GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)

Seshadri Rao Knowledge Village, Gudlavalleru – 521 356.

Department of Computer Science and Engineering

HANDOUT

on

OBJECT ORIENTED PROGRAMMING THROUGH JAVA

Vision

To be a Centre of Excellence in computer science and engineering education and training to meet the challenging needs of the industry and society.

Mission

- To impart quality education through well-designed curriculum in tune with the growing software needs of the industry.
- To serve our students by inculcating in them problem solving, leadership, teamwork skills and the value of commitment to quality, ethical behavior & respect for others.
- To foster industry-academia relationship for mutual benefit and growth.

Program Educational Objectives

PEO1 : Identify, analyze, formulate and solve Computer Science and Engineering problems both independently and in a team environment by using the appropriate modern tools.

PEO2 : Manage software projects with significant technical, legal, ethical, social, environmental and economic considerations.

PEO3 : Demonstrate commitment and progress in lifelong learning, professional development, leadership and Communicate effectively with professional clients and the public.

HANDOUT ON OBJECT ORITENTED PROGRAMMING THROUGH JAVA

Class & Sem. :II B.Tech – I Semester Year: 2019-20

Branch : CSE Credits: 3

1.Brief History and Scope of the Subject

- The Java platform was developed at Sun in the early 1990s with the objective of allowing programs to function regardless of the device they were used on, sparking the slogan "Write once, run anywhere" (WORA). Java is regarded as being largely hardware- and operating system-independent.

- Java was initially promoted as a platform for client-side *applets* running inside web browsers. Early examples of Java applications were the Hot Java web browser and

  the Hot Java Views suite. However, since then Java has been more successful on the server side of the Internet.

- The platform consists of three major parts: the Java programming language, the Java Virtual Machine (JVM), and several Java Application Programming Interfaces (APIs).

- Java is an object-oriented programming language. Since its introduction in late 1995, it became one of the world's most popular programming languages.

- Java programs are compiled to byte code, which can be executed by any JVM, regardless of the environment.

- The Java APIs provide an extensive set of library routines. These APIs evolved into the *Standard Edition* (Java SE), which provides basic

infrastructure and GUI functionality; the *Enterprise Edition* (Java EE), aimed at large software companies implementing enterprise-class application servers; and the *Micro Edition* (Java ME), used to build software for devices with limited resources, such as mobile devices.

- On November 13, 2006, Sun announced it would be licensing its Java implementation under the GNU General Public License; it released its Java compiler and JVM at that time
- Java 8 was released on 18 March 2014 and included some features that were planned for Java 7 but later deferred.

2.Pre-Requisites

Basic knowledge on programming language constructs.

3.Course Objectives:

- To familiarize with the concepts of object oriented programming
- impart the knowledge of AWT components in creation of GUI

4.Course Outcomes:

CO1 : Apply object oriented approach to design software .

CO2 : Create user defined interfaces and packages for a given problem

CO3 : Develop code to handle exceptions.

CO4 : Implement multi tasking with multi threading.

CO5 : Develop applets for web applications.

CO6 : Design and develop GUI programs using AWT components

5.Program Outcomes:

Graduates of the Computer Science and Engineering Program will have ability to

a.apply knowledge of computing, mathematics, science and engineering fundamentals to solve complex engineering problems.

b.formulate and analyze a problem, and define the computing requirements appropriate to its solution using basic principles of mathematics, science and computer engineering.

c. design, implement, and evaluate a computer based system, process, component, or software to meet the desired needs.

d.design and conduct experiments, perform analysis and interpretation of data and provide valid conclusions.

e.use current techniques, skills, and tools necessary for computing practice.

f. understand legal, health, security and social issues in Professional Engineering practice.

g.understand the impact of professional engineering solutions on environmental context and the need for sustainable development.

h.understand the professional and ethical responsibilities of an engineer.

i. function effectively as an individual, and as a team member/ leader in accomplishing a common goal.

j.communicate effectively, make effective presentations and write and comprehend technical reports and publications.

k. learn and adopt new technologies, and use them effectively towards continued professional development throughout the life.

l. understand engineering and management principles and their application to manage projects in the software industry.

6.Mapping of Course Outcomes with Program Outcomes:

	a	b	c	d	e	f	g	h	i	j	k	l
CO1	M		H									
CO2			M									
CO3	M			H								
CO4					M							
CO5					H						M	M
CO6					H						H	H

H- High Level **M-** Medium **L-**Low Level

7.Prescribed Text Books

a) Herbert Schildt, "Java The Complete Reference", TMH, 7th edition.

b) Sachin Malhotra, Saurabh choudhary, "Programming in JAVA", Oxford, 2nd edition.

8.Reference Text Books

a) Joyce Farrel, Ankit R.Bhavsar, "JAVA for Beginners", Cengage Learning, 4th edition.

b) Y.Daniel Liang, "Introduction to Java Programming", Pearson, 7th edition.

c) P.Radha Krishna, "Object Oriented Programming Through Java", Universities Press

9.URLs and Other E-Learning Resources

CDs :

Subject: object oriented system design

Faculty: Prof. A.K. Mazundar IIT, Kharagpur Units : 36

Websites:

www.java.sun.com

www.roseindia.net/java

www.javabeginner.com/learn-java/introduction-to-java-programming

www.tutorialspoint.com/java/index.htm

10.Digital Learning Materials:

http://nptel.ac.in/courses/106103115/36

http://www.nptelvideos.com/video.php?id=1472

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-00-

introduction-to-computer-science-and-programming-fall-2008/video-lectures/lecture-14/

http://192.168.0.49/videos/videosListing/435 (our library IP)

11.Lecture Schedule / Lesson Plan

Topic	No. of Periods
UNIT-I:Fundamentals of OOP and Java	
Need of OOP	1
Principles of OOP Languages	1
Procedural Languages vs OOP	1
Java Virtual Machine	1
Java Features	1
Variables, primitive data types	1
Identifiers, keywords, literals, operators	1
Arrays, type conversion and casting	1
UNIT- II: Class Fundamentals &Inheritance	
Class Fundamentals, Declaring Objects	1
Methods, Constructors	1
this keyword	1

Overloading methods and constructors	1
access control	1
Inheritance Basics, types	1
Using super keyword	1
Method overriding, Dynamic method dispatch	1
Abstract classes, using final with inheritance	1
Object class	1
UNIT –III: Interfaces and Packages	
Interfaces: Defining an interface, Implementing interfaces	2
Nested interfaces	1
Variables in interfaces and extending interfaces	1
Packages: Defining, Creating and Accessing a Package	3
UNIT – IV: Exception Handling & Multithreading	
Exception-Handling	1
Exception handling fundamentals, uncaught exceptions	1

Using try and catch, Multiple catch clauses	1
Nested try statements, throw	1
throws, finally	1
User-defined exceptions	1
Multithreading: Introduction to multi tasking thread life cycle	2
Creating threads	1
Synchronizing threads	2
thread groups	1
UNIT – V: Applets & Event Handling	
Applets: Concepts of Applets	1
Differences between applets and applications, life cycle of an	1
Applet	1
Creating applets	1

Event Handling: Events, Event sources	1
Event classes, Event Listeners, Delegation event model	2
Handling mouse and keyboard events	2
Adapter classes	1

UNIT – VI: AWT

The AWT class hierarchy	1
User interface components- label, button	2
Checkbox, checkboxgroup	1
Choice, list,textfield	1
Scrollbar	1
Layout managers – Flow, Border	1
Grid, Card, GridBag layout	2
Total No.of Periods:	**56**

12. Seminar Topics

- Forms of Inheritance
- AWT hierarchy
- Applet life cycle
- Menu Creation

UNIT – I

Objective:

- To get acquainted with the concepts of object-oriented programming.

Syllabus:

Need of OOP, Principles of OOP Languages, Procedural Languages vs OOP, Java Virtual Machine, Java Features.

Java Programming constructs: variables, primitive data types, identifiers, keywords, literals, operators, arrays, type conversion and casting

Learning Outcomes:

At the end of the unit, students will be able to

- Understand the principles of object oriented programming.
- Differentiate between Oriented Programming and Procedural Oriented Programming.
- Know the Evolution and Features of java.
- Understand Syntax of basic Java Programming Constructs and apply them in writing simple programs.
- Distinguish between Implicit and Explicit Casting.

LEARNING MATERIAL

➢ NEED OF OOP

Procedural Languages:

- C, PASCAL, FORTRAN languages are all procedural languages.
- Procedure oriented programming basically consists of writing a list of instructions for the computer to follow, and organizes these instructions into groups known as functions.

- Problems with Procedural languages are
    - Functions have unrestricted access to global data.
    - Cannot model real world problems very well.
    - Complexity increases as the length of a program increases.
    - Not extensible.

Object Oriented Programming Language:

- The major motivating factor in the invention of object-oriented approach is to remove some of the flaws encountered in the procedural approach.
- In the real-world situations, we have objects which have some attributes and behavior.
- OOP can represent the real-world objects.
- Objects are defined by their unique identity, state and behavior.
- The state of an object is identified by the value of its ***attributes*** and behavior by ***methods.***
- Attributes defines the **data** for an object, as every object has some attributes. For example, attributes of an Account Holder object are Name, DoB, Acount_Number, Aadhar_number and PAN.
- Behavior is synonym to functions or methods, called to perform some task and may manipulate the attributes of an object. For example, behavior exhibited by a account holder are withDrawMoney(), checkBalance(), transferFunds().
- OOP organizes a program around its data(i.e., objects) and a set of well-defined interfaces to that data**.**
- An object-oriented program can be characterized as data controlling access to the code.
- The organization of data and function(s) in object-oriented programs is shown in the figure 1.1.

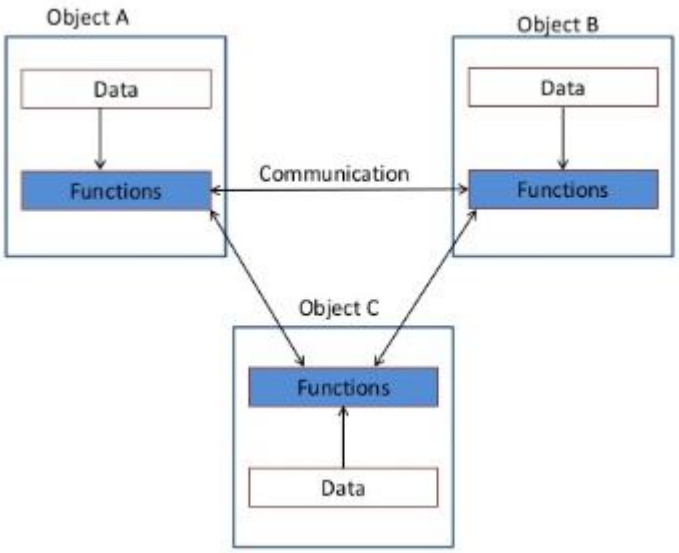

Fig 1.1: Organization of Data and Functions in OOP

- The data of an object can be accessed only by the functions associated with that object. However, functions of one object can access the functions of other objects.

➢ PRINCIPLES OF OOP LANGUAGES

The following are the general concepts of OOP

1. Objects
2. Classes
3. Abstraction
4. Encapsulation
5. Inheritance
6. Polymorphism

1. OBJECTS:

- An object is an entity in the real-world that can be distinguishable with other objects that have some attributes and exhibits some behavior.
- Objects are the ***basic run time entities*** in an object-oriented system.
- They may represent a person, a place, a bank account, a table of data or any item that the program has to handle.
- Objects take up space in the memory and have an associated address.

- When a program is executed, the objects interact with each other by sending messages to one another.
- For example, if “customer” and “account” are two objects in a program, then the customer object may send a message to the account object requesting for the bank balance.
- Each object contains data and code to manipulate data.

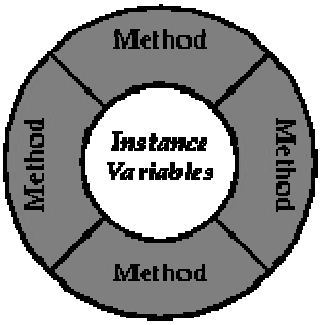

Fig 1.2: Object = Data+Methods

2. CLASSES:

- A class is defined as a blueprint of an object. It serves as a template.
- A class is a combination of common attributes and common behavior, thus we can represent class a ***collection of similar type of objects.***
- Object is an ***instance of a class.***
- The entire set of data and code of an object can be made a **user-defined data type** with the help of class. Objects are variables of the class type.
- Once a class has been defined, we can create any number of objects belonging to that class. Each object is associated with the data of type class with which they are created.
- For example Mango, Apple and Orange are objects from class fruit.
- Classes are user-defined data types and behave like the built-in types of a programming language.

3. ABSTRACTION:

- Abstraction refers to the "***act of representing essential features without including the background details or explanation"***.
- Classes use the concept of abstraction and are defined as a list of abstract attributes and functions operate on these attributes.
- The attributes are called data members because they hold information.
- The functions that operate on these data are called methods or member functions.

4. ENCAPSULATION:

- The process of binding together code and data it manipulates, to hide them from the outside world is called Encapsulation.
- Encapsulation is the most striking feature of a class. The data is not accessible to the outside world, and only those functions which are wrapped in the class can access the data.
- This insulation of the data from direct access by the program is called **data hiding** or **information hiding.**

5. INHERITANCE:

- Inheritance is the process by which ***one object acquires the properties of another object.***
- It supports the concept of hierarchical classification.
- For example the bird **robin** is a part of class '**flying bird**' which is again a part of the class '**bird**'. The principal behind this sort of division is that each derived class shares common characteristics with the class from which it is derived as illustrated in fig 1.3.

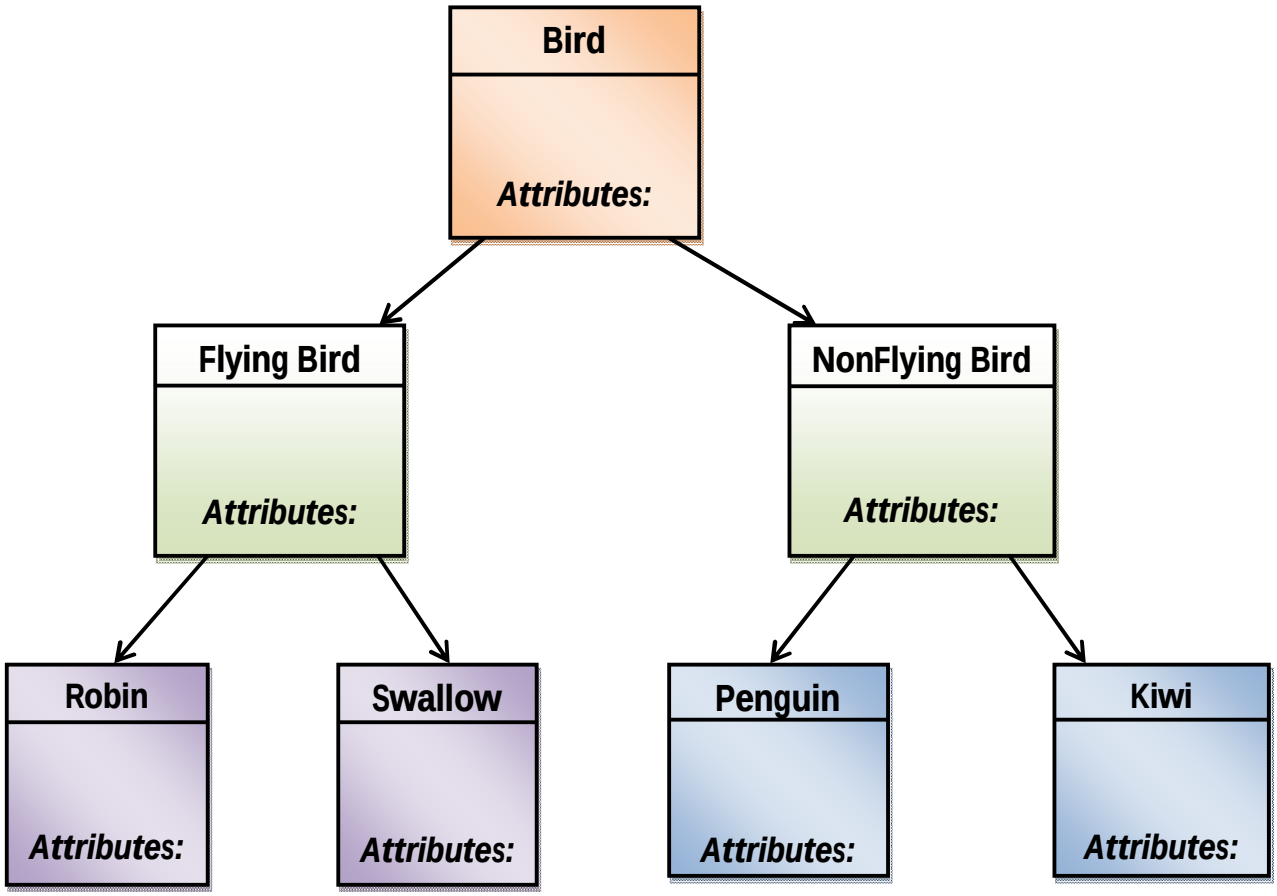

Fig 1.3: Inheritance

- In OOP, the concept of inheritance provides the idea of **reusability**. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class(sub class) from the existing class(super class). The new class will have the combined features of both the classes.

6. POLYMORPHISM:

- Polymorphism, a Greek term, means the ***ability to take more than one form***.
- For example, an operation may exhibit different behaviour at different instances. The behaviour depends upon the types of data used in the operation.

- Consider the operation of addition. For two numbers, the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation.

- This is similar to polysemy (a word having different meanings depending on the context the word is used).
- The figure 1.4 illustrates that a single function name can be used to handle different number and different types of arguments.

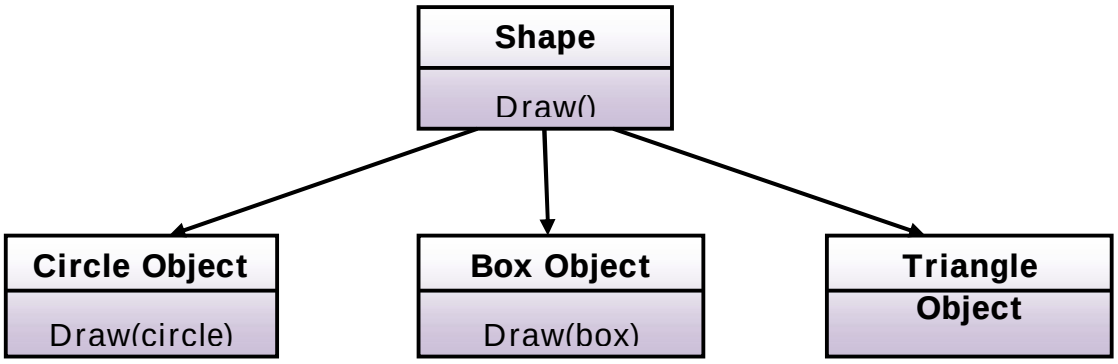

Fig 1.4: Polymorphism

➢ PROCEDURAL LANGUAGES Vs OOP:

Procedural language	Object Oriented language
Separates data from function that operate on them	Encapsulate data and methods in a class
Not suitable for defining abstract types	Suitable for defining abstract types
Debugging is difficult	Debugging is easier
Difficult to implement change	Easier to manage and implement change
Not suitable for larger applications/programs	Suitable for larger programs and applications
Analysis and design not so easy	Analysis and Design Made Easier
Faster	Slower
Less flexible	Highly flexible
Data and procedure based	Object oriented
Less reusable	More reusable

Only data and procedures are there	Inheritance, encapsulation and polymorphism are key features
Uses top down approach	Uses bottom up approach
Only a function calls another function	Object communication is there
C, Basic, FORTRAN	JAVA,C++, VB.NET, C#.NET

➢ JAVA VIRTUAL MACHINE (JVM):

- The key that allows Java to solve both the security and portability problems is that the output of a Java compiler is not executable code rather it is ***byte code.***
- Byte code is a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the *Java Virtual Machine* (*JVM*).
- That is, in its standard form, the JVM is an *interpreter for byte code.*
- Translating a java program into byte code allows us to run a program in a wide variety of environments because only JVM details will differ from platform to platform although all understands the same java byte code.
- Just In Time (JIT) compiler is part of the JVM, which compiles selected portions of the byte code into executable code in real-time, on the fly.
- Only the sequences of byte code that will get benefit from compiling will be given to JIT, the code that requires run-time check during run-time will be given to the interpreter.

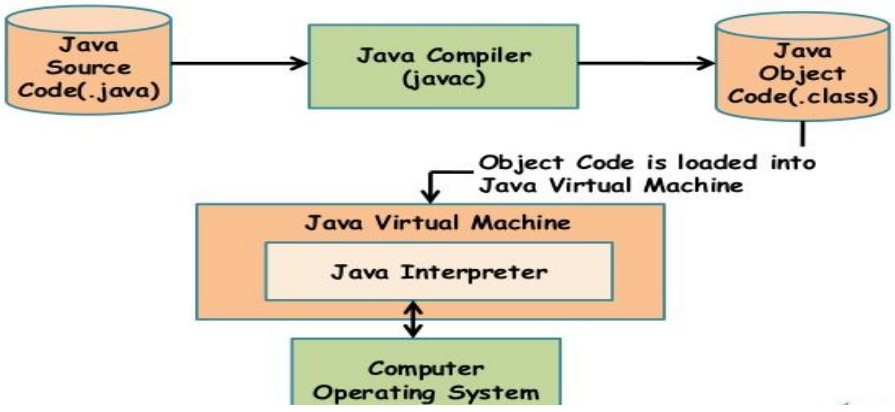

Fig 1.5: JVM

➢ JAVA FEATURES

The key considerations were summed up by the Java team in the following list of buzzwords/features:

- Simple
- Secure
- Portable
- Object-oriented
- Robust
- Multithreaded
- Architecture-neutral
- Interpreted and High performance
- Distributed
- Dynamic

1. Simple:

- Java was designed to be easy for the professional programmer to learn and use effectively.
- Java inherits the C/C++ syntax and many of the object-oriented features of C++, most programmers have little trouble in learning Java.
- Many of C & C++ language features that result in unreliable code were not included in Java

2. Secure:

- Java has several language features that protect the integrity of the system and prevent several common attacks.
- Security becomes an important issue for a language that is being used on internet.
- Java provides security through lack of pointer arithmetic.
- Garbage collection makes java program more secure and robust by automatically freeing the memory.
- Has strict compile-time checking which makes java programs more robust and avoids run-time errors. The compiler also ensures that a program does not access any uninitialized variables.

- Java security model focuses on protecting users from hostile programs downloaded fromuntrusted sources across a network. Programs downloaded over the internet are executed in a ***sandbox***, cannot take any action outside of the boundaries specified by the sandbox.
- By using a Java-compatible web browser, applets can be downloaded from internet without fear of viral infection or malicious intent.

3. Portable:

- Portability is the major aspect of the internet because different types of computers and operating systems connected to it.
- The output of a Java compiler is not executable code. Rather, it is byte code.
- Translating a Java program into byte code makes it much easier to run a program in a wide variety of environments.
- Java Programs can be easily moved from one computer system to another anywhere and at anytime.

4. Object-Oriented:

- Java is pure object-oriented language, i,e., the outermost level of data structure in java is the object.
- Everything in java (constants, variables and methods) are defined inside a class and accessed through objects.
- But some constraints violate the purity of java, which was mainly designed for OOP, but with some procedural elements. For examples, java supports primitive data types that are not objects.

- **Robust:** Java is a strictly typed language, it checks the code both at compile time and runtime.
- The two of the main reasons for program failure are:
  (1) Memory Management Mistakes and
  (2) Mishandled Exceptional Conditions (that is, Run-Time errors).

- Java virtually eliminates memory Management Mistakes: Deallocation is completely automatic in Java as it provides garbage collection for unused objects.
- Java also incorporates the concept of handling the exceptional conditions that may arise in situations such as division by zero or file not found, and thus eliminates the abnormal termination of program.

5. Multithreaded:

- Java was designed to meet the real-world requirement of creating interactive, networked programs.
- To accomplish this, Java supports multithreaded programming, allows writing a program that does many independent subtasks simultaneously.
- For example, while typing in a word-processor the spell check will also does it task simultaneously.

6. Architecture-Neutral:

- One of the main problem faced by programmers is that no guarantee exists that if you write a program today, it will run tomorrow—even on the same machine.
- Operating system upgrades, processor upgrades, and changes in core system resources can all make a program malfunction.
- The goal of Java designers was "**write once; run anywhere, any time, forever."** To a great extent, this goal was accomplished by bytecode, the output from java compiler.

7. Interpreted and High Performance:

- Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode, leads to impressive performance.
- This code can be executed on any system that implements the Java Virtual Machine.
- Also due to the incorporation of multithreading, java improved the overall execution speed of java programs.

- Java byte code was carefully designed so that it would be easy to translate it directly into native machine code for very high performance by using a just-in-time compiler.

8. Distributed:

- Java is designed for the distributed environment of the Internet because it handles TCP/IP protocols
- Java also supports Remote Method Invocation (RMI). This feature enables a program to invoke methods across a network.
- This enables multiple programmers at multiple remote locations can work together on a single project.

9. Dynamic:

- Java programs have run-time type information that is used to verify and resolve accesses to objects at run time. This makes it possible to dynamically link code in a safe and convenient manner.
- Java Supports functions written in other languages such as C and C++. These functions are known as Native Methods, which can be linked dynamically at Run-time.

JAVA PROGRAMMING CONSTRUCTS

➢ VARIABLES:

- The variable is the basic unit of storage in a Java program.
- A variable is defined by the combination of an identifier, a type, and an optional initializer.
- In addition, all variables have a scope, which defines their visibility, and a lifetime.

Declaring a Variable:

- In Java, all variables must be declared before they can be used. So java is termed as a *strongly typed language.*
- The basic form of a variable declaration is shown here:

type identifier [= value][, identifier [= value] ...] ;

- The type is one of Java's atomic types, or the name of a class or interface.
- The identifier is the name of the variable.
- To declare more than one variable of the specified type, use a comma separated list.

Examples:

```
int a, b, c;            // declares three int variables, a, b, and c.
int d = 3, e, f = 5;    // declares three int variables, initializing d and f.
byte z = 22;            // initializes z.
double pi = 3.14159;    // declares and initializes pi.
char x = 'x';           // the variable x has the value 'x'.
```

Dynamic Initialization:

- Java allows variables to be initialized dynamically, using any expression valid at the time the variable is declared.
- **Example:**

```
classDynInit
{
    public static void main(String args[])
    {
    double a = 3.0, b = 4.0;
    double c = Math.sqrt(a * a + b * b);
      // c is dynamically initialized
    System.out.println("Hypotenuse is " + c);
    }
}
```

- Here, three local variables—a, b, and c—are declared. The first two a and b, are initialized by constants. However, c is initialized dynamically to the length of the hypotenuse

The Scope and Lifetime of Variables:

- Java allows variables to be declared within any block.
- "A block begins with an opening curly brace and ends with a closing curly brace". A block defines a scope. Thus, a new scope is created each time a new block starts.
- "A scope determines what objects are visible to other parts of your program". It also determines the lifetime of those objects.
- The scope defined by a method begins with its opening curly brace. However, if that method has parameters, they too are included within the method's scope.
- As a general rule, variables declared inside a scope are not visible (that is, accessible) to code outside that scope.
- Thus, when we declare a variable within a scope, we are localizing that variable and protecting it from unauthorized access and/or modification.
- Scopes can be nested. When this occurs, the outer scope encloses the inner scope. This means that objects declared in the outer scope will be visible to code within the inner scope. However, the reverse is not true. Objects declared within the inner scope will not be visible outside it.
- **Example:**

```
class Scope
{
        public static void main(String args[])
        {
                int x;                  // known to all code within main
                x = 10;
                if(x == 10)
                {                                       // start new scope
                        int y = 20;                     // known only to this block
                        // x and y both known here.
```

```
                System.out.println("x and y: " + x + " " + y);
                x = y * 2;
            }
            y = 100;                        // Error! y not known here
            System.out.println("x is " + x); // x is still known here.
        }
}
```

- Within a block, variables can be declared at any point, but are valid only after they are declared.
- Thus, if you define a variable at the start of a method, it is available to all of the code within that method.
- Conversely, if you declare a variable at the end of a block, it is effectively useless, because no code will have access to it.
- For example, this fragment is invalid because count cannot be used prior to its declaration:

```
// This fragment is wrong!
        count = 100; // oops! cannot use count before it is declared!
        int count;
```

- Variables are created when their scope is entered, and destroyed when their scope is left. This means that a variable will not hold its value once it has gone out of scope.
- Also, a variable declared within a block will lose its value when the block is left. Thus, the lifetime of a variable is confined to its scope.
- If a variable declaration includes an initializer, then that variable will be reinitialized each time the block in which it is declared is entered.

```
// Demonstrate lifetime of a variable.
class LifeTime
{
    public static void main(String args[])
```

```
        {
            int x;
            for(x = 0; x < 3; x++)
            {
                int y = -1;    // y is initialized each time block is entered
                System.out.println("y is: " + y); // this always prints -1
                y = 100;
                System.out.println("y is now: " + y);
            }
        }
    }
```

Output:

```
y is: -1
y is now: 100
y is: -1
y is now: 100
y is: -1
y is now: 100
```

y is reinitialized to –1 each time the inner for loop is entered. Even though it is subsequently assigned the value 100, this value is lost.

- Although blocks can be nested, you cannot declare a variable to have the same name as one in an outer scope.

- For example, the following program is illegal:

```
/ / This program will not compile
    class ScopeErr
    {
    public static void main(String args[])
    {
        int bar = 1;
        {                          / / creates a new scope
            int bar = 2;  / / Compile-time error – bar already defined!
        }
    }
    }
```

➢ PRIMITIVE DATA TYPES

- Java defines eight primitive data types they are: **byte, short, int, long, char, float, double, and boolean.**
- The primitive types are also commonly referred to as simple types
- These can be put in four groups:
  - **Integers:** This group includes byte, short, int, and long, which are for whole-valued signed numbers.
  - **Floating-point numbers:** This group includes float and double, which represent numbers with fractional precision
  - **Characters:** This group includes char, which represents symbols in a character set, like letters and numbers.
  - **Boolean:** This group includes boolean, which is a special type for representing true/false values.
- The primitive types represent single values—not complex objects.
- Although Java is otherwise completely object-oriented, the *primitive types are not*. They are analogous to the simple types found in most other non–

object-oriented languages. The reason for this is efficiency. Making the primitive types into objects would have degraded performance too much.

- The following chart summarizes the default values for all the above data types.

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	Null
boolean	false

1. Integers:

- Java defines four integer types: *byte, short, int, and long.* All of these are signed values.
- The width and ranges of these integer types vary widely, as shown in this table:

Name	Width(in bits)	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

a. byte:

- The smallest integer type is byte. This is a signed 8-bit type that has a range from –128 to 127.
- Variables of type byte are especially useful while working with
  - a stream of data from a network or file.
  - raw binary data that may not be directly compatible with Java's other built-in types.
- Byte variables are declared by use of the keyword **byte**.
- **Example:** byte b, c;

b. short:

- short is a signed 16-bit type.
- It has a range from –32,768 to 32,767. It is probably the least-used Java type.
- **Examples**: short s, t;

c. int:

- The most commonly used integer type is int.
- It is a signed 32-bit type that has a range from –2,147,483,648 to 2,147,483,647.
- In addition to other uses, variables of type int are commonly employed to control loops and to index arrays.
- **Examples**: int a, b =5;

d. long:

- long is a signed 64-bit type and is useful in cases where an int type is not large enough to hold the desired value.
- The range of a long is quite large. This makes it useful when big, whole numbers are needed.
- **Examples:** long d, s;

Example program for all Integer Types:

```
public class Demo
    {
```

```
        public static void main(String[] args)
        {
                byte b =100;
                short s =123;
                int v = 123543;
                int calc = -9876345;
                longamountVal = 1234567891;
                System.out.println("byte Value = "+ b);
                System.out.println("short Value = "+ s);
                System.out.println("int Value = "+ v);
                System.out.println("int second Value = "+ calc);
                System.out.println("long Value = "+ amountVal);
        }
}
```

Output:

```
byte Value = 100
short Value = 123
int Value = 123543
int Second value = -9876345
long Value = 1234567891
```

2. Floating-Point Types:

- Floating-point numbers, also known as real numbers, are used when evaluating expressions that require fractional precision.
- For example, calculations such as square root, or transcendental such as sine and cosine, result in a value whose precision requires a floating-point type.
- There are two kinds of floating-point types, ***float and double***, which represent single- and double-precision numbers, respectively.

Name	Width in Bit	Approximate Range
double	64	4.9e–324 to 1.8e+308
float	32	1.4e–045 to 3.4e+038

a. float:

- The type float specifies a single-precision value that uses 32 bits of storage.
- Single precision is faster on some processors and takes half as much space as double precision, but will become imprecise when the values are either very large or very small
- **Examples**: float hightemp, lowtemp;

b. double:

- Double precision, as denoted by the double keyword, uses 64 bits to store a value.
- Double precision is actually faster than single precision on some modern processors that have been optimized for high-speed mathematical calculations. .
- All transcendental math functions, such as sin( ), cos( ), and sqrt( ), return double values.
- Here is a short program that uses double variables to compute the area of a circle:

```
// Compute the area of a circle.
class Area
{
        public static void main(String args[])
        {
                double pi, r, a;
                r = 10.8;               // radius of circle
                pi = 3.1416;                    // pi, approximately
                a = pi * r * r;                 // compute area
                System.out.println("Area of circle is " + a);
        }
}
```

3. Characters:

- In Java, the data type used to store characters is char.
- In C/C++, char is 8 bits wide. This is not the case in Java.

- Instead, Java uses ***Unicode*** to represent characters. "*Unicode defines a fully international character set that can represent all of the characters found in all human languages*". It is a unification of dozens of character sets, such as Latin, Greek, Arabic, Cyrillic, Hebrew, Katakana, Hangul, and many more. For this purpose, it requires 16 bits. Thus, in ***Java char is a 16-bit type***.
- The range of a char is 0 to 65,535. There are no negative chars.

Example:

```
// Demonstrate char data type.
    classCharDemo
    {
        public static void main(String args[])
        {
            char ch1, ch2;
            ch1 = 88;           // code for X
            ch2 = 'Y';
            System.out.print("ch1 and ch2: ");
            System.out.println(ch1 + " " + ch2);
        }
    }
```

Output: ch1 and ch2: X Y

- Although char is designed to hold Unicode characters, it can also be thought of as an integer type on which you can perform arithmetic operations.
- For example, you can add two characters together, or increment the value of a character variable.
- **Example:**

```
    // char variables behave like integers.
    class CharDemo2
    {
        public static void main(String args[])
        {
            char ch1;
            ch1 = 'X';
```

```
        System.out.println("ch1 contains " + ch1);
        ch1++;                  // increment ch1
        System.out.println("ch1 is now " + ch1);
    }
}
```

Output:

```
ch1 contains X
ch1 is now Y
```

In the program, ch1 is first given the value X. Next, ch1 is incremented. This results in ch1 containing Y, the next character in the ASCII (and Unicode) sequence.

4. Boolean:

- Java has a primitive type, called boolean, for logical values.
- It can have only one of two possible values, true or false.
- boolean is also the type required by the conditional expressions that govern the control statements such as if and for.
- **Example:**

```
// Demonstrate boolean values.
class BoolTest {
public static void main(String args[])
{
    boolean b;
    b = false;
    System.out.println("b is " + b);
    b = true;
    System.out.println("b is " + b);
    // a boolean value can control the if statement
    if(b)       System.out.println("This is executed.");
    b = false;
    if(b)   System.out.println("This is not executed.");
    // outcome of a relational operator is a boolean value
    System.out.println("10 > 9 is " + (10 > 9));
}
}
```

Output:

```
b is false
```

b is true
This is executed.
10 > 9 is true

➢ IDENTIFIERS:

- Identifiers are used for class names, method names, and variable names.
- An identifier may be any combination of uppercase and lowercase letters, digits, or the underscore and dollar-sign characters.
- They must not begin with a digit.
- Java's reserved words keywords cannot used as identifiers.
- Java is case-sensitive, so VALUE is a different identifier than Value.
- **Examples**
  - valid identifiers are:

    AvgTemp count a4 $test this_is_ok
  - Invalid identifier names include these:

    2count high-temp Not/ok

➢ LITERALS:

- A constant value in Java is created by using a literal representation of it.
- Different types of literals those can be assigned to a variable are integer, floating-point, boolean, character and string literals.
- True, false and null are reserved literals in java.
- For example, here are some literals:

  100 98.6 'X' "This is a test"
- Left to right, the first literal specifies an integer, the next is a floating-point value, the third is a character constant, and the last is a string.
- A literal can be used anywhere a value of its type is allowed.

➢ OPERATORS

- An operator performs an action on one or more operands.
- An operator that performs operation on one operand is called a *unary operator(+, -, ++, --)* , on two operands called as *binary operator(+,-,/,*,<<,>>,<,> and more)* and on 3 operands called as *ternary operator(?:).*

Java supports all the three types of operators, in addition to special operators like instanceof, .(dot), new, (type) casting operators.

1. Arithmetic Operators:

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

Operator	Result
+	Addition
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+ =	Addition assignment
– =	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
– –	Decrement

- The operands of the arithmetic operators must be of a numeric type.
- We cannot use them on boolean types, but can use them on char types, since the char type in Java is, essentially, a subset of int.

- **Example:**

```
public class Test
{
public static void main(String args[])
 {
int a = 10;
int b = 20;
int c = 25;
int d = 25;
System.out.println("a + b = " + (a + b) );
System.out.println("a - b = " + (a - b) );
System.out.println("a * b = " + (a * b) );
System.out.println("b / a = " + (b / a) );
System.out.println("b % a = " + (b % a) );
System.out.println("c % a = " + (c % a) );
System.out.println("a++   = " +  (a++) );
System.out.println("b--   = " +  (a--) );
    / / Check the difference in d++ and ++d
System.out.println("d++   = " +  (d++) );
System.out.println("++d   = " +  (++d) );
 }
}
```

Output:

```
a + b = 30
a - b = -10
a * b = 200
b / a = 2
b % a = 0
c % a = 5
a++   = 10
b--   = 11
d++   = 25
++d   = 27
```

2. The Bitwise Operators:

- In Java these will be operated on int and long values.
- If any of the operand is shorter than an int, it is automatically promoted to int before performing the operations.

- These operators act upon the individual bits of their operands.
- Negative numbers are represented in 2's complement arithmetic and then the operators are applied.
- They are summarized in the following table:

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

- **Example:**

```
public class Test
{
        public static void main(String args[])
        {
        int a = 60;          /* 60 = 0011 1100  in binary*/
        int b = 13;          /* 13 = 0000 1101 */
        int c = 0;

            c = a & b;              /* 12 = 0000 1100 */
        System.out.println("a & b = " + c );

            c = a | b;              /* 61 = 0011 1101 */
        System.out.println("a | b = " + c );
```

```
    c = a ^ b;              /* 49 = 0011 0001 */
System.out.println("a ^ b = " + c );

    c = ~a;                 /*-61 = 1100 0011 */
System.out.println("~a = " + c );

    c = a << 2;             /* 240 = 1111 0000 */
System.out.println("a << 2 = " + c );

    c = a >> 2;              /* 15 = 1111 */
System.out.println("a >> 2  = " + c );

    c = a >>> 2;            /* 15 = 0000 1111 */
System.out.println("a >>> 2 = " + c );
  }
}
```

Output:

```
a& b = 12
a | b = 61
a ^ b = 49
~a = -61
a << 2 = 240
a>> 15
a>>> 15
```

3. Relational Operators:

- The relational operators determine the relationship that one operand has to the other.
- Specifically, they determine equality and ordering. The relational operators are shown here:

Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

- The outcome of these operations is a boolean value.
- The relational operators are most frequently used in the expressions that control the if statement and the various loop statements.
- Any type in Java, including integers, floating-point numbers, characters, and boolean can be compared using the equality test, ==, and the inequality test, !=.
- **Example:**

```
public class Test
{
public static void main(String args[])
 {
        int a = 10;
        int b = 20;
        System.out.println("a == b = " + (a == b) );
        System.out.println("a != b = " + (a != b) );
        System.out.println("a > b = " + (a > b) );
        System.out.println("a < b = " + (a < b) );
        System.out.println("b >= a = " + (b >= a) );
        System.out.println("b <= a = " + (b <= a) );
  }
}
```

Output:

```
a == b = false
a != b = true
a > b = false
a < b = true
b >= a = true
b <= a = false
```

Boolean Logical Operators:

- The Boolean logical operators shown here operate only on boolean operands or expressions. These operators combine two boolean values to form a resultant boolean value.

Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR(exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

```
// Demonstrate the boolean logical operators.
class BoolLogic
{
        public static void main(String args[])
        {
                boolean a = true;
                boolean b = false;
                boolean c = a | b;
                boolean d = a & b;
                boolean e = a ^ b;
                boolean f = (!a & b) | (a & !b);
                boolean g = !a;
                System.out.println(" a = " + a);
                System.out.println(" b = " + b);
                System.out.println(" a|b = " + c);
                System.out.println(" a&b = " + d);
                System.out.println(" a^b = " + e);
                System.out.println("!a&b|a&!b = " + f);
                System.out.println(" !a = " + g);
```

```
    }
}
```

Output:

```
a = true
b = false
a|b = true
a&b = false
a^b = true
a&b|a&!b = true
!a = false
```

4. Short-Circuit Logical Operators:

- These are secondary versions of the Boolean AND (&&) and OR (||) operators, and are known as short-circuit logical operators, conditionally evaluate the second operand or expression.

Examples:

(i) In case of AND if the first operand is false, no matter what the second operand is, the answer is false. No need to evaluate the second operand.

```
if (0 == 1 && 2 + 2 == 4)
{
        System.out.println("This line won't be printed.");
}
```

Java does the following:

1. Evaluate 0 == 1, discovering that 0 == 1 is false.
2. Realize that the condition (0 == 1 && whatever) can't possibly be true, no matter what the condition happens to be.
3. Return false (without bothering to check if 2 + 2 == 4).

(ii) In case of OR, if the first operand is true, no matter what the second operand is, the answer is true.

```
if (2 + 2 == 4 || 0 == 1)
```

```
{
    System.out.println("This line will be printed.");
}
```

Java does the following:

1. Evaluate 2 + 2 == 4, discovering that 2 + 2 == 4 is true.
2. Realize that the condition (2 + 2 == 4 || whatever) must be true, no matter what the whatever condition happens to be.
3. Return true (without bothering to check if 0 == 1).

5. The Assignment Operator:

- The assignment operator is the single equal sign i.e. **=**
- It has this general form:

  var = expression;

  Here, the type of var must be compatible with the type of expression.
- The assignment operator allows you to create a chain of assignments.
- For example, consider this fragment:

```
int x, y, z;
x = y = z = 100; // set x, y, and z to 100
```

This fragment sets the variables x, y, and z to 100 using a single statement.

6. The ?: Operator

- Java includes a special ternary (three-way) operator that can replace certain types of if-then-else statements. This operator is the ?:
- General form:

  expression1 ?expression2 : expression3
- Here, expression1 can be any expression that evaluates to a boolean value. If expression1 is true, then expression2 is evaluated; otherwise, expression3 is evaluated. The result of the ?operation is that of the expression evaluated.
- Both expression2 and expression3 are required to return the same type, which can't be void.

- **Example:**

```
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

➢ ARRAYS:

- **Definition**: An array is a memory space allocated that can store multiple values of same data type in contiguous locations.(ex., array 'marks' to represent set of marks of a group of students).
- This memory space can be accessed with a common name and a specific element is accessed by using a subscript or an index inside the brackets, along with the name of the array.(ex., marks[5], stores marks of a fifth student),each individual value in array called as elements.
- Arrays offer a convenient means of grouping related information.
- Arrays of any type can be created and may have one or more dimensions.

ONE-DIMENSIONAL ARRAYS:

- A one-dimensional array is, essentially, a list of like-typed variables.
- **Creating Arrays:**
    - declare a variable of the desired array type.
    - allocate the memory that will hold the array, using new
    - initializing/assigning valuesinto an array.

- **Declaring:** To create an array, you first must create an array variable of the desired type.
- The general form of a one-dimensional array declaration is

  typearray_name[ ];
- Here, type declares the base type of the array. The base type determines the data type of each element that comprises the array. Thus, the base type for the array determines what type of data the array will hold.
- **Creating memory location:** ***new*** is a special operator that allocates memory.
- The general form of **new** as it applies to one-dimensional arrays appears as follows:

  Array_name = new type[size];

  - Here, type specifies the type of data being allocated,
  - size specifies the number of elements in the array,
  - andarray_name is the array variable that is linked to the array.
- **Initializing:** The elements in the array allocated by new will automatically be initialized to zero.
- To assign values to an array: **Array_name[index]=value;**

  Or **type Array_name[ ]={list of values};**
- **Example:** Allocating a 12-element array of integers and links them to month_days.

  month_days = new int[12];

  month_days will refer to an array of 12 integers and all elements in the array will be initialized to zero.
- We can access a specific element in the array by specifying its index within square brackets. All array indexes start at zero.
- The for loops can be used to assign and access values from an array.

- To obtain the number of elements in an array, use the ***length*** property associated with all the arrays in java. I.e., use array name followed by dot operator and the variable *length.*
- **Example:**

month_days[1] = 28;

/ / assigns the value 28 to the second element of month_days.

System.out.println(month_days[3]);/ /displays the value stored at index 3

Example:

```
// Demonstrate a one-dimensional array.
class Array
{
        public static void main(String args[])
        {
        intmarks[]={9,8,6,5,10};
        int n=marks.length;
        System.out.println("marks of a student in a class test are:");
        for(int i=0;i<n;i++)
                System.out.println(marks[i]);
        }
}
```

Output: marks of a student in a class test are: 9 8 6 5 10

MULTIDIMENSIONAL ARRAYS

- In Java, multidimensional arrays are actually arrays of arrays.
- To declare a multidimensional array variable, specify each additional index using another set of square brackets.
- Suppose to store the marks of students in different subjects, need a 2D array conceptualized in the form of table, with rows representing marks of each student and columns represent the subjects.
- For example, the following declares a two dimensional array variable called marks.

int marks[][] = new int[5][6];

This allocates a 5 by 6 array and assigns it to marks, to store marks of 5 students in 6 subjects. Internally this matrix is implemented as an array of arrays of int.

- **Example:**

```
// Demonstrate a two-dimensional array.
class TwoDArray
{
        public static void main(String args[])
        {
                int twoD[][]= new int[4][5];
                int i, j, k = 0;
                for(i=0; i<4; i++)
                {
                        for(j=0; j<5; j++)
                        {
                                twoD[i][j] = k;
                                k++;
                        }
                }
                for(i=0; i<4; i++)
                {
                        for(j=0; j<5; j++)
                                System.out.print(twoD[i][j] + " ");
                        System.out.println();
                }
        }
}
```

Output:

```
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
```

Initializing Multidimensional Arrays:

- Enclose each dimension's initializer within its own set of curly braces.

- The following program creates a matrix where each element contains the product of the row and column indexes.

```
// Initialize a two-dimensional array.
class Matrix
{
        public static void main(String args[])
        {
                double m[][] = {{ 0*0, 1*0, 2*0, 3*0 },
                                { 0*1, 1*1, 2*1, 3*1 },
                                { 0*2, 1*2, 2*2, 3*2 },
                                { 0*3, 1*3, 2*3, 3*3 }};
                int i, j;
                for(i=0; i<4; i++)
                {
                        for(j=0; j<4; j++)
                                System.out.print(m[i][j] + " ");
                        System.out.println();
                }
        }
}
```

Output:

```
0.0  0.0  0.0  0.0
0.0  1.0  2.0  3.0
0.0  2.0  4.0  6.0
0.0  3.0  6.0  9.0
```

Alternative Array Declaration Syntax:

- There is a second form that may be used to declare an array:

  type[ ] var-name;

  Here, the square brackets follow the type specifier, and not the name of the array variable.

- For example, the following two declarations are equivalent:

```
int al[] = new int[3];
int[] a2 = new int[3];
```

The following declarations are also equivalent:

```
char twod1[][] = new char[3][4];
char[][] twod2 = new char[3][4];
```

- This alternative declaration form offers convenience when declaring several arrays at the same time.
- For example,
  int[] num, num2, num3; // creates three array variables of type int.

➢ TYPE CONVERSION AND CASTING

- If the two types are compatible, then Java will perform the conversion automatically.
- For example, it is always possible to assign an int value to a long variable.
- However, not all types are compatible, and thus, not all type conversions are implicitly allowed.
- For instance, there is no automatic conversion defined from double to byte. Fortunately, it is still possible to obtain a conversion between incompatible types.
- To do so, you must use a cast, which performs an explicit conversion between incompatible types.

Java's Automatic Conversions

- When one type of data is assigned to another type of variable, an automatic type conversion will take place if the following two conditions are met:
  - The two types are compatible.
  - The destination type is larger than the source type.
- When these two conditions are met, a **widening conversion** takes place.
- For example, a smaller box can be placed in short, short in an int, int in long, and so on. Any value can be assigned to double. Any value except a

double can be assigned to a float. Any whole number can be assigned to longand int, short, byte and char all can fit inside int.

- byte b=10; //byte variable
- inti=b; // implicit widening byte to int

- For widening conversions, the numeric types, including integer and floating-point types, are compatible with each other.
- However, there are no automatic conversions from the numeric types to char or boolean. Also, char and boolean are not compatible with each other.
- Java also performs an automatic type conversion when storing a literal integer constant into variables of type byte, short, long, or char.

Casting Incompatible Types

- For example, what if you want to assign an int value to a byte variable? This conversion will not be performed automatically, because a byte is smaller than an int.
- For example, a bigger box has to be placed in a small box. Then the small box has to be chopped(casted) so that the bigger box (which has now become smaller) can be placed in the small box.
- This kind of conversion is sometimes called a **narrowing conversion**, and also termed as casting, since you are explicitly making the value narrower so that it will fit into the target type.
- To create a conversion between two incompatible types, you must use a cast.
- A cast is simply an explicit type conversion.
- It has this general form:

  (target-type) value

  Here, target-type specifies the desired type to convert the specified value to.

- For example, the following fragment casts an int to a byte. If the integer's value is larger than the range of a byte, it will be reduced modulo (the remainder of an integer division by the) byte's range.

```
int a;
byte b;
// ...
b = (byte) a;
```

- A different type of conversion will occur when a floating-point value is assigned to an integer type: truncation. As you know, integers do not have fractional components. Thus, when a floating-point value is assigned to an integer type, the fractional component is lost.
- For example, if the value 1.23 is assigned to an integer, the resulting value will simply be 1. The 0.23 will have been truncated.
- The following program demonstrates some type conversions that require casts:

```
// Demonstrate casts.
class Conversion
{
public static void main(String args[])
 {
        byte b;
        int i = 257;
        double d = 323.142;
        System.out.println("\nConversion of int to byte.");
        b = (byte) i;
        System.out.println("i and b " + i + " " + b);
        System.out.println("\nConversion of double to int.");
        i = (int) d;
        System.out.println("d and i " + d + " " + i);
        System.out.println("\nConversion of double to byte.");
        b = (byte) d;
        System.out.println("d and b " + d + " " + b);
  }
}
```

Output:

```
Conversion of int to byte.
i and b 257 1
Conversion of double to int.
d and i 323.142 323
Conversion of double to byte.
d and b 323.142 67
```

Automatic Type Promotion in Expressions

- In addition to assignments, there is another place where certain type conversions may occur: in expressions.
- In an expression, the precision required of an intermediate value will sometimes exceed the range of either operand.
- For example, examine the following expression:

```
byte a = 40;
byte b = 50;
byte c = 100;
int d = a * b / c;
```

The result of the intermediate term a*b easily exceeds the range of either of its byte operands.

- To handle this kind of problem, Java automatically promotes each byte, short, or char operand to int when evaluating an expression. This means that the sub expression a*b is performed using integers—not bytes. Thus, 2,000, the result of the intermediate expression, 50 * 40, is legal even though a and b are both specified as type byte.
- As useful as the automatic promotions are, they can cause confusing compile-time errors.
- For example, this seemingly correct code causes a problem: byte b = 50;

```
b = b * 2; / / Error! Cannot assign an int to a byte!
```

The code is attempting to store 50 * 2, a perfectly valid byte value, back into a byte variable. However, because the operands were automatically promoted to int when the expression was evaluated, the result has also been

promoted to int. Thus, the result of the expression is now of type int, which cannot be assigned to a byte without the use of a cast.

The Type Promotion Rules

- Java defines several type promotion rules that apply to expressions.
- They are as follows:

1. All byte, short, and char values are promoted to int
2. If one operand is a long, the whole expression is promoted to long.
3. If one operand is a float, the entire expression is promoted to float.
4. If any of the operands is double, the result is double.

- **Example:** Demonstrates how each value in the expression gets promoted to match the second argument to each binary operator:

```
class Promote
{
        public static void main(String args[])
        {
                byte b = 42;
                char c = 'a';
                short s = 1024;
                int i = 50000;
                float f = 5.67f;
                double d = .1234;
                double result = (f * b) + (i / c) - (d * s);
                System.out.println((f * b) + " + " + (i / c) + " - " + (d * s));
                System.out.println("result = " + result);
        }
}
```

- **Explanation:** In the first subexpression, f * b, b is promoted to a float and the result of the subexpression is float. Next, in the subexpression i/c, c is promoted to int, and the result is of type int. Then, in d*s, the value of s is promoted to double, and the type of the subexpression is double. Finally, these three intermediate values, float, int, and double, are considered. The outcome of float plus an int is a

float. Then the resultant float minus the last double is promoted to double, which is the type for the final result of the expression.

UNIT-I

Assignment-Cum-Tutorial Questions

SECTION-A

Objective Questions

1. Java programs are _____ [ ]

(a) Compiled (b) Interpreted

(c) Both Compiled & Interpreted (d) None of these

2. The outcome of a Java Compiler is _____ file [ ]

3. .class (b) .obj (c) .exe (d)None of these

4. If an expression contains double, int, float, long, then whole

a. expression will promoted into which of these data types? [ ]

b. (a) long (b) int (c) double (d) float

5. Which of these can be returned by the operator & . [ ]

a. (a) int (b)boolean (c)char (d) int or boolean

6. Consider the statement **c=a-(b*(a/b)).** Here c contains ___ [ ]

(a) Difference of a and b (b)Sum of a and b

(c) Quotient of a/b (d) Remainder of a/b

7. With x = 1, which of the following are legal lines of Java code for changing the value of x to 2 [ ]

i. (1) x++; (2) x=x+1; (3) x+=1; (4)x=+1

(a) 1, 2 & 3 (b) 1 & 4 (c) 1, 2, 3 & 4 (d) 3 & 2

8. What is the output of the following program? [ ]

9. class increment {

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

double var1 = 1 + 5;
```

```
        double var2 = var1 / 4;
        int var3 = 1 + 5;
        int var4 = var3 / 4;
        System.out.print(var2 + " " + var4); } }
```

(a) 1 1 (b) 0 1 (c) 1.5 1 (d) 1.5 1.0

10. Consider the following statements

```
byte b;        / / statement1
int i=100;     / / statement2
b=i;           / / statement3
```

Which of the above 3 statements will cause a compilation error:

(a) statement 1 (b) statement 2 (c)statement3 (d)none

11. What is the output of the following program? [ ]

```
class conversion {
    public static void main(String args[]) {
        double a = 295.04;
        int  b = 300;
        byte c = (byte) a;
        byte d = (byte) b;
        System.out.println(c + " "  + d); } }
```

(a) 38 43 (b) 39 44 (c) 295 300 (d) 295.04 300

12. What does this code print? [ ]

```
int arr[] = new int [5];
System.out.print(arr);
```

(a) 0 (b) value stored in arr[0] (c) 00000 (d) None

13. What is the output of this program? [ ]

```
class bitwise_operator {
public static void main(String args[]) {
        int a = 3;
        int b = 6;
        int c = a | b;
        int d = a & b;
        System.out.println(c + " "  + d); } }
```

(a) 7 2 (b) 7 7 (c) 7 5 (d) 5 2

14. What is the output of this program? [ ]

```
class Modulus {
        public static void main(String args[]) {
            double a = 25.64;
            int  b = 25;
```

```
        a = a % 10;
        b = b % 10;
        System.out.println(a + " "  + b); } }
```

(a)5.640000000000001 5 (b)5.640000000000001 5.0

(c)5 5 (d) 5 5.640000000000001

15. What is the output of this program? [ ]

```
class Output {
    public static void main(String args[]) {
        int a = 1;
        int b = 2;
        int c;
        int d;
        c = ++b;
        d = a++;
        c++;
        b++;
        ++a;
        System.out.println(a + " " + b + " " + c); } }
```

(a) 3 2 4 (b)3 2 3 (c)2 3 4 (d) 3 4 4

16. Which of these can be returned by the operator & . [ ]

(a) int (b)boolean (c)char (d) int or boolean

SECTION-B

SUBJECTIVE QUESTIONS

1) Summarize the Need of OOP.

2) List and explain the Principles of OOP paradigm

3) Differentiate Procedure Oriented Programming (POP) with Object Oriented Programming (OOP).

4) List and explain the Features of java.

5) Outline the role of JVM in making Java platform independent.

6) Consider the statements below:

```
byte b;     // statement1
int a;      // statement2
a=b;        // statement3
b=a;        // statement4
```

Comment about statement 3 and statement4.

7) Write a java program to do linear search on a list of integers

8) Write a java program to check whether a given number is prime or not.

9) Write a java to multiply 2 numbers without using * operator.
[**HINT:** use the operator + and loop statement]

10) Write a java program to sort given list of integers in ascending order.