1.1 INTRODUCING JAVA

- Java is a blend of the best elements of its rich heritage combined with the innovative concepts required by its unique mission.
- Although Java has become inseparably linked with the online environment of the Internet, it is important to remember that Java is first and foremost a programming language.
- Computer language innovation and development occurs for two fundamental reasons:
  (i) To adapt to changing environments and uses
  (ii) To implement refinements and improvements in the art of programming
- Java is related to C++, which is a direct descendant of C. Much of the character of Java is inherited from these two languages. From C, Java derives its syntax. Many of Java's object-oriented features were influenced by C++.
- Java was deeply rooted in the process of refinement and adaptation that has been occurring in computer programming languages for the past several decades.
- In an OOP environment, data is considered as a critical element and is tied to functions that operate on it. A set of data that is encapsulated in an object is known as the member data and the functions that operates on these data are known as the member functions.
- Java is a general-purpose computer programming language which enables computer programmers to write computer instructions using English based commands, instead of having to write in numeric codes.
- Like English language, Java has a set of rules (syntax) that determine how the instructions are written. Once, a program has been written, the high-level instructions are translated into numeric codes that computers can understand and execute.

1.2 A SHORT HISTORY OF JAVA

- Java was developed at Sun Microsystems (which has since been acquired by Oracle) in 1991, by a team James Gosling, Patrick Naughton, Chris Wrath, Ed Frank and Mike Sheridan as its members. The language was initially called Oak. It was later termed as Java.
- Between the initial implementation of Oak in the fall of 1992 and the public announcement of Java in the spring of 1995, many more people contributed to the design and evolution of the language. Bill Joy, Arthur van Hoff, Jonathan Payne, Frank Yellin, and Tim Lind holm were key contributors to the maturing of the original prototype.
- Java is a general-purpose, Object-Oriented Programming (OOP) language. James Gosling is called Father of Java Programming.
- Java is an Object-Oriented Programming language means Java enables us not only to organize our program code into logical unit called objects but also to take advantage of encapsulation, inheritance and polymorphism and so on.

- As of March 2021, the latest version is Java 16, with Java 11, a currently supported long-term support (LTS) version, released on September 25, 2018. Oracle released the last zero-cost public update for the legacy version Java 8 LTS in January 2019 for commercial use, although it will otherwise still support Java 8 with public updates for personal use indefinitely. Other vendors have begun to offer zero-cost builds of OpenJDK 8 and 11 that are still receiving security and other upgrades.
- Oracle (and others) highly recommend uninstalling outdated versions of Java because of serious risks due to unresolved security issues. Since Java 9, 10, 12, 13, 14, and 15 are no longer supported, Oracle advises its users to immediately transition to the latest version (currently Java 16) or an LTS release.
- An application is a type of software that allows user to perform specific task. Sun Microsystems Inc. has divided Java into following parts:
  1. Java SE (Java Standard Edition): Contains basic core Java classes and used to develop standard applets and applications.
  2. J2EE (Java Enterprise Edition): Contains classes that are beyond Java SE and used to providing business solutions on a network.
  3. J2ME (Java Micro Edition): Java ME is for developers who develop code for portable devices like a PDA, a cellular phone and so on.

1.3 FEATURES OR BUZZWORDS OF JAVA

- The fundamental forces that necessitated the invention of Java are portability and security, other factors also played an important role in molding the final form of the language.
- The key considerations were summed up by the Java team in the following list of buzzwords:
  1. **Simple:** Java is a simple language. There are various concepts that makes the Java as a simple language. Java is designed to be easy to learn. Programs in Java are easy to write and debug because Java does not use the pointers, preprocessor header files, operator overloading etc.
  2. **Secure:** Java is aimed to be used in networked or distributed environments. Java does not allocate direct pointer to memory, this makes it impossible to accidentally reference memory that belongs to other program.
  3. **Portable:** Being architectural-neutral and having no implementation dependent aspects of the specification makes Java portable. Java code is portable. It was an important design goal of Java that it be portable so that as new architectures (due to hardware, operating system, or both) are developed, the Java environment could be ported to them.
  4. **Object-oriented:** Java is a truly object oriented language. Almost everything in java is an object. Java has a wide variety of classes. These classes are organized in various packages. We can use them in our programs while implementing inheritance. The object model in java is simple and easy to extend.
  5. **Robust:** Java is a robust language because it provides many safeguards to ensure reliable code. Java does not allow pointer which avoids security problem. There is no

concept of reference variable. This eliminates the possibility of overwriting memory and corrupting data. Automatic garbage collection eliminates memory leaks.

6. **Multithreaded:** Java was designed to meet the real-world requirement of creating interactive, networked programs. To accomplish this, Java supports multithreaded programming, which allows you to write programs that do many things simultaneously. The Java run-time system comes with an elegant yet sophisticated solution for multi-process synchronization that enables you to construct smoothly running interactive systems.
7. **Architecture-neutral:** A central issue for the Java designers was that of code longevity and portability. One of the main problems facing 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 combine to make a program malfunction. Their goal was "write once; run anywhere, anytime, forever." To a great extent, this goal was accomplished.
8. **Interpreted:** As described earlier, Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode. Most previous attempts at cross-platform solutions have done so at the expense of performance.
9. **High Performance:** As explained earlier, the Java bytecode was carefully designed so that it would be easy to translate directly into native machine code for very high performance by using a just-in-time compiler.
10. **Distributed:** Java facilitates the building of distributed application by a collection of classes for use in networked applications. Java is designed for the distributed environment of the Internet.
11. **Dynamic:** Java language is capable of dynamically linking in new class libraries, methods and objects. Because Java is interpreted, Java is an extremely dynamic language, at runtime, the Java environment can extends itself by linking in classes that may be located on remote servers on a network, (for example; the Internet).

1.4 JAVA ENVIRONMENT - COMPILER, INTERPRETER, JVM

- Java environment includes a large number of development tools [part of system known as JDK (Java Development Kit)].

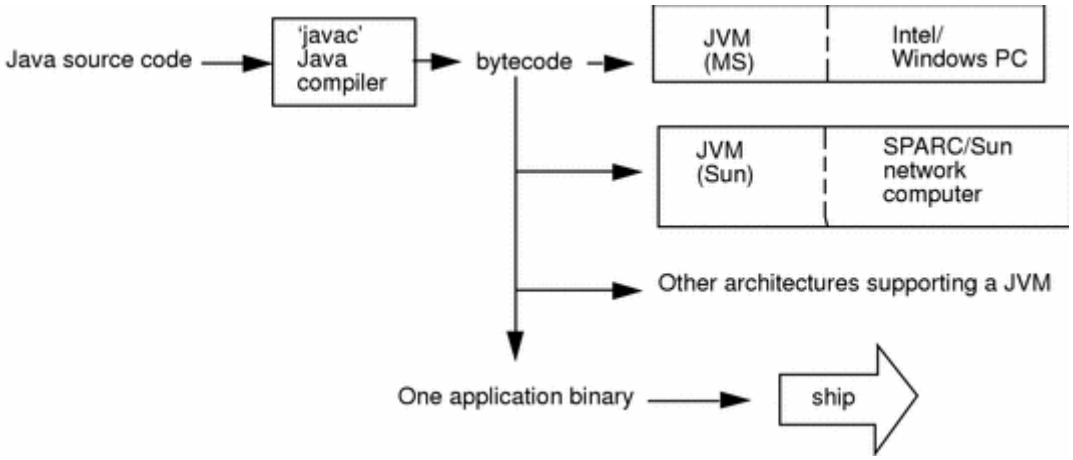

Fig. 1.1: Java Environment

- The Java source file is stored as a '.java' file and is compiled to get an executable '.class' file.
- A ClassLoader is use to download the executable file, the bytecode (a byte long instruction) of which is then interpreted, a process which makes the file understood by the underlying processor which executes the file instructions.
- The so-called 'native code' that the processor understands is never stored, and has to be generated from the .class file each time. Using a 'Just-In-Time (JIT)' compiler with the interpreter.
- The JIT compiler enables all the frequently used class files to be compiled into native code just the once, rather than each time they are used. This improves the speed of execution of the commands to an acceptable level. The stored native code last only for the lifetime of the application.
- The API provided by the language is constantly available to the JVM, and the entire package, excepting the compiler that generates the .class file from the .java file, is referred to as the Java Runtime Environment (JRE).

COMPILER(JIT)

- JIT (Just In Time) compiler is a module inside the JVM which helps in compiling certain parts of byte code into the machine code for higher performance.
- JVM translates bytecode into machine language as they are called just-in-time compilation (JIT).The Just-In-Time (JIT) compiler is a component of the Java Runtime Environment that improves the performance of Java applications at run time.
- Java programs consists of classes, which contain platform neutral bytecode that can be interpreted by a JVM on many different computer architectures.
- At run-time, the JVM loads the class files, determines the semantics of each individual bytecode, and performs the appropriate computation. The additional processor and

memory usage during interpretation means that a Java application performs more slowly than a native application.

- The JIT compiler helps improve the performance of Java programs by compiling bytecode into native machine code at run time.

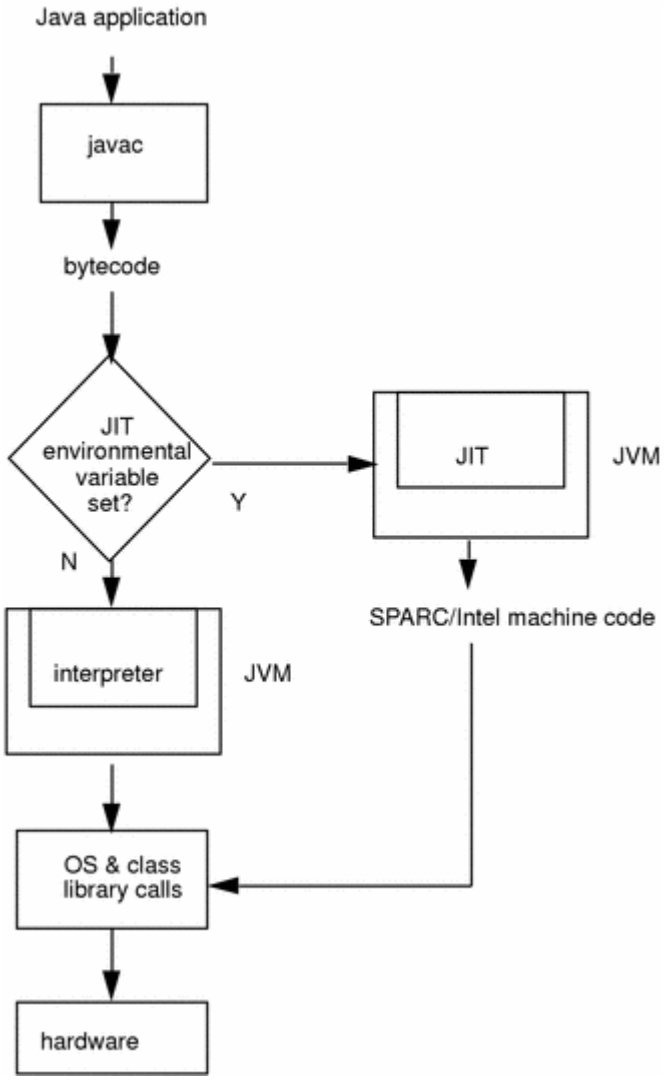

Fig. 1.2: JIT Compile Process

- When the JIT compiler environment variable is on (the default), the JVM reads the .class file for interpretation and passes it to the JIT compiler. The JIT compiler then compiles the bytecodes into native code for the platform on which it is running. The next figure illustrates the JIT compile process.
- The JIT compiler is enabled by default, and is activated when a Java method is called. The JIT compiler compiles the bytecode of that method into native machine code, compiling it "just in time" to run.

[redacted] (Interpreter) JRE

- Java Runtime Environment (JRE) facilitates the execution of programs in Java. The Java Runtime Environment is a set of software tools which are used for developing java applications. It is used to provide runtime environment.
- It mainly includes the following:
  1. Java Virtual Machine,
  2. Runtime Class Libraries,
  3. User Interface Toolkits, and
  4. Deployment Technologies.
- Java language has been designed to work in a platform-independent manner, means that the result is independent of the hardware used, the CPU executing the command or the operating system supporting the system.
- In order to achieve this dream concept of Write Once and Run Anywhere (WORA), we have a Java Runtime Environment (JRE) running in the machine on top of the operating system.

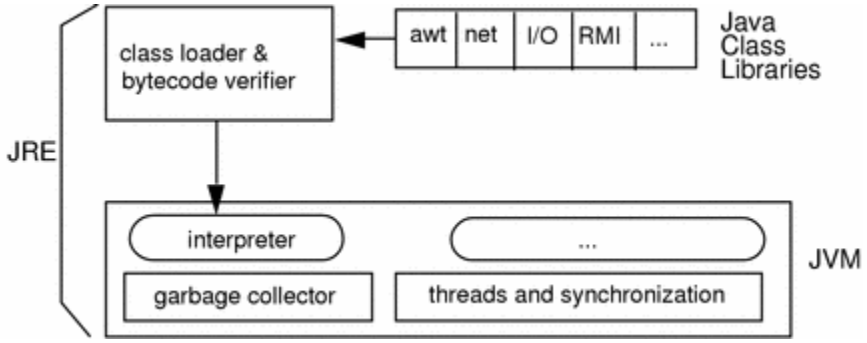

Fig. 1.3: Java Runtime Environment (JRE)

- As illustrated in Fig. 1.3 the JRE has following two major components:
  1. Java Base Classes is the implementation of the Java APIs used in the program. JRE effectively uses these implements to achieve the results.
  2. The Java Virtual Machine (JVM) provides a platform-independent way of executing code, by abstracting the differences between operating systems and CPU architectures.

[redacted] JVM

- Java Virtual Machine (JVM) is a virtual machine that provides runtime environment to execute java byte code.
- When writing Java applications, the developer does not need to directly call Windows (or other operating systems) library functions. The Java compiler does not write native instructions, instead it generates bytecodes for a virtual machine called Java Virtual Machine (JVM).

- The JVM can be considered as a virtual machine that executes the byte code form of the Java code using the available physical machine. In other words, it is a virtual machine that works like a CPU of a computer.

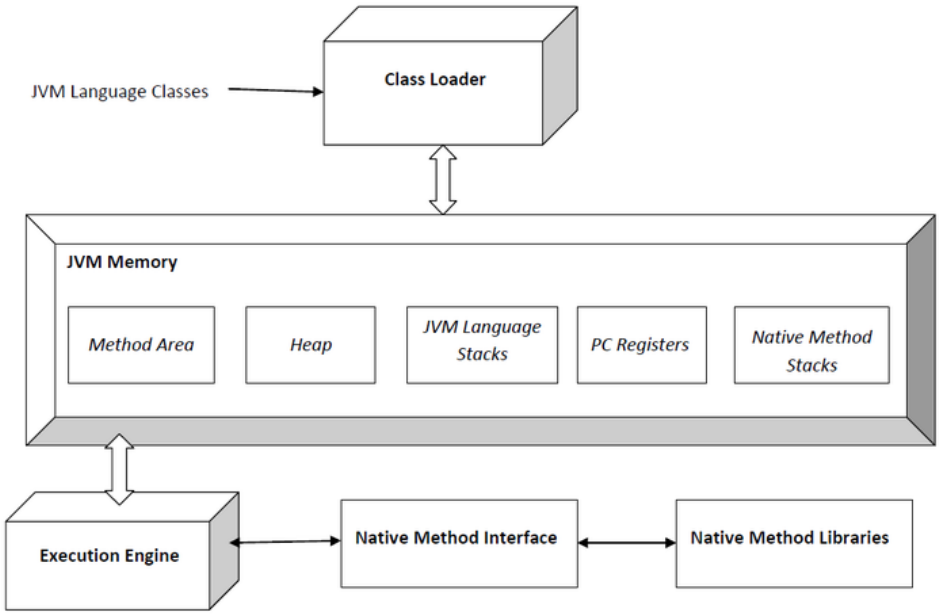

Fig. 1.4: Architecture of JVM

- JVM architecture contains following components:
  1. **Class Loader:** Class loader loads the Class for execution.
  2. **Method area:** Stores pre-class structure as constant pool.
  3. **Heap:** Heap is in which objects are allocated.
  4. **Stack:** Local variables and partial results are store here. Each thread has a private JVM stack created when the thread is created.
  5. **Program register:** Program register holds the address of JVM instruction currently being executed.
  6. **Native method stack:** It contains all native used in application.
  7. **Execution Engine:** Execution engine controls the execute of instructions contained in the methods of the classes.
  8. **Native Method Interface:** Native method interface gives an interface between java code and native code during execution.
  9. **Native Method Libraries:** Native Libraries consist of files required for the execution of native code.

1.5 SIMPLE JAVA PROGRAM

Java Program Structure:

- A Java language, program may contain many classes of which main() is defined in only one class.

- Classes in Java program contain data members and methods that operate on the data members of the class. Methods in Java Program may contain datatype declarations and executable statements.
- To write a Java program, we first define classes and then put them together.

Documentation Section	<----- Suggested
Package Statements	<----- Optional
Import Statements	<----- Optional
Interface Statements	<----- Optional
Class Definitions	<----- Optional
Main method class	<----- Essential

Fig. 1.5: Structure of Java Program

- A Java program structure contains following sections:
  1. **Documentation Section:** This section is a set of comment lines giving the name of the program, the author and the other details, which the programmer would like to refer.
  2. **Package Statement:** In Java files, the first statement allowed is a package statement. This statement declares a package name and informs the compiler that the classes defined here belong to this package.

     For example: package employee;

     It is optional because no need and our class to be part of packages.
  3. **Import Statement:** Similar to #include statement in C.
     - For example, import employee.test;
     - In above example, import statement instructs the interpreter to load the test class contained in package employee.
     - Using import statement; we can access to classes that part of other named packages.
  4. **Interface Statement:** An interface is like class but includes group of methods declaration. Interfaces are used when we want to implement multiple inheritance feature.
  5. **Class Definition:** Java program may contain many/multiple class definition. Classes are primary feature of Java program. The classes in Java are used to map real-world problems.
  6. **Main Method Class:** Java stand-alone program requires main method as starting point and this is essential part of program Main method in Java program creates object of various classes and on reaching, end of the main the program terminate and the control passes back to the operating system.

Program 1.1: First Java Program.

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

```
    {
        System.out.print("My First Java Program with Kamil");
    }
}
```

Output:

```
My First Java Program with Kamil
```

Compiling the Program:

- To compile the above program, execute the compiler, javac specifying the name of the source file on the command line. This is shown below:

```
C:\> javac SimpleJava.java
```

- Javac compiler creates a file named 'MyProgram.class' which contains the byte code version of the program. It is an intermediate representation of the program.

Executing the Program:

- To actually run the program you must use the Java application launcher called 'java'. This is shown below:

```
C:\> java SimpleJava
```

Explanation

- **Public:** It is an access specifier. We should use a public keyword before the main() method so that jvm can identify the execution point of the program. If we use private, protected, and default before the main() method, it will not be visible to jvm.
- **Static:** You can make a method static by using the keyword static. We should call the main() method without creating an object. Static methods are the method which invokes without creating the objects, so we do not need any object to call the main() method.
- **Void:** In java, every method has the return type. Void keyword acknowledges the compiler that main() method does not return any value.
- **Main():** It is a default signature which is predefined in the jvm. It is called by jvm to execute a program line by line and end the execution after completion of this method. We can also overload the main() method.
- **String args[]:** The main() method also accepts some data from the user. It accepts a group of strings, which is called a string array. It is used to hold the command line arguments in the form of string values.

1.6 TYPES OF COMMENTS

- Comments are non-executable statements and are ignored by the Java compiler. The comments increases the readability of the programs.
- Comments are an integral part of any program. They help the person reading the code (often you) better understand the intent and functionality of the program.
- Java language provides three styles of comments as given below:

Sr. No.	Comment	Description
1.	/* text */	The compiler ignores everything from /* to */.
2.	// text	The compiler ignores everything from // to the end of the line.
3.	/** documentation */	This is a documentation comment and in general its called doc comment. The JDK javadoc tool uses doc comments when preparing automatically generated documentation.

1. **Line Comments:**
   - It start with two forward slashes (//) and continue to the end of the current line. Line comments do not require an ending symbol.
   - Java's single line comments are proved useful for supplying short explanations for variables, function declarations, and expressions.

   Example: //This is Single Line Comment
2. **Block Comments:**
   - Java's multi-line or slash-star or traditional comment is a piece of text enclosed in slash-star (/*) and star-slash (*/).
   - Java's multi-line comments are useful when the comment text does not fit into one line; therefore needs to span across lines.

   Example: /*This is Multi Line Comment */
3. **Javadoc Comments:**
   - Javadoc comments are used to document the new classes you create as a programmer i.e. provide information about the class for users of the class to read.
   - Java documentation comments is given by /**.....*/.

   Example: /** Welcome to Nirali Prakashan */

Program 1.2: Program for Java comment.

```
/**
   This is a Documentation Comment.
*/
public class CommentsDemoExample
{
   public static void main(String args[])
   {
      /* This is a Multi-line Comment */
      System.out.println("Hello World");// This is a Single Live Comment.
   }
}
```

Output:

```
Hello World
```

1.7 DECLARING 1D, 2D ARRAY

- An array is a group of like-typed variables that are referred to by a common name. Arrays of any type can be created and may have one or more dimensions.
- A specific element in an array is accessed by its index. Arrays offer a convenient means of grouping related information.
- Java provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- Array is a contiguous fixed-length structure for storing multiple values of the same type. Array in Java is index based, first element of the array is stored at 0 index.

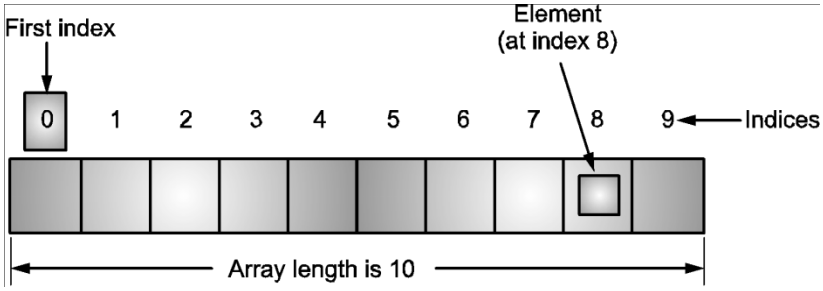

Fig. 1.6: Arrays

One-Dimensional (1D) Array

- An array is a sequence of variables of the same data type. Array having only one subscript variable is called One-Dimensional array.
- It is also called as Single Dimensional Array or Linear Array.
- A one-dimensional array is, essentially, a list of like-typed variables. To create an array, you first must create an array variable of the desired type.
- To declare an array, you specify the name of the array and the data type, as you would for any other variable. Adding an empty set of brackets ([ ]) indicates that the variable is an array.
- The general **form/syntax** of a one-dimensional array declaration is given below:

```
datatype array_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.
- **For example**, the following declares an array named month_days with the type "array of int":

```
int month_days[];
```

Instantiating Arrays:

- As we mentioned earlier, Java arrays are objects, so to allocate memory for an array, you need to instantiate the array using the new keyword. Here, is the **syntax** for instantiating an array:

  ```
  array_name = new datatype[size];
  ```

  where, size is an expression that evaluates to an integer and specifies the number of elements in the array.

  For example: `month_days = new int[12];`

Assigning Initial Values to Arrays:

- Java allows you to instantiate an array by assigning initial values when the array is declared. To do this, you specify the initial values using a comma-separated list within curly braces as given below:

  ```
  datatype array_name[] = {value0, valuel, value2, □ valueN};
  ```

  where, valueN is an expression that evaluates to the data type of the array and is the value to assign to the element at index N.

- Note that we do not use the new keyword and we do not specify a size for the array, The number of elements in the array is determined by the number of values in the initialization list.

- For example, following statement declares and instantiates an array of Numbers:

  ```
  int nine = 9;
  int[ ] Numbers = {2, 4, 3, 8, one, one + 2, 13, 14, 17, 18};
  ```

- Because 10 values are given in the initialization list, this array has 10 elements, Notice that the values can be an expression, for example, one and one + 2.

Program 1.3: Program for one-dimensional (1D) array.

```
import java.util.*;
class arraydemo
{
   public static void main(String args[])
   {
      inta[]={2,4,6,3,1};
      System.out.println("Number of elements in array a: "+a.length);
      System.out.println("Elements in array a:" + a.length);
      for(int i=0;i <a.length;i++)
      System.out.print (a[i] + "\t");
   }
}
```

Output:

```
Number of elements in array a: 5
Elements in array a
2       4      6       3       1
```

Program 1.4: Program for how to create, initialize, and process arrays.

```
public class TestArray
{
   public static void main(String[] args)
   {
      double[] myList = {1.9, 2.9, 3.4, 3.5};
      // Print all the array elements
      for (int i = 0; i < myList.length; i++)
      {
         System.out.println(myList[i] + " ");
      }
      // Summing all elements
      double total = 0;
      for (int i = 0; i < myList.length; i++)
      {
         total += myList[i];
      }
      System.out.println("Total is " + total);
      // Finding the largest element
      double max = myList[0];
      for (int i = 1; i < myList.length; i++)
      {
         if (myList[i] > max) max = myList[i];
      }
      System.out.println("Max is " + max);
   }
}
```

Output:

```
1.9
2.9
3.4
3.5
Total is 11.7
Max is 3.5
```

Multidimensional Array (2D)

- In Java, multidimensional arrays are actually arrays of arrays. These arrays look and act like regular multidimensional arrays. However, as you will see, there are a couple of subtle differences.
- To declare a multidimensional array variable, specify each additional index using another set of square brackets. **For example,** the following declares a two dimensional array variable called two D.

```
int twoD[][] = new int[4][5];
```

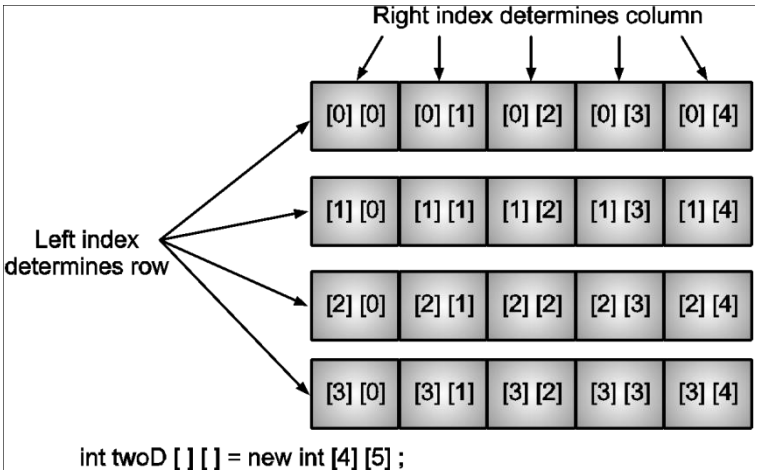

Fig. 1.7

- To declare a multidimensional array, we use the same syntax as for a array, except that we include an empty set of brackets for each dimension.
- Here is the **general syntax** for declaring a Two-Dimensional (2D) array:
  datatype array_Name[] [];
- Here is the general **syntax** for declaring a Three-Dimensional (3D) array:
  datatype array_name [] [] [];

Instantiating Multidimensional Arrays:

- Just like instantiating 1D arrays, you instantiate a multidimensional array using the new keyword. Here, is the **syntax for instantiating a 2D array:**
  array_name = new datatype [expl][exp2];
  where, exp1 and exp2 are expressions that evaluate to integers and specify, respectively, the number of rows and the number of columns in the array.
- Above statement allocates memory for the array. The number of elements in a two-dimensional array is equal to the sum of the number of elements in each row.
- When all the rows have the same number of columns, the number of elements in the array is equal to the number of rows multiplied by the number of columns.

Program 1.5: Program for Two-Dimensional (2D) 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
```

Program 1.6: Program for 2D array.

```
class twodim
{
    final static int r = 3;
    final static int c = 3;
    public static void main(String[] args)
    {
        int mult[] [] = new int[r][c];
        int row, column,
        int i, j;
        for (i=1; i<=r; i++)
        {
            for (j=1; j<=c; j++)
            {
                mult [i][j] = i * j;
                System.out.println(" " + mult [i][j]);
            }
        }
        System.out.println("  ");
    }
}
```

Output:

```
The given matrix:
1 2 3
2 4 6
3 6 9
```

Program 1.7: Program addition of two matrices.

```
class Testarray
{
   public static void main(String args[])
   {
      //creating two matrices
      int a[][]={{1,3,4},{3,4,5}};
      int b[][]={{1,3,4},{3,4,5}};
      //creating another matrix to store the sum of two matrices
      int c[][]=new int[2][3];
      //adding and printing addition of Two matrices
      for(int i=0;i<2;i++)
      {
         for(int j=0;j<3;j++)
         {
            c[i][j]=a[i][j]+b[i][j];
            System.out.print(c[i][j]+" ");
         }
      System.out.println();//new line
      }
   }
}
```

Output:

```
2   6   8
6   8   10
```

Program 1.8: Program for sorting elements in array.

```
import java.util.Arrays;
class SortIntArrayExample
{
   public static void main(String[] args)
   {
      //create an int array
      int[] i1 = new int[]{3,2,5,4,1};
      //print original int array
      System.out.print("Original Array: ");
      for(int index=0; index < i1.length ; index++)
      System.out.print(" " + i1[index]);
      Arrays.sort(i1);
      //print sorted int array
      System.out.print("Sorted int array: ");
      for(int index=0; index < i1.length ; index++)
```

```
        System.out.print(" " + i1[index]);
        int[] i2 = new int[]{5,2,3,1,4};
         Arrays.sort(i2,1,4);
        //print sorted int array
        System.out.print("Partially Sorted int array: ");
        for(int index=0; index < i2.length ; index++)
        System.out.print(" " + i2[index]);
    }
}
```

Output:

```
Original Array: 3 2 5 4 1
Sorted int array: 1 2 3 4 5
Partially Sorted int array: 5 1 2 3 4
```

Program 1.9: Program to reverse elements in array.

```
import java.util.*;
class ReverseArray
{
    public static void main(String args[])
    {
    // Create java.util.Scanner object for taking input
    Scanner s=new Scanner(System.in);
    // Take no.of elements and store it in n
    System.out.println("Enter the no.of elements");
    int n=s.nextInt();
    // Create array of size n
    int a[]=new int[n];
    // Read elements into the array
    System.out.println("Enter the elements into the array");
        for(int i=0;i<n;i++)
        {
            a[i]=s.nextInt();
        }
    // Reverse elements in the array
    reverse(a);
        // Print the array
        for(int i=0;i<n;i++)
        {
        System.out.printf("a[%d]=%d\n",i,a[i]);
        }
    }
    public static void reverse(int[] a)
```

```
    {
        // Loop for length/2 times only else re-swapping takes place
            for(int i=0;i<a.length/2;i++)
            {
            int temp=a[i];
            a[i]=a[(a.length-1)-i];
            a[(a.length-1)-i]=temp;
            }
        }
    }
```

Output:

```
Enter the no.of elements
4
Enter the elements into the array
1
2
3
4
a[0]=4
a[1]=3
a[2]=2
a[3]=1
```

Program 1.10: Program for transpose matrix.

```
class TransposeaMatrix
{
    public static void main(String args[])
    {
        int m, n, c, d;
        Scanner in = new Scanner(System.in);
        System.out.println("Enter the number of rows and columns of matrix:");
        m = in.nextInt();
        n = in.nextInt();
        int matrix[][] = new int[m][n];
        System.out.println("Enter the elements of matrix:");
        for (c = 0 ; c < m ; c++)
        for (d = 0 ; d < n ; d++)
        matrix[c][d] = in.nextInt();
        int transpose[][] = new int[n][m];
        for (c = 0 ; c < m ; c++)
        {
            for (d = 0 ; d < n ; d++)
```

```
            transpose[d][c] = matrix[c][d];
         }
         System.out.println("Transpose of entered matrix:-");
         for (c = 0 ; c < n ; c++)
         {
            for (d = 0 ; d < m ; d++)System.out.print(transpose[c][d]+"\t");
            System.out.print("\n");
         }
      }
   }
```

Output:

```
Enter the number of rows and columns of matrix:
2  3
Enter the elements of matrix:
1  2  3
4  5  5
Transpose of entered matrix:
1    4
2    5
3    5
```

1.8 ACCEPTING INPUT USING COMMAND LINE ARGUMENT

- Sometimes you will want to pass information into a program when you run it. This is accomplished by passing command-line arguments to main( ).
- A command-line argument is the information that directly follows the program's name on the command line when it is executed.
- A Java application can accept any number of arguments from the command line. This allows the user to specify configuration information when the application is launched.
- To access the command-line arguments inside a Java program is quite easy – they are stored as strings in a String array passed to the args parameter of main( ). The first command-line argument is stored at args[0], the second at args[1], and so on.
- The following program displays all of the command-line arguments that it is called with:

```
public class CommandLine
{
   public static void main(String args[])
   {
      for(int i = 0; i<args.length; i++)
      {
         System.out.println("args[" + i + "]: " + args[i]);
      }
   }
```

```
}
```

- Try executing this program as shown here:

```
$java CommandLine this is a command line 200 -100
```

- This will produce the following **output**:

```
args[0]: this
args[1]: is
args[2]: a
args[3]: command
args[4]: line
args[5]: 200
args[6]: -100
```

1.9 ACCEPTING INPUT FROM CONSOLE (USING BUFFEREDREADER AND SCANNER CLASS)

- There are many ways to get information from the user. In many cases, the user must be told that they should enter some information. This is known as prompting the user. The java.io package contains most of I/O classes.
- These are given below:
  1. **java.io.InputStream:** Stores information about the connection between an input device and the computer or program.
  2. **java.io.InputStreamReader:** Used to translate data bytes received from InputStream objects into a stream of characters.
  3. **java.io.BufferedReader:** Used to buffer (store) the input received from an inputStreamReader object.

BufferedReader

- BufferedReader improves performance by buffering input. The BufferedReader class does have a method called readLine that does return a line of text as type by the user.
- The contents of the stream can be read into a buffer to speed up the execution. The size of the buffer can be set explicitly, if required.
- In addition to the methods defined by the Reader class, this class has a method readLine() to read a line from the stream. It has two forms of constructors.
  1. **BufferedReader(Reader inStream):** This form creates a buffered character stream with a default buffer size.
  2. **BufferedReader(Reader inStream, int buffSize):** The size of the buffer is passed in buffSize.

Program 1.11: Program for BufferedReader.

```
import java.io.*;
public class BufferedReaderExample
{
```

```
    public static void main(String args[])throws Exception
    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        // Accepting String value
        System.out.println("Enter your name");
        String name=br.readLine();
        System.out.println("Welcome "+name);
        // Accepting integer value
        System.out.println("Enter your Roll Number");
        int roll_no=Integer.parseInt(br.readLine());
        System.out.println("Roll number is "+roll_no);
        // Accepting Float value
        System.out.println("Enter your Percentage");
        float per=Float.parse Float(br.readLine());
        System.out.println("Percentage is "+per);
    }
}
```

Scanner

- We have already learnt about Java BufferedReader to input data from user. In Java, the java.util.Scanner or Scanner class is one of them which allows the user to read values of various types in java.
- A Java Scanner is the fastest and the easiest way to get input from a user in Java than java.io.InputStream and java.io.InputStrearnReader.
- The Scanner class is a standardized class that uses different methods for reading in values from either the keyboard or a file.
- Scanner is the complement of Formatter. Added by JDK 5, Scanner reads formatted input and converts it into its binary form. Although it has always been possible to read formatted input, it required more effort than most programmers would prefer.
- Because of the addition of Scanner, it is now easy to read all types of numeric values, strings, and other types of data, whether it comes from a disk file, the keyboard, or another source.
- This next line creates a Scanner that reads from standard input, which is the keyboard by default:

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

- In general, to use Scanner, follow this procedure:
  1. Determine if a specific type of input is available by calling one of Scanner's hasNextX methods, where X is the type of data desired.
  2. If input is available, read it by calling one of Scanner's nextX methods.
  3. Repeat the process until input is exhausted.

Program 1.12: Use Scanner to compute an average of the values.

```
import java.util.*;
class AvgNums
{
   public static void main(String args[])
   {
      Scanner conin = new Scanner(System.in);
      int count = 0;
      double sum = 0.0;
      System.out.println("Enter numbers to average.");
      // Read and sum numbers.
      while(conin.hasNext())
      {
         if(conin.hasNextDouble())
         {
            sum += conin.nextDouble();
            count++;
         }
         else
         {
            String str = conin.next();
            if(str.equals("done")) break;
            else
            {
               System.out.println("Data format error.");
               return;
            }
         }
      }
      System.out.println("Average is " + sum / count);
   }
}
```

Output:

```
Enter numbers to average.
1
2
3
4
5
done
Average is 3.25
```

- Java programming language provides if, if...else, switch decision making statements. Java programming language provides for, while, do...while loops to handle looping requirements. Java supports break and continue loop control statements.

1.10 CLASSES AND OBJECTS

Overview of Class and Object:

- In Object-Oriented Programming (OOP) technique, we design a program using objects and classes. Object is the physical as well as logical entity whereas class is the logical entity only.

1. **Class in Java:**
   - Classes in Java provide a convenient method for packing together a group of logically related data items and functions that work on them.
   - In Java language, the data items are called fields and the functions are called methods. Calling a specific method in an object is described as sending the object of message.
   - A class can be defined as, "a template/blue print that describes the behaviors/ states that object of its type support".
2. **Object in Java:**
   - A real-world entity that has state and behavior is known as an object.
   - These real-world objects share two characteristics i.e. state and behavior.
   - **Example:** A cat has states - color, name, etc. as well as behaviors like walking and eating.
   - In simple words, object is an instance of a class.
   - Fig. 1.8 shows a real world example of class and objects. Fig. 1.8 shows class Car has objects like Audi, Nissan, Volvo.

Fig. 1.8: Example of Class and Object

Difference between Class and Object:

- There are many differences between object and class. A list of differences between object and class are given below:

Sr. No.	Object	Class
1.	Object is an instance of a class.	Class is a blueprint or template from which objects are created.
2.	Object is a real world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.	Class is a group of similar objects.

3.	Object is a physical entity.	Class is a logical entity.
4.	Mainly Object is created through new keyword like Student s1=new Student();	Class is declared using class keyword like class Student{ ... }
5.	Object is created many times as per requirement.	Class is declared once.
6.	Object allocates memory when it is created.	Class doesn't allocated memory when it is created.

1.11 DEFINING YOUR OWN CLASSES

- A class is declared by use of the class keyword. The classes that have been used up to this point are actually very limited examples of its complete form. Classes can (and usually do) get much more complex.
- A simplified general **form/syntax** of a class definition is shown here:

```
class classname
{
   type instance-variable1;
   type instance-variable2;
   type instance-variableN;
   type methodname1(parameter-list)
   {
      // body of method
   }
   type methodname2(parameter-list)
   {
      // body of method
   }
   // ...
   type methodnameN(parameter-list)
   {
      // body of method
   }
}
```

- **Example of Class:**

```
Class Employee
{
   int EmpID:
   float salary;
}
```

Methods in Java:

- A Java method is a collection of statements that are grouped together to perform a certain task.
- Method describes behaviour of an object. A method is a collection of statements that are group together to perform an operation.

Syntax:

```
return_type methodName(parameter_list)
{
   //body of method 
}
```

- **Example of a Method:**

```
public String getName(String st)
{
   String name="Kamil Ajmal Khan ";
   name=name+st;
   return name;
}
```

- **For example:**

```
class Box
{
   double width;
   double height;
   double depth;
   }
// This class declares an object of type Box.
class BoxDemo
{
   public static void main(String args[])
   {
      Box mybox = new Box();
      double vol;
      // assign values to mybox's instance variables
      mybox.width = 10;
      mybox.height = 20;
      mybox.depth = 15;
      // compute volume of box
      vol = mybox.width * mybox.height * mybox.depth;
      System.out.println("Volume is " + vol);
   }
}
```

Output:

```
Volume is 3000.0
```

Program 1.13: Program for class.

```
public class JavaClassExample
{
   public static void main(String[] arg)
   {
      for (int i = 0; i<4; i++)
      {
         System.out.println("Number is: "+i);
      }
   }
}
```

Output:

```
Number is: 0
Number is: 1
Number is: 2
Number is: 3
```

Declaring Objects:

- When you create a class, you are creating a new data type. You can use this type to declare objects of that type. However, obtaining objects of a class is a two-step process.

(i) Declaration:

- We have to specify what type (i.e. class) the object will be. A variable declaration with a variable name with an object type.

  Syntax: `<class name> object name;`

  Where, class name is the name of the already defined class and the object name is a valid identifier.

(ii) Instantiation (Creating Objects):

- Objects are created using the 'new' keyword. This 'new' keyword creates an object of the specified class and returns the reference of that object.
- The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

  Syntax: `<objectname>=new classname([arguments]);`
- Let us create an object for the already defined Box class in the above program segment.

  ```
  Box mybox = new Box();
  ```
- This statement combines the two steps just described. It can be rewritten like this to show each step more clearly:

  ```
  Box mybox; // declare reference to object
  mybox = new Box(); // allocate a Box object
  ```

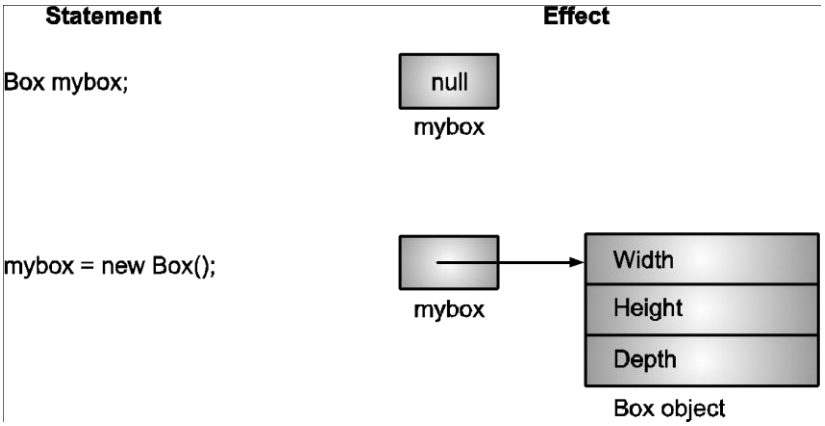

Fig. 1.9

(iii) Accessing Class Members:

- The dot operator (.) or member selection operator is used to access the instance variables of the class.
- We cannot access the instance variables outside of the class without referring to the object.
- Member access follows the following **syntax**:

  ```
  objectname.variablename
  ```

  Here, the 'objectname' is the name of the object and 'variablename' is name of the instance variable.
- So, the instance variables of the class Employee can be accessed as,

  ```
  Employye.E_id=1234;
  Employee.salary=30000;
  ```

Program 1.14: Program to calculate the specifications of the circle.

```
import java.io.*;
class Circle
{
   int radius;
   float perimeter;
   float area;
}
class MyCircle
{
   public static void main(String args[])
   {
      final float pi = 3.14f;
      Circle c = new Circle();
      BufferedReadaer in = new BufferedReader
                                        (new InputStreamReader(System.in));
      System.out.print("Enter radius: ");
```

```
        try
        {
           c.radius = Integer.parseInt(in.readLine());
           c.perimeter = 2.0f * pi * (float) c.radius;
        }
        catch (IOException ioe)
        {
        ioe.printstackTrace();
        }
        c.area = pi * (float) (c.radius * c.radius);
        System.out.println("Perimeter: "+c.perimeter);
        System.out.println("Area: "+c.area);
     }
  }
```

Output:

```
Enter radius: 23
Perimeter: 144.44
Area: 1661.06
```

Create Your Own Classes:

- To use Java effectively, you want to create and use your own classes. This is one of the great powers of Object-Oriented Languages, the ability to construct programs out of independent building-blocks that cut large problems down into small, easily solvable ones.
- We are going to create a very simple object class that will print a "Hello" message to the screen when called.
- This class will have two different methods that will do the same thing, though we will vary the messages they print a bit so that we can see which one is doing the work.

```
public class HelloClass
{
   /** sayHello() prints "Hello!" to the Java console output
   */
   public void sayHello()
   {
      System.out.println("Hello!\n");
   }
   /**
   * doHello() prints "Hello, hello!" to the Java console output
   * It's static, so you don't need to instatiate a HelloClass
   * object to use it
   */
```

```
    public static void doHello()
    {
        System.out.println("Hello, hello!\n");
    }
} // End of HelloClass
```

- The two methods we have to print messages are sayHello() and doHello(). To use sayHello(), we need to have a HelloClass object created, then call that object's sayHello()method.
- doHello(), however, is a static method. This means is belongs to the class, not to any object of the class. So we can use it without any HelloClass objects being created first.

1.12 ACCESS SPECIFIERS (PUBLIC, PROTECTED, PRIVATE, DEFAULT)

- In Java, it may be necessary in some situations to restrict the access to certain variables and methods from outside the class. For example, we may not like the objects of a class directly alter the value of a variable or access a method. We can achieve this in Java by applying visibility modifiers. The visibility modifiers are also known as access modifiers.
- Java provides a number of access modifiers to set access (scope) levels for classes, variables, methods and constructors. The four access levels are:
  1. Visible to the package (default/friendly),
  2. Visible to the class only (private),
  3. Visible to the world (public), and
  4. Visible to the package and all subclasses (protected).
- Let us see above access specifiers in detail.

1. Default / Friendly Access Modifier (No Keyword):

- If we give no specifier at all, the access is referred as friendly. It means that all the other classes in the current package have access to the friendly member but to all the classes outside the package the members appears to be private.
- Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.
- A variable or method declared without any access control modifier is available to any other class in the same package.
- The fields in an interface are implicitly public static final and the methods in an interface are by default public.
- Variables and methods can be declared without any modifiers, as in the following examples:

```
String version = "1.5.1";
boolean processOrder()
{
    return true;
```

```
}
```

2. Private Access Modifier (private):

- Methods, variables and constructors that are declared private can only be accessed within the declared class itself.
- Private access modifier is the most highest degree restrictive access level. Class and interfaces cannot be private.
- Variables that are declared private can be accessed outside the class if public getter methods are present in the class.
- Using the private modifier is the main way that an object encapsulates itself and hide data from the outside world.
- The following class uses private access control:

```
public class PLog
{
   private String format;
   public String getFormat()
   {
      return this.format;
   }
   public void setFormat(String format)
   {
      this.format = format;
   }
}
```

- Here, the format variable of the PLog class is private, so there's no way for other classes to retrieve or set its value directly.
- So to make this variable available to the outside world, we defined two public methods i.e., getFormat(), which returns the value of format, and setFormat(String), which sets its value.

Private Protected Access Modifier:

- A field can be declared with two keywords private and protected together such as,

```
private protected int RollNumber;
```

- This modifier gives a visibility level in between the "protected" access and "private" access.
- This modifier makes the fields visible in the subclasses regardless of what package they are in. Remember that these fields are not accessible by other classes in the same package.

3. **Public Access Modifier (public):**

- A class, method, constructor, interface etc. declared public has the widest possible visibility and accessible everywhere. Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.
- However, if the public class we are trying to access is in a different package, then the public class still need to be imported.
- Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.
- The following function uses public access control:

```
public static void main(String[] arguments)
{
   // ...
}
```

- The main() method of an application has to be public. Otherwise, it could not be called by a Java interpreter (such as java) to run the class.

4. **Protected Access Modifier (protected):**

- In Java, the visibility level of a "protected" field lies in between the public access and friendly access.
- Variables, methods and constructors which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.
- The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.
- Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.
- The following parent class uses protected access control, to allow its child class override openSpeaker() method:

```
class AudioPlayer
{
   protected boolean openSpeaker(Speaker sp)
   {
      // implementation details □
   }
}
class StreamingAudioPlayer
{
   boolean openSpeaker(Speaker sp)
```

```
    {
        // implementation details
    }
}
```

- In above program code, if we define openSpeaker() method as private, then it would not be accessible from any other class other than AudioPlayer. If we define it as public, then it would become accessible to all the outside world. But our intension is to expose this method to its subclass only, that's why we used protected modifier.

- Following table summarises visibility of field in a class of Java.

Access Location	Public modifier	Protected modifier	Friendly (default) modifier	Private protected modifier	Private modifier
Same class	Yes	Yes	Yes	Yes	Yes
Subclass in same package	Yes	Yes	Yes	Yes	No
Other classes in same package	Yes	Yes	Yes	No	No
Subclass in other packages	Yes	Yes	No	Yes	No
Non-subclasses in other packages	Yes	No	No	No	No

Program 1.15: Program for access modifier.

```
class Rectangle
{
    public int length;            //public Access
    public int breadth;           //public Access
    static int rectCount = 0;     //default Access
    //Constructor to initialize length and breadth Rectangle()
    {
        rectCount++;
    }  //method to calculate area of rectangle
    public int area()
    {
        int rectArea;
        rectArea = length * breadth;
        return rectArea;
    }
}
class AccessSpecifer
{
    public static void main(String[] args)
    {
        //create first rectangle object
        Rectangle firstRect = new Rectangle();
```

```
        //accessing public members outside that Rectangle class
        firstRect.length = 10;
        firstRect.breadth = -30;
        System.out.println("Area of Rectangle is: "+firstRect.area());
        //Rectangle.rectCount = 5;
        //accessing member with default access
        System.out.println("Number of Object Created"+Rectangle.rectCount);
    }
}
```

Output:

```
Area of Rectangle is : – 300
Number of Object Created : 1
```

1.13 ARRAY OF OBJECTS

- In Java, rather than creating object of a particular or class, we can also create array of objects.
- In Java, an array of objects can also be created to store and process the objects. We can also create array of obj. The process to declare and create an array of objects is similar to the process of declaring and creating an array of simple data types.
- It is possible to create array of objects of user created/defined class.
- When we create an array of objects then the data type get replaced with the class name. The following is the **syntax** to declare and create an array of objects:

  ```
  class_name arr_name [ ] = new class_name [size];
  ```

  where,
  - class_name is the name of the class whose objects are used to create an array of objects.
  - arr_name is the name of the array, which is to be created.
  - new is the Java operator, which is used to allocate memory to an array.
  - size is the number of objects that an array can hold.
- Let's now suppose that we have a class, ArrayDemo. We have to create an array of objects of this class. To create an array of objects to ArrayDemo class see the following example:

  ```
  ArrayDemo a_obj[ ] = new ArrayDemo [2]:
  ```
- The a_obj array can hold 2 objects of ArrayDemo class. To use the a_obj array, we have to create objects of every array index.
- See the following code snippet to create objects of every index of a_obj array:

  ```
  a_obj [0] = new ArrayDemo();
  a_obj [1] = new ArrayDemo();
  ```
- We can use loop to create array of objects as follows:

  ```
  ArrayDemo[] arr = new ArrayDemo[10];
  ```

```
    for(int i = 0; i = 10; i++)
    arr[i] = new ArrayDemo();
```

Program 1.16: Program for array of object.

```
class Employee
{
    int id;
    String name;
    public void setData(int id, String name)
    {
        this.id = id;
        this.name = name;
    }
    public void displayData()
    {
        System.out.println("Employee ID : "+this.id);
        System.out.println("Employee Name : "+this.name);
    }
}
class Main
{
    public static void main(String args[])    //OR
    {      //Employee[] emp=new Employee[2];
        Employee[] emp = new Employee[2];      //for(int i = 0; i < 2; i++)
        emp[0].setData(1,"Amar");              //emp[i] = new Employee()
        emp[1].setData(2,"Akbar");             //emp[i].displayData();
        emp[0].displayData();
        emp[1].displayData();
    }
}
```

Output:

```
Employee ID : 1
Employee Name : Amar
Employee ID : 2
Employee Name : Akbar
```

1.14 CONSTRUCTOR

- A constructor is a special method of a class in Java programming that initializes an object of that type.
- Constructors have the same name as the class itself. A constructor is automatically called when an object is created.

- Constructor constructs the values i.e. provides data for the object that is why it is known as constructor.

  Syntax:

```
class_Name
{
   Constructor_Name (same as class_Name); // Constructor
   {
      //constructor body…..
   }// end of Constructor
}// end of Class
```

- There are basically following rules defined for the constructor:
  1. Constructor name must be same as its class name.
  2. Constructor must have no explicit return type.
  3. Constructors may be private, protected or public.
  4. Multiple constructors may exist, but they must have different signatures, i.e., different numbers and/or types of input parameters.

Program 1.17: Program for constructors.

```
class Rectangle
{
int length;
int breadth;
//constructor to initialize length and breadth of rectangle
Rectangle()
{
   length = 5;
   breadth= 6;
}
//method to calculate area of rectangle
int area()
{
   int rectArea = length * breadth;
   return rectArea ;
}
}
//class to create rectangle objects and calculates area
class ConstructorExample
{
   public static void main(String[] args)
   {
      Rectangle firstRect = new Rectangle();
```

```
        System.out.println("Area of Rectantangle: "+ firstRect.area());
    }
}
```

Output:

```
Area of rectangle: 30
```

- Constructors can be classified into two types, Default Constructors and Parametarized Constructors as shown in Fig. 2.3.

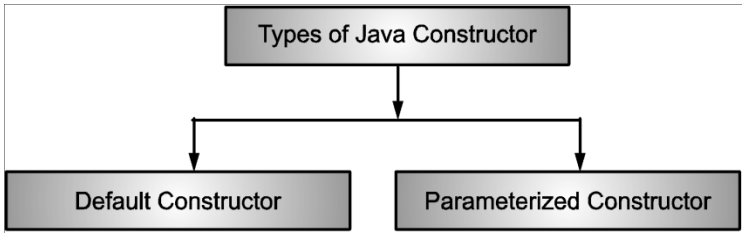

Fig. 1.10: Types of Constructor

1. Default Constructor:

- The constructors which does not accept any argument is called Default Constructor. The argument list is void.
- In other word, when the object is created Java creates a no-argument constructor automatically known as Default Constructor.
- It does not contain any parameters nor does it contain any statements in its body. Its only purpose is to enable you to create object of class type.
- Default constructor provides the default values to the object like 0, null etc. depending on the type.

  Syntax: `<class_name>(){}`

Program 1.18: Program for default constructor.

```
class StudentInfo
{
    int rollno;
    int marks1;
    int marks2;
    int total;
        StudentInfo()    //no-arg default constructor
        {
            rollno=2;
            marks1=30;
            marks2=40;
        }
        //method to calculate total
        int Total()
```

```
        {
            total = marks1 + marks2;
            return total;
        }
        //method to display the result
        void displayResult()
        {
            System.out.println("Roll no of student is" + rollno );
            System.out.println("marks1 are" + marks1 );
            System.out.println("marks2 are" + marks2);
            System.out.println("total is" + total);
        }
    }
    class StudentResult
    {
    public static void main(String args[])
    {
        int total1,total2,grandtotal;
        StudentInfo student1 =new StudentInfo();       //creates first object
        StudentInfo student2 =new StudentInfo();      //creates second object
        total1=student1.Total();
        total2=student2.Total();
        grandtotal=total1+total2;
        student1.displayResult();
        student2.displayResult();
        System.out.println("Grand Total is" + grandtotal);
        }
    }
```

Output:

```
Roll no of student is 2
marks1 are 30
marks2 are 40
total is 70
Roll no of student is 2
marks1 are 30
marks2 are 40
total is 70
Grand Total is 140
```

---

2. Parameterized Constructor:

- Constructor can take value, value is called as argument. A constructor that has parameters is known as Parameterized Constructor. Parameterized constructor is used to provide different values to the distinct objects.
- Using parameterized constructor, it is possible to initialize objects with different set of values at the time of their creation. These different set of values initialized to objects must be passed as arguments when constructor is invoked.
- The parameter list can be specified in the parentheses in the same way as parameter list is specified in the method.
- The **syntax** for constructor is as follows:

```
ConstructorName([parameterList])
{
//constructor body 
}
```

- Here, the ConstructorName is same as the class name it belongs to. The parameterList is the list of optional zero or more parameter(s) that is specified after the classname in parentheses. Each parameter specification, if any, consists of a type and a name and is separated from each other by commas.

Program 1.19: Program for parametrized constructor.

```
class StudentInfo
{
   int rollno;
   int marks1;
   int marks2;
   int total;
   StudentInfo(int roll_no,int m1,int m2)  //Parameterized constructor
   {
      rollno=roll_no;
      marks1=m1;
      marks2=m2;
   }
   //method to calculate total
   int Total()  //Method declaration
   {
      total = marks1 + marks2;
      return total;
   }
   //method to display the result
   void displayResult()
   {
      System.out.println("Roll no of student is" + rollno );
```

```
        System.out.println("marks1 are" + marks1 );
        System.out.println("marks2 are" + marks2);
        System.out.println("total is" + total);
    }
}
class StudentResult
{
    public static void main(String args[])
    {
        int total1,total2,grandtotal;
        StudentInfo student1 = new StudentInfo(1,50,80);
        StudentInfo student2 = new StudentInfo(2,40,60);
        total1=student1.Total();
        total2=student2.Total();
        grandtotal=total1+total2;
        student1.displayResult();
        student2.displayResult();
        System.out.println("Grand Total is" + grandtotal);
    }
}
```

Output:

```
Roll no of student is 1
marks1 are 50
marks2 are 80
total is 130
Roll no of student is 2
marks1 are 40
marks2 are 60
total is 100
Grand Total is 230
```

Overloading Constructors

- Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.
- The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.
- Constructors having the same name with different parameter list is called as Constructor overloading.

Program 1.20: Program for constructor overloading.

```
class Student
{
```

```
    int id;
    String name;
    int age;
    Student(int i,String n)
    {
       id = i;
       name = n;
    }
    Student(int i,String n,int a)
    {
       id = i;
       name = n;
       age=a;
    }
    void display(){System.out.println(id+" "+name+" "+age);
}
public static void main(String args[])
{
    Student s1 = new Student(1,"Amar", 22);
    Student s2 = new Student(2,"Amol",20);
    s1.display();
    s2.display();
}
```

Output:

```
1 Kamil 22
2 Prajakta 20
```

Methods Overloading:

- In Java, we can define number of methods in a class with the same name. Defining two or more methods with the same name in a class is called Method Overloading.
- The overloaded constructors are called here as method overloading because we are allowed to create methods with same name, but different parameters can be passed with different definitions. This is referred as Polymorphism.
- When we call a method with its object, Java first checks for the method name and then for the number of parameters and its types. Then it decides which method should be executed.

Program 1.21: Program for method overloading.

```
class overload
{
    int m, n;
    overload()                  //default constructor
```

```
    {
        m = 5;
        n = 28;
    }
    overload (int p, int q)      //parameterised constructor
    {
        m = p;
        n = q;
    }
    overload (double x, double y)
    {
        m = x;
        n = y;
    }
    void display()
    {
        System.out.println(m);
        System.out.println(n);
    }
}
class mainover
{
    public static void main(String args[])
    {
        overload ob1 = new overload();
        overload ob2 = new overload(10, 5);
        overload ob3 = new overload(25.6, 80.5);
        ob1.display();
        ob2.display();
        ob3.display();
    }
}
```

Output:

```
for ob1
    for ob2
for ob3
```

- Here, for object ob1, the default constructor is called. For object ob2 the parameterised constructor "overload (int p, int q)" is called, whereas for object ob3, the parameterised constructor "overload (double x, double y)" is called. But because of the data type of m and n are int, the value of x and y i.e. 25.6 and 80.5 get truncated to integer part.

Difference between Constructor and Method:

Sr. No.	Constructor	Method
1.	Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.
2.	Constructor name must be same as the class name.	Method name may or may not be same as class name.
3.	Constructor is invoked implicitly.	Method is invoked explicitly.
4.	The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.
5.	Constructor must not have return type.	Method must have return type.

Use of 'this' Keyword

- Many times it is necessary to refer to its own object in a Method or a Constructor. To allow this Java defines the 'this' keyword.
- As we know in C++, sometimes a method will need to refer to the object that invoked it. To do so, Java also has a 'this' keyword. 'this' can be used inside any method to refer to the current object. It means that 'this' is always a reference to the object on which the method was invoked.
- The 'this' is used inside the method or constructor to refer its own object i.e., 'this' is always a reference to the object of the current class' type.

Usage of this Keyword:

1. 'this' keyword can be used to refer current class instance variable.
2. 'this' keyword can be used to invoke current class constructor.
3. 'this' keyword can be used to invoke current class method (implicitly).
4. 'this' can be passed as an argument in the method call.

Syntax: `this.field`

- Fig. 1.11 shows use of 'this' keyword.

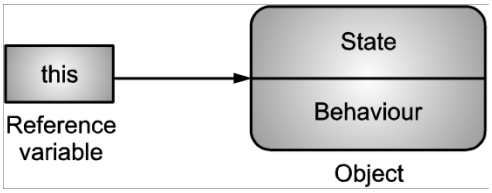

Fig. 1.11: this keyword

- **Example:**

```
class BoxDim
{
   int height;
   int depth;
   int length;
```

```
    BoxDim(int height, int depth, int length)
    {
        this.height = height;
        this.depth = depth;
        this.length = length;
    }
}
```

- The class 'BoxDim' contains three instance variables i.e. height, depth and length. The constructor of the class also contains three different local variables with the same names of instance variables.
- Compiler will not show any error here. Then how differentiate among these variables? The 'this' keyword does this job. 'this' will refer to the variables of its own class. It is acting as the object of the current class.
- In method also the 'this' keyword is used to differentiate between instance and local variables. This concept is also referred as 'Instance variable hiding'. This resolves the name-space collisions.

Program 1.22: Program for 'this' keyword.

```
class Rectangle
{
    int length,breadth;
    void show(int length,int breadth)
    //Formal and instance variable name are same
    {
        this.length=length;    //Use of this keyword
        this.breadth=breadth;
    }
    int calculate()
    {
    return(length*breadth);
    }
}
/* Main class */
public class UseOfThisOperator
{
    public static void main(String[] args)
    {
        Rectangle rectangle=new Rectangle();
        rectangle.show(8,6);
        int area = rectangle.calculate();
        System.out.println("The area of a Rectangle is: " + area);
```

```
    }
}
```

Output:

```
The area of a Rectangle is: 48
```

1.15 STATIC BLOCK, STATIC FIELDS AND METHODS

Static Block

- Java supports a special block, called static block (also called static clause) which can be used for static initializations of a class. This code inside static block is executed only once: the first time the class is loaded into memory. For example, check output of following Java program.

  Example: Program use of static block

```
class Test
{
    static int i;
    int j;    // start of static block
    static
    {
        i = 10;
        System.out.println("static block called ");
    }
}
class Main
{
    public static void main(String args[])
    {
        System.out.println(Test.i);
    }
}
```

  Output:

```
static block called
10
```

Static Fields & Static Methods

- The static keyword is used to create methods that will exist independently of any instances created for the class.
- Static methods do not use any instance variables of any object of the class they are defined in. Static methods take all the data from parameters and compute something from those parameters, with no reference to variables.

Program 1.23: Program use of static field and methods.

```
public class CountInstance
{
   private static int numInstances = 0;
   protected static int getCount()
   {
      return numInstances;
   }
   private static void addInstance()
   {
      numInstances++;
   }
   CountInstance()
   {
      CountInstance.addInstance();
   }
   public static void main(String[] arguments)
   {
      System.out.println("Starting with " + CountInstance.getCount()
                                                          + " instances");
      for (int i = 0; i < 500; ++i)
      {
         new CountInstance();
      }
      System.out.println("Created " + CountInstance.getCount()
                                                          + " instances");
   }
}
```

Output:

```
Starting with 0 instances
Created 500 instances
```

1.16 PREDEFINED CLASSE - Object Class Methods (equals(), toString(), hashcode())

- Java provides different types of predefined classes that programmers can use to build sophisticated programs. If you have a choice of using a predefined standard class or devising your own custom class that provides the same functionality, choose the standard class.
- The advantages of using a standard class include:
  1. The effort to produce the class is eliminated entirely; you can devote more effort to other parts of the application's development.

2. Java compiler automatically imports all the classes in the java.lang package into every source file.

- Some of the most important classes of the java.lang package:
  - Object
  - Math
  - The wrapper classes
  - String
  - StringBuffer

Object Class:

- The object class is the parent class of all the classes in Java by default. In other words, it is the topmost class of java.
- The java.lang.Object class is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.
- Following is the declaration for java.lang.Object class:

```
public class Object
```

- The Object class provides some common behaviours to all the objects such as object can be compared, object can be cloned, object can be notified etc.

Methods of Object Class:

- We already known object class is the superclass of all classes. Some methods of the object class are:
  - **protected Object clone():** Creates and returns a copy of this object.
  - **boolean equals(Object obj):** Indicates whether some other object is "equal to" this one.
  - **protected void finalize():** This method is called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
  - **Class getClass():** Returns the runtime class of this Object.
  - **int hashCode():** Returns a hash code value for the object.
  - **void notify():** Wakes up a single thread that is waiting on this object's monitor.
  - **void notifyAll():** Wakes up all threads that are waiting on this object's monitor.
  - **String toString():** Returns a string representation of the object.
  - **void wait():** Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.
  - **void wait(long timeout):** Causes the current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
  - **void wait(long timeout, int nanos):** Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

Program 1.24: Program for object class.

```
class Rectangle extends Object
{
   private double length,breadth;
   Rectangle(double x,double y)
   {
      length = x ;
      breadth = y ;
   }
   public void area()
   {
      System.out.println("Area of Rectangle is = " + breadth);
   }
   public void circumference()
   {
      System.out.println("Circumference of
                                        Rectangle is="+2*(length+breadth));
   }
}
class ObjectClass
{
   public static void main(String[] args)
   {
      Rectangle r = new Rectangle(10,20);
      Rectangle rl = new Rectangle(10,20);
      System.out.println("String Representation = " + r.toString());
      System.out.println("Class Name = " + r.getClass());
      System.out.println("Hash Code = " + r.hashCode());
      System.out.println("r.equals(rl) = " + r.equals(rl));
   }
   }
```

Output:

```
String Representation = Rectangle@76ed5528
Class Name = class Rectangle
Hash Code = 1995265320
r.equals(rl) = false
```

1.17 GARBAGE COLLECTION (FINALIZE() METHOD)

- We know that constructors are used to initialise an object when it is declared. This process is referred as initialisation. Java runtime is a automatic garbage collecting system.

- In C++, if we create objects dynamically with new operator then the memory allocated for these objects gets deallocated manually; just by using delete operator.
- In Java, the deallocation is handled automatically. The technique which deallocates memory [assigned to objects] automatically is called garbage collection.
- In Java, garbage means unreferenced objects. Garbage collector is automatically invoked when the program is being run.
- Garbage collector can be called by calling gc() method of runtime class. There are many ways, in which an object can be unreferenced:
  1. By nulling the reference,
  2. By assigning a reference to another, and
  3. By annonymous object etc.
- Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.
- In Java, it is performed automatically. So, Java provides better memory management.

Advantage of Garbage Collection:

1. It is automatically done by the garbage collector(a part of JVM) so we don't need to make extra efforts.
2. It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.

finalize() Method:

- When the reference of an object is not exists, then it is assumed that object is no longer needed. And the memory which is occupied by the object is reclaimed. So garbage collector automatically frees the memory resources used by the objects.
- But sometimes the object may hold some non-java resources such as file handle or window character font. The garbage collector cannot free these resources.
- In order to free these resources we need finalize() method. This is called finalization. By using the finalization, we can define specific action that will occur when an object being destroyed by the garbage collector.
- To add a finalizer to our class, we have to define it as a finalize() method.
- The finalize() method has following general **form/Syntax**:

```
protected void finalize()
{
   // finalization code here 
}
```

Here, the keyword **protected** is a specifier that prevents access to **finalize()** by code defined outside its class.

- The garbage collector runs periodically. It checks for objects which are no longer referenced by any running state or indirectly through any running object. Before it gets free, the java run-time calls the finalize() method.
- Protected keyword prevents the access of finalize() code from outside class. Java does not support destructors.

Program 1.25: Following example shows that the objects allocated external resources should provide a finalize method that cleans them up or class will create a resource leak.

```
import java.util.*;
public class ObjectfinDemo extends GregorianCalendar
{
   public static void main(String[] args)
   {
      try
      {
         // create a new ObjectfinDemo object
         ObjectfinDemo cal = new ObjectfinDemo();
         // print current time
         System.out.println("" + cal.getTime());
         // finalize cal
         System.out.println("Finalizing...");
         cal.finalize();
         System.out.println("Finalized.");
      } catch (Throwable ex)
      {
         ex.printStackTrace();
      }
   }
}
```

Output:

```
Sat Sep 22 00:27:21 EEST 2014
Finalizing...
Finalized.
```

[redacted]

- James Gosling is called Father of Java Programming. Java is an Object Oriented Programming language means Java enables us not only to organize our program code into logical unit called objects but also to take advantage of encapsulation, inheritance and polymorphism and so on.
- Various features of java includes Simple, Secure, Portable, Object-oriented, Robust, Multithreaded, Architecture-neutral, Interpreted, High Performance, Distributed, Dynamic.

- Java Virtual Machine (JVM) is a virtual Machine that provides runtime environment to execute Java byte code.
- Comments are non-executable statements and are ignored by the Java compiler.
- The comments increases the readability of the programs.
- An array is a group of like-typed variables that are referred to by a common name. Arrays of any type can be created and may have one or more dimensions.
- Array is a data structure where we store similar elements. We can store only fixed set of elements in a Java array. Types of array-1D array, 2D array.
- A one-dimensional array having only one subscript. In Java 2D array is a matrix of rows and columns.
- A Java program is basically a collection of classes. A class is defined by a set of declaration statements and methods containing executable statements.
- Object is the physical as well as logical entity whereas class is the logical entity only.
- A constructor is a special method of a class in Java programming that initializes an object of that type. Constructors have the same name as the class itself. A constructor is automatically called when an object is created.
- We know that constructors are used to initialise an object when it is declared. This process is referred as initialisation. Java runtime is a automatic garbage collecting system.

[redacted]

1. What allows the programmer to destroy an object x?
   (a) x.delete()
   (b) x.finalize()
   (c) Runtime.getRuntime().gc()
   (d) Only the garbage collection system can destroy an object.
2. What is the stored in the object obj in following lines of code?
   box obj;
   (a) Memory address of allocated memory of object
   (b) NULL
   (c) Any arbitrary pointer
   (d) Garbage
3. Which of these operators is used to allocate memory for an object?
   (a) malloc (b) alloc
   (c) new (d) give
4. What is the output of this program?

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

```
    {
        int x = 9;
        if (x == 9)
        {
            int x = 8;
            System.out.println(x);
        }
    }
}
```

(a) 9 (b) 8
(c) Compilation error (d) Runtime error

5. 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
6. Which three are legal array declarations?
   int [] myScores [];
   char [] myChars;
   int [6] myScores;
   Dog myDogs [];
   Dog myDogs [7];
   (a) 1, 2, 4 (b) 2, 4, 5
   (c) 2, 3, 4 (d) All are correct
7. Which one of the following will declare an array and initialize it with five numbers?
   (a) Array a = new Array(5); (b) int [] a = {23,22,21,20,19};
   (c) int a [] = new int[5]; (d) int [5] array;

Answers

1. (d)	2. (b)	3. (c)	4. (c)	5. (a)	6. (a)	7. (b)

[redacted]

Q.I Answer the following Questions in short:

1. What is Java?
2. Why Java is platform-neutral language?
3. Explain atleast five features of Java.
4. Explain the secure feature of Java.
5. Why Java needs compiler and interpreter?
6. Why Java is called portable?
7. Which are the tasks performed by Java environment?
8. How to compile Java program?
9. How to run Java program?
10. How Garbage collection is done in java?
11. How many types of access specifiers are provided by java?

12. Explain 'this' keyword with example.
13. What is array?
14. How a comment is added in java program?
15. What are the types of array?

Q.II Long Answers the following Questions:

1. Explain the significance of each of the following:
   (i) finalize() (ii) public
   (iii) protected (iv) private
2. Discuss garbage collection in Java.
3. What are objects? How to create objects?
4. What is a constructor? How many types of constructors are present in Java?
5. Explain static block.
6. Explain static fields and methods.
7. Explain various types of access modifiers in detail.
8. Create a class circle, consider pi and radius. Calculate the area of a circle and display it.
9. Write a program to find the prime numbers.
10. Write a program to generate a fibonacci series with do-while loop.
11. Explain Java environment in detail.
12. Write short note on: JVM,Compiler, Interpriter, constructor, garbage collections, predefined classes, array of objects, static fields and methods.
13. Explain accepting input from console (Using BufferedReader and Scanner class).
14. Explain accepting input using Command line argument.
15. Predict the output:

```
Public class Test
{
   Public static void main (string[ ] args)
   {
      boolean [ ] [ ] x = new boolean [3] [ ];
      x [0] = new boolean [1];
      x [1] = new boolean [2];
      x [2] = new boolean [3];
      System. out. println ("x[2] [2] is '' + x [2] [2]) ;
   }
}
```

Ans. `x[2] [2] is false`

16.
```
Class Test
{
   Public static void main (string [ ] args)
   {
      int [ ] x = { 1, 2, 3, 4};
```

```
        int [ ] y = x ;
        x = new int [2];
        for (int i = 0; i < y.length; i ++)
        System. out. println (y [i]);
    }
}
```

Ans. 1
2
3
4

Q.III Define the terms:

1. this keyword
2. Compiler
3. interpriter
4. Array
5. class
6. object
7. Constructor
8. garbage collections

---

1. Which is not valid keyword in Java?
   (i) Final (ii) Native (iii) Double (iv) This
2. "Import statement is not essential in Java." Justify True/False.
3. Create a class Employee with data member as name and salary. Accept the information of 5 employees and display it (use arrayof objects).
4. Explain any three features of Java.
5. What is a nested class? Explain static nested class and non-static nested class with suitable example.
6. Explain the various ways used to accept input from console in Java with proper syntax and example.

---

1. When does method overloading determine?
   (i) At a run time (ii) At a compile time
   (iii) At a coding time (iv) At an execution time
2. What is the output of relational operator?
   (i) Integer (ii) Boolean
   (iii) Char (iv) Double
3. What is constructor?
4. Explain constructor and destructor with a suitable example.

5. Create student class having data member (roll_no, name, percentage) accept values and display details (use commandline argument).