ignou
THE PEOPLE'S
UNIVERSITY

Indira Gandhi National Open University
School of Computer and Information Sciences (SOCIS)

MCS-206
Object Oriented Prgramming using Java

MCS-206
Object Oriented Programming using Java

BLOCK 1	**Java Fundamentals**	5
UNIT 1	Introduction to Java	7
UNIT 2	Basics of Java	26
UNIT 3	Decisions and Loops Constructs	54
UNIT 4	Class and Objects	72
BLOCK 2	**Inheritance, Polymorphism and Packages**	109
UNIT 5	Inheritance	111
UNIT 6	Polymorphism	138
UNIT 7	Assertions, Annotations and Exception Handling	156
UNIT 8	Packages and Interfaces	186
BLOCK 3	**Multithreading and I/O**	217
UNIT 9	Multithreading	219
UNIT 10	Number Character and Strings	246
UNIT 11	I/O Streams	280
UNIT 12	Java API	309
BLOCK 4	**Graphical User Interface and Java Database Connectivity**	341
UNIT 13	Introduction to GUI in Java	343
UNIT 14	Working with UI Controls	373
UNIT 15	JDBC Part - 1	395
UNIT 16	JDBC Part - 2	421

PROGRAMME DESIGN COMMITTEE

Prof. (Retd.) S.K. Gupta , IIT, Delhi
Prof. T.V. Vijay Kumar JNU, New Delhi
Prof. Ela Kumar, IGDTUW, Delhi
Prof. Gayatri Dhingra, GVMITM, Sonipat
Mr. Milind Mahajani, Impressico Business Solutions, New Delhi
Sh. Shashi Bhushan Sharma, Associate Professor, SOCIS, IGNOU
Sh. Akshay Kumar, Associate Professor, SOCIS, IGNOU
Dr. P. Venkata Suresh, Associate Professor, SOCIS, IGNOU
Dr. V.V. Subrahmanyam, Associate Professor, SOCIS, IGNOU
Sh. M.P. Mishra, Assistant Professor, SOCIS, IGNOU
Dr. Sudhansh Sharma, Assistant Professor, SOCIS, IGNOU

COURSE DESIGN COMMITTEE

Prof. S. Balasundaram, JNU, New Delhi
Prof. Sonajahria Minz, JNU, New Delhi
Dr. Aditi Sharan, JNU, New Delhi
Dr. Sanjay Kumar Dubey, Amity University, Noida
Dr. Rahul Johri, GGSIPU, New Delhi
Dr. Gurmeet Kaur, School of Law, IGNOU
Dr. Anand Gupta, School of Law, IGNOU
Sh. Shashi Bhushan Sharma, Associate Professor, SOCIS, IGNOU
Sh. Akshay Kumar, Associate Professor, SOCIS, IGNOU
Dr. P. Venkata Suresh, Associate Professor, SOCIS, IGNOU
Dr. V.V. Subrahmanyam, Associate Professor, SOCIS, IGNOU
Sh. M.P. Mishra, Assistant Professor, SOCIS, IGNOU
Dr. Sudhansh Sharma, Assistant Professor, SOCIS, IGNOU

SOCIS FACULTY

Prof. P. Venkata Suresh, Director, SOCIS, IGNOU
Prof. V.V. Subrahmanyam, SOCIS, IGNOU
Prof. Sandeep Singh Rawat, SOCIS,IGNOU
Prof. Divakar Yadav, SOCIS,IGNOU
Dr. Akshay Kumar, Associate Professor, SOCIS, IGNOU
Dr. M.P. Mishra, Associate Professor, SOCIS, IGNOU
Dr. Sudhansh Sharma, Assistant Professor, SOCIS, IGNOU
Dr.Manish Kumar, Assistant Professor, SOCIS, IGNOU

PREPARATION TEAM

Unit Writers:

Dr. Sunil Kumar Singh, Assistant Professor, Dept. of Computer Science and Information Technology, Mahatma Gandhi Central University, Motihari **(Units 1,5, and 6)**
Dr. M.P.Mishra, Associate Professor, SOCIS, IGNOU, New Delhi **(Units 2, 4, and 10)**
Ms. Poonam Trikha, IT Training & Support Exec., NIELIT, New Delhi **(Units 3,8, and 14)**
Mr. Vikas Dublish, Sr. Technical Architect HCL Technologies Ltd. Noida **(Unit 7)**
Prof. Rekha Gupta, Area Chair-IT & Systems, Lal Bahadur Shastri Institute of Management, New Delhi **(Unit 9)**
Dr. Gaurav Baranwal, Assistant Professor, Dept. of Computer Science, BHU, Varanasi **(Unit 11)**
Dr. Kavita Pabreja Associate Professor, Dept. of Computer Applications, Maharaja Surajmal Institute,New Delhi **(Unit 12)**
Mr. Vinay Kumar Sharma, Senior Software Engineer Swayam Project, COE, IGNOU, New Delhi **(Unit 13)**
Dr. Achal Kaushik, Professor and HOD (CSE), Bhagwan Parshuram Institute of Technology, New Delhi **(Units 15 and 16)**

Content Editor
Dr. Rahul Johari,
Associate Professor
GGSIP, New Delhi

Language Editor
Prof. Anju Sahgal Gupta (Retired),
SOH, IGNOU, New Delhi

Course Coordinator: Dr. M.P.Mishra, Associate Professor, SOCIS, IGNOU

PRINT PRODUCTION

Sh. Sanjay Aggarwal
Assistant Registrar, MPDD
IGNOU, New Delhi

July, 2023

©*Indira Gandhi National Open University, 2023*

ISBN-

All rights reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from the Indira Gandhi National Open University.

Further information on the Indira Gandhi National Open University courses may be obtained from the University's office at Maidan Garhi, New Delhi-110068.

Printed and published on behalf of the Indira Gandhi National Open University, New Delhi by MPDD, IGNOU.

Laser Typesetter: Tessa Media & Computers, C-206, Shaheen Bagh, Jamia Nagar, New Delhi-110025

COURSE INTRODUCTION

A skill of problem solving is an essential and core of computer science study. It helps to solve problems and translate them in program using programming languages. As a programmer you have to solve many different types of problems. For that you have to learn programming languages like C, C++, C#, Java etc. In this course you will learn Java programming language. Java is a powerful general-purpose class-based, object-oriented programming language programming language. It is fast, portable and available in different platforms (works on different operating systems such as Windows, Mac, and Linux). Learning Java coding will provide you opportunity to demonstrate your problem-solving skills by writing java program. Java was created at Sun Microsystems, Inc., by James Gosling and a team of researchers led by him. James Gosling is known as the father of Java.

Basic features of Object Oriented Programming such as class, object, data hiding, inheritance, polymorphism helps in developing systems to solve real life problems with ease and effectiveness. Inheritance is very important feature of OOP in which a new class is created by inheriting the features from the already existing class. Java support inheritance and provide facility of code reusability. Exception handling in Java is the process of responding to the occurrence of exceptions. Java Exception handling framework provides options to handle runtime errors. Java support Multi-threaded programming which enables you to write program in a way where multiple activities can be processed concurrently in the same program. In Java APIs are important software components bundled with the JDK. The Java APIs include classes, interfaces, and user Interfaces that makes it easier to work with collections of objects. APIs enable you to integrate various applications and websites and offer real-time information. Some of the Java Collections API are lists, maps, stacks etc. One more very popular Java APIs is Java Database Connectivity (JDBC).

You can use Java to develop:

- Desktop Applications
- Web Applications
- Gamming Applications
- Mobile Applications (especially Android apps)
- Embedded systems
- Cloud –based Applications
- Big data Technologies
- IoT Applications

In this course you will learn core Java programming and some Java features like multithreading, JDBC, JavaFX etc.

ignou
THE PEOPLE'S
UNIVERSITY

BLOCK 1
JAVA FUNDAMENTALS

BLOCK 1 INTRODUCTION

Java is a general purpose object oriented programming language. It is very popular programming language and used in IT industry for developing different types of software systems including desktop applications, mobile applications, web applications, gamming applications, IoT applications etc. To learn Java programming first you need to learn basic concepts and principles of object oriented programming. This block is on Java fundamentals. In this block you will learn about basics of OOP, Java data types, operators, decision and looping constructs, concept and implementation of class and objects.

This block consists of 4 units and is organised as follows:

Unit 1: This unit introduce object oriented programming and its basic concepts which include class and objects, abstraction, encapsulation, inheritance and polymorphism. Features of Java programming language are also explained in this unit. Finally, Java development environments and tools such as JDK, JRE, JVM and JIT are discussed.

Unit 2: In this unit data types, operators and keywords of Java are explained. Also, this unit discuss Unicode. In this unit you will learn writing basic statements and expressions in Java program.

Unit 3: This unit explains decision constructs: if, if-else and switch-case. Also, loop constructs: for, while-loop, do-while loop are explained in this unit. How to use declare and array in Java is also explained in this unit.

Unit 4: This is one of the important units of this course. It explains concepts of class, object, encapsulation, access modifier. Also, use of constructor, use of *this key* word and use of *finalize method* are explained in this unit.

UNIT 1 INTRODUCTION TO JAVA

Structure **Page No.**

1.0 Introduction 7
1.1 Objectives 8
1.2 Introduction to Object Oriented Programming 8
1.2.1 Objects and Class 8
1.2.2 Data Hiding 9
1.2.3 Abstraction 9
1.2.4 Encapsulation 10
1.2.5 Inheritance 10
1.2.6 Polymorphism 12
1.3 Features of Java Programming Language 13
1.3.1 Use of Java in Industry 17
1.4 Java Development Environment and Tools 17
1.4.1 JDK 18
1.4.2 JRE 18
1.4.3 JIT and JVM 18
1.5 Environment Setup for Java Application Development 20
1.6 Summary 23
1.7 Solutions/Answer to Check Your Progress 24
1.8 References and Further Reading 25

1.0 INTRODUCTION

It is a modern, high-level, evolutionary programing language that combines a well-designed structure with powerful features. In addition to the core language components, its software distribution is enriched with power supporting libraries for the purposes such as database, network, and GUI programming.

Java is a popular computer language with object-orientation concepts. To have a clear understanding of Java, we have to understand the reason behind its creation that forces and shapes the legacy that it inherits.

- Java is adaptive to the changing environments and uses. It supports the thinking of real-life situations in the form of objects and classes.
- To implement refinements and improvements in the art of programming in terms of data-centric applications development.

If we look at the history ofthe Java programming language, it was developed by James Gosling and his colleagues at Sun Microsystems in early 1990. Initially, it was started as a project called “Oak” with the goal to run it on a Virtual machine and have greater simplicity than C/C++. The first public implementation of Java 1.0 was released in 1995 with the goal of “Write Once, Run Anywhere”. Now, after many upgrades, we have Java 18 as the most updated version in the year 2022.

There are many application areas of Java programming language; including mobile application developments, it is considered the official language and compatible with well-known software Android Studio. Desktop GUI applications can be easily developed in Java programming language because of its Abstract Window Toolkit, Swing and JavaFX concepts. It is also suited for many other areas like Web-based

applications, Enterprise Applications, Scientific Applications, Gaming Applications etc. Its garbage collection module facilitates the 3-D game development.

This unit covered the foundation of object-orientedprogramming fundamentals and its founding concepts like Inheritance, Polymorphism, encapsulation, and abstraction. Further, we will discuss the Java programming language features that make it unique among all the programming languages. Also, robustness and portability are well-known properties of the Java programming language.

The Java development environment is an essential component for writing and executing programs. This includes various components like JDK, JRE, JVM, etc. Also, we will write and runour first Java program using Integrated Development Environment (IDE) in this unit.

1.1 OBJECTIVES

After going through this unit, you will be able to:

- Explain characteristics of Object-Oriented Programming,
- Explain features of Java,
- Explain the use of JDK, JIT and JVM, and
- Write and execute a simple Java program.

1.2 INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING

Java programming supports concepts of classes and objects. Classes in Java can be considered as logical constructs upon which the entire java language is built.

Java programming languages have inherited object-orientation concepts from previously developed object-oriented programming languages such as C++ etc.. Some basic characteristics of Object Orientated programming, which you will extensively be using in this course of Java Programming, are explained here.

1.2.1 Objects and Class

An object is nothing but a real word entity that can be distinctly identified.Also objects pose their characteristics and featurs. A class is a template or blueprint that defines data fields and methods of objects of the same type.

Objects have the states and behaviours, while a class describes the behaviour/state of the object of its type support. When we look at real-world examples, cars, dogs, humans etc.are the objects. Dog shows its state and behaviour like name, breed and color, barking, wagging the tail, and running,etc..

In figure 1. 2, a circle class is shown with its three objects. We can see that each circle object is represented bythe different radius. Here radius is the data field. Every object has its own data field value and methods implementation is common for all the objects. You will learn more about objects and classes in Unit 4 of this Block.

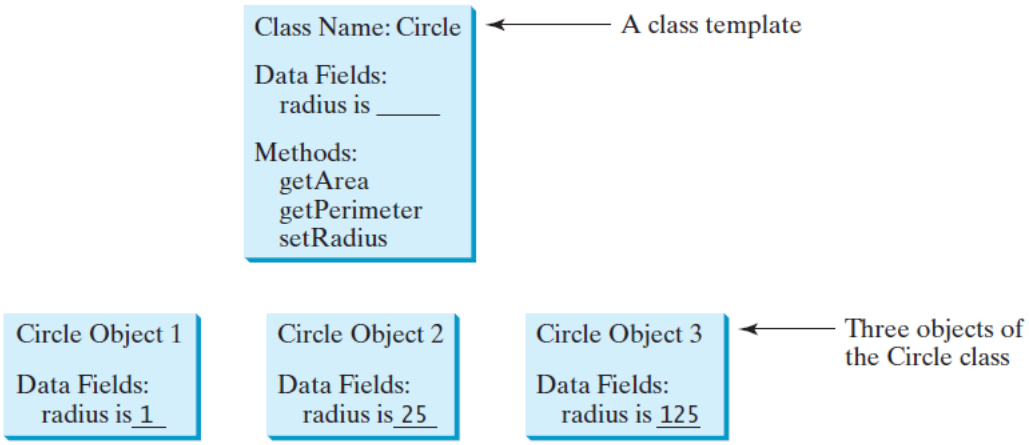


Figure 1.1: A class template along with objects

1.2.2 Data Hiding

Data hiding is one of the key aspects of Object-Oriented Programming that enables developers to protect their data and hide the implementation details from outside interference. To implement the data hiding, variables in a class should have a private access modifier. Sowhen someone needs to access or modify the private variables from outside of the class, they should only be accessible by using the "get" and can be modified by using "set" methods.

1.2.3 Abstraction

Data abstraction is the process of hiding detailed information and only presenting the essential information to the user. For example, a car can be viewed as a car rather than its individual components. Data abstraction can also be seen for identifying the important characteristics of an object by ignoring the useless details. In Java programming, abstraction can be achieved by using the abstract class/interfaces.

You can also apply the method abstraction to separate the use of the method from its implementation. Any user(client) can use a method without knowing about its implementation details. The details about the implementation are encapsulated in the method, and it will be hidden from the client who invokes this method. Further, if you change the implementation of a method, it will not affect the client program till you do not change the method signature.

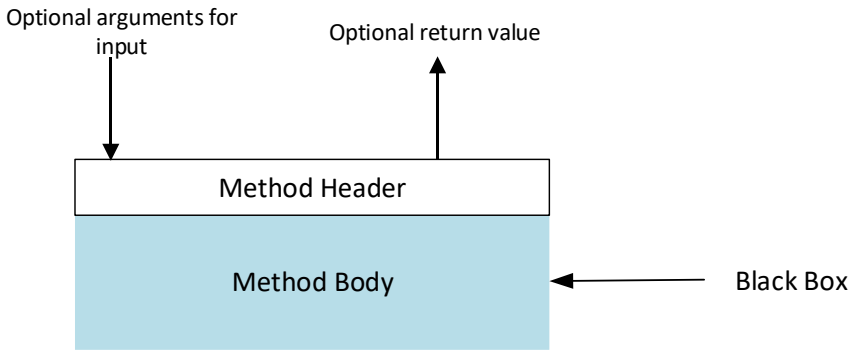


Figure 1.2: Method body as a Black Box

In figure 1.2, we can see that method body as a black box as its implementation is hidden from the user (client). For example when we use the *System.out.println method* to display a string, we do not bother about how it is implemented.

Throughout this course, we will see extensive use of the method System.out.println. As a user, we require to write the code to invoke the method that is the example method abstraction.

The abstraction concept can be applied to the process of developing large programs; while writing the code, divide-and-conquer (also known as stepwise refinement) technique can be applied to decompose the subproblems. Similarly, class abstraction can separate the class implementation from how the class is used. The class owner describes the functions of the class and lets the user know how the class can be used.

1.2.4 Encapsulation

Encapsulation is another key concept used in object-oriented programming. It is a process of binding the data members and methods together. It is done in the form of a secure manner by making the data field accessible only to class members by declaring the data fields as private. The power of encapsulated code is that everyone outside of the class knows how to access and use the methods without thinking about implementation details. In Java, setter or getter methods are also used to ensure the read-only or write-only operations.

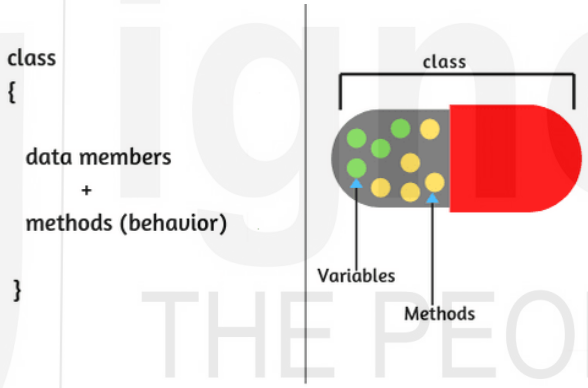


Figure 1.3: Encapsulation in Java

Figure 1.3 shows the encapsulation in Java, in which data members and methods are bound together and can be protected from outside classes.

Class abstraction and encapsulation are two sides of the same coin. Many real-life examples illustrate the class abstraction concept. Let's take an example of getting a loan, in which a specific loan can be seen as an object of the Loan class. The interest rate, loan amount, and loan period are its data fields. Calculating the monthly payment and total payments are its methods. The data fields are also known as data members.

For example, when you take a car loan, a loan object is created by instantiating the class Loanwith the loan interest rate, loan amount, and loan period. As a user of the Loan class, you are not required to know how these methods are implemented.You can use the methods to find your car loan's monthly payment and loan payment.

1.2.5 Inheritance

Inheritance provides reusability by inheriting the property in a new class from the derived class. It is a process by which one object acquires the property of another object.The Inheritance allows forthe creationof a new class similar to the previously defined class, and the new class can have its own data members and methods. This feature of Java supports the concept of hierarchical classification.

Let us consider a Four-Wheeler class that contains data members and methods. The data members are engine capacity, number of gears, length, breadth, power steering, audio System, etc., and methods are forward movement, backward movement, stopping movement, etc. Let us assume that thisFour-Wheeler class is defined and inuse. Now a new class Car is needed to define which is similar to the defined class but needs some additional data members such as air conditioner, number of airbags, anti-breaking System and methods such as controlling air conditioner, automatic transmission, etc. The class Car need not include all data members and methods of the class Four-Wheeler. The class Car can reuse the features defined in class Four-Wheeler by inheriting it so that the Car class contains all data members and methods of the Four-Wheeler and data members and methods of its own. This new class Caralone needs to be defined and tested. This new class inherits all the data and methods from the base class (in this case from class Four-Wheeler). The word Inheritance reflects how children inherit many of their characteristics from their parents. But the children also have their own characteristics. In OOP, a base class is seen as a Super Class(Parent class) and a derived class as a Sub Class (child class).

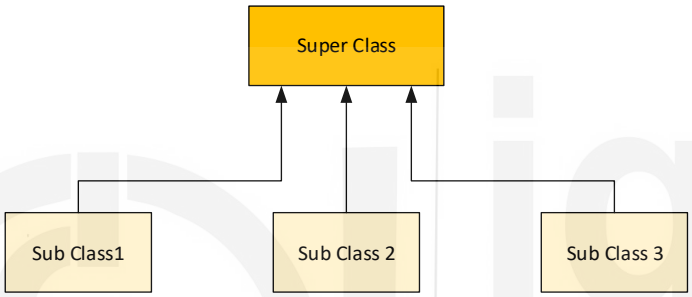


Figure 1.4: Representation of Inheritance

There are several advantages of Inheritance. It can reduce the development time of the program because reusability of the code requires less memory, less execution time, and redundancy of the code can be minimized. The real-life example of Inheritance is a child, and parents, like a child, can inherit all their father's properties.

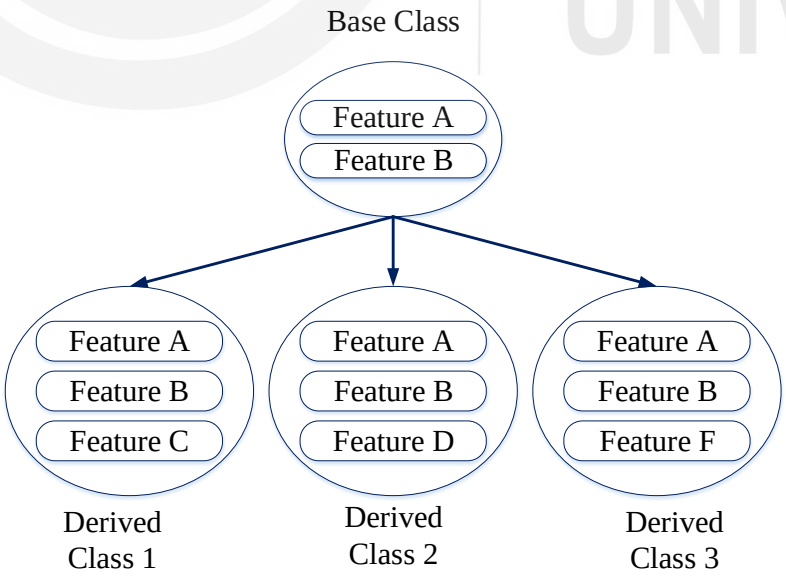


Figure 1.5: Inheritance Hierarchy

Figure 1.5 shows the working of Inheritance; like you can see, three classes are derived from the base class. Features A, and B of the base class are inherited in all the three derived classes. Further, you can also see that derived classes have their

own features like Feature C, Feature D, and Feature F in Derived Class1, 2, and 3, respectively. Therefore, new classes use the concept of code reusability and extend their functionality. You will learn more about Inheritance and its implementation in Java in the next Block of this course.

1.2.6 Polymorphism

Polymorphism is the ability of an object to take many forms. It is a feature that allows one interface to be used for a general class of actions.In programming languages, polymorphism is the ability of an object to take on many forms. Polymorphism is derived from a Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So Polymorphism means many forms. Polymorphism is one of the important characteristics of OOPs. Many object-oriented programming languages support this characteristic, but their implementations vary from one object-oriented programming language to another object-oriented programming language. The polymorphism principle is implemented in java programming with method overloading and method overriding.

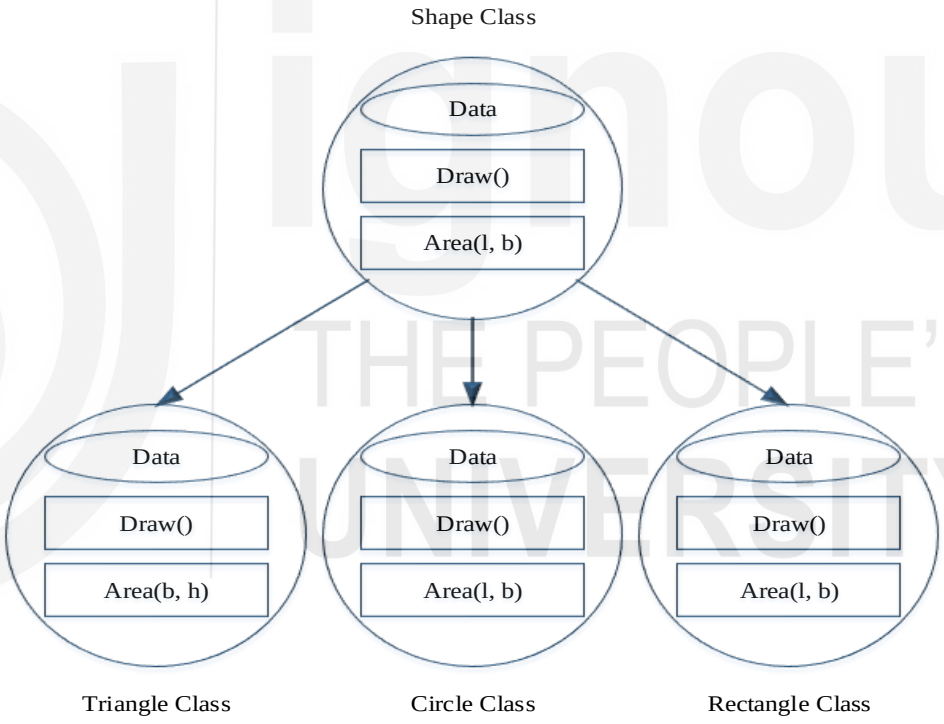


Figure 1.6: Polymorphism

From figure 1.6, we can see that a single method draw () can be used to draw the different shapes like Triangle, Circle, and Rectangle. So here, we can observe that name of the function is the same, but the functionality(implementation) is different. This is known as function overriding, and it is a type of the polymorphism.

The polymorphism principle is divided into two sub-principles. They are:

- Static Polymorphism
- Dynamic Polymorphism

Static binding is also called as early binding or compiler time-bindingof the method with object. Overloading is compile time Polymorphism where more than one method shares the same name with different parameters or signatures and different

return types. The overloaded method should always be part of the same class, with the same name but different parameters. The parameters may differ in their type or number, or both. Methods may have the same or different return types. Method overloading is the best example of static binging. In Method overloading, which method to be calledis decided by the compiler at compile time.

Dynamic binding is also called *late binding* or *run-time binding*. Dynamic polymorphism is achieved at run-time. Here, Java compiler does not understand which method is called at time of compilation of the code. Only JVM decides which method is to be called at run-time. Method overloading and method overriding using instance methods are examples of dynamic polymorphism. Method overriding is the best example of dynamic binding. In method overriding, which method will be called is decided at run-time. Dynamic method dispatch is a mechanism by which a call to an overridden method is resolved at run-time. This is how Java implements run-time polymorphism. When a reference calls an overridden method, Java determines the version of method to be executed based on the type of object it refers to. In the next Block of this course, you will learn the details of polymorphism and its implementation in Java.

☞Check Your Progress – 1

1) What is a class? How is a class different from objects?

……

……

……

……

……

2) What is encapsulation?

……

……

……

……

……

3) What is Polymorphism? What are the different types of Polymorphism?

……

……

……

……

……

1.3 FEATURES OF JAVA

Java is a well established popular programming language because of its popular features like portability and security. In addition, many other factors also played an essential role in moulding the final form of the language. Some important features of Java programming are briefly discussed in this section.

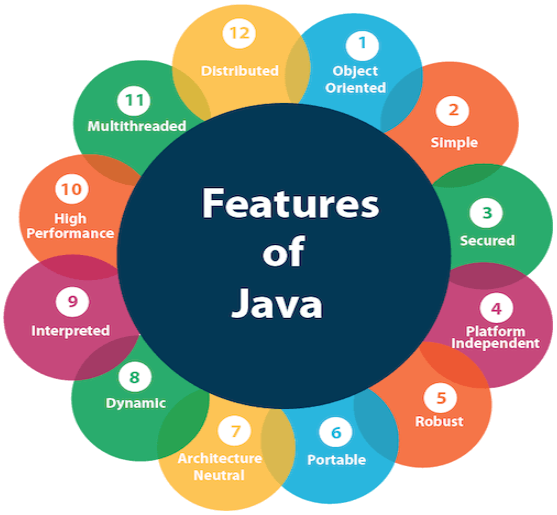


Figure 1.7: Features of Java

1) Object-Oriented

Java is a programming language which is mainly based on object-oriented programming concepts, and we can say that everything in Java is an object. Object-oriented programming is so integral to Java that it is best to understand its basic principles before you begin writing even simple java programs. Objects have two fundamental purposes, which are as given below.

- To promote understanding of the real-world problems in a natural way.
- To provide practical basics for implementations of the design of the system.

An object contains data and methods which manipulate these data. For example, a student object may have the name, age, number, gender, course-name, university-name etc as data member, which represents the attribute of this object. The student object can perform a number of functions such as receiving this data, storing, finding the course-name, university-name etc. Also, in a system there can be many object students in a university with their unique identities.

People often hesitate to learn Java (OOP) because its learning curve is steeper than top-down programming (like C programming language). But once you learn the basics of OOP, you will find that it is easier and more intuitive approach for developing big projects or complex systems.

2) Simple

Java was designed to be easy to understand and easy to write for the professional programmer. If you have some programming experience, you will be able to grab the java concept quickly. In case you have an understanding of object-oriented programming, then learning Java will be easy. The one who has C++, programming experience, learning Java will be pretty easy for them.Although Java is case sensitive, it provides user-friendly syntax that can be understood easily.

3) Secure

Java is one of the well-known programming languages for security. You can develop a virus-free system using java programming language because it has no explicit pointer;it runs inside a virtual machine sandbox. Java also verifies the illegal Bytecode that can violate the access rights to the objects.

4) Portable

Portability is one of the significant challenges for internet applications because many computing devices and operating system are connected to the applications. Due to the different architecture of the devices/operating system, running a single application on them is a challenge. Sometimes application code is required to be written considering platform-specific requirements when an application is developed using programming languages like C or C++. Java is an architecture-neutral programming language, and it is portable because it facilitates you to carry the java bytecode to any platform. With the help of bytecode, you can run the Java program on any computer virtually. More about bytecode you may learn from MCSL-210 course.

In general, Java provides three types of portability which are as follows.

i. Source Code Portability

When we run any given Java program, it must produce identical results regardless of the underlying CPU, operating system or Java compiler.

ii. CPU Architecture Portability

Currently, existing Java compilers produce object code (called Bytecode) to a CPU which physically does not yet exist. For executing the same Bytecode on different real CPU, a Java interpreter or Java Virtual Machine(JVM) is used. This nonexistent CPU(i.e. JVM) allows the same object code to be executed in any CPU for which there is a JVM.

iii. OS/GUI

Similar to the JVM, which presents a virtual CPU, the Java libraries have a virtual operating system/GUI. Java solves this problem by providing a set of library functions (in libraries provided by Java, such as awt, util and lang) that converse with an imaginary operating system and an imaginary GUI(Graphical User Interafce). Each Java version provides libraries that implement this virtual operating system/ GUI. The Java programs using these libraries easily provide the required port of operating system functionality and GUI.

5) Robust

Java provides an automatic garbage collector that runs on the virtual machine and destroys the objects which are not being used in the program anymore. The java designers paid enough attention so that the program could run reliably in various systems. While designing the Java, they focused on the program's robustness on high priority. There are two main reasons to better understand the robust java programming behaviour: strong memory management and well-handled exceptions.

Exceptional conditions in traditional environments often arise in situations such as division by zero, file not found etc. Java easily handles all these exceptions by providing object-oriented exception handling. In Block 3 of this course, you will learn more about Java Exceptions Handling.

6) Multithreaded

Java programming is designed to meet the real-world requirement of creating interactive, networked programs. For this, Java uses the concepts of multithreadingto write the code, which can do so many things simultaneously.

The Java run-time systems comes with an elegant solution for multiprocess synchronization that enables you to construct smoothly running interactive systems.You will learn more about the multithreading of Java in Block 3 of this course.

7) Architecture-neutral

Code longevity and portability were central issues for java designers. While designing Java, they have faced the issue that there is no guarantee that the program written today will run tomorrow−even on the same machine. Any update in the computer system can cause the malfunction of the system. The major goal was to "write once; run anywhere, anytime". They designed Java Virtual Machine(JVM) with their hard efforts to overcome this situation. To a great extent, this goal was achieved.

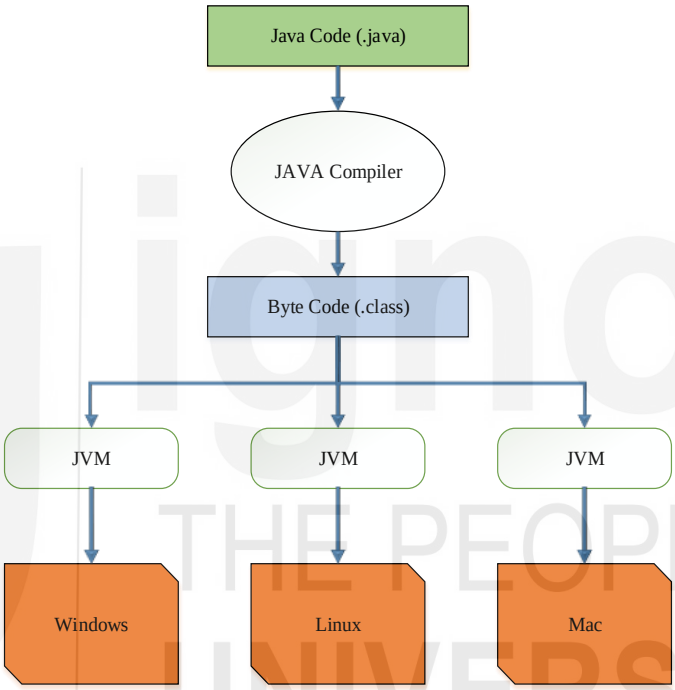


Figure 1.8: Java is Architecture-neutral

From figure 1.8, it can be seen clearly why Java programming is architecture-neutral. The only reason is its capability to generate the Bytecode, which runs on any JVM.

8) Interpreted and High performance

Java Bytecode can be executed on any machine that implements the Java Virtual Machine(JVM). It is designed in such a way that it can be translated into native machine code for every high-performance using a just-in-time compiler. Java run-time systems that present this feature lose none of the benefits of the platform-independent code.

You can achieve high performance because its *Just-in-Time compilers* are faster than the standard JVM. It also facilitates running one time and catching the results for future uses. Also, it is possible to optimize the code over time. Therefore, we can say that it is a high-performance language.

9) Distributed

Java programming is designed for the distributed environment of the Internet because it handles TCP/IP protocols. Java programming also supports Remote

Method Invocation (RMI), which enables a program to invoke methods across a network.

The reason behind saying Java isa distributed language is that after compiling the Java program on one machine, it can be transferred on another machine to execute because of its Bytecode facility. It will be beneficial for users who wish to use Remote Computers to execute their programs. Further, it can also facilitate the Local and Remote locations programmers to work together on the same project.

10) Dynamic

Java programs carry substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time. This enables to link code safely and expediently dynamically. It may affect the robustness of the java environment, in which a tiny fragment of Bytecode may be dynamically updated on a running system. Here dynamic just means changing something at run time that is not explicitly coded in the source code.

Another way in which Java is dynamic is the class loader, in which you can load a new already compiled class at run time in a standard language supported way. Java reflection method can also be used to operate on classes and instances specified by strings. For example, even without loading a new code, you can decide which already-loaded class you can use Class.forName() to get the class and instantiate an instance. If the string holding the name is read at therun-time, there's no way to know which class you'd be instantiating at compile time.

1.3.1 Use of Java in Industry

Java programming language has a significant impact on industry, especially in the IT industry in India and globally. Many projects running for ICT support and services are developed using the Java programming language. In many applications, it is being used as HTTP request and HTTP response. A most famous " local server" with Java framework used is **tomcat**. It is the most popular Java application server. Another Java programming based tool for Big Data is HADOOP. It is an open-source tool built over Java programming and used to store and processa large volume of data. Java programming is used in many scientific applications, for example : MATLAB is one of the most famous scientific applications, its both front-end and back-end is written in Java. Many simulators including ClodSim are developed using Java technologies.

The Java programming language is being used to design and develop many web applications and mobile apps that may run on a single device or may be distributed among servers and clients in a network. Twitter is a well popular free social networking and micro-blogging Java-based service application being used widely. Many other applications like Operaminin, Signal, and Wikipedia Search are developed using Java programming language.

1.4 JAVA DEVELOPMENT ENVIRONMENT AND TOOLS

Java is a general-purpose computer programming language which is class-based, object-orientedandpossess many other features. Its applications are typically compiled to Bytecode and can execute on any java virtual machine (JVM). The latest version of Java is JDK 18, recently released in the year 2022.

Creating, compiling, and editing Java programs is flexible. You can extend it from one programming environment to another easily. In general, there are two basic approaches; a command-line environment, where the user types command and the computer responds, and another is an integrated development environment (IDE) in which we use the keyboard and mouse to interact with well designed graphical user interface(GUI) for writing, compiling and running the Java projects.

This section will discuss about the Java development environment and related considerations. More about IDE and execution of Java program you will learn in the course MCSL-210

1.4.1 JDK

JDK is an acronym for Java Development Kit. It is a software development environment which is used to develop Java applications. It physically exists and contains JRE along with development tools.

JDK is an implementation of any one of the below java platforms released by Oracle Corporation, which are as follows.

- Standard Edition Java Platform
- Enterprise Edition Java Platform
- Micro Edition Java Platform

In general, JDK contains a private Java Virtual Machine (JVM) and a few other resources like interpreter/loader, java compiler, archivers (jar files) and Javadoc for the documentation generator to complete the development of Java Applications. Its software can be downloaded from the link: https://www.oracle.com/java/ technologies/ downloads/ as given.

1.4.2 JRE

The Java Runtime Environment (JRE) is a software layer that runs on top of a computer's operating system software and provides the class libraries and other required resources to run a specific java program. JRE is one of the three interrelated components for developing Java programs along with JDK and JVM.

The JDK and JRE are the two components that interact with each other to create a sustainable run-time environment that enables the seamless execution of Java-based applications in virtually any operating system.

JRE run-time architecture generally uses the ClassLoader, Bytecode verifier, and Interpreter. ClassLoader dynamically loads all the necessary classes to run a Java program. It also automates the loading of classes as per requirement, so it can also effectively utilize the memory. Bytecode verifier checks the format and accuracy of Java code before it passes to the interpreter. And finally,the interpreter helps to run the java program natively on the underlying machine.

1.4.3 JIT and JVM

JIT stands for *just in time compiler* and is an integral part of Java Virtual Machine(JVM) and Garbage Collector. It is capable of compiling the Java code into machine code, which can significantly improve the performance of Java Applications.

While the compilation of the codes, it is not guaranteed which code will be compiled. JIT usually compile the code as per its requirement to JVM, and once

method execution crosses the certain limit (of approx. 10K execution), then it will be converted into machine code.

To run a simple program in Java programming, we have to use the Java compiler to convert the source code into Java class code. And then, after that, JVM (is responsible for running the program.

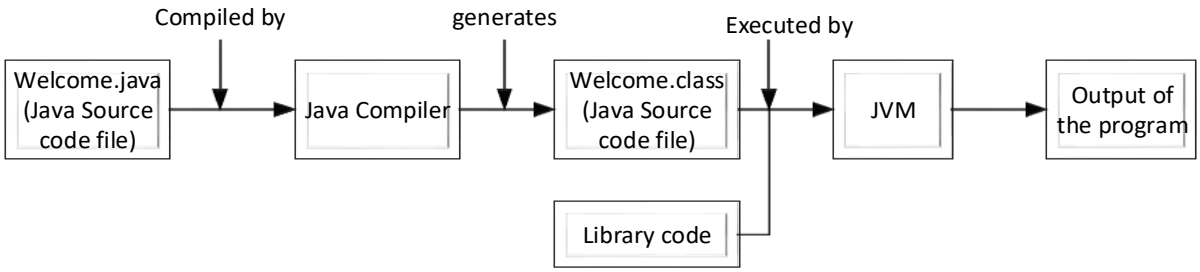


Figure 1.9: Flow of Java Program Execution

The role of JVM can be understood that a program written in a high-level programming language cannot run directly on any computer. First, it has to be translated into machine language using a compiler program. After being translated, it can run any number of times on one type of machine only. Another alternative way to compile the program is an interpreter, which translates the program instruction-by-instruction. It also has the capability to enable a machine language program to run from one computer to a completely different type of computer.

The designer of Java utilized the capability of both compilation and interpretation. Programs written in Java are compiled into machine language, but it is a machine language for a computer that does not really exist. This is what we call a "virtual" computer and is known as Java Virtual Machine(JVM).The machine language for the JVM is known as Java Bytecode. This code enables us to run the program on any computer.

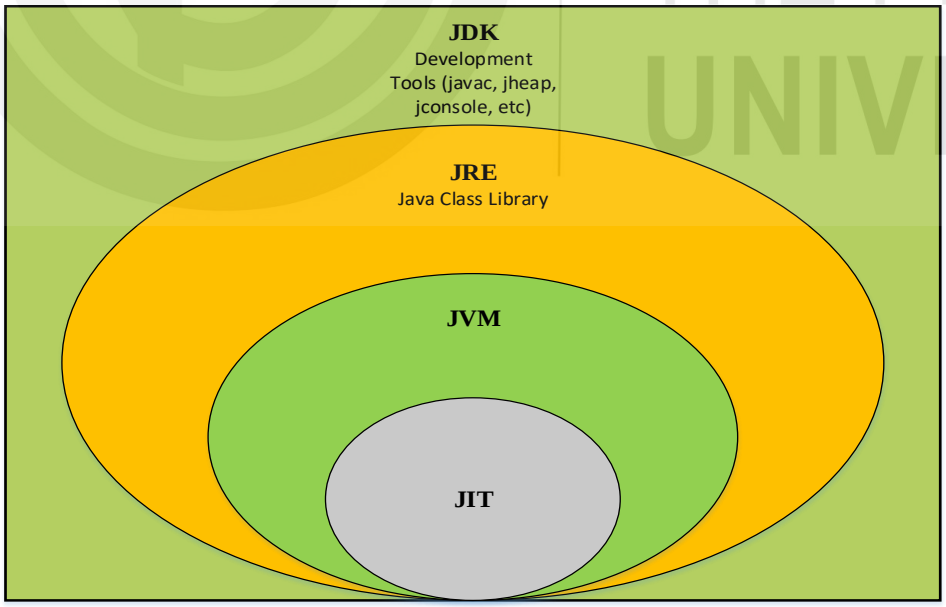


Figure 1.10: JIT, JVM, JRE and JDK in Java

Java Development Kit (JDK) is enriched with many features; it has numerous libraries, tools, and frameworks. Once you run a Java program, it can be executed on any platform, which saves time. It also facilitates installing the Java platform on various Windows, Unix, and Mac operating systems.

In figure 1.10, we can see that every module of Java comes under JDK. JVM can be seen as an instance of JRE while executing the Java program, and it is widely known as a run-time interpreter. JVM is the cornerstone on top of which Java technology is built. It is a component which makes Java programming hardware & platform-independent. Just-in-time (JIT) compiler is the part of JVM that is used to speed up the execution time. It compiles the parts of the byte code simultaneously that have similar functionality.

1.5 ENVIRONMENT SETUP OF JAVA APPLICATION DEVELOPMENT

If you are writing your Java program in any editor like notepad etc. and going to run it with the help of command-line execution,you have to follow the step as given below.

Step1: Write a simple Java program; for example the Java code as given below.

```
class Welcome
 {
 public static void main(String[] args)
  {
    //Display a welcome message on console.
    System.out.println("Welcome to Java.");
  }
}
```

Let's try to know the meaning of key terms used in the above program: class, public, static, void, main, String [] is, System.out.println().

- class is a keyword which is used to declare a class in Java.
- public keyword is an access modifier(specifier) thatdecides the visibility of the method or data member. The public means it is visible to all.
- static is a keyword and if we declare any method as static, it belongs to the class. The core advantage of the static method is that there is no need to create an object to invoke the static method. The JVM executes the main() method, before creating any object, so it doesn't require to create an object to invoke the main() method, which also saves the memory.
- void is the return type of the method. It means that this method doesn't return any value.
- main represents the starting point of the program.
- String[] args or String args[] is used for passing the command-line argument.
- System.out.println() is used to print a statement. Here, *System* is a class, *out* is an object of the *PrintStream* class, *println()* is a method of the *PrintStream* class.

Step2: Now save the program using its class name as file name; for example, for the above program, it will be Welcome.java

Step3: Compile the program using the command given below.

$$\rightarrow javacWelcome.java$$

Step4: Run the program using the command by writing Java and then name of the class file. For example

$$\rightarrow javaWelcome$$

Then it will print the output on the console like “Welcome to Java.”

Another way to write and run the java programs is to use the Integrated Development Environment (IDE), in which everything you need to create, compile, and run the program with the help of a single application. IDE also provides a graphical user interface.

NetBeans and ECLIPSE are two free and popular IDEs for developing Java programs. These IDEs are pretty simple and easy to understand. You are recommended to use one of them for developing the java programs.

ECLIPSE:

The ECLIPSE IDE (integrated development environment) is one of the well-known java programming platforms. In the year 2022 itself, it has approximately one million downloads per month, making it one of the leading IDEs. ECLIPSE has many pre-packages and plug-ins for the support of java programmers.

NetBeans:

NetBeans IDE is a free, open-source integrated development environment for application development on Windows, Mac, Linux, and Solaris operating systems. It provides intelligent overviews to help you manage, understand, and manage your applications, including out-of-the-box support for popular technologies. NetBeans IDE offers tools for Java web, enterprise, desktop and mobile application development. This is the IDE which supports the latest version of JDK, Java EE and JavaFX.

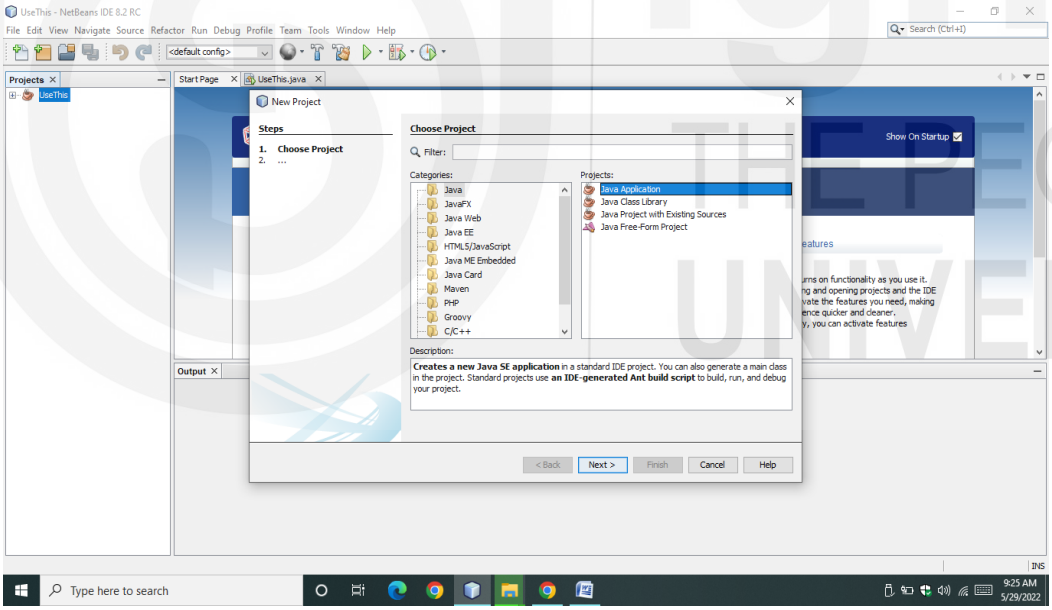


Figure 1.11: Netbeans IDE

A few key features of IDEs are as follows, which motivate programmers to use it.

- **Code completion or code insight**

The IDE is capable of providing the hint for keywords and function names which is quite crucial. It also helps us to remove the typographical errors and provide the suggestion with a list of appropriate functions based on the situation.

- **Resource management**

IDE is able to manage the resource effectively, especially the one required for library and header files. An IDE should be aware of any required resources so that

errors can be spotted at the development stage, not at the time of compiling or building.

- **Debugging tools**

The program written using IDE is easy to debug; it may be able to give the variable values at certain points during the program's execution.

- **Compile and build**

In the programming language that requires a compile or build stage, IDEs translate the code from high-level language to the object code of the targeted platform. In general, IDE specializes in one language or a set of similar languages.

Advantages of using IDE: IDE can be advantageous in many aspects while writing and running a program. A few key advantages are as listed below.

1. Less time and effort

The main purpose of an IDE is to make application development faster and easier. Its tools and features are developed to help programmers to organize resources, prevent mistakes, and provide shortcuts.

2. Enforce project or company standards

IDEs proved the same standered development environment for all its users. This provides a chance to a group of programmers to adhere to a standard way of doing things. Standards can be further enforced if the IDE offers predefined templates or if code libraries are shared between different team members/teams which is working on the same project.

3. Project management

Project management using IDE can be in twofold. First, many IDEs have documentation tools that either automate the entry of developer comments or may force developers to write comments in different areas. Second, simply by having a visual presentation of resources using GUI , it makes a lot easier to know how an application is laid out instead of traversing the file system.

In the following figure1.12 you may see the screenshot of Netbeans IDE in which the program given above is written and executed.

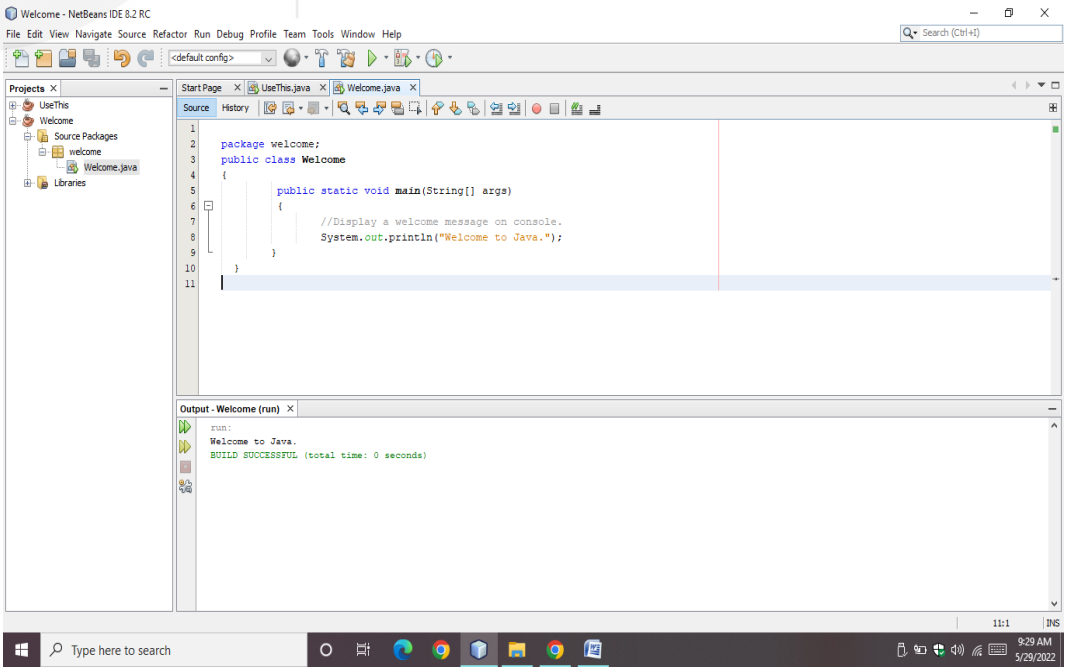


Figure 1.12: Writing and Running Java Program in Netbeans IDE

You will learn more about writing Java programs and running them in the MCSL-210 course. Here in this unit, you will have a glimpse of how to write Java program. In the next unit, you will learn more about basics like variables/literals, data types, and operators and use them in writing Java programs.

☞Check Your Progress – 2

1) What is JDK ?

……

……

……

2) What will be the output of the following code segment?

```
public class Example1
{
   public static void main (String args[])
   {
System.out.println(" ************************************");
System.out.println(" Let us Learn " +"Java Programming");
System.out.println(" ************************************");
System.out.println(" Use IDE to write Java Programamming" );
   }
}
```

……

……

……

3) What is the JIT compiler?

……

……

4) What aresome available IDEs for developing Java Applications?

……

……

1.6 SUMMARY

Object-oriented programming takes a different approach to solving problems by using a program model that mirrors the real-world problems. It allows you to decompose a problem into subgroups of related parts easily. In this unit, we have discussed the basic features of object-oriented programming. Java is a very popular programming language because of its features. Some important features of Java, like platform-independent, portable, high performance, and distributed, makes it popular among application developers. In this unit, we also discussed the Java development environment and tools.

1.7 SOLUTIONS/ANSWER TO CHECK YOUR PROGRESS

☞Check Your Progress – 1

1) A class is a template or blueprint that defines data members and methods (member functions)of objects of the same type. On the other hand, an object is a real word entity that participates in problem-solving. Each object poses its attributes and features. Objects have the states and behaviours, while a class describes the behaviour/state of the object of its type support. When we look at real-world examples, cars, dogs, humans etc. are the objects. Dog shows its state and behaviour like name, breed and color are the state and barking, wagging the tail, and running are the dog's behaviour.

2) Encapsulation is a key concept of object-oriented programming. It is a process of binding the data members and methods together. The power of encapsulated code is that everyone outside of the class knows how to access and use the methods without thinking about implementation details.

3) Polymorphism is the ability of an object to take many forms. It is a feature that allows one interface to be used for a general class of actions. It means different actions can be performed using a single interface. In programming languages, Polymorphism is the ability of an object to take on many forms.

   The Polymorphism is of two types:

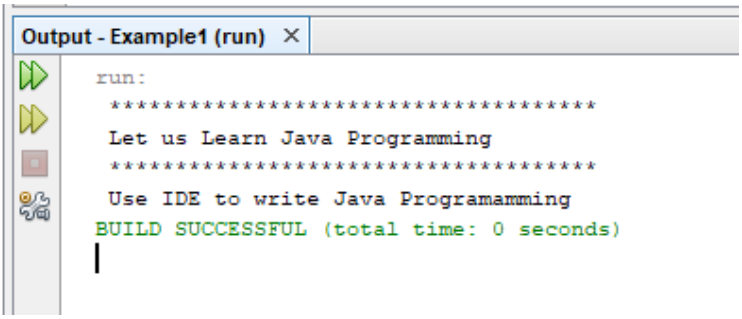
   - Static Polymorphism
   - Dynamic Polymorphism

☞Check Your Progress – 2

1) Java Development Kit (JDK) has the bundle of capabilities to run the Java programs smoothly and effectively. A few important points related to JDK are listed below.

   - JDK- JDK is the package that contains various tools, Compiler, Java Runtime Environment, etc. It is used for making java programs, we need some tools that are provided by JDK (Java Development Kit).
   - JRE - JRE provides the environment forthe execution of the Java program. JRE contains a library of Java classes + JVM.
   - JVM (Java Virtual Machine) : It is a part of JRE that executes the Java program at the end. JVM convertsBytecode into machine-executable code to execute on a specific OS(hardware).

2) **Output:**



3) JIT compilers can access dynamic run-time information, whereas a standard compiler does not and can make better optimizations like inlining functions that are used frequently. The JIT compiler compiles Bytecodes into native machine code at run time, which helps in improving the performance of Java programs.

4) An IDE for Java applications development is a software that allows Java developers to write and debug Java programs easily. It is a collection of various programming tools, accessible via a single interface along with several helpful features for developers, such as code completion and syntax highlighting. Some of the most popular Java IDEs are NetBeans, ECLIPSE,**IntelliJ** IDEA, and Codenvy.

1.8 REFERENCES AND FURTHER READING

- Herbert Schildt, "Java The Complete Reference", McGraw-Hill,2017.
- Savitch, Walter, " Java: An introduction to problem solving & programming", Pearson Education Limited, 2019.

ignou
THE PEOPLE'S
UNIVERSITY

UNIT 2 BASICS OF JAVA

Structure **Page No**

2.0 Introduction 26
2.1 Objectives 26
2.2 Data Types 27
2.2.1 Integers and Floating Point Data Type 29
2.2.2 Character and Boolean Types 30
2.2.3 Enumerated Types 31
2.3 Unicode 32
2.4 Variables 33
2.4.1 Java Keywords 33
2.4.2 Variables and Literals 35
2.5 Operators 38
2.6 Statements and Expressions 45
2.7 Example Program Using Scanner Class 49
2.8 Summary 50
2.9 Solutions/ Answer to Check Your Progress 51
2.10 References/Further Readings 53

2.0 INTRODUCTION

Java is an object-oriented, general-purpose programming language. The popularity of Java programming language is more because of its features. Basic features of Java include security, robustness, distributed, multithreading, and a rich library for Graphical User Interface (GUI). However, one of its main characteristic is : being Architecture-Neutral and platform- independent, which we have studied in the previous unit of this Block. To write a program in any programming language, first of all, we need to know the basic facilities that the programming language provides. For example, support of different data types, operators, basic programming constructs for decision making or constructs of looping etc., are the basic facilities Java provides. In this unit, we will learn to use basic building blocks of Java, such as data types, java keywords, variables, literals, operators, statements and expressions. To improve your programming concepts, you can move towards learning of declaration and initiations of different data types and variables. An operator is a symbol that informs the compiler to perform some operation on the given data. Java provides several operators. Java expression is built from variables using different operators. In addition to expression, you will learn statements in Java, which are equivalent to sentences in natural languages. A statement is a complete unit of execution. At the end of this unit, you will be able to write java programs using the scanner class.

2.1 OBJECTIVES

After going through this unit, you should be able to:

- describe the different data types in Java,
- declare and initialize variables in Java,
- know reserved words in Java,

- define Variables and Literals,
- use appropriate Operators in writing programs,
- use statements and expressions in program, and
- write Java programs using Scanner class.

2.2 DATA TYPE

In programming, we use variables to store values. A variable stores values according to its declared type. Every programming language provides some primitive data types for storing values and evaluating expressions. Java provides primitive data types(basic types), which are **statically-typed**, which means you need to declare all variables before using them.

There are eight basic data types in Java, which include four types of integer values in 'int' data type, two types of fractional values in 'float' data type, character data type in 'char' and true or false values in 'boolean' data type. The int, float, char and boolean are keywords that can not be used as variable names. Java primitive types are given in Table 2.1

Table 2.1: Java Primitive Data Types

Data Type	Size in byte	Description	Example	Groups	Range
byte	1	byte-length integer	15, 25	Integer	-128 to 127
short	2	short integer	550, 986		-32,768 to 32,767
int	4	Integer	9, 95, 378, 465		-2^31 to 2^31-1
long	8	long integer	989L, -545L		-2^63 to 2^63-1
float	4	Single-precision floating point	809.5f, -232.2f,	Floating point numbers	-3.4e38 to 3.4e38
double	8	Double-precision floating point	2289.5,-4535.5, 27.6E5		-1.7e308 to 1.7e308
char	1	A single character	'a', '9', 't' 'A'	Characters	true or false
boolean	1 bit	A boolean value (0 or 1)	true or false	Boolean	'\u0000' to '\uffff'

Following is the Java program to display the size and range of Java primitive data types.

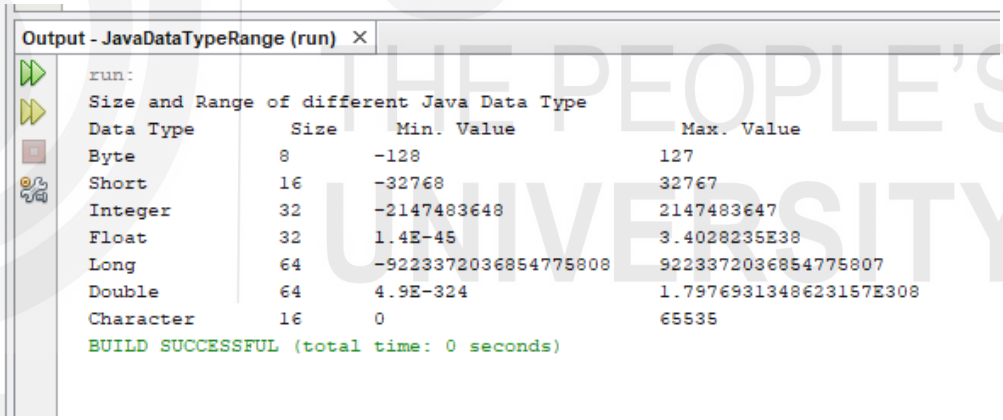
Example Program:

```
package javadatatyperange;
public class JavaDataTypeRange
{
public static void main(String args[])
{
System.out.println("Size and Range of different Java Data Type ");
```

```
System.out.println("Data Type\t Size\t  Min. Value\t\t  Max. Value\t");
System.out.println("Byte\t\t" + Byte.SIZE + "\t" + Byte.MIN_VALUE
                + "\t\t\t" + Byte.MAX_VALUE);
System.out.println("Short\t\t" + Short.SIZE + "\t" + Short.MIN_VALUE
                + "\t\t\t" + Short.MAX_VALUE);
System.out.println("Integer\t\t" + Integer.SIZE + "\t" + Integer.MIN_VALUE
                + "\t\t" + Integer.MAX_VALUE);
System.out.println("Float\t\t" + Float.SIZE + "\t" + Float.MIN_VALUE
                + "\t\t\t" + Float.MAX_VALUE);
System.out.println("Long\t\t" + Long.SIZE + "\t" + Long.MIN_VALUE + "\t"
                + Long.MAX_VALUE);
System.out.println("Double\t\t" + Double.SIZE + "\t" + Double.MIN_VALUE
                + "\t\t" + Double.MAX_VALUE);
System.out.println("Character\t" + Character.SIZE +"\t" +
(int)Character.MIN_VALUE
                + "\t\t\t" + (int)Character.MAX_VALUE);
 }
}
```

If you write and run(execute) the above Java program , it gives the following output.

Output:



In the above programe you can see that to print the **size** of a data type ***(name of datatype).SIZE*** is used. To print the minimum value, a data type can store (***Name of data type). MIN_VALUE*** is used. Similarly, to print the maximum value, a data type can store ***(Name of data type). MAX_VALUE*** is used. You can notice that the first word of the data type is written in capital. Also Character.MIN_VALUE and Character.MAX_VALUE is converted into an integer (here, type casting is done). "/t"(tab) is used for proper alignments of output, the use of "/t" is similar to pressing the tab on our keyboard.

In most of programming languages, the format and size of primitive data types depend on the platform on which program code is running. But in Java , the size and format of its primitive data types is specified, and it frees programmers from the system dependencies while writing program and using the data types. For example,

an **int** is always 32 bits in Java, regardless of the platform on which the Java program runs. This allows programs to be written with the guarantee to run *without porting* on any machine architecture. All the primitive data types can be divided into four major groups, as shown in Table-1.1. Now let us learn the use of these primitive data types in detail.

2.2.1 Integers and Floating Point Data Type

Integers and Floating-point numbers are used to store numeric data values. Java has six numeric data types, as shown in Table 1.1. They differ in size and precision of the numbers they can hold. The size of a data type means how many bits are needed to represent that data type. The range of a data type represents how small/big value can be stored in that data type with the precision of numbers.

Integers

To store an integer value, you need to declare a variable of integer type, and *int* keyword is used for it. The integer variable can be declared as follows:

```
int  age;
```

The data type of the variable declared above is *int*, and the name of the variable is age. The declaration ends with a semicolon. You can store only an integer value in variable age. There are four types available in Java for integer values. These four types are signed variables that can store either a positive or negative value, as shown in Table 2.2.

Table 2.2: Java integer types

Sl. No.	Data Type	Data Range	Bytes
1.	Byte	-128 to + 128	1
2.	Short	-32768 to + 32767	2
3.	Int	-2147483648 to 2147483647	4
4.	Long	-9223372036854775808 to +9223372036854775808	8

As a programmer, you need to choose a variable of any of the four types according to your need in the application. You may use byte if you are expecting a very small integer value for your variable. For example, to store student age, you may use a variable of type byte. The *int* is mostly used to store an integer value; for a bigger range of integer values, the long integer type is used. Let us see the four types of integers declared below:

```
byte age;
short year;
int No_of_Smart_City;
long AccoutNo;
```

The above declarations of variables of integer types are according to the range of values they can store.

Floating Point Numbers

Those numbers containing fractional parts are called real numbers or floating point numbers. Table 2.3 shows that real numbers can also be divided into two categories in Java: float and double. The double type is used where more accuracy is required for the fractional part.

As integral values are stored in integer variables, non-integral values are stored in floating-point variables. In other words, fractional values or values with a decimal point are stored in floating-point variables. If you have to store a value 16.045 it can be stored in a floating-point variable. The number of decimal places after the decimal point is called the precision. There are two primitive data types, *float* and *double,* for floating-point data.

Table 2.3: Two categories of real numbers

Sl. No.	Data Type	Data Range	Bytes
1.	Float	-3.4×10^{38} to $+ 3.4 \times 10^{38}$ (OR) -3.4 E38 to + 3.4 E38	4
2.	double	-1.7×10^{308} to + 3.4 E38 (OR) -1.7 E308 to 1.7 E308	8

As an example, the notation of 10^{35} can also be presented as E35. In table 2.3 , 10^{38} is presented as E38 in the table. The declaration of a floating-point variable is as follows:

float price = 156. 50F;

This statement declares a variable *price* by initializing it with a floating point constant 156.50F. if F is not appended to the constant, then the value is, by default, double type.

The declaration for the double type is the same as that for the float type but will not have the F towards the end of the constant. The following is a variable earning_ratio declared as double.

double earning_ratio = 52225.55557;

2.2.2 Character and Boolean Types

Characters

In Java, the character data type- *char* holds a single character. A char is a single character: a letter, a digit, a punctuation mark, a tab, a space or something similar. Each character is a number or character code belonging to a character set, which is an indexed list of symbols. For example, **ASCII** (American Standard Code for Information Interchange) is a character set. The ASCII character set ranges from **0 to 127 and** needs 8 bits(1 byte) to represent a character. In Java, character variables are stored as **Unicode,** which uses 2 bytes of memory. The character variable is declared and initialized as follows:

Char MyChar = ‘P’;

The above statement initializes the variable MyChar with a Unicode character representing P. The character constant assigned to a character variable must be enclosed within single quotes. “The character enclosed within single quotes is treated as a character constant and that without single quotes is treated as a variable by the compiler”.

Apart from character variables and constants, Java also provides Escape Sequences character constants. All the character constants are specified inside a pair of single quotes. Some characters cannot be entered directly using the pair of quotes. A backslash followed by a character is used to indicate to the compiler that it has a different meaning. Such sequences of characters are called Escape Sequences or

Escape Characters. The sequence is typed as two characters with no blank in between the characters. The use of backslash suppresses the meaning of an escape sequence character. To display double quotes inside a string, \" is used so that it will display double quotes and not the end of the string. Though the escape sequence is of two characters, it is treated as a single character. A list of Escape Sequences is shown in Table 2.4.

Table 2.4: Escape Sequences

Sl. No.	Escape Sequence	Meaning
1.	\'	Single quote
2.	\"	Double quote
3.	\\	Backslash
4.	\b	Backspace
5.	\f	Form feed
6.	\n	New line
7.	\r	Carriage return
8.	\t	Horizontal Tab

Booleans

In Java, Boolean variables are used as logical variables with either true or false values. The Boolean variable has only two values: true or false. These two values are Boolean constants. Different attributes of the boolean data type are given in *Table2.5*.

Table 2.5: Boolean Data Type

Type	Values	Default	Size
boolean	true, false	false	1 bit used in 32 bit integer

In Java a boolean variable can be declared as follows:
boolean aTeacher = true;

In the above statement, a boolean variable a Teacher is declared and initialized with the boolean value true.

2.2.3 Enumerated Types

The constants are those variables whose values are unchangeable. An enumerated data type is a special data type in Java . It is a user-defined data type which consists of a set of predefined values.

Enumerations: An enum is a special "class" used for representing a group of **constants**, were added to the Java language in JDK5. In Java, enumeration defines a class type. It is created using the enum keyword. Each enumeration constant is *public, static* and *final* by default. In Java, final variables are also constant variables. About the final variable, you will learn later in this course. The enumeration variables can be used and declared in much the same way as a primitive variable.

An enumeration can be defined simply by creating a list of *enum* variables. You may create an enum using the enum keyword, and you have to separate the constants using a comma. The general format of an enumerated type declaration:

enumTypeName { One or more enum constants }

Example of creating enumeration :

enum Size { LOW, MEDIUM, HIGH }

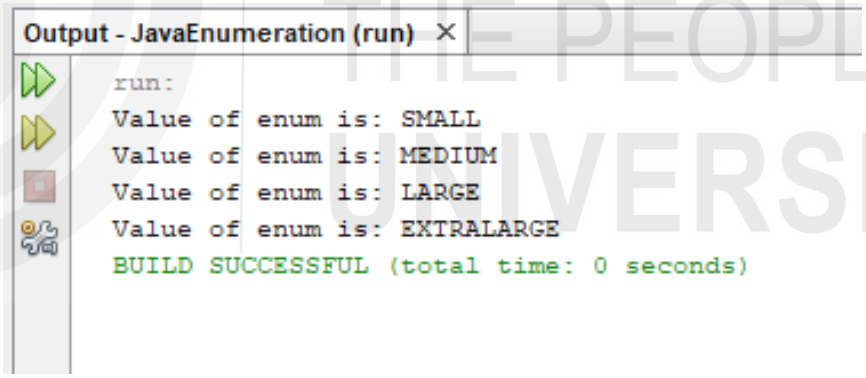
You can access enum constants with the **dot(.)** syntax:

Level myVar = Size.MEDIUM;

Program:

```
package javaenumeration;
enum Size {   SMALL, MEDIUM, LARGE, EXTRALARGE }
public class JavaEnumeration
{
  public static void main(String[] args)
  {
    System.out.println("Value of enum is: " + Size.SMALL);
    System.out.println("Value of enum is: " + Size.MEDIUM);
    System.out.println("Value of enum is: " + Size.LARGE);
    System.out.println("Value of enum is: " + Size.EXTRALARGE);
  }
}
```

Output:



Another Example:

Let us try to define PGDCA_NEW Ist semester Courses as enumeration.

enum PGDCA_NEW_Courses-I

{ MCS-201, MCS-202, MCS-203, MCS-204, MCSL-205}

Here identifiers such as MCS-201, MCS-202, MCS-203, MCS-204, and MCSL-205 are called enumeration constants. These are public, static, and final by default.

2.3 UNICODE

In Java to fully understand the char type, you need to know about the Unicode encoding scheme. Before Unicode, many other coding standards, such as ASCII , ISO 8859-1, KOI-8, GB18030 etc. were used in different regions of the world. Having the use of different coding standards caused two main problems:

1. A code value corresponds to different letters in the different encoding schemes.
2. The encodings in the case of those languages which have large character sets have variable lengths.

To solve these problems, Unicode was designed to solve these problems. With the attempt of the unification effort started in the 1980s, it was decided to have a fixed 2-byte code to encode all characters used in all languages in the world. The "Unicode 1.0 was released, using slightly less than half of the available 65,536 code values. Java was designed from the ground up to use 16-bit Unicode characters, which was a major advance over other programming languages that used 8-bit characters".

Unicode is **a universal character encoding standard**. It is "an international encoding standard for use with different languages and scripts, by which each letter, digit, or symbol is assigned a unique numeric value that applies across different platforms and programs." This standard includes roughly about 100000 characters to represent characters of different languages. While ASCII uses only 1 byte, Unicode uses 4 bytes to represent characters. Hence, it provides a very wide variety of encoding. "Unicode covers **all the characters for all the writing systems of the world, modern and ancient"**. It also includes technical symbols, punctuations, and many other characters used in writing text. The most widely used forms of Unicode are:

- UTF-32, is most appropriate for single-encoding characters. It uses 32-bit code units, each of which is used for storing a single code point.
- UTF-16 uses one or two 16-bit code units for each code point. It is the default encoding for Unicode.
- UTF-8, with one to four 8-bit code units (bytes) for each code point.

2.4 VARIABLES

In Java there are some predefined keywords; they are particular words that act as a key to a code. These keywords are also known as reserved words by Java, so you can not use them as a variable or object name or class name. As you know, a variable is a titled portion of the memory used to store data that is required in any program, perhaps even in other than Java programming language also. A variable can store only one type of data. In this section, you will learn more about java keywords, variables, and literals.

2.4.1 Java Keywords

A keyword or reserved word in Java are predefined word with a special meaning and cannot be used as a user-defined variable or identifier. The keywords are used by the compiler, and if you use them as variable names, the compiler will generate errors. The following Table 2.6 gives a list of Java keywords and their defined purpose. Remember that these keywords cannot be used as variable names or function names in any Java program.

Table 2.6: List of Java Keywords

Keyword	Purpose
abstract	Used to declare that a class or method is abstract
assert	Used to specify assertions
boolean	Declares that variable is boolean type

break	Used to exit a loop before the end of the loop is reached
byte	Tossed to declares that variable is byte type
case	for representing different conditions in a switch statement
catch	Used for handling exceptions
char	To declare that variable is character type
class	For a class definition
const	It is a keyword reserved by Java, but now it is not in use
continue	Use for prematurely returning to the beginning of a loop
default	For defining default action in a switch statement
do	For beginning a do while loop
double	To declare a variable of double type
else	Used for indicating the code to be executed when if condition executes to false
extends	Used to Specify the class base from which the correct class is inherited
final	It declares that a class may not be extended or that a field or method may not be overridden
finally	Used to declare a block of code guaranteed to be executed
float	For defining a floating point variable
for	To start a for loop
goto	This keyword is reserved by Java but now it is not in use
if	Used to represent a conditional statement
implements	Declares that this class implements the given interface
import	Permits access to a class or group of classes in a package
instance of	Tests whether an object is an instance of a class
int	For declaring an integer variable
interface	Signals the beginning of an interface definition
long	For declaring a long integer variable
native	It declares that a method that is implemented in native code
new	To allocates memory to an object dynamically
package	For defining the package to which this source code file belongs
private	Used for declaring a method or member variable to be private
protected	To declare a class, method or member variable to be protected
public	To declare a class, method or member variable to be public
return	To return a value from a method
short	For declaring a short integer variable
static	To declare that a field or a method belongs to a class rather than an object
strictfp	For declaring a method or class must be run with exact IEEE 754 semantics
super	Used to refer to the parent of the current object
switch	Used to test for the truth of various possible cases
synchronized	Used to indicate that a section of code is not thread-safe
this	Use as a reference to the current object

throws	It declares the exceptions thrown by a method
transient	Used to make data non-serialized
try	Used in exception handling. It attempts an operation that may throw an exception
void	It declares that a method does not return a value
volatile	Indicates to the compiler that a variable changes asynchronously
while	For beging of a while loop

Java is case-sensitive, so even though the break is a keyword, Break is not a keyword at all. At the time of designing Java, its designers decided to **make it case-sensitive because,** first of all, this enhances the readability of code; secondly, it reduces the compilation time and increases the efficiency of the compiler. All the Java technology keywords are in lowercase. The words true, false, and null are reserved words. Java keeps on adding new keywords as per need. **Java 1.4** adds the **assert** keyword to specify assertions. As we have seen above in table 2.6, each keyword has a specific and well-defined purpose.

2.4.2 Variables and Literals

Variables

In programming, we use variables to store data that is used in any program. Like other languages, Java also expects the variables to be declared and used. A variable is a titled portion of the memory. You can use a variable to store only one type of data. For example, if a variable is declared to store an integer, it can store only integer data. Java predefines the type of data stored in a variable. Java variables are declared by a name and their type. The name given to a variable is called an identifier. There are a few guiding rules for naming a variable, which is as follows:

- The name of a variable may be of any length.
- The name of a variable must start with an alphabet, underscore (_) or a dollar symbol ($).
- The remaining characters of the variable name can be alphabetic or numeric.
- Special characters such as @,% etc., are not allowed as part of the name of variables.
- Underscore (_) can be used to connect two words such as first _ name, student_marks etc.
- As Java is case-sensitive hence the variables Age and age are treated as two different variables.
- Blanks or tabs may not be used in variable names.

As a programmer, you are free to give any names to the variables, and only you cannot use the keywords defined by Java as the name of the variable. Also, it is advisable that naming a variable should reflect the purpose for which it is declared in a program. The variable names should be meaningful such as book_title, name, username and password, age, marks etc. This makes your program more readable. In Java, the naming convention followed is:

- the variable name starts with a lowercase alphabet
- the remaining words, the first alphabet is an uppercase alphabet

Declaring and Initialization Variables

As you are aware, before using any variable, you have to declare it. After it is declared, you can then assign values to it. Also you can declare and assign a value to a variable at the same time.

Java actually has three kinds of variables:

- Instance variables
- Class variables
- Local Variables.

You will learn more about instance variables and class variables in unit 4 of this Block. Instance variables are those variables which are used to define the attributes of a particular object. Class variables are similar to instance variables, except their values is the same for all the objects of a class rather than having different values for each object of the class. Local variables are declared for the local consumption of the methods; for example, for index counters in a for loop, temporary variables to keep some values or to hold values that you need only inside the method definition itself.

Variable Declarations

In Java, a general variable declaration looks like the following:

datatype identifier [= default value] {, identifier [= defaultvalue] };

Let us consider the following variable declarations:

byte b;

int age;

boolean male;

char c;

Also, it is possible to declare multiple variables of one type in one expression, as given below:

int age, enrollnum, accountno;

Variable declarations can be anywhere in your code as long as they precede the first use of the variable. However, it is common practice to place the declarations at the top of each block of code. Variable names in Java can only start with a letter, an underscore (_), or a dollar sign ($). They cannot start with a number. After the first character, your variable names can include letters, numbers, or a combination of both.

As you know, the Java language is case sensitive, implying that the variable x in lowercase is different from variable X in uppercase. In Java, a variable name cannot start with a digit or hyphen. The names of the variables should be kept in such a way which may help you understand what is happening in your program. Therefore you should keep the name of your variables intelligently or according to their role in the program. Your variable's name cannot contain any white space. There is no limit to the length of a variable name in Java.

Suppose you want to begin a variable name with a digit, prefix underscore, with the name, e.g. _1Name. You can also use the underscore in the variable name to provide better readability and meaning, e.g. Student_age.

Variable Assignment and Initialization

Once you have declared the type of a variable, you can initialize it with some value.
variable name = some value

For example, consider the following example:
int Student_age;
Student_age = 21; // here the value 21 is assigned to the variable 'Student_age'

You will learn more about assignment operators later in this unit.

Literals

The constants or data values used in a program are called literals. Each literal is of a particular type. For example, the percentage of marks for a student is 78.25, which is a literal and fractional value, and its type is elaborated on in the subsequent section.

Primitive types in Java are called **literals**. A literal is the source code representation of a fixed value in memory. Literals are nothing but pieces of Java code that indicate explicit values. For example, "Hello IGNOU!" is a String literal. The double quote marks indicate to the compiler that this is a string literal. The quotes indicate the start and the end of the string, but remember that the quotation marks themselves are not a part of the string. Similarly, Character Literals are enclosed in single quotes, and it must have exactly one character. TRUE and FALSE are boolean literals that mean true and false. Number, double, long and float literals also exist there. See examples of all types of literals in the following *Table 2.7*.

Table 2.7: Examples of literals

Types of literal	Example
Number Literals	145, 104L, 777, 0XFF, 2.56F, 15e45, .36E-2
Boolean Literals	TRUE, FALSE
Character Literals	'c', '#', '5', \n, \\, \"
String Literals	"Java Programming"
Double Literals	16.5, 85.6, 96.4E8
Long Literals	54L
Float Literals	35.6f, 86.4E, 8F, 1.5F

Constants

Constants are used for fixed values. Their value never changes during the program execution. To declare constants in Java keyword final is used. The following statement defines an integer constant x containing a value of 25.

final int x =25;

☞ Check Your Progress – 1

1) What is literal in Java?

……

……

……

2) Write a program in Java to calculate the Area of a circle. Show the use of the 'final' keyword in your program.

……

……

……

3) What is boolean data type in Java?

……

……

……

In the next section, we will explore the different types of operators in Java with their meaning and uses in the programs.

2.5 OPERATORS

An operator is a symbol that informs the compiler to perform some operation on the given data. Java provides several operators. The operations can be classified into four categories: arithmetic, relations, logical and bitwise. Each category is explained briefly in the subsequent sections.

Arithmetic operators

You must be friendly with arithmetic expressions in mathematics like 'A – B'. In this expression A and B are operands and the subtraction sign ' – ' is the Operator. The same terminology is also used here in the programming language. The following table lists the basic arithmetic operators provided by the Java programming language, along with their descriptions and uses. Descriptions of arithmetic operators are defined in Table 2.8(a). Operators are divided into two categories **Binary and Unary.** Addition, subtraction, multiplication etc. are binary operators and applied only on two operands. Increment and decrement are unary operators and are applied on a single operand.

Table 2.8(a): Description of arithmetic operators(Binary Operators)

Operator	Use	Description
+	A + B	Adds A and B
-	A – B	Subtracts B from A
*	A * B	Multiplies A by B
/	A / B	Divides A by B
%	A % B	Computes the remainder of dividing A by B

Table 2.8(b): Description of arithmetic operators(Unary Operator)

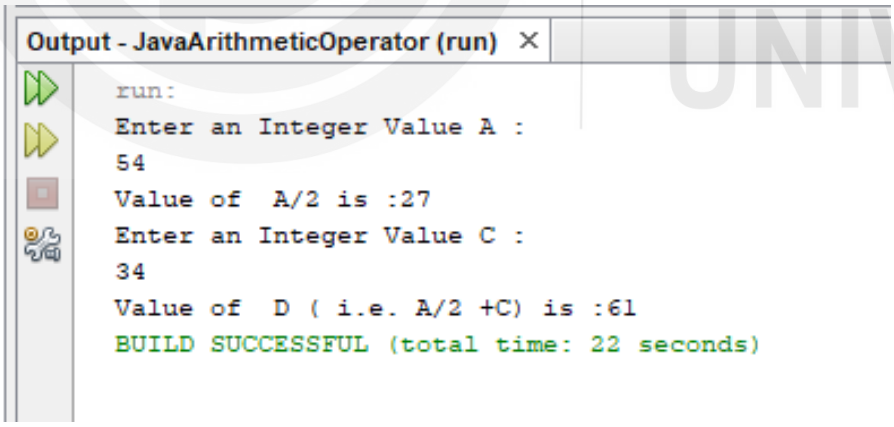
Operator	Use	Description
++ "Post-increment"	A++	Post-increment: The value is assigned before the increment is made, e.g. A = 5; B = A++; Then B will hold 5, and A will hold 6
-- "Post-decrement"	A--	Post-decrement: The value is assigned before the decrement is made, e.g. : A = 2; B = A--;

		Then B will hold 2, and A will hold 1.
++ "Pre-increment"	++A	Pre-increment: The value is assigned after the increment is made, e.g. A = 1; B = ++A; Then B will hold 2, and A will hold 2.
-- "Pre-decrement"	--A	Pre-decrement: The value is assigned after the decrement is made, e.g. A = 5; B = --A; Then B will hold 4, and A will hold 4.

Example Program: Use of Arithmetic Operators

```
package javaarithmeticoperator;
import java.util.Scanner;
public class JavaArithmeticOperator
{
public static void main(String[] args)
 {
int A,B,C,D,E;
Scanner s = new Scanner (System.in);
System.out.println ("Enter an Integer Value A : ");
A = s.nextInt ( ) ;
B = A/2;
System.out.println ("Value of  A/2 is :" + B);
System.out.println ("Enter an Integer Value C : ");
C = s.nextInt ( ) ;
D = B+C;
System.out.println ("Value of  D ( i.e. A/2 +C) is :" + D);
}
}
```

Output:

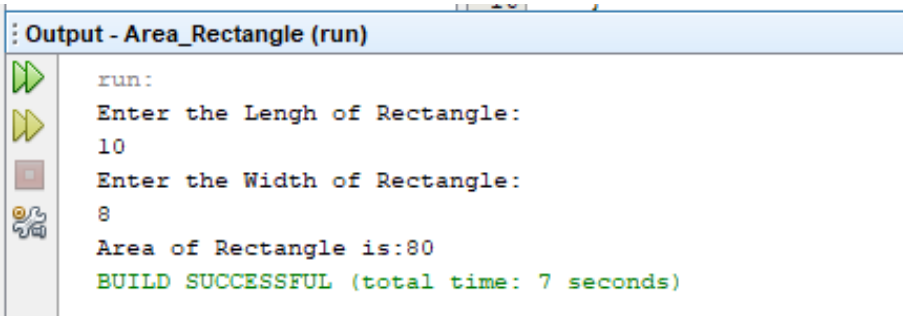


Example Proram: Find Area of Rectangle

```
package area_rectangle;
import java.util.Scanner;
public class Area_Rectangle
{
public static void main(String[] args)
 {
int Width, Length,Area;
Scanner s = new Scanner (System.in);
System.out.println ("Enter the Lengh of Rectangle: ");
Length = s.nextInt ( ) ;
```

```
System.out.println ("Enter the Width of Rectangle: ");
Width = s.nextInt ( ) ;
Area = Length * Width;
System.out.println ("Area of Rectangle is:" + Area);
}
}
```

Output:



Relational Operators

Relational operators, also called comparison operators, are used to determine the relationship between two values in an expression. As given in *Table 2.9*, the return value depends on the operation.

Table 2.9: Use of relational operators

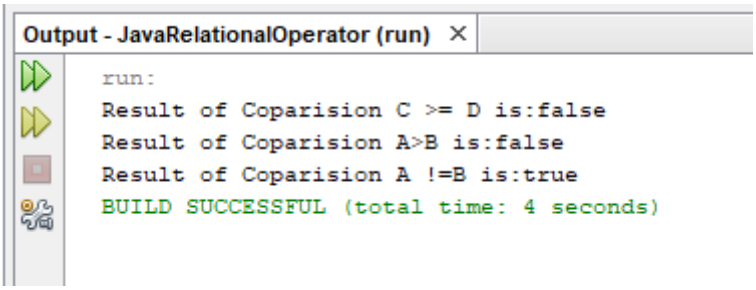
Operator	Use	Returns true if
>	A > B	A is greater than B
>=	A >= B	A is greater than or equal to B
<	A < B	A is less than B
<=	A <= B	A is less than or equal to B
==	A == B	A and B are equal
!=	A != B	A and B are not equal

Program: Use of Relational Operators

```
package javarelationaloperator;
public class JavaRelationalOperator
{
public static void main(String[] args)
 {

int A, B, C,D,E;
boolean i,j,k;
A = 6;
B = 4;
C = 5
D = 6
i = C> = D;
j = A<B;
k = A != B;
System.out.println("Result of Comparision C >= D is:"+i);
System.out.println("Result of Comparision A>B is:"+j);
System.out.println("Result of Comparision A !=B is:"+k);
}
}
```

Output:



Boolean Operators

Boolean operators allow you to combine the results of multiple expressions to return a single value that evaluates to either true or false. *Table 2.10* describes the use of each boolean Operator.

Table 2.10: Description of Boolean operators

Operator	Use	Description
&&	A && B	Conditional AND :If both A and B are true, result is true. If either A or B are false, the result is false. But if A is false, B will not be evaluated. For example, (A > 5 && B <= 10) will evaluate to true if A is greater than 5, and B is less than or equal to 10.
||	A || B	Conditional OR: If either A or B are true, the result is true. But if A is true, B will not be evaluated. For example, (A > 10 || B > 5) will evaluate to true if either A is greater than 10 or B is greater than 5.
!	! A	Boolean NOT: If A is true, the result is false. If A is false, the result is true.
&	A & B	Boolean AND: If both A and B are true, the result is true. If either A or B are false, the result is false, and both A and B are evaluated before the test.
|	A | B	Boolean OR: If either A or B are true, the result is true. Both A & B are evaluated before the test.
^	A ^ B	Boolean XOR: If A is true and B is false, the result is true. If A is false and B is true, the result is true. Otherwise, the result is false. Both A and B are evaluated before the test.

Bitwise operators

As you know, in computers, data is represented in binary (1's and 0's) form. The binary representation of the number 43 is 0101011. The first bit from right to left in the binary representation is the least significant, i.e., the value is 1 here. Each Bitwise Operator allows you to manipulate integer variables at bit level. *Table 2.11* given below, describes the use of the bitwise Operator with the help of a suitable example for each.

Table 2.11: Description and use of Bitwise operators

Operator	Use	Description
>>	A >> B	Right shift: Shift bits of A right by distance B, 0 is introduced to the vacated most significant bits, and the vacated least significant bits are lost.
<<	A << B	Left shift: Shift bits of A left by distance B , the most significant bits are lost as the number moves left, and the vacated least significant bits are 0.
>>>	A >>> B	Right shift unsigned: Shift A to the right by B bits.

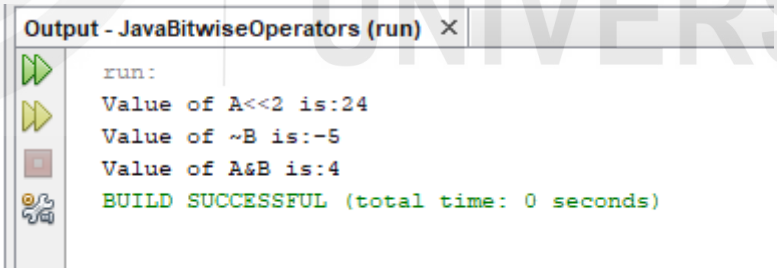
		Low-order bits are lost. Zeros fill in left bits regardless of sign example.
~	~B	Bitwise complement: The bitwise Complement changes the bits. 1, into 0, and bit 0, to 1.
&	A & B	Bitwise AND: AND is 1 if both the bits are 1; otherwise AND is 0. The bitwise AND is true only if both bits are set.
|	A | B	Bitwise OR: The bitwise OR is true if either bits are set. Or if the bit is 1 if either of the two bits is 1; otherwise, it is 0.
^	A ^ B	Bitwise Exclusive OR: The bitwise Exclusive OR is true if either bits are set, but not both. XOR of the bits is 1 if one bit is 1 & other bit is 0; otherwise, it is 0.

Example Proram: Use of Bitwise Oprators

```
package javabitwiseoperators;
public class JavaBitwiseOperators
{
 public static void main(String[] args)
 {

     int A, B, C,D,E;
     A = 6;
     B = 4;
     C = A<<2;
     D = ~B;
     E = A&B;
     System.out.println("Value of A<<2 is:"+ C);
     System.out.println("Value of ~B is:"+ D);
     System.out.println("Value of A&B is:"+E);
 }
}
```

Output:



Assignment Operators

The basic assignment operator '=' is used to assign value to the variables. For example, I = J; in which we assign the value of J to I. Similarly, let us see how to assign values to different data types.

```
int  IntVal = 50;
float FloatVal = 15.5;
char CharacrcterVal = 'P';
Boolean BooleanVal = true;
```

Along with basic assignment operators, Java has defined shortcut assignment operators that allow you to perform arithmetic, shift, or bitwise operation with one Operator. For example, if you want to add a number to a variable and assign the result back into the same variable, then

you will write something like i = i + 2;

but using shortcut operator '+=' , you can shorten this statement, like
i += 2
Here remember, i = i + 2; and i += 2; both statements are the same for the compiler.

As given in *Table 2.12*, Java programming language provides different *shortcut assignment operators*:

Combined Assignment Operators

In an assignment operator, the same variable may appear on either side. To simplify the assignment, Java has provided a set of arithmetic assignment operators which are listed in Table 2.12.

Table 2.12: Assignment arithmetic operators

Sl. No.	Operator	Meaning
1.	+=	Addition Assignment
2.	-=	Subtraction Assignment
3.	*=	Multiplication Assignment
4.	/=	Division Assignment
5.	%=	Modulus Assignment

One variable named *var* is initialized as
int *var* = 50;
to realize the assignment arithmetic operation. Here the variable *var* has the value of 50
.
When the following statements make use of the assignment arithmetic operators
var = *var* + 1 (or) *var* + = 1;

The above notations are equal, and the usage of these notations is left to the discretion of the programmer. The value of *var* is 50; it is added with 1, and the results of the above statements assign 51.

The next operator is -= which is used as follows: *var* - = 5;

The value of the variable *var* is 51 is subtracted by 5, and the result 46 is assigned to *var*.

The next Operator *= is used in the following statement
var * = 3 ;

The value of the variable *var* is 46 multiplied by 3, and the result 138 is assigned to a.

The next operator /= is used in the following statement
var / = 4 ;

The value of the variable *var* is 138 which is divided by 4; the result is 34.5, and the integer part of the result, that is, 34 is assigned to *var* . Therefore *var* will have a value 34.

The next operator %= is then applied in the following statement

a % = 5;;

The value of the variable *var* is 34, and modulus operation with 5 actually divides the 34 by 5 and gives the remainder 4, which is assigned to *var*.

Multiple Assignments

Multiple assignments are possible in the following format:
IdentifierA= Identifier B= Identifier C=…….= Expression
In this case, all the Identifiers are assigned the value of Expression.

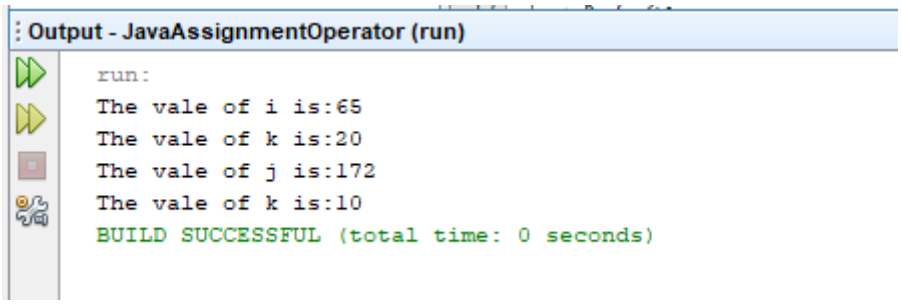
Let's see one example:

P=R=501+101+x+y+z; this is the same if you write:
P=501+101+x+y+z;
Q=501+101+x+y+z;
R=501+101+x+y+z;

Example Program

```
package javaassignmentoperator;
public class JavaAssignmentOperator
{
 public static void main(String[] args)
 {
 int i,j,k,l;
 float p,q,r;
 i= 45;
 k=20;
 i +=20;
 System.out.println("The vale of i is:"+i);
 j = ++i +k;
 System.out.println("The vale of k is:"+k);
 j *=2;
 System.out.println("The vale of j is:"+j);
 k /=2;
 System.out.println("The vale of k is:"+k);
 }
 }
```

Output:



Class and Object Operators

The use of class and object operators is given in the following Table 2.13. Here these operators are listed for the sake of completeness of the topic. Use of some of these operators you will see in unit 4 of this Block.

Table 2.13: Class and Object Operators

Operator	Name	Description
instance of	Class Test Operator	The first operand must be an object reference. For example, 'A instance of B', Returns true if A is an instance of B. Otherwise, it return false.
new	Class Instantiation	Creates a new object. For example, new A, in this A is either a call to a constructor or an array specification.
"."	Class Member Access	It accesses a method or field of a class or object. For example, A.B is used for 'field access for object A', and A.B() is used for 'method access for object A'
( )	Method Invocation	For example, A(parameters) , Declares or calls the method named A with the specified parameters.
(type)	Object Cast	(type) A, in this example () operator Cost (convert) A to a specific type. An exception will be thrown if the type of A is incompatible with the specified type. In this type can be an object or any primitive data type.

☞ **Check Your Progress – 2**

1) What is the use of relational operators? Explain with the help of an example.

……

……

……

2) What is a combined assignment operator? Explain with the help of an example.

……

……

……

3) Explain how to use & and | operators with the help of a program.

……

……

……

2.6 EXPRESSIONS AND STATEMENTS

Here we will learn about the statements and expressions. Expressions are segments of code that perform some computations and return some values. Expressions are formed by combining literals, variables and operators. Certain expressions can be made into statements, which are complete units of execution. In a program, statements are normally executed in the sequence in which they appear in the program. But there are number of situations where you may have to change the order of execution of statements based on certain conditions, or some tie you need to

repeat a group of statements until the certain specified condition(s) are met. You may find more details on statements in unit 3 of this Block.

2.6.1 Expressions

An expression is a series of variables, operators and method calls that evaluates to a single value. Also, sometimes you can write compound expressions by combining expressions as long as the types required by all of the operators involved in the compound expression are compatible/ correct. The Java programming language allows programmers to construct compound expressions and statements from smaller expressions. In Java, it is allowed as long as the data types required by one part of the expression match the data types of the other. Let us consider the following example of a compound expression:

P = A × B / C;

In this example, the order in which the expression is evaluated is unimportant because the results of multiplication and division are independent of order. However, it is not true for all expressions. For example, where addition/subtraction and division/multiplication operations are involved, you will get different results depending on the order of operation (i.e. whether you perform the addition or the division operation first). Such expressions are called ambiguous expression.

For example

R= A + B / 50; //ambiguous expression

As a programmer you can specify exactly how you want an expression to be evaluated by using balanced parentheses '(' and ')'.

For example, you could write:

R= (A + B)/ 50; //unambiguous and recommended way

Suppose as a programmer; you don't explicitly indicate the order of the operations in a compound expression to be performed. In that case, it is determined by the precedence assigned to the operators and the operators which have higher precedence get evaluated first.

Let us consider the above example again; the division operator has higher precedence than the addition operator. Thus, the two following statements are equivalent:

R= A + B / 50;

R= A + (B / 50);

Therefore, it is better to write compound expressions, using parentheses to specify which operators should be evaluated first. This also will make your code easier to read and maintain. The following table 2.14 shows Operator's precedence

Table 2.14: Operators Precedence

Special operators	[ ] Array element reference Member selection (params) Function call
unary operators	++, --, +, -, ~, !
Creation or cast	new (type)expression
multiplicative	*, /, %
Additive	+, -
Shift	<<, >>, >>>

Relational	<, >, <=, >=, instance of
Equality	= =, !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
Conditional	? :
Assignment	=, += ,-=, *= ,/=, %=, &=, ^=, |=, <<= >>=, >>>=

2.6.2 Statements

In a program a statement is a complete unit of execution. Statements in Java are equivalent to sentences in natural languages. It is an executable combination of tokens ending with a semicolon (;) mark. A token of expression can be any variable, constant, Operator or expression. By terminating the expression with a semicolon (;), the following types of expressions can be made into a statement:

- Assignment expressions
- Any use of ++ or --
- Object creation expression
- Method calls

Let us consider one example of expression statements:

```
float piValue = 3.141; //assignment statement
counter++; //increment statement
System.out.println(piValue); //method call statement
Integer integerObject = new Integer(4); //object creation statement
```

In addition to these kinds of expression statements, there are two other kinds of statements.

- A declaration statement declares a variable such as:

  ```
  int counter;
  double xValue = 55.24;
  ```

- A control statement regulates the flow of execution of statements based on the changes to the state of a program.

The while loop, for loop and *if statement* are examples of control flow statements. In this category of statements, you will learn Unit 3 of this Block.

Assignment Statement

The assignment statement assigns the values to variables. The equal (=) sign is used as the assignment operator. The assignment statement has the following format:

Variable = constant (or) variable (or) expression;

The assignment operator has a right-hand side (RHS) and a left-hand side (LHS). The LHS must be a variable, and the RHS may have a constant or variable or expression.

If the assignment statement has a constant on the LHS, the value is assigned to the RHS variable. If it has a variable on the RHS, the constant is assigned to the LHS

variable. Suppose it has a variable on the RHS; the value of the variable is assigned to the LHS variable. If the RHS has an expression it is evaluated, and the result is assigned to the LHS variable. The following statement assigns values to the variables

- StudentName = “Mohit”;
- AccountBalance = previousBalance;
- Area = pi * radius * radius;

The first statement is to assign a string constant to the variable StudentName, the second assigns the value of previousBalance to the variableAccountBalance, and the last statement calculates the result by multiplying the values of pi, and radius twice and assigns it to the variable Area.

Also, while writing a program sometimes the same variable may appear on both the RHS and LHS as follows :

count =20;
count = count + 1 ;

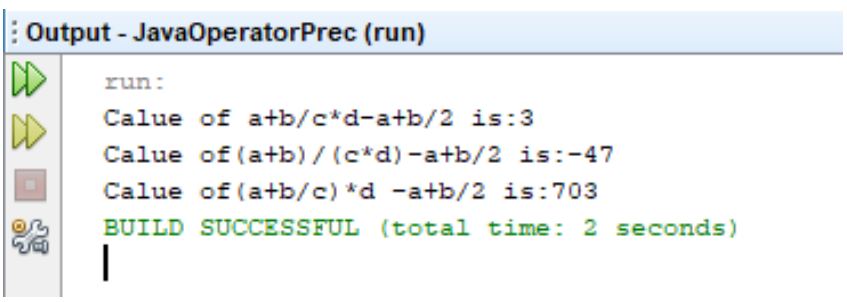
It is an assignment statement where the RHS is evaluated first by taking the present value of the variable count; for example, it is 20, and it is added with the constant 1, and the result 21 is assigned to the variable count on the LHS. Now the value of the count is 21.

Example Program:

```
package javaoperatorprec;
public class JavaOperatorPrec
{
   public static void main(String[] args)
   {
      int a , b, c, d,,e;
      a= 50;
      b=6;
      c=12;
      d=15;
      e = a+b/c*d-a+b/2;
      System.out.println("Calue of a+b/c*d-a+b/2 is:"+e);
      e = (a+b)/(c*d)-a+b/2;
      System.out.println("Calue of(a+b)/(c*d)-a+b/2 is:"+e);
      e = (a+b/c)*d -a+b/2;
      System.out.println("Calue of(a+b/c)*d -a+b/2 is:"+e);
   }

}
```

Output:



2.7 EXAMPLE PROGRAM USING SCANNER CLASS

Java Scanner class provides many methods which a programmer may use to read inputs of different types. This class is part of import java.util package.

Table 2.14: Methods of Scanner Class

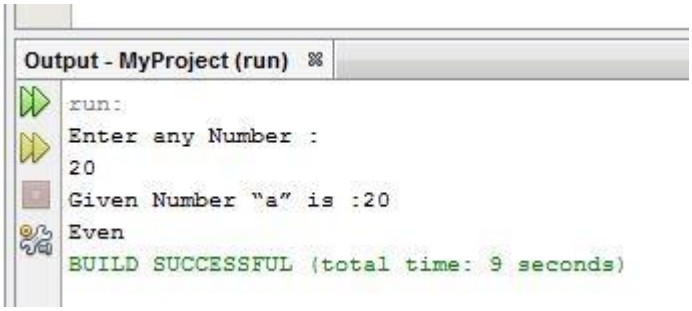
Method	Description
nextInt()	Use to read an int value from the user
nextFloat()	It reads a float value given by the user
nextBoolean()	Used for reading a boolean value from the user
nextLine()	Used for reading a line of text (String) from the user
next()	It reads a word from the user
nextByte()	Used for reading a byte value from the user
nextDouble()	Used for reading a double value from the user
nextShort()	Used for reading a short value from the user
nextLong()	Used for reading a long value from the user

The following Java Program print whether the given number is odd or even. Using method nextInt() input number is taken from the user.

```
//Program to print the odd or even numbers using objects
import java.util.Scanner;
class EvenOdd
{
int a;
public void get ( )
{
Scanner s = new Scanner (System.in);
System.out.println ("Enter any Number : ");
a = s.nextInt ( ) ;
System.out.println ("Given Number “a” is :" + a);
}
public void display ( )
{
if (a%2==0)
{
System.out.println ("Even");
}
else
{
System.out.println ("Odd");
}
}
public static void main (String[ ] args)
{
EvenOdd e=new EvenOdd();
e.get ( ) ;
e.display ( );
```

```
}
}
```

When you run the above java program, it will give the output as follows:



For a better understanding of the use of scanner class, some more examples are also given in the next unit of this Block.

☞ Check Your Progress – 3

1) What will be the output of the following Java program if 7 is given as input?

```
package circlearea;
import java.util.Scanner;
public class CircleArea
{
public static void main(String[] args)
 {
int radius;
float area, pi;
pi = 3.14F;
Scanner s = new Scanner (System.in);
System.out.println ("Enter the Radius of Circle: ");
radius = s.nextInt ( ) ;
area = pi * radius * radius;
System.out.println ("Area of Circle:" + area);
}
}
```

……

……

……

2) Write a Java program that takes the user's name, address and mobile number as input and displays it.

……

……

……

2.8 SUMMARY

This unit has described to you the different basic data types in Java. Basic data types include four groups Integer, Floating Point Numbers, Characters and Boolean. This unit explained Unicode's use and different Java keywords that have special meanings for the compiler and cannot be used as a user-defined identifier. Before using the Java variables in your program, you should declare them and assign them value. You learnt how to declare variables and use them. You have learnt different types of

operators in Java. You have also learnt about the use of expressions and statements. Expression is a series of variables, operators, and method calls that evaluates to a single value. Statements in Java are equivalent to sentences in natural languages. A statement is a complete unit of execution. Toward the end of this unit, you learned to use some Scanner class methods.

2.9 SOLUTIONS/ANSWERS TO CHECK YOUR PROGRESS

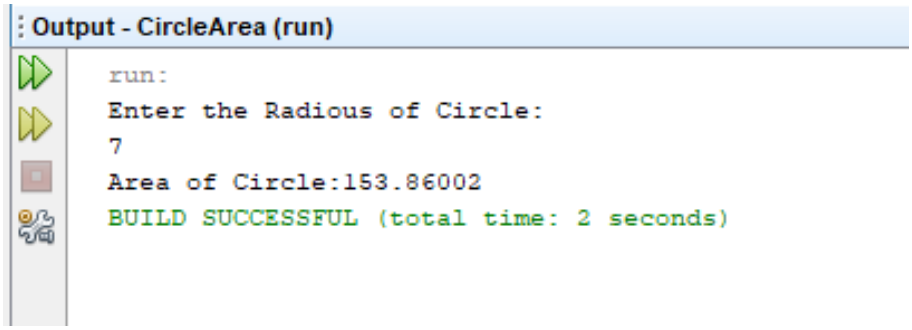
☞ Check Your Progress – 1

1) A literal represents a value of a certain type, and the type describes how that value behaves. Primitive types in Java are called **literals**. A literal is the source code representation of a fixed value in memory. There are different types of literals, namely number literals, character literals, boolean literals, and string literals.

2) Program:

```
package circlearea;
import java.util.Scanner;
public class CircleArea
{
public static void main(String[] args)
 {
final float  pi = 3.14F;
int r;
float area ;
Scanner s = new Scanner (System.in);
System.out.println ("Enter the Radius of Circle: ");
r = s.nextInt ( ) ;
Area = pi * r * r;
System.out.println ("Area of Circle:" + area);
}
}
```

Output:



2) In Java, boolean data type is used to define variables for use as logical variables. A boolean variable can have either true or false values. The boolean variable has only two values: true or false. These two values are boolean constants.

☞ Check Your Progress – 2

1) Relational operators are used to find the relations between two variables(operands). For example, suppose you need to find the relation between the Marks of two students in MCS_206 you can use the relational operators.

   To check whether the marks are equal you can use == Operator. Similarly, you may use other operators such as <, > etc.

2) A combination of assignment operators is a combination of any arithmetic operator(+, -,* etc.) with the assignment operator(=).

For example, if you have an integer variable a and suppose it has a value of 20, then

a *=2;

It will be similar to a = a*2;

3) The following program show how to use & and | operators:

```
package javabitwiseoperators;
public class JavaBitwiseOperators
{
 public static void main(String[] args)
 {
int A, B, C,D,E;
A =16;
B = 4;
C = A&B;
D = A|B;
E = A&D;
System.out.println("Value of A&B  is:"+ C);
System.out.println("Value of A|B is:"+ D);
System.out.println("Value of A&D is:"+E);
 }
}
```

Output:

```
Output - JavaBitwiseOperators (run)
run:
Value of A&B  is:0
Value of A|B is:20
Value of A&D is:16
BUILD SUCCESSFUL (total time: 0 seconds)
```

☞ Check Your Progress – 3

1)

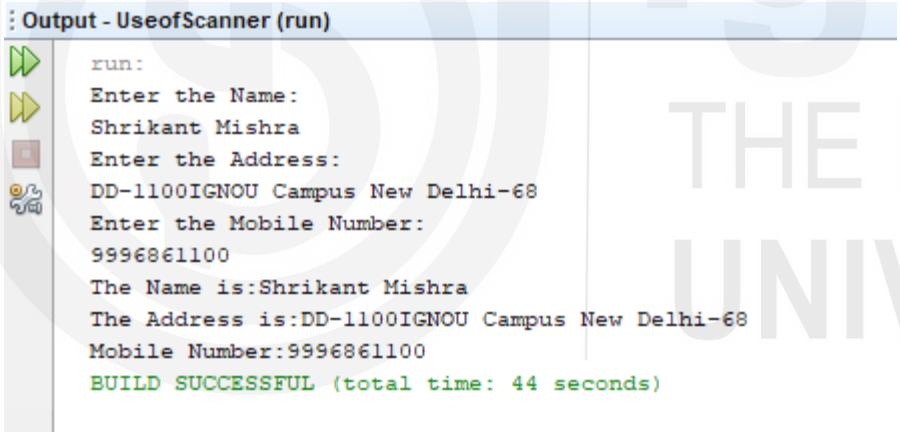
```
Output - CircleArea (run)
run:
Enter the Radious of Circle:
7
Area of Circle:153.86002
BUILD SUCCESSFUL (total time: 3 seconds)
```

2)

Java Program

```
package useofscanner;
import java.util.Scanner;
public class UseofScanner
{
public static void main(String[] args)
 {
String Name,Address;
long Mobile_No;
Scanner s = new Scanner (System.in);
System.out.println ("Enter the Name: ");
Name = s.nextLine ( ) ;
System.out.println ("Enter the Address: ");
Address = s.nextLine ( ) ;
System.out.println ("Enter the Mobile Number: ");
Mobile_No = s.nextLong ( ) ;
System.out.println ("The Name is:" + Name +'\n' + "The Address is:"+ Address +'\n'+ "Mobile Number:"+Mobile_No);
}
}
```

Output:



2.10 REFERENCES/FURTHER READINGS

- S.Sagayaraj,R. Denis, P.Karthik and D.Gajalakshmi, "Java Programming for Core and Advanced Learners", University Press, 2018.
- Herbert Schildt, "Java The Complete Reference", McGraw-Hill,2017.
- Savitch, Walter, " Java: An introduction to problem solving & programming", Pearson Education Limited, 2019.
- CAY S.Horstmann, " Core Java Fundamentals", Pearson, Tenth Ed. , 2017.
- MCS-024 Course of IGNOU BCA(Revised) Programme, Block - Unit-
- https://www.w3schools.com/java/
- http://www.cs.unc.edu/~weiss/COMP14/18-JavaKeyWords.html
- https://www.w3schools.com/java/java_ref_keywords.asp
- https://www.programiz.com/java-programming/scanner

UNIT 3 DECISIONS AND LOOPS CONSTRUCTS

Structure **Page No.**

3.0 Introduction 54
3.1 Objective 54
3.2 Decision Constructs 55
3.2.1 The if statement 55
3.2.2 The if then else statement 56
3.2.3 The switch-case statement 57
3.2.4 The jump statement 58
3.3 Loop Constructs 60
3.3.1 while Loop 61
3.3.2 do-while Loop 61
3.3.3 for Loop 62
3.4 Array 64
3.5 Summary 68
3.6 Solutions/Answers to Check Your Progress 69
3.7 References/Further Readings 71

3.0 INTRODUCTION

The previous unit of this block introduced the basics of Java programming, such as data type, Variables, Operators and Statements/expressions. These basic building blocks of Java programming are used in writing any Java program. Now you are well aware of the basics. In this unit, we will discuss some more features of Java like decision statements, looping statements and arrays. Normally, you know that the statements are executed sequentially in the order they are written in the program, but sometimes situations arise in the program to execute a certain set(s) of code(s) on the basis of arising conditions in the program or repeat some section of code(s) based on the specified condition. For this purpose, Java supports selection statements, iteration statements, jump statements etc. The decision-making statements allow you to execute a particular section of program code based on the result of some boolean conditions in the program. Java supports two selection statements known as ‘if’ and ‘switch’. Besides these, you will also learn some iterative statements such as do, do-while and for loop, which allows you to repeat a block of code under certain conditions. At the end of the unit, you will learn about arrays in Java language. You have already studied these concepts in the MCS-201 course of your first semester. That learning will help you pursue this unit. This unit consists of Java program examples which are executed in NetBeans IDE.

3.1 OBJECTIVES

After going through this unit, you should be able to:

- used various kinds of decision statements in Java program,
- write Java programs using various iterative statements,
- Explain how to use array in programming, and
- differentiate between static array and dynamic array.

3.2 DECISION CONSTRUCTS

The program is not always executed in a sequential manner; sometimes, the flow of the program requires executing part of a program based on the condition or transferring control from one part to another part of the program. The Java language provides decision making statements to take a decision based upon the result of a specified condition and execute a set of codes based on that. The selection statement is the *'if'* statement and the *'switch'* statement. The condition defined with the selection statement is checked at the run time. Actually, the flow of programs and the output of a particular part of the program are decided by the conditions. Java supports two jump statements as *'break'* and *'continue'*.You will learn these selections and jump statements in the following section.

3.2.1 The if Statement

The functioning of 'if' statement in Java language is similar to the 'if' statement in any other programming language. It is syntactically identical to the 'if' statement in C and C++. The simplest form of the 'if' statement is shown here:

if(*condition*) statement;

where *condition* is a boolean expression, and it must be enclosed with parentheses. If the condition returns true value, then the statement is executed; otherwise, the statement is bypassed. For example:

if(a > b) System.out.println("a is greater than b");

The message *a is greater than b* will be printed only when if(a>b) returns true.

You can execute multiple statements when a single condition is true. For this, you can use a set of statements or block of statements which take the following form:

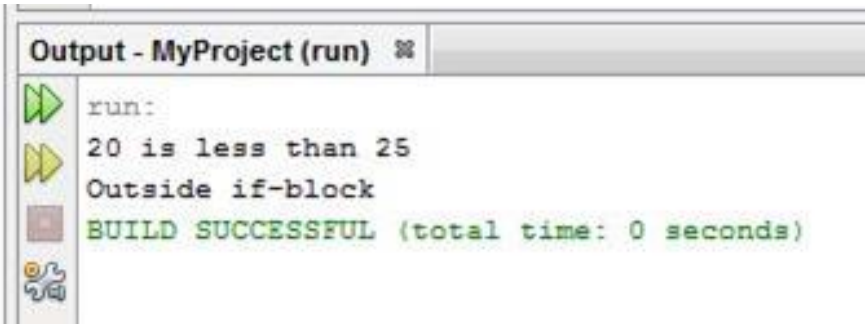
```
if(condition)
  {
       Statement1
       Statement2
       . . .
}
```

Example-1

Consider the following example for if statement. In this example, x is initialized with 20. Using 'if' statement, the condition is checked and if it is true then only "20 is less than 25" gets printed.

```
class IfExample
 {
public static void main(String args[])
{
int a = 20;
if (a < 25)
System.out.println("20 is less than 25");
System.out.println("Outside if-block");
}
}
```

Output:



3.2.2 The if-then-else Statement

The if-then-else statement is represented in Java as *if-else* statement and is used to execute a certain part of codes. The 'if' statement is followed by a test condition, and when the specified condition evaluates to true, then the block of code of *'if'* part is executed. If the specified condition evaluates to false, then control is transferred to *else* part of the program.

The following is the syntax of if...else statement:

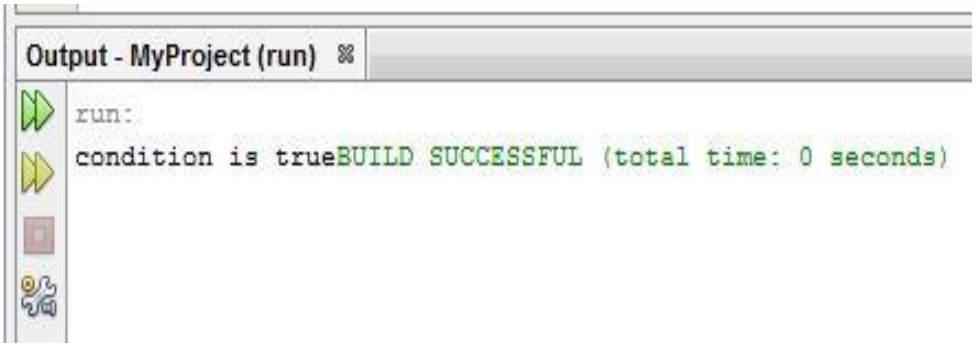
```
if(condition)
{
  // executes code when the condition is true
}
else
 {
  // executes code when the conditionis false
}
```

Example-2

Consider the following example for if-else statement:

```
class Example1
{
public static void main(String args[])
{
int a = 50;
if( a > 20 )
{
System.out.print("condition is true");
}
else
{
System.out.print("condition is false");
}
}
}
```

Output:



3.2.3 The switch-case Statement

The if-else concept can be bulky when you have to work with multiple selections with many alternatives. For this purpose, Java provides *switch-case* statement. The basic purpose of the switch statement is used to select one statement (block) from the multiple code blocks for execution. If there are N number of case values for a switch condition, then each case will have a different value. Each case statement can have a break statement which is optional. When control comes at the break statement, it terminates the statement sequence inside the switch and control transfers after the switch statement. If a break statement is not there, it executes the next case of the switch statement.

The syntax of the switch statement in Java is as follows:

```
switch (condition)
{
case value1:
   // Java code
         break;
case value2:
   // Javacode
         break;
         ...
         ...
default:
   // default statements
 }
```

When you run your Java program with a switch statement, the condition is evaluated once and compared with the values of each case. If the condition matches with value1, the Java code of the case value1 is executed. In a similar manner, the code of case value2 is executed if the condition matches the value2. If there is no match, then the code of the default case is executed.

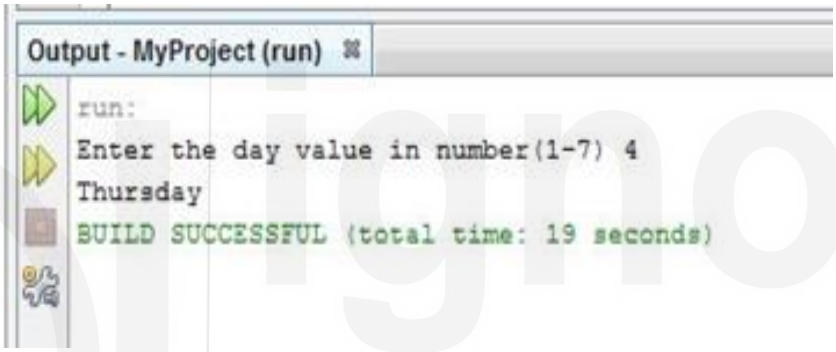
Example-3

The following Java example will ask the user for the input value in numbers and will display the output according to the value entered by the user.

```
// Java program using switch statement
import Java.util.Scanner;
class switchExample
{
public static void main(String[] args)
{
int day;
```

```
// Create a Scanner object for keyboard input.
     Scanner input = new Scanner(System.in);
     System.out.print("Enter the day value in number(1-7) ");
     day = input.nextInt();
switch (day)
{
case 1:System.out.println("Monday");break;
case 2:System.out.println("Tuesday");break;
case 3:System.out.println("Wednesday.");break;
case 4:System.out.println("Thursday");break;
case 5:System.out.println("Friday");break;
case 6:System.out.println("Saturday");break;
default: System.out.println("Sunday");
}
}
}
```

Output:



3.2.4 The jump statement

While working with loops, it is sometimes desirable to skip some statements within the loop or stop the loop instantly without evaluating the test condition. For this purpose, Java provides jump statement. The jump statement is used to transfer the flow control from one part to another part of the program code. Mostly, two Jump statements i.e. break and continue statements in Java language, are used. As the name implies the meaning of both statements, the *'break'* keyword is used to designate break statements and *'continue*' keyword is used to designate the continue statement in Java programming. The third jump statement i.e. return, is used to terminate the current method. In the below section, both break and continue statements are discussed in detail with examples.

Break Statement

In Java, the break statement is broadly used with *'for'* loop, *'while'* loop, *'do-while'* loop and *'switch'* statement. The use of *a break statement in a switch statement terminates* the statement sequence. It can also be used to exit from loop execution and control transfer to the next statement immediately following the loop.

Example-4

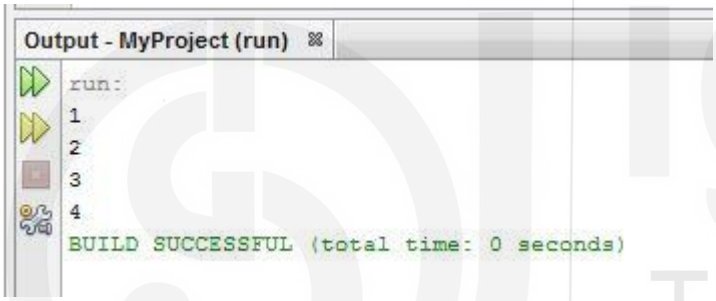
The following is a simple Java program to demonstrate the use of break statement:

```
// Java program using break statement
classBreakExample
```

```
{
public static void main(String[] args)
{
  // for loop
for (int x = 1; x <= 100; ++x)
{
// if the value of x is 5 the loop terminates
if (x == 5)
 {
break;
 }
System.out.println(x);
 }
 }
}
```

When you run the above program, it will print the value of x in each iteration using *'for'* loop (You will read in detail about the 'for' loop in section 3.3.3 of this unit). When the value of x is equivalent to 5, the loop terminates. Hence you will get the output with values less than 5 only.

Output:



Continue Statement

In Java, the continue statement is used with *'for'* loop, *'while'* loop and *'do-while'* loop. It is used to skip the current iteration of a loop and control transfer to the next statement or end of the loop. It all depends on the condition that occurs in the program. Also , the continue statement is also used in decision-making statements like *if..else* statement.

The following is the syntax of the continue statement.

continue;

Example-5

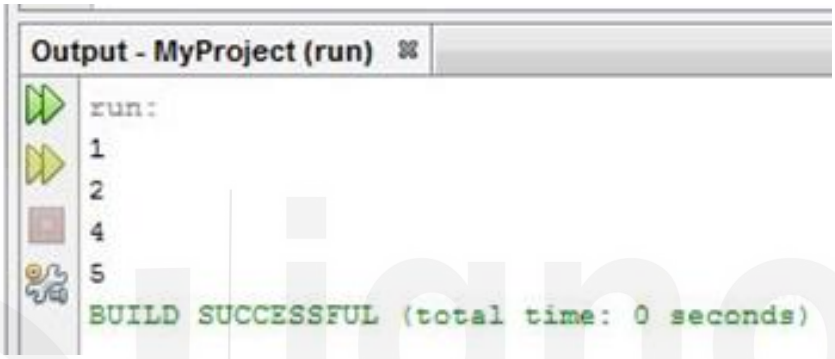
The following is a simple Java program to demonstrate the use of the *'continue'* statement:

```
//Java program for continue statement
classContinueExample
{
public static void main(String[] args)
 {
   //for loop
for(int x=1;x<=5;x++)
{
if(x==3)
```

```
{
 //using continue statement
continue;//  here, it will skip the rest statement
}
System.out.println(x);
 }
}
}
```

When you run the above program, it will print the series like 1,2,4,5 and 3 will not be displayed on the console. It is because the loop is continued when it reaches to 3. After running the program, you will get the result like:

Output:



☞ Check Your Progress- 1

1) Explain the decision-making statement in Java with examples.

……

……

……

2) What is a branching statement in Java? What are the types of branching statements in Java? Explain any one of them with an example.

……

……

……

3) What is the break statement? Explain its use in Java with an example.

……

……

……

3.3 LOOP CONSTRUCTS

In the previous section, you have learned about the decision-making statement. This section describes looping statements or iterating statements, which allow you to execute the same set of instructions repeatedly until a certain condition is fulfilled. Java supports three types of loops such as *while loop, do-while loop* and *for loop*. All three loops of Java are discussed below with examples.

i. while Loop

The while loop is a most fundamental looping statement in Java which repeats a statement or block of code until its condition is true. Below is its general syntax:

```
while(condition)
{
  // body of loop
}
```

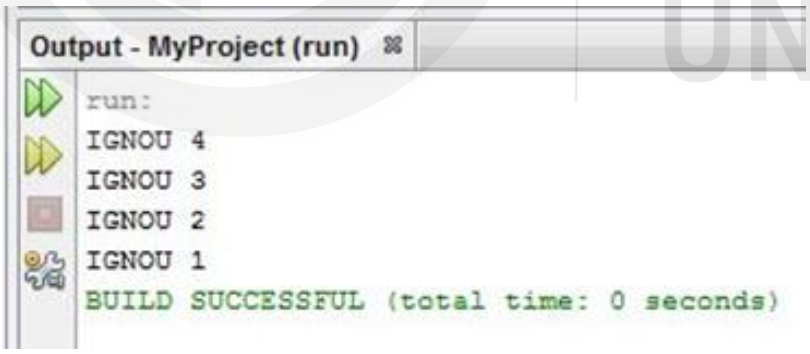
The condition can be any boolean expression, and the body of the loop will be repeated until the condition is true. When the condition turns false, the control transfers to the next line of program code immediately after the while loop.

Example-6

Here is an example to show the use of the while loop. This program prints " IGNOU " words with numbers in decreasing order four times.

```
// Demonstration of while loop
classwhileLoop
{
public static void main (String args[])
{
int i = 4;
while(i > 0)
 {
System.out.println("IGNOU"+" "+ i);
i--;
}
}
}
```

Output:



ii. do-while Loop

The do-while loop in Java is used to execute a part of the program code repeatedly until the specified condition is true. The statements within do-while loop are executed once, and thereafter the condition gets evaluated; if this condition is true, then the loop will repeat. Otherwise, the loop terminates, and control is transferred to the next statement after do-while loop.

A do-while loop in Java is similar to the while loop with one exception. The while loop checks the condition first and then executes the statement(s), whereas do-while loop will execute the statement(s) at least once, and then the condition is checked.

The syntax of do-while loop is as follows:

do

{

// body of loop

}while (condition);

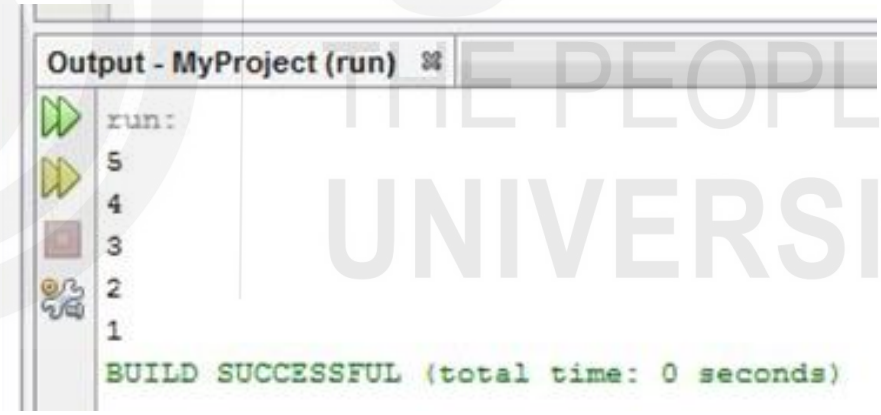
Example-7

Code Snippet to demonstrate the use of do-while loop:

```
// Demonstration of do-while loop
classdoWhileExample
{
public static void main(String args[])
{
int x=5;
do
{
System.out.println(x);
x--;
}while(x>0);
}
}
```

When you run the above program, it will display the following output:

Output:



iii. for Loop

The 'for' loop allow(s) you to execute a block of statement repetitively, a fixed number of times on the basis of the decided pre-condition. The for loops is one of the very commonly used looping constructs in programming. The 'for' loop is a very general construct of looping that supports the iteration of a block of code controlled by a counter or some updated variable after every iteration.

The basic syntax of for loop is as follows:

for(initialization; condition; increment/decrement)

{

//body of loop

}

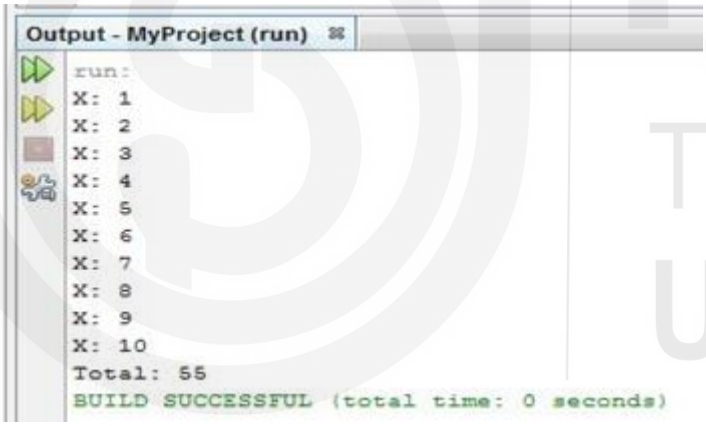
When the 'for' loop first starts, the initialization part is executed only once. Next, the condition is evaluated. This condition must be a boolean expression. It checks the loop control variable against a target value. If this test condition is true, then the body of the loop is executed. If it is false, the loop terminates. Next, the increment/decrement or iteration part of the loop is executed.

Example-8

Consider the following Java program, which prints the series of the first 10 natural numbers and also calculates the total sum of the numbers using for loop.

```
class Total
{
public static void main(String[] args)
{
int total = 0;
for(int x=1; x<=10; x++)
{
System.out.println("X: " + x);
total += x;
}
System.out.println("Total: " + total);
 }
}
```

Output:



ignou
THE PEOPLE'S
UNIVERSITY

☞ Check Your Progress - 2

1) Write a Java program using while loop to display numbers from 1 to 'N', where the program will ask the user to input the number 'N'.

   ……

   ……

   ……

2) What is the difference between while and do-while loop?

   ……

   ……

   ……

3) Write a Java program which prints odd numbers between 0 and 10 using 'for' loop.

……

……

……

3.4 ARRAY

There is always a need to store multiple data items of the same data types in programming. For example, storing marks of one subject of all the students in a class. For this purpose, Java uses an array. An array is a collection of similar data type items which are stored in contiguous memory locations. These multiple values are stored in a single variable rather than declaring separate variables for each value.

One-Dimensional Arrays

A one-dimensional array is a list of variables of similar/same data type. For declaring an array, you must define the variable type with square brackets. The general form of one-dimensional array declaration is as follows:

type var_name[];

Here, type indicates the *data type* of array. For example, a variable named 'Bus' is declared as an array of strings.

String[] Bus;

You can use an array literal to insert values to it and place the values in a comma-separated list within curly braces, as shown below.

String[] Bus= {"Volvo", "TATA", " Ashok Leyland", " Mahindra & Mahindra"};

Accessing the value of an array element is done using the index number. The index of an array variable determines on which value of the array you want to access or work on. The following example shows how to access the value of the first element in the 'vehicle' array:

Example-

```
String[] vehicle = {"car", "bus", "scooter", "motorcycle"};
System.out.println(vehicle[0]);
// Outputs: car
```

In the above array- vehicle, there are four values stored, and the value stored at index 0 is "car"; similarly, at index 3 value "motorcycle" is stored. Note that indexing start from 0 and goes up to N-1, where N is the total elements in the array.

In a similar way, you can create an array of integers.

int[] Marks = {1, 2, 3, 4};

It is noted that Java uses zero-based indexing i.e. indexing of arrays in Java starts with zero(0). In the array variable Marks, given above if we access Marks[2] we will get the value 3.

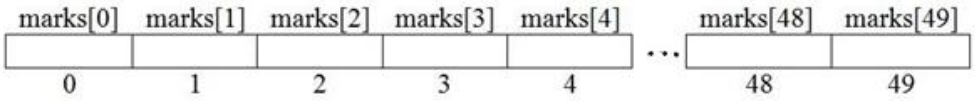
Creating an Array using New Operator

The main objective of the *new operator* is to allocate memory for a variable or an object during the run time. The array length need not be a constant: new int [n] creates an array of length n. Like the following, statement declares and initializes an array of 100 integers. This is general-form of the *new operator* which applies to one-dimensional array and appears as follows:

int [] a = new int [100] ;

For **example**, you may declare and initializes an integer type of 50 elements of marks as an array.

int [ ] marks = new int [50] ;



The memory allocation for the array variable marks is given as following:

In the above figure, array indexes are from 0 to 49, and the array locations are accessed using marks [0], marks [1]…marks [49]. Since a single name refers to all the locations, the values can be stored and retrieved more easily rather than individual variables.

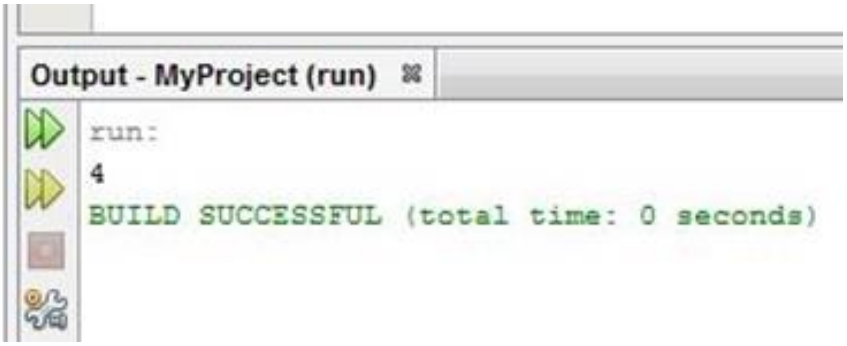
You can use the length property of array in Java to find out the total number of elements contained by all the dimensions of that array.

Example-10

```
public class ArrayLength
{
public static void main(String[] args)
{
int[] a = {1,2,3,4};
System.out.println(a.length);
}
}
```

When you run the above program, it will display the following output:

Output:



Multidimensional Array

A multidimensional array is an array of arrays. Such arrays use more than one index to access the array elements. The general syntax of multidimensional array is as under:

data_type[1st dimension][2nd dimension][]……[Nth dimension]

array_name = new data_type[size1][size2]…….[sizeN];

Where data_type represents the type of data to be stored in the array, such as int, char, etc.,and the *dimension* indicates the dimension of the array created, i.e. 2D, 3D etc., **array_name** is the name of the array variable and **size1, size2, …, sizeN** shows sizes of the dimensions respectively.

Two-dimensional arrays

The two-dimensional array is the simplest form of a multidimensional array. In a two-dimensional array, data is stored in rows and columns. You can access the elements using both the row index and column index. It is noted that Java uses zero-based indexing i.e. indexing of arrays in Java starts with zero(0).The following statement shows that two-dimensional array consists of 12 elements, i.e. 3 rows and 4 columns.

int[][] TwoD = new int[3][4];// it can store a total of (3*4) =12 elements.

The following table shows the memory allocation of 12 elements of two-dimensional array:

	Column 1	**Column 2**	**Column 3**	**Column 4**
Row 1	2D[0][0]	2D[0][1]	2D[0][2]	2D[0][3]
Row 2	2D[1][0]	2D[1][1]	2D[1][2]	2D[1][3]
Row 3	2D[2][0]	2D[2][1]	2D[2][2]	2D[2][3]

When you will create a two-dimensional array, add each array within its own set of curly braces as shown below:

int[][] NumArray = { {1, 2, 3, 4, 5}, { 6, 7, 8, 9} };

NumArray is two dimensional array. For accessing the elements of the NumArray, you can specify two indexes: one for the array and another for the element inside the array.

Three-dimensional Arrays

The following three dimensional integer arrays is accomplished to store 6000 elements:

int[][][] ThreeDArray = new int[10][20][30];// it can store a total of (10*20*30) = 6000 elements

Example-11

The following Java program demonstrates the two-dimensional array in which each element displays the number values from left to right and top to bottom:

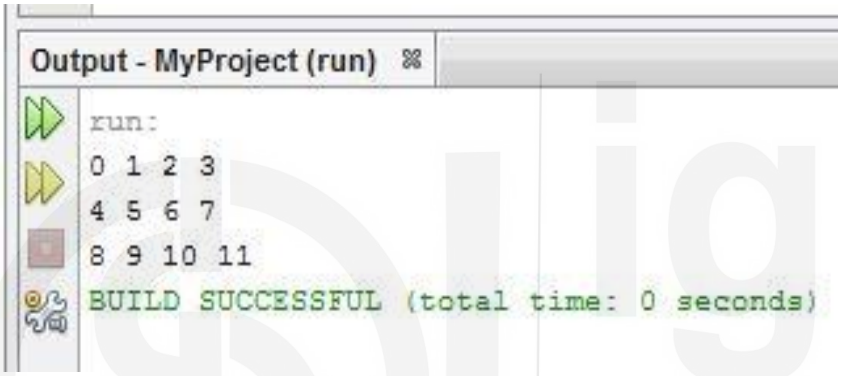
//demonstrates the two-dimensional array

```
class TwoDArray
{
public static void main(String[] args)
 {
// Array consists of 12 elements i.e. 3 row and 4 column
  int[][] TwoDim = new int[3][4];
```

```
 int x, y, z = 0;
  for (x = 0; x <3; x++)
    for (y = 0; y <4; y++)
         {
TwoDim[x][y] = z;
 z++;
  }
   for (x = 0; x <3; x++)
{
    for (y = 0; y <4; y++)
      System.out.print(TwoDim[x][y] + " ");
      System.out.println();
   }
 }
}
```

When you run the above program, it will display the following output:

Output:



Variable-length array (Dynamic Array)

As yet, you have learned fixed-sized or static array. Static arrays are declared before runtime and are assigned values while writing the code. It means that static arrays are allocated memory at compile time. The variable-length array or dynamic array is declared while writing the code, and memory is allocated at the runtime. The memory is allocated from the heap. In other words, the dynamic arrays can be assigned values in run-time. If you want to create dynamic arrays of variable length, you can use collections like *array list*. Java also provides the facility to specify the size of the remaining dimensions separately, except for the first dimension. The following program explains this concept in which the second dimension size to array variable is allocated manually.

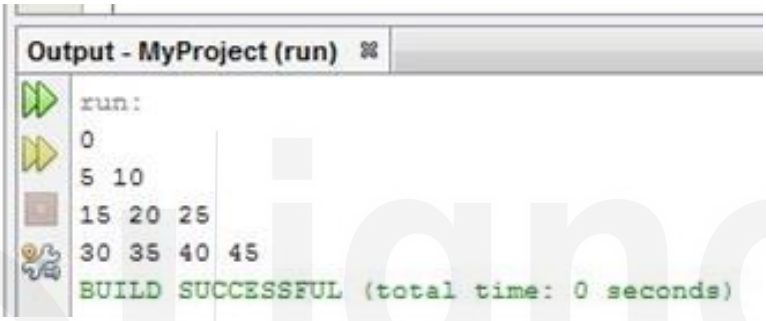
Example-12

```
class ArrayDSize
{
public static void main (String args[])
{
int x, y, z=0;
int twoDim [ ][ ] = new int [4][ ];
twoDim[0] = new int[1];
twoDim[1] = new int[2];
twoDim[2] = new int[3];
twoDim[3] = new int[4];
for ( x= 0; x<4 ; x++)
{
for (y = 0; y<x+1; y++)
```

```
{
twoDim[x][y] = z + z*4;
z++;
}
}
for ( x= 0; x <4 ; x++)
{
for (y = 0; y< x+1; y++)
System.out.print(twoDim[x][y] + " ");
System.out.println();
}
}
}
```

When you run the above program, it will display the following output:

Output:



☞ **Check Your Progress - 3**

1) What is an array in Java? Also, define a multidimensional array in JAVA.

……

……

……

2) Write a Java program to print the sum of all the array elements.

……

……

……

3) What is the difference between a static array and a dynamic array?

……

……

……

3.5 SUMMARY

In this unit you have been explained the Java decision and looping statements. Selection statements allow a Java program to check numerous conditions and execute a block of code based on the condition given. You have learned here if, if-else, switch statements and jump statements. Java developer mainly uses two jump statements, i.e. break and continue statement. The break statement is used to terminate the statement sequence. The continue statement is used to skip the current iteration of loops (i.e. for, while, or do-while loop).You can make multiple-choice

decisions using a switch statement based on a single condition value. The switch test condition evaluates true, control transfers to the matching case and executes the appropriate case statement.

Java supports three loops constructs such as *for, do and do-while*. The 'for' loop allows you to execute a block of the statement(s) repetitively for a fixed number of times based on the condition given. Additionally, you have also learned the *while* loop and *do-while* loop. Also, in this unit, you also learned about the array, such as declaring or creating an array and inserting values into the array. This unit also discussed the multidimensional array in Java program.

3.6 SOLUTIONS/ANSWERSTO CHECK YOUR PROGRESS

☞ **Check Your Progress -1**

1) Usually, the program is written in a sequential manner and executed in the order in which they appear, but the program appearance is not always sequential. Sometimes as per need of program, you have to define conditions. Using Java decision-making statements, you can define conditions on which you can make a decision and get a result based on that condition. It happens only when the jumping of statements or repetition of specific code is necessary. The decision-making statements allow you to execute specific parts of the program on the basis of a specified condition. Java supports some control statements such as if, if-else, and switch-case statements.

2) Branching statements are used to transfer the flow of execution from one section of a source code to another section. This kind of statement is widely used within the control statements. Java has mainly three branching statements, i.e., break, continue, and return, but programmers mostly use two of them (break, continue). The branching statements allow you to exit from a control statement when certain conditions meet.

   The continue statement is used with *for loop, while loop* and *do-while loop*. This statement is used to skip the current iteration of a loop. For the example, you can refer to section 3.2.4 of this unit.

3) The break statement is broadly used with *for loop, while loop, do-while loop* and *switch statement*. The use of a break statement within the switch statement terminates the statement sequence.

   Example Program

```
class Test
{
public static void main(String args[])
{
for (int x = 0; x < 10; x++)
{
if (x == 6)
{
break;  //break statement
  }
System.out.println(x);
}
}
}
```

☞ Check Your Progress - 2

1) Program code which displays the numbers from 1 to N.

```
import java.util.Scanner;
class exampleWhile
{

public static void main(String[] args)
 {
 int num;
Scanner input = new Scanner(System.in); //use of Scanner class to take input
System.out.print("Enter the N number : ");
num = input.nextInt();
int x = 1;
   // while loop from 1 to 10
   while(x <= num)
 {
     System.out.println(x);
  x++;
   }
 }
}
```

2) A do-while loop in Java is similar to the while loop with one exception. In while loop the condition is checked first and then executes the statement(s), whereas do-while loop will execute the statement(s) at least once and then checks the condition.

3) The 'for' loop allows you to execute a block of a statement(s) repetitively a fixed number of times on the basis of the condition. When the 'for' loop starts, the initialization part is executed only once. Next, the *condition* is evaluated. This *condition* must be a boolean expression. It checks the loop control variable against a target value. If this test condition is true, then the body of the loop is executed. If it is false, the loop terminates. Next, the *loop's increment/decrement or iteration part* is executed.

Example Program

```
ClassforLoopExample
{
public static void main(String[] args)
{
for (int x = 0; x<=10; x=x+3)
{
System.out.println(x);
}
}
}
```

☞ **Check Your Progress- 3**

1) An array provide you chance to create variable which is a collection of similar data type items, and are stored in contiguous memory locations. These multiple values are stored in a single variable rather than declaring separate variables for each value. A one-dimensional array is a list of the same type of variables. A multidimensional array is an array of arrays. In multidimensional arrays, more than one indexes are used to access the array elements.

2) The Java program given below prints the sum of all the elements of the given array.

```
class ArrayExample
 {
   public static void main(String[] args)
 {
       //Initialize array
       int [] A = new int [] {10, 20, 30, 40, 50};
       int sum = 0;
       //'For' Loop for calulating the sum of elements
       for (int x = 0; x < A.length; x++)
 {
         sum = sum + A[x];
       }
       System.out.println("Sum of elements of an array: " + sum);
 }
 }
```

3) Static Arrays are declared before runtime and are assigned values while writing the code. It means that static arrays are allocated memory at compile time. The variable-length array or dynamic array is declared while writing the code and can be assigned values in runtime. In other words, the dynamic arrays are allocated memory at the runtime, and the memory is allocated from the heap. If you want to create dynamic arrays of variable length, you can use collections *like array list.*

3.7 FURTHER READINGS

- Herbert Schildt, "Java The Complete Reference", McGraw-Hill,2017.
- Savitch, Walter, " Java: An introduction to problem solving & programming", Pearson Education Limited, 2019.
- S.Sagayaraj,R. Denis, P.Karthik and D.Gajalakshmi, "Java Programming for Core and Advanced Learners", University Press, 2018.
- Neil, O. , "Teach yourself JAVA", Tata McGraw-Hill Education, 1999.
- https://www.w3schools.com/Java/
- https://docs.oracle.com/Javase/tutorial/Java/nutsandbolts/arrays.html

UNIT 4 CLASS AND OBJECTS

Structure **Page No.**

4.0 Introduction 72

4.1 Objectives 73

4.2 Class Fundamentals 73

4.2.1 Creating objects 75

4.2.2 Assigning object reference variables 78

4.3 Introducing Methods 80

4.3.1 General Methods 80

4.3.2 Static methods 81

4.4 Encapsulation 84

4.5 Access Modifiers 85

4.6 Constructors in Java 87

4.6.1 Overloading constructors 90

4.7 This Keyword 93

4.8 Using Objects as Parameters 96

4.8.1 Argument passing 96

4.8.2 Returning objects 99

4.9 Garbage Collection 99

4.9.1 the finalize ( ) Method 100

4.10 Summary 103

4.11 Solutions/Answers 103

4.12 References/Further Readings 107

4.0 INTRODUCTION

In Unit 1 of this Block you have learned basic features of Object Oriented programming. As of now you are aware of the basic concepts of Object Orientation. When we look at object orientation closely, it is observed that the principles used in this approach are based on real-life philosophy. In Object Oriented Programming (OOP), data is given emphasis. To achieve OOP objectives, the concept of classification and encapsulation in designing and defining classes is emphasized. As classes are the core component of object-oriented programming, you need to focus on identifying and defining them in your program to develop any solution. OOP makes you think about objects and interaction between objects while developing solutions.

In this unit, you will study how class and objects are defined in Java. You will learn to create and use objects in problem-solving. You will also study how to create methods and call them in your program. In this unit, you will learn to define the constructor which is used to initialize the objects. Further, you will learn the use of ***this*** keyword in writing programs. Also, this unit explains the memory deallocation and garbage collection in Java and the use of finalize method.

4.1 OBJECTIVES

After going through this unit you will be able to:

- Define class and objects in Java,
- Define and use methods in Java program,
- Use the concept of Encapsulation in Java programming,
- Use Access Modifiers,
- Define Constructors in Java programs,
- Use this keyword ,
- Pass objects as parameters,
- Describe garbage collection in Java , and
- Use finalize ( ) method in Java Program.

4.2 CLASS FUNDAMENTALS

You have been introduced the concept of class in unit 1 of this Block. Now let us learn how classes are defined and used in Java.

The class is a basic construct which makes Java an object-oriented programming language. A class is a group of values with a set of operations to manipulate these values. The operation of the class provides functionality to it. The group of values are used to represent the attribute (characteristic) of the objects belonging to the class. Classes bring modularity to the system and also provide the scope of implementing information hiding. Classes are used for defining a new data type. Once you define a new data type, you can create variables of this data type and use them in your program. Classes are used to create *objects*. The *objects* carry the properties of the class with values and actually participate in problem solving. Thus, it can be said that *"a class is a template for an object, and object is an instance of a class".* You can create as many instances of a class as required in your program for problem-solving.

Before you define a class, it must be clear to you for what purpose you are going to create that class. i.e. "the nature and exact form of the class" should be reflected in your class definition.

Now let us see how a class is defined in Java.

The general form of a Class:

```
class  NameOfClass
{
type var1;
type var2;
…
type varN;
return type Method1( list of arguments….)
{
  // body of method
}
return type Method2 ( list of arguments…….)
```

```
{
//body of method
}
}
```

A class is declared using the ***class*** *keyword*. Inside the class, variables and methods are defined. The data (variables) and methods of the class are called a *member* of the class. Data are called ***member data*** *or* ***instance variables*** of the class, and methods are called ***member functions*** of the class. The member functions are defined inside the body of the class and perform the desired objective for which they are implemented. One method should fulfill only one well defined objective.

Before you define a class, it is necessary that you decide what should be the data members and member functions of that class. For that, you as a programmer need to know the basic requirements that should be fulfilled by the class you are going to create. If we take the example of Student class and want to display the student's basic information, first you should decide the data members required for representing students' basic information, and then you need to identify the member functions that will serve the basic functionality. A member function that may display the basic information is identified and implemented in the example below.

Now you can define a class Student as:

```
class  Student
{
String name;
String  course
int  roll_no;
}
```

The class Student declared above is not yet complete because only data members are declared in this class. Still as per our need a member function to display basic information is to be defined. So, complete class student definition will have three data members : name, course and age, and one member function Display_Info( ).

```
//class definition
class Student
{
String name;
String course ;
int roll_no;
int age;
void Display_Info( ) // function for displaying basic information
{
System.out.println(“ Student Information”);
System.out.println(“Name:”+name);
System.out.println(“Course:”+course);
System.out.println(“Roll Number:”+roll_no);
System.out.println(“Age:”+age);
}
}
// end of Student class
```

As mentioned earlier, a class defines a new data type. In this case, now Student is a new data type, and Student can be used to declare objects of this class.

An object of Student class can be created and memory assigned as follows, using *new* operator:

Student student1 = new Student( );

When the statement given above is executed, an object named *student1* of type Student is created. You will see a detailed discussion of this type of statement in a later section of this unit. Once an object is created, it can be used for performing the defined operations. Each time you create an object of a class, a copy of each instance variable defined in the class is created. In other words, you can say that each object of a class has its own copy of data members defined in the class. Member functions have only one copy and is shared by all the objects of that class. All the objects may have their own value of instance variables. As shown in *Figure 4.1, there are three objects* of the Student class, and every object has its own copy of name, course, and age, but only one copy of method Display_Info( ) for all the objects.

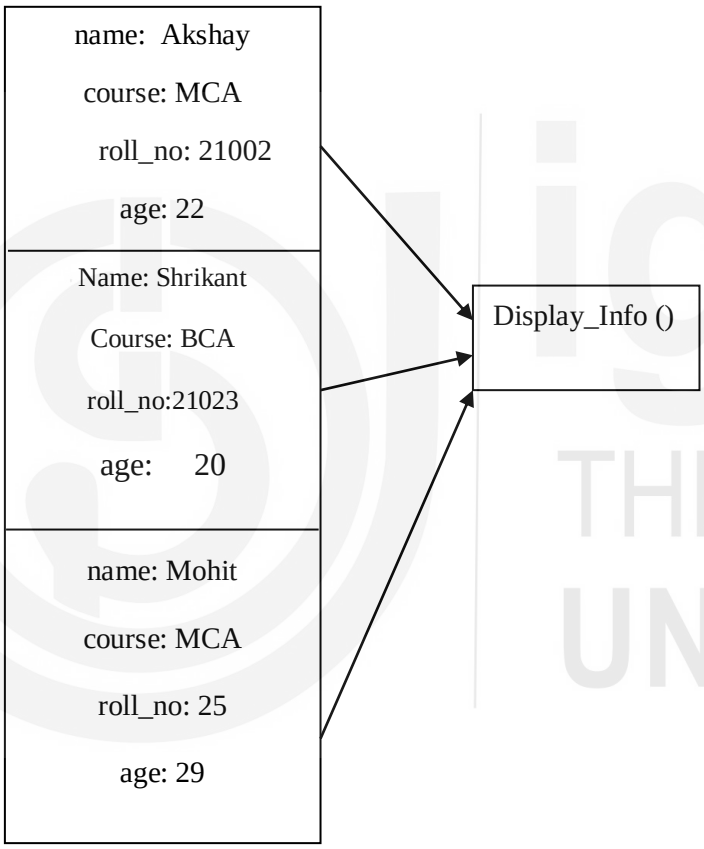


Figure 4.1: Objects of Student class

Now let us see how objects are declared and used inside a program.

4.2.1 Creating Objects

An object of a class can be created by performing these two steps.

1. Declare an object of the class, similar to the declaration of any other variables you have done so far.
2. Acquire space for an object and bind it to the object.

Can you guess why two steps are used?

The first point indicates that first, only a reference to a variable(object) is created.

And at this stage, no actual object exists. The memory needs to be allocated for the actual existence of an object, and that is done using the **new** operator.

The **new** operator in Java is used for dynamically allocating memory for an object. It returns a reference to the object and also binds the reference with the object. Here the reference to object means the address of the object. So, it is essential in Java that all the objects must be allocated memory dynamically.

Declaring objects of Student class

Step 1:

Student student1; // declaring reference to object

Step 2:

Student1 = new Student( ); // allocating memory

In normal practice, these two steps are written in a single statement as

Student student1 = **new** Student( );

There are two ways to assign value to the object's instance variables. The first way is to create an object and initialize all the instance variables of the objects one by one, as shown in the Java program given below. Initialization of instance variable is done by using dot(.) operator. Remember that the dot(.) operator is used to access the object's members (data and method both).

object_name.variable_name = value;

The way mentioned above for initializing objects is not convenient as you have to carefully initialize all instance variables of the object. This exercise should be performed for all objects before they are used in any operation of the program. In this way of initialization, when there is large numbers of objects to be initialized, there is a chance that the programmer may skip some variables unassigned. To overcome this problem the second approach of object initialization may be performed using **constructors.** We will discuss how constructors are used in the coming section of this unit.

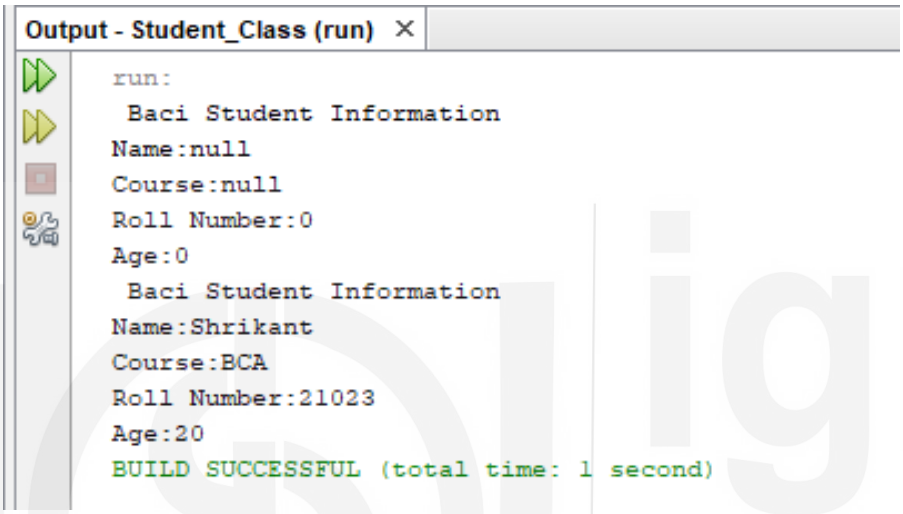
Now you can see a complete Java program for displaying basic information of students.

Java Program

```
package student_class;
class Student
{
String name;
String course ;
int roll_no;
int age;
void Display_Info( ) // method to displaying basic information
{
System.out.println(" Basic Student Information");
System.out.println("Name:"+name);
System.out.println("Course:"+course);
System.out.println("Roll Number:"+roll_no);
System.out.println("Age:"+age);
}
}
// end of Student class
public class Student_Class
{
public static void main( String para[])
```

```
{
Student  student1;//object declaration
student1 = new Student(); // Assigning memory
student1.Display_Info(); // invoking Display_Info( ) method  on an object which is not initialised
student1.name = "Shrikant"; //assigning value to name variable of  student1 object
student1.course = "BCA"; //assigning value to course variable of  student1 object
student1.roll_no = 21023; //assigning value to roll_no variable of  student1 object
student1.age = 20; //assigning value to age variable of  student1 object
student1.Display_Info(); // invoking Display_Info( ) method on student1 object
}
}
```

Output:



If you observe this program, instance variables of the object named student1 are assigned values. You may ask, is there any other way of assigning value to instance variables? For this question, the answer is yes, and you will learn about it later in this unit. An object is usable only if its instance variables contain some value. The values of instance variables represent the state of those objects; for example, the student1 object in the above program has assigned some value to its data members. The object student1 represents a student named Shrikant, a BCA student of age 20 whose Roll Number is 21023. When there is any change in the value of any of the instance variables, then the state of the object changes. Also, note that when Display_Info() method is invoked without initializing the member data, then default values of member data are displayed. It means if you do not initialize any member data, it will contain a default value or garbage value.

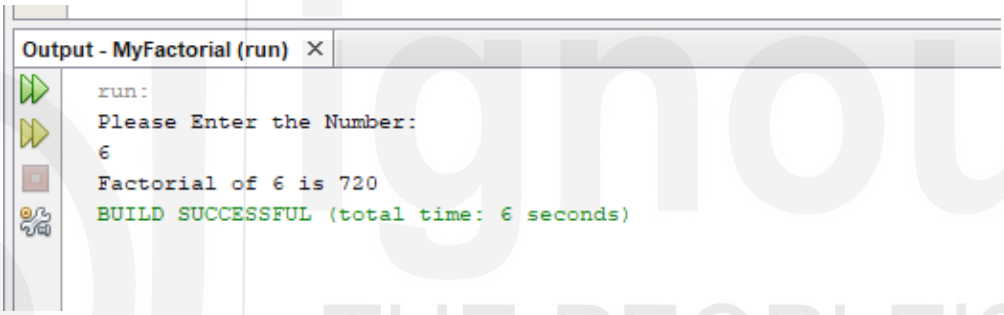
Also, an example program to find the factorial of a given number is written here. This program is just to show you how to create class and objects and use them. You already know the use of Scanner class and its methods in this program. Also, you are aware of using Integer.parseInt(S) method which converts string to integer.

Java Program

```
package myfactorial;
import java.util.Scanner;  //Scanner class
// Java program to find factorial of  a given number
class Factorial
{
  //int number;
// method to find factorial of given number
    int factorial( int n)
```

```
    {
        if (n == 0)
        return 1;
        return n*factorial(n-1);
    }
}
public class MyFactorial
{
public static void main(String[] args)
    {
   Factorial Fact = new Factorial();
   //read the number
    Scanner myObj1 = new Scanner(System.in);  // Scanner object
    System.out.println("Please Enter the Number:");
    String S = myObj1.nextLine();
    int number = Integer.parseInt(S);
    System.out.println("Factorial of "+ number + " is " + Fact.factorial(number));
    }
}
```

Output:



4.2.2 Assigning Object Reference Variables

Now let us see how one object can be assigned to another object of the same type. Also, it is to note that an object assignment is different from a normal assignment. Let us see it with the help of a program in which a class Person with some data members and one member function Display.

Java Program

```
package javatest1;
class Person
{
String name;
int age ;
char sex;
String address;
void Display( )
{
System.out.println ("Name:"+name);
System.out.println ("Sex:"+sex);
System.out.println( "Age:"+age );
System.out.println ("Address:"+address);
}
}
public class JavaTest1
{
```

```
public static void main(String[] args)
{
Person First_P = new Person();
Person Second_P = new Person();
First_P.name = "Mr. Mohi";
First_P.sex = 'm';
First_P.age = 20;
First_P.address = "Saket, New Delhi";
System.out.println(" First Person Information");
First_P.Display();
Second_P = First_P;// Second_P refer to First_P
Second_P.name = "Mr.Shrikant";
Second_P.address = ",IGNOU, New Delhi";
System.out.println(" First Person Information after modification in Second Person
Information");
First_P.Display();//change in Second_P will have chane in First_P
System.out.println(" Second Person Information");
Second_P.Display();
}
}
```

Output:

In this program, two objects, First_P and Second_P, are created. Object First_P is initialized with some values, and then the Display method is called on object First_P. It displays the value of data members of object First_P. Subsequently, object First_P is assigned to object Second_P as a reference variable. Here both objects First_P and object Second_P refer to the same object. Thus any change made in object First_P or Second_P will change the value in both the objects First_P and Second_P. You can see in the program that changes made in name and address by Second_P have also changed the values in First_P. Whenever this type of referencing of variables is used in a program, care should be taken while changing the values of instance variables of the object being referred.

☞ Check Your Progress - 1

1) Explain how objects are defined in Java.

……

……

……

2) What is the advantage of having member data and member functions within the class?

……

……

……

3) What care should be taken in the case of object referencing?

……

……

……

4.3 INTRODUCING METHODS

From the programmer's point of view, a Java class is a group of values with a set of methods. The user of a class perform the functionality of the objects of that class only through the methods of that class. Methods are designed with the objective to fulfil a specific objective of large and complex calculations of the system. A method is created as a self-contained block of code which provide some specified function. Every method has a name, and the same method can be executed from many different objects, as per requirement. In Java, "a method describes the behavior of an object and is also a collection of statements that are grouped together to perform operations". Java program executes a method by calling it on objects. A method may or may not return a value. Also, the methods that do not return a value are to be called without assigning them to any variables. When a method which returns a value is called, its returned value should be assigned to a variable. Alternatively, you may directly print the returned value. Also, you may use the returned value in the evaluation of an expression.

4.3.1 General Methods

In Java, a method is defined as follows:

type name_of_method (*argument_list*)
{
// body of method
}

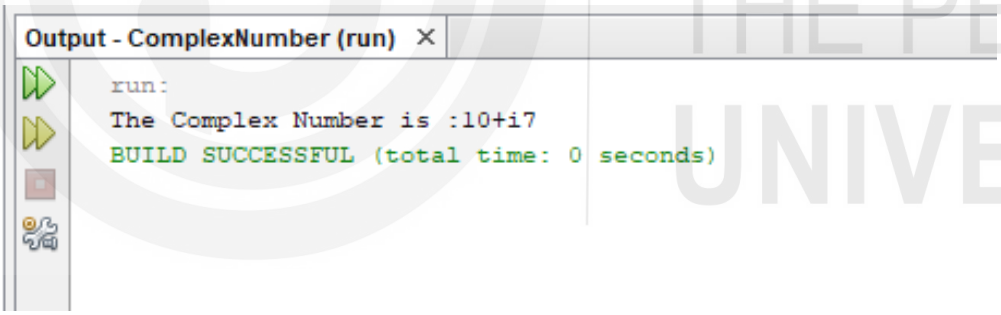
type specifies the type of data that will be returned by the method. This can be any data type, including class types. In case a method does not return any value, its return type must be void. The *argument_list* is a list of type and identifier pairs separated by commas. Arguments are basically variables that receive the values at the time of method invocation.

Now let us define a class to represent complex numbers. The complex class definition shown in the program given below illustrates how complex numbers can be represented. Two data member/variables *real* and *imag*, are declared. These represent the real and imaginary parts, respectively, of a complex number. In this program also, three methods, assignReal - assign values to the *real* variable and assignImag()- assign value to the *imaginary* variable and method showComplexl() shows the complex number respectively.

Java Program

```
package complexnumber;
class Complex
{
int real;
int imag;
void assignReal( int r)
{
real = r;
}
void assignImag( int i)
{
imag= i;
}
void showComplex ( )
{
System.out.println("The Complex Number is :"+ real +"+i"+imag);
}
}
public class ComplexNumber
{
public static void main(String[] args)
{
Complex R1 = new Complex();
R1.assignReal(10);
R1.assignImag(7);
R1.showComplex();
}
}
```

Output:



4.3.2 Static Methods

A static method is a characteristic of a class, not of the objects it has created.

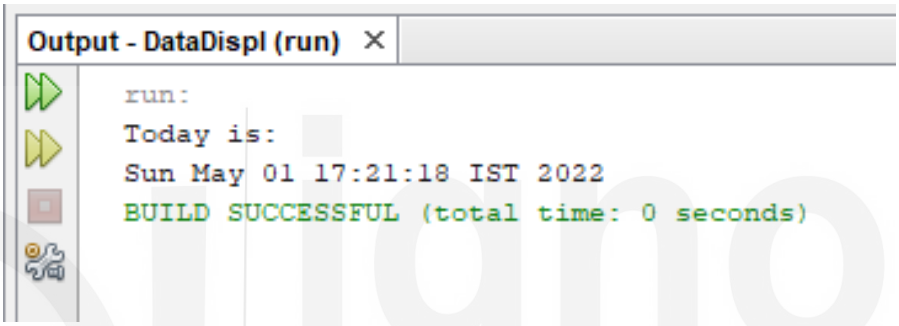
To refer to instance methods and variables, you must instantiate the class first, then obtain the methods and variables from the instance. Static variables and methods are class variables or class methods since each class variable and each class method occur once per class. Non-static methods and variables belong to objects of the class. Instance methods and variables occur once per object(instance) of a class, or you can say every object has its own copy of instance variables. In a program, you may execute a static method without creating an object of the class. On the other hand, all other methods must be invoked using an object. Hence an object must exist before it can be used. As you have noticed, every Java application program has one main()method. This method is always static because Java starts execution from this method, and at that time, no object is created.

Let us see one example Java in which program:

Java Program

```
package datadispl;
import java.util.Date;
public class DataDispl
{
   public static void main(String[] args)
   {
      Date today = new Date();
      System.out.println("Today is:");
      System.out.println(today);
    }
  }
```

Output:



In this program, the last line of the main() method uses the **System** class from java.lang package to display the current date and time. See the line of code that invokes the *out.println ()* method.

System.out.println(today);

Now, look at the calling of *println()* method closely. When you call a static method, you write the class name instead of an object name.

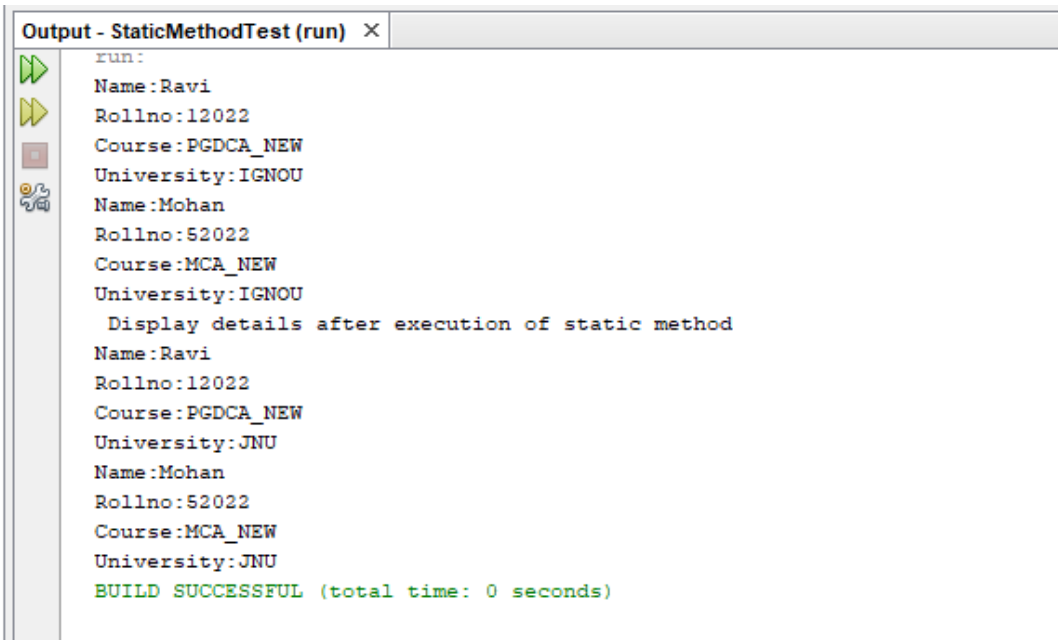
System.out refers to the out variable of the System class. You already know that, to refer to static variables and methods of a class, you use a syntax similar to the C and C++ syntax for obtaining the elements in a structure. You join the name of the class with the name of the static method or static variable using dot (.) for accessing it. The point which should be noticed here is that the application in using method "**System.out.println",** never instantiated the System class and that out is referred to directly from the class. This is because *out* is declared as *a static variable*: *a* variable associated with the class rather than with an instance of the class. When you call a static method, you write the class name instead of an object name.

One more example of the use of static method is shown in the following program in which the variable name University is changed using static method as it is a common variable that belongs to Student class rather than individual objects of the Student class. Therefore, change in one place will reflect in all the objects.

```
package staticmethodtest;
//Java Program, which demonstrates the use of a static method
class Student
{
 String Name;
 int Rollno;
```

```
String Course;
static String University = "IGNOU";
    //static method to change the value of static variable
static void change_University()
 {
    University = "JNU";
 }
  //method to display values of students
void display()
 {
   System.out.println("Name:"+Name);
   System.out.println("Rollno:"+Rollno);
   System.out.println("Course:" +Course);
   System.out.println("University:" +University);
 }
}
public class StaticMethodTest
{
public static void main(String args[])
{
Student S1 = new Student();
Student S2 = new Student();
S1.Name= "Ravi";
S1.Rollno =12022;
S1.Course = "PGDCA_NEW";
S2.Name= "Mohan";
S2.Rollno = 52022;
S2.Course = "MCA_NEW";
//calling display method  of Student Class
S1.display();
S2.display();
System.out.println(" Display details after execution of static method");
Student.change_University();//calling the static  change_University method  on class
name
S1.display();
S2.display();
 }
}
```

Output:



4.4 ENCAPSULATION

An *object* is *an* instance of a class. The class determines the features its objects will have and how those features are implemented in methods. A class encapsulates methods, data, and implementation of methods. You have already learned the basic concept of encapsulation in Unit 1 of this Block. The encapsulation is like a contract between the implementer of the class and the user of that class. Encapsulation help in hiding the details of how a piece of software works. A basic definition of encapsulation is "**the process of hiding all the details of a class definition that are not necessary to understanding how objects of the class are used**". In the context of programming, As a user of the software is not interested in the internal details of the system, a piece of software should be encapsulated so that only the necessary controls are visible and its details are hidden. Encapsulation is important as it simplifies the job of the programmer. It is important that for useful encapsulation, a class definition must be given in such a way that a programmer can use the class without seeing the hidden details. Encapsulation, when done correctly, neatly divides a class definition into two parts, which we call the *interface* and the *implementation*. The **class interface** tells programmers all they need to know to use the class in their programs. The class interface consists of the headings for the *public methods* and *public named constants* of the class, along with comments that tell to a user/programmer how to use these public methods and constants.

The **implementation** of a class consists of all of the private elements of the class definition, which includes the private instance variable of the class, along with the definition of both the public methods and private methods. Note that the class interface and the implementation of a class definition are not separate in your Java code but are mixed together.

Following are some of the basic guidelines to be used for defining a well-encapsulated class:

- Place a comment before the class definition that describes how the programmer should think about the class data and methods. For example, if the class describes an amount of money, the programmer should think in terms of rupee, dollars etc., and not in terms of how the class represents money.
- Declare all the instance variables in the class as private.
- Provide public accessor methods to retrieve the data by an object. Also, provide public methods for any other basic needs that a programmer will have for manipulating the data in the class.
- Try to give a comment before each public method to specify how to use the method.
- Make any helping methods use in the program as private.
- Write comments within the class definition to describe the implementation

Sometimes as a programmer, you may come up with a more efficient way to implement a method so that it may run faster. In such cases, you will need to change the implementation details of the class definition. When you have used the concept of encapsulation while defining your class, you should be able to go back and change the implementation details of the class definition without requiring changes in any program that uses the class (its methods). Also, as a learner this is a good way to test whether you have written a well-encapsulation class definition.

For example, in the case of online ticket booking systems for Train or Airlines, you may have a class for seat booking and payments. When there is any change in the rules for processing the ticket charge or payments, you need to modify the implementation without affecting the user experience.

4.5 ACCESS MODIFIERS

Access modifiers are keywords used to define the scope and behavior of the classes, methods and variables. You have seen the use of *public* keywords in many example Java programs so far. In Java, class definitions have different access levels. Access level determines which variables and methods in a class are available for access by the objects. In many cases, you need variables and code written for the internal consumption of the class and do not want to provide their access to the outside world. There are four possible access specifiers: *public, protected, private and default* access. About access specifiers, you will learn more in the next Block of this course. Below is the basic introduction to different access specifiers.

public: The *public* access refers to the data members and methods that can be freely accessed by code from any other class, except the class that inherits from this class and classes in the same package –which refers to a collection of classes – basically all the classes stored in the same directory.

protected: This access specifier provides controlled access to methods and variables outside the class. The *protected* methods and variables can be accessed by the same package or by any subclass (inherited class) outside of the package.

private: The *private* access means that the member will not be accessed by code in any other class, including classes that inherit from this one.

default: There is the fourth possibility, default access, which is used if no access specifier is mentioned.

Table 4.1 shows the access to members permitted by each modifier.

Modifier	Generally Used by	Basic Description
public	constructors, inner classes, methods and variables	This is most generic/liberal access level. Accessed by classes outside the package
protected	constructor, inner classes, methods and variables	This provide controlled access. Accessed by the same package or by any subclass outside of package
private	constructor, inner classes, methods and variables	This is most restrictive access level. Access only within class where they are declared.
No modifier (Default)	outer classes, inner classes, interface, constructor, method and variable	Access only within the package where they exist

Important point to remember that you cannot use more than one access modifier for a class, method or variable. Also, note that all the modifiers are optional, and it is up to the requirement of the programmer to use them as per the requirement of their design/solution. At the same time, it is a good programming style to use access specifiers as per need.

Need for Private Methods

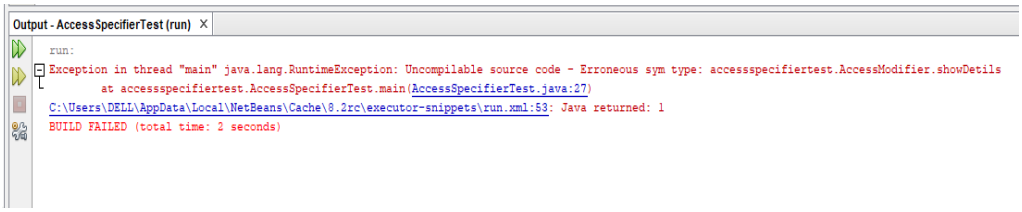
When you are implementing a class, many a time you need to define all data fields *private* because public data are dangerous. Also, as a good practice, data access

outside the class should be controlled to meet the basic philosophy of Object Orientation, which advocates for data protection. But why do we need private methods? In practice, most methods are public, but you need private methods for internal consumption of the class in certain circumstances. The private methods provide you the option to break up the code for computation into separate helper methods. Typically, these helper methods affect the functionality of public methods that work as the program's public interfaces. The private methods are implemented using *private* access specifier. You can see the below java program in which a private method named showDetails( ) is called by an object, and when you compile and run this program, as output, you will get compile-time error. This is because a private method can not be accessed from outside of the class. Similar test you may do by trying to access the private data members from outside of the class and observe the compiler error.

Java Program

```
package accessspecifiertest;
class AccessModifier
{
private String Name = "Shrikant";
private char Sex = 'M';
private int Age = 16;
public void displayDetails ( )
{
System.out.println ("Name :" +Name);
System.out.println ("Sex :" +Sex);
System.out.println ("Age :" +Age);
}
private void showDetails( )
{
System.out.println ("Name :" +Name);
System.out.println ("Sex :" +Sex);
System.out.println ("Age :" +Age);
}
}
public class AccessSpecifierTest
{
public static void main (String args [] )
{
AccessModifier  AM = new AccessModifier ( );
//AM.displayDetails();
AM.showDetils ( );
 }
}
```

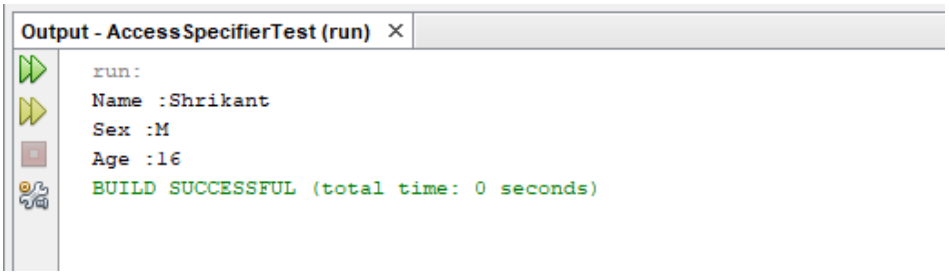
Output:



Instead of calling
AM.showDetails();

in the above program if you call AM.displayDetails();, you will get following result.

Output:



4.6 CONSTRUCTORS IN JAVA

So far you have initialized the instance variables of the objects by assigning values to individual variables. Now we will discuss use of constructors for the initialization of objects (instance variables). A constructor is used for initialising the objects at the time of their creation. Constructors have the same name as the name of their class. Once a constructor is defined and used for object creation, it is automatically called, and the memory is allocated before the **new** operation completes. The constructor does not have any return type. It has an implicit return type of the class in which it is defined. In other words a constructor returns a class type. The job of a constructor is to initialize the instance variables of an object so that the created object is usable just after creation. For programmers, object construction is important, as it provides ready objects which can be used in solving the problem. Now let us see an example program, which we have used earlier in this unit and try to define a constructor for it.

In the following program class Complex does not have any constructor, and for initializing imaginary and real variables, two methods have been defined and used. Both the methods have been invoked on the object one by one for initializing the object's instance variables. In place of methods now, we will define a constructor and use it for initialization. We will make necessary changes in the class Complex. We will remove methods used for initializing variables and use one method having the same name of the class, i.e., Complex with two arguments.

Original Java Program

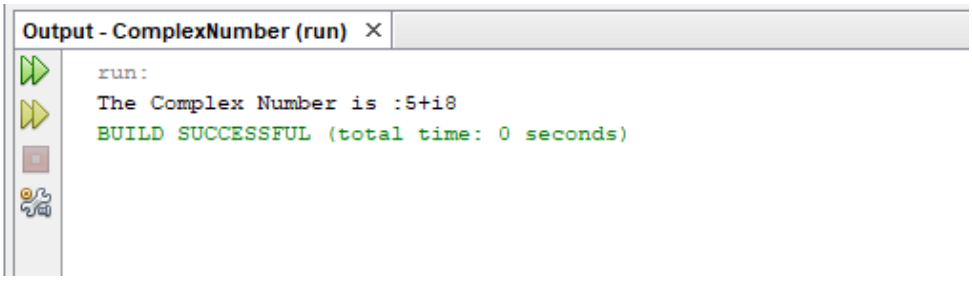
```
package complexnumber;
class Complex
{
int real;
int imag;
void assignReal( int r)
{
real = r;
}
void assignImag( int i)
{
imag = i;
}
void showComplex ( )
{
System.out.println("The Complex Number is :"+ real +"+i"+imag);
```

```
}
}
public class ComplexNumber
{
public static void main(String[] args)
{
Complex R1 = new Complex();
R1.assignReal(10);
R1.assignImag(7);
R1.showComplex();
}
}
```

Modified Java Program in which constructor has beed defined

```
package complexnumber;
class Complex
{
int real;
int imag;
Complex( int r, int i) // defining constructor
{
real = r;
imag = i;
}
void showComplex ( )
{
System.out.println("The Complex Number is :"+ real +"+i"+imag);
}
}
public class ComplexNumber
{
public static void main(String[] args)
{
Complex R1 = new Complex(5,8);
R1.showComplex();
}
}
```

Output:



If you compare this program and the previous program, you will find that in this program, the instance variable of object R1 is initialized with the help of constructor Complex (5, 8), value 5 has been assigned to the real variable, and 8 has been assigned to the imag variable. In the previous program, to initialize these variables, methods assignReal() and assignImag() were used.

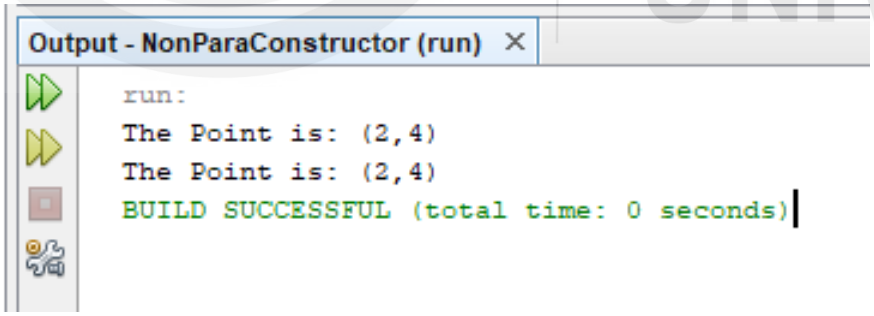
Further, if we look into the requirements of differet types of constructors, we can be defined in two ways. First a constructor which *may not have a parameter*. This type

of constructor is known as *non-parameterized constructor*. The second, a constructor which may take parameters. This type of constructor is known as *parameterized constructor*. If non-parameterized constructor is used for object creation, instance variables of the object are initialized by fixed values at the time of definition of constructor itself. You can see the following program in which objects of class Point is created using the non-parameterized constructor.

Example Java Program

```
package nonparaconstructor;
class Point
{
int i;
int j;
Point() // Non-parameterized constructor
{
i = 2;
j = 4;
}
void Display_Point()
{
System.out.println("The Point is: ("+i+","+j+")");
}
}
public class NonParaConstructor
{
public static void main( String args[])
{
Point  A1 = new Point();
Point A2 = new Point();
A1.Display_Point();
A2.Display_Point();
}
}
```

Output:



constructor Point( ) in above program is a non-parameterized constructor. Both the objects A1 and A2 are created by using the constructor Point( ). You can use this type of constructors to initialise those objects for which the initial value of instance variables is already known to you. When you need to give values of the instance variables at the time of object creation, use parameterized constructors.

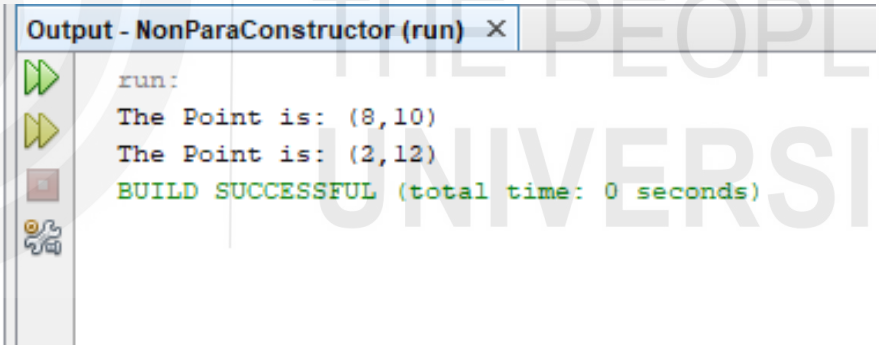
If you want to create objects of the class Point by giving your initial values of x and y coordinates, you can use parameterized constructor. You can do this by modifying the above program with the following changes.

1. In place of non-parameterized constructor, define parameterized constructor.
2. Pass appropriate values as arguments to constructors.

Java Example Program

```
package nonparaconstructor;
class Point
{
int i;
int j;
Point( int a, int b) // Parameterized constructor
{
i= a;
j= b;
}
void Display_Point()
{
System.out.println("The Point is: ("+i+","+j+")");
}
}
public class NonParaConstructor
{
public static void main( String args[])
{
Point  A1 = new Point(8,10);
Point A2 = new Point(2,12);
A1.Display_Point();
A2.Display_Point();
}
}
```

Output:



In this program, you can see that points A1 and A2 are initialized by values of programmer's choice by using parameterized constructor.

4.6.1 Overloading Constructors

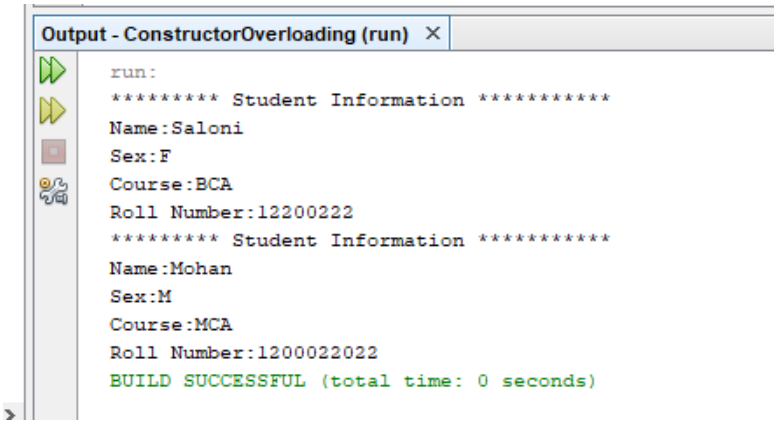
You may ask a very valid question here: Can we have more than one constructor in a class? The answer is yes, you may have more than one constructor in a class. You have to keep all the constructors in a class by either having different types of arguments in the constructor or a different number of arguments in the constructors. This is essential because it would not be possible to identify which of the constructors is being invoked without this. These differences in the constructor's signature will help the compiler differentiate the constructors by considering the number of parameters passed to the constructor or the type of parameters passed to the constructors. "Having more than one constructor in a single class is known as constructor overloading". Constructor overloading enables the programmer to create

objects with the desired number of attributes to be initialized. Basically, "constructor overloading is a technique in which a class can have any number of constructors that differ in parameter lists". In the program given below, in the class Student, more than one constructor are defined.

Example Java Program

```
package constructoroverloading;
class Student
{
String name;
int roll_no;
char sex;
String course;
Student( String n, int r, String c)
{
name = n;
roll_no  = r;
course = c;
sex = 'F';
}
Student(String n, int r, char c )
{
name  = n;
roll_no = r;
course = "MCA";
sex =c;
}
void Student_Info( )
{
System.out.println("********* Student Information **********");
System.out.println("Name:"+name);
System.out.println("Sex:"+sex);
System.out.println("Course:"+course);
System.out.println("Roll Number:"+roll_no);
}
}
public class ConstructorOverloading
{
public static void main(String[] args)
{
Student Student1 = new Student("Saloni", 12200222,"BCA");
Student Student2 = new Student("Mohan", 1200022022,'M');
Student1.Student_Info();
Student2.Student_Info();
}
}
```

Output:



Both the constructors of this program can be used for creating two different objects. Here, different objects have different values assigned to instance variables of objects during their initialization through constructors. By using the constructor with three arguments three instance variables name, roll_no and course, are initialized. In another constructor, with the three arguments instance variables name and roll_no and sex are initialized. While calling, these constructors are identified by the type of arguments passed while constructor call.

You can observe that methods and constructors look similar, but there are some differences between them. These differences are given in the following table.

Table 4.2: Difference between constructor and method

Constructor	Method
A constructor is used to initialize an object.	A method is used to implement the behavior of an object.
The constructor must not have a return type.	The method must have a return type.
Constructor is invoked implicitly or explicitly.	A method is invoked explicitly.
The Java compiler provides a default constructor if there is no constructor.	The compiler does not provide a method in any case.
The constructor name must be the same as the class name.	In Java method name may or may not be the same as the class name. But it is not recommended to have the method name the same as the object.

☞ Check Your Progress- 2

1) Explain the use of the constructor with the help of a program.

……

……

……

2) Write a program in Java to create SavingBankAccount class, which defines two different constructors to create objects and also defines a method to display details of the objects.

……

……

……

3) Run the following program and explain its output.

//Java Program

```
package boxttest;
class Box
{
int height;
int depth;
int length;
Box ( ) // Non parameterized constuctor
{
height=30;
depth=20;
length=10;
}
Box (int i, int j) //Parameterized constructor
{
height=i;
depth=j;
}
Box (int i, int j, int k) //Parameterized constructor
{
height=i;
depth=j;
length=k;
}
}
public class BoxtTest
{
public static void main (String args [ ] )
{
Box bx = new Box ( ) ;
System.out.println ("Depth of Box- bx : "+bx.depth);
System.out.println ("Height of Box- bx: "+bx.height);
System.out.println ("Lengtht of Box- bx: "+bx.length);
Box bx1 = new Box (10,15 ) ; // custructor call
System.out.println ("Height of Box- bx1:"+bx1.height);
System.out.println ("Depth of Box- bx1:"+bx1.depth);
System.out.println ("Length of Box- bx1: "+bx1.length);
Box bx2= new Box (12, 15, 20);//cunstrutor call
System.out.println ("Height of Box-bx2:"+bx2.height);
System.out.println ("Depth of Box- bx2:"+bx2.depth);
System.out.println ("Length of Box- bx2:"+bx2.length);
}
}
```

……

……

……

4.7 THIS KEYWORD

Java provides the keyword ***this*** that gives reference to the current object within the body of a method. By using ***this*** on the current object, programmers would be able to access the instance variables of that object. If a method wants to refer to the object through which it is invoked, it can refer to it by using ***this*** keyword. As a

programmer, you know it is illegal to have two variables of the same name within the same scope in a program. Suppose local variables in a method, or formal parameters of the method, have the same name(overlap) as the name of instance variables of the class, then to differentiate between local variables or formal parameters and instance variables, **this** keyword is used. The reason to use **this** keyword to resolve any name conflict that might occur between instance variables and local variable, because ***this*** keyword can be used to refer to the objects directly.

Also, *this* keyword would allow the programmer to pass the current object as an argument to another method. The *this* keyword can be used anywhere the current object might appear. The dot(.) notation is used to refer to the object's instance variables.

Use of *this* keyword

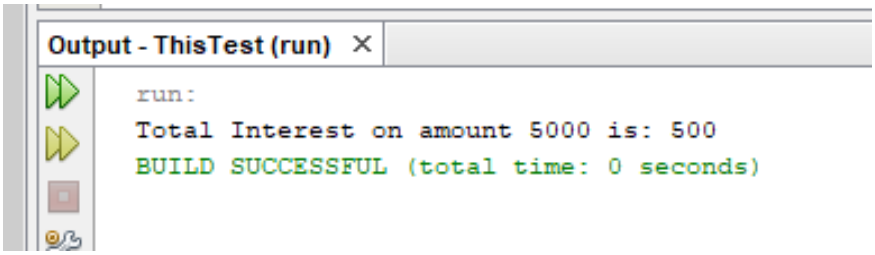
- this.memberdata; // here memberdata is instance variable for *this*
- this. DisplayDetails(this); // method DisplayDetails(.) defined in this class will be called, and passes it to the current object.
- return this; // returns the current object

You can see this in the program given below. This program has one variable named ***rate*** and a method ***Total_Interest*** in the Account. The method ***Total_Interest*** is also having one local variable named ***rate***. To avoid conflict between both the **rate variables, *this*** keyword has been used with ***rate*** variable of the class Account.

Example Program

```
package thistest;
class  Account
{
int rate ;
int amount;
int interest;
Account( int r, int a)
{
rate = r;
amount =a;
}
void Total_Interest( )
{
int rate = 5;
rate = this.rate+rate; // use of this keyword
interest = rate*amount/100;
System.out.println("Total Interest on amount "+amount+" is: "+interest);
}
}
public class ThisTest
{
 public static void main(String[] args)
{
Account  Acc1 = new Account( 5, 5000);
Acc1.Total_Interest();
}
}
```

Output:

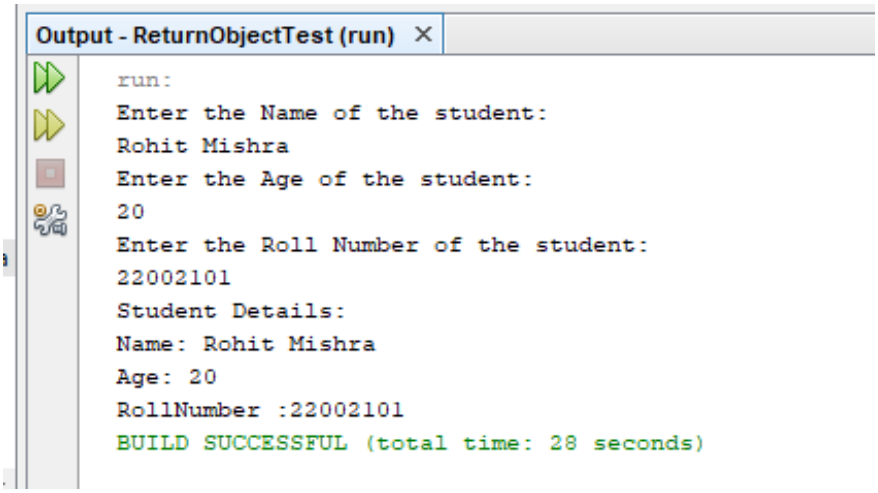


The following java program demonstrates how to return an object using this keyword. Last line of code in method SetValues() i.e *return.this* is returning the current object.

Example Java Program

```
package returnobjecttest;
import java.util.*;
class BCAStudent
{
  private String Name;
  private int Age;
  private int  RollNo;
  public BCAStudent SetValues() // method to set values of data members
  {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the Name of the student: ");
    String name = sc.nextLine();
    System.out.println("Enter the Age of the student: ");
    int age = sc.nextInt();
    System.out.println("Enter the Roll Number of the student: ");
    int rollNo = sc.nextInt();
    this.Name = name;
    this.Age = age;
    this.RollNo = rollNo;
    return this;
  }
  public void DisplayDetails()
  {
    System.out.println("Name: "+Name);
    System.out.println("Age: "+Age);
    System.out.println("RollNumber :"+RollNo);
   }
}
public class ReturnObjectTest
{
 public static void main(String args[])
 {
    BCAStudent BCAObj = new BCAStudent();
    BCAObj = BCAObj.SetValues();
    System.out.println("Student Details:");
    BCAObj.DisplayDetails();
  }
}
```

Output:



4.8 USING OBJECTS AS PARAMETERS

Sometimes you need to pass objects as a parameter to methods, and also, objects may be returned from the methods. Let us look at these two aspects one by one.

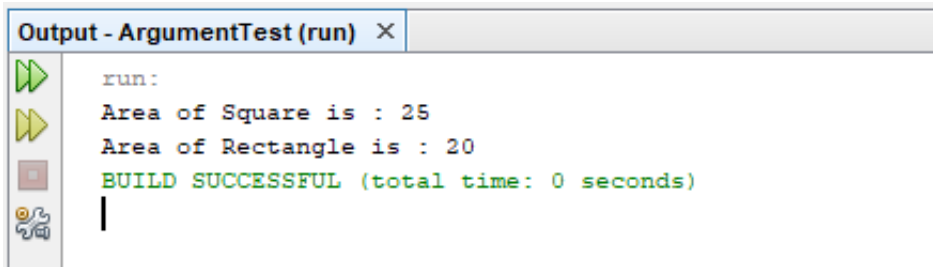
4.8.1 Object Passed as Argument in Methods

During problem-solving, when you define methods, sometimes you need to pass objects as arguments so that you may generalize a method. These generalized methods can be used for performing operations on a variety of data.In the program given below for finding the area of objects square and rectangle, the methods Area_Square and Area_Rectangle for finding the area of square and rectangle respectively are defined. In this program, primitive variables are passed as an argument rather than objects.

Java Program

```
package argumenttest;
class Area
{
int Area_Square( int i)
{
 return i*i;
}
int Area_Rectangle(int a,int b)
{
 return a*b;
}
}
public class ArgumentTest
{
public static void main(String args[])
{
 Area a = new Area();
 int area;
 area = a.Area_Square(5);
 System.out.println("Area of Square is : "+area);
 area = a.Area_Rectangle(5,4);
 System.out.println("Area of Rectangle is : "+area);
}
}
```

Output:

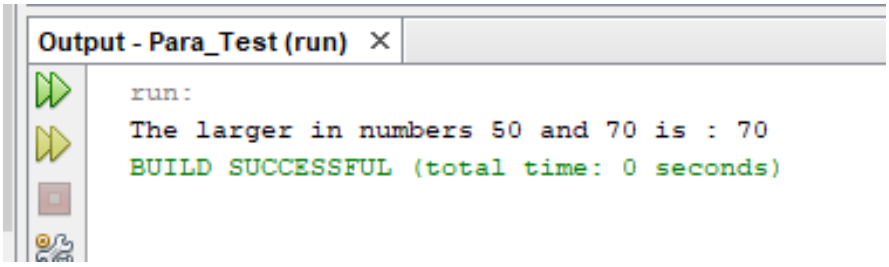


You can see that passing parameters in methods Area_S and Area_R is *call by value*. In your first semester, you have already studied two basic ways call by value and call by reference of parameter passing in functions in the course MCS 201. Pass-by-value means that when you call a method, a copy of the *value* of each of the actual parameter is passed to the method. You can change that copy inside the method, but this will have no effect on the actual parameters. In Java everything except the absolute value is passed by reference. You can see in the program given below in the method max, that the two parameters are passed by value. The values of variables are of primitive type.

Example Program

```
package para_test;
class Para_Test
{
static    int  max(int a, int b)
{
if (a > b)
return a;
else
return  b;
}
public static void main(String[] args)
{
int num1 = 50, num2 = 70, num3;
num3 = max( num1, num2);
System.out.println("The larger in numbers "+num1 +" and "+num2+" is : "+num3);
}
}
```

Output:

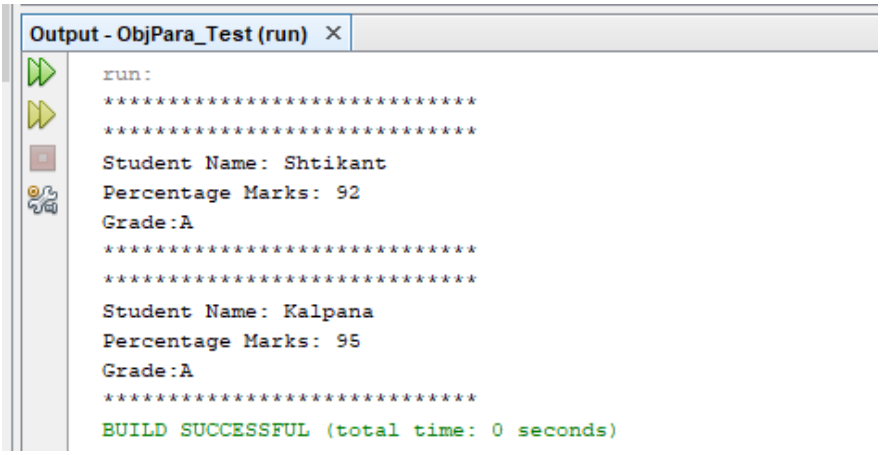


The objects passing as a parameter in Java are called variable passing by reference. As mentioned earlier in this section in Java, we can pass a reference of the object to the formal parameter in the methods. In such a case, any changes to the local object inside the method will modify the object that was passed to the method as argument. In the program given below to method Set_Grade , object of the class Marks is passed as argument. In this method, the instance variable *grade* of the object passed is assigned some value, and this assignment of value changes the object itself.

Java Program:

```
package objpara_test;
class Marks
{
String name;
int percentage;
String grade;
Marks(String n, int m)
{
name = n;
percentage = m;
}
void Display()
{
System.out.println("******************************");
System.out.println("Student Name: "+name);
System.out.println("Percentage Marks: "+percentage);
System.out.println("Grade:"+grade);
System.out.println("******************************");
}
static void Set_Grade(Marks m)
{
if (m.percentage >= 60)
m.grade ="A";
else if( m.percentage >=40)
m.grade = "B";
else
m.grade = "F";
}
}
public class ObjPara_Test
{
public static void main( String a[])
{
Marks ob1 = new Marks("Shtikant",92);
Marks ob2 = new Marks("Kalpana",95);
Marks.Set_Grade(ob1);
System.out.println("******************************");
ob1.Display();
Marks.Set_Grade(ob2);
ob2.Display();
}
}
```

Output:



4.8.2 Returning Objects

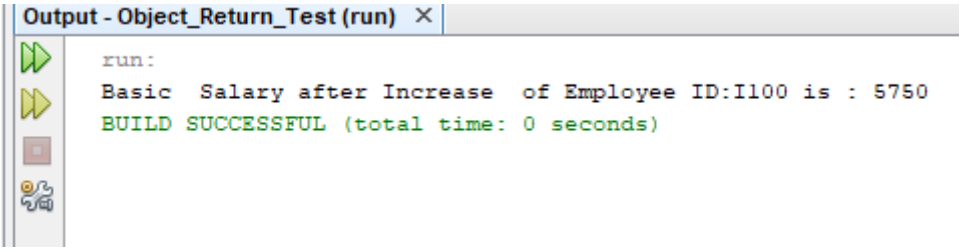
Similar to other primitive data types, in Java a method can return data of class type, i.e. object of a class. When such a method is called, an object returned from that method can be stored in any other object of that class, It is similar to the value of basic type returned is stored in a variable of that data type. You can see in the program below where an object of class Salary is returned by method Increae_Salary. Also you may notice that to this method object is passed as an argument.

Salary Increase_Salary (Salary s) // object is passed as argument and also object is returned from the method.

```
package object_return_test;

class Salary
{
int basic ;
String Emp_id;
Salary( String a, int b)
{
Emp_id = a;
basic = b;
}
Salary  Increase_Salary ( Salary s )
{
s.basic = basic*115/100;
return s;
}
}
public class Object_Return_Test
{
public static void main(String[] args)
{
Salary s1 = new Salary("I100",5000);
Salary s2; // A new salary object
s2 = s1.Increase_Salary( s1);
System.out.println("Basic  Salary after Increase  of Employee ID:"+ s2.Emp_id +" is
: " + s2.basic);
}
}
```

Output:



4.9 GARBAGE COLLECTION

One very important feature of Java is garbage-collection. It is also known as a garbage-collected heap. In the garbage collection process, Java free the dynamically allocated memory which is no longer referenced. Java takes care of reclaiming

memory that is no longer in use by garbage collection. It shields the substantial complexity of memory allocation and the garbage collection process from the programmer. Because the heap is garbage-collected, Java programmers do not have to explicitly free the allocated memory. In Java "*new*" operator allocate memory to object at the time of creation on the heap at run time. The JVM's heap stores all the objects created by an executing Java program. In any program when an object no longer in reference, the heap space it occupies must be freed so that the space is available for new objects in the program or system.

Advantages of garbage collection

Giving the job of garbage collection to the JVM has several advantages. Two main advantages are:

1. First, it can make programmers more productive.
2. Second advantage of garbage collection is that it ensure program integrity.

Disadvantages of garbage collection

One major disadvantage of garbage collection is that it adds overhead to the system, which adversely affects program performance. As the JVM has to keep track of objects that are being referenced by the executing program and free unreferenced objects on the fly. This activity will likely eat up more CPU time than would have been required when the program explicitly freed unrefined memory. The second disadvantage of garbage collection is that "the programmers in a garbage-collected environment have less control over the scheduling of CPU time" as it is devoted to freeing objects that are no longer needed.

The JVM has a low-priority thread that checks the graph of objects to find orphaned objects. In the process, the objects those are found are marked as available to be collected. A separate process is run by JVM, when necessary, to reclaim that memory.

4.9.1 the finalize ( ) method

You can request garbage collection by calling System.gc( ). Calling this method is merely a request for garbage collection. It may or may have any effect; it is not guaranteed to run when one asks for garbage collection.

Sometimes some objects have to take independent resources therefore before the garbage collector takes the object. The programmer needs to have provision take independent resources from those objects before garbage collection of that object. To perform this operation, finalized () method is used. The finalized () is called by the garbage collector when it determines no more references to the object exist.

fnalized() method has the following properties:

1. Every class inherits the finalize() method from Java.lang.Object.
2. The garbage collector calls this method when it determines no more references to the object exist.
3. The **Object class** finalize method performs no actions but it may be overridden by any derived class.
4. Normally it should be overridden to clean up **non-Java resources**, i.e. closing a file, taking file handle, etc.

You can specify code to run when an object is collected by writing a finalize( ) method in a class. There is no guarantee that this method will ever run. The advantages of garbage collection are.

- The programmer does not need to worry about dereferencing an object.
- It is done automatically by the JVM.
- Increases memory efficiency and decreases the chances of memory leak.

One can add a finalize method to any class. The finalize method will be called before the garbage collector sweeps away the object.

The syntax for finalize () method is as follows:

```
protected void finalize ( )
 {
// finalize-code
}
```

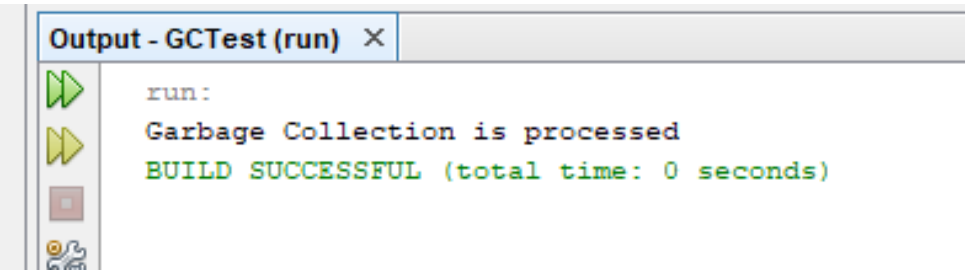
gc ( ) method

The static gc ( ) method is defined in the System class to encourage the JVM to do some garbage collecting and recover the memory that the objects occupy using System.gc ( ). This is a best-effort deal on the part of the JVM for memory management. When the gc ( ) method returns, the JVM has tried to reclaim the space occupied by discarded objects, but there is no guarantee that it has all been recovered. It totally depends on the processing of JVM.

Java Program

```
package gctest;
public class GCTest
{
public static void main (String [ ] args )
{
        GCTest  Obj = new GCTest ( );
        Obj= null;
        System.gc ( );
}
protected void finalize ()
{
System.out.println ("Garbage Collection is processed");
}
}
```

Output:



☞ **Check Your Progress- 3**

1) Run the following code and show the result.

```
package usethis;
class Student
 {
String Name;
int RollNo;
String Course;
Student (String Name,int RollNo, String Course)
 {
this.Name = Name;
this.RollNo = RollNo;
this.Course = Course;
}
void DisplayDetails ( )
{
System.out.println (" Name of Student:" + Name);
System.out.println ("Student Rellnumber:" + RollNo);
System.out.println ("Course Enrolled:" + Course);
 }
}
public class UseThis
{
public static void main (String args [ ])
{
Student s1 = new Student ( "Manish",202022,"MCA") ;
Student s2 = new Student ("Akshay",202221,"BCA") ;
System.out.println("**********Student 1 Details**********");
s1.DisplayDetails ( ) ;
System.out.println("**********Student 2 Details**********");
s2.DisplayDetails ( ) ;
}
}
```

……..

……..

……..

2) Run the following code and show the result.

```
package returnobjtest;
class Shape
{
int length;
int breadth;
Shape(int i, int j)
{
length = i;
breadth = j;
}
Shape getShapeObject(Shape s )
{
Shape spo;
spo = s;
```

```
return spo;
}
}
public class ReturnObjTest
{
public static void main (String args [ ] )
{
Shape ob1 = new Shape(40, 50);
Shape ob2;
ob2 = ob1.getShapeObject(ob1 ) ;
System.out.println ("ob1.length: "+ ob1.length);
System.out.println ("ob1.breadth: "+ ob1.breadth);
System.out.println ("ob2.lenght:" + ob2.length);
System.out.println ("ob2.breadth:"+ ob2.breadth);
}
}
```

3) What is advantage of garbage collection?

……

……

……

4.10 SUMMARY

This unit explained defining classes and methods in Java. It has been demonstrated how to assign objects as a reference variable. Every object has its state and behavior. Objects behavior are defined inside the class in terms of member functions. Different types of methods, such as general and static methods, have been explained. The concept of encapsulation has been explained in this unit with example. The use of different access specifiers in Java programming is demonstrated with examples. This unit also discussed the defining constructors with the help of example programs. This unit explained arguments passing to the function. Towards the end, this unit explains the garbage collection feature of Java.

4.11 SOLUTIONS/ANSWERS

☞ Check Your Progress - 1

1) Object definition in Java is a two-step process:

i Declare variable of Account type (class type).

Account Acc // declaration of object named Acc of type Account

ii. Allocate required space to the object using new operator:
Acc = new Account(); // allocating memory.

2) Objects are the basic elements of a program used for solving the problem. Objects are variables of class type and known as instance of classe. Objects should be considered a black box with public interfaces to access their functionality. Interfaces provide the functionality with the help of member functions.

In object oriented programming, data is given importance. Member functions and member data are kept inside the class to restrict access of data members and member functions from outside of the class or avoid accidental

modification. Data hiding is ensured by keeping data inside the class. Definition of member functions/methods ares kept inside the class, which provides the scope of modification in the definition of a method without affecting the outside world.

3) In Java one object can be used as a reference to another object of the same type i.e. both are of the same class type. When objects are used as a reference, care should betake in performing operations on objects because if there is any change invalues data members of one object, values of respective data members of the second object also get changed.

☞ Check Your Progress - 2

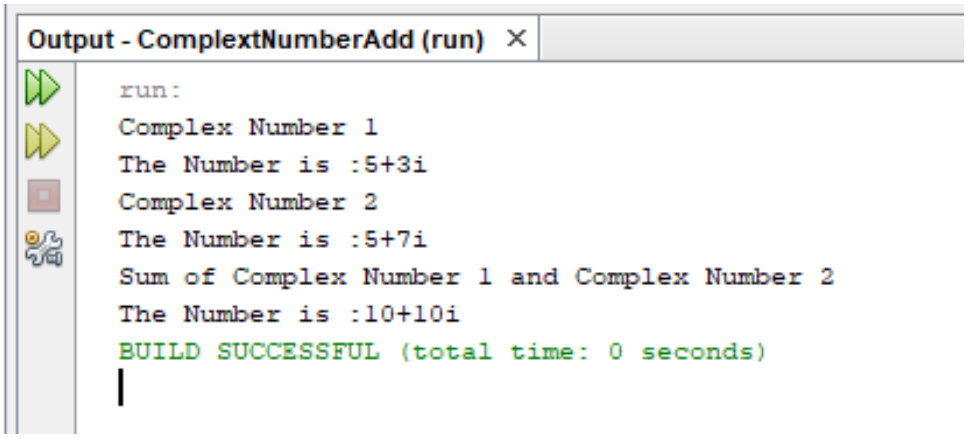
1) To intialise the objects, constructors are used in Java programs. Using constructors, you get the objects which are ready to use. Following is a Java program in which objects of Complex class are created using constructor defined in this class.

Java Program

```
package complextnumberadd;

//program
class Complex
{
 int real;
int imaginary;
Complex ( int r, int i)
{
real = r;
imaginary  = i;
}
void ShowNumber()
{
System.out.println("The Number is :" + real+"+" +imaginary+"i");
}
}
public class ComplextNumberAdd
{
public static void main(String args[])
{
Complex C1= new Complex(5,3);
Complex C2 = new Complex (5,7);
Complex C3 = new Complex (0,0);
System.out.println("Complex Number 1");
C1.ShowNumber();
System.out.println("Complex Number 2");
C2.ShowNumber();
C3.real = C1.real +C2.real;
C3.imaginary = C1.imaginary +C2.imaginary;
System.out.println("Sum of Complex Number 1 and Complex Number 2");
C3.ShowNumber();
}
}
```

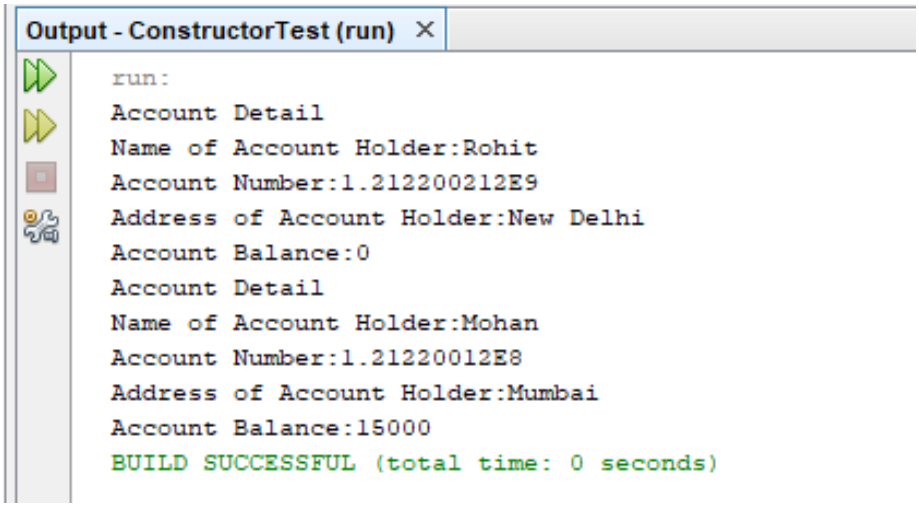
Output:



2) In this program there are two different constructors of SavingBankAccount class are defined. The method Display_Details( ) is defined to display details of the objects.

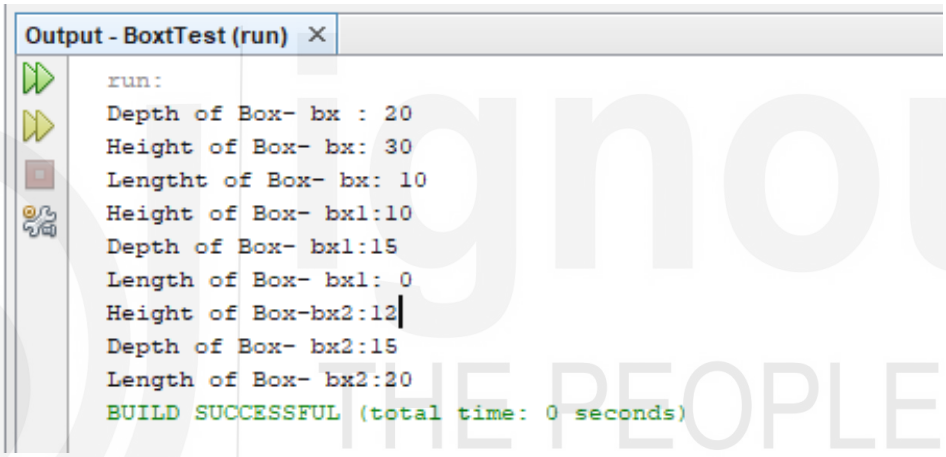
```
package constructortest;
class SavingBankAccount
{
 private String Name;
 private double  Account_No;
 private String Address;
 private int  Balance;
SavingBankAccount(String n, double a, String addr)
{
Name =  n;
Account_No = a;
Address = addr;
}
SavingBankAccount (String n, String addr, double a , int b)
{
Name =  n;
Address = addr;
Account_No = a;
Balance = b;
}
void Display_Details()
{
System.out.println("Account Detail");
System.out.println("Name of Account Holder:"+Name);
System.out.println("Account Number:"+Account_No);
System.out.println("Address of Account Holder:"+Address);
System.out.println("Account Balance:"+Balance);
}
}
public class ConstructorTest
{
public static void main( String args[])
{
SavingBankAccount AC1 = new SavingBankAccount("Rohit", 1212200212,"New Delhi");
SavingBankAccount AC2 = new SavingBankAccount( "Mohan", "Mumbai", 121220012,15000);
AC1.Display_Details();
AC2.Display_Details();
}
}
```

Output:



3)

Output:



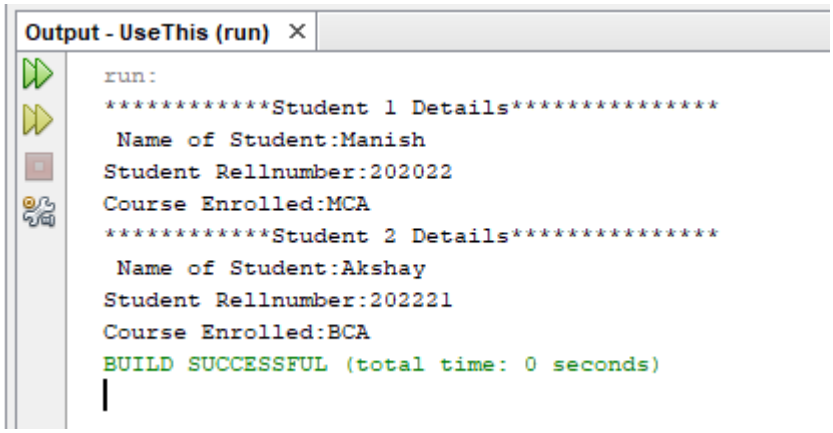
The calling of different constructors is demonstrated in the program. For creating object bx1 when the constructor with two variables is called, the variable *length* is assigned the default value.

☞ Check Your Progress- 3

1)

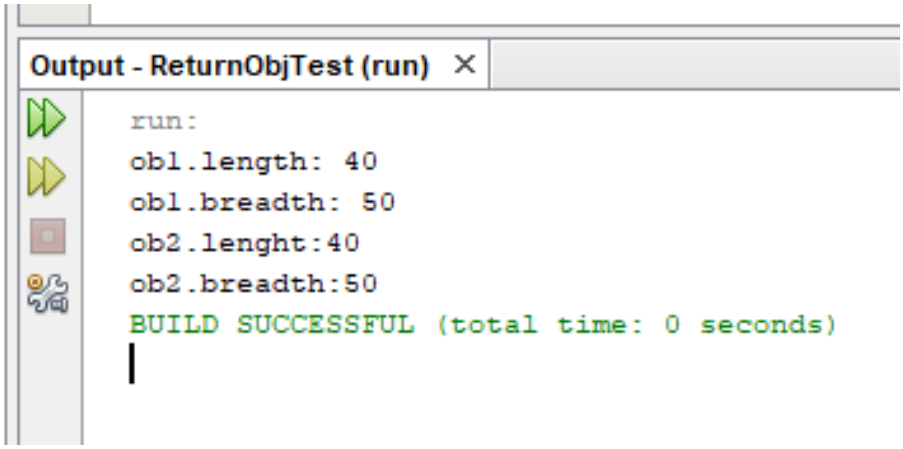
Output:

The following output demonstrates the creation of objects using constructors and their use in the program.



2)

Output:



Above is the output of the program, which demonstrates how an object is returned from a method.

3) There are two major advantages of automatic garbage collection
   i. Increase programmer productivity.
   ii. Ensure program integrity.

4.12 REFERENCES/FURTHER READINGS

- Herbert Schildt, "Java The Complete Reference", McGraw-Hill, 2017.
- Savitch, Walter, " Java: An introduction to problem solving & programming", Pearson Education Limited, 2019.
- S.Sagayaraj,R. Denis, P.Karthik and D.Gajalakshmi, "Java Programming for Core and Advanced Learners", University Press, 2018.
- Neil, O. , "Teach yourself JAVA", Tata McGraw-Hill Education, 1999.

ignou
THE PEOPLE'S
UNIVERSITY