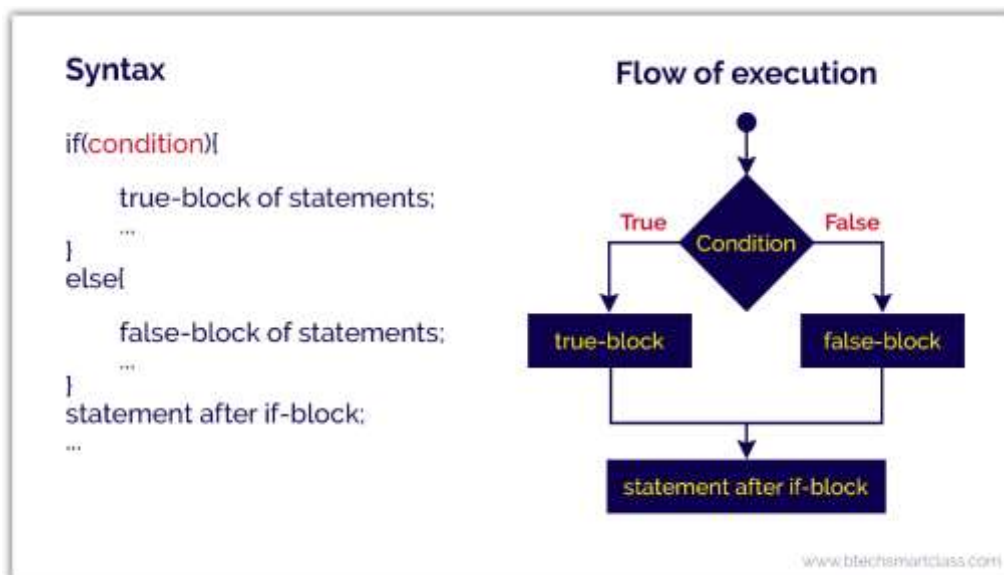
---

In java, we use the if-else statement to test a condition and pick the execution of a block of statements out of two blocks based on that condition result. The if-else statement checks the given condition then decides which block of statements to be executed based on the condition result. If the condition is True, then the true block of statements is executed and if it is False, then the false block of statements is executed.

### 3.2 Syntax of *if else* statement

---

The syntax and execution flow of if-else statement is as follows.



### 3.3 Example of *if else* statement

---

Let's look at the following example java code.

#### Java Program

```
import java.util.Scanner;

public class IfElseStatementTest {

    public static void main(String[] args) {

        Scanner read = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int num = read.nextInt();

        if((num % 2) == 0) {
            System.out.println("We are inside the true-block!");
            System.out.println("Given number is EVEN number!!");
        }
        else {
            System.out.println("We are inside the false-block!");
            System.out.println("Given number is ODD number!!");
        }

        System.out.println("We are outside the if-block!!!");

    }

}
```

When we run this code assume 12 is entered, it produce the following output.

Enter any number:

We are inside the true-block!

Given number is EVEN number!!

We are outside the if-block!!!"

### 4.0 Conclusion

---

The selection *if else* statement on one hand execute if the condition is true, and on the other hand also execute if the condition is false.

### 5.0 Summary

---

In this unit, you have learnt:

- i. How *if else* statement works through its syntax.
- ii. How the java operator is used to form the condition.

## 6.0 Tutor-Marked Assignment

---

1. Write the syntax of java if else statement
2. Write a simple java program that can find if a given number is even or odd.

## 7.0 References/ Further Readings

---

Premchand S. Nair (2008) Java Programming Fundamentals: Problem Solving Through Object Oriented Analysis and Design

Iuliana Cosmina (2019) Java for Absolute Beginners: Learn to Program the Fundamentals the Java 9+ Way

Scott Sanderson. (2016) Java: Java Programming For Beginners - A Simple Start to Java Programming



# Unit 3

---

## Switch Statement

### Contents

- 1.0 Introduction
- 2.0 Learning Outcomes
- 3.0 Learning Contents
  - 3.1 Introduction to *switch* statement
  - 3.2 Syntax of *switch* statement and example
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment (TMA)
- 7.0 References/Further Readings

## 1.0 Introduction

---

Using the switch statement, one can select only one option from more number of options very easily. In the switch statement, we provide a value that is to be compared with a value associated with each option. Whenever the given value matches the value associated with an option, the execution starts from that option. In the switch statement, every option is defined as a **case**.

## 2.0 Learning Outcomes

---

At the end of this unit, the student will be able to

- i. Define java switch statement
- ii. Understand the syntax of switch statement
- iii. Use the *switch* statements for simple problem solving

## 1.0 Learning Contents

---

### 3.1 Introduction to switch statement

---

Using the switch statement, one can select only one option from more number of options very easily. In the switch statement, we provide a value that is to be compared with a value associated with each option. Whenever the given value matches the value associated with an option, the execution starts from that option. In the switch statement, every option is defined as a **case**.

### 3.2 Syntax of switch statement and example

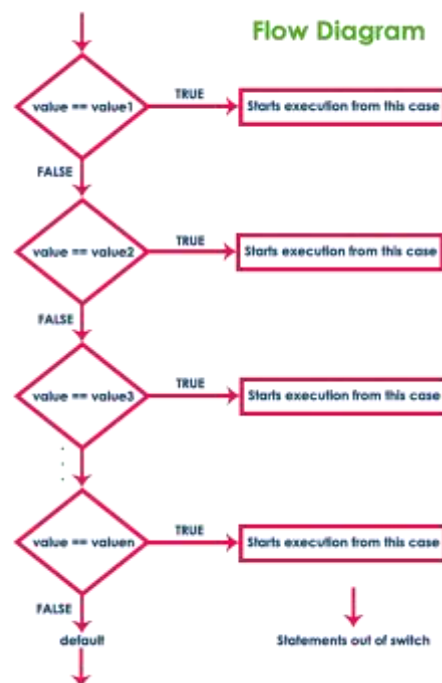
---

The switch statement has the following syntax and execution flow diagram.

#### Syntax

```
switch ( expression or value )
{
    case value1: set of statements;
                ....
    case value2: set of statements;
                ....
    case value3: set of statements;
                ....
    case value4: set of statements;
                ....
    case value5: set of statements;
                ....
    .
    .
    default: set of statements;
}
```

#### Flow Diagram



Let's look at the following example java code.

### Java Program

```
import java.util.Scanner;
```

```
public class SwitchStatementTest {
```

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

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

```
        System.out.print("Press any digit: ");
```

```
        int value = read.nextInt();
```

```
        switch( value )
```

```
        {
```

```
            case 0: System.out.println("ZERO") ; break ;
```

```
            case 1: System.out.println("ONE") ; break ;
```

```
            case 2: System.out.println("TWO") ; break ;
```

```
            case 3: System.out.println("THREE") ; break ;
```

```
            case 4: System.out.println("FOUR") ; break ;
```

```
            case 5: System.out.println("FIVE") ; break ;
```

```
            case 6: System.out.println("SIX") ; break ;
```

```
            case 7: System.out.println("SEVEN") ; break ;
```

```
            case 8: System.out.println("EIGHT") ; break ;
```

```
            case 9: System.out.println("NINE") ; break ;
```

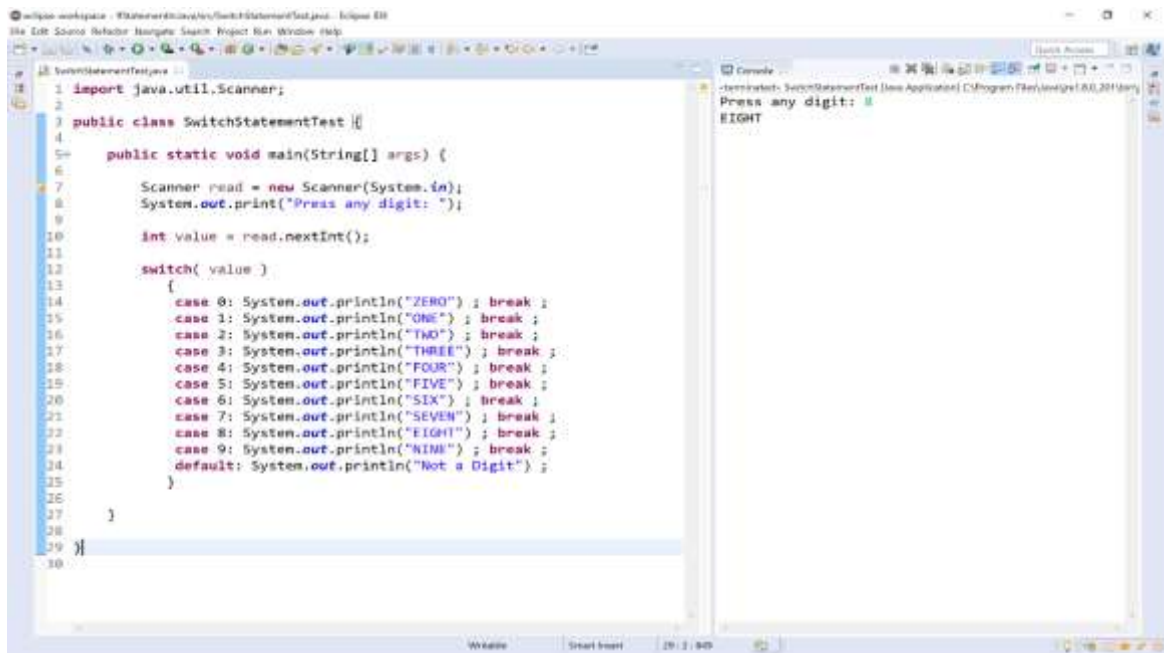
```
            default: System.out.println("Not a Digit") ;
```

```
        }
```

```
    }
```

```
}
```

When we run this code, it produce the following output.



```
1 import java.util.Scanner;
2
3 public class SwitchStatementTest {
4
5     public static void main(String[] args) {
6
7         Scanner read = new Scanner(System.in);
8         System.out.println("Press any digit: ");
9
10        int value = read.nextInt();
11
12        switch( value )
13        {
14            case 0: System.out.println("ZERO"); break ;
15            case 1: System.out.println("ONE"); break ;
16            case 2: System.out.println("TWO"); break ;
17            case 3: System.out.println("THREE"); break ;
18            case 4: System.out.println("FOUR"); break ;
19            case 5: System.out.println("FIVE"); break ;
20            case 6: System.out.println("SIX"); break ;
21            case 7: System.out.println("SEVEN"); break ;
22            case 8: System.out.println("EIGHT"); break ;
23            case 9: System.out.println("NINE"); break ;
24            default: System.out.println("Not a Digit");
25        }
26
27    }
28
29 }
30
```

Console

```
Press any digit: 8
EIGHT
```

## 4.0 Conclusion

---

Even though, the execution of *if else* statement has the capability to return either of the Boolean value, it is not without limitation. For multiple conditions, it is more effective and faster to use switch statement instead of the former.

## 5.0 Summary

---

In this unit, you have learnt:

- i. How *switch* statement works through its syntax.
- ii. How the java operator is used to form the condition.

## 6.0 Tutor-Marked Assignment

---

1. Write the syntax of java switch statement.

## 7.0 References/ Further Readings

---

Herbert Schildt (2017) Java: A Beginner's Guide

Jamie Chan (2016) Learn Java in One Day and Learn It Well

Programming for the Absolute Beginner, 2009

# Module 4

---

## Control Statements: Java Looping Statements

- Unit 1: For loop
- Unit 2: While loop
- Unit 3: Do while loop
- Unit 4: Java Break statement

# Unit 1

---

## For Loop

### Contents

- 1.0 Introduction
- 2.0 Learning Outcomes
- 3.0 Learning Contents
  - 3.1 Brief introduction to *for* loop
  - 3.2 Syntax of *for* loop
  - 3.3 Simple example of *for* loop
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment (TMA)
- 7.0 References/Further Readings

## 1.0 Introduction

---

In programming languages, loops are used to execute a set of instructions/functions repeatedly when some conditions become true. Loop constructs is sometimes refer to as Iterative or Repetitive construct. There are three types of loops in Java

## 2.0 Learning Outcomes

---

At the end of this unit, you should be able to achieve the following Objectives:

1. Describe looping statements
2. Understand the syntax of the *for* statement
3. Use the *for looping* statements for simple problem solving

## 3.0 Learning Contents

---

### 3.1 Brief Introduction to for loop

---

The Java for loop is a control flow statement that iterates a part of the programs multiple times. Among other loops, for loop is essentially used if the number of iteration is fixed, it is recommended to use for loop.

### 3.2 Syntax of *for* loop.

---

The simple syntax of this type of loop is as follows:

```
for(init;condition;incr/decr){ .....1  
// statement to be executed .....2  
} .....3
```

From step1, *for* is a keyword and the parenthesis contains three parameters which are as follows: *init* stands for initialization, for example, letter *i* = 0, *condition*, and *incr/decr* stands for increment/decrement. Step 2 contains the java codes to be executed within the curly braces{...}.

A simple for loop is the same as C/C++. We can initialize the variable, check condition and increment/decrement value. It consists of four parts:

1. **Initialization:** It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.
2. **Condition:** It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.
3. **Statement:** The statement of the loop is executed each time until the second condition is false.
4. **Increment/Decrement:** It increments or decrements the variable value. It is an optional condition.

### Syntax:

1. for(initialization;condition;incr/decr){
2. //statement or code to be executed
3. }

### Self-Assessment Exercise(s) 1

1. When essentially can *for* loop be used?
2. What are the components of *for* loop

### 3.3 Simple Example of *for* loop

---

The example below uses *for* loop to print or output numbers from 1 to 10.

```
//for loop
for(int i=1;i<=10;i++){
System.out.println(i);
}
```

In this case, the initialization is *i=1*, letter *i* is declared with datatype *int*

The condition is *i<=10*; this implies that *i=1* continue to print and quickly stop as soon as 10 is printed.

In this example, increment is the stepwise action since the number starts from 1 to 10 thus, *i++*.

### 4.0 Conclusion

---

The loop consist of initialization, condition, increment/decrement and executed statements as the components. If the number of iteration is fixed, it is recommended to use *for* loop.

### 5.0 Summary

---

*For* loop has compacted components, with a simple line of code, it take cares of multiple conditions.

### 6.0 Tutor Marked Assignment

---

1. When essentially *for* loop is used?
2. Write the syntax of *for* loop.
3. Use *for* loop to output numbers of 1 to 10 in java.



## 7.0 References.

---

Herbert Schildt (2017) Java: A Beginner's Guide

Jamie Chan (2016) Learn Java in One Day and Learn It Well

Programming for the Absolute Beginner, 2009

# Unit 2

---

## While Loop

### Contents

- 1.0 Introduction
- 2.0 Learning Outcomes
- 3.0 Learning Contents
  - 3.1 Brief introduction of *while* loop
  - 3.2 Syntax of *while* loop
  - 3.3 Simple example of *while* loop
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment (TMA)
- 7.0 References/Further Readings

## 1.0 Introduction

---

The Java while loop is a control flow statement that executes a part of the programs repeatedly on the basis of given Boolean condition. The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

## 2.0 Learning Outcomes

---

At the end of studying this unit, you should be able to achieve the following Objectives:

- i. Understand the syntax of the *while* statement
- ii. Use the *while looping* statements for simple problem solving

## 3.0 Learning Contents

---

### 3.1 Brief introduction of *while loop*

---

The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

### 3.2 Syntax of while loop

---

Syntax:

1. `while(condition){`
2. `//statement to be executed`
3. `}`

### 3.3 Simple example of *while loop*

---

```
//while loop
int i=1;
while(i<=10){
System.out.println(i);
i++;
}
```

In this case, the initialization is `i=1`, letter `i` is declared with datatype `int`

The condition is `i<=10`; this implies that `i=1` continue to print and quickly stop as soon as 10 is printed.

In this example, increment is the stepwise action since the number starts from 1 to 10 thus, `i++`.

## 4.0 Conclusion

---

The loop consist of initialization, condition, increment/decrement and executed statements as the components. If the number of iteration is not fixed, it is recommended to use while loop.

## 5.0 Summary

---

While loop has compacted components, with a simple line of code, it take cares of multiple conditions.

## Tutor Marked Assignment

---

- i. When essentially *while* loop is used?
- ii. Write the syntax of *while* loop.
- iii. Use *while* loop to output numbers of 1 to 10 in java.

## 7.0 References

---

Eckel, B. (2003). *Thinking in JAVA*. Prentice Hall Professional.

Baesens, B., Backiel, A., & Vanden Broucke, S. (2015). *Beginning Java programming: the object-oriented approach*. John Wiley & Sons.

Halim, S., Halim, F., Skiena, S. S., & Revilla, M. A. (2013). *Competitive programming 3*. Lulu Independent Publish.

Schildt, H., & Coward, D. (2014). *Java: the complete reference* (p. 1312). New York: McGraw Hill Education.

# Unit 3

---

## Do While Loop

### Contents

- 1.0 Introduction
- 2.0 Learning Outcomes
- 3.0 Learning Contents
  - 3.1 Brief introduction of *do while* loop
  - 3.2 Syntax of *do while* loop
  - 3.3 Simple example of *do while* loop
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment (TMA)
- 7.0 References/Further Readings

## 1.0 Introduction

---

The Java do while loop is a control flow statement that executes a part of the programs at least once and the further execution depends upon the given Boolean condition. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop.

## 2.0 Learning Outcomes

---

At the end of studying this unit, you should be able to achieve the following Objectives:

1. Understand the syntax of the *do while* statement
2. Use the *do while looping* statements for simple problem solving

## 2.0 Learning Contents

---

### 3.1 Brief introduction of *do while* loop

---

The Java *do-while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop. The Java *do-while loop* is executed at least once because condition is checked after loop body.

### 3.2 Syntax of *do while* loop

---

#### **Syntax:**

1. do{
2. //code to be executed
3. }while(condition);

### 3.3 Simple Example of *Do While* Loop

---

```
//do-while loop
int i=1;
do{
System.out.println(i);
i++;
}while(i<=10);
```

In this case, the initialization is i=1, letter i is declared with datatype int

The condition is i<=10; this implies that i=1 continue to print and quickly stop as soon as 10 is printed.

In this example, increment is the stepwise action since the number starts from 1 to 10 thus, i++.

## 4.0 Conclusion

---

The loop consist of initialization, condition, increment/decrement and executed statements as the components. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

## 5.0 Summary

---

Do While loop has compacted components, with a simple line of code, it take cares of multiple conditions. The basic difference between *while loop* and *do while loop* is that the former checks the condition before execution and the latter execute before condition is checked.

## 6.0 Tutor Marked Assignment

---

- i. When essentially *do while* loop is used?
- ii. Write the syntax of *do while* loop.
- iii. Use *do while* loop to output numbers of 1 to 10 in java.
- iv. State the basic difference between *while* and *do while* loop.

## 7.0 References

---

Eckel, B. (2003). *Thinking in JAVA*. Prentice Hall Professional.

Baesens, B., Backiel, A., & Vanden Broucke, S. (2015). *Beginning Java programming: the object-oriented approach*. John Wiley & Sons.

Halim, S., Halim, F., Skiena, S. S., & Revilla, M. A. (2013). *Competitive programming 3*. Lulu Independent Publish.

Schildt, H., & Coward, D. (2014). *Java: the complete reference* (p. 1312). New York: McGraw-Hill Education.



# Unit 4

---

## Java Break Statement

### Contents

- 1.0 Introduction
- 2.0 Learning Outcomes
- 3.0 Learning Contents
  - 3.1 Von Neumann Model
  - 3.2 Fetch-Decode-Execute cycle
  - 3.3 Base Machine Organization
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment (TMA)
- 7.0 References/Further Readings

## 1.0 Introduction

---

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop. The Java break statement is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop. We can use Java break statement in all types of loops such as for loop, while loop and do while loop.

## 2.0 Learning Outcomes

---

At the end of studying this unit, you should be able to achieve the following Objectives:

1. Describe break statement.
2. Understand how break statement is used in the looping statements.

## 2.0 Learning Contents

---

### 3.1 How break statement works

---

Break Statement is a loop control statement that is used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stop there, and control returns from the loop immediately to the first statement after the loop.

The **break** statement in Java programming language has the following two usages –

- When the **break** statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- It can be used to terminate a case in the **switch** statement.

### 3.2 Syntax of Break statement

---

The simple syntax of break statement, which is a single statement inside any loop is as follows:

```
break;
```

## 4.0 Conclusion

---

We can use break statement with a label. The feature is introduced since JDK 1.5. So, we can break any loop in Java now whether it is outer or inner loop.

## 5.0 Summary

---

Basically, break statements are used in situations when we are not sure about the actual number of iterations for the loop or we want to terminate the loop based on some condition.

**Break:** In Java, the break is majorly used for:

- Terminate a sequence in a switch statement (discussed above).

- To exit a loop.
- Used as a “civilized” form of goto.

## 6.0 Tutor Marked Assignment

---

- i. Briefly explain how break statement works.
- ii. State three major uses of java break statement.

## 7.0 References

---

Eckel, B. (2003). *Thinking in JAVA*. Prentice Hall Professional.

Baesens, B., Backiel, A., & Vanden Broucke, S. (2015). *Beginning Java programming: the object-oriented approach*. John Wiley & Sons.

Halim, S., Halim, F., Skiena, S. S., & Revilla, M. A. (2013). *Competitive programming 3*. Lulu Independent Publish.

Schildt, H., & Coward, D. (2014). *Java: the complete reference* (p. 1312). New York: McGraw-Hill Education.

# Module 5

---

## Exception Handling

Unit 1: Java throwable's inheritance hierarchy

Unit 2: Try, catch, throw and throws in Java

# Unit 1

---

## Java throwable's inheritance hierarchy

### Contents

- 1.0 Introduction
- 2.0 Learning Outcomes
- 3.0 Learning Contents
  - 3.1 Throwable Class Hierarchy in Java
  - 3.2 Methods of Throwable Class in Java
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment (TMA)
- 7.0 References/Further Readings

## 1.0 Introduction

---

All exceptions and errors in Java are represented by classes. All these classes are organized as subclasses in a hierarchy under a superclass called Throwable.

Throwable in Java is a class that is the superclass of all exceptions and errors which may occur in Java program. It extends object class. Therefore, throwable class is itself a subclass of the superclass of all Java classes, Object class.

## 2.0 Learning Outcomes

---

At the end of this unit, you will be able to:

1. Define a program
2. Differentiate between the three main programming languages
3. Identify the different program translator
4. Differentiate between procedural programming and object-oriented programming

## 3.0 Learning Contents

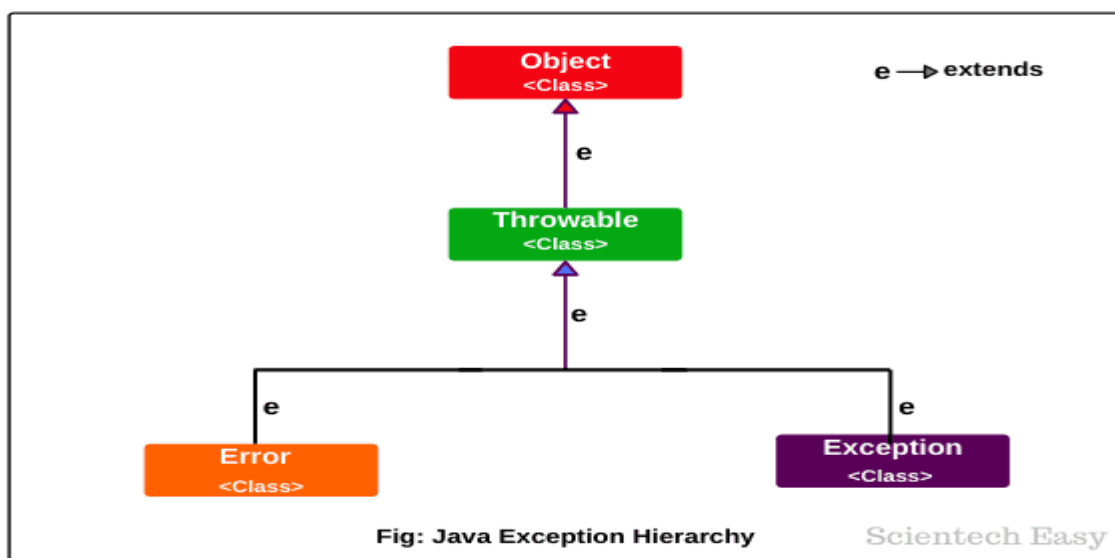
---

### 3.1 Throwable Class Hierarchy in Java

---

Throwable class is the top most class in the exception class hierarchy but it is also an immediate subclass of object class that is located at the root of exception hierarchy.

The throwable class hierarchy is shown in the below diagram.



As shown in the above figure, Throwable class which is derived from the Object class is the top of exception hierarchy from which all exception classes are derived. It is present in java.lang package.

Throwable class is the superclass of all exceptions in java. This class has two subclasses: Error and Exception. Errors or exceptions occurring in java programs are objects of these classes. Using Throwable class, you can also create your own custom exceptions.

### Self-Assessment Exercise(s) 1

1. Throwable class is derived from which class?
2. What are the two subclasses of throwable class?
3. Throwable class is present in which java package?

## 3.2 Methods of Throwable Class in Java

---

All exception classes including user-defined class use all the methods provided by Throwable class. Exception class does not define its own any methods.

Java provides the following methods in the Throwable class that can be used to retrieve and know details of an exception. An important list of methods present in Throwable class is as follows:

1. **getMessage():** The getMessage() method returns detail message of exception that has recurred. The syntax for getMessage() is as follows:

```
public String getMessage()
```

The detail message is initialized in the throwable constructor that can be displayed using the following statement:

```
System.out.println(e.getMessage());
```

Since getMessage() method returns a string, we can store string message returned by getMessage() method in a String variable and the content of string variable can be displayed using the following statement:

```
String str = e.getMessage();  
System.out.println(str);
```

2. **toString():** The toString() method returns information about an exception in the form of string. The throwable class overrides toString() method of object class. The information can be displayed on the screen using the println() method. The following syntax can be used:

```
public String toString()
```

For example:

```
System.out.println(e.toString());  
or,
```

```
String str = e.toString(); // Here, str is a reference variable that stores
information about an exception returned by toString() method in the form
string.
System.out.println(str);
```

3. **printStackTrace():** This method prints the stack trace of an exception. It gives information about where the exception occurs in the case of nested method calls. The syntax for `printStackTrace()` method is as follows:

```
public void printStackTrace() // It does not return anything.
```

For example:

```
e.printStackTrace(); // e is an exception type variable.
```

4. **fillInStackTrace():** This method returns a `Throwable` object that contains a completed stack trace. The object can be rethrown. The syntax for `fillInStackTrace()` method is given below:

```
public Throwable fillInStackTrace()
```

5. **getStackTrace():** The `getStackTrace()` method returns an array that contains each element on the stack trace. The element at index 0 represents the top of call stack and the last element represents the bottom of call stack. The syntax is as follows:

```
public StackTraceElement[] getStackTrace()
```

6. **getCause():** The `getCause()` method returns the exception that caused the occurrence of current exception. If there is no caused exception then null is returned. The syntax is as follows:

```
public Throwable getCause()
```

7. **initCause():** The `initCause()` method joins “causeExc” with the invoking exception and returns a reference to the exception. The syntax is given below:

```
public Throwable initCause(Throwable causeExc)
```

---

Java language provides a hierarchy of classes that represent different types of exceptions. These classes are derived from `java.lang.Throwable` class.

## 5.0 Summary

---

In this unit, you learned that:

- **Throwable class in Java** defines several useful methods that can be used for getting and knowing the details of an exception. Hope that this tutorial has covered almost all important points related to `Throwable` class.
- You can throw only objects that derive from the `Throwable` class.



- The top three classes in this hierarchy (the **Throwable**, **Error**, and **Exception** classes) are all defined in the **java.lang** package (which is automatically imported into every class file).

## 6.0 Tutor-Marked Assignments

---

- i. Briefly explain five methods that java provided for throwable class.
- ii. Which of the java method of throwable class returns an array that contains each element on the stack trace?
- iii. Which of the java method of throwable class prints the stack trace of an exception?

## 7.0 References/Further Readings

---

Scott Sanderson. (2016) Java: Java Programming For Beginners - A Simple Start to Java Programming.

Alvaro Felix (2016) JAVA: Easy Java Programming for Beginners, Your Step-By-Step Guide to Learning Java Programming

# Unit 2

---

## Try, catch, throw and throws in Java

### Contents

- 1.0 Introduction
- 2.0 Learning Outcomes
- 3.0 Learning Contents
  - 3.1 What is an Exception and why does an Exception occur?
  - 3.2 Blocks & Keywords used for exception handling
  - 3.3 When to Checked and Unchecked Exceptions
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment (TMA)
- 7.0 References/Further Readings

## 1.0 Introduction

---

**Exception in Java** is an event that interrupts the execution of program instructions and disturbs the normal flow of program execution. It is an object that wraps an error event information that occurred within a method and it is passed to the runtime system. In Java, exceptions are mainly used for indicating different types of error conditions.

There are two types of errors:

1. Compile time errors
2. Runtime errors

Compile time errors can be again classified again into two types:

- Syntax Errors
- Semantic Errors

### **Syntax Errors Example:**

Instead of declaring `int a;` you mistakenly declared it as `in a;` for which compiler will throw an error.

Example: You have declared a variable `int a;` and after some lines of code you again declare an integer as `int a;`. All these errors are highlighted when you compile the code.

## 2.0 Learning Outcomes

---

By the end of this unit, you should be able to:

- i. Define exception in java
- ii. List the two types of errors
- iii. List the block keywords used for exception handling

## 1.0 Learning Contents

---

### 3.1. What is an Exception and why does an Exception occur?

---

An exception is an “unwanted or unexpected event”, which occurs during the execution of the program that is, at run-time, that disrupts the normal flow of the program’s instructions. When an exception occurs, execution of the program gets terminated.

An exception can occur due to several reasons like Network connection problem, Bad input provided by user, Opening a non-existing file in your program etc

## 3.2 Blocks & Keywords used for exception handling

---

1.**try**: The try block contains set of statements where an exception can occur.

```
try
{
    // statement(s) that might cause exception
}
```

2.**catch** : Catch block is used to handle the uncertain condition of try block. A try block is always followed by a catch block, which handles the exception that occurs in associated try block.

```
catch
{
    // statement(s) that handle an exception
    // examples, closing a connection, closing
    // file, exiting the process after writing
    // details to a log file.
}
```

3.**throw**: Throw keyword is used to transfer control from try block to catch block.

4.**throws**: Throws keyword is used for exception handling without try & catch block. It specifies the exceptions that a method can throw to the caller and does not handle itself.

5.**finally**: It is executed after catch block. We basically use it to put some common code when there are multiple catch blocks.

## 3.3 When to Checked and Unchecked Exceptions

---

It's a good practice to use exceptions in Java so that we can separate error-handling code from regular code. However, we need to decide which type of exception to throw. The Oracle Java Documentation provides guidance on when to use checked exceptions and unchecked exceptions:

“If a client can reasonably be expected to recover from an exception, make it a checked exception. If a client cannot do anything to recover from the exception, make it an unchecked exception.”

## Self-Assessment Exercise(s) 1

1. When to use unchecked exception in java?
2. When to use checked exception in java?

### 4.0 Conclusion

---

This unit has introduced various blocks of java keywords for handling exceptions, which include try, catch, throw, throws and finally. Besides, checked and unchecked exceptions are equally accounted for.

### 5.0 Summary

---

1. We use the keyword throws to throw the checked exception up the stack to the calling method to handle. This is what FileInputStream has just done to you.
2. With try catch block, you simply wrap the Java code which throws the checked exception within the block.
3. Checked exceptions are the exceptions that are checked at compile time.
4. Unchecked exceptions are the exceptions that are not checked at compile time.

### 6.0 Tutor-Marked Assignments

---

- i. What is the difference between throw and throws keywords?
- ii. State the difference between checked and unchecked exceptions.
- iii. What are the two types of errors in java program?
- iv. Define an exception in java.
- v. State the reason why exception occur.

### 7.0 References/Further Readings

---

Scott Sanderson. (2016) Java: Java Programming For Beginners - A Simple Start to Java Programming.

Alvaro Felix (2016) JAVA: Easy Java Programming for Beginners, Your Step-By-Step Guide to Learning Java Programming.

## Answers to Self-Assessment Exercises

---

### MODULE 1

#### UNIT 1

##### SELF-ASSESSMENT EXERCISE 1:

- i. **A computer program** is a set of instructions that you write to tell a computer what to do.
- ii. True
- iii. A programmer is anyone who can write or develop computer program
- iv. True

##### SELF-ASSESSMENT EXERCISE 2:

- i. instructions
- ii. high level and low level programming language
- iii. Java, BASIC, FORTRAN, PASCAL, Python, C, C#, C++,
- iv. High level programming languages
- v. True

#### UNIT 2

##### SELF-ASSESSMENT EXERCISE 1:

1. An object-oriented programming language (OOPL) is a high-level programming language based on the object-oriented model. To embark on object-oriented programming, one needs an object-oriented programming language.
2. Java, Python, C++, C#, VB.net, SmallTalk
3. They are:
  - i. Develop reusable software modules
  - ii. Reduce production cost
  - iii. Quicken the completion time for software development
  - iv. Reduce maintenance cost
4. Encapsulation, Abstraction, Inheritance and Polymorphism
5. Abstraction

#### UNIT 3

##### SELF-ASSESSMENT EXERCISE 1:

1. Because Java supports encapsulation, abstraction, inheritance and polymorphism.
2. Through translator (compiler).
3. Class className {  
,,,,,,,,,,,,,,,,,,,,,  
}. For example, `public class AddTwoNumbers {`
  4. `public static void main(String[] args) {`
  5. (a) Java is robust (b) Java is secure (c) Java is multithreaded (d) Java is portable

#### UNIT 4

##### SELF-ASSESSMENT EXERCISE 1:

1. Java produces java byte code when the java source code is compiled. That is, java byte code is a product of java program compilation.
2. At the execution of the java program or source code
3. Java standard edition (SE), Java enterprise edition (EE) and Java Micro Edition (ME)
4. JDK, JRE and JVM
5. JVM – Java Virtual Machine, Java EE – Enterprise Edition and JDK – Java Development Kits

## **MODULE 2**

### **UNIT 2**

#### **SELF-ASSESSMENT EXERCISE 1:**

1. See Table in page 40
2. An Identifier is a string of alphanumeric characters that begins with an alphabetic character or an underscore character, used to represent a programming element. The requirements are as follows:
  - i. A Java identifier must begin with a letter of the English alphabet, a non-English letter (such as  $\alpha$  or  $\pi$ ), an underscore, or a dollar sign. A class name cannot begin with a digit
  - ii. A Java identifier can contain only letters, digits, underscores, or dollar signs
  - iii. A Java identifier cannot be a reserved keyword, such as public or class
  - iv. A Java identifier cannot be one of the following values: true, false, or null. These are not Keywords, but they are reserved and cannot be used

### **UNIT 3**

#### **SELF-ASSESSMENT EXERCISE 1:**

1. They are addition, subtraction, multiplication, division, modulo operations
2. Assignment operator takes single (=) sign while equality takes double (==) signs.

## **MODULE 4**

### **UNIT 1**

#### **SELF-ASSESSMENT EXERCISE 1:**

1. for loop is essentially used if the number of iteration is fixed
2. initialization, condition, increment/decrement and statement

## **MODULE 5**

### **UNIT 1**

#### **SELF-ASSESSMENT EXERCISE 1:**

1. Object class
2. Error and Exception
3. java.lang package (java.lang.Throwable)

### **UNIT 2**

#### **SELF-ASSESSMENT EXERCISE 1:**

1. If a client cannot do anything to recover from the exception, make it an unchecked exception
2. If a client can reasonably be expected to recover from an exception, make it a checked exception.