2020

JAVA Programming

Sourav Kumar Giri

Email- sourav.giri4@gmail.com

Disclaimer

Contents of this material have been prepared using JAVA 2: Complete Reference by Herbert Schildt , TMH Publications. Any typological errors may be brought to the notice and also suggestions are welcomed at sourav.giri4@gmail.com

This material is designed considering syllabus of B.Tech/ MCA/ B.Sc/ BCA/ M.Tech courses offered by various universities and colleges in India.

Note: *Any typing error or suggestions are welcomed at* sourav.giri4@gmail.com

CONTENTS

Chapter 1 INTRODUCTION TO PROGRAMMING LANGUAGE

Programming language

A programming language is a language that is designed to be used (read and written) by humans to create programs that can be executed by computers. In other words we can say that programming languages provides the way so that the users may interact with the computer to give it commands and instructions to perform certain tasks. There are two main types of computer programming languages.

- Low-level languages
- High-level languages

Low Level Programming Languages

These languages are near to computer hardware and far from human languages. Computer can understand these languages easily. Following are two low-level languages:

- Machine Language
- Assembly Language

Machine Language

A computer language in which instructions are written in binary form (0 and 1) is called machine language. It is the only language that is directly understood by the computer. Machine language is the native language of computer. Machine language is also known as first generation language.

Advantages of Machine Languages

- Very fast program execution: Because the machine language is the native language of computer that computers directly understand and execute. There is no need of a translator program, because computer can already understand and execute the machine language instructions directly without the need of translation.

Disadvantages of Machine Languages

- Machine Language is difficult to understand
- Machine Language is difficult to learn
- Programs of Machine Language are difficult to modify
- Machine Language requires deep knowledge of hardware
- Programs of Machine Language are difficult to remove errors
- Programs of Machine Language are Machine dependent

Assembly Language

Assembly language is a low-level language. In assembly language, symbols are used instead of binary code. These symbols are easy to remember. For example Add instruction is used to add two numbers. Assembly language is also known as second generation language

Advantages of Assembly Language

- Assembly language programs are executed with fast speed
- Assembly language programming is easier to learn, understand and modify than machine language

Disadvantages of Assembly Language

- Assembly language programs are machine dependent
- Assembly language programming requires deep knowledge of hardware

High Level Programming Languages

A type of language that is close to human languages is called high level language. High-level languages are easy to understand. Instructions of these languages are written in English-like words e.g. Print, Display, Write etc.

Examples of High Level Programming Languages

- COBOL
- BASIC
- PASCAL
- C Programming Language
- C++
- JAVA
- Visual Basic

Advantages of High Level Programming Languages

- High Level Programming Languages are Easy to learn and understand
- Programs written in High Level Programming Languages are Easy to modify
- It is Easy to remove errors in the Programs written in High Level Programming Languages
- Programs written in High Level Programming Languages are Machine independent
- High Level Programming Languages have Better documentation

Disadvantages of High Level Programming Languages

- A disadvantage of High Level Programming Languages is slower program execution.
- High Level Programming Languages provide programming facilities for performing certain operations.

Few examples of High Level Programming Languages

- **FORTRAN:** FORTRAN stands for **FOR**mula **TRAN**slation. It is mainly used for writing applications related to science and engineering fields.
- **BASIC:** BASIC stands for Beginners All-purpose Symbolic Instruction Code. It was invented in late 1960's to teach programming skills to students.

- **COBOL:** COBOL stands for Common Business Oriented Language. It was especially designed for business applications.
- **C Language:** C Language is one of the most popular programming languages among students and beginners in the field of computer programming. C Language was designed by Dennis Ritchie at AT&T Bell Labs in 1972. Sometimes C Language is called a middle level language because it provides facilities to develop application software as well as system software. So C Language combines some important qualities of a high level language and a low level programming language.

Translators

Computers only understand machine code (binary). But on the other hand, programmers prefer to use a variety of high and low-level programming languages. To get rid of this issue, the high-level and low-level program code (source code) needs to pass through a translator. A translator converts the source code into machine code (object code). Different types of translators are:

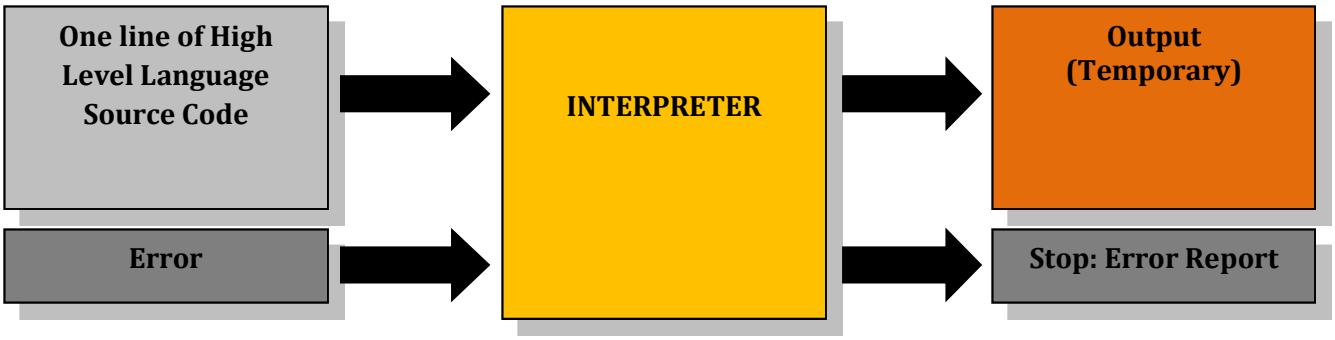
- Compiler
- Interpreter
- Assembler

Compiler

- It is a translator which is used to translate program written in a high-level language into machine code (object code). A compiler takes the whole program and translates it into corresponding machine code which can be executed thereafter.
- The process of compilation may take some time but the translated program can be used again and again without the need for recompilation.
- An error report is often produced after the full program has been translated. Errors in the program code may cause a computer to crash.
- These errors can only be fixed by changing the original source code and compiling the program again.

Interpreter

- Interpreter is a translator which is able to read, translate and execute one statement at a time from a high-level language program.
- The interpreter stops when a line of code is reached that contains an error. Interpreters are often used during the development of a program.
- They make debugging easier as each line of code is analyzed and checked before execution. Interpreted programs will launch immediately, but our program may run slower than a complied file.
- No executable file is produced.
- The program is interpreted again from scratch every time you launch it.



Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby use interpreters.	Programming language like C, C++ use compilers.

Assembler

Assemblers are used to translate a program written in a low-level assembly language into a machine code (object code) file so it can be used and executed by the computer. Once assembled, the program file can be used again and again without re-assembly.

Difference between Low-level & High-level Language

High-level Language	Low-level Language
1. High-level languages are easy to learn	1. Low-level languages are difficult to learn.
2. near to human languages.	2. far from human languages.
3. Programs in high-level languages are slow in execution.	3. Programs in low-level languages are fast in execution.
4. Programs in high-level languages are easy to modify.	4. Programs in low-level languages are difficult to modify.
5. Deep knowledge of hardware is not required to write programs.	5. Deep knowledge of hardware is required to write programs.

High-level programming languages

The high-level programming languages can be categorized into different types on the basis of the application area in which they are employed as well as the different design paradigms supported by them. The high-level programming languages are designed for use in a number of areas. Each high-level language is designed by keeping its target application area in mind. Some of the high-level languages are best suited for business domains, while others are apt in the scientific domain only. The high-level language can be categorized on the basis of the various programming paradigms approved by them. The programming paradigms refer to the approach employed by the programming language for solving the different types of problem.

Categorization based on Application

On the basis of application area the high level language can be divided into the following types:

- **Commercial languages:** These programming languages are dedicated to the commercial domain and are specially designed for solving business-related problems. These languages can be used in organization for processing handling the data related to payroll, accounts payable and tax building applications. COBOL is the best example of the commercial based high-level programming language employed in the business domain.
- **Scientific languages:** These programming languages are dedicated to the scientific domain and are specially designed for solving different scientific and mathematical problems. These languages can be

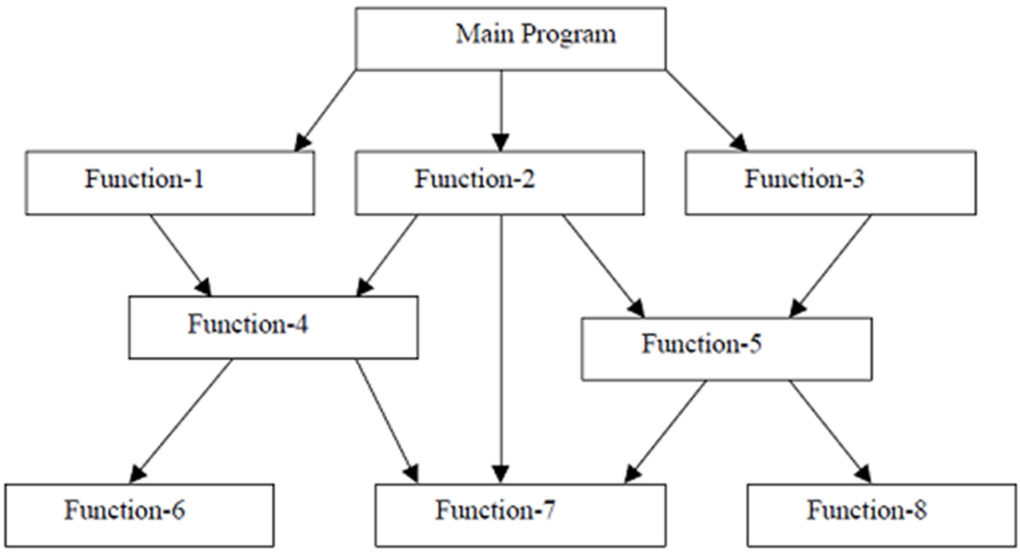
used to develop programs for performing complex calculation during scientific research. FORTRAN is the best example of scientific based language.

- **Special purpose languages:** These programming languages are specially designed for performing some dedicated functions. For example, SQL is a high-level language specially designed to interact with the database programs only. Therefore we can say that the special purpose high-level language is designed to support a particular domain area only.
- **General purpose languages:** These programming languages are used for developing different types of software application regardless of their application area. The various examples of general purpose high-level programming languages are BASIC, C, C++, and java.

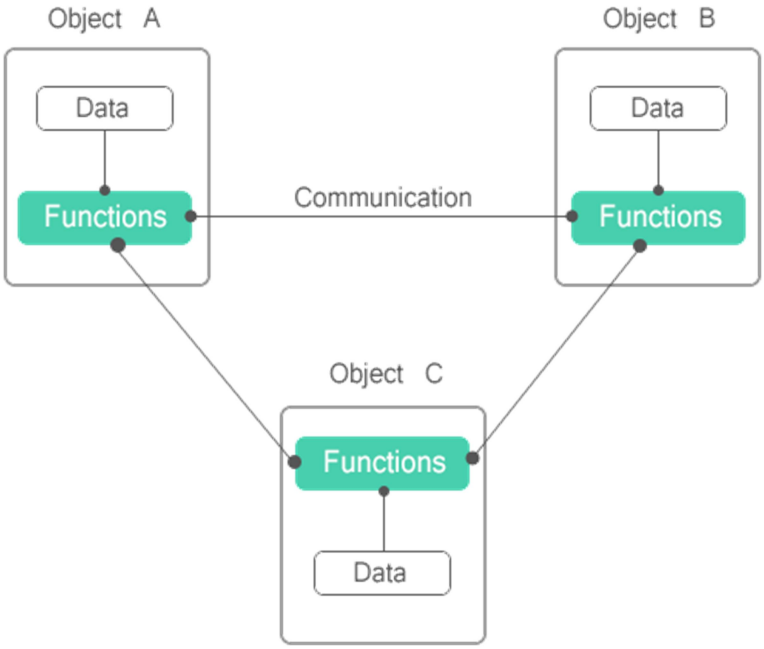
Categorization based on Design paradigm

On the basis of design paradigms the high level programming languages can be categorised into the following types:

- **Procedure or function oriented languages:** These programming languages are also called an imperative programming language. In this language, a program is written as a sequence of procedures. Each procedure contains a series of instruction for performing a specific task. Each procedure can be called by the other procedures during the program execution. In this type of programming paradigms, a code once written in the form of a procedure can be used any number of times in the program by only specifying the corresponding procedure name. Therefore the procedure-oriented language allows the data to move freely around the system. The various examples of procedure-oriented language are FORTRAN, ALGOL, C, BASIC, and ADA.



- **Logic-oriented languages:** These languages use logic programming paradigms as the design approach for solving various computational problems. In this programming paradigm, predicate logic is used to describe the nature of a problem by defining the relationship between rules and facts. *Prolog* is the best example of the logic-oriented programming language.
- **Object-oriented languages:** These languages use object-oriented programming paradigms as the design approach for solving a given problem. In this programming language, a problem is divided into a number of objects which can interact by passing messages to each other. C++, JAVA etc are the examples of object-oriented programming language.



Advantages of Object Oriented Programming

Simplicity: software objects model real world objects, so the complexity is reduced and the program structure is very clear.

Modularity: each object forms a separate entity whose internal workings are decoupled from other parts of the system.

Modifiability: it is easy to make minor changes in the data representation or the procedures in an OO program. Changes inside a class do not affect any other part of a program, since the only public interface that the external world has to a class is through the use of methods.

Extensibility: adding new features or responding to changing operating environments can be solved by introducing a few new objects and modifying some existing ones.

Maintainability: objects can be maintained separately, making locating and fixing problems easier.

Re-usability: objects can be reused in different programs.

Features of Object Oriented Programming

Various features of object oriented programming are:

- Object
- Class
- Data encapsulation
- Data hiding
- Data abstraction
- Inheritance
- Polymorphism etc.

Let us briefly discuss all these features one by one.

Object: Objects are basic entities in an object oriented system. An object is a real world entity which posses certain properties (also called attributes) and behavior (also called operation).

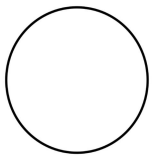
- Objects may represent a person, place, flower, a bank account, a table of data or any other item that a program has to handle.
- In a program, properties of an object are represented using variables which are also known as data (or data members) and behaviors are represented using functions which are also known as member functions.
- Objects are also known as instance of a class.

Few Examples of object are:

Student **Book** **Mango** **Rose** **Circle**







A student object has properties like name, roll no, DOB, email etc. A book object has properties like title, author name, pages price etc. Similarly a circle object has properties like radius, coordinates of center etc.

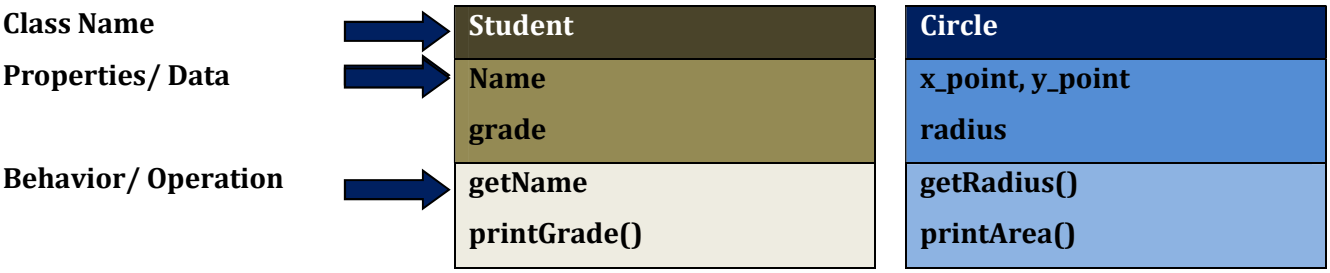
Class: Collection of similar objects which possesses common properties (also called attributes) and behavior (also called operation) is known as class. It is a **template** or **blueprint** from which objects are created.

Few examples of class:

Student **Book** **Fruit**



Data Encapsulation: The wrapping of data members and member functions into a single unit is known as data encapsulation.



Data hiding: The insulation of data and methods of a class from direct access by other classes is known as data hiding. Essentially, access privileges members of a class is assigned to other classes with the help of access specifiers (like default, private, protected & public) and OOP principles like inheritance, polymorphism etc.

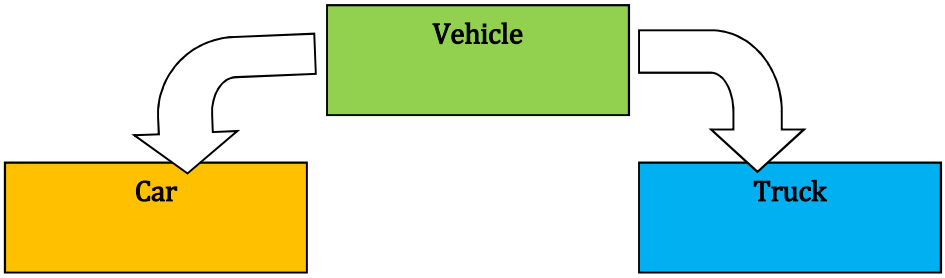
Data abstraction: The act of representing the essential features of an entity by omitting their complete details is known as data abstraction. Class supports the principle of data abstraction with the concept of abstract classes and hence class is also known as abstract data type (ADT).

Inheritance: Inheritance is a mechanism by which objet of one class acquires the properties of object of another class. The class from which properties are taken into a new class is known as parent class or **super class**. The class which takes properties from an existing class is known as child class or **sub class**. It is an important part of OOPs (Object Oriented programming system). The main advantages of inheritance are:

- Code Reusability
- Code Minimization
- Maintaining class Hierarchy
- Well Programming Construct etc

There are various types of inheritance (like single, hierarchical, multi level and hybrid etc) in java which are discussed later in this course. In java inheritance is achieved through the key word **extends**.

A simple example of hierarchical inheritance:



Here, class Car and Truck inherits the properties of the class vehicle. Class Vehicle is called super class and class Car and Truck are called sub classes.

Syntax of above example:

```
class Vehicle
{
        ----------------
        ----------------
}
class Car extends Vehicle
{
        ----------------
        ----------------
}
class Truck extends Vehicle
{
        ----------------
        ----------------
}
```

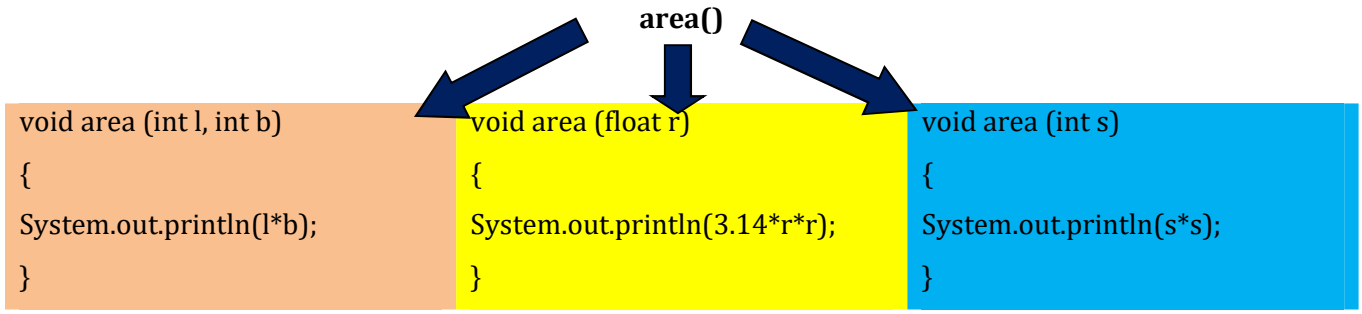
Polymorphism:

The word polymorphism means having many forms (Poly means many and morph means forms). In simple words, we can define polymorphism as the ability of an entity to exhibit multiple behaviors or definitions depending upon the working environment.

Real life example of polymorphism: A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So, the same person has different behavior in different situations.

Example of polymorphism in programming context:

Same function area can be use to find area of a rectangle, circle, square etc (this mechanism is known as *function* or *method overloading*) which is shown in the below diagram:



Review Questions of Chapter 1

MCQ Type

1. Choose the most appropriate answer:

a) The type of language understandable by a computer is

a) Assembly Language b) Machine Language
c) High level language d) JAVA language

b) A software which translates high level language to machine language by considering the program a whole is called

a) Compiler b) Interpreter
c) Either a) or b) d) Both a) and b)

c) Which one of the following provides a template or blue print for similar objects?

a) Class b) Structure
c) Inheritance d) Polymorphism

d) Which one of the following is/ are examples of low level language?

a) Assembly language b) Machine Language
c) JAVA d) Both a) and b)

e) Which one of the following is/ are OOPs characteristics?

a) Inheritance b) Data hiding
c) Encapsulation d) All of these

f) The property of OOPs by which an entity can acquire multiple forms is known as

a) Inheritance b) Polymorphism
c) Encapsulation d) Data Aggregation

g) The act of representing essential feature of a class without specifying its complete details is known as

a) Inheritance b) Polymorphism
c) Data Encapsulation d) Data Abstraction

h) The act of wrapping data and methods into a single unit is known as

a) Inheritance b) Polymorphism
c) Data Encapsulation d) Data Abstraction

i) The technique by which object of one class acquires the properties of object of some other class is known as

a) Inheritance b) Polymorphism
c) Data Encapsulation d) Data Abstraction

j) What is use of interpreter?

a) They convert byte code to machine language code
b) They read high level code and execute them

c) They are intermediated between JIT and JVM
d) It is a synonym for JIT

k) The technique by which object of one class acquires the properties of object of some other class is known as

a) Inheritance b) Polymorphism
c) Data Encapsulation d) Data Abstraction

Short Type

2. Answer briefly

a) Define translator. Differentiate between a compiler and an interpreter.
b) Differentiate between high level and low level programming language.
c) Define data encapsulation.
d) Define inheritance. List different types of inheritance.
e) What is polymorphism?
f) What is data abstraction?
g) List feature of object oriented programming system.
h) List advantages & disadvantages of high level languages.
i) List advantages & disadvantages of low level language.
j) Why C language is sometimes called middle level language?

Long Type

3. Answer in details

a) Discuss the characteristics of Object oriented Programming System.
b) Explain the advantages of Object Oriented Programming System.
c) Explain different types of programming languages with their advantages & disadvantages in details.

****

Chapter 2 INTRODUCTION TO JAVA

History of JAVA

The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of java starts with Green Team. Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was suited for internet programming. Later, Java technology was incorporated by Netscape.

James Gosling

Oak Tree

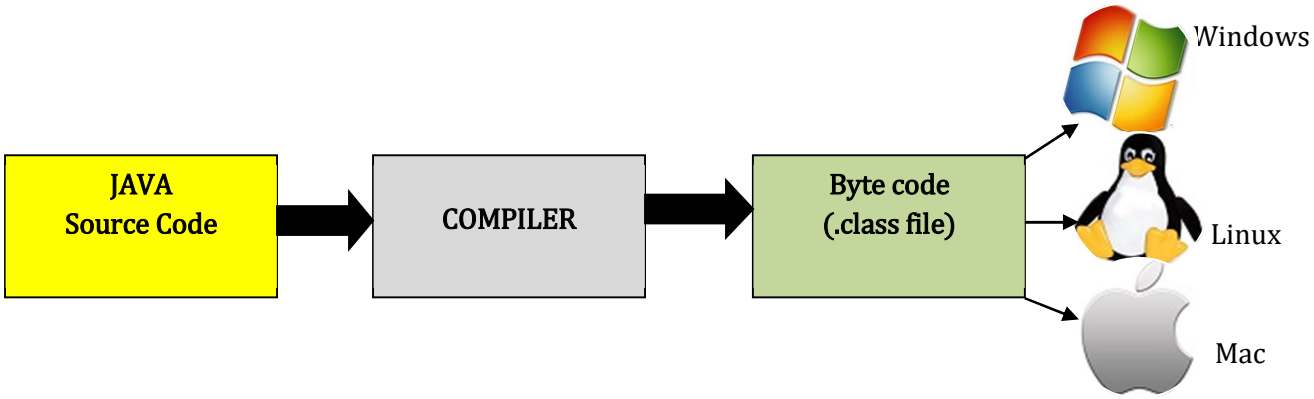
JAVA Coffee

- **James Gosling** is the founder of java.
- *James Gosling*, *Mike Sheridan*, and *Patrick Naughton* initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.
- Originally designed for small, embedded systems in electronic appliances like set-top boxes. Firstly, it was called "*Greentalk*" by James Gosling, and file extension was .gt. After that, it was called Oak and was developed as a part of the Green project.
- **Why Oak?** Oak is a symbol of strength and chosen as a national tree of many countries like U.S.A., France, Germany, Romania, etc.
- In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.
- Java is an island of Indonesia where first coffee was produced (called java coffee).
- Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.
- In 1995, Time magazine called **Java one of the Ten Best Products of 1995**. JDK 1.0 released in January 23, 1996.

JAVA Byte Code

The intermediate machine independent codes which are formed after the successful compilation of a java program are called byte code.

Byte code of a java program is always saved in the extension **.class** file. The Java byte code is not completely compiled, but rather just an intermediate code sitting in the middle because it still has to be interpreted and executed by the **JVM** installed on the specific platform such as Windows, Mac or Linux. Due to the above reason, java is called a platform independent language or write once run anywhere (**WORA**) language.



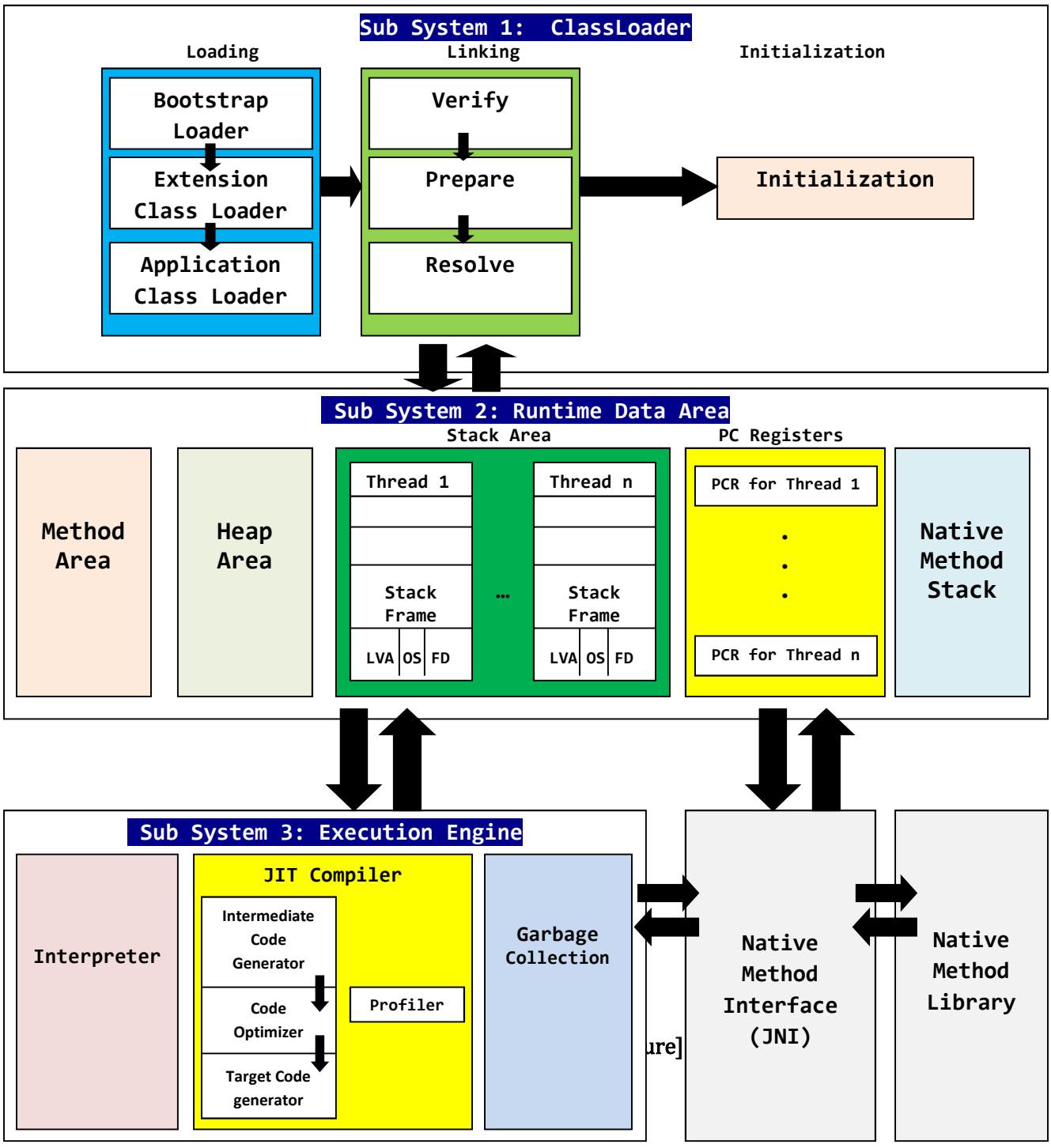
JVM- Java Virtual Machine

A Virtual Machine is a software implementation of a physical machine. Java was developed with the concept of WORA (Write Once Run Anywhere), which runs on a VM. The compiler compiles the Java file into a Java .class file, then that .class file is input into the JVM, which loads and executes the class file. Below is a diagram of the Architecture of the JVM.

JVM Architecture

As shown in the below architecture diagram, the JVM is divided into three main subsystems:

- ClassLoader
- Runtime Data Area
- Execution Engine



Subsystem 1: ClassLoader

Java's dynamic class loading functionality is handled by the ClassLoader subsystem. It loads, links and initializes the class file when it refers to a class for the first time at runtime, not compile time.

Loading

Classes will be loaded by this component. BootStrap ClassLoader, Extension ClassLoader, and Application ClassLoader are the three ClassLoaders that will help in achieving it.

- **BootStrap ClassLoader** – Responsible for loading classes from the bootstrap classpath, nothing but **rt.jar.** Highest priority will be given to this loader.
- **Extension ClassLoader** – Responsible for loading classes which are inside the ext folder **(jre\lib).**
- **Application ClassLoader** –Responsible for loading Application Level Classpath, path mentioned Environment Variable, etc.

The above ClassLoaders will follow Delegation Hierarchy Algorithm while loading the class files.

Linking

- **Verify** – Bytecode verifier will verify whether the generated bytecode is proper or not if verification fails we will get the verification error.
- **Prepare** – For all static variables memory will be allocated and assigned with default values.
- **Resolve** – All symbolic memory references are replaced with the original references from Method Area.

Initialization

This is the final phase of ClassLoading; here, all static variables will be assigned with the original values, and the static block will be executed.

Subsystem 2: Runtime Data Area

The Runtime Data Area is divided into five major components:

1. **Method Area** – All the class-level data will be stored here, including static variables. There is only one method area per JVM, and it is a shared resource.
2. **Heap Area** – All the Objects and their corresponding instance variables and arrays will be stored here. There is also one Heap Area per JVM. Since the Method and Heap areas share memory for multiple threads, the data stored is not thread-safe.
3. **Stack Area** – For every thread, a separate runtime stack will be created. For every method call, one entry will be made in the stack memory which is called Stack Frame. All local variables will be created in the stack memory. The stack area is thread-safe since it is not a shared resource. The Stack Frame is divided into three subentities:
    - **Local Variable Array** – Related to the method how many local variables are involved and the corresponding values will be stored here.

- **Operand stack** – If any intermediate operation is required to perform, operand stack acts as runtime workspace to perform the operation.
- **Frame data** – All symbols corresponding to the method is stored here. In the case of any **exception**, the catch block information will be maintained in the frame data.

4. **PC Registers** – Each thread will have separate PC Registers, to hold the address of current executing instruction once the instruction is executed the PC register will be updated with the next instruction.
5. **Native Method stacks** – Native Method Stack holds native method information. For every thread, a separate native method stack will be created.

Execution Engine

The bytecode, which is assigned to the **Runtime Data Area,** will be executed by the Execution Engine. The Execution Engine reads the bytecode and executes it piece by piece.

1. **Interpreter** – The interpreter interprets the bytecode faster but executes slowly. The disadvantage of the interpreter is that when one method is called multiple times, every time a new interpretation is required.
2. **JIT Compiler** – The JIT Compiler neutralizes the disadvantage of the interpreter. The Execution Engine will be using the help of the interpreter in converting byte code, but when it finds repeated code it uses the JIT compiler, which compiles the entire bytecode and changes it to native code. This native code will be used directly for repeated method calls, which improve the performance of the system.
   - **Intermediate Code Generator** – Produces intermediate code
   - **Code Optimizer** – Responsible for optimizing the intermediate code generated above
   - **Target Code Generator** – Responsible for Generating Machine Code or Native Code
   - **Profiler** – A special component, responsible for finding hotspots, i.e. whether the method is called multiple times or not.
3. **Garbage Collector**: Collects and removes unreferenced objects. Garbage Collection can be triggered by calling System.gc(), but the execution is not guaranteed. Garbage collection of the JVM collects the objects that are created.

Java Native Interface (JNI): JNI will be interacting with the Native Method Libraries and provides the Native Libraries required for the Execution Engine.

Native Method Libraries: This is a collection of the Native Libraries, which is required for the Execution Engine.

Java Buzz words: Features of Java

The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role

in the popularity of this language. The features of Java are also known as java *buzzwords*. A list of most important features or buzz words of Java language is:

- Simple
- Secure
- Portable
- Object-Oriented
- Robust
- Multithreaded
- Architecture neutral
- Distributed
- Dynamic

Now let us briefly explain the above buzz words.

Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because:

Java syntax is based on C++ (so easier for programmers to learn it after C++).

Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.

There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Secure

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because n**o explicit pointer is used in java &java programs run inside a virtual machine sandbox, which is insulated from virus.**

Portable

Java is portable because it facilitates you to carry the Java byte code to any platform. It doesn't require any implementation.

Object-oriented

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior. Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Robust

Robust simply means strong. Java is a robust language due to following reason:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

Multi-threaded

Java programs can execute many tasks simultaneously by defining multiple threads thereby achieving multitasking. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

Architecture Neutral

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed. In C programming, *int* data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

Distributed

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

Dynamic

Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e. C and C++. Java supports dynamic compilation and automatic memory management (garbage collection).

Comparison between C++ & Java

There are many differences and similarities between the C++ programming language and Java. A list of top differences between C++ and Java are given below:

Comparison Parameter	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
Design Goal	C++ was designed for systems and applications programming. It was an extension of C programming language.	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed with a goal of being easy to use and accessible to a broader audience.
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform	Java uses compiler and interpreter both. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces

	dependent.	output. Java is interpreted that is why it is platform independent.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for same.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comment.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Inheritance Tree	C++ creates a new inheritance tree always.	Java uses a single inheritance tree always because all classes are the child of Object class in java. The object class is the root of the inheritance tree in java.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.
Object-oriented	C++ is an object-oriented language. However, in C language, single root hierarchy is not possible.	Java is also an object-oriented language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from java.lang.Object.

Point To Remember :

- Java doesn't support default arguments like C++.
- Java does not support header files like C++. Java uses the import keyword to include different classes and methods.

Review Questions of Chapter 2

MCQ Type

1. Choose the most appropriate answer:

a) A platform is the hardware or software environment in which a program runs. Which of the following is/are java platform component(s)?

a) HTML b) Java Virtual machine
c) Hot Java d) Java API

b) Which of the following provides run time environment for java byte code to be executed?

a) JVM b) JRE
c) JDK d) JAVAC

c) Which component is used to compile, debug and execute java program?

a) JVM b) JRE
c) JDK d) JIT

d) Which component is responsible for converting byte code into machine specific code?

a) JVM b) JRE
c) JDK d) JIT

e) Which component is responsible to run java program?

a) JVM b) JRE
c) JDK d) JIT

f) Which component is responsible to optimize byte code to machine code?

a) JVM b) JRE
c) JDK d) JIT

g) Which statement is true about java?

a) Platform dependent b) Platform independent
c) Code dependent d) Sequence dependent

h) What is the extension of java code files?

a) .java b) .class
c) .binary d) .obj

i) What is the extension of java byte code files?

a) .java b) .class
c) .binary d) .obj

j) Java language is

a) Robust b) Purely object oriented
c) Portable d) All of these

Short Type

2. Answer briefly

a) Define byte code.
b) Define JVM.
c) Why java is called a portable language?
d) Why java is called a robust language?
e) Why java is called a dynamic language?
f) Why java is architecture neutral?
g) What is JDK, JIT & JRE?
h) Define multithreading.
i) Who is the father of JAVA?
j) Why java is called a platform independent language?
k) Why java is called write once run anywhere language (WORA)?
l) Why java programs are secure over internet?

Long Type

3. Answer in details

a) Discuss the features of java which makes it a powerful language.
b) What is JVM? Explain the architecture of JVM with the help of a neat diagram.
c) Differentiate between C++ and JAVA in details.

****

Chapter 3 **BASIC ELEMENTS OF JAVA PROGRAMMING**

First JAVA Program

Let us write our first JAVA program which will display the following message: Welcome to the JAVA world.

Program 3.1: Write a program to print a message "Welcome to the JAVA world"

Solution:

```
// First.java
class First
{
      public static void main (String args[])
      {
      System.out.println("Welcome to the JAVA world ");
      }
}
```

Output:

Welcome to the JAVA world

How to compile & execute the above program

Compilation:

To compile the above program, use the following syntax on the command line:

javac *java_filename*.java

Here we will use the following command to compile our program:

javac First.java

If successfully compiled, the compiler creates a file called First.class that contains byte code version of our java program.

Execution:

Now to run a java program, use the following syntax:

java *java_filename*

Here we will use following command to run the program:

java First

Explanation of the above program

class First

In a purely object oriented programming system, all our codes remain inside a class definition. Therefore, our program begins with a class definition. Here class is a keyword used to define a class & First is the name of our class.

{

Here this opening brace indicates the beginning of the class definition block.

public static void main(String args[])

All Java applications begin execution from main ( ) function. (This is just like C/C++.)

- The **public** keyword is an access specifier, which allows the programmer to control the visibility of class members. When a class member is preceded by public, then that member may be accessed by code outside the class in which it is declared.
- The keyword **static** allows main ( ) to be called without having to instantiate a particular instance of the class.
- The keyword **void** simply tells the compiler that main ( ) does not return a value. As stated, main ( ) is the method called when a Java application begins. Keep in mind that Java is case-sensitive. Thus, Main is different from main.
- **String** args**[ ]** declares a parameter named args, which is an array of instances of the class String. This args parameter is essential when we enter some input through the command line while executing a program.

{

Here this opening brace indicates the beginning of main method definition block.

System.out.println("Welcome to the JAVA world ");

This line outputs the string “Welcome to the JAVA world” followed by a new line on the screen.

Output is actually accomplished by the built-in println() method. In this case, println( ) displays the string which is passed to it.

}

Here this closing brace indicates the end of main method definition block.

}

Here this closing brace indicates the end of the class definition block.

Data Types

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

- **Primitive data types:** The primitive data types include *boolean, char, byte, short, int, long, float* and *double*.
- **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Java Primitive Data Types

Category	Data Type	Default Value	Default size
Boolean	boolean	false	1 bit
Character	char	'\u0000'	2 byte
Integer	byte	0	1 byte
	short	0	2 byte
	int	0	4 byte
	long	0L	8 byte
Float/ Real	float	0.0f	4 byte
	double	0.0d	8 byte

Note: In general, if n bits are required for a data type, then its range is -2^{n-1} to $2^{n-1}-1$.

Solved Question 3.2 Why char uses 2 byte in java unlike C/C++ which uses 1 byte of memory only and what is \u0000?

Solution: It is because java uses Unicode system, not ASCII (American Standard Code for Information Interchange) code system. Unicode system represents a character set which can be used to represent all languages available world. \u0000 is the lowest range of Unicode system.

JAVA Tokens

The smallest unit of code appearing in a program which is recognized by the compiler is called as **token**. A token in a java program are of following types:

- Identifiers
- Literals/ Constant
- Keywords
- Operators
- Comment
- Special characters

Let us discuss all the tokens in details.

Identifiers

These are names assigned to **variables, methods**, **classes**, **packages** & **interfaces**. Following are the rules to construct an identifier:

- An identifier contains alphabets ($A-Z, a-z$), digits ($0-9$), underscore (_) or dollar ($) only.
- The first character in an identifier must be an alphabet or underscore (_) or dollar ($).
- Blank spaces are not allowed within an identifier.
- Keywords can't be identifiers.
- Identifier name are case sensitive.
- There is no limit on the length of a Java identifier.

Examples of some valid identifiers are $\boldsymbol{x}, \boldsymbol{a1}, \$\boldsymbol{a}, \boldsymbol{India}, \$\boldsymbol{myVariable}$ etc.

Literals or constant

Literals in Java are a sequence of characters (digits, letters, and other characters) that represent constant values to be stored in variables. Java offers following types of literals:

- **Integer literals:** These are numeric values without a decimal point. Examples of integer literals are $1, -34, 1098$ etc.
- **Floating literals:** These are numeric values with a decimal point. Examples of floating literals are $1.99, -34.07, 2000.99$ etc.
- **Character literals:** Any single character enclosed in a single quote (' ') is termed as a character literal. Examples of character literals are 'A', ' + ', ' ', '.' etc.
- **String literals:** One or more character enclosed in a double quote (" ") is termed as a string literal. Examples of string literals are "*Java Programming*", "5", "*Hello Java*" etc
- **Boolean literals:** These are literals which acquire Boolean values *true* or *false*.
- **Null literals:** The *null* in Java is a literal of the null type. It cannot be cast to a primitive type such as int. float etc, but can be cast to a reference type. Also, null does not have the value 0 necessarily.

Keywords

Keywords or Reserved words are the words that represent some predefined actions. These words are therefore not allowed to use as a variable names or objects. There are **49** keywords in java which are listed below:

abstract	Java abstract keyword is used to declare abstract class. Abstract class can provide the implementation of interface. It can have abstract and non-abstract methods.
boolean	Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.

break	Java break keyword is used to break loop or switch statement. It breaks the current flow of the program at specified condition.
byte	Java byte keyword is used to declare a variable that can hold an 8-bit data values.
case	Java case keyword is used to with the switch statements to mark blocks of text.
catch	Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.
char	Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters
class	Java class keyword is used to declare a class.
continue	Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.
default	Java default keyword is used to specify the default block of code in a switch statement.
do	Java do keyword is used in control statement to declare a loop. It can iterate a part of the program several times.
double	Java double keyword is used to declare a variable that can hold a 64-bit floating-point numbers.
else	Java else keyword is used to indicate the alternative branches in an if statement.
enum	Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.
extends	Java extends keyword is used to indicate that a class is derived from another class or interface.
final	Java final keyword is used to indicate that a variable holds a constant value. It is applied with a variable. It is used to restrict the user.
finally	Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether exception is handled or not.
float	Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.
for	Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some conditions become true. If the number of iteration is fixed, it is recommended to use for loop.
if	Java if keyword tests the condition. It executes the if block if condition is true.
implements	Java implements keyword is used to implement an interface.
import	Java import keyword makes classes and interfaces available and accessible to the current source code.

instanceof	Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.
int	Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
interface	Java interface keyword is used to declare an interface. It can have only abstract methods.
long	Java long keyword is used to declare a variable that can hold a 64-bit integer.
Native	Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
new	Java new keyword is used to create new objects.
null	Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.
package	Java package keyword is used to declare a Java package that includes the classes.
private	Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.
protected	Java protected keyword is an access modifier. It can be accessible within package and outside the package but through inheritance only. It can't be applied on the class.
public	Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.
return	Java return keyword is used to return from a method when its execution is complete.
short	Java short keyword is used to declare a variable that can hold a 16-bit integer.
static	Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is used for memory management mainly.
strictfp	Java strictfp is used to restrict the floating-point calculations to ensure portability.
super	Java super keyword is a reference variable that is used to refer parent class object. It can be used to invoke immediate parent class method.
switch	The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.
synchronized	Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.
this	Java this keyword can be used to refer the current object in a method or constructor.
throw	The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exception. It is followed by an instance.
throws	The Java throws keyword is used to declare an exception. Checked exception can be propagated with throws.

transient	Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.
try	Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.
void	Java void keyword is used to specify that a method does not have a return value.
volatile	Java volatile keyword is used to indicate that a variable may change asynchronously.
while	Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

Operators in Java

Operator in java is a symbol that is used to perform operations on one or more operands. Depending upon number of operands, operators are classified as shown in the below table.

Type of operator	Number of operands	Example
Unary	1	$-, ++, --$ etc
Binary	2	$+, -, *, \%,$ / etc
Ternary	3	?:

Various types of operators in java are

- Arithmetic Operator
- Relational Operator
- Logical Operator
- Increment & decrement Operator
- Assignment Operator
- Ternary Operator
- Bitwise and Shift Operator

.

Arithmetic Operator

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists various arithmetic operators:

Arithmetic operator	Yields	Example
Addition (+)	addition result	$2 + 9 = 11$
Subtraction (−)	difference result	$23 - 7 = 16$
Multiplication (*)	product result	$23 * 7 = 161$

Division (/)	division result	$23/7 = 3, 6/4 = 1.5$ etc
Modular Division (%)	remainder result	$23\%6 = 5,\ \ 6\%10 = 6$ etc

Relational Operator

These operators are binary operators that take two operands, whose values are being compared. Comparison operators are used in conditional statements, especially in loops, where the result of the comparison decides whether execution should proceed or not. They form the key to program flow control, known as conditional processing.

The following table lists various relational operators:

Relational operator	Meaning	Example
>	Strictly greater than	2>5 yields false
>=	Greater than or equal to	5>=4 yields true
<	Strictly less than	5<100 yields true
<=	Smaller than or equal to	5<=5 yields true
==	Double equal to	6==7 yields false
!=	Not equal to	6!=7 yields true

Logical operator

Logical operators are used when we want to check multiple conditions together. The logical operators are given below:

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

Logical AND Operator

We can combine many relational expressions using "Logical And" operators .The result will be a boolean type. This operator is represented by the symbol "&&".

Consider the operation "operand1 && operand2"

Here operand1 and operand2 can be relational expressions or boolean types. The result of the operation "operand1 && operand2" will be

- "true" only if both operand1 and operand2 are "true"

- "false" if any of the operands (operand1 or operand2) is "false" or both the operands (operand1 or operand2) are "false".

[Truth Table of logical AND (&&) operator]

A	B	A&&B
true	true	true
true	false	false
false	true	false
false	false	false

Logical OR Operator

We can combine many relational expressions using "Logical OR" operators .The result will be a boolean type. This operator is represented by the symbol "||".

Consider the operation "operand1 || operand2"

Here operand1 and operand2 can be relational expressions or boolean types. The result of the operation "operand1 || operand2" will be

- "true" if any of the operands (operand1 or operand2) is "true" or both the operands (operand1 or operand2) are "true".
- "false" only if both operand1 and operand2 are "false"

[Truth Table of OR (||)]

A	B	A||B
true	true	true
true	false	true
false	true	true
false	false	false

Logical Not Operator

Logical Operator can be used with a boolean type variable or with a relational expression or with a logical expression. Logical not operator is denoted by the symbol "!".

- Assume the original value is "true". If we use this operator, the value will be changed to "false".
- Assume the original value is "false". If we use this operator, the value will be changed to "true".

[Truth Table of logical NOT (!) operator]

A	!A
true	true
true	false

Increment and Decrement Operator

- It is one of the variations of "Arithmetic Operator".
- Increment and decrement operators are unary operators, which operates on one operand.
- Increment operator is used to increment value stored inside variable on which it is operating.
- Decrement Operator is used to decrement value of Variable by 1 (default).

Meaning

++ (Increment operator) increments the value by 1

− − (Decrement operator) decrements the value by 1

Pre-Increment Operator

- "++" is written before variable name. Value is incremented first and then incremented value is used in expression.
- "++" cannot be used over "Constant" of "final Variable".

Post increment: Increment value of a variable after assigning.

Pre Decrement

- "− −" is written before variable name. Value is decremented first and then decremented value is used in expression.
- "− −" cannot be used over "Constant" of "final Variable".

Post decrement: Decrement value of a variable after assigning.

Assignment Operator

Assignment operators are used in Java to assign values to variables.

For example:

int age;

age = 5;

The assignment operator assigns the value on its right to the variable on its left. Here, 5 is assigned to the variable age using = operator.

Shorthand assignment operator: Operators like +=, -=, *=, /= etc are known as shorthand assignment operator.

x+=1 is equivalent to x=x+1

y/=9 is equivalent to y=y/9

w*=2 is equivalent to w=w*2

p%=7 is equivalent to p=p%2 etc

Conditional operator

The conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide; which value should be assigned to the variable. The operator is written as:

variable x = (expression)? value if true: value if false

Program 3.3: Write a program to demonstrate the use of conditional operator

Solution:

```
// Test.java
public class Test
{
	public static void main(String args[])
	{
		int a, b;
		a = 10;
		b = (a == 1) ? 20: 30;
		System.out.println("Value of b is: " + b);
		b = (a == 10) ? 20: 30;
		System.out.println("Value of b is: " + b);
	}
}
```

Output:

Value of b is: 30

Value of b is: 20

Bitwise and shift operator

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

Bitwise operator works on bits and performs the bit-by-bit operation. Assume if a = 60 and b = 13; now in binary format they will be as follows –

a = 0011 1100

b = 0000 1101

---

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

The following table lists the bitwise operators −

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

Comments in JAVA

The java comments are statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code for specific time.

In Java there are three types of comments:

- Single line comments
- Multi line comments

- Documentation comments

Single line comments

A beginner level programmer uses mostly single-line comments for describing the code functionality. It is the easiest type of comments.

Multi line comments

To describe a full method in a code or a complex snippet single line comments can be tedious to write, since we have to give '//' at every line. So to overcome this multi line comments can be used.

Documentation comments

Documentation comments starts with a forward slash followed by two asterisks, and ends with an asterisk followed by a forward slash. This comment contains descriptions about the code and is used to create API document. This comment is usually added in the beginning of the java file explaining why this program is designed and how to use it.

Typecasting

The method of converting one primitive data type into another data type is called type casting. There are two types of typecasting: implicit type casting and explicit type casting.

Implicit type casting or widening casting

This is the process of converting a smaller type to a larger type.

$byte \rightarrow short \rightarrow char \rightarrow int \rightarrow long \rightarrow float \rightarrow double$

This is done automatically by the compiler. Hence it is also known as implicit type casting. The implicit type casting is demonstrated below.

Program 3.4: Write a program to demonstrate the use of implicit type casting.

Solution:

```
// ImplicitTC.java
public class ImplicitTC
{
      public static void main(String[] args)
      {
          int myInt = 9;
          double myDouble = myInt;        // Automatic casting: int to double
          System.out.println(myInt);      // Outputs 9
          System.out.println(myDouble);   // Outputs 9.0
      }
}
```

Output:

9

9.0

Explicit type casting or narrowing casting

This type casting is done manually by placing the type in parentheses in front of the value. This is done manually by the programmer. Hence it is also known as explicit type casting. The explicit type casting is demonstrated below.

Program 3.5: Write a program to demonstrate the use of explicit type casting.

Solution:

```
// ExplicitTC.java
public class ExplicitTC
{
      public static void main(String[] args)
      {
          double myDouble = 9.78;
          int myInt = (int) myDouble;     // Manual casting:double to int
          System.out.println(myDouble);   // Outputs 9.78
          System.out.println(myInt);      // Outputs 9
      }
}
```

Output:

9.78

9

Java's Control Statements

The JAVA control statements inside a program are usually executed sequentially. Sometimes a programmer wants to break the normal flow and jump to another statement or execute a set of statements repeatedly. The statements which break the normal sequential flow of the program are called control statements. Java's program control statements can be put into the following categories: *selection or decision making statement, iteration or loop statement, and jump*.

- **A Selection or decision making statement** allows program to choose different paths of execution based upon the truth value of an expression.

- **An iteration or loop statement** allows program execution to repeat one or more statements many number of time based upon some condition.
- **Jump statements** allows program to jump out of a loop before the normal exit of loop body.

Selection or decision making statements

Different types of selection or decision statements in java are:

- if statement
- if else statement
- if else if statement

Let us discuss all of these one by one.

if Statement

This statement is used when the decision is made taking only one condition into consideration.

Syntax:

```
if (condition)
{
        //These codes are executed when condition is true
}
```

Program 3.6: Write a program to input an integer from keyboard and display it if it is positive.

Solution:

```
// DispPositive.java
import java.util.*;
class DispPositive
{
   public static void main(String args[])
   {
      int n;
      System.out.println("Enter an integer:");
      Scanner input = new Scanner(System.in);
      n = input.nextInt();
      if(n>0)
      {
      System.out.println("The integer = " + n);
```

```
        }
    }
}
```

Output:

Enter an integer:

7

The integer = 7

if else statement

This statement is used when the decision is made taking exactly two conditions into consideration.

Syntax:

```
if (condition)
{
        //These codes are executed when condition is true
}
else
{
        //These codes are executed when condition is false
}
```

Program 3.7: Write a program to input an integer from keyboard and check whether it is even or odd.

Solution:

```
// IsEvenOdd.java
import java.util.*;
class IsEvenOdd
{
   public static void main(String args[])
   {
      int n;
      System.out.println("Enter an integer:");
      Scanner input = new Scanner(System.in);
      n = input.nextInt();
```

```
        if(n%2==0)
        {
        System.out.println(n+" is an even integer");
        }
        else
        {
        System.out.println(n+" is an odd integer");
        }
    }
}
```

Output:

Enter an integer:

7

7 is an odd integer

if-else-if statement

This statement is used when the decision is made taking more than two conditions into consideration.

Syntax:

```
if (condition1)
{
        //These codes are executed when condition1 is true
}
else if (condition2)
{
        //These codes are executed when condition2 is true
}
……..
……..
else
{
        //These codes are executed when all conditions are false
}
```

Program 3.8: **Write a program to calculate division obtained by a student with his average mark entered through the keyboard as per following rules:**

percentage mark $\geq$ 60, 1st division

50 $\leq$ percentage mark $<$ 60, 2nd division

40 $\leq$ percentage mark $<$ 50, 3rd division

20 $\leq$ percentage mark $<$ 50, Fail

Solution:

```
//FindDivision.java
import java.util.*;
class FindDivision
{
	public static void main(String args[])
	{
		int avg_mark;
		System.out.println("Enter average mark:");
		Scanner input = new Scanner(System.in);
		avg_mark = input.nextInt();
		if(avg_mark >=60)
		{
		System.out.println("1st Division");
		}
		else if(avg_mark >=50 && avg_mark <60)
		{
		System.out.println("2nd Division");
		}
		else if(avg_mark >=40 && avg_mark <50)
		{
		}
		else
		{
		System.out.println("3rd  Division");
		}
	}
}
```

Output:

Enter average mark:

75

1st Division

Switch case statements

The switch statement is Java's multi way branch statement. It provides an easy way to dispatch execution to different parts of your code based on the value of an expression. As such, it often provides a better alternative to a large series of if-else-if statements.

Syntax:

```
switch (expression)
 {
        case value1:
        // These codes are executed when value of expression is value1
        break;
        case value2:
        // These codes are executed when value of expression is value2
        break;
        ……..
        ……..
        case valueN:
        // These codes are executed when value of expression is valueN
        break;
        default:
        // These codes are executed when value of expression is other than above case values
}
```

Program 3.9: Write a program to display a month name, if its corresponding integer value is entered through the keyboard. Integer 1 corresponds to January, 2 corresponds to February and so on.

Solution:

```
// FindYear.java
import java.util.*;
```

```
class FindYear
{
	public static void main(String args[])
	{
		int m;
		System.out.println("Enter an integer:");
		Scanner input = new Scanner(System.in);
		m = input.nextInt();
		switch(m)
		{
		case 1:
		System.out.println("January");
		break;
		case 2:
		System.out.println("February");
		break;
		case 3:
		System.out.println("March");
		break;
		case 4:
		System.out.println("April");
		break;
		case 5:
		System.out.println("May");
		break;
		case 6:
		System.out.println("June");
		break;
		case 7:
		System.out.println("July");
		break;
		case 8:
		System.out.println("August");
		break;
```

```
            case 9:
            System.out.println("September");
            break;
            case 10:
            System.out.println("October");
            break;
            case 11:
            System.out.println("November");
            break;
            case 12:
            System.out.println("December");
            break;
            default:
            System.out.println("Invalid month");
            }
      }
}
```

Output:

Enter an integer:

9

September

Iteration or loop statements

Java's iteration statements are for, while, and do-while. These statements create what we commonly call loops. As you probably know, a loop repeatedly executes the same set of instructions until a termination condition is met.

while loop

The while loop is Java's most fundamental looping statement. It repeats a statement or block while its controlling expression is true.

The condition can be any Boolean expression. The body of the loop will be executed as long as the conditional expression is true. When condition becomes false, control passes to the next line of code immediately following the loop.

Note: The curly braces are unnecessary if only a single statement is being repeated.

Syntax of while loop:

```
initialization;
while (condition)
{
        // these codes are executed in each iteration of the loop
        //increment or decrement statement which advances the loop;
}
```

Program 3.10: Write a program to display the square and cube of all integers from 1 to n, for any positive integer.

Solution:

```
//SquareCube1toN.java
import java.util.*;
class SquareCube1toN
{
      public static void main(String args[])
      {
            int n, i=1, s, c;
            System.out.println("Enter a positive integer:");
            Scanner input = new Scanner(System.in);
            n = input.nextInt();
            System.out.println("Integer\t\tSquare\t\tCube");
            while(i<=n)
            {
            s=i*i;
            c=i*i*i;
            System.out.println(i+"\t\t"+s+"\t\t"+c);
            i++;
            }
      }
}
```

```
Output:
Enter a positive integer:
10
Integer     Square      Cube
1           1           1
2           4           8
3           9           27
4           16          64
5           25          125
6           36          216
7           49          343
8           64          512
9           81          729
10          100         1000
```

do while Loop

If the conditional expression controlling a while loop is initially false, then the body of the loop will not be executed at all. However, sometimes it is desirable to execute the body of a while loop at least once, even if the conditional expression is false to begin with. The do-while loop always executes its body at least once, because its conditional expression is at the bottom of the loop.

Syntax of do while loop:

```
Initialization;
do
{
        // body of loop
        Increment/ decrement;
} while (condition);
```

Program 3.11: Write a program to find factorial of a positive integer given from the keyboard.

Solution:

```
//Factorial.java
```

```
import java.util.*;
class Factorial
{
    public static void main(String[] args)
    {
            int n, i=1;
            long fact=1;
            System.out.println("Enter a positive integer:");
            Scanner input = new Scanner(System.in);
            n = input.nextInt();
           do
            {
                  fact = fact * i;
                  i++;
            } while (i <= n);
            System.out.println("Factorial of " + n + " is: " + fact);
    }
}
```

Output:

Enter a positive integer:

7

Factorial of 7 is 5040

for loop

This loop is the most popular loop.

Syntax of for loop:

```
for(initialization; condition; iteration)
{
        // these codes are executed in each iteration of the loop
}
```

The for loop operates as follows:

- When the loop first starts, the initialization portion of the loop is executed. Generally, this is an expression that sets the value of the loop control variable, which acts as a counter that controls the loop. It is important to understand that the initialization expression is only executed once.
- Next, condition is evaluated. This must be a Boolean expression. It usually tests the loop control variable against a target value. If this expression is true, then the body of the loop is executed. If it is false, the loop terminates.
- Next, the iteration portion of the loop is executed. This is usually an expression that increments or decrements the loop control variable. The loop then iterates, first evaluating the conditional expression, then executing the body of the loop, and then executing the iteration expression with each pass. This process repeats until the controlling expression is false.

Program 3.12: Write a program to print all even integers from 1 to n, for any positive integer n input through the keyboard.

Solution:

```
//Integers1toN.java
import java.util.*;
class Integers1toN
{
	public static void main(String args[])
	{
		int n, i;
		System.out.println("Enter a positive integer:");
		Scanner input = new Scanner(System.in);
		n = input.nextInt();
		System.out.println("Even integers from 1 to "+n+" are:");
		for(i=1; i<=n; i++)
		{
			if(i%2==0)
			{
			System.out.print(i+"\t");
			}
		}
	}
}
```

Output:

Enter a positive integer:

25

Even integers from 1 to 25 are:

2 4 6 8 10 12 14 16 18 20 22 24

Jump Statements

Java offers following jump statements:

break statement: A break statement takes the program control out of the loop. When break statement is used, program control immediately jump to the statement immediately follows the loop statement.

continue statement: A continue statement takes control to the beginning of the loop.

Note: goto statement is not supported by java.

Program 3.13: Write a program to check whether an integer entered through keyboard is prime or not.

Solution:

```
//checkPrime.java
import java.util.*;
class checkPrime
{
	public static void main(String args[])
	{
		int n, i;
		System.out.println("Enter an integer:");
		Scanner input = new Scanner(System.in);
		n = input.nextInt();
		for(i=2; i<n; i++)
		{
			if(n%i==0)
			{
			System.out.println(n+" is not Prime");
			break;
			}
		}
		if(i==n)
```

```
            System.out.println(n+" is  Prime");
      }
}
```

Output:

Enter an integer:

11

11 is Prime

Taking input from keyboard

Taking input from keyboard is one of the most essential parts of a programming language. In java, input can be taken from the keyboard using either of these following methods:

a) Command line arguments
b) Scanner class
c) BufferedReader class

Command line argument method

The command line arguments are the arguments that we pass during the execution of a program. Accessing command line arguments inside a java program is quite easy. They are stored in the String array (named args[] in our examples. 1st argument is stored in args[0], 2nd argument is stored in args[1] and so on) which is passed as a parameter to the main() method. It is clearly explained in the following example.

Program 3.14: Write a program to find addition of two integers, where the inputs are given as command line arguments.

Solution:

```
//CommandLine.java
class CommandLine
{
      public static void main(String args[])
      {
            int x, y, res;
            x=Integer.parseInt(args[0]);
            y=Integer.parseInt(args[1]);
            res=x+y;
            System.out.println("Required addition result is "+res);
```

```
	}
}
```

```
Output:
E:\JAVA>java CommandLine 65  111
Required addition result is 176
```

Explanation:

The inputs given during program execution get stored as string value in args[0] & args[1] respectively. Here Integer.parseInt() method is used to convert string to its corresponding integer.

Scanner class method

Scanner is a class in java.util package used for obtaining the input of the primitive types like int, double, etc. and strings. It is the easiest way to read input in a Java program, though not very efficient if we want an input method for scenarios where time is a constraint like in competitive programming.

- To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream. We may pass an object of class File if we want to read input from a file.
- To read numerical values of a certain data type xyz, the function to use is nextXyz(). For example, to read a value of type short, we can use **nextShort()**. To read an integer value, we use **nextInt()**. To read a long value we use **nextShort()**. To read a double value, we use **nextDouble()** and so on.
- To read strings, we use **nextLine().**
- To read a single character, we use next().charAt(0). next() function returns the next token/word in the input as a string and charAt(0) function returns the first character in that string.

Program 3.15: Write a program to find addition of two integers, where the inputs are given from the keyboard.

Solution:

```
//InputScanner.java
import java.util.*;
class InputScanner
{
	public static void main(String args[])
	{
		int x, y, res;
		System.out.println("Enter two integers:");
```

```
            Scanner input = new Scanner(System.in);
            x = input.nextInt();
            y = input.nextInt();
            res=x+y;
            System.out.println("Required addition result is "+res);
      }
}
```

```
Output:
Enter two integers:
67   89
Required addition result is 156
```

BufferedReader class method

It is the most efficient method of reading inputs from the keyboard. This is the Java classical method to take input, Introduced in JDK1.0. This method is used by wrapping the **System.in** (standard input stream) in an **InputStreamReader** which is wrapped in a **BufferedReader**, we can read input from the user in the command line. The main demerit is that wrapping code is hard to remember and we have to import built in package **java.io**.

Program 3.16: Write a program to find addition of two numbers given from the keyboard, using BufferedReader class method.

Solution:

```
import java.io.*;
class BufferedReaderAddition
{
      public static void main(String args[])
      {
            BufferedReader br = new BufferedReader(new
            InputStreamReader(System.in));
            String str;
            int x, y, z;
            try
            {
                  System.out.println("Enter 1st number:");
```

```
                str = br.readLine();
                x=Integer.parseInt(str);
                System.out.println("Enter 2nd number:");
                str = br.readLine();
                y=Integer.parseInt(str);
                z=x+y;
                System.out.print(“Addition Result is: ”+z);
            }
            catch (Exception e)
            {
                System.out.println("Invalid Number ..Try Again!!!!!!!");
            }
        }
}
```

Output:

Enter 1st number:

45

Enter 2nd number:

89

Addition Result is: 134

Practice Questions

1. Choose the most appropriate answer:

a) JAVA characters take _____memory.

a) 2 Bytes b) 1 Byte
c) 3 Bytes d) 4 Bytes

b) Which of the following is a class in java?

a) int b) float
c) char d) String

c) In java, which of the following data type groups are of same size?

a) int and long b) long and float
c) float and double d) int and float

d) Why an array is called "*a homogeneous collection of data*"?

a) It stores different types of data
b) Size of an array is limited
c) An array can store only one type of data
d) An array uses indices for addressing an item stored in it

e) Find the output of below code:

```
class Test
{
        public static void main(String[] args)
        {
        for(int i = 0; true; i++)
        {
        System.out.println("Hello");
        break;
        }
        }
}
```

a) Hello b) Hello infinite number of times
c) Compilation Error d) None of these

f) Find the output of below code:

```
class Test
{
        public static void main(String[] args)
        {
        for(int i = 0; 1; i++)
        {
        System.out.println("Hello");
        break;
        }
        }
}
```

a) Hello b) Hello infinite times
c) Compilation Error d) None of these

g) Which of these can be returned by the operator &?
a) Integer b) Boolean
c) Character d) Integer or Boolean

h) Which of these is long data type literal?
a) 0x99fffL b) ABCDEFG
c) 0x99fffa d) 99671246

i) Which of these operators is used to allocate memory to array variable in Java?
a) malloc b) alloc
c) new d) new malloc

j) Which of these is an incorrect array declaration?
a) int arr[] = new int[5] b) int [] arr = new int[5].
c) int arr[] = new int[5] d) int arr[] = int [5] new

k) What will this code print?
```
int arr[] = new int [5];
System.out.print(arr);
```
a) 0 b) value stored in arr[0].
c) 00000 d) Class name@ hashcode in hexadecimal form

l) Which of these is necessary to specify at time of array initialization?
a) Row b) Column
c) Both Row and Column d) None of the mentioned

m) Which one is a valid declaration of a boolean?
a) boolean b1 = 1; b) boolean b2 = 'false';
c) boolean b3 = false; d) boolean b4 = 'true'

Short Type

2. Answer briefly

a) Why a java character takes 2 bytes of memory unlike C or C++?
b) Write the syntax to accept an integer input using scanner class.
c) What are different methods to accept input in java?
d) Name the data structure used to maintain recursive function calls in a program.
e) Define an array. How memory is allocated to an array dynamically?
f) Differentiate between = and ==.
g) Define conditional operator in java. Give its syntax.
h) What are different comment lines in java?
i) Define literals in java.
j) What are bit wise operators?

Long Type

3. Answer in details

a) Define tokens. Explain various types of tokens available in java in details.
b) Explain various loop control statements in java with examples.
c) Explain various selection control statement in java with examples.
d) Define operator. Explain various types of operator with examples.

Programming Exercise

4.

a) Write a program to find sum of first n natural numbers, where value of integer is supplied from the keyboard.
b) Write a program to find sum of digits of an integer given through the keyboard.
c) Write a program to print first n terms of a Fibonacci sequence using a recursive function. Assume first two terms of the Fibonacci sequence are 0 and 1.
d) Write a program to find GCD of two integers using recursive function. Your program should able to pass the following test cases:

Input		Output
5	7	1
20	0	20
0	7	7
9	6	3
5	h	Invalid input

e) Write a program to print sum of even factors of an integer.
f) Write a program to check whether an integer is prime number or not.
g) Write a program to check whether integer is Armstrong or not.
h) Write a program to print the sum of all diagonal elements of a square matrix containing numeric elements.
i) Write a program to multiply two n × n matrices.
j) Write a program to find maximum value from a list containing n numeric values.

****