INTRODUCTION ON JAVA

HISTORY OF JAVA

James Gosling, **Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.

Java is an island of Indonesia where the first coffee was produced (called java coffee). Java name was chosen by James Gosling while having coffee near his office.

What is Java?

Java is a **programming language** and a **platform**.Java is a high level, robust, object-oriented and secure programming language.

Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

Application

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in, javatpoint.com, etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc

Types of Java Applications

There are mainly 4 types of applications that can be created using Java programming:

1) Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications, etc. is called enterprise application. It has advantages of the high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4) Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

Java Platforms / Editions

There are 4 platforms or editions of Java:

1) Java SE (Java Standard Edition)

It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

2) Java EE (Java Enterprise Edition)

It is an enterprise platform which is mainly used to develop web and enterprise applications. It is built on the top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.

3) Java ME (Java Micro Edition)

It is a micro platform which is mainly used to develop mobile applications.

4) JavaFX

It is used to develop rich internet applications. It uses a light-weight user interface API.

Java Version History

Many java versions have been released till now. The current stable release of Java is Java SE 10.

Java SE Version	Version Number	Release Date
JDK 1.0(Oak)	1.0	January 1996
JDK 1.1	1.1	February 1997
J2SE 1.2(Playground)	1.2	December 1998
J2SE 1.3(Kestrel)	1.3	May 2000
J2SE 1.4(Merlin)	1.4	February 2002
J2SE 5.0(Tiger)	1.5	September 2004
Java SE 6(Mustang)	1.6	December 2006
Java SE 7(Dolphin)	1.7	July 2011
Java SE 8	1.8	March 2014
Java SE 9	9	September, 21st 2017
Java SE 10	10	March, 20th 2018
Java SE 11	11	September, 25th 2018
Java SE 12	12	March, 19th 2019
Java SE 13	13	September, 17th 2019
Java SE 14	14	March, 17th 2020
Java SE 15	15	September, 15th 2020
Java SE 16	16	March, 16th 2021
Java SE 17	*17*	*Expected on Sept. 2021*

Features of Java

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

A list of most important features of Java language is given below.

1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic

Simple

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

- Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Object-oriented

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

Platform Independent

Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides a software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

Java code can be run on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere(WORA).

Secured

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- **No explicit pointer**
- **Java Programs run inside a virtual machine sandbox**
- **Classloader:** Classloader in Java is a part of the Java Runtime Environment(JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access right to objects.
- **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

Robust

Robust simply means strong. Java is robust because:

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

Architecture-neutral

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

Portable

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

High-performance

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

Distributed

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

Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

Dynamic

Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

Java supports dynamic compilation and automatic memory management (garbage collection

C++ vs Java

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

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
Design Goal	C++ was designed for systems and applications programming. It was an extension of C programming language.	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed with a goal of being easy to use and accessible to a broader audience.
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses compiler and interpreter both. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform independent.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comment.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Inheritance Tree	C++ creates a new inheritance tree always.	Java uses a single inheritance tree always because all classes are the child of Object class in java. The object class is the root of the inheritance tree in java.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.
Object-oriented	C++ is an object-oriented language. However, in C language, single root hierarchy is not possible.	Java is also an object-oriented language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from java.lang.Object.

Data Types in Java

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

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

Java Primitive Data Types

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

Java is a statically-typed programming language. It means, all variables must be declared before its use. That is why we need to declare variable's type and name.

There are 8 types of primitive data types:

- boolean data type
- byte data type ●char data type
- short data type
- int data type
- long data type
- float data type
- double data type

Data Type	Default Value	Default size
boolean	False	1 bit
Char	'\u0000'	2 byte
Byte	0	1 byte
short	0	2 byte
Int	0	4 byte
Long	0L	8 byte
Float	0.0f	4 byte
double	0.0d	8 byte

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example: Boolean one = false

Byte Data Type

The byte data type is an example of primitive data type. It isan 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type. **Example:** byte a = 10, byte b = -20

Short Data Type

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example: short s = 10000, short r = -5000

Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range lies between -2,147,483,648 (-2^31) to 2 ,147,483,647 (2^31 -1) (inclusive). Its minimum value is -2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory. **Example:** int a = 100000, int b = -200000

Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is -9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

Example: long a = 100000L, long b = -200000L

Float Data Type

The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example: float f1 = 234.5f

Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

Example: double d1 = 12.3

Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

Example: char letterA = 'A'

Why char uses 2 byte in java and what is \u0000 ?

It is because java uses Unicode system not ASCII code system. The \u0000 is the lowest range of Unicode system. To get detail explanation about Unicode visit next page.

Java Variables

A variable is a container which holds the value while the java program is executed. A variable is assigned with a datatype.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of data types in java: primitive and non-primitive.

Variable

Variable is name of *reserved area allocated in memory*. In other words, it is a *name of memory location*. It is a combination of "vary + able" that means its value can be changed.

1. int data=50;//Here data is variable

Types of Variables

There are three types of variables in java:

- local variable
- instance variable
- static variable

1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.

It is called instance variable because its value is instance specific and is not shared among instances.

3) Static variable

A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

Example to understand the types of variables in java

```
class A{
int data=50;//instance variable
static int m=100;//static variable
void method(){
int n=90;//local variable
}
} //end of class
```

Java Variable Example: Add Two Numbers

```
class Simple{ public static

void main(String[] args){ int

a=10; int b=10; int c=a+b;

System.out.println(c);
}}
```

Output:

```
20
```

Java Variable Example: Widening

```
class Simple{ public static
void main(String[] args){ int
a=10; float f=a;
System.out.println(a);
System.out.println(f);
}}
```

Output:

```
10
10.0
```

Java Variable Example: Narrowing (Typecasting)

```
class Simple{ public static
void main(String[] args){
float f=10.5f;
//int a=f;//Compile time
error int a=(int)f;
System.out.println(f);
System.out.println(a);
}}
```

Output:

```
10.5
10
```

Java Variable Example: Overflow

```
class Simple{ public static
void main(String[] args){
//Overflo
w int
a=130;
byte
b=(byte);
System.out.println(a);
System.out.println(b);
}}
```

Output:

```
130
-126
```

Java Variable Example: Adding Lower Type

```
class Simple{ public static
void main(String[] args){
byte a=10; byte b=10;
//byte c=a+b;//Compile Time Error: because a+b=20
will be int byte c=(byte)(a+b);
System.out.println(c);
}}
```

Output: 20

Simple Java Program Structure

```
public class first
{

  public static void main(String[] args)
  {

       System.out.println("Hello World");

  }

}
```

Java Data Types

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Operators in Java

Operator Type	Category	Precedence
Unary	postfix	expr++ expr--
	prefix	++expr --expr +expr -expr ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Reading Input from User

```
import java.util.*;
public class first {

public static void main(String[] args) {
// TODO Auto-generated method stub

System.out.println("Enter your name");
Scanner test= new Scanner(System.in);
String name=test.nextLine();
System.out.println(name);

}

}
```

If else Statement

- If
- If else
- If else ladder
- Nested if else statement

Syntax - if

```
if(condition)
{
   //code to be executed
}
```

Syntax – if else

```
if(condition)
{
   //Statements that will be executed if condition is
true
}else
{
   //Statements that will be executed if condition is
false
```

Nested If else

```
if(condition1)
  {
  //execute statements if condition1 is true
  }else if(condition2){
  //statements to be executed if condition2 is true
  }
  else if(condition3){
  //statements to be executed if condition3 is true
  }
  ...
  else{
  //statements to be executed if all the conditions are false
  }
```

Switch Statement Syntax

```
switch(expression)
{
      case value1:
       //statements;
       break;
      case value2:
       //statements;
       break;
      ......

      default:
       Code to be executed
      Break;
}
```

Loops in Java

- For
- While
- Do while

Syntax For Loop

```
for(initialization;condition;incr/decr)
{
   //statements
}
```

Loops in Java

```
public class hello {

public static void main(String[] args) {
// TODO Auto-generated method stub

for(int i=0;i<5;i++)
{
System.out.println("Hello");
}

}

}
```

While loop Syntax

```
while(condition)
{
      //statements
}
```

While Loop

```
public class hello {

public static void main(String[] args) {
// TODO Auto-generated method stub

int i=0;
while(i<5)
{
System.out.println("Hello");
i++;
}

}

}
```

Do While Loop Syntax

```
do
{
   //statements

}while(condition);
```

Do while loop

```
public class hello {

public static void main(String[] args)
{
// TODO Auto-generated method stub

   int i = 0;
   do
   {
      System.out.println("Hello");
      i++;
   } while (i < 5);

}

}
```

Arrays in Java

- An array is a container that holds data (values) **of one single type**.
- For example, you can create an array that can hold 100 values of int type.

Array Declaration

int[] arrayname = new int[5];

Initializing an array

int[] test = {1,2,3,4,5};

Multidimensional arrays in Java

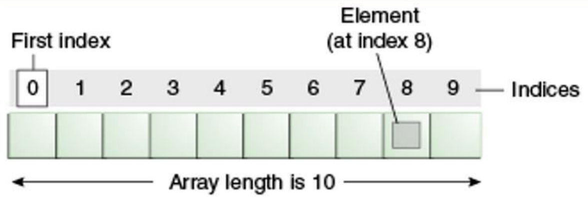
int[][] test = {{1,1},{2,2}};

Java Array

Normally, an array is a collection of similar type of elements that have a contiguous memory location.

Java array is an object which contains elements of a similar data type. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in java is index-based, the first element of the array is stored at the 0 index.



Advantages

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

Disadvantages

- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array in Java

Syntax to Declare an Array in Java

1. dataType[] arr; (or)
2. dataType []arr; (or)
3. dataType arr[];

Instantiation of an Array in Java

1. arrayRefVar=new datatype[size];

Example of Java Array

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

1. //Java Program to illustrate how to declare, instantiate, initialize

```
//and traverse the Java array. class
Testarray{ public static void
main(String args[]){ int a[]=new
int[5];//declaration and instantiation
a[0]=10;//initialization a[1]=20;
a[2]=70; a[3]=40; a[4]=50;
//traversing array for(int
i=0;i<a.length;i++)//length is the property of
array
System.out.println(a[i]);
}}
```
Output:
```
10
20
70
40
50
```

Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

```
int a[]={33,3,4,5};//declaration, instantiation and initialization
```
Let's see the simple example to print this array.

```
//Java Program to illustrate the use of declaration, instantiation
//and initialization of Java array in a single line
class Testarray1{
public static void main(String args[]){
int a[]={33,3,4,5};//declaration, instantiation and initialization
//printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
```
Output:
```
33
3
4
5
```

Passing Array to Method in Java

We can pass the java array to method so that we can reuse the same logic on any array.
Let's see the simple example to get the minimum number of an array using a method.

```
//Java Program to demonstrate the way of passing an array
//to method.
class Testarray2{
//creating a method which receives an array as a parameter
static void min(int arr[]){
int min=arr[0];
for(int i=1;i<arr.length;i++)
if(min>arr[i])
min=arr[i];

System.out.println(min);
}

public static void main(String args[]){
int a[]={33,3,4,5};//declaring and initializing an array
min(a);//passing array to method
}}
```
Output:
```
3
```
Anonymous Array in Java

Java supports the feature of an anonymous array, so you don't need to declare the array while passing an array to the method.

```
//Java Program to demonstrate the way of passing an anonymous array
//to method.
public class TestAnonymousArray{
//creating a method which receives an array as a parameter
static void printArray(int arr[]){
for(int i=0;i<arr.length;i++)
System.out.println(arr[i]);
}

public static void main(String args[]){
printArray(new int[]{10,22,44,66});//passing anonymous array to
method
}}
```
Output:
```
10
22
44
66
```
Returning Array from the Method

We can also return an array from the method in Java.

```
//Java Program to return an array from the method
class TestReturnArray{
//creating method which returns an array
static int[] get(){
return new int[]{10,30,50,90,60};
}

public static void main(String args[]){
//calling method which returns an array
int arr[]=get();
//printing the values of an array
for(int i=0;i<arr.length;i++)
System.out.println(arr[i]);
}}
```
Output:
```
10
30
50
90
60
```
ArrayIndexOutOfBoundsException

The Java Virtual Machine (JVM) throws an ArrayIndexOutOfBoundsException if length of the array in negative, equal to the array size or greater than the array size while traversing the array.

```
//Java Program to demonstrate the case of
//ArrayIndexOutOfBoundsException in a Java Array.
public class TestArrayException{
public static void main(String args[]){
```

```
int arr[]={50,60,70,80};
for(int i=0;i<=arr.length;i++){
System.out.println(arr[i]);
}
}}
```

Output:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4 at
        TestArrayException.main(TestArrayException.java:5)
50
60
70
80
```

Multidimensional Array in Java

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in Java

```
dataType[][] arrayRefVar; (or)
dataType [][]arrayRefVar; (or)
dataType arrayRefVar[][]; (or)
dataType []arrayRefVar[];
```

Example to instantiate Multidimensional Array in Java

```
int[][] arr=new int[3][3];//3 row and 3 column
```

Example to initialize Multidimensional Array in Java

```
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;
```

Example of Multidimensional Java Array

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

```
//Java Program to illustrate the use of multidimensional array
class Testarray3{
public static void main(String args[]){
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
//printing 2D array
for(int i=0;i<3;i++){
for(int j=0;j<3;j++){
System.out.print(arr[i][j]+" ");
}
System.out.println();
}
}} Output:
```

```
1 2 3
2 4 5
4 4 5
```

Jagged Array in Java

If we are creating odd number of columns in a 2D array, it is known as a jagged array. In other words, it is an array of arrays with different number of columns. //Java Program to illustrate the jagged array class

```
TestJaggedArray{ public static void main(String[] args){ //declaring a 2D array with odd columns int arr[][] =
new int[3][]; arr[0] = new int[3]; arr[1] = new int[4]; arr[2] = new int[2];
        //initializing a jagged array
        int count = 0; for (int i=0;
        i<arr.length; i++) for(int j=0;
        j<arr[i].length; j++) arr[i][j] =
        count++;
        //printing the data of a jagged
        array for (int i=0; i<arr.length;
        i++){ for (int j=0;
        j<arr[i].length; j++){
            System.out.print(arr[i][j]+"
          "); }
          System.out.println();//new line
        }}}
```

Output:

```
0 1 2
3 4 5 6
7 8
```

What is the class name of Java array?

In Java, an array is an object. For array object, a proxy class is created whose name can be obtained by getClass().getName() method on the object.

```
//Java Program to get the class name of array in
Java class Testarray4{ public static void
main(String args[]){ //declaration and
initialization of array int arr[]={4,4,5};
//getting the class name of Java array
Class c=arr.getClass();
String name=c.getName();
//printing the class name of Java array
System.out.println(name);
}}
```

Output:

```
I
```

Copying a Java Array

We can copy an array to another by the arraycopy() method of System class.

Syntax of arraycopy method

```
public static void arraycopy(
Object src, int srcPos,Object dest, int destPos, int length
)
```

Example of Copying an Array in Java

```
//Java Program to copy a source array into a destination array in Java
class TestArrayCopyDemo {
public static void main(String[] args) {
//declaring a source array
char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
'i', 'n', 'a', 't', 'e', 'd' };
//declaring a destination array
char[] copyTo = new char[7];
//copying array using System.arraycopy() method
System.arraycopy(copyFrom, 2, copyTo, 0, 7);
//printing the destination array
System.out.println(String.valueOf(copyTo));
}
} Output: caffein
```

Addition of 2 Matrices in Java

Let's see a simple example that adds two matrices.

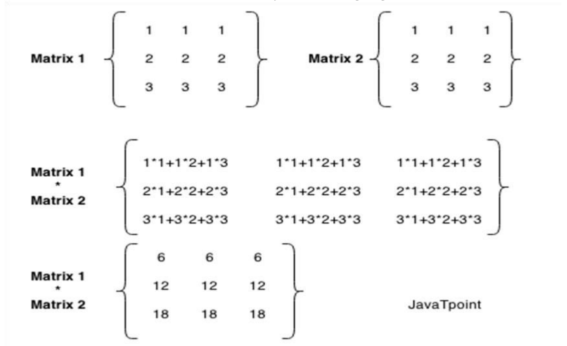
```
//Java Program to demonstrate the addition of two matrices in Java
class Testarray5{
public static void main(String args[]){
//creating two matrices
int a[][]={{1,3,4},{3,4,5}};
int b[][]={{1,3,4},{3,4,5}};

        //creating another matrix to store the sum of
two matrices
int c[][]=new int[2][3];

//adding and printing addition of 2 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
```

Multiplication of 2 Matrices in Java

In the case of matrix multiplication, a one-row element of the first matrix is multiplied by all the columns of the second matrix which can be understood by the image given below.



Let's see a simple example to multiply two matrices of 3 rows and 3 columns.

```
//Java Program to multiply two matrices
public class MatrixMultiplicationExample{
public static void main(String args[]){
//creating two matrices
int a[][]={{1,1,1},{2,2,2},{3,3,3}};
int b[][]={{1,1,1},{2,2,2},{3,3,3}};
        //creating another matrix to store the multiplication of two
matrices
int c[][]=new int[3][3];  //3 rows and 3 columns
//multiplying and printing multiplication of 2 matrices
for(int i=0;i<3;i++){
for(int j=0;j<3;j++){
c[i][j]=0;
for(int k=0;k<3;k++)
{
c[i][j]+=a[i][k]*b[k][j];
}//end of k loop
System.out.print(c[i][j]+" ");  //printing matrix element
}//end of j loop
System.out.println();//new line
}}}
```

Output:

```
6 6 6
12 12 12
18 18 18
```

Java For-each Loop | Enhanced For Loop

The Java for-each loop or enhanced for loop is introduced since J2SE 5.0. It provides an alternative approach to traverse the array or collection in Java. It is mainly used to traverse the array or collection elements. The advantage of the for-each loop is that it eliminates the possibility of bugs and makes the code more readable. It is known as the for-each loop because it traverses each element one by one.

The drawback of the enhanced for loop is that it cannot traverse the elements in reverse order. Here, you do not have the option to skip any element because it does not work on an index basis. Moreover, you cannot traverse the odd or even elements only.

But, it is recommended to use the Java for-each loop for traversing the elements of array and collection because it makes the code readable.

Advantages

- It makes the code more readable.
- It eliminates the possibility of programming errors.

Syntax

The syntax of Java for-each loop consists of data_type with the variable followed by a colon (:), then array or collection.

```
for(data_type variable : array | collection){
//body of for-each loop
}
```

How it works?

The Java for-each loop traverses the array or collection until the last element. For each element, it stores the element in the variable and executes the body of the for-each loop.

For-each loop Example: Traversing the array elements

```
//An example of Java for- each loop
class ForEachExample1{
  public static void main(String args[]){
   //declaring an array
   int
arr[]={12,13,14,44}
;
//traversing the array with for- each loop
for(int i:arr){
 System.out.println(i);
}}}
```
2.

Test it Now

Output:

```
12
12
14
44
```

Let us see another of Java for-each loop where we are going to total the elements.

```
class ForEachExample1{
public static void main(String args[]){
int arr[]={12,13,14,44};
int total=0;
for(int i:arr){
total=total+i;
}
System.out.println("Total: "+total);
}
}
```

Output:

```
Total: 83
```

Java String

In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

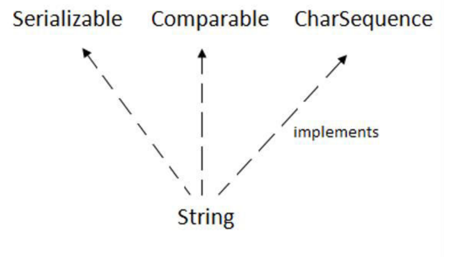
```
1. char[]
   ch={'j','a','v','a','t','p','o','i'
   ,'n','t'};
2. String s=new String(ch);
```

is same as:

```
1. String s="javatpoint";
```

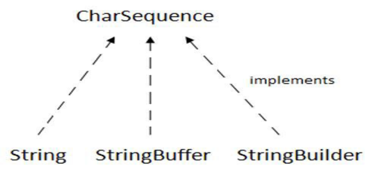
Java String class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.



CharSequence Interface

The CharSequence interface is used to represent the sequence of characters. String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in java by using these three classes.



The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.

We will discuss immutable string later. Let's first understand what is String in Java and how to create the String object.

What is String in java

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

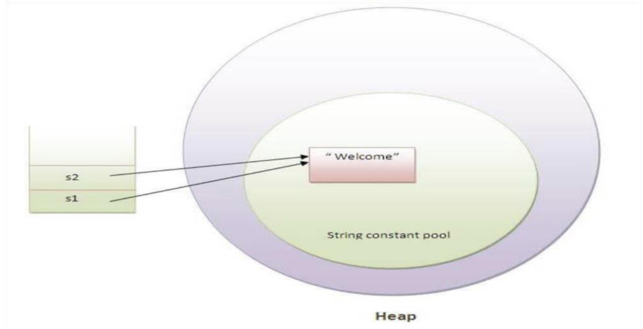
1) String Literal

Java String literal is created by using double quotes. For Example:

```
1. String s="welcome";
```

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

```
1. String s1="Welcome";
2. String s2="Welcome";//It doesn't create a new instance
```



In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool, that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

Note: String objects are stored in a special memory area known as the "string constant pool".

Why Java uses the concept of String literal?

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

2) By new keyword

```
String s=new String("Welcome");//creates two objects and one reference variable
```

In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

Java String Example

```
public class StringExample{
public static void main(String args[]){
String s1="java";//creating string by java string literal
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);//converting char array to string
String s3=new String("example");//creating java string by new keyword
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```

```
java
strings
example
```

Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

The java.lang.String class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string such as trimming, concatenating, converting, comparing, replacing strings etc.

Java String is a powerful concept because everything is treated as a string if you submit any form in window based, web based or mobile application.

Let's see the important methods of String class.

No.	Method	Description
1	char charAt(int index)	returns char value for the particular index
2	int length()	returns string length
3	static String format(String format, Object... args)	returns a formatted string.
4	static String format(Locale l, String format, Object... args)	returns formatted string with given locale.
5	String substring(int beginIndex)	returns substring for given begin index.
6	String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index.
7	boolean contains(CharSequence s)	returns true or false after matching the sequence of char value.
8	static String join(CharSequence delimiter, CharSequence... elements)	returns a joined string.
9	static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	returns a joined string.
10	boolean equals(Object another)	checks the equality of string with the given object.
11	boolean isEmpty()	checks if string is empty.
12	String concat(String str)	concatenates the specified string.
13	String replace(char old, char new)	replaces all occurrences of the specified char value.
14	String replace(CharSequence old, CharSequence new)	replaces all occurrences of the specified CharSequence.
15	static String equalsIgnoreCase(String another)	compares another string. It doesn't check case.
16	String[] split(String regex)	returns a split string matching regex.
17	String[] split(String regex, int limit)	returns a split string matching regex and limit.

18	String intern()	returns an interned string.
19	int indexOf(int ch)	returns the specified char value index.
20	int indexOf(int ch, int fromIndex)	returns the specified char value index starting with given index.
21	int indexOf(String substring)	returns the specified substring index.
22	int indexOf(String substring, int fromIndex)	returns the specified substring index starting with given index.
23	String toLowerCase()	returns a string in lowercase.
24	String toLowerCase(Locale l)	returns a string in lowercase using specified locale.
25	String toUpperCase()	returns a string in uppercase.
26	String toUpperCase(Locale l)	returns a string in uppercase using specified locale.
27	String trim()	removes beginning and ending spaces of this string.
28	static String valueOf(int value)	converts given type into string. It is an overloaded method.

Java String format()

The **java string format()** method returns the formatted string by given locale, format and arguments.

If you don't specify the locale in String.format() method, it uses default locale by calling *Locale.getDefault()* method.

The format() method of java language is like *sprintf()* function in c language and *printf()* method of java language.

Internal implementation

```
public static String format(String format, Object... args) {
return new Formatter().format(format, args).toString();
}
```

Signature

There are two type of string format() method:

1. **public static** String format(String format, Object... args)
2. and,
3. **public static** String format(Locale locale, String format, Object... args)

Parameters

locale : specifies the locale to be applied on the format() method.

format : format of the string. **args** : arguments for the format

string. It may be zero or more.

Returns

formatted string

Throws

NullPointerException : if format is null.

IllegalFormatException : if format is illegal or incompatible.

Java String format() method example

```
public class FormatExample{
public static void main(String args[]){
String name="sonoo";
String sf1=String.format("name is %s",name);
String sf2=String.format("value is %f",32.33434);
String sf3=String.format("value is %32.12f",32.33434);//returns 12 char fractional part filling with 0
System.out.println(sf1);
System.out.println(sf2);
System.out.println(sf3);
}}
```

Test it Now

```
name is sonoo
value is 32.334340
value is          32.334340000000
```

Format Specifier	Data Type	Output
%a	floating point (except *BigDecimal*)	Returns Hex output of floating point number.
%b	Any type	"true" if non-null, "false" if null
%c	character	Unicode character
%d	integer (incl. byte, short, int, long, bigint)	Decimal Integer
%e	floating point	decimal number in scientific notation
%f	floating point	decimal number
%g	floating point	decimal number, possibly in scientific notation depending on the precision and value.
%h	any type	Hex String of value from hashCode() method.
%n	none	Platform-specific line separator.
%o	integer (incl. byte, short, int, long, bigint)	Octal number
%s	any type	String value
%t	Date/Time (incl. long, Calendar, Date and TemporalAccessor)	%t is the prefix for Date/Time conversions. More formatting flags are needed after this. See Date/Time conversion below.
%x	integer (incl. byte, short, int, long, bigint)	Hex string.

Java String join()

The **java string join()** method returns a string joined with given delimiter. In string join method, delimiter is copied for each elements.

In case of null element, "null" is added. The join() method is included in java string since JDK 1.8.

There are two types of join() methods in java string.

Signature

The signature or syntax of string join method is given below:

```
public static String join(CharSequence delimiter, CharSequence... elements)
and
public static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)
```

Parameters

delimiter : char value to be added with each element **elements**

: char value to be attached with delimiter

Returns

joined string with delimiter

Java String join() method example

```
public class StringJoinExample{
public static void main(String args[]){
String joinString1=String.join("-","welcome","to","javatpoint");
System.out.println(joinString1);
}}
```

Test it Now

```
welcome-to-javatpoint
```

Java String join() Method Example 2

We can use delimeter to format the string as we did in the below example to show date and time.

```
public class StringJoinExample2 {
public static void main(String[] args) {
String date = String.join("/","25","06","2018");
System.out.print(date);
String time = String.join(":", "12","10","10");
System.out.println(" "+time);
}
}
```

```
25/06/2018 12:10:10
```

Java String split()

The **java string split()** method splits this string against given regular expression and returns a char array.

Signature

There are two signature for split() method in java string.

```
public String split(String regex)
and,
public String split(String regex, int limit)
```

Parameter

regex : regular expression to be applied on string. **limit** : limit for the number of strings in array. If it is

zero, it will returns all the strings matching regex.

Java String split() method example

The given example returns total number of words in a string excluding space only. It also includes special characters.

```
public class SplitExample{
public static void main(String
args[]){
String s1="java string split method by javatpoint";
String[] words=s1.split("\\s");//splits the string based on whitespace
//using java foreach loop to print elements of string array
for( String w:words ){
System.out.println(w);
}
}}
```

Test it Now

```
java
string
split
method
by
javatpoint
```

Java String split() method with regex and length example

```
public class SplitExample2{
public static void main(String args[]){
String s1="welcome to split world";
System.out.println("returning words:");
for(String w:s1.split("\\s",0)){
System.out.println(w);
}
System.out.println("returning words:");
for(String w:s1.split("\\s",1)){
.System.out.println(w);
.}
.System.out.println("returning words:");
.for(String w:s1.split("\\s",2)){
.System.out.println(w);
.}

.}}
```

Test it Now

```
returning words:
welcome
to
split
world
returning words:
welcome to split world
returning words:
welcome
to split world
```

Java String intern()

The **java string intern()** method returns the interned string. It returns the canonical representation of string. It can be used to return string from memory, if it is created by new keyword. It creates exact copy of heap string object in string constant pool.

Java String intern() method example

```
public class InternExample{
public static void main(String args[]){
String s1=new String("hello");
String s2="hello";
String s3=s1.intern();//returns string from pool, now it will be same as s2
System.out.println(s1==s2);//false because reference variables are pointing to different instance
System.out.println(s2==s3);//true because reference variables are pointing to same instance
}}
```

Test it Now

```
false
true
```

Java String intern() Method Example 2

Let's see one more example to understand the string intern concept.

```
public class InternExample2 {
public static void main(String[] args) {
String s1 = "Javatpoint";
String s2 = s1.intern();
String s3 = new String("Javatpoint");
String s4 = s3.intern();
System.out.println(s1==s2); // True
System.out.println(s1==s3); // False
System.out.println(s1==s4); // True
System.out.println(s2==s3); // False
System.out.println(s2==s4); // True
System.out.println(s3==s4); // False
}}
```

Test it Now

```
true
false
true
false
true
false
```

Java String toUpperCase() and toLowerCase() method

The java string toUpperCase() method converts this string into uppercase letter and string toLowerCase() method into lowercase letter.

```
String s="Sachin";
System.out.println(s.toUpperCase());//SACHIN
System.out.println(s.toLowerCase());//sachin
System.out.println(s);//Sachin(no change in original)
```

Output

```
SACHIN
```

sachin
Sachin
Java String trim() method

The string trim() method eliminates white spaces before and after string.

1. String s=" Sachin ";
2. System.out.println(s);// Sachin
3. System.out.println(s.trim());//Sachin

<u>Output</u>
Sachin
Sachin
Java String startsWith() and endsWith() method

1. String s="Sachin";
2. System.out.println(s.startsWith("Sa"));//true
3. System.out.println(s.endsWith("n"));//true

<u>Outp</u>
<u>ut</u>
true
true
Java String charAt() method

The string charAt() method returns a character at specified index.

1. String s="Sachin";
2. System.out.println(s.charAt(0));//S
3. System.out.println(s.charAt(3));//h

<u>Output</u>
S
h
Java String length() method

The string length() method returns length of the string.

1. String s="Sachin";
2. System.out.println(s.length());//6

<u>Output</u>
6
Java String intern() method

A pool of strings, initially empty, is maintained privately by the class String.
When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the equals(Object) method, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.

1. String s=new
String("Sachin"); 2. String
s2=s.intern();
3. System.out.println(s2);//Sachin

<u>Output</u>
Sachin
Java String valueOf() method

The string valueOf() method coverts given type such as int, long, float, double, boolean, char and char array into string.

1. int a=10;
2. String s=String.valueOf(a);
3. System.out.println(s+10);

Output:
1010

Java String replace() method

The string replace() method replaces all occurrence of first sequence of character with second sequence of character.

1. String s1="Java is a programming language. Java is a platform. Java is an Island.";
2. String replaceString=s1.replace("Java","Kava");//replaces all occurrences of "Java" to "Kava" 3. System.out.println(replaceString);

Output:
Kava is a programming language. Kava is a platform. Kava is an Island.

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

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

For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.
An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.
Object Definitions:

- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.
- The object is *an instance of a class*.

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

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

Syntax to declare a class:

```
class <class_name>{
field;
method;
}
```

Instance variable in Java
A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

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

- Code Reusability
- Code Optimization

---

new keyword in Java
The new keyword is used to allocate memory at runtime. All objects get memory in the Heap memory area.

Object and Class Example: main within the class
In this example, we have created a Student class which has two data members id and name. We are creating the object of the Student class by new keyword and printing the object's value.
Here, we are creating a main() method inside the class.
File: Student.java

```
//Java Program to illustrate how to define a class and fields
//Defining a Student class.
class Student{
//defining fields
int id;//field or data member or instance variable
String name;
//creating main method inside the Student class
public static void main(String args[]){
//Creating an object or instance
Student s1=new Student();//creating an object of Student
//Printing values of the object
System.out.println(s1.id);//accessing member through reference variable 13.   System.out.println(s1.name);
}
}
```

Test it Now
Output:

```
0
n
u
l
l
```

Object and Class Example: main outside the class
In real time development, we create classes and use it from another class. It is a better approach than previous one. Let's see a simple example, where we are having main() method in another class.
We can have multiple classes in different java files or single java file. If you define multiple classes in a single java source file, it is a good idea to save the file name with the class name which has main() method.
File: TestStudent1.java

```
//Java Program to demonstrate having the main method in
//another class
//Creating Student class.
class Student{
int id;
String name;
}
//Creating another class TestStudent1 which contains the main method
class TestStudent1{
public static void main(String args[]){
Student s1=new Student();
System.out.println(s1.id);
System.out.println(s1.name);
}
}
```

Test it Now
Output:

```
0
n
u
l
l
```

3 Ways to initialize object
There are 3 ways to initialize object in java.

1. By reference variable

2. By method
3. By constructor

1) Object and Class Example: Initialization through reference

Initializing an object means storing data into the object. Let's see a simple example where we are going to initialize the object through a reference variable.

File: TestStudent2.java

```
class Student{
int id;
String name;
}
class TestStudent2{
public static void main(String args[]){
Student s1=new Student();
s1.id=101;
s1.name="Sonoo";
System.out.println(s1.id+" "+s1.name);//printing members with a white space
}}
```

Test it Now

Output:

```
101 Sonoo
```

We can also create multiple objects and store information in it through reference variable.

File: TestStudent3.java

```
class Student{
int id;
String name;
}
class TestStudent3{
public static void main(String args[]){
//Creating objects
Student s1=new Student();
Student s2=new Student();
//Initializing objects
s1.id=101;
s1.name="Sonoo";
s2.id=102;
s2.name="Amit";
//Printing data
System.out.println(s1.id+" "+s1.name);
System.out.println(s2.id+" "+s2.name);
}}
```

Test it Now

Output:

```
101 Sonoo
102 Amit
```

2) Object and Class Example: Initialization through method

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

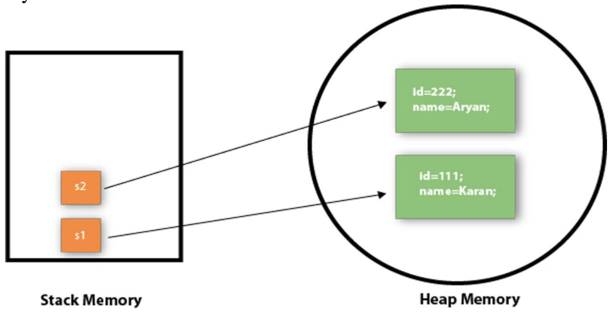
File: TestStudent4.java

```
class Student{
int rollno;
String name;
void insertRecord(int r, String n){
rollno=r;
name=n;
}
void displayInformation(){System.out.println(rollno+" "+name);}
}
class TestStudent4{
public static void main(String args[]){
Student s1=new Student();
Student s2=new Student();
s1.insertRecord(111 ,"Karan");
s2.insertRecord(222,"Aryan");
s1.displayInformation();
s2.displayInformation();
}
}
```

Test it Now

Output:

```
111 Karan
222 Aryan
```



As you can see in the above figure, object gets the memory in heap memory area. The reference variable refers to the object allocated in the heap memory area. Here, s1 and s2 both are reference variables that refer to the objects allocated in memory.

3) Object and Class Example: Initialization through a constructor

We will learn about constructors in java later.

Object and Class Example: Employee

Let's see an example where we are maintaining records of employees.

File: TestEmployee.java

```
class Employee{
int id;
String name;
float salary;
void insert(int i, String n, float s) {
id=i;
name=n;
salary=s;
}
void display(){System.out.println(id+" "+name+" "+salary);}
```

```
    }
    public class TestEmployee {
    public static void main(String[] args) {
    Employee e1=new Employee(); Employee e2=new Employee();
 Employee e3=new Employee();
 e1.insert(101,"ajeet",45000);
 e2.insert(102,"irfan",25000);
 e3.insert(103,"nakul",55000);
 e1.display();        e2.display();
 e3.display();
 }
 }
```

Test it Now

Output:

```
101 ajeet 45000.0
102 irfan 25000.0
103 nakul 55000.0
```

Object and Class Example: Rectangle

There is given another example that maintains the records of Rectangle class.

File: TestRectangle1.java

```
class Rectangle{
int length;
int width;
void insert(int l, int w){
length=l;
width=w;
}
void calculateArea(){System.out.println(length*width);}
}
class TestRectangle1{
public static void main(String args[]){
Rectangle r1=new Rectangle();
Rectangle r2=new Rectangle();
r1.insert(11 ,5);
r2.insert(3,15);
r1.calculateArea();
r2.calculateArea();
}
}
```

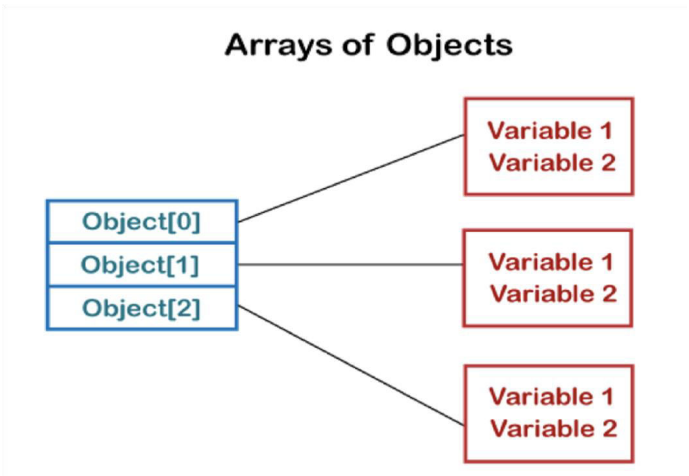
Test it Now

Output:

```
55
45
```

Array of Objects in Java

Java is an object-oriented programming language. Most of the work is done with the help of **objects**. We know that an array is a collection of the same data type that dynamically creates objects and can have elements of primitive types. Java allows us to store objects in an array. In Java, the class is also a user-defined data type. An array that contains **class type elements** are known as an **array of objects**. It stores the reference variable of the object.



Creating an Array of Objects

Before creating an array of objects, we must create an instance of the class by using the new keyword. We can use any of the following statements to create an array of objects.

Syntax:

1. ClassName obj[]=new ClassName[array_length]; //declare and instantiate an array of objects Or 2. ClassName[] objArray;

Or

3. ClassName objeArray[];

Suppose, we have created a class named Employee. We want to keep records of 20 employees of a company having three departments. In this case, we will not create 20 separate variables. Instead of this, we will create an array of objects, as follows.

4. Employee department1[20]; 5. Employee department2[20];
6. Employee department3[20];

The above statements create an array of objects with 20 elements.

Let's create an array of objects in a Java program.

In the following program, we have created a class named Product and initialized an array of objects using the constructor. We have created a constructor of the class Product that contains product id and product name. In the main function, we have created individual objects of the class Product. After that, we have passed initial values to each of the objects using the constructor.

ArrayOfObjects.java

```
public class ArrayOfObjects
```

```
{
public static void main(String args[])
{
//create an array of product object
Product[] obj = new Product[5]  ;
//create & initialize actual product objects using constructor
obj[0] = new Product(23907,"Dell Laptop") ;
obj[1] = new Product(91240,"HP 630") ;
obj[2] = new Product(29823,"LG OLED TV") ;
obj[3] = new Product(11908,"MI Note Pro Max 9") ;
obj[4] = new Product(43590,"Kingston USB") ;
//display the product object data
System.out.println("Product Object 1:") ;
obj[0] .display();
System.out.println("Product Object 2:") ;
obj[1] .display();
System.out.println("Product Object 3:") ;
obj[2] .display();
System.out.println("Product Object 4:") ;
obj[3] .display();
System.out.println("Product Object 5:") ;
obj[4] .display();
}
}
//Product class with product Id and product name as attributes
class Product
{
int pro_Id;
String pro_name;
//Product class constructor
Product(int pid, String n)
{
pro_Id = pid;
pro_name = n;
}
public void display()
{
System.out.print("Product Id = "+pro_Id + " " + " Product Name = "+ pro_name);
System.out.println();
}
}
```

Output:

```
Product Object 1:
Product Id = 23907   Product Name = Dell Laptop
Product Object 2:
Product Id = 91240   Product Name = HP 630
Product Object 3:
Product Id = 29823   Product Name = LG OLED TV
Product Object 4:
Product Id = 11908   Product Name = MI Note Pro Max 9
Product Object 5:
Product Id = 43590   Product Name = Kingston USB
```

Constructors in Java

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the object is created, and memory is allocated for the object.

It is a special type of method which is used to initialize the object.

When is a constructor called

Every time an object is created using new() keyword, at least one constructor is called. It calls a default constructor.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Note: We can use access modifiers while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

```
< class_name >(){ }
```

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

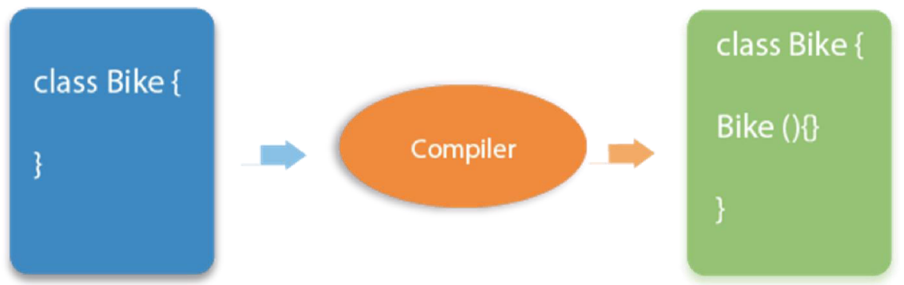
```
//Java Program to create and call a default constructor
class Bike1{
//creating a default constructor
Bike1(){System.out.println("Bike is created");}
//main method
public static void main(String args[]){
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

Test it Now

Output:

```
Bike is created
```

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.



What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Example of default constructor that displays the default values

```
//Let us see another example of default constructor
//which displays the default values
class Student3{
int id;
String name;
//method to display the value of id and name   void display(){System.out.println(id+" "+name);}

public static void main(String args[]){
//creating objects
Student3 s1=new Student3();
Student3 s2=new Student3();
//displaying values of the object
s1.display();
s2.display();
}
}

```

Output:

```
0 null
0 null
```

Explanation: In the above class,you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to the distinct objects. However, you can provide the same values also.

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
//Java Program to demonstrate the use of parameterized
constructor class Student4{
   int id;
   String name;
   //creating a parameterized constructor
Student4(int i,String
   n){ id = i;
   name = n;
   }
   //method to display the values void
   display(){System.out.println(id+" "+name);}
   public static void main(String args[]){
   //creating objects and passing values
   Student4 s1 = new Student4(111 ,"Karan");
   Student4 s2 = new Student4(222,"Aryan");
   //calling method to display the values of
   object s1.display(); s2.display();
   }
}
```

Test it Now

Output:

```
111 Karan
222 Aryan
```

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Example of Constructor Overloading

```
      //Java program to overload constructors in
java class Student5{
   int id;
   String
   name; int
   age;
   //creating two arg
   constructor Student5(int
   i,String n){ id = i;
   name = n;
   }
   //creating three arg
   constructor Student5(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[]){
   Student5 s1 = new Student5(111
   ,"Karan"); Student5 s2 = new
   Student5(222,"Aryan",25); s1.display();
   s2.display();
   }
}
```

Test it Now

Output:

```
111 Karan 0
222 Aryan 25
```

Difference between constructor and method in Java

There are many differences between constructors and methods. They are given below.

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behaviour of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as class name.

Java Copy Constructor

There is no copy constructor in java. However, we can copy the values from one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in java. They are:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

In this example, we are going to copy the values of one object into another using java constructor.

```
   1. //Java program to initialize the values from one object to
another class Student6{ int id;
  String name;
  //constructor to initialize integer and
  string Student6(int i,String n){ id = i;
  name = n;
  }
  //constructor to initialize another
  object Student6(Student6 s){ id =
  s.id;
  name =s.name;
  }
  void display(){System.out.println(id+" "+name);}
  public static void main(String args[]){
  Student6 s1 = new
  Student6(111,"Karan"); Student6 s2
  = new Student6(s1); s1.display();
  s2.display();
  }
}
```

Test it Now

Output:

```
111 Karan
111 Karan
```

Copying values without constructor

Constructors in Java

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the object is created, and memory is allocated for the object.

It is a special type of method which is used to initialize the object.

When is a constructor called

Every time an object is created using new() keyword, at least one constructor is called. It calls a default constructor.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Note: We can use access modifiers while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

```
1.      < class_name >(){}
```

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
//Java Program to create and call a default constructor
class Bike1{
//creating a default constructor
Bike1(){System.out.println("Bike is created");}
//main method
public static void main(String args[]){
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

Test it Now

Output:

```
Bike is created
```

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.

What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Example of default constructor that displays the default values

```
//Let us see another example of default constructor
//which displays the default values
class Student3{
int id;
String name;
//method to display the value of id and name 7.   void display(){System.out.println(id+" "+name);} 8.
public static void main(String args[]){
//creating objects
Student3 s1=new Student3();
Student3 s2=new Student3();
//displaying values of the object
s1.display();
s2.display();
}
}
```

Output:

```
0 null
0 null
```

Explanation: In the above class,you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to the distinct objects. However, you can provide the same values also.

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
//Java Program to demonstrate the use of parameterized
constructor class Student4{
   int id;
   String name;
   //creating a parameterized
   constructor Student4(int i,String
   n){ id = i;
   name = n;
   }
   //method to display the values void
   display(){System.out.println(id+" "+name);}
   public static void main(String args[]){
   //creating objects and passing values
   Student4 s1 = new Student4(111 ,"Karan");
   Student4 s2 = new
   Student4(222,"Aryan"); //calling method
   to display the values of object
   s1.display(); s2.display();
  }
}
```

Test it Now

Output:

```
111 Karan
222 Aryan
```

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Example of Constructor Overloading

```
//Java program to overload constructors in
java class Student5{
   int id;
   String
   name; int
   age;
   //creating two arg
   constructor Student5(int
   i,String n){ id = i;
   name = n;
   }
   //creating three arg
   constructor Student5(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[]){
   Student5 s1 = new Student5(111
   ,"Karan"); Student5 s2 = new
   Student5(222,"Aryan",25); s1.display();
   s2.display();
  }
}
```

Test it Now

Output:

```
111 Karan 0
222 Aryan 25
```

Difference between constructor and method in Java

There are many differences between constructors and methods. They are given below.

Java Constructor	**Java Method**
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as class name.

Java Copy Constructor

There is no copy constructor in java. However, we can copy the values from one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in java. They are:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

In this example, we are going to copy the values of one object into another using java constructor.

```
1.      //Java program to initialize the values from one object to
another class Student6{
   int id;
   String name;
   //constructor to initialize integer and
   string Student6(int i,String n){ id = i;
   name = n;
   }
   //constructor to initialize another
   object Student6(Student6 s){ id =
   s.id;
   name =s.name; } void
   display(){System.out.println(id+" "+name);}
   public static void main(String args[]){
   Student6 s1 = new Student6(111
   ,"Karan"); Student6 s2 = new
   Student6(s1); s1.display();
   s2.display();
  }
}
```

Test it Now

Output:

```
111 Karan
111 Karan
```

Copying values without constructor

We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.

```
class Student7{
   int id;
   String name;
   Student7(int i,String
   n){ id = i;
   name
   = n; }
   Student7(){} void
   display(){System.out.println(id+" "+name);}
   public static void main(String args[]){
   Student7 s1 = new Student7(111
   ,"Karan"); Student7 s2 = new
   Student7(); s2.id=s1.id;
   s2.name=s1.name;
   s1.display();
   s2.display();
  }
}
```

Test it Now

Output:

```
111 Karan
111 Karan
```

Q) Does constructor return any value?

Yes, it is the current class instance (You cannot use return type yet it returns a value).

Can constructor perform other tasks instead of initialization?

Yes, like object creation, starting a thread, calling a method, etc. You can perform any operation in the constructor as you perform in the method.

Method Overloading in Java

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

In java, Method Overloading is not possible by changing the return type of the method only.

1) Method Overloading: changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating static methods so that we don't need to create instance for calling methods.

```
class
Adder
{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return
a+b+c;}
} class TestOverloading1{ public
static  void  main(String[]  args){
System.out.println(Adder.add(11,
11)) ;
System.out.println(Adder.add(11,11,11)) ;
}}
```

Test it Now

Output:

```
22
33
```

2) Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

```
class Adder{ static int add(int a, int b){return
a+b;}  static  double  add(double  a,  double
b){return a+b;}
} class TestOverloading2{ public
static void main(String[] args){
System.out.println(Adder.add(11,11)) ;
System.out.println(Adder.add(12.3,12.6));
}}
```
Test it Now

Output:
```
22
24.9
```

Q) Why Method Overloading is not possible by changing the return type of method only?

In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur:
```
class Adder{ static int add(int a,int
b){return a+b;} static double add(int
a,int b){return a+b;}
} class TestOverloading3{ public
static void main(String[] args){
System.out.println(Adder.add(11,11)) ;//ambiguity
}}
```
Test it Now

Output:
```
Compile Time Error: method add(int,int) is already defined in class Adder
```
System.out.println(Adder.add(11,11)) ; //Here, how can java determine which sum() method should be called?

Note: Compile Time Error is better than Run Time Error. So, java compiler renders compiler time error if you declare the same method having same parameters.

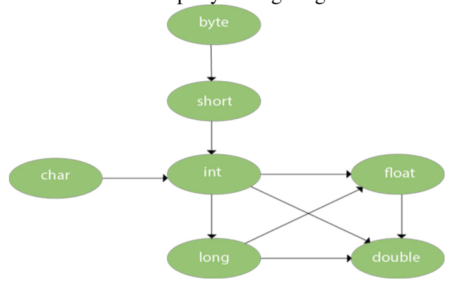
Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only. Let's see the simple example:

```
1. class TestOverloading4{
2. public static void main(String[] args){System.out.println("main with String[]");}
3. public static void main(String args){System.out.println("main with String");} 4. public static void
   main(){System.out.println("main without args");}
5.     }
```
Test it Now

Output:
```
main with String[]
```
Method Overloading and Type Promotion

One type is promoted to another implicitly if no matching datatype is found. Let's understand the concept by the figure given below:



As displayed in the above diagram, byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int,long,float or double. The char datatype can be promoted to int,long,float or double and so on.

Example of Method Overloading with TypePromotion
```
class OverloadingCalculation1{ void sum(int
  a,long b){System.out.println(a+b);} void sum(int
  a,int b,int c){System.out.println(a+b+c);}
  public static void main(String args[]){
  OverloadingCalculation1 obj=new OverloadingCalculation1();
  obj.sum(20,20);//now second int literal will be promoted to long
  obj.sum(20,20,20);
    }
}
```
Test it Now

Output:40
```
60
```

Example of Method Overloading with Type Promotion if matching found

If there are matching type arguments in the method, type promotion is not performed.
```
class OverloadingCalculation2{ void sum(int a,int
b){System.out.println("int arg method invoked");} void sum(long a,long
b){System.out.println("long arg method invoked");}
  public static void main(String args[]){
  OverloadingCalculation2 obj=new OverloadingCalculation2();
  obj.sum(20,20);//now int arg sum() method gets invoked
  }
}
```
Test it Now

Output:intarg method invoked

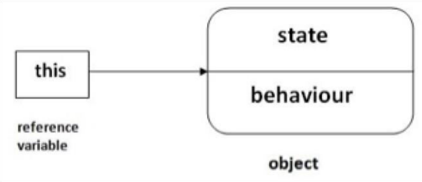
Example of Method Overloading with Type Promotion in case of ambiguity

If there are no matching type arguments in the method, and each method promotes similar number of arguments, there will be ambiguity.
```
class OverloadingCalculation3{ void sum(int a,long
  b){System.out.println("a method invoked");} void sum(long
  a,int b){System.out.println("b method invoked");}
  public static void main(String args[]){
  OverloadingCalculation3 obj=new OverloadingCalculation3();
  obj.sum(20,20);//now ambiguity
  }
}
```
Test it Now

Output:Compile Time Error

This keyword in java

There can be a lot of usage of **java this keyword**. In java, this is a **reference variable** that refers to the current object.



Classname reference variable= new classname();

Usage of java this keyword

Here is given the 6 usage of java this keyword.

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

Suggestion: If you are beginner to java, lookup only three usage of this keyword.

---

1) this: to refer current class instance variable

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

Understanding the problem without this keyword

Let's understand the problem if we don't use this keyword by the example given below:

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
rollno=rollno;
name=name;
fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis1{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Test it Now

Output:

```
0 null 0.0
0 null 0.0
```

In the above example, parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable.

Solution of the above problem by this keyword

```
class
Student{ int
rollno;
String
name; float
fee;
Student(int rollno,String name,float
fee){ this.rollno=rollno;
this.name=name; this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
```

```
class TestThis2{ public static void
main(String args[]){ Student s1=new
Student(111,"ankit",5000f); Student
s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
```

Test it Now

Output:

```
111 ankit 5000
112 sumit 6000
```

If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program:

Program where this keyword is not required

```
class Student{
int rollno;
String name;
float fee;
Student(int r,String n,float f){
rollno=r;
name=n;
fee=f;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}

}

class TestThis3{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

```
111 ankit 5000
112 sumit 6000
```

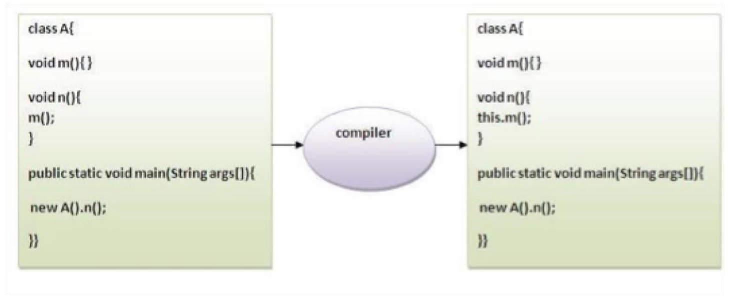
Test it Now

Output:

It is better approach to use meaningful names for variables. So we use same name for instance variables and parameters in real time, and always use this keyword.

2) this: to invoke current class method

You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method. Let's see the example



```
class A{
void m(){}
void n(){
m();
}
public static void main(String args[]){
new A().n();
}}
```

```
class A{
void m(){}
void n(){
this.m();
}
public static void main(String args[]){
new A().n();
}}
```

```
class A{
void m(){System.out.println("hello
m");} void n(){
System.out.println("hello n");
//m();//same as this.m()
this.m();
}
}
class TestThis4{
public  static  void  main(String
args[]){ A a=new A(); a.n();
```

Test it Now

Output:

```
hello n
hello m
```

3) this() : to invoke current class constructor

The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

Calling default constructor from parameterized constructor:

```
ss A{
{System.out.println("hello
a");} nt x){ s();
stem.out.println(x);
ss TestThis5{
blic static void main(String args[]){
=new A(10);
```

Test it Now

Output:

```
hello a
10
```

Calling parameterized constructor from default constructor:

```
class A{
A(){
this(5);
System.out.println("hello a");
}
A(int x){
System.out.println(x);
}
}
class TestThis6{

public static void main(String args[]){
A a=new A();
}}
```

Test it Now

Output:

```
5
hello a
```

Real usage of this() constructor call

The this() constructor call should be used to reuse the constructor from the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining. Let's see the example given below that displays the actual use of this keyword.

```
class
Student{ int
rollno;
String name,course;
float fee;
Student(int rollno,String name,String course){
this.rollno=rollno;
this.name=name;
this.course=course;
}
Student(int rollno,String name,String course,float
fee){ this(rollno,name,course);//reusing constructor
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+course+" "+fee);}
}
class TestThis7{ public static void
main(String args[]){ Student s1=new
Student(111,"ankit","java"); Student s2=new
Student(112,"sumit","java",6000f);
s1.display(); s2.display();
}}
```

```
111 ankit java null
112 sumit java 600
```

Test it Now

Output:

Rule: Call to this() must be the first statement in constructor.

```
class Student{
int rollno;
String name,course;
float fee;
Student(int rollno,String name,String course){
this.rollno=rollno;
this.name=name;
this.course=course;
}
Student(int rollno,String name,String course,float fee){
this.fee=fee;
this(rollno,name,course);//C.T.Error
}
void display(){System.out.println(rollno+" "+name+" "+course+" "+fee);}
}
class TestThis8{
public static void main(String args[]){
Student s1=new Student(111,"ankit","java");
Student s2=new Student(112,"sumit","java",6000f);
s1.display();
s2.display();
}}
```

Test it Now

```
Compile Time Error: Call to this must be first statement  in  constructor
```

4) this: to pass as an argument in the method

The this keyword can also be passed as an argument in the method. It is mainly used in the event handling. Let's see the example:

```
class S2{
  void m(S2 obj){
  System.out.println("method is invoked");
  }
  voi
  d
  p()
  {
  m(this);
  }
  public static void main(String args[]){
  S2 s1 = new S2();
  s1.p();
  }
}
```

Test it Now

Output:

```
  method is  invoked
```

Application of this that can be passed as an argument:

In event handling (or) in a situation where we have to provide reference of a class to another one. It is used to reuse one object in many methods.

---

5) this: to pass as argument in the constructor call

We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes. Let's see the example:

```
class B{
  A4 obj;
  B(A4 obj){
    this.obj=obj;
  }
  void display(){
    System.out.println(obj.data);//using data member of A4 class
  }
}
class A4{
  int data=10;
  A4(){
   B b=new B(this); b.display(); }
  public static void main(String
  args[]){ A4 a=new A4();
  }
```

Test it Now

```
Output:10
```

---

6) this keyword can be used to return current class instance

We can return this keyword as an statement from the method. In such case, return type of the method must be the class type (non-primitive). Let's see the example:

Syntax of this that can be returned as a statement

```
return_type method_name(){
return this;
}
```

Example of this keyword that you return as a statement from the method

```
class A{
A
getA(){
return
this; }
void msg(){System.out.println("Hello java");}
} class
Test1{
public static void main(String args[]){
new A().getA().msg();
}
}
```

Test it Now

Output:

```
Hello java
```

Proving this keyword

Let's prove that this keyword refers to the current class instance variable. In this program, we are printing the reference variable and this, output of both variables are same.

```
class A5{
void m(){
System.out.println(this); //prints same reference ID
}
public static void main(String args[]){
A5 obj=new A5();
System.out.println(obj);//prints the reference ID
obj.m();
}
}
```

Test it Now

Output:

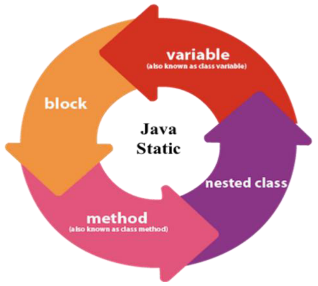
```
A5@22b3ea59
A5@22b3ea59
```

Java static keyword

The **static keyword** in Java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than an instance of the class.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested class



1) Java static variable

If you declare any variable as static, it is known as a static variable.

- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

Advantages of static variable

It makes your program **memory efficient** (i.e., it saves memory).

Understanding the problem without static variable

```
class Student
{
int rollno;
String name;
String college="ITS";
}
```

Suppose there are 500 students in my college, now all instance data members will get memory each time when the object is created. All students have its unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all objects. If we make it static, this field will get the memory only once.

Java static property is shared to all objects.

Example of static variable

```
//Java Program to demonstrate the use of static variable
class Student{
int rollno;//instance variable
String name;
static String college ="ITS";//static variable
//constructor
Student(int r, String n){
rollno = r;
name = n;
}
//method to display the values
void display (){System.out.println(rollno+" "+name+" "+college);}
}
//Test class to show the values of objects
public class TestStaticVariable1{
public static void main(String args[]){
Student s1 = new Student(111,"Karan");
Student s2 = new Student(222,"Aryan");
//we can change the college of all objects by the single line of code
//Student.college="BBDIT";
s1.display();
s2.display();
}
}
```

Test it Now

Output:

```
111  Karan  ITS
222  Aryan  ITS
```

Program of the counter without static variable

In this example, we have created an instance variable named count which is incremented in the constructor. Since instance variable gets the memory at the time of object creation, each object will have the copy of the instance variable. If it is incremented, it won't reflect other objects. So each object will have the value 1 in the count variable.

```
//Java Program to demonstrate the use of an instance variable
//which get memory each time when we create an object of the class.
class Counter{
int count=0;//will get memory each time when the instance is created
Count
er() {
count++;//incrementing value
    System.out.println(count);
}
public static void main(String args[]){
//Creating objects
Counter c1=new Counter();
Counter c2=new Counter();
Counter c3=new Counter();
}
}
```

Test it Now

Output:

```
1
1
1
```

Program of counter by static variable

As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

```
//Java Program to illustrate the use of static variable which
//is shared with all objects.
class Counter2{
static int count=0;//will get memory only once and retain its value
Counter2(){
count++;//incrementing the value of static variable
System.out.println(count);
}
public static void main(String args[]){
//creating objects
Counter2 c1=new Counter2();
Counter2 c2=new Counter2();
Counter2 c3=new Counter2();
}
}
```

Test it Now

Output:

```
1
```

```
2
3
```

2) **Java static method**

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

Example of static method

```
//Java Program to demonstrate the use of a static method.
class Student{
    int rollno;
    String
    name;
    static String college = "ITS";
    //static method to change the value of static
    variable static void change(){ college = "BBDIT";
    }
    //constructor to initialize the
    variable Student(int r, String n){
    rollno = r; name = n;
    }
    //method to display values
    void display(){System.out.println(rollno+" "+name+" "+college);}
}
//Test class to create and display the values of
object public class TestStaticMethod{ public
static void main(String args[]){
Student.change();//calling change method
  //creating objects
  Student s1 = new Student(111,"Karan");
  Student s2 = new Student(222,"Aryan");
  Student s3 = new Student(333,"Sonoo");
  //calling display
  method s1.display();
  s2.display();
  s3.display();
  }
}
```

Test it Now

```
Output:111 Karan BBDIT
     222 Aryan BBDIT
     333 Sonoo BBDIT
```

Another example of a static method that performs a normal calculation

```
//Java Program to get the cube of a given number using the static method
class Calculate{
static int cube(int x){
return x*x*x;
}

public static void main(String args[]){
int result=Calculate.cube(5);
System.out.println(result);
}
.}
```

Test it Now

```
Output:125
```

Restrictions for the static method

There are two main restrictions for the static method. They are:

1. The static method can not use non static data member or call non-static method directly.
2. this and super cannot be used in static context.

```
class A{
int a=40;//non
static
 public static void main(String args[]){
 System.out.println(a);
 }
 }
```

Test it Now

```
Output:Compile Time Error
```

Q) Why is the Java main method static?

Ans) It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.

3) Java static block

- Is used to initialize the static data member.
- It is executed before the main method at the time of classloading.

Example of static block

```
class A2{
 static{System.out.println("static block is invoked");}
 public static void main(String args[]){
 System.out.println("Hello main");
 }
}
```

Test it Now

```
Output:static block  is invoked
     Hello main
```

Q) Can we execute a program without main() method?

Ans) No, one of the ways was the static block, but it was possible till JDK 1.6. Since JDK 1.7, it is not possible to execute a java class without the main method.

```
class A3{
static{
System.out.println("static block is invoked");
System.exit(0);
}
}
```

Test it Now

Output:

```
static block is invoked
```

Since JDK 1.7 and above, output would be:

```
Error: Main method not found in class A3, please define the main method as:
   public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application
```

Access Modifiers in java

There are two types of modifiers in java: **access modifiers** and **non-access modifiers**.

The access modifiers in java specifies accessibility(scope) of a data member, method, constructor or class.
There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

1) private access modifier

The private access modifier is accessible only within class.

Simple example of private access modifier

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is compile time error.

```
clas
s A
{
private int data=40;
private void msg(){System.out.println("Hello java");
}
}
public class Simple
{
public static void main(String args[])
{
A obj=new A();
System.out.println(obj.data);//Compile Time Error
obj.msg();//Compile Time Error
}
}
```

Role of Private Constructor

If you make any class constructor private, you cannot create the instance of that class from outside the class. For example:

```
class A
{
private
A()
{}//private constructor
void msg(){System.out.println("Hello java");}
}
public class Simple{
 public static void main(String args[])
{
  A obj=new A();//Compile Time Error
 }
}
```

2) default access modifier

If you don't use any modifier, it is treated as **default** bydefault. The default modifier is accessible only within package.

Example of default access modifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

```
//save by A.java
package pack;
class A{
void msg(){System.out.println("Hello");}
}
//save by B.java
package
mypack; import
pack.*; class B{
public static void main(String
 args[]){ A obj = new A();//Compile
 Time Error obj.msg();//Compile Time
 Error
 }
}
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

3) protected access modifier

The **protected access modifier** is accessible within package and outside the package but through inheritance only.
The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

Example of protected access modifier

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

```
//save by A.java
package pack;
public class A
{
protected void msg(){System.out.println("Hello");
}
}
//save by B.java
package mypack;
import pack.*;
class B extends A
{ public static void main(String
 args[])
{
B obj = new B();
 obj.msg();
}
}
```

Output:Hello

4) public access modifier

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

Example of public access modifier

```
//save by A.java
package pack;
public class A
{ public void
msg()
{System.out.println("Hello");}
}
//save by B.java
package
mypack;
```

```
import
pack.*; class
B {
public static void main(String args[]){
A obj = new A();
 obj.msg();
}
}
```

Output:Hello

Understanding all java access modifiers

Let's understand the access modifiers by a simple table.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Java access modifiers with method overriding

If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.

```
class A{
protected void msg(){System.out.println("Hello java");}
}

   public class Simple extends A{
   void msg(){System.out.println("Hello java");}//C.T.Error
   public static void main(String args[]){
   Simple obj=new Simple();
   obj.msg();
   }
}
```

The default modifier is more restrictive than protected. That is why there is compile time error

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

extends Keyword

extends is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

Syntax

```
class Super { .....
    ..... } class Sub
extends Super { .....
    .....
}
```

Sample Code

Following is an example demonstrating Java inheritance. In this example, you can observe two classes namely Calculation and My_Calculation.

Using extends keyword, the My_Calculation inherits the methods addition() and Subtraction() of Calculation class.

Copy and paste the following program in a file with name My_Calculation.java

Example

Live Demo class

```
Calculation { int z;
    public void addition(int x, int y) { z = x
        + y;
        System.out.println("The sum of the given numbers:"+z);
    }
    public void Subtraction(int x, int y) { z = x
        - y;
        System.out.println("The difference between the given numbers:"+z);
    }
}
public class My_Calculation extends Calculation {
    public void multiplication(int x, int y) { z = x * y;
        System.out.println("The product of the given numbers:"+z);
    }
    public static void main(String args[]) { int a =
        20, b = 10;
        My_Calculation demo = new
        My_Calculation(); demo.addition(a, b);
        demo.Subtraction(a, b); demo.multiplication(a,
        b);
    }
}
```

Compile and execute the above code as shown below.

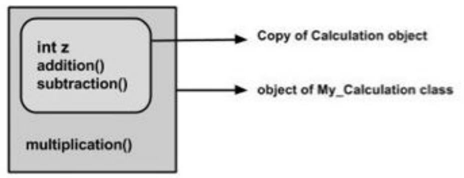
```
javac My_Calculation.java
java My_Calculation
```

After executing the program, it will produce the following result −

Output

```
The sum of the given numbers:30
The difference between the given numbers:10
The product of the given numbers:200
```

In the given program, when an object to **My_Calculation** class is created, a copy of the contents of the superclass is made within it. That is why, using the object of the subclass you can access the members of a superclass.



The Superclass reference variable can hold the subclass object, but using that variable you can access only the members of the superclass, so to access the members of both classes it is recommended to always create reference variable to the subclass.

If you consider the above program, you can instantiate the class as given below. But using the superclass reference variable (**cal** in this case) you cannot call the method **multiplication**(), which belongs to the subclass My_Calculation.

```
Calculation demo = new My_Calculation();
demo.addition(a, b); demo.Subtraction(a,
b);
```

Note − A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

The super keyword

The **super** keyword is similar to **this** keyword. Following are the scenarios where the super keyword is used.

- It is used to **differentiate the members** of superclass from the members of subclass, if they have same names.
- It is used to **invoke the superclass** constructor from subclass.

Differentiating the Members

If a class is inheriting the properties of another class. And if the members of the superclass have the names same as the sub class, to differentiate these variables we use super keyword as shown below.

```
super.variable
super.method();
```

Sample Code

This section provides you a program that demonstrates the usage of the **super** keyword.

In the given program, you have two classes namely *Sub_class* and *Super_class*, both have a method named display() with different implementations, and a variable named num with different values. We are invoking display() method of both classes and printing the value of the variable num of both classes. Here you can observe that we have used super keyword to differentiate the members of superclass from subclass.

Copy and paste the program in a file with name Sub_class.java.

Example

Live Demo class

```
Super_class  {  int
num = 20;
    // display method of superclass public
    void display() {
        System.out.println("This is the display method of superclass");
    }
}
public class Sub_class extends Super_class { int
    num = 10;
    // display method of sub class public
    void display() {
        System.out.println("This is the display method of subclass");
    }
    public void my_method() {
        // Instantiating subclass
        Sub_class sub = new Sub_class();
        // Invoking the display() method of sub class sub.display();
        // Invoking the display() method of superclass super.display();
        // printing the value of variable num of subclass
        System.out.println("value of the variable named num in sub class:"+ sub.num);
        // printing the value of variable num of superclass
        System.out.println("value of the variable named num in super class:"+ super.num);
    } public static void main(String args[]) {
    Sub_class obj = new Sub_class();
    obj.my_method();
    }
}
```

Compile and execute the above code using the following syntax.

```
javac Super_Demo
java Super
```

On executing the program, you will get the following result −

Output

```
This is the display method of subclass This is the
display method of superclass value of the variable
named num in sub class:10 value of the variable
named num in super class:20
```

Invoking Superclass Constructor

If a class is inheriting the properties of another class, the subclass automatically acquires the default constructor of the superclass. But if you want to call a parameterized constructor of the superclass, you need to use the super keyword as shown below. super(values); **Sample Code**

The program given in this section demonstrates how to use the super keyword to invoke the parametrized constructor of the superclass. This program contains a superclass and a subclass, where the superclass contains a parameterized constructor which accepts a integer value, and we used the super keyword to invoke the parameterized constructor of the superclass.

Copy and paste the following program in a file with the name Subclass.java

Example

Live Demo class

```
Superclass  {  int
age;
    Superclass(int age) { this.age
        = age;
    }
    public void getAge() {
            System.out.println("The value of the variable named age in super class is: " +age);
    }
}
public class Subclass extends Superclass {
    Subclass(int age) { super(age);
    } public static void main(String argd[]) {

    Subclass s = new Subclass(24); s.getAge();

    }
}
```

Compile and execute the above code using the following syntax.

```
javac Subclass
java Subclass
```

On executing the program, you will get the following result −

Output

```
The value of the variable named age in super class is: 24
```

IS-A Relationship

IS-A is a way of saying: This object is a type of that object. Let us see how the **extends** keyword is used to achieve inheritance.

```
public class Animal {
}
public class Mammal extends Animal {
}
public class Reptile extends Animal {
}
public class Dog extends Mammal {
}
```

Now, based on the above example, in Object-Oriented terms, the following are true −

- Animal is the superclass of Mammal class.
- Animal is the superclass of Reptile class.
- Mammal and Reptile are subclasses of Animal class.
- Dog is the subclass of both Mammal and Animal classes.

Now, if we consider the IS-A relationship, we can say −

- Mammal IS-A Animal
- Reptile IS-A Animal
- Dog IS-A Mammal
- Hence: Dog IS-A Animal as well

With the use of the extends keyword, the subclasses will be able to inherit all the properties of the superclass except for the private properties of the superclass.
We can assure that Mammal is actually an Animal with the use of the instance operator.

Example

Live Demo

```
class
Animal {
} class Mammal extends
Animal { }
class Reptile extends Animal {
} public class Dog extends
Mammal {
    public static void main(String args[]) {
        Animal a = new Animal();
        Mammal m = new Mammal();
        Dog d = new Dog();
        System.out.println(m instanceof Animal);
        System.out.println(d instanceof Mammal);
        System.out.println(d instanceof Animal); }
}
```

This will produce the following result −

Output

```
true
true
true
```

Since we have a good understanding of the **extends** keyword, let us look into how the **implements** keyword is used to get the IS-A relationship.
Generally, the **implements** keyword is used with classes to inherit the properties of an interface. Interfaces can never be extended by a class.

Example

```
public interface Animal {
}
public class Mammal implements Animal {
}
public class Dog extends Mammal {
}
```

The instanceof Keyword

Let us use the **instanceof** operator to check determine whether Mammal is actually an Animal, and dog is actually an Animal.

Example

Live Demo interface Animal{}

```
class    Mammal    implements

Animal{} public class Dog extends

Mammal {

    public static void main(String args[]) {
        Mammal m = new Mammal();
        Dog d = new Dog();
        System.out.println(m instanceof Animal);
        System.out.println(d instanceof Mammal);
        System.out.println(d instanceof Animal); }
}
```

This will produce the following result −

Output

```
true
true
true
```

HAS-A relationship

These relationships are mainly based on the usage. This determines whether a certain class **HAS-A** certain thing. This relationship helps to reduce duplication of code as well as bugs.
Lets look into an example −

Example

```
public class Vehicle{} public
class Speed{}
public class Van extends Vehicle { private
    Speed sp;
}
```

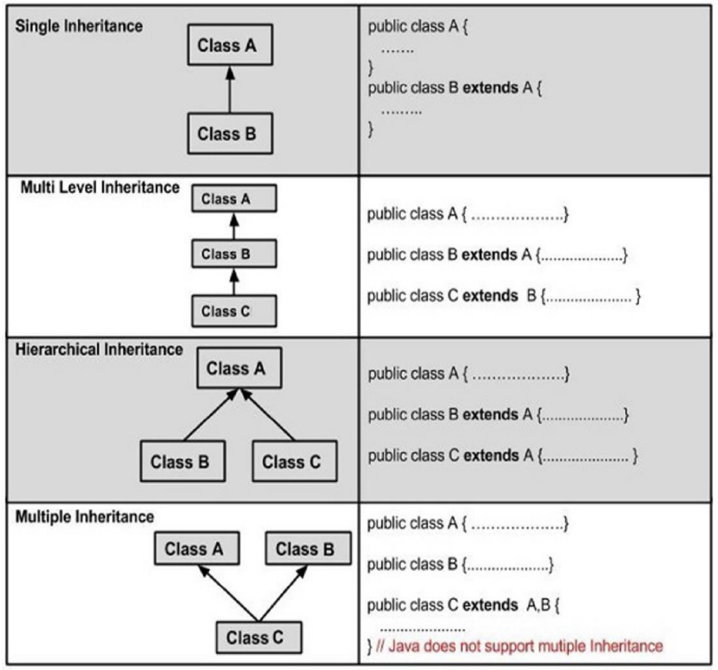
This shows that class Van HAS-A Speed. By having a separate class for Speed, we do not have to put the entire code that belongs to speed inside the Van class, which makes it possible to reuse the Speed class in multiple applications.
In Object-Oriented feature, the users do not need to bother about which object is doing the real work. To achieve this, the Van class hides the implementation details from the users of the Van class. So, basically what happens is the users would ask the Van class to do a certain action and the Van class will either do the work by itself or ask another class to perform the action.

Types of Inheritance

There are various types of inheritance as demonstrated below.

A very important fact to remember is that Java does not support multiple inheritance. This means that a class cannot extend more than one class. Therefore following is illegal −
Example public class extends Animal, Mammal{}
However, a class can implement one or more interfaces, which has helped Java get rid of the impossibility of multiple inheritance.

Inheritance (IS-A)
Inheritance is one of the key features of Object Oriented Programming. Inheritance provided mechanism that allowed **a class to inherit property of another class**. When a Class extends another class it inherits all non-private members including fields and methods. Inheritance in Java



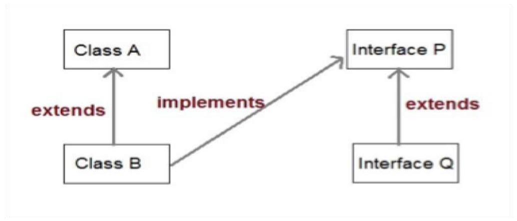
```
public class A {
  .......
}
public class B extends A {
  .........
}
```

```
public class A { ...................}
public class B extends A {...................}
public class C extends  B {..................... }
```

```
public class A { ...................}
public class B extends A {...................}
public class C extends A {..................... }
```

```
public class A { ...................}
public class B {...................}
public class C extends  A,B {
  .....................
} // Java does not support mutiple Inheritance
```

can be best understood in terms of Parent and Child relationship, also known as **Super class**(Parent) and **Sub class**(child) in Java language.
Inheritance defines **is-a** relationship between a Super class and its Sub class. extends and implements keywords are used to describe inheritance in Java.



Let us see how **extends** keyword is used to achieve Inheritance.

```
class Vehicle.
{
   ......
}
class Car extends Vehicle
{
  .......   //extends the property of vehicle class.
}
```

Now based on above example. In OOPs term we can say that,

- **Vehicle** is super class of **Car**.
- **Car** is sub class of **Vehicle**.
- Car IS-A Vehicle.

Purpose of Inheritance

1. It promotes the code reusabilty i.e the same methods and variables which are defined in a parent/super/base class can be used in the child/sub/derived class.
2. It promotes polymorphism by allowing method overriding.

Disadvantages of Inheritance
Main disadvantage of using inheritance is that the two classes (parent and child class) gets **tightly coupled**. This means that if we change code of parent class, it will affect to all the child classes which is inheriting/deriving the parent class, and hence, **it cannot be independent of each other**.

Simple example of Inheritance

```
class Parent
{ public void p1()
          {
          System.out.println("Parent method"); }
}
public class Child extends Parent { public
          void c1()
          {
          System.out.println("Child method");
          }
          public static void main(String[] args)
          {
          Child cobj = new Child(); cobj.c1();
          //method of Child class cobj.p1();
          //method of Parent class
          }
}
```

Output-Child method Parent method

Another example of Inheritance

```
class Vehicle
{
          String vehicleType;
}
public class Car extends Vehicle {
          String modelType;
          public void
          showDetail()
          {
          vehicleType = "Car";       //accessing Vehicle class member modelType =
          "sports";
          System.out.println(modelType+" "+vehicleType);
          } public static void main(String[] args)
          {
          Car car =new Car(); car.showDetail();
          }
}
```

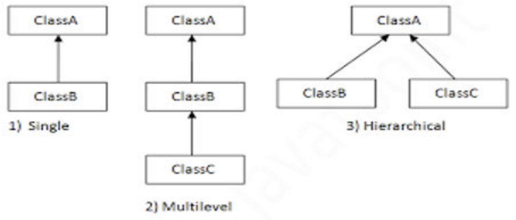
Output-sports Car

Types of Inheritance

1. Single Inheritance
2. Multilevel Inheritance

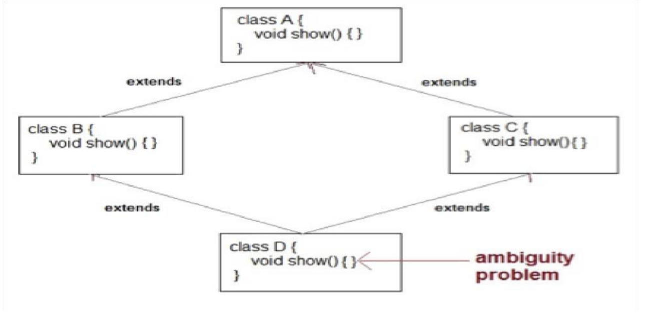
3. Heirarchical Inheritance

NOTE : Multiple inheritance is not supported in java



Why multiple inheritance is not supported in Java

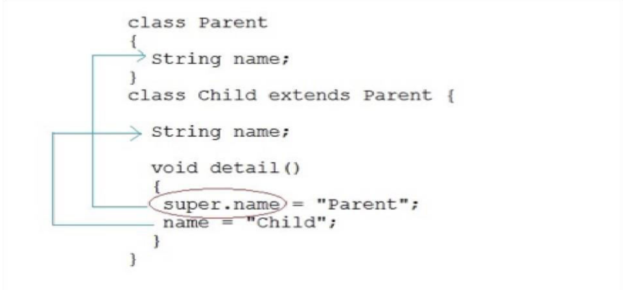
- To remove ambiguity.
- To provide more maintainable and clear design.



```
class A {
   void show() {}
}
class B {
   void show() {}
}
class C {
   void show(){}
}
class D {
   void show(){}
}
```

super keyword

In Java, super keyword is used to refer to immediate parent class of a child class. In other words **super** keyword is used by a subclass whenever it need to refer to its immediate super class.



```
class Parent
{
  String name;
}
class Child extends Parent {

  String name;

  void detail()
  {
   super.name = "Parent";
   name = "Child";
  }
}
```

Example of Child class reffering Parent class property using super keyword

```
class Parent
{
	String name;
}
public class Child extends Parent {
	String name; public void
	details()
	{
	super.name = "Parent"; //refers to parent class member name =
	"Child";
	System.out.println(super.name+" and "+name);
	} public static void main(String[] args){
	Child cobj = new Child(); cobj.details();
	}}
```

Output-Parent and Child

Example of Child class refering Parent class methods using super keyword

```
class Parent{
	String name; public
	void details(){
	name = "Parent";
	System.out.println(name);}}
public class Child extends Parent {
	String name; public void
	details()
	{ super.details();   //calling Parent class details() method name =
	"Child";
	System.out.println(name);
	} public static void main(String[] args){
	Child cobj = new Child(); cobj.details();
	}}
```

Output-Parent Child

Example of Child class calling Parent class constructor using super keyword

```
class Parent{
	String name;
	public Parent(String n)
	{
	name
	= n;
	}}
public class Child extends Parent { String
	name;
	public Child(String n1, String n2){
	super(n1); //passing argument to parent class constructor this.name = n2;
	}
	public void details(){
	System.out.println(super.name+" and "+name);
	} public static void main(String[] args){
	Child cobj = new Child("Parent","Child"); cobj.details();
	}}
```

Parent and Child

Note: When calling the parent class constructor from the child class using super keyword, super keyword should always be the first line in the method/constructor of the child class.

Super class reference pointing to Sub class object.

In context to above example where Class B extends class A.

A a=new B();

is legal syntax because of IS-A relationship is there between class A and Class B.

Q. Can you use both this() and super() in a Constructor?

NO, because both super() and this() must be first statement inside a constructor. Hence we cannot use them together.

Day 1 (Basic Programming)

Question No. 1: -

Read the radius and print the area of a circle.

Code:

```
import java.util.Scanner; public class Areaofcircle {      public
static void main(String[] args) {
          System.out.println("Bijan Shrestha\n");
          System.out.println("Reg no. 20BCE2904\n");
     Scanner s= new Scanner(System.in);
System.out.println("Enter the radius:");
double r= s.nextDouble();        d
ouble  area=(22*r*r)/7 ;
      System.out.println("Area of Circle is: " + area);
   }
}
```

Output:

Question No. 2: -

Read the number and check whether it is divisible by 3 and 5.

Code:

```
import java.util.Scanner; public class
Check_divisibility {
   public static void main(String[] args) {
                     System.out.println("Bijan Shrestha\n");
System.out.println("Reg no. 20BCE2904\n");
 int n;
             Scanner s = new Scanner(System.in);
 System.out.print("Enter any number:");
n = s.nextInt();
             if((n % 5 == 0) & (n % 3 == 0))
             { System.out.println(n+" is divisible by 3 and  5");
             }
else{
                System.out.println(n+" is not divisible by 3 and 5");
             }
       }      }
```

Output:

Question No. 3: -

Display Subject Name based on room number. If the user enters 604 then display Java Programming, if the user enters 605 then display Python programming for any other input display Invalid input to the user.

Code:

```
import java.util.Scanner;
public class Subject { public static void main(String[] args) {
                   System.out.println("Bijan Shrestha\n");
System.out.println("Reg no. 20BCE2904\n");
Scanner s = new Scanner(System.in);
                   System.out.println("Enter the room number:");
                   int n = s.nextInt();
                   if (n == 604) {
                            System.out.println("Java Programming");
                            }
                   else if (n == 605) {
                            System.out.println("Python programming");
                            }
                   else {
                            System.out.println("Invalid input");
                            } } }
```

Output:

Print the sum of first n numbers. If n is 3 then print the sum of 1+2+3 to the user. Get n from the user

Code:

```
import java.util.Scanner; public class Sum {
          public static void main(String[] args) {
                   System.out.println("Bijan Shrestha\n");
                   System.out.println("Reg no. 20BCE2904\n");
                   int num, count, total = 0;
         System.out.println("Enter the value of n:");
Scanner scan = new Scanner(System.in);                  n
um = scan.nextInt();
scan.close();
         for(count = 1; count <= num; count++){
                  total = total + count;
                  }
                  System.out.println("Sum of first "+num+" natural numbers is: "+total);
         }
```

Print the sum of the series $1^2 + 2^2 + 3^2$ up to n terms.

Code:

```
import java.util.Scanner;
 public class Squaresum{
                  System.out.println("Bijan Shrestha\n");
                           System.out.println("Reg no. 20BCE2904\n");
public static void main(String [] args){
                  Scanner sc=new Scanner(System.in);
                  System.out.println("Enter the number of series for sum:");
                  int num=sc.nextInt(); int sum=0;
                  for(int i=1;i<=num;i++)
                  sum+=(i*i);
                  System.out.println("sum="+ sum);
```

Output:

Print the multiplication table by getting the n from the user.

Code:

```
import java.util.Scanner; public class Multiplication_Table {
public static void main(String[] args) {
         System.out.println("Bijan Shrestha\n");
         System.out.println("Reg no. 20BCE2904\n");
     Scanner s = new Scanner(System.in);     System.out.print("Enter
number:");              int n=s.nextInt();
      for(int i=1; i <= 10; i++)
      {
            System.out.println(n+" * "+i+" = "+n*i);
          }
      }
  }
```

Output:

Provide the option of adding two numbers to the user until the user wants to exit.

Code:

```
import java.util.Scanner;
public class Twosum { public static void main(String [] args){
                System.out.println("Bijan Shrestha\n");
                System.out.println("Reg no. 20BCE2904\n");
                Scanner sc=new Scanner(System.in);
                System.out.println("enter 1 to continue and 0 to stop"); int choice=sc.nextInt();
                while(choice==1)
                {
                        System.out.println("enter 1 numbers"); int a=sc.nextInt();
                        System.out.println("enter 2 numbers");
                        int b=sc.nextInt();
                        System.out.println("sum="+ (a+b));
                        System.out.println("enter 1 to continue and 0 to stop"); choice=sc.nextInt();
                } } }
```

Question No. 8: -

Print this pattern for n lines.

```
    a) *
       **
       ***
       ****
```

Code:

```
import java.util.Scanner; public class pattern1   {
 public static void main(String args[])  {
System.out.println("Bijan Shrestha\n");
System.out.println("Reg no. 20BCE2904\n");
Scanner scanner = new Scanner(System.in);
System.out.println("Enter number of row' : ");
 int row = scanner.nextInt();

; i++)  {
j++)   {

.out.print("* ");

.out.println();
 }
```

Output:

```
    b) 1234
       123
       12
       1
```

Code:

```
import java.util.Scanner; public class pattern2   {
 public static void main(String args[])   {
                                          System.out.println("Bijan Shrestha\n");
                         System.out.println("Reg no. 20BCE2904\n");
Scanner scanner = new Scanner(System.in);
System.out.println("Enter number of row' : ");
int row = scanner.nextInt();
                int i, j;
                for(i=row; i>=1; i--)   {
                 for(j=1; j<=i; j++)   {
                System.out.print(j );
                                }
                        System.out.println();
                        }}}
```

Output:

```
    c) 1
       12
       123
       1234
       1234
       123
       12
       1
```

Code:

```
import java.util.Scanner; public class pattern3   {
  public static void main(String args[])   {
                System.out.println("Bijan Shrestha\n");
                        System.out.println("Reg no. 20BCE2904\n");
 Scanner scanner = new Scanner(System.in);
System.out.println("Enter number of row' : ");
int row = scanner.nextInt();
                int i, j;
                for (i=1;i<=row;i++) {
                 for(j=1;j<=i;j++){
System.out.print(j);
                         }
                System.out.println();
                }
                for(i=row; i>=1; i--)   {
                 for(j=1; j<=i; j++)   {
                                System.out.print(j );
                         }
                System.out.println();
 }   }}
```

Output:

Day 2 (Programming based on Arrays Part 1)

Question No. 1: -

Sort an array of element using bubble sort.

Code:

```
import java.util.Scanner; public class
BubbleSort {
    static void bubbleSort(int[] arr) {
        System.out.println("Bijan Shrestha\n");
System.out.println("Reg no. 20BCE2904\n");
int n = arr.length;
        int temp = 0;
for(int i=0; i < n; i++){
for(int j=1; j < (n-i); j++){
  if(arr[j-1] > arr[j]){
 temp = arr[j-1];
arr[j-1] = arr[j];
arr[j] = temp;
                }
            }
        }
    }
    public static void main(String[] args) {
 int n, i,j=0;
        Scanner s = new Scanner(System.in);
        System.out.print("Enter number of elements in the array:");
 n = s.nextInt();
        int arr[] = new int[n];
        System.out.println("Enter "+n+" elements ");
 for( i=0; i < n; i++){
            arr[i] = s.nextInt();
        }
        System.out.println();
        bubbleSort(arr);//sorting array elements using bubble sort
        System.out.println("Array After Bubble Sort");
        for(i=0; i < arr.length; i++){
System.out.print(arr[i] + " ");
        }

    } } Output:
```

Question No. 2: -

Remove duplicate elements from a sorted array

Code:

```
import java.util.Arrays;
 import  java.util.Scanner;
class Duplicate
{
    public static int DuplicateRemoval(int array[], int n){
if (n==0 || n==1){
            return n;
        }        int k = 0;
 for (int i=0; i < n-1; i++){
  if (array[i] != array[i+1]){
              array[k++] = array[i];
          }
        }
        array[k++] = array[n-1];
        return k;
    }
    public static void main(String args[])
    {
        System.out.println("Bijan Shrestha\n");
System.out.println("Reg no. 20BCE2904\n");
 int n, i,j=0;
        Scanner s = new Scanner(System.in);
        System.out.print("Enter number of elements in the array:");
n = s.nextInt();
        int array[] = new int[n];
        System.out.println("Enter "+n+" elements ");
for( i=0; i < n; i++)
        {
            array[i] = s.nextInt();
        }
        Arrays.sort(array);
int return_value= DuplicateRemoval(array, array.length);
 System.out.println("Sorted Array after removing the Duplicate
Elements:");
        for (i=0; i<return_value; i++)
System.out.print(array[i]+" ");
    }
}
```

Output:

Question No. 3: -

Reverse the contents inside an array

Code:

```
import java.util.Scanner; class reverseArray{
    public static void main(String[] args)
    {
        System.out.println("Bijan Shrestha\n");
System.out.println("Reg no. 20BCE2904\n");
int n, i,j=0;
        Scanner s = new Scanner(System.in);
        System.out.print("Enter number of elements in the array:");
  n = s.nextInt();
 int array[] = new int[n];
int rev[] = new int[n];
        System.out.println("Enter "+n+" elements ");
for( i=0; i < n; i++) {
            array[i] = s.nextInt();
        }
        System.out.println("Reverse of an array is :");
 for( i=n;i>0 ; i--,j++){
rev[j] = array[i-1];
            System.out.print(rev[j]+" ");} }}
```

Output:

Question No. 4: -

Search for an element inside the array using linear search

Code:

```
import java.util.Scanner; class location {
   public static void main(String[]args)
   {
      System.out.println("Bijan Shrestha\n");
System.out.println("Reg no. 20BCE2904\n");
 int n,num, i,j=0;
      Scanner s = new Scanner(System.in);
      System.out.print("Enter number of elements in the array:");
 n = s.nextInt();
 int array[] = new int[n];
      System.out.println("Enter "+n+" elements ");
  for( i=0; i < n; i++)
{
         array[i] = s.nextInt();
      }
      System.out.print("Enter the element to be searched: ");
 num=s.nextInt();
for(i=0;i<n;i++){
         if  (array[i]==num) {
System.out.println("The number is present at the index of " + i);
            j++;
         }
}
      if (j==0){
         System.out.println("The number is not in present in the array");
      }
   }

}
```

Output

Day 3 (Programming based on Arrays Part 2)

Question No. 1: -

Write a Java program to sort an array of positive integers of an given array, in the sorted array the value of the first element should be maximum, second value should be minimum value, third should be second maximum, fourth second be second minimum and so on.

code

```
import java.util.Arrays; import java.util.Scanner;
public class maximini
{
   static int[] rearrange(int[] new_arra, int n)
   {
      int temp[] = new int[n];
      int small_num = 0, large_num = n-1;
boolean flag = true;
      for (int i=0; i < n; i++)
      {
         if (flag)
            temp[i] = new_arra[large_num--];
 else
            temp[i] = new_arra[small_num++];
flag = !flag;
      }
      return temp;


   }
   public static void main(String[] args)
   {
      System.out.println("Bijan Shrestha\n");
      System.out.println("Reg no. 20BCE2904\n");
      Scanner input=new Scanner(System.in);
      System.out.println("Enter the length of the array");
 int len=input.nextInt();
 int arr[]=new int[len];
      int temp[];
      System.out.println("Enter the element");
for(int i=0;i<len;i++){
         arr[i]=input.nextInt();
      }
      int result[];
      System.out.println("Original Array Given by user");
      System.out.println(Arrays.toString(arr));
result = rearrange(arr,arr.length);
      System.out.println("New Array after sorting ");
      System.out.println(Arrays.toString(result));

   }
}
```

Output:

Question No. 2: -

Write a Java program to separate even and odd numbers of a given array of integers. Put all even numbers first, and then odd numbers. **Code:**

```
import java.util.*;
class EvenOdd {
   public static void main(String args[])
   {
         System.out.println("Bijan Shrestha\n");
           System.out.println("Reg no. 20BCE2904\n");
      Scanner sc=new Scanner(System.in);
 System.out.println("Enter the range");
      int no=sc.nextInt();
      int a[]=new int[no];
 int i;
      System.out.println("Enter the elements");
 for(i=0;i<no;i++) {
         a[i]=sc.nextInt();
```

```
        }
        System.out.println("Even numbers are");
 for(i=0;i<no;i++) {
            if(a[i]%2==0) {
               System.out.println(a[i]);
            }
        }
        System.out.println("Odd numbers are");
for(i=0;i<no;i++) {
            if(a[i]%2!=0)
            {
               System.out.println(a[i]);
            } }}}
```

Output:

Question No. 3: -

Write a Java program to remove the duplicate elements of a given array and return the new length of the array.

Code:

```
import java.util.Arrays; import
java.util.Scanner; class duplength
{
    public static int DuplicateRemoval(int array[], int n){
 if (n==0 || n==1){
return n;
        }
 int k = 0;
 for (int i=0; i < n-1; i++){
 if (array[i] != array[i+1]){
               array[k++] = array[i];
            }
        }
        array[k++] = array[n-1];
        return k;
    }
    public static void main(String args[])  {
            System.out.println("Bijan Shrestha\n");
            System.out.println("Reg no. 20BCE2904\n");
        int n, i,j=0;
        Scanner s = new Scanner(System.in);
        System.out.print("Enter number of elements in the array:");
n = s.nextInt();
        int array[] = new int[n];
        System.out.println("Enter "+n+" elements ");
for( i=0; i < n; i++)
        {
            array[i] = s.nextInt();
        }
        Arrays.sort(array);
  int return_value= DuplicateRemoval(array, array.length);
 System.out.print("The new length of array is: "+return_value);
    }
}
```

Output:

Question No. 4

Write a Java program to find the sum of the two elements of a given array which is equal to a given integer.

Code:

```
import java.util.Arrays; import
java.util.Scanner; public class SumCheck{
    public static void main(String[] args)
    {
        System.out.println("Bijan Shrestha\n");
        System.out.println("Reg no. 20BCE2904\n");
        Scanner input=new Scanner(System.in);
        System.out.println("Enter the length of the array");
  int len=input.nextInt();
        System.out.println("Enter the original sum");
  int org_sum=input.nextInt();
        int arr[]=new int[len];
        System.out.println("Enter the element");
        for(int i=0;i<len;i++){
            arr[i]=input.nextInt();
        }
        System.out.println("Original Array Given by user");
        System.out.println(Arrays.toString(arr));
int sum=0,chk=0;
        for (int i = 0; i < arr.length-1; i++) {
            for (int j = i+1; j < arr.length; j++) {
                if (arr[i] + arr[j] == org_sum) {
                    System.out.println("Pair found (" + arr[i] + ", " + arr[j] + ")");
chk++;
                } } }
        if (chk==0) {
            System.out.println("Pair not found");
        } } }
```

Output:

Question No. 5

Display the sum of rows in a matrix. **Code:**

```
import java.util.Scanner;
public class Rowsum
{
    public static void main(String args[]) {
System.out.println("Bijan Shrestha\n");
System.out.println("Reg no. 20BCE2904\n");
int i, j;
        System.out.println("Enter total rows and columns: ");
        Scanner s = new Scanner(System.in);
        int row = s.nextInt();
 int column = s.nextInt();
int arr[][] = new int[row][column];
System.out.println("Enter matrix:");
        for (i = 0; i < row; i++) {
for (j = 0; j < column; j++) {
arr[i][j] = s.nextInt();
System.out.print("");
          }
        }
        System.out.println("The matrix is ");
        for (i = 0; i < row; i++) {
for (j = 0; j < column; j++) {
System.out.print(arr[i][j] + " ");
          }
          System.out.println(" ");
        }
        for(i = 0; i < arr.length; i++) {
          int sum = 0;
          for(j = 0; j < arr[0].length; j++) {
              sum = sum + arr[i][j];
          }
          System.out.println("\nThe Sum of Row Matrix Items is: " + i + " row = " + sum);
        } } }
```

Question No. 6

Display the transpose of a matrix.

Code:

```
import java.util.Scanner;
public class transpose {
    public static void main(String args[]) {
            System.out.println("Bijan Shrestha\n");
             System.out.println("Reg no. 20BCE2904\n");
        int i, j;
        System.out.println("Enter total rows and columns: ");
        Scanner s = new Scanner(System.in);
        int row = s.nextInt();
int column = s.nextInt();
 int arr[][] = new int[row][column];
System.out.println("Enter matrix:");
        for(i = 0; i < row; i++) {
          for(j = 0; j < column; j++) {
                arr[i][j] = s.nextInt();
System.out.print("");
          }
        }
        System.out.println("The matrix before Transpose is ");
for(i = 0; i < row; i++) {
          for(j = 0; j < column; j++) {
              System.out.print(arr[i][j]+" ");
          }
          System.out.println(" ");
        }
        System.out.println("The matrix after Transpose is ");
 for(i = 0; i < column; i++) {
          for(j = 0; j < row; j++) {
              System.out.print(arr[j][i]+" ");
          }
          System.out.println(" ");
```

Day 4 (Program based on classes)

Question No. 1: -

Design a class named Rectangle to represent a rectangle. The class contains:
Two double data fields named width and height that specify the width and height of the rectangle. The default values are 1 for both width and height.

i. A default constructor that creates a default rectangle.
ii. A constructor that creates a rectangle with the specified width and height.
iii. A method named getArea() that returns the area of this rectangle.
iv. A method named getPerimeter() that returns the perimeter.

Implement the class. Write a test program that creates two Rectangle objects— one with width 5 and height 50 and the other with width 2.5 and height 45.7.
Display the width, height, area, and perimeter of each rectangle in this order.

Code:

```
import java.util.*;
class Rectangle {
    double height, width;
    Rectangle() {
        height = 1;
        width = 1;
    }
    Rectangle(double h, double w) {
        height = h;
        width = w;
    }
    void displayHW() {
        System.out.println("The height and width of rectange is:" + height + "," + width);
    }
    void getArea() {
        System.out.println("The area of rectangle is: " + height * width);
    }
    void getPerimenter() {
        System.out.println("The Perimeter of rectangle is: " + 2 * (height + width));
    }
}
public class Rectangle1 {
    public static void main(String[] args) {
        System.out.println("Bijan Shrestha\n");
        System.out.println("Reg no. 20BCE2904\n");
        Rectangle R1 = new Rectangle();
        R1.displayHW();
        R1.getArea();
        R1.getPerimenter();
        Scanner value = new Scanner(System.in);
        for (int i = 0; i < 2; i++) {
            System.out.println("Enter height and width of rectangle");
            double H = value.nextDouble();
            double W = value.nextDouble();
            Rectangle R2 = new Rectangle(H, W);
            // printing area and perimeter
            R2.displayHW();
            R2.getArea();
            R2.getPerimenter();
        }  }}
```

Question No. 2: -

Write a Java program to create a class called Student having data members Regno, Name, Course being studied and current CGPA. Include constructor to initialize objects. Create array of objects with at least 10 students and find 9- pointers.

Code:

```
import java.util.*;
class Student {
    String regnum;
    String Name;
    String Course;
    double cgpa;
    Student() {
        regnum = "20BCE2904";
        Name = "Bijan Shrestha";
        Course = "B.tech CSE";
        cgpa = 9.3;
    }
    Student(String re_n, String nam, String course, double gpa) {
        regnum = re_n;
        Name = nam;
        Course = course;
        cgpa = gpa;
    }
}
class studentdata {
    public static void main(String[] args) {
        int chk = 0;
        Scanner s = new Scanner(System.in);
        Scanner n = new Scanner(System.in);
        System.out.println("Enter the number of Students to be stored: ");
        int num = n.nextInt();
        Student s1[] = new Student[num];
        for (int i = 0; i < num; i++) {
            System.out.print("For Student " + (i + 1));
            System.out.println();
            System.out.print("Name: ");
            String st_name = s.nextLine();
            System.out.print("Registration no.: ");
            String reg = s.nextLine();
            System.out.print("Course id: ");
            String c_id = s.nextLine();
            System.out.print("CGPA: ");
            double cgpa = s.nextDouble();
            System.out.println();
            s1[i] = new Student(reg, st_name, c_id, cgpa);
            String ch = s.nextLine();
        }
        System.out.println("List of 9 pointers: ");
        for (int i = 0; i < num; i++) {
            if (s1[i].cgpa >= 9) {
                System.out.println("Name:  " + s1[i].Name + "(" + s1[i].regnum + ")");
                chk++;
            }
        }
        if (chk == 0) {
            System.out.println("Null");
        }}}
```

Question No. 3: -

Write a Java program that displays that displays the time in different formats in the form of HH,MM,SS using constructor Overloading.

Code:

```
import java.util.*;
import java.time.format.DateTimeFormatter;
import java.time.LocalDateTime;
public class timeformat {
    LocalDateTime date = LocalDateTime.now();
    DateTimeFormatter t1 = DateTimeFormatter.ofPattern("HH");
    DateTimeFormatter t2 = DateTimeFormatter.ofPattern("mm");
    DateTimeFormatter t3 = DateTimeFormatter.ofPattern("ss");
    String hrs = date.format(t1);
    String mins = date.format(t2);
    String sec = date.format(t3);
    timeformat(int n) {
        System.out.println("HH:mm:ss = " + hrs + ":" + mins + ":" + sec);
    }
    timeformat() {
        int Hour = Integer.parseInt(hrs);
        int Minute = Integer.parseInt(mins);
        int Second = Integer.parseInt(sec);
        if (Hour > 12) {
            Hour = Hour - 12;
        }
        System.out.println("HH:mm:ss = " + Hour + ":" + Minute + ":" + Second);
    }
    public static void main(String args[]) {
        System.out.println("Bijan Shrestha\n");
        System.out.println("Reg no. 20BCE2904\n");
        System.out.print("Time in 24 hour format: ");
        timeformat hrs = new timeformat(1);
        System.out.print("Time in 12 hour format: ");
        timeformat completetime = new timeformat();
    }
}
```

Day 5 (Program based on jagged array)

Question No. 1: -

Write a program to demonstrate the knowledge of students in multidimensional arrays and looping constructs. Eg., If there are 4 batches in BTech - “CSE1007” course, read the count of the slow learners (who have scored <25) in each batch. Tutors should be assigned in the ratio of 1:4 (For every 4 slow learners, there should be one tutor). Determine the number of tutors for each batch. Create a 2-D jagged array with 4 rows to store the count of slow learners in the 4 batches. The number of columns in each row should be equal to the number of groups formed for that particular batch (Eg., If there are 23 slow learners in a batch, then there should be 6 tutors and in the jagged array, the corresponding row should store 4, 4, 4, 4, 4,3). Use for-each loop to traverse the array and print the details. Also print the number of batches in which all tutors have exactly 4 students.

Code:

```
import java.util.*;
public class jaggedarray {
   public static void main(String[] args) {
      System.out.println("Bijan Shrestha\n");
      System.out.println("Reg no. 20BCE2904\n");
      int i, j;
      double t;
      // Declaring 2-D array with 4 rows
      int arr[][] = new int[4][];
      // input for each batch
      Scanner sc = new Scanner(System.in);
      for (i = 0; i < arr.length; i++) {
         System.out.print("Enter number of slow learner for batch " + (i + 1) + ": ");
         t = sc.nextDouble();
         arr[i] = new int[(int) Math.ceil(t / 4)];
         for (j = 0; j < arr[i].length; j++) {
            if (t >= 4)
               arr[i][j] = 4;
            else
               arr[i][j] = (int) t;
            t = t - 4;
         }
      }
      sc.close();
      // Displaying the values of 2D Jagged array
      int cfour = 0;
      System.out.println("The Contents of 2D Jagged Array are :");
      for (i = 0; i < arr.length; i++) {
         for (j = 0; j < arr[i].length; j++) {
            System.out.print(arr[i][j] + " ");
            if (arr[i][j] == 4)
               cfour++;
         }
         System.out.println();
      }
      System.out.println("Number of tutors with 4 students are: " + cfour);
   }
}
```

Day 5 (Program based on String)

Question No. 1: -

Write a java Program to check whether given string is palindrome or not.

Code:

```
import java.util.Scanner;

class palindrome
{
  public static void main(String args[])
  {
    System.out.println("Bijan Shrestha\n");
    System.out.println("Reg no. 20BCE2904\n");
    String str, rev = "";
    Scanner ch = new Scanner(System.in);
    System.out.println("Enter a string to check:");
    str = ch.nextLine();
    int length = str.length();
    for ( int i = length - 1; i >= 0; i-- )
      rev = rev + str.charAt(i);
    if (str.equals(rev)){
       System.out.println(str+" is a palindrome");
    }
  else{
      System.out.println(str+" is not a palindrome");
  }}}
```

Question No. 2: -

Write a Java program to sort a string array in ascending order.
Input the string: hello world welcome to vit
Expected Output: cdeeehillllmoooorttvww

Code:

```
import java.util.*;
public class ascending_order {
   public static void main(String[] args)
   {
      System.out.println("Bijan Shrestha\n");
      System.out.println("Reg no. 20BCE2904\n");
      String str;
      Scanner s =new Scanner(System.in);
      System.out.println("Enter the string : ");
      str=s.nextLine();
      // Converting string into an array for computation
      char arr[] = str.toCharArray();
      char temp;
      int i = 0;
      while (i <= arr.length) {
         int j = i + 1;
         while (j <= arr.length-1) {
            if (arr[j] < arr[i]) {
               temp = arr[i];
               arr[i] = arr[j];
               arr[j] = temp;
            }
            j += 1;
         }
         i += 1;
      }

      System.out.println("Ascending order string is : ");
      System.out.println(arr);
   }
}
```

Output

Question No. 3: -

Write a java program to sort the names in descending order.

Code:

```
import java.util.*;

public class descending_order {
    public static void main(String[] args) {
        System.out.println("Bijan Shrestha\n");
        System.out.println("Reg no. 20BCE2904\n");
        int n;
        String temp;
        Scanner s = new Scanner(System.in);
        System.out.print("Enter the number of names to be added: ");
        n = s.nextInt();
        String names[] = new String[n];
        Scanner s1 = new Scanner(System.in);
        System.out.println("Enter all the names:");
        for (int i = 0; i < n; i++) {
            names[i] = s1.nextLine();
        }
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (names[i].compareTo(names[j]) < 0) {
                    temp = names[i];
                    names[i] = names[j];
                    names[j] = temp;
                }
            }
        }
        System.out.print("Names in descending Order:");
        for (int i = 0; i < n - 1; i++) {
            System.out.print(names[i] + ",");
        }
        System.out.print(names[n - 1]);
    }
}
```

Question No. 4: -

Write a java Program to check whether the given two strings are anagram or not.
Example: Listen silent

Code:

```
import java.util.*;
public class anagrams {
  static void checkanagram(String str1, String str2) {
    String s1 = str1.replaceAll("\\s", "");
    String s2 = str2.replaceAll("\\s", "");
    boolean status = true;
    if (s1.length() != s2.length()) {
      status = false;
    } else {
      char[] ArrayS1 = s1.toLowerCase().toCharArray();
      char[] ArrayS2 = s2.toLowerCase().toCharArray();
      Arrays.sort(ArrayS1);
      Arrays.sort(ArrayS2);
      status = Arrays.equals(ArrayS1, ArrayS2);
    }
    if (status) {
      System.out.println(s1 + " and " + s2 + " are anagrams");
    } else {
      System.out.println(s1 + " and " + s2 + " are not anagrams");
    }
  }
  public static void main(String[] args) {
    System.out.println("Bijan Shrestha\n");
    System.out.println("Reg no. 20BCE2904\n");
    String str1, str2;
    Scanner s = new Scanner(System.in);
    System.out.println("Enter two strings");
    str1 = s.nextLine();
    str2 = s.nextLine();
    checkanagram(str1, str2);
  }
}
```

Day 5 (Program based on Overloading)

Question No. 1: -

Write a Java program that displays area of different Figures (Rectangle, Square, Triangle) using the method overloading.

Code:

```
import java.lang.Math;
import java.util.*;
class area {
    void area(double s) {
        System.out.println("the area of the square is " + Math.pow(s, 2) + " sq units");
        System.out.println();
    }
    void area(double h, double b) {
        System.out.println("the area of the rectangle is " + h * b + " sq units");
        System.out.println();
    }
    void area(double a, double b, double c) {
        double s = (a + b + c) / 2;
        double ar = s * (s - a) * (s - b) * (s - c);
        double Z = Math.sqrt(ar);
        System.out.println("the area of the triagle with 3 sides is " + Z + " sq units");
        System.out.println();
    }
}
public class overload1 {
    public static void main(String[] args) {
        System.out.println("Bijan Shrestha\n");
        System.out.println("Reg no. 20BCE2904\n");
        area ob = new area();
        Scanner in = new Scanner(System.in);
     System.out.print("Enter side of square: ");
```

```
        double side = in.nextDouble();
        ob.area(side);
        System.out.print("Enter length of rectangle: ");
        double l = in.nextDouble();
        System.out.print("Enter breadth of rectangle: ");
        double b = in.nextDouble();
        ob.area(l, b);
        System.out.print("Enter length of sides: ");
        double x = in.nextDouble();
        double y = in.nextDouble();
        double z = in.nextDouble();
        ob.area(x, y, z);
    }
}
```

Question No. 2: -

In a school, students of all classes from std I to X appear for the MathPremierLeague examination. Define a class MPL which stores the details of the marks scored by each class. It should contain the following 4 data members: Standard, number of students, marks[] array to store the scores of all the students of the class in MPL exam. Define a parameterized constructor which receives the values for the first two data members from the main() method. Create a Form within the constructor, read the marks of all students and hence find the first mark. Define a method findBestClass() to display the standard which has secured the highest mark. Overload this method to display the standard with the highest class average. The marks array should be declared dynamically based on the strength of the class.

Code:

```
import java.util.*;
class MPL {
    int numberOfStudents;
    Integer[] marks = new Integer[10];
    int standard;
    int firstMark = -1;
    MPL(int standard, int numberOfStudents) {
        this.standard = standard;
        this.numberOfStudents = numberOfStudents;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter marks for " + numberOfStudents + " students of class " + standard);
        for (int i = 0; i < numberOfStudents; ++i) {
            marks[i] = sc.nextInt();
            if (marks[i] > firstMark)
                firstMark = marks[i];
        }
    }
    float getAverage() {
        int sum = 0;
        for (int i = 0; i < numberOfStudents; ++i)
            sum += marks[i];
        return (sum / numberOfStudents);
    }
}
public class overload2 {
    public static void main(String args[]) {
        System.out.println("Bijan Shrestha\n");
        System.out.println("Reg no. 20BCE2904\n");
        MPL[] m = new MPL[4];
        m[0] = new MPL(6, 5);
        m[1] = new MPL(9, 5);
        m[2] = new MPL(8, 6);
        m[3] = new MPL(7, 7);
        findBestClass(m);
        findBestClass(m, 1);
    }
    static void findBestClass(MPL[] m) {
        int max = 0;
        for (int i = 0; i < 4; ++i) {
            if (m[i].firstMark > m[max].firstMark)
                max = i;
        }
        System.out.println("Best Class = " + m[max].standard + " Mark = " + m[max].firstMark);
    }
    static void findBestClass(MPL[] m, int avg) {
        int max = 0;
        for (int i = 0; i < 4; ++i) {
            if (m[i].getAverage() > m[max].getAverage())
                max = i;
        }
        System.out.println("Best Average Class = " + m[max].standard + " Mark = " + m[max].getAverage());
    }
}
```

Question No. 3: -

Read the following details of 'n' students using Scanner class methods and display the same.

- Registration number (String)
- Name (String that may contain first name, middle name and last name)
- CGPA (Floating point number)
- Programme Name(String)
- School Name (String with multiple words)
- Proctor Name (String that may contain first, middle and last names).

Code:

```
import java.util.*;

class student_regist {
    String Reg_no;
    String Name;
    float CGPA;
    String programme;
    String school;
    String proctor;
    void getDetails() {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the registration number:");
        Reg_no = sc.next();
        sc.nextLine();
        System.out.print("Enter the name of the student:");
        Name = sc.nextLine();
        System.out.print("Enter the CGPA:");
        CGPA = sc.nextFloat();
        System.out.print("Enter the programme name:");
        programme = sc.next();
        sc.nextLine();
        System.out.print("Enter the school:");
        school = sc.nextLine();
        System.out.print("Enter the proctor name:");
        proctor = sc.nextLine();
    }
    void showDetails() {
        System.out.println("Registration no:" + Reg_no);
        System.out.println("Name:" + Name);
        System.out.println("CGPA:" + CGPA);
        System.out.println("Programme:" + programme);
        System.out.println("School:" + school);
        System.out.println("Proctor:" + proctor);
    }
}
public class overload3 {
    public static void main(String[] args) {
        System.out.println("Bijan Shrestha\n");
        System.out.println("Reg no. 20BCE2904\n");
        int n, i;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of students:");
        n = sc.nextInt();
        student_regist[] s = new student_regist[n];
        for (i = 0; i < n; i++) {
            System.out.println("For Student " + (i + 1));
            s[i] = new student_regist();
            s[i].getDetails();
            System.out.println();
        }
        System.out.println();
        System.out.println("List of Registered Student: ");
        for (i = 0; i < n; i++) {
            s[i].showDetails();
            System.out.println();
        }
    }
}
```