
V2V EDTECH LLP
Online Coaching at an Affordable Price.
OUR SERVICES:
• Diploma in All Branches, All Subjects
• Degree in All Branches, All Subjects
• BSCIT / CS
• Professional Courses
+91 93260 50669
V2V EdTech LLP
v2vedtech.com
v2vedtech

1. Define Constructor. Explain its types with example

Ans:

- Constructor in JAVA is a special type of method that is used to initialize the object.
- JAVA constructor is invoked at the time of object creation.
- It constructs the values i.e. provides data for the object that is why it is known as constructor.
- A constructor has same name as the class in which it resides.
- Constructor in JAVA cannot be abstract, static, final or synchronized.
- These modifiers are not allowed for constructor.

There are two types of constructors

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

1. Default Constructor

• A constructor that have no parameter is known as default constructor.

Syntax of default constructor

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

Example of default constructor.

```
class Bike1
{
Bike1( )
{
System.out.println("Bike is created");
}
public static void main(String args[ ])
{
Bike1 b=new Bike1();
}
}
```

This will produce the following result Bike is created

Note – If there is no constructor in a class, compiler automatically creates a default constructor.

2. Parameterized Constructor

• A constructor that have parameters is known as parameterized constructor.

• Parameterized constructor is used to provide different values to the distinct objects.

Example of parameterized constructor.

```
class Student
{
int id;
String name;
Student(int i, String n)
{
```

```
id = i;
name = n;
}
void display( )
{
System.out.println(id+" "+name);
}
public static void main(String args[ ])
{
Student s1 = new Student4(111,"Ram");
Student s2 = new Student4(222,"Shyam");
s1.display();
s2.display();
}
}
```

This will produce the following result

111 Ram

222 Shyam

2. Explain access specifiers in Java with example

Ans:

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it. There are four types of Java access modifiers:

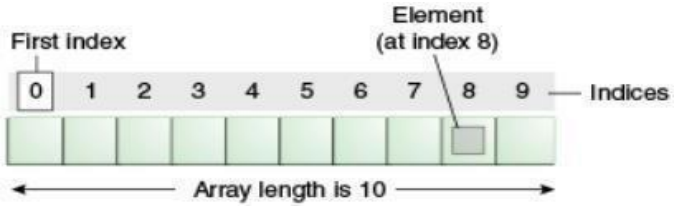
1. Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. Default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. Public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

3. Define array. Explain its types with examples.

Ans:

- An array is a collection of similar type of elements which has contiguous memory location.
- Java array is an object which contains elements of a similar data type.
- 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 0th index, 2nd element is stored on 1st index and so on.

- Java provides the feature of anonymous arrays which is not available in C/C++.



Type of Array There are two types of array.

- One/ Single Dimensional Array
- Two/Multidimensional Array

1. One/ Single Dimensional Array :

The One dimensional array can be represented as

Array a[5]

10	20	70	40	50
0	1	2	3	4

Example

```
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]=30;
a[3]=40;
a[4]=50;
//traversing array
for(int i=0;i<a.length;i++)
System.out.println(a[i]);
}
}
```

OUTPUT

```
10
20
70
40
50
```

2. Two dimensional Array :

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

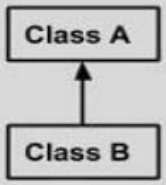
Example of Multidimensional Java Array

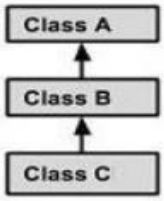
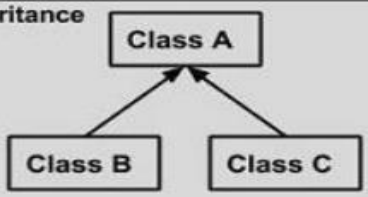
```
class Testarray2
{ public static void main(String args[])
{
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}}; //printing 2D array
for(int i=0;i<3;i++)
{
for(int j=0;j<3;j++)
{
System.out.print(arr[i][j] +" ");
}
System.out.println();
}
}
}
```

Output

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

4. Explain types of inheritance which is supported by java with example
Ans:

Single Inheritance 	public class A {
	.....
	}
	public class B **extends** A {
	.....
	}

Multi Level Inheritance 	public class A { .....} public class B **extends** A {.....} public class C **extends** B {..... }
Hierarchical Inheritance 	public class A { .....} public class B **extends** A {.....} public class C **extends** A {..... }

Program for implanting Single Inheritance

```
class Teacher
{
void Display()
{
System.out.println("This is teacher ");
}
}
class student extends Teacher
{
void show()
{
System.out.println("This is Student");
}
}
class TestInheritance
{
public static void main(String args[])
{
Student s=new student();
s.show();
d.display();
}
}
```

Output
This is student
This is teacher

5. Write a java program to implement multilevel inheritance with 4 levels of hierarchy.

Ans:

```
class emp
{
int empid;
String ename;
emp(int id, String nm)
{
empid=id;
ename=nm;
}
}
class work_profile extends emp
{
String dept;
String job;
work_profile(int id, String nm, String dpt, String j1)
{ super(id,nm);
dept=dpt; job=j1;
}
}
class salary_details extends work_profile
{
int basic_
salary; salary_details(int id, String nm, String dpt, String j1,int bs)
{
super(id,nm,dpt,j1);
basic_salary=bs;
}
double calc()
{
double gs;
gs=basic_salary+(basic_salary*0.4)+(basic_salary*0.1);
return(gs);
}
}
class salary_calc extends salary_details
```

```
{
salary_calc(int id, String nm, String dpt, String j1,int bs)
{
super(id,nm,dpt,j1,bs);
}
public static void main(String args[])
{
salary_calc e1=new salary_calc(101,"abc","Sales","clerk",5000);
double gross_salary=e1.calc();
System.out.println("Empid :"+e1.empid);
System.out.println("Emp name :"+e1.ename);
System.out.println("Department :"+e1.dept);
System.out.println("Job :"+e1.job);
System.out.println("BAsic Salary :"+e1.basic_salary);
System.out.println("Gross salary :"+gross_salary);
}
}
```

6. Explain garbage collection and State use of finalize( ) method with its syntax.

Ans:

Garbage collection

- In JAVA destruction of object from memory is done automatically by the JVM.
- When there is no reference to an object, then that object is assumed to be no longer needed and the memory occupied by the object are released.
- This technique is called Garbage Collection.
- This is accomplished by the JVM.
- Unlike C++ there is no explicit need to destroy object.

finalize( ) method

- Sometime an object will need to perform some specific task before it is destroyed such as closing an open connection or releasing any resources held.
- To handle such situation finalize() method is used.
- finalize()method is called by garbage collection thread before collecting object.
- Its the last chance for any object to perform cleanup utility.

Signature of finalize() method

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

Some Important Points to Remember

1. finalize() method is defined in JAVA.lang.Object class, therefore it is available to all the classes.
2. finalize() method is declare as protected inside Object class.
3. finalize() method gets called only once by GC(Garbage Collector) threads.

6. Explain package & Give syntax to create a package and accessing package in java

Ans:

Packages:

Java provides a mechanism for partitioning the class namespace into more manageable parts called package (i.e package are container for a classes). The package is both naming and visibility controlled mechanism. We can define classes inside a package that are not accessible by code outside that package. We can also define class members that are only exposed to members of the same package.

Creating Packages:- (Defining Packages)

Creation of packages includes following steps:

1) Declare a package at the beginning of the file using the following form.

package package_name

e.g.

package pkg; - name of package

package is the java keyword with the name of package. This must be the first statement in java source file.

2) Define a class which is to be put in the package and declare it public like following way.

```
Package first_package;
Public class first_class
{
Body of class;
}
```

In above example, “first_package” is the package name. The class “first_class” is now considered as a part of this package.

3) Create a sub-directory under the directory where the main source files are stored.

4) Store the listing of, as classname.java file is the sub-directory created. e.g. from the above package we can write “first_class.java”

5) Compile the source file. This creates .class file is the sub-directory. The .class file must be located in the package and this directory should be a sub-directory where classes that will import the package are located.

Accessing a package:-

To access package In a Java source file, import statements occur immediately. Following the package statement (if it exists) and before any class definitions.

Syntax:

```
import pkg1[.pkg2].(classname|*);
```

Here, “pkg1” is the name of the top level package. “pkg2” is the name of package is inside the package1 and so on.
“classname”-is explicitly specified statement ends with semicolon.
• Second way to access the package
import packagename.*;

7. Explain the syntax of try-catch-finally blocks with examples.
Ans:
Syntax of try-catch-finally blocks

```
try {
 ……
}
catch(…….) {
……
}
catch (…….) {
……
}
finally {
……
}
```

Example:

```
class finallyEx
{ public static void main(String[] args)
{
try
{
int[] myNumbers = {1, 2, 3};
System.out.println(myNumbers[10]);
}
catch (Exception e)
{
System.out.println("Something went wrong.");
}
finally {
System.out.println("The 'try catch' is finished.");
}
}
}
```

OUTPUT:
Something went wrong. The 'try catch' is finished.

8. Define thread. Explain 2 ways to create thread.

Ans:

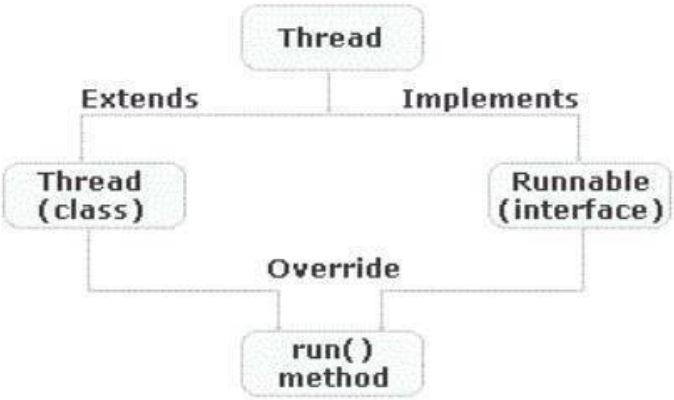
Definition: "A thread is a lightweight subprocess, the smallest unit of processing."

- It is a separate path of execution.
- Threads are independent.
- If exception occurs in one thread, it doesn't affect other threads.
- It uses a shared memory area.
- We can increase the speed of application using thread.

Creating thread:

Thread can be implemented through any one of two ways:

1. Extending the Java.lang.Thread class
2. Implementing Java.lang.Runnable interface



Extending Thread class

1. Extending the java.lang.Thread class:
   a. Extend Java.lang.Thread class
   b. Override run() method in subclass from Thread class
   c. Create instance of this subclass. This subclass may call a Thread class constructor by subclass constructor.
   d. Invoke start() method on instance of class to make thread eligible for running.

Example:

```
class Multi extends Thread
{
public void run()
{
System.out.println("thread is running...");
}
public static void main(String args[])
{
Multi t1=new Multi(); t1.start();
}
}
```

Output:
thread is running...

Implementing Runnable interface
2. Implementing Java.lang.Runnable interface :
a. An object of this class is Runnable object.
b. Create an object of Thread class by passing a Runnable object as argument.
c. Invoke start() method on the instance of Thread class.

Example:

```
class Multi3 implements Runnable
{
public void run()
{
System.out.println("thread is running...");
}
public static void main(String args[])
{
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1); // Using the constructor
Thread(Runnable r)
t1.start();
}
}
```

Output:
thread is running...

9. Define error Enlist any four compile time and Runtime errors.

Ans:

The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.

- Error may produce
- An incorrect output
- may terminate the execution of program abruptly
- may cause the system to crash
- It is important to detect and manage properly all the possible error conditions in program.

Types of Errors

1. Compile time Errors: Detected by javac at the compile time
2. Run time Errors: Detected by java at run time

1. Compile Time Errors(Syntactical errors) :

- Errors which are detected by javac at the compilation time of program are known as compile time errors.
- Most of compile time errors are due to typing mistakes, which are detected and displayed by javac.
- Whenever compiler displays an error, it will not create the .class file.
- Typographical errors are hard to find.
- The most common problems:
- Missing semicolon
- Missing or mismatch of brackets in classes and methods
- Misspelling of identifiers and keywords
- Missing double quotes in strings
- Use of undeclared variables
- Incompatible types in assignments/ Initialization
- Bad references to objects
- Use of = in place of = = operator etc.
- Other errors are related to directory path like command not found

2. Run time Error(Logical Error)

- There is a time when a program may compile successfully and creates a
- .class file but may not run properly.
- It may produce wrong results due to wrong logic or may terminate due to
- errors like stack overflow, such logical errors are known as run time errors.
- Java typically generates an error message and aborts the program.
- It is detected by java (interpreter)
- Run time error causes termination of execution of the program.

- The most common run time errors are:
  - ♣ Dividing an integer by zero.
  - ♣ Accessing element that is out of the bounds of an array.
  - ♣ Trying a store a value into an array of an incompatible class or type.
  - ♣ Passing a parameter that is not in a valid range or value for a method.
  - ♣ Attempting to use a negative size of an array
  - ♣ Converting invalid string to a number.

10. Write a program to check whether the given number is prime or not.

Ans:

```
public class Main {
public static void main(String[] args) {
int num = 29;
boolean flag = false;
for (int i = 2; i <= num / 2; ++i) {
```

```
// condition for nonprime number
if (num % i == 0) {
flag = true;
break;
}
}

if (!flag)
System.out.println(num + " is a prime number.");
else
System.out.println(num + " is not a prime number.");
}
}
```

11. Explain logical & Relational operators in Java with example

Ans:

Logical Operators :

Assume Boolean variables A holds true and variable B holds false then the following table lists the logical operators

Operator	Description	Syntax
&&	Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true.
!	Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

The following program Demonstrates the use of logical operators.

```
public class Test
{
public static void main(String args[])
{
boolean a = true;
boolean b = false;
System.out.println("a && b = " + (a&&b));
System.out.println("a || b = " + (a||b) );
System.out.println("!(a && b) = " + !(a && b));
}
```

}
Output from above program will be :
a && b = false
a || b = true
!(a && b) = true

Relational Operators :
The relational operators determine the relationship that one operand has to the other.

Operator	Meaning	Syntax
==	Equal to	(A == B)
!=	Not equal to	(A != B)
>	Greater than	(A > B)
<	Less than	(A < B)
>=	Greater than or equal to	(A >= B)
<=	Less than or equal to	(A <= B)

Following program demonstrate the use of Relational Operator ==, !=, >,= and <=

```
class RelOptrDemo
{
public static void main(String[] args)
{
int a = 10, b = 15, c = 15;
System.out.println("Relational Operators and returned values");
System.out.println(" a > b = " + (a > b));
System.out.println(" a < b = " + (a < b));
System.out.println(" b >= a = " + (b >= a));
System.out.println(" b <= a = " + (b <= a));
System.out.println(" b == c = " + (b == c));
System.out.println(" b != c = " + (b != c));
}
}
```

Output from above program will be:
a > b = false
a < b = true
b >= a = true
b <= a = false
b == c = true
b != c = false

12. Write a program to find the reverse of a number.

Ans:

```
class Main
{
public static void main(String[] args)
{
int num = 1234, reversed = 0;

System.out.println("Original Number: " + num);

// run loop until num becomes 0
while(num != 0) {

// get last digit from num
int digit = num % 10;
reversed = reversed * 10 + digit;

// remove the last digit from num
num /= 10;
}

System.out.println("Reversed Number: " + reversed);
}
}
```

13. Describe instance Of, dot (.) in Java with suitable example

Ans:

Instanceof :

In Java, instanceof is a keyword used for checking if a reference variable contains a given type of object reference or not. Following is a Java program to show different behaviors of instanceof. Henceforth it is known as a comparison operator where the instance is getting compared to type returning boolean true or false as in Java we do not have 0 and 1 boolean return types.

```
import java.io.*;

// Main class
class GFG {
public static void main(String[] args)
{
// Creating object of class inside main()
GFG object = new GFG();
// Returning instanceof
System.out.println(object instanceof GFG);
```

```
}
}
```
OUTPUT
True

Dot operator :
It is just a syntactic element. It denotes the separation of class from a package, separation of method from the class, and separation of a variable from a reference variable. It is also known as separator or period or member operator.
o It is used to separate a variable and method from a reference variable.
o It is also used to access classes and sub-packages from a package.
o It is also used to access the member of a package or a class.

```
public class DotOperatorExample1
{
void display()
{
double d = 67.54;
//casting double type to integer
int i = (int)d;
System.out.println(i);
}
public static void main(String args[])
{
DotOperatorExample1 doe = new DotOperatorExample1();
//method calling
doe.display();
}
}
```
Output
67

14. Differentiate between String and StringBuffer.
Ans:

Sr. No	String	StringBuffer
1	The length of string object is fixed.	The length of StringBuffer can be increased.
2	The String object is immutable, that means we can not modify the string once created.	The StringBuffer class is immutable.
3	It is slower in performance.	It is faster in performance.
4	It consumes more memory.	It consumes less memory.

15. Differentiate between array and vector.
Ans:

Array	Vector
1) An array is a structure that holds multiple values of the same type.	1)The Vector is similar to array holds multiple objects and like an array; it contains components that can be accessed using an integer index.

2) An array is a homogeneous data type where it can hold only objects of one data type.	2) Vectors are heterogeneous. You can have objects of different data types inside a Vector.
3) After creation, an array is a fixed-length structure.	3) The size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created.
4) Array can store primitive type data element.	4) Vector are store non-primitive type data element.
5)Declaration of an array : int arr[] = new int [10];	5)Declaration of Vector: Vector list = new Vector(3);
6) Array is the static memory allocation.	6) Vector is the dynamic memory allocation.

16. List any four methods of String & StringBuffer class and state the use of each.

Ans:

String :

Method	Description
s1.charAt(position)	Return the character present at the index position.
s1.compareTo(s2)	If s1<s2 then it returns positive. If s1>s2 then it return negative and if s1=s2 then it returns zero.
s1.concat(s2)	It returns the concatenated string of s1 and s2.
s1.equals(s2)	If s1 and s2 are both equal then it returns true.
s1.equalsIgnoreCase(s2)	By ignoring case, if s1 and s2 are equal then it returns true.
s1.indexof('c')	It returns the first occurrence of character 'c' in the string s1.
s1.indexof('c', n)	It returns the position of 'c' that occur at after nth position in string s1.
s1.lenght()	It gives the length of string s1.
String.valueof(var)	Converts the value of the variable passed to it into string type.

StringBuffer:

Name of method	Description
append(String str)	Appends the string to the buffer
capacity()	It returns the capacity of the string buffer
charAt(int index)	It returns a specific character from the sequence which is specified by the index.
delete(int start, int end)	It deletes the characters from the string specified by the starting and ending index.
insert(int offset, char ch)	It inserts the character at the positions specified by the offset.
length()	It return the length of the string buffer.
setCharAt(int index, char ch)	The character specified by the index from the stringbuffer is set to ch.

17. Explain four methods of vector class with examples?
Ans:

Methods	Description
void addElement(object)	For adding the element in the vector
Object elementAt(int index)	Return the element present at specified location(index) in vector.
void insertElementAt(object obj, int pos)	For inserting the element in the vector specified by its position.
boolean removeElement(object ele)	Removes the specified element
void removeAllElements()	For removing all the elements from the vector.
void removeAllElements(int pos)	The elements specified by its position gets deleted from the vector.
int capacity()	Returns the capacity of the vector.
int size()	It returns the total no. of elements present in the vector.
boolean isEmpty()	Return true if the vector is empty.
Object firstElement()	Return the first element of the vector.
int indexOf(object ele)	Returns the index of corresponding element in the vector.
void setSize(int size)	This method is for setting the size of the vector.

18. Differentiate between class and interfaces

Ans:

Class	Interface
It has instance variable.	It has final variable.
It has non abstract method.	It has by default abstract method.
We can create object of class.	We can,,t create object of interface.
Class has the access specifiers like public, private, and protected.	Interface has only public access specifier
Classes are always extended.	Interfaces are always implemented.
The memory is allocated for the classes.	We are not allocating the memory for the interfaces.
Multiple inheritance is not possible with classes	Interface was introduced for the concept of multiple inheritance

class Example
 	
{
void method1()
{
Body
}
void method2()
{
body
}
}
	interface Example
	
	{
	int x =5;
	void method1();
	void method2();
	}

19. Describe interface in Java with suitable example.

Ans:

Interface is also known as kind of a class. Interface also contains methods and variables but with major difference, the interface consist of only abstract method (i.e. methods are not defined, these are declared only) and final fields (shared constants). This means that interface do not specify any code to implement those methods and data fields contains only constants. Therefore, it is the responsibility of the class that implements an interface to define the code for implementation of these methods. An interface is defined much like class.

Syntax:

access interface InterfaceName

{

return_type method_name1(parameter list);

....

```
return_type method_nameN(parameter list);
type final-variable 1 = value1;
….
type final-variable N = value n;
}
Example :
interface Area // interface defined
{
final static float pi=3.14F;
float Cal(float x, float y);
}
class Rectangle implements Area
{
public float Cal(float x, float y)
{
return(x * y);
}
}
class Circle implements Area
{
public float Cal(float x, float y)
{
return(pi *x * x);
}
}
class InterfaceDemo
{
Public static void main(String args[ ])
{
Rectangle r = new Rectangle( );
Circle c = new Circle( );
Area a ; // interface object
a = r;
System.out.println(“Area of Rectangle=” +a.Cal(10,20));
a = c;
System.out.println(“Area of Circle=” +a.Cal(10, 0));
}}
Output :-
C:\java\jdk1.7.0\bin> javac InterfaceDemo.java
C:\java\jdk1.7.0\bin> java InterfaceDemo
Area of Rectangle=200
Area of Circle=314
```

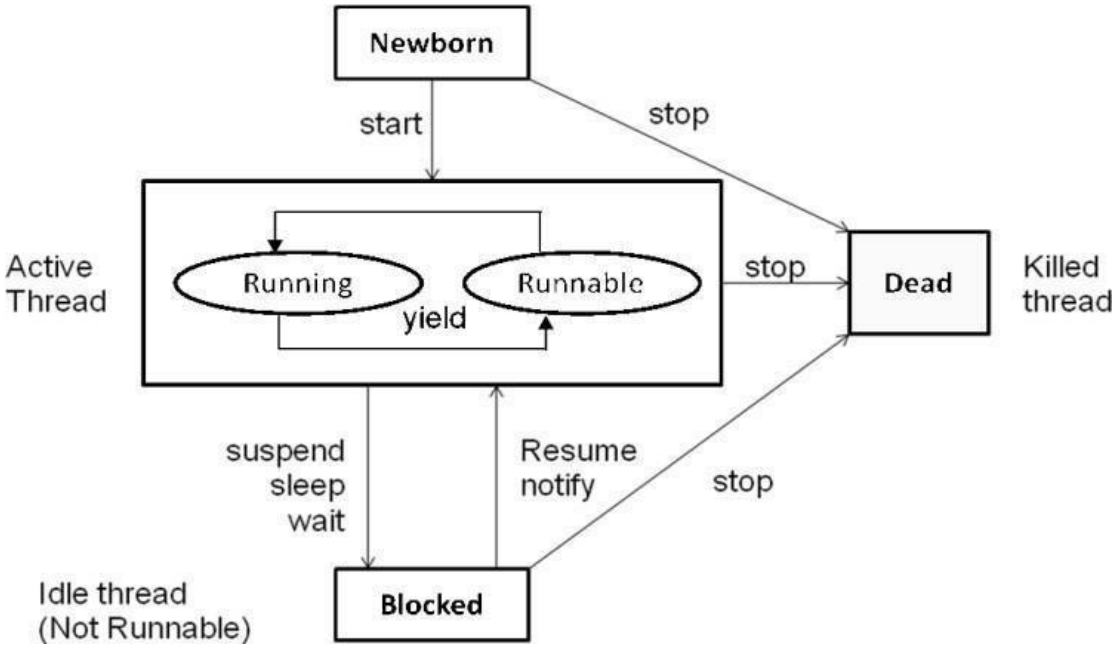
20. Explain the life cycle of thread.

Ans:

During the life time of a thread ,there are many states it can enter, They are:

1. Newborn state
2. Runnable state
3. Running state
4. Blocked state
5. Dead state

A thread is always in one of these five states. It can move from one state to another via a variety of ways.



1) New born state :

- Whenever a new thread is created, it is always in the new state.
- At this time we can scheduled it for running, using start() method or kill it using stop () method.
- If we attempt to use any other method at this stage, an exception will be thrown.

2) Runnable state :

- The thread is ready for execution and is waiting for the availability of the processor.
- The threads has joined waiting queue for execution.
- If all threads have equal priority, then they are given time slots for execution in round robin fashion. i.e. first-come, first serve manner.
- This process of assigning time to thread is known as time-slicing.

- If we want a thread to relinquish(leave) control to another thread of equal priority before its turn comes, then yield( ) method is used.

3) Running state :

- The processor has given its time to the thread for its execution.
- The thread runs until it relinquishes control on its own or it is preempted by a higher priority thread.
- A running thread may change its state to another state in one of the following situations.
    1) When It has been suspended using suspend( ) method.
    2) It has been made to sleep( ).
    3) When it has been told to wait until some events occurs.

4) Blocked state/ Waiting :

- A thread is waiting for another thread to perform a task. The thread is still alive.
- A blocked thread is considered “not runnable” but not dead and so fully qualified to run again.

5) Dead state/ Terminated :

- Every thread has a life cycle. A running thread ends its life when it has completed executing its run ( ) method.
- It is natural death. However we can kill it by sending the stop message.
- A thread can be killed as soon it is born or while it is running or even when it is in “blocked” condition.

21. Explain the command line arguments with suitable example.

Ans:

- The command line argument is the argument passed to a program at the time when it is run.
- To access the command-line argument inside a JAVA program is quite easy, they are stored as string in String array passed to the args parameter of main() method.

Example

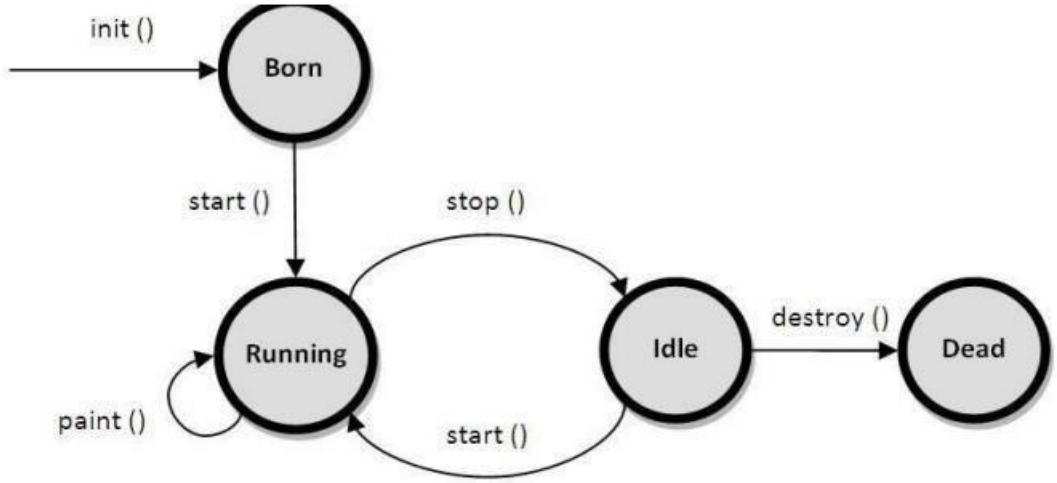
```
class demo
{
public static void main(String args[])
{
int n1=Integer.parseInt(args[0]);
int n1=Integer.parseInt(args[1]);
int add=n1+n2;
System.out.println(add);
}
}
C:\>javac demo.java
C:\>java demo
10 20
30
```

22. Write a program to check whether the string provided by the user is palindrome or not.

Ans:

```
class Main {
public static void main(String[] args) {
String str = "Radar", reverseStr = "";
int strLength = str.length();
for (int i = (strLength - 1); i >=0; --i) {
reverseStr = reverseStr + str.charAt(i);
}
if (str.toLowerCase().equals(reverseStr.toLowerCase())) {
System.out.println(str + " is a Palindrome String.");
}
else {
System.out.println(str + " is not a Palindrome String.");
}
}
}
```

23. Applet life Cycle
Ans:



Applets are small applications that are accessed on an Internet server, transported over the Internet, automatically installed, and run as part of a web document. The applet states include:
Born or initialization state Running state Idle state Dead or destroyed state

a) Born or initialization state:
Applet enters the initialization state when it is first loaded. This is done by calling the init() method of Applet class. At this stage the following can be done: Create objects needed by the applet Set up initial values Load images or fonts Set up colors Initialization happens only once in the life time of an applet.

```
public void init()
{
//implementation
}
```

b) Running state:
Applet enters the running state when the system calls the start() method of Applet class. This occurs automatically after the applet is initialized. start() can also be called if the applet is already in idle state. start() may be called more than once. start() method may be overridden to create a thread to control the applet.

```
public void start()
{
//implementation
```

```
}
```

c) Idle or stopped state:
An applet becomes idle when it is stopped from running. Stopping occurs automatically when the user leaves the page containing the currently running applet. stop() method may be overridden to terminate the thread used to run the applet.

```
public void stop()
{
//implementation
}
```

d) Dead state:
An applet is dead when it is removed from memory. This occurs automatically by invoking the destroy method when we quit the browser. Destroying stage occurs only once in the lifetime of an applet. destroy() method may be overridden to clean up resources like threads.

```
Public void destroy()
{
//implementation
}
```

e) Display state:
Applet is in the display state when it has to perform some output operations on the screen. This happens after the applet enters the running state. paint() method is called for this. If anything is to be displayed the paint() method is to be overridden.

```
public void paint(Graphics g)
{
//implementation
}
```

24. How to create GUI application using class Frame. Also write simple program on it.

Ans:

The Frame class in AWT is used to create a top-level window that can contain various components like buttons, text fields, labels, etc. You can create a GUI application by creating a subclass of Frame or by directly using the Frame class.

Example:

```
import java.awt.*;
import java.awt.event.*;
public class SimpleFrameExample {
public static void main(String[] args) {
// Creating a Frame object
Frame f = new Frame("Simple Frame Example");
// Creating a Button component
Button b = new Button("Click Me");
// Setting button position and size
b.setBounds(100, 100, 100, 50);
// Adding button to the frame
f.add(b);
// Setting frame size
f.setSize(300, 200);
// Making the frame visible
f.setVisible(true);
// Adding window close operation
f.addWindowListener(new WindowAdapter() {
public void windowClosing(WindowEvent we) {
System.exit(0); // Exit the application
}
});
}
}
```

25. Explain overloaded constructors of following AWT classes:

a) Button
b) Label
c) CheckBox

Ans:

Button:

- Button() → Creates a button without text.
- Button(String label) → Creates a button with a label.

Label:

- Label() → Creates a label without text.
- Label(String text) → Creates a label with text.
- Label(String text, int alignment) → Creates a label with text and alignment.

Checkbox:

- Checkbox() → Creates a checkbox without a label.
- Checkbox(String label) → Creates a checkbox with a label.
- Checkbox(String label, boolean state) → Creates a checkbox with a label and initial state.
- Checkbox(String label, CheckboxGroup group, boolean state) → Creates a checkbox with a label, group, and initial state.

26. How to use class CheckBoxGroup to create Radiobutton? Write an example on it.

Ans:

In AWT, the CheckboxGroup class is used to group multiple Checkbox components such that only one checkbox can be selected at a time, making them function like radio buttons.

```
import java.awt.*;

public class RadioButtonExample {

public static void main(String[] args) {

Frame frame = new Frame("Radio Button Example");

// Create a CheckboxGroup

CheckboxGroup group = new CheckboxGroup();

// Create radio button-style checkboxes

Checkbox rb1 = new Checkbox("Option 1", group, true); // Default selected

Checkbox rb2 = new Checkbox("Option 2", group, false);

rb1.setBounds(50, 80, 100, 30);

rb2.setBounds(50, 120, 100, 30);

// Add to frame
```

```
frame.add(rb1);
frame.add(rb2);
// Set frame properties
frame.setSize(200, 200);
frame.setLayout(null);
frame.setVisible(true);
// Close operation
frame.addWindowListener(new java.awt.event.WindowAdapter() {
public void windowClosing(java.awt.event.WindowEvent e) {
System.exit(0);
}
});
}
}
```

27. **Write a program to demonstrate BorderLayout.**

Ans:

```
import java.awt.*;
public class BorderLayoutExample {
public static void main(String[] args) {
// Create a frame
Frame frame = new Frame("BorderLayout Example");
// Set BorderLayout as the layout manager
frame.setLayout(new BorderLayout());
frame.add(new Button("North"), BorderLayout.NORTH);
frame.add(new Button("South"), BorderLayout.SOUTH);
frame.add(new Button("East"), BorderLayout.EAST);
frame.add(new Button("West"), BorderLayout.WEST);
frame.add(new Button("Center"), BorderLayout.CENTER);
frame.setSize(300, 200);
```

```
frame.setVisible(true);
// Add window closing event
frame.addWindowListener(new java.awt.event.WindowAdapter() {
public void windowClosing(java.awt.event.WindowEvent e) {
System.exit(0);
}
}); } }
```

28. Write a program to demonstrate GridLayout. Ans:

```
import java.awt.*;
public class GridLayoutExample {
public static void main(String[] args) {
// Create a frame
Frame frame = new Frame("GridLayout Example");
// Set GridLayout with 3 rows and 2 columns
frame.setLayout(new GridLayout(3, 2));
// Add buttons to the grid
frame.add(new Button("Button 1"));
frame.add(new Button("Button 2"));
frame.add(new Button("Button 3"));
frame.add(new Button("Button 4"));
frame.add(new Button("Button 5"));
frame.add(new Button("Button 6"));
// Set frame properties
frame.setSize(300, 200);
frame.setVisible(true);
// Add window closing event
frame.addWindowListener(new java.awt.event.WindowAdapter() {
public void windowClosing(java.awt.event.WindowEvent e) {
System.exit(0); }
}); } }
```

29. Difference between AWT and Swing.

Ans:

S.NO	AWT	Swing
1.	Java AWT is an API to develop GUI applications in Java	Swing is a part of Java Foundation Classes and is used to create various applications.
2.	The components of Java AWT are heavy weighted.	The components of Java Swing are light weighted.
3.	Java AWT has comparatively less functionality as compared to Swing.	Java Swing has more functionality as compared to AWT.
4.	The execution time of AWT is more than Swing.	The execution time of Swing is less than AWT.
5.	The components of Java AWT are platform dependent.	The components of Java Swing are platform independent.
6.	MVC pattern is not supported by AWT.	MVC pattern is supported by Swing.
7.	AWT provides comparatively less powerful components.	Swing provides more powerful components.
8	AWT components require java.awt package	Swing components requires javax.swing package

30. Explain constructors of following Swing classes

a. JLabel()
b. JButton()
c. JTextField()
d. JRadioButton()

Ans:

a) JLabel Constructors:

1. JLabel()

o Explanation: Creates an empty JLabel without any text or icon. You can later set the text or icon using methods like setText() or setIcon().

2. JLabel(String text)

o Explanation: Creates a JLabel with the specified text. This is useful when you want to initialize the label with some text content.

3. JLabel(String text, Icon icon, int horizontalAlignment)

o Explanation: Creates a JLabel with the specified text and icon, along with the specified alignment for the label's content.

Parameters:

text: Text for the label.

icon: Icon to display on the label.

horizontalAlignment: The alignment of the label's content (e.g., Swing.LEFT, Swing.CENTER, Swing.RIGHT).

4. JLabel(Icon icon)

o Explanation: Creates a JLabel with only the specified icon. This is useful if you only want to display an image or icon on the label without text.

b) JButton Constructors:

1. JButton()

o Explanation: Creates an empty JButton with no text or icon. You can later add text or icons using methods like setText() or setIcon().

2. JButton(String text)

o Explanation: Creates a JButton with the specified text. The button is labeled with the provided text.

3. JButton(Icon icon)

o Explanation: Creates a JButton with the specified icon. This constructor is used when you want the button to only show an icon without any text.

4. JButton(String text, Icon icon)

o Explanation: Creates a JButton with both the specified text and icon. This constructor allows you to have both text and an icon on the button.

c) JTextField Constructors:

1. JTextField()

o Explanation: Creates an empty JTextField with no initial text and a default column size (usually 0). You can later add text or set the number of columns.

2. JTextField(String text)

o Explanation: Creates a JTextField initialized with the specified text. This is used when you want to provide some initial text in the text field.

3. JTextField(int columns)

o Explanation: Creates a JTextField with the specified number of columns (which determines the width of the text field). It is initially empty.

4. JTextField(String text, int columns)

Explanation: Creates a JTextField with the specified text and number of columns. The text field is initialized with the given text, and its width is determined by the number of columns.

d) JRadioButton Constructors:

1. JRadioButton()

o Explanation: Creates an empty JRadioButton with no text or icon. You can later set text or an icon using methods like setText() or setIcon().

2. JRadioButton(String text)

o Explanation: Creates a JRadioButton with the specified text. The radio button will be labeled with the provided text.

3. JRadioButton(String text, boolean selected)

Explanation: Creates a JRadioButton with the specified text and initial selected state (either true or false). This allows you to specify whether the radio button is initially selected or not.

4. JRadioButton(Icon icon)

o Explanation: Creates a JRadioButton with the specified icon instead of text.

5. JRadioButton(String text, Icon icon)

o Explanation: Creates a JRadioButton with both the specified text and icon.

31. Explain use of swing class JScrollPane & JTable. Ans:

a. JScrollPane:

JScrollPane is a Swing class used to provide scrollbars to components that exceed the visible area. It enables you to make content within a container scrollable when the content size is larger than the container's size. You can use JScrollPane to add scrollable areas to components like JTextArea, JTable, JList, and any other component that might need scrolling.

Key Features:

1. Horizontal and Vertical Scrollbars: By default, JScrollPane adds both vertical and horizontal scrollbars when necessary, but you can control them individually.

2. Automatic Handling: It automatically handles the appearance and functionality of scrollbars based on the content's size relative to the container's size.

3. Content Wrapping: The content inside the JScrollPane can be any Swing component, and it will "wrap" itself to the scrollable area.

b. JTable:

JTable is a powerful Swing class that provides a way to display and edit tabular data in a grid format. It is commonly used to display data in rows and columns, similar to a spreadsheet or a database table.

Key Features of JTable:

1. Displays Tabular Data: JTable allows you to display tabular data in a grid of rows and columns.

2. Editable Cells: By default, cells in a JTable are editable. You can allow or prevent editing based on your requirements.

3. Custom Rendering and Editing: You can create custom renderers and editors to display and edit data in a way that fits your application's needs.

4. Sorting and Filtering: You can add sorting and filtering capabilities to the data in the table.

32. Explain following Event Classes.

a. class ActionEvent
b. class TextEvent
c. class KeyEvent

Ans:

a) ActionEvent Class

Purpose: The ActionEvent class in Java represents an event that is triggered by an action performed on a component, such as a button press or a menu selection. It is typically used for event handling in buttons, menu items, and other action-performed components.

Key Points:

- It is part of the java.awt.event package.
- It is generated when an action occurs on a source component (like a button or a menu item).
- The ActionEvent contains information about the event such as the source of the event and the action command.

b) TextEvent Class

Purpose: The TextEvent class is used to handle text-based events. It is typically generated by text components like JTextField or JTextArea. This event occurs whenever the text content is changed in the text field or text area.

Key Points:

- It is part of the java.awt.event package.
- It is used to monitor changes in text fields or areas, for instance, when a user types something.
- This class provides methods to get the text that triggered the event, enabling developers to perform actions based on text updates.

c) KeyEvent Class

Purpose: The KeyEvent class represents a key press or key release on a keyboard. It is used for handling events when a user presses or releases keys while interacting with a component, like a text field or a button.

Key Points:

- It is part of the java.awt.event package.
- It provides methods to detect key presses, key releases, and key typing events.
- KeyEvent can detect a variety of keys, including special keys (e.g., Shift, Ctrl, Enter).

33. Explain following listener interfaces.

a. interface ActionListener
b. interface ItemListener
c. interface MouseListener
d. interface TextListener

Ans:

a) ActionListener Interface

Purpose: The ActionListener interface is used to handle action events, such as when a user performs an action on a button, menu item, or any other component that can trigger an action. For example, clicking a button or selecting a menu item.

Key Points:

- Package: java.awt.event
- The actionPerformed() method is called when an action occurs on a component (e.g., button click).
- Common Use: Used with components like JButton, JMenuItem, and JCheckBox.

Method:

actionPerformed(ActionEvent e): This method is invoked when an action occurs, and it contains the event object.

b) ItemListener Interface

Purpose: The ItemListener interface is used to listen for changes in the state of items in components like checkboxes, radio buttons, and combo boxes. It is invoked when an item is selected or deselected.

Key Points:

- Package: java.awt.event
- The itemStateChanged() method is called when the state of an item changes (e.g., checking or unchecking a checkbox, selecting or deselecting a radio button).
- Common Use: Used with components like JCheckBox, JRadioButton, and JComboBox.

Method:

itemStateChanged(ItemEvent e): This method is invoked when the item state changes (selected or deselected).

c) MouseListener Interface

Purpose: The MouseListener interface is used to handle mouse events such as clicks, presses, and releases on a component.

Key Points:

- Package: java.awt.event
- The MouseListener interface provides methods to detect mouse clicks, presses, releases, and mouse enter/exit events on a component.
- Common Use: Used with components like JPanel, JButton, and JLabel.

Methods:

mouseClicked(MouseEvent e): Invoked when the mouse button is clicked.

mousePressed(MouseEvent e): Invoked when a mouse button is pressed.

mouseReleased(MouseEvent e): Invoked when a mouse button is released.

mouseEntered(MouseEvent e): Invoked when the mouse enters a component.

mouseExited(MouseEvent e): Invoked when the mouse exits a component.

d) TextListener Interface

Purpose: The TextListener interface is used to handle text-based events in text components, such as text fields and text areas. It is invoked whenever the text content is modified (e.g., typing or deleting text).

Key Points:

- Package: java.awt.event
- The textValueChanged() method is called when the content of the text component changes.
- Common Use: Used with components like JTextField, JTextArea, and other text-related components.

Method:

textValueChanged(TextEvent e): This method is invoked when the text in a text component changes.

34. Explain factory & instance methods of class InetAddress with program. Ans:

Key Factory Methods:

1. getByName(String host):

o This method returns the InetAddress object corresponding to the host name or IP address string. If the argument is a host name, the method resolves it to an IP address.

2. getByAddress(String host, byte[] addr):

o This method returns an InetAddress object from a raw IP address (byte array). It is used when you have a byte array representation of an IP address (for example, 192.168.1.1).

3. getLocalHost():

o This method returns the InetAddress object representing the local host (the machine on which the program is running).

4. getAllByName(String host):

o This method returns an array of InetAddress objects for the specified host, which might resolve to multiple IP addresses in case the host has multiple addresses (e.g., in a DNS setup with multiple records).

Program:

```
import java.net.*;
```

```
public class InetMethod {
public static void main(String[] args) throws UnknownHostException {
System.out.println(InetAddress.getByName("www.google.com"));
System.out.println(InetAddress.getByAddress("Local", new byte[] {
(byte)
192, (byte) 168, (byte) 1, (byte) 1
}));
System.out.println(InetAddress.getLocalHost());
InetAddress[] google = InetAddress.getAllByName("google.com");
for (InetAddress addr : google) {
System.out.println(addr.getHostAddress());
}}}
```

Instance Methods of Class InetAddress

a) getHostAddress()

 Description: This method returns the IP address of the InetAddress object as a string in its textual representation (e.g., "192.168.1.1" for IPv4).

 Usage: It is used when you need to get the IP address (in the form of a string) of a particular host or network interface.

b) getHostName()

 Description: This method returns the host name associated with the InetAddress object. If the InetAddress was created from a host name, it will return the same host name. If it was created from an IP address, it will try to perform a reverse DNS lookup to find the host name.

 Usage: It is used to get the host name associated with a particular IP address or domain.

c) isMulticastAddress()

 Description: This method checks if the InetAddress is a multicast address. A multicast address is used to send data to multiple hosts in a network at once.

 Usage: It is used to check if the IP address is within the multicast address range (224.0.0.0 to 239.255.255.255 for IPv4).

d) toString()

 Description: This method returns a string representation of the InetAddress object, which includes both the host name and IP address.

 Usage: It is used to display both the hostname and IP address in a readable format.

e) isReachable(int timeout)

 Description: This method checks whether the given InetAddress is reachable within a specified timeout (in milliseconds). It sends an ICMP ping to test connectivity.

 Usage: It is used to verify if a particular host is accessible over the network.

35. Explain Socket & ServerSocket class and its constructors.

Ans:

Socket Class:

The Socket class in Java is used for establishing a connection between a client and a server. It enables two-way communication using TCP (Transmission Control Protocol). A Socket object represents the client-side endpoint of a connection.

Socket(): Creates an unconnected socket.

Socket(String host, int port): Connects to a server using a host name and port.

Socket(InetAddress address, int port): Connects to a server using an InetAddress and port.

Socket(String host, int port, InetAddress localAddr, int localPort): Connects to a server with a specific local address and port.

ServerSocket Class:

The ServerSocket class in Java is used to create a server that listens for client requests on a specified port. It waits for incoming connections and establishes communication with clients.

1. ServerSocket(int port): Creates a server socket that listens on the specified port.

2. ServerSocket(int port, int backlog): Creates a server socket on the specified port with a defined maximum number of client connections in the queue (backlog).

3. ServerSocket(int port, int backlog, InetAddress bindAddr): Creates a

server socket bound to a specific local address (IP) and port with a defined connection queue size (backlog).

36. Explain any four methods of class URL with example program.

Ans:

getHost(): Returns the host (domain/IP) of the URL.

getProtocol(): Returns the protocol (e.g., "http", "https").

getPath(): Returns the path (file or directory) in the URL.

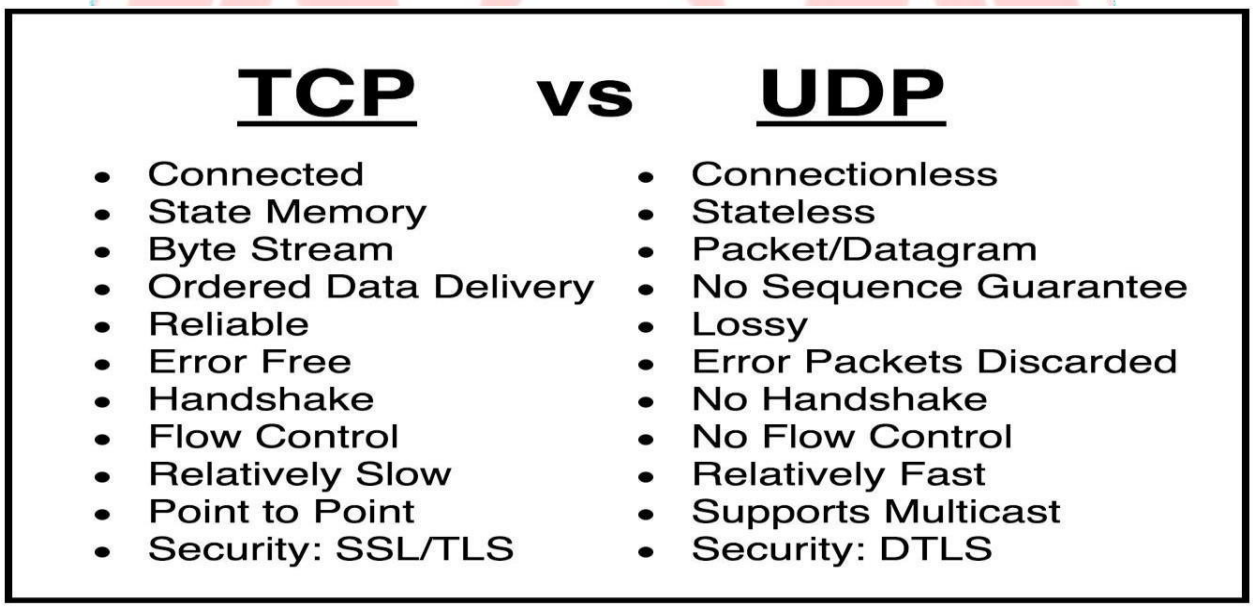
getPort(): Returns the port number used in the URL (or -1 if default port is used).

```
import java.net.*;

public class URLExample {

public static void main(String[] args) throws MalformedURLException {

// Creating a URL object
```

```
URL url = new URL("https://www.example.com:8080/index.html");
// Get and display the host
System.out.println("Host: " + url.getHost()); // Output: www.example.com
// Get and display the protocol
System.out.println("Protocol: " + url.getProtocol()); // Output: https
// Get and display the path
System.out.println("Path: " + url.getPath()); // Output: /index.html
// Get and display the port
System.out.println("Port: " + url.getPort()); // Output: 8080
} }
```

37. Difference between TCP and UDP protocols.

Ans:



38. Write the difference between ServerSocket and DatagramPacket.

Ans:

Feature	ServerSocket (TCP)	DatagramPacket (UDP)
Protocol Used	TCP (Transmission Control Protocol)	UDP (User Datagram Protocol)

Connection Type	Connection-oriented (establishes a connection before data transfer)	Connectionless (sends data without establishing a connection)
Reliability	Reliable (ensures data delivery and order)	Unreliable (no guarantee of delivery or order)
Data Transfer	Stream-based (continuous flow of data)	Packet-based (data sent as independent packets)
Class Used	ServerSocket and Socket	DatagramSocket and DatagramPacket
Overhead	Higher (due to connection setup, acknowledgment, and error checking)	Lower (no connection setup, minimal overhead)
Use Case	Used for applications requiring reliable communication (e.g., web servers, chat applications)	Used for fast, lightweight communication where occasional data loss is acceptable (e.g., live streaming, DNS lookups)

39. Explain URLConnection class methods and example.

Ans:

Methods of URLConnection Class (Any 4)

1. connect()

o Establishes a connection to the specified URL resource.

o Must be called before reading data from the connection.

2. getContentType()

o Returns the MIME type of the resource (e.g., text/html, application/json).

o Helps determine the type of content being fetched.

3. getInputStream()

o Returns an InputStream to read data from the URL.

o Used for reading the contents of web pages or files from a server.

4. setRequestProperty(String key, String value)

o Sets request headers like User-Agent, Authorization, etc.

o Useful for modifying HTTP requests before connecting.

```
import java.io.*;
import java.net.*;
```

```
public class URLConnectionExample {
public static void main(String[] args) {
try {
URL url = new URL("https://www.v2vclass.com");
URLConnection urlcon = url.openConnection();
InputStream stream = urlcon.getInputStream();
int i;
while ((i = stream.read()) != -1) {
System.out.print((char) i);
}} catch (Exception e) { System.out.println(e);
}}
```

40. Explain URL class with constructors and example.

Ans:

URL Class in Java

The URL (Uniform Resource Locator) class in Java, part of the java.net package, is used to represent and manipulate URLs. It allows accessing web resources like web pages, files, and APIs.

```
import java.io.IOException;
import java.net.URL;
public class URLExample {
public static void main(String [] args) {
try {
URL url = new URL("https://www.tutorialspoint.com/index.htm?language=en#j2se");
System.out.println("URL is " + url.toString());
System.out.println("protocol is " + url.getProtocol());
System.out.println("authority is " + url.getAuthority());
System.out.println("file name is " + url.getFile());
System.out.println("host is " + url.getHost());
System.out.println("path is " + url.getPath());
System.out.println("port is " + url.getPort());
System.out.println("default port is " + url.getDefaultPort());
System.out.println("query is " + url.getQuery());
System.out.println("ref is " + url.getRef());
```

```
} catch (IOException e) {
e.printStackTrace();
}
}
}
```

Constructors of URL Class

1. URL(String spec)

o Creates a URL object from a string representation of a URL.

o Example: URL url = new URL("https://www.example.com");

2. URL(String protocol, String host, int port, String file)

o Creates a URL with a specific protocol (e.g., HTTP), host, port, and file path.

o Example: URL url = new URL("https", "www.example.com", 8080, "/index.html");

3. URL(String protocol, String host, String file)

o Similar to the previous constructor but without specifying a port (default port is used).

o Example: URL url = new URL("https", "www.example.com", "/about.html");

4. URL(URL context, String spec)

o Resolves a relative URL against a base URL.

o Example: URL fullURL = new URL(new URL("https://www.example.com/"), "contact.html"); (Results in https://www.example.com/contact.html)

41. Write a program using URL class to retrieve the host,protocol,port and file of URL http://www.msbte.org.in

Ans:

```
import java.net.*;
public class URLDetails {
public static void main(String[] args) {
try {

URL url = new URL("http://www.msbte.org.in");
System.out.println("Protocol: " + url.getProtocol());
System.out.println("Host: " + url.getHost());
System.out.println("Port: " + url.getPort()); // Returns -1 if no port is specified
```

```
System.out.println("File: " + url.getFile());
} catch (MalformedURLException e) {
System.out.println("Invalid URL: " + e.getMessage());
}}
```

42. Write a program using Socket and ServerSocket to create an chat application.

Ans:

It consists of two programs:

1. Server Program (Receives and sends messages)

2. Client Program (Connects to the server and communicates)

Server Program (ChatServer.java)

```
import java.io.*;
import java.net.*;
public class ChatServer {
public static void main(String[] args) {
try {
ServerSocket serverSocket = new ServerSocket(12345);
System.out.println("Server started. Waiting for client...");
Socket socket = serverSocket.accept();
System.out.println("Client connected!");
BufferedReader input = new BufferedReader(new InputStreamReader(socket.getInputStream()));
PrintWriter output = new PrintWriter(socket.getOutputStream(), true);
BufferedReader consoleInput = new BufferedReader(new InputStreamReader(System.in));
String message;
while (true) {
message = input.readLine();
if (message.equalsIgnoreCase("exit")) break;
System.out.println("Client: " + message);
System.out.print("You: ");
output.println(consoleInput.readLine());
}
```

```
socket.close();
serverSocket.close();
} catch (IOException e) {
e.printStackTrace();
}
}
}
```

Client Program

```
import java.io.*;
import java.net.*;
public class ChatClient {
public static void main(String[] args) {
try {
Socket socket = new Socket("localhost", 12345);
System.out.println("Connected to server!");
BufferedReader input = new BufferedReader(new InputStreamReader(socket.getInputStream()));
PrintWriter output = new PrintWriter(socket.getOutputStream(), true);
BufferedReader consoleInput = new BufferedReader(new InputStreamReader(System.in));
String message;
while (true) {
System.out.print("You: ");
message = consoleInput.readLine();
output.println(message);
if (message.equalsIgnoreCase("exit")) break;
System.out.println("Server: " + input.readLine());
}
socket.close();
} catch (IOException e) {
e.printStackTrace(); }
} }
```

43. Explain II tier and III tier JDBC Application architecture with diagram.

Ans:

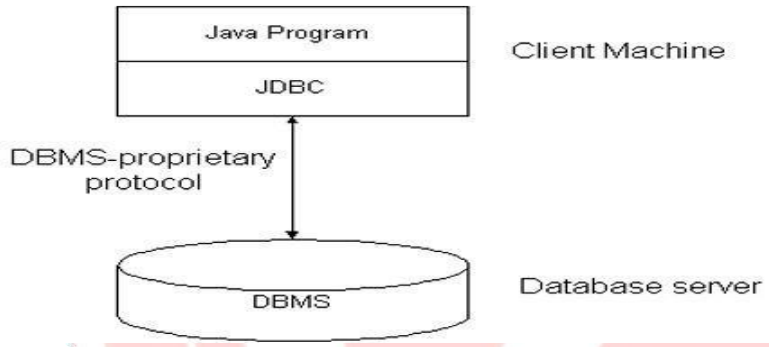
JDBC (Java Database Connectivity) allows Java applications to interact with databases using SQL queries. Based on how the database connection is established and managed, JDBC applications can be categorized into Two-Tier (II-Tier) and Three-Tier (III-Tier) architectures.

Two-Tier (II-Tier) Architecture

In a two-tier architecture, the application directly interacts with the database. It consists of two layers:

How It Works:

- The client application uses JDBC drivers (e.g., Type 1, Type 2, Type 4) to establish a connection with the database.
- Queries are executed on the database, and results are returned directly to the client.

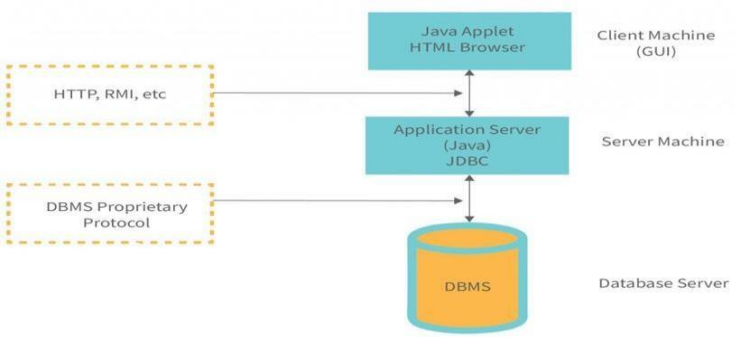


Three-Tier (III-Tier) Architecture

In a three-tier architecture, an intermediate layer (Application Server) is introduced between the client and the database. This improves security, scalability, and maintainability.

How It Works:

- The client sends requests to the application server.
- The application server processes the request, fetches data from the database, applies business logic, and sends the processed data back to the client.



44. Explain four types of JDBC drivers.

Ans: Types of JDBC Drivers

1. Type 1: JDBC-ODBC Bridge Driver

o Converts JDBC calls into ODBC calls and then interacts with the database.

o Requires ODBC drivers installed on the client machine.

o Platform-dependent and slow due to double conversion.

o Deprecated from Java 8 onwards.

2. Type 2: Native-API Driver

o Uses database-specific native libraries to translate JDBC calls into database API calls.

o Faster than Type 1 but still platform-dependent.

o Requires native client libraries for different databases.

o Not suitable for web applications.

3. Type 3: Network Protocol Driver (Middleware Driver)

o Uses a middleware server to convert JDBC calls into database- specific calls.

o Fully written in Java and platform-independent.

o Suitable for internet-based applications but requires middleware setup.

o More scalable than Type 1 and Type 2.

4. Type 4: Thin Driver (Pure Java Driver)

o Directly converts JDBC calls into database-native protocol without middleware.

o Fully Java-based, platform-independent, and the fastest driver.

o Commonly used in modern applications.

o Examples include MySQL Connector/J, Oracle Thin Driver, and PostgreSQL JDBC Driver.

45. List any five classes and interfaces of java.sql package with their short description.

Ans:

DriverManager (Class)

- Manages JDBC drivers and establishes database connections.
- Provides the getConnection() method to connect to a database.

Connection (Interface)

- Represents a connection to the database.
- Provides methods to create statements and manage transactions.

Statement (Interface)

- Used to execute static SQL queries.
- Provides methods like executeQuery(), executeUpdate(), and execute().

PreparedStatement (Interface)

- Extends Statement and allows precompiled SQL queries.
- Improves performance and prevents SQL injection.

ResultSet (Interface)

- Represents the result of a database query.
- Provides methods like next(), getInt(), and getString() to retrieve data.

46. Difference between Statement and Prepared Statement interface.

Ans:

Statement	PreparedStatement
It is used when SQL query is to be executed only once.	It is used when SQL query is to be executed multiple times.
You can not pass parameters at runtime.	You can pass parameters at runtime.
Used for CREATE, ALTER, DROP statements.	Used for the queries which are to be executed multiple times.
Performance is very low.	Performance is better than Statement.
It is base interface.	It extends statement interface.
Used to execute normal SQL queries.	Used to execute dynamic SQL queries.
We can not use statement for reading binary data.	We can use Preparedstatement for reading binary data.
It is used for DDL statements.	It is used for any SQL Query.
We can not use statement for writing binary data.	We can use Preparedstatement for writing binary data.
No binary protocol is used for communication.	Binary protocol is used for communication.

47. Write a program to insert a record in table.

Ans:

```
import java.sql.*;

public class InsertData {
```

```
public static void main(String[] args) {
String url = "jdbc:mysql://localhost:3306/student";
String user = "root";
String password = "";
try {
Class.forName("com.mysql.jdbc.Driver");
Connection conn = DriverManager.getConnection(url, user,
password);
System.out.println("Connected to database!");
String insertSQL = "INSERT INTO login (username, emailid, password)VALUES( ? , ? , ? )";
PreparedStatement pstmt = conn.prepareStatement(insertSQL); pstmt.setString(1, "John Doe");
pstmt.setString(2, "rajan@gmail.com");
pstmt.setString(3, "1234567");
int rowsInserted = pstmt.executeUpdate();
if (rowsInserted > 0) {
System.out.println("A new student record inserted successfully!");
}
pstmt.close(); conn.close();
}
catch (Exception e) {
e.printStackTrace();
}}}
```

48. Write a program to update record in a database.

Ans:

```
import java.sql.*;
public class UpdateData {
public static void main(String[] args) {
String url = "jdbc:mysql://localhost:3306/student";
String user = "root";
```

```
String password = "";
try {
Class.forName("com.mysql.jdbc.Driver");
Connection conn = DriverManager.getConnection(url, user, password);
System.out.println("Connected to database!");
String updateSQL = "UPDATE login SET emailid = ? WHERE emailid = ?";
PreparedStatement pstmt = conn.prepareStatement(updateSQL);
pstmt.setString(1, "rajan@accunityservices.com");
pstmt.setString(2, "rajan@gmail.com");
int rowsUpdated = pstmt.executeUpdate();
if (rowsUpdated > 0) {
System.out.println("Student recordupdated successfully!");
} else {
System.out.println("No matching record found.");
}
pstmt.close();
conn.close();
} catch (Exception e) {
e.printStackTrace();
}}}
```

49. Write a program to implement delete statement.

Ans:

```
import java.sql.*;
public class DeleteData {
public static void main(String[] args) {
String url = "jdbc:mysql://localhost:3306/student";
String user = "root";
String password = "";
try {
```

```
Class.forName("com.mysql.jdbc.Driver");
Connection conn = DriverManager.getConnection(url, user, password);
String deleteSQL = "DELETE FROM login WHERE username = ?";
PreparedStatement pstmt = conn.prepareStatement(deleteSQL);
pstmt.setString(1, "John Doe");
int rowsDeleted = pstmt.executeUpdate();
System.out.println(rowsDeleted > 0 ? "Record deleted successfully!" :
"No matching record found.");
pstmt.close();
conn.close();
} catch (Exception e) {
e.printStackTrace();
}}}
```

50. Develop JDBC program to retrieve data from table using ResultSet interface.

Ans:

```
import java.sql.*;
public class RetrieveStudentRecords {
public static void main(String[] args) {
String url = "jdbc:mysql://localhost:3306/school"; // Change 'school' to your database name
String user = "root";
String password = "";
try {
// Step 1: Load JDBC Driver
Class.forName("com.mysql.cj.jdbc.Driver");
// Step 2: Establish connection
Connection conn = DriverManager.getConnection(url, user, password);
// Step 3: Create SQL query
String selectSQL = "SELECT * FROM student";
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(selectSQL);
```

```
// Step 4: Process the ResultSet
System.out.println("Student Records:");
while (rs.next()) {
int id = rs.getInt("id");
String name = rs.getString("name");
int age = rs.getInt("age");
String grade = rs.getString("grade");
System.out.println(id + " | " + name + " | " + age + " | " + grade);
}
// Close resources
rs.close();
stmt.close();
conn.close();
} catch (Exception e) {
e.printStackTrace();
}}}
```

51. Explain types and method of ResultSet.

Ans:

Types of ResultSet in JDBC-

1. TYPE_FORWARD_ONLY

o Default type.

o Moves only forward through the result set.

o Example: stmt = conn.createStatement(ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY);

2. TYPE_SCROLL_INSENSITIVE

o Allows scrolling both forward and backward.

o Does not reflect database changes after fetching data.

o Example: stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);

3. TYPE_SCROLL_SENSITIVE

o Allows scrolling both forward and backward.

o Reflects real-time changes in the database.

o Example: stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);

Methods of ResultSet in JDBC-

1. **next()**

- Moves the cursor to the next row in the result set.
- Returns true if there is a next row; otherwise, false.
- Used to iterate through all rows in the result set.

2. **getString(String columnName)**

- Retrieves the value of a column as a String.
- Takes the column name as a parameter.
- Useful for fetching text-based data like names.

3. **getInt(int columnIndex)**

- Retrieves the value of a column as an int.
- Takes the column index (starting from 1) as a parameter.
- Used for fetching numerical data like age or ID.

4. **first() (Only for scrollable ResultSet)**

- Moves the cursor to the first row of the result set.
- Useful when working with scrollable result sets to reset the position.

5. **absolute(int row) (Only for scrollable ResultSet)**

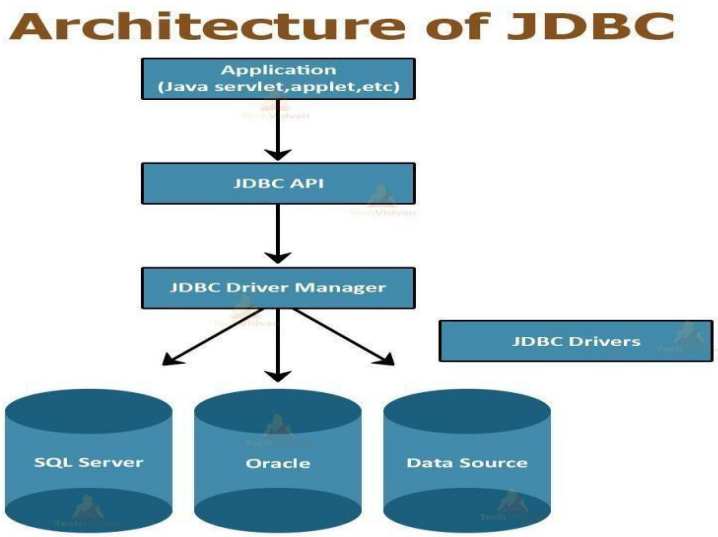
- Moves the cursor to the specified row number in the result set.
- Helps in jumping directly to a specific row.

6. **close()**

- Closes the ResultSet object to free up database resources.
- Should always be called after data retrieval to avoid memory leaks.

52. Explain with neat diagram JDBC architecture.

Ans:



JDBC Architecture

JDBC (Java Database Connectivity) is an API that allows Java applications to interact with databases. The JDBC architecture consists of four key components:

1. JDBC API

- Provides a set of classes and interfaces for connecting to a database.
- Allows Java applications to send SQL queries and retrieve results.

2. JDBC Driver Manager

- Manages different types of JDBC drivers.
- Establishes a connection between the Java application and the database.

3. JDBC Driver

- A software component that translates Java JDBC calls into database-specific calls.
- Different types of JDBC drivers include:

o JDBC-ODBC Bridge Driver

o Native API Driver

o Network Protocol Driver

o Thin Driver (Pure Java Driver)

4. Database

- The actual database where data is stored.
- JDBC API sends queries to the database and retrieves the results.