1. Difference between procedural programming and object-oriented
programming.
Procedural Oriented Programming Object Oriented Programming
In procedural programming, program is divided into small parts called functions.
In object-oriented programming, program is divided into small parts called objects.
Procedural programming follows top-down approach.
Object oriented programming follows bottom-up approach.
There is no access specifier in procedural programming.
Object oriented programming have access specifiers like private, public, protected etc.
Adding new data and function is not easy. Adding new data and function is easy.
Procedural programming does not have any proper way for hiding data so it is less secure.
Object oriented programming provides data hiding so it is more secure.
In procedural programming, overloading is not possible.
Overloading is possible in object-oriented programming.
In procedural programming, function is more important than data.
In object-oriented programming, data is more important than function.
Procedural programming is based on unreal world.
Object oriented programming is based on real world.
Examples: C, FORTRAN, Pascal, Basic etc. Examples: C++, Java, Python, C# etc.

2. How java differs from C++?

- C++ is platform-dependent whereas Java is platform-independent.
- 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.
- Just like C++, Java is also statistically typed object-oriented programming language with a similar with an incompatible syntax. Java has extensive documentation as well which is known as Javadoc.
- C++ uses pointers while Java does not involve pointers. Java uses the concept of 'restricted pointers'.• A C++ program is run and the compilation is done using its compiler. The compiler in C++ converts source code into machine level language which makes C++ a platform-dependent language. In Java, java source code is first converted into bytecode at the time of compilation. This bytecode is then interpreted by the interpreter and runtime to produce the output that makes Java a platform-independent language.
- C++ does not provide built-in support for threads whereas Java support threads implicitly.
- C++ supports multiple inheritance.Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
- Java is much like C++ but does not involve complex concepts like structures, operator overloading, pointers, templates, unions, etc. Also, Java doesn't have support for conditional compile (#ifdef/#ifdef type).
- Every entity is an object in java except fundamental types. Java has a single root hierarchy as everything emerged from java.lang.Object.

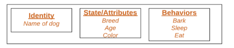
3. What is OOPS?
Object-Oriented Programming System (OOPs) is a programming concept that works on the principles of abstraction, encapsulation, inheritance, and polymorphism. It allows users to create objects they want and create methods to handle those objects.
The basic concept of OOPs is to create objects, re-use them throughout the program, and manipulate these objects to get results.

4. Explain the basic concept of object-oriented programming. Or List the basic principles of Object Orientation.
Object-Oriented Programming or OOPs refers to languages that use objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc. in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.
OOPs Concepts:

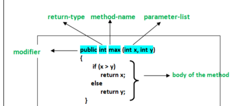
- Class
- Object
- Method
- Encapsulation
- Abstraction
- Polymorphism
- Inheritance
- Message Passing

5. Explain class
Class: A class is a user defined blueprint or prototype from which objects are created.
It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:
Modifiers: A class can be public or has default access (Refer this for details).
Class name: The name should begin with a initial letter (capitalized by convention).Superclass(if any): The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
Interfaces (if any): A comma-separated list of interfaces implemented by the class, if any, proceeded by the keyword implements. A class can implement more than one interface.
Body: The class body surrounded by braces, { }.

6. Explain object
Object: It is a basic unit of Object Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of:
State: It is represented by attributes of an object. It also reflects the properties of an object.
Behaviour: It is represented by methods of an object. It also reflects the response of an object with other objects.
Identity: It gives a unique name to an object and enables one object to interact with other objects.
Example of an object: dog



Method: A method is a collection of statements that perform some specific task and return result to the caller. A method can perform some specific task without returning anything. Methods allow us to reuse the code without retyping the code. In Java, every method must be part of some class which is different from languages like C, C++ and Python.
Methods are time savers and help us to reuse the code without retyping the code.
Method Declaration
In general, method declarations have six components:
Access Modifier: Defines access type of the method i.e. from where it can be accessed in your application. In Java, there are 4 types of the access specifiers.
public: accessible in all class in your application.
protected: accessible within the package in which it is defined and in its subclass(es)(including subclasses declared outside the package)
private: accessible only within the class in which it is defined.
default (declared/defined without using any modifier): accessible within same class and package within which its class is defined.
The return type: The data type of the value returned by the method or void if does not return a value.
Method Name: the rules for field names apply to method names as well, but the convention is a little different.Parameter list: Comma separated list of the input parameters are defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses ().
Exception list: The exceptions you expect by the method can throw, you can specify these exception(s).
Method body: it is enclosed between braces. The code you need to be executed to perform your intended operations.



7. Define Encapsulation
Encapsulation: Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.
Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.
As in encapsulation, the data in a class is hidden from other classes, so it is also known as data-hiding.
Encapsulation can be achieved by declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.

8. Define Abstraction
Abstraction: Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.
Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviours of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.
Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc. in the car. This is what abstraction is.

9. Explain Inheritance?
Inheritance: Inheritance is an important pillar of OOP(Object Oriented Programming).
It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.
Important terminology:
Super Class: The class whose features are inherited is known as superclass(or a base class or a parent class).
Sub Class: The class that inherits the other class is known as subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
Reusability: Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.
The keyword used for inheritance is extends.
Syntax:

```
class derived-class extends base-class
{
 //methods and fields
}
```

10. Explain Polymorphism
Polymorphism: The word 'polymorphism' literally means 'a state of having many shapes' or 'the capacity to take on different forms'. When applied to object-oriented programming languages like Java, it describes a language's ability to process objects of various types and classes through a single, uniform interface.
Polymorphism in Java has two types: Compile time polymorphism (static binding) and Runtime polymorphism (dynamic binding). Method overloading is an example of static polymorphism, while method overriding is an example of dynamic polymorphism. An important example of polymorphism is how a parent class refers to a child class object. In fact, any object that satisfies more than one IS-A relationship is polymorphic in nature. For instance, let's consider a class Animal and let Cat be a subclass of Animal. So, any cat IS animal. Here, Cat satisfies the IS-A relationship for its own type as well as its super class Animal
11. **Explain Message Passing** Message Passing: Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

12.Explain Static Polymorphism?
In Java, static polymorphism is achieved through method overloading. Method overloading means there are several methods present in a class having the same name but different types/order/number of parameters.
At compile time, Java knows which method to invoke by checking the method signatures. So, this is called compile time polymorphism or static binding. The concept will be clear from the following example:

```
classDemoOverload{
publicint add(int x, int y){ //method 1
returnx+y;
 }
publicint add(int x, int y, int z){ //method 2
returnx+y+z;
 }
publicint add(double x, int y){ //method 3
return (int)x+y;
 }
publicint add(int x, double y){ //method 4
return x+(int)y;
 }
}
class Test{
public static void main(String[] args){
DemoOverload demo=new DemoOverload();
System.out.println(demo.add(2,3)); //method 1
calleSystem.out.println(demo.add(2,3,4)); //method 2 called
System.out.println(demo.add(2,3.4)); //method 4 called
System.out.println(demo.add(2.5,3)); //method 3 called
 }
}
```

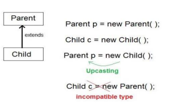
In the above example, there are four versions of add methods. The first method takes two parameters while the second one takes three. For the third and fourth methods, there is a change of order of parameters. The compiler looks at the method signature and decides which method to invoke for a particular method call at compile time.

13.Explain Dynamic Polymorphism?
Suppose a subclass overrides a particular method of the superclass. Let's say we create an object of the subclass and assign it to the superclass reference. Now, if we call the overridden method on the superclass reference then the subclass version of the method will be called.
Have a look at the following example.

```
class Vehicle{
public void move(){
System.out.println("Vehicles can move!!");
 }
}
classMotorBike extends Vehicle{
public void move(){
System.out.println("MotorBike can move and accelerate too!!");
 }
}
class Test{
public static void main(String[] args){
 Vehicle vh=new MotorBike();
vh.move(); // prints MotorBike can move and accelerate too!!
vh=new Vehicle();
vh.move(); // prints Vehicles can move!!
 }
}
```

It should be noted that in the first call to move(), the reference type is Vehicle and the object being referenced is MotorBike. So, when a call to move() is made, Java waits until runtime to determine which object is actually being pointed to by the reference.
In this case, the object is of the class MotorBike. So, the move() method of MotorBike class will be called. In the second call to move(), the object is of the class Vehicle. So, the move() method of Vehicle will be called.
As the method to call is determined at runtime, this is called dynamic binding or late binding.

14.Give an account on dynamic method dispatch in java.
Runtime polymorphism or Dynamic Method Dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile-time.
In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.



When Parent class reference variable refers to Child class object, it is known as Upcasting. In Java this can be done and is helpful in scenarios where multiple child classes extend one parent class. In those cases we can create a parent class reference and assign child class objects to it.

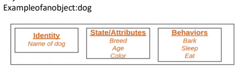
15.Difference between Static binding and Dynamic binding in Java?

- Static binding in Java occurs during compile time while dynamic binding occurs during runtime.
- Static binding uses type (Class) information for binding while dynamic binding uses instance of class (Object) to resolve calling of method at runtime.
- Overloaded methods are bonded using static binding while overridden methods are bonded using dynamic binding at runtime.
- In simpler terms, Static binding means when the type of object which is invoking the method is determined at compile time by the compiler whileDynamic binding means when the type of object which is invoking the method is determined at run time by the compiler.

16.What are benefits of OOP?

- Code reusability new objects can be derived from old objects, allowing for improvement and refinement of the code at each stage and also preserving parts of the code for other programs. This is used to develop many class libraries using class codes that have already been written, for example, Microsoft Foundation Classes (MFC).
- Code Modularity Everything in OOP is an object; these objects can be interchanged or removed to meet the users' needs.
- Easier maintenance Inheritance usually reduces maintenance because of the 'domino effect it has on derived classes when a change is made in a base class.
- Design stability once a stable base class has been developed, the new classes that are derived may have fewer less errors and bugs.
- Improved communication between developers and users Objects can be broken down into real life entities; hence it is easier it communicate ideas.
- Seamless transition from design to implementation this is mainly because communications are improved.

17. Explain the features of object orientation.
AttributesState
Identity
Operation
Behaviour
An object is a single unit having both data and the processes that operate on that data.
For example, in object oriented programming language like C++, the data and functions are bundled together as a self-contained unit called an object. An object is an entity which has some properties and behaviour associated with it. Objects are the basic run time entities in an object oriented system. Programming problems are analysed in terms of objects. The main purpose of using objects is following.
They correspond to the real life entities.
They provide interactions with the real world.
They provide practical approach for the implementation of the solution.
All the objects have a state, behaviour and identity.
Attributes: They describe information about the object.
State: It is represented by attributes of an object. It also reflects the properties of an object.
Behaviour: It specifies what the object can do. It defines the operation performed on objects. It is represented by methods of an object. It also reflects the response of an object with other objects.
Identity: It gives a unique name to an object and enables one object to interact with other objects.
Operation:An Operation is a service that can be requested from any object of the Class to affect behavior. An Operation can either be a command or a question. A question should never change the state of the object only a command can. The outcome of the Operation depends on the current state of the object.
Exampleofanobject:dog



18.What is Dynamic Binding?
Dynamic binding or late binding is the mechanism a computer program waits until runtime to bind the name of a method called to an actual subroutine. It is an alternative to early binding or static binding where this process is performed at compile-time. Dynamic binding is more expensive computationally, but it has the advantage of being more likely to avoid version conflicts when binding functions of a linked library.The ability to perform dynamic bindingis a common characteristic of high-level languages such as C++, Java, and LISP.

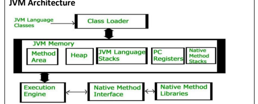
2

1. What is the importance of java API?
In simple terms, API, or Application Programming Interface, is a connection that allows you to make software.APIs are important software components bundled with the JDK. APIs in Java include classes, interfaces, and user Interfaces. They enable developers to integrate various applications and websites and offer real-time information.

2. What is a byte code?
Byte codeis program code that has been compiled from source code into low-level code designed for a software interpreter. Byte code is the compiled format for Java programs. Once a Java program has been converted to byte code, it can be transferred across a network and executed by Java Virtual Machine (JVM). Byte code files generally have a .class extension.

3. How JVM works? Explain JVM Architecture?
OR
4. What is JVM?
Java Virtual Machine (JVM) is a engine that provides runtime environment to drive the Java Code or applications. It converts Java bytecode into machines language. JVM is a part of Java Run Environment (JRE). In other programming languages, the compiler produces machine code for a particular system. However, Java compiler produces code for a Virtual Machine known as Java Virtual Machine.
How JVM works: First, Java code is compiled into byte code. This byte code gets interpreted on different machines Between host system and Java source, Byte code is an intermediary language.JVM in Java is responsible for allocating memory space. Java applications are called WORA (Write Once Run Anywhere). This means a programmer can develop Java code on one system and can expect it to run on any other Java-enabled system without any adjustment. This is all possible because of JVM.When we compile a .java file, .class files (contains byte-code) with the same class names present in .java file are generated by the Java compiler. This .class file goes into various steps when we run it. These steps together describe the whole JVM.

JVM Architecture



Class Loader Subsystem
It is mainly responsible for three activities.
Loading
Linking
Initialization
Loading: The Class loader reads the ".class" file, generate the corresponding binary data and save it in the method area. For each ".class" file, JVM stores the following information in the method area.

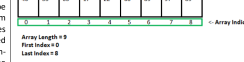
- The fully qualified name of the loaded class and its immediate parent class.
- Whether the ".class" file is related to Class or Interface or Enum.
- Modifier, Variables and Method information etc.

After loading the ".class" file, JVM creates an object of type Class to represent this file in the heap memory.
Linking: Performs verification, preparation, and (optionally) resolution.
Initialization: In this phase, all static variables are assigned with their values defined in the code and static block(if any). This is executed from top to bottom in a class and from parent to child in the class hierarchy.
JVM Memory
Method area: In the method area, all class level information like class name, immediate parent class name, methods and variables information etc. are stored, including static variables. There is only one method area per JVM, and it is a shared resource.
Heap area: Information of all objects is stored in the heap area. There is also one Heap Area per JVM. It is also a shared resource.
Stack area: For every thread, JVM creates one run-time stack which is stored here. Every block of this stack is called activation record/stack frame which stores methods calls. All local variables of that method are stored in their corresponding frame. After a thread terminates, its run-time stack will be destroyed by JVM. It is not a shared resource.
PC Registers: Store address of current execution instruction of a thread. Obviously, each thread has separate PC Registers.
Native method stacks: For every thread, a separate native stack is created. It stores native method information.
Execution Engine
that can be used to verify and resolve accesses to objects at run-time.
Execution engine executes the ".class" (bytecode). It reads the byte-code line by line, uses data and information present in various memory area and executes instructions. It can be classified into three parts:
Interpreter: It interprets the bytecode line by line and then executes. The disadvantage here is that when one method is called multiple times, every time interpretation is required.
Just-In-Time Compiler(JIT) : It is used to increase the efficiency of an interpreter. It compiles the entire bytecode and changes it to native code so whenever the interpreter sees repeated method calls, JIT provides direct native code for that part so reinterpretation is not required, thus efficiency is improved.
Garbage Collector: It destroys un-referenced objects.
Java Native Interface (JNI) : It is an interface that interacts with the Native Method Libraries and provides the native libraries(C, C++) required for the execution. It enables JVM to call C/C++ libraries and to be called by C/C++ libraries which may be specific to hardware.
Native Method Libraries: It is a collection of the Native Libraries(C, C++) which are required by the Execution Engine.

5. Explain important features of java.
Following are the notable features of Java:
Object Oriented: In Java, everything is an Object. Java can be easily extended since it is based on the Object model.Platform
Independent: Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform-independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
Simple: Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
Secure: With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.Architecture-neutral:Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on any processors, with the presence of Java runtime system.
Portable: Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. The compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
Robust: Java makes an effort to eliminate error-prone situations by emphasizing mainly on compile time error checking and runtime checking.
Multithreaded: With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
Interpreted: Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
High Performance: With the use of Just-In-Time compilers, Java enables high performance.
Distributed: Java is designed for the distributed environment of the internet.
Dynamic: Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry an extensive amount of runtime information

6. How can we use arrays in java?

- In Java all arrays are dynamically allocated.
- Since arrays are objects in Java, we can find their length using the object property length. This is different from C/C++ where we find length using sizeof.
- A Java array variable can also be declared like other variables with [] after the data type.
- The variables in the array are ordered and each have an index beginning from 0.
- Java array can be also be used as a static field, a local variable or a method parameter.
- The size of an array must be specified by an int or short value and not long.
- The direct superclass of an array type is Object.
- Every array type implements the interfaces Clone able and java.io.Serializable.
- Array can contain primitives (int, char, etc.) as well as object (or non-primitive) references of a class depending on the definition of the array. In case of primitive data types, the actual values are stored in contiguous memory locations. In case of objects of a class, the actual objects are stored in heap segment



7. What is array? How one dimensional array are Created, Initialized, and Accessed?
Arrays are objects which store multiple variables of the same type. It can hold primitive types as well as object references. In fact most of the collection types in Java which are the part of java.util package use arrays internally in their functioning. Since Arrays are objects, they are created during runtime .The array length is fixed.
One-Dimensional Arrays:
The general form of a one-dimensional array declaration is
type var-name[];
OR
type[] var-name;
An array declaration has two components: the type and the name. type declares the element type of the array. The element type determines the data type of each element that comprises the array. Like an array of integers, we can also create an array of other primitive data types like char, float, double, etc. or user-defined data types (objects of a class). Thus, the element type for the array determines what type of data the array will hold.
Example:
// both are valid declarations
int intArray[];
or int[] intArray;
Instantiating an Array in Java
When an array is declared, only a reference of array is created. To actually create or give memory to array, you create an array like this:The general form of new as it applies to one-dimensional arrays appears as follows:
var-name = new type [size];

Example
double d = 200.06;

Here, type specifies the type of data being allocated, size specifies the number of elements in the array, and var-name is the name of array variable that is linked to the array. That is, to use new to allocate an array, you must specify the type and number of elements to allocate.
Example:

```
int intArray[]; //declaring array
intArray = new int[20]; // allocating memory to array
OR
int[] intArray = new int[20]; // combining both statements in one
```

Array Literal
In a situation, where the size of the array and variables of array are already known, array literals can be used.int[] intArray = new int[]{ 1,2,3,4,5,6,7,8,9,10 };
// Declaring array literal
The length of this array determines the length of the created array.
There is no need to write the new int[] part in the latest versions of Java
Accessing Java Array Elements using for Loop
Each element in the array is accessed via its index. The index begins with 0 and ends at (total array size)-1. All the elements of array can be accessed using Java for Loop.

```
for (int i = 0; i<arr.length; i++)
 System.out.println("Element at index " + i + " : "+ arr[i]);
```

8. Multidimensional Arrays
Multidimensional arrays are arrays of arrays with each element of the array holding the reference of other array. These are also known as Jagged Arrays. A multidimensional array is created by appending one set of square brackets ([]) per dimension.
Examples:

```
int[][] intArray = new int[10][20]; //a 2D array or matrix
int[][][] intArray = new int[10][20][10]; //a 3D array
```

9. What is type casting? Explain different types.
Type casting is nothing but assigning a value of one primitive data type to another. When assign the value of one data type to another, be aware of the compatibility of the data type. If they are compatible, then Java will perform the conversion automatically known as Automatic Type Conversion and if not, then they need to be casted or converted explicitly.
There are two types of casting in Java as follows:
Widening Casting (automatically) –This type of casting takes place when two data types are automatically converted. It is also known as Implicit Conversion. This happens when the two data types are compatible and also when we assign the value of a smaller data type to a larger data type. This involves the conversion of a smaller data type to the larger type size.
byte -> short -> char -> int -> long -> float -> double
Example

```
int i = 200;
//automatic type conversion
long l = i;
//automatic type conversion
float f = l;
```

Narrowing Casting (manually) –In this case, to assign a value of larger data type to a smaller data type, perform Explicit type casting or narrowing. This is useful for incompatible data types where automatic conversion cannot be done.This involves converting a larger data type to a smaller size type.
double -> float -> long -> int -> char -> short -> byte

```
//explicit type casting
long l = (long)d;
//explicit type casting
int i = (int)l;
```

10.What is variable?
A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type. Variable is a name of memory location.
int data=50;//Here data is variable
There are three types of variables in java: local, instance and static.
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.
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.
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.

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

11.How will you give comments in java?
Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.
Single-line comments start with two forward slashes (//).
Any text between // and the end of the line is ignored by Java (will not be executed).
This example uses a single-line comment before a line of code:System.out.println("Hello World"); // This is a comment
This example uses a single-line comment before a line of code:
// This is a comment
System.out.println("Hello World");
Multi-line comments start with /* and ends with */.
Any text between /* and */ will be ignored by Java.
This example uses a multi-line comment (a comment block) to explain the code:
/* The code below will print the words Hello World
to the screen, and it is amazing */
System.out.println("Hello World");

12.List the different types of literals in java.
Literal:Any constant value which can be assigned to the variable is called as literal/constant.
int x = 100; // Here 100 is a constant/literal.
There are five types of literals in Java.

- Integer Literals
- Boolean Literals
- Character Literals
- String Literals
- Floating Point Literals

Integral literals: The integer literal can be used to create int value. They can be used to initialize the group data types like byte, short, int and long.The java compiler treats all the integer literals as int by default. We can assign different values to integer literal.
They are:
Decimal literals (Base 10) : In this form the allowed digits are 0-9.
int x = 101;
Octal literals (Base 8) : In this form the allowed digits are 0-7.
The octal number should be prefix with 0.
int x = 0146;
Hexa-decimal literals (Base 16) : In this form the allowed digits are 0-9 and characters are a-f. We can use both uppercase and lowercase characters. As we know that java is a case-sensitive programming language but here java is not case-sensitive.
The hexa-decimal number should be prefix with 0X or 0x.
int x = 0X123Face;
Binary literals : From 1.7 onward we can specify literals value even in binary form also, allowed digits are 0 and 1. Literals value should be prefixed with 0b or 0B.int x = 0b1111;
Floating-Point literal: The floating point literal can be used to represent the decimal values with a fractional component.
These literals are double data type. These literals contain the fractional parts and if the floating point literal is suffixed with letter f or F then it is float type.
These literals include the float and double data types. In floating point literals double is the default data type. To represent the float literal f is suffixed and to represent the double literal d is suffixed.
For example:
float x = 2.7f;
double x = 54.888d;
Char literalFor char data types we can specify literals in 4 ways:
Single quote: We can specify literal to char data type as single character within single quote.
char ch = 'a';
Char literal as Integral literal: we can specify char literal as integral literal which represents Unicode value of the character and that integral literals can be specified either in Decimal, Octal and Hexadecimal forms. But the allowed range is 0 to 65535.char ch = 062;
Unicode Representation: We can specify char literals in Unicode representation
'\uxxxx'. Here xxxx represents 4 hexadecimal numbers.
char ch = '\u0061';// Here /u0061 represent a.
Escape Sequence: Every escape character can be specify as char literals.
char ch = '\n';
String literal
Any sequence of characters within double quotes is treated as String literals.
String s = "Hello";
String literals may not contain unescaped newline or linefeed characters.
boolean literals
Only two values are allowed for Boolean literals i.e. true and false.boolean b = true;

13.What is the purpose of import statement?
The import statement can be used to import an entire package or sometimes import certain classes and interfaces inside the package. The import statement is written before the class definition and after the package statement (if there is any). Also, the import statement is optional.
the general form of the import statement:
import pkg1[.pkg2].(classname|*);
Here, pkg1 is the name of top-level package, and pkg2 is the name of subordinate package inside outer package separated by dot (.). There is no practical limit on the depth of a package hierarchy, except that imposed by the file system. Now, you specify either an explicit classname or a star (*), indicates that the Java compiler should import the full package. Below code fragment shows both forms in use :
import java.util.Date;
import java.io.*;
All the basic Java classes included with Java are stored in a package called java

14.Explain in detail the different types of operator in java.
Operators are the constructs which can manipulate the values of the operands. Consider the expression 2 + 3 = 5, here 2 and 3 are operands and + is called operator.
Java supports the following types of operators:

- Arithmetic Operators
- Assignment Operators
- Logical Operators
- Relational Operators
- Unary Operators
- Bitwise Operators
- Ternary Operators
- Shift Operators

Arithmetic Operators: Arithmetic Operators are used to perform mathematical operations like addition, subtraction, etc. Assume that A = 10 and B = 20 for the below table.

Operator	Description	Example
+ Addition	Adds values on either side of the operator	A+B=30
- Subtraction	Subtracts the right-hand operator with left-hand operator	A-B=-10
* Multiplication	Multiplies values on either side of the operator	A*B=200
/ Division	Divides left hand operand with right hand operator	A/B=0
% Modulus	Divides left hand operand by right hand operand and returns remainder	A%B=0

Assignment Operators: An Assignment Operator is an operator used to assign a new value to a variable. Assume A = 10 and B = 20 for the below tabl

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b
+=	It adds right operand to the left operand and assigns the result to left operand	c += a
-=	It subtracts right operand from the left operand and assigns the result to left operand	c -= a
*=	It multiplies right operand with the left operand and assigns the result to left operand	c *= a
/=	It divides left operand with the right operand and assigns the result to left operand	c /= a
%=	It takes modulus using two operands and assigns the result to left operand	c %= a
^=	Performs exponential (power) calculation on operators and assign value to the left operand	c ^= a

Relational Operators: These operators compare the values on either side of them and decide the relation among them.
Assume A = 10 and B = 20

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(A == B) is not true
!=	If the values of two operands are not equal, then condition becomes true.	(A != B) is true
>	If the value of the left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true
<	If the value of the left operand is less than the value of right operand, then condition becomes true.	(a < b) is true
>=	If the value of the left operand is greater than or equal to the value of the right operand, then condition becomes true.	(a >= b) is not true
<=	If the value of the left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true

Explain various logical and bitwise operators in java.
Logical Operators: The following are the Logical operators present in Java:

Operator	Description	Example
&& (and)	True if both the operands is true	a<10 && a<20
|| (or)	True if either of the operands is true	a<10 || a<20
! (not)	True if an operand is false (complements the operand)	!(x<10 && a<20)

Bitwise Operator: Bitwise operations directly manipulate bits. In all computers, numbers are represented with bits, a series of zeros and ones.

Operator	Description	Example
& (AND)	returns bit by bit AND of input	a&b
| (OR)	returns OR of input values	a|b
^ (XOR)	returns XOR of input values	a^b
~ (Complement)	returns the one's complement. (all bits reversed)	~a

```
int a = 58; //111010
int b=13; //1101
System.out.println(a&b); //returns 8 = 1000
System.out.println(a|b); //63=111111
System.out.println(a^b); //55=11011
System.out.println(~a); //-59
```

Unary Operator :Unary operators are the one that needs a single operand and are used to increment a value, decrement or negate a value.

Operator	Description	Example
++	increments the value by 1. There is post-increment and pre-increment operators	a++ and ++a
—	decrements the value by 1. There is post decrement and pre decrement operators	a– or –a
!	invert a boolean value	!a

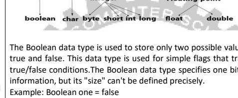
Ternary Operators in Java: The ternary operator is a conditional operator that decreases the length of code while performing comparisons and conditionals. This method is an alternative for using if-else and nested if-else statements. The order of execution for this operator is from left to right.
Syntax:
(Condition) ? (Statement1) : (Statement2);
Condition: It is the expression to be evaluated which returns a Boolean value.
Statement 1: It is the statement to be executed if the condition results in a true state.
Statement 2: It is the statement to be executed if the condition results in a false state.
Shift Operators in Java: Shift operators are used to shift the bits of a number left or right, thereby multiplying or dividing the number. There are three different types of shift operators, namely left shift operator()<<, signed right operator(>>) and unsigned right shift operator(>>>).
Syntax: number shift_opnumber_of_places_to_shift;

```
int a=58;
 System.out.println(a<<2); //232=11101000
System.out.println(a>>2); //returns 14=1110
```

15.What is the difference between m++ and ++m statements in java?
m++ and ++mare very similar but not exactly the same. Both increment the number, but ++m increments the number before the current expression is evaluated, whereas m++ increments the number after the expression is evaluated.

```
int i = 3;
int a = i++; // a = 3, i = 4
int b = ++a; // b = 4, a = 4
```

16.Explain various data types used 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:
Primitive data types: The primitive data types include boolean, char, byte, short, int, long, float and double.
Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays.
17. Explain the primitive data types in java.
In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.



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
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.
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 = -5000Example: byte a = 10, byte b = -20
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
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 = -200000LThe 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
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
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'
Non-primitive data types
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.

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

An interface in Java is a blueprint of a class. It has static constants and abstract methods.

```
interface <interface_name>{
 // declare constant fields
 // declare methods that abstract
 // by default.
}
```

array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. 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 0th index, 2nd
element is stored on 1st index and so on.

18.Explain the syntax of switch statement with example.

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.The syntax of the switch statement in Java is:

```
switch(expression) {
 case value :
 // Statements
 break; // optional
 case value :
 // Statements
 break; // optional
 // You can have any number of case statements.
default : // Optional
 // Statements
}
```

Example:

```
public class Test {
public static void main(String args[]) {
 char grade = 'C';
switch(grade) {
 case 'A' :
 System.out.println("Excellent!");
 break;
 case 'B' :
 case 'C' :
 System.out.println("Well done");
 break;
 case 'D' :
 System.out.println("You passed");
 case 'F' :
 System.out.println("Better try again");
 break;
default :
 System.out.println("Invalid grade");
 }
 System.out.println("Your grade is " + grade);
 }
}
```

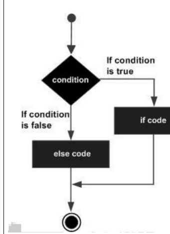
The switch statement work:
The expression is evaluated once and compared with the values of each case label.
If there is a match, the corresponding code after the matching case label is executed.
For example, if the value of the expression is equal to'D', the code after case 'D': is
executed.
If there is no match, the code after default: is executed.
The following rules apply to a switch statement –
• The variable used in a switch statement can only be integers, convertable integers
(byte, short, char), strings and enums.
• You can have any number of case statements within a switch. Each case is
followed by the value to be compared to and a colon.
• The value for a case must be the same data type as the variable in the switch and it must be a constant or a literal.
• When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
• When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
• Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.
• A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

19.Explain the structure and function of if else statement with example.

An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.
Following is the syntax of an if...else statement –

```
if(Boolean_expression) {
 // Executes when the Boolean expression is true
}
else {
 // Executes when the Boolean expression is false
}
```

If the boolean expression evaluates to true, then the if block of code will be executed, otherwise else block of code will be execute




```
public class Test {
public static void main(String args[]) {
 int x = 30;
if( x< 20 ) {
System.out.print("This is if statement");
 }
else {
System.out.print("This is else statement");
 }
 }
}
```

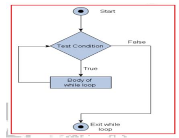
20.Explain various looping structures in java.

Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.Java provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.
There are three types of looping statements in Java. They are as follows:
• While loop
• Do while loop
• For loop

21.Explain the structure and function of while loop construct with example.

while loop: A while loop is a control flow statement that allows code to be executed
repeatedly based on a given Boolean condition. The while loop can be thought of as a
repeating if statement.Syntax :

```
while (boolean condition)
{
 loop statements...
}
```



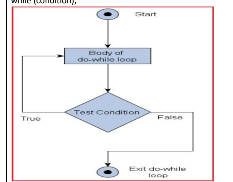
While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called Entry control loop Once the condition is evaluated to true, the statements in the loop body are executed. Normally the statements contain an update value for the variable being processed for the next iteration.
When the condition becomes false, the loop terminates which marks the end of its life cycle.
Example

```
class whileLoopDemo
{
 public static void main(String args[])
 {
 int x = 1;
 while (x <= 4)
 {
 System.out.println("Value of x:" + x);
 // Increment the value of x for
 // next iteration
 x++;
 }
 }}
```

22.Explain the working of while and do-While with syntax and example.

While loop working explained in above question
do-while evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within the do block are always executed at least once.do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of Exit Control Loop.
Syntax:

```
do
{
statements..
}
while (condition);
```



do while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
After the execution of the statements, and update of the variable value, the condition is checked for true or false value. If it is evaluated to true, next iteration of loop starts.When the condition becomes false, the loop terminates which marks the end of its life cycle.It is important to note that the do-while loop will execute its statements at least once before any condition is checked, and therefore is an example of exit control.
Example

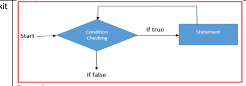
```
class dowhileloopDemo
{
 public static void main(String args[])
 { int x = 21;
 do
 {
 System.out.println("Value of x:" + x);
 x++;
 }
 while (x < 20);
 }}
```

23.Explain structure and function of for statement in java with examples.

for loop: for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping. The syntax to use for the loop is given below.

```
for(initialization; condition; increment/decrement)
{
 Statement(s);
}
```

The initialization expression initializes the loop and is executed only once at the beginning when the loop begins. The termination condition is evaluated every loop and if it evaluates tofalse, the loop is terminated. And lastly, increment/decrement expression is executed after each iteration through the loop.



Example

```
class ForLoopDemo
{
 public static void main(String args[])
 {
for(int num = 1; num <= 5; num++)
 {
 System.out.println(num);
 }}}
```

24.Explain how data and methods are organized in an object oriented program with suitable illustration.OOP languages involve using a series of classes to keep data and functions organized. Every class will contain variables and functions specific to that class that are called from elsewhere in the program where that class is used.First an object has some properties and that properties represent the data
Example:- Student object has some properties like name,age,roll_noetc so that is data about the object.Let as see program

```
class Student
 {
 String name;
 int age;
 int roll_no;
 }
```

Now second come methods(functions),the methods are the operations which are performed on the data of the object.
Example:- student class has some function like getdetail,display etc.

```
class Student
 {
 String name;
 int age;
void getdetail(){
name="java";
age=20;
}
void display(){
System.out.println("Name="+name+"Age="+age);}
}
```

25.What are the two sections in class definition?

A class is a user defined data type which contains the set of similar data and the functions that the objects posses.A class serves as a blueprint or template for its objects.Once a class has been defined, any number of objects belonging to that class can be created.The objects of a class are also known as instances or the variables of that class.
Syntax for defining a class as follows:

```
class class_name
{
//variables declaration
//methods declaration
}
```

Variables declared in the class are known as instance variables. Variables and methods declared within the class are collectively known as members of the class.
Example

```
class Rectangle
{
int length;
int breadth;
int area()
{return(length*breadth);}
}
```

26.Explain about the classes inside classes.

Writing a class within another is allowed in Java. The class written within is called the nested class, and the class that holds the inner class is called the outer class.
Syntax
Following is the syntax to write a nested class. Here, the class Outer_Demo is the outer class and the class Inner_Demo is the nested class.

```
class Outer_Demo {
 class Inner_Demo {
 }
}
```

Nested classes are divided into two types
• Inner class\Non-static nested classes – These are the non-static members of a class.
• Static nested classes – These are the static members of a class.
Inner class means one class which is a member of another class. There are basically three
types of inner classes in java.
1) Nested Inner class
Nested Inner class can access any private instance variable of outer class. Like any other instance variable, we can have access modifier private, protected, public and
default modifier.

```
class Outer {
 class Inner {
 public void show() { System.out.println("In a nested class method"); }
 }
}
class Main { public static void main(String[] args) {
Outer.Inner in = new Outer().new Inner();
in.show();
 }
}
```

2) Method Local inner classes
Inner class can be declared within a method of an outer class. In the following example, Inner is an inner class in outerMethod().

```
class Outer {
 void outerMethod()
 { System.out.println("inside outerMethod");
 class Inner {
 void innerMethod() { System.out.println("inside innerMethod"); }
 }
 Inner y = new Inner();
y.innerMethod();
 }
}
class MethodDemo {
 public static void main(String[] args) {
 Outer x = new Outer();
x.outerMethod();
 }
}
```

3) Anonymous inner classes

An inner class declared without a class name is known as an anonymous inner class. In case of anonymous inner classes, we declare and instantiate them at the same time. Generally, they are used whenever you need to override the method of a class or an interface. The syntax of an anonymous inner class is as follows
Syntax

```
AnonymousInneran_inner = new AnonymousInner() {
 public void my_method() {
 ........
 ........
 }
};
```

Static nested classesStatic nested classes are not technically an inner class. They are like a static member of outer class.

```
class Outer {
 private static void outerMethod() { System.out.println("inside outerMethod"); }
 // A static inner class
 static class Inner {
 public static void main(String[] args) {
 System.out.println("inside inner class Method");
outerMethod();
 }
 }
}
```

27.Explain constructor in java.

OR

28.What are constructors? How they are invoked in java? Also explain the
different types of constructors.

Constructor is a special method in Java which is used to initialize the object. It looks like a normal method however it is not. A normal java method will have return type whereas the constructor will not have an explicit return type. A constructor will be called during the time of object creation (i.e) when we use new keyword follow by class name.
Rules for writing Constructor:
• Constructor(s) of a class must have same name as the class name in which it
resides.
• A constructor in Java cannot be abstract, final, static and Synchronized.
• Access modifiers can be used in constructor declaration to control its access i.e
which other class can call the constructor.
• A Constructor cannot have a explicit return type.
How they are invoked in java
For Example: Let's say we have class by the name "Test", we will create object for Test class like below
Test t = new Test();
This will invoke the default constructor of the Test class.
There are two type of Constructors in Java, they are• Default Constructor (or) no-arg Constructor
• Parameterized Constructor
Default Constructor (or) No-argument constructor: A constructor that has no parameter is known as default constructor. If we don't define a constructor in a class, then compiler creates default constructor (with no arguments) for the class.In the below code we have created the no-arg constructor which gets called during the time of object creation (Car c = new Car())

```
public class Car
{
Car()
 {
 System.out.println("Default Constructor of Car class called");
 }
 public static void main(String args[])
 {
 //Calling the default constructor
 Car c = new Car();
 }
}
```

Parameterized Constructor: A Constructor which has parameters in it called as Parameterized Constructors, the Parameterized constructor is used to assign different values for the different objects.If we want to initialize fields of the class with own values, then use a parameterized constructor.In the below example we have a parameterized constructor for the car class which set the value for the parameter "carColor"

```
public class Car
{
 String carColor;Car(String carColor)
 { this.carColor = carColor;
 }
 public void disp()
 { System.out.println("Color of the Car is : "+carColor);
 }
 public static void main(String args[])
 {
 //Calling the parameterized constructor
 Car c = new Car("Blue");
c.disp();
 }}
```

29. Explain the Basic structure of java program?

A Java program involves the following sections:
Documentation Section
Package Statement
Import Statements
Interface Statement
Class Definition
Main Method Class
Main Method Definition
Documentation Section: You can write a comment in this section. Comments are beneficial for the programmer because they help them understand the code. These are optional.
Package statement: You can create a package with any name. A package is a group of classes that are defined by a name. That is, if you want to declare many classes within one element, then you can declare it within a package. It is an optional part of the program. Here, the package is a keyword that tells the compiler that package has been created.It is declared as:package package_name;
Import statements: This line indicates that if you want to use a class of another package, then you can do this by importing it directly into your program.import calc.add;**Interface statement:** Interfaces are like a class that includes a group of method declarations. It's an optional section and can be used when programmers want to implement multiple inheritances within a program.
Class Definition: A Java program may contain several class definitions. Classes are the main and essential elements of any Java program.**Main Method Class:** Every Java stand-alone program requires the main method as the starting point of the program. This is an essential part of a Java program. There may be many classes in a Java program, and only one class defines the main method. Methods contain data type declaration and executable statements.

30.Explain the structure of simple java program.

Example

```
//Name of this file will be "Hello.java"
public class Hello
{
 /* Description:
 Writes the words "Hello Java" on the screen */
 public static void main(String[] args)
 {
 System.out.println("Hello Java");
 }
}
```

public class Hello
This creates a class called Hello.All class names must start with a capital letter.The public word means that it is accessible from any other classes.
/* Comments */ The compiler ignores comment block. Comment can be used anywhere in the program to add info about the program or code block, which will be helpful for developers to understand the existing code in the future easily.public static void main
• When the main method is declared public, it means that it can also be used by code outside of its class, due to which the main method is declared public.
• The word static used when we want to access a method without creating its object, as we call the main method, before creating any class objects.
• The word void indicates that a method does not return a value. main() is declared as void because it does not return a value.
• main is a method; this is a starting point of a Java program.
String[] args It is an array where each element of it is a string, which has been named as "args". If your Java program is run through the console, you can pass the input parameter, and main() method takes it as input.System.out.println(); This statement is used to print text on the screen as output, where the system is a predefined class, and out is an object of the PrintWriter class defined in the system. The method println prints the text on the screen with a new line. You can also use print() method instead of println() method. All Java statement ends with a semicolon.

31.Differentiate between break and continue in java

S.NO.	Break	Continue
01.	The break statement is used to terminate from the loop immediately.	The continue statement is used to skip the current iteration of the loop.
02.	break keyword is used to indicate break statements in java programming.	continue keyword is used to indicate continue statement in java programming.
03.	We can use a break with the switch statement.	We can not use a break with the switch statement.
04.	The break statement terminates the whole loop early.	The continue statement brings the next iteration early.
05.	It stops the execution of the loop.	It does not stop the execution of the loop.

32.Explain the syntax and function of break and continue statements?

The break and continue statements are the jump statements that are used to skip some statements inside the loop or terminate the loop immediately without checking the test expression. These statements can be used inside any loops such as for, while, do-while loop.
Break: The break statement in java is used to terminate from the loop immediately. When a break statement is encountered inside a loop, the loop iteration stops there, and control returns from the loop immediately to the first statement after the loop. Basically, break statements are used in the situations when we are not sure about the actual number of iterations for the loop, or we want to terminate the loop based on some condition.
Syntax:
break;
In Java, a break statement is majorly used for:
To exit a loop.
Used as a "civilized" form of goto.
Terminate a sequence in a switch statement.example

```
for (int i = 0; i< 10; i++) {
 // Terminate the loop when i is 5
 if (i == 5)
 break;
 System.out.println("i: " + i);
 }
```

Continue: The continue statement in Java is used to skip the current iteration of a loop. We can use continue statement inside any types of loops such as for, while, and do-while loop. Basically, continue statements are used in the situations when we want to continue the loop but do not want the remaining statement after the continue statement.
Syntax:
continue;
Example

```
for (int i = 0; i< 10; i++) {
 // If the number is 2
 // skip and continue
 if (i == 2)
 continue;
System.out.print(i + " ");
 }
```

33.Explain the methods of StringBuffer class with example.

OR

34.Write a short note on StringBuffer class.

Java StringBuffer class is used to create mutable (modifiable) string.StringBuffer is a peer class of String that provides much of the functionality of strings. String represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences.Important Constructors of StringBuffer classConstructor Description
StringBuffer() creates an empty string buffer with the initial capacity of 16.StringBuffer(String str) creates a string buffer with the specified string.StringBuffer(int capacity) creates an empty string buffer with the specified capacity as length.Important methods of StringBuffer classappend() method: is used to append the specified string with this string. The append()
method is overloaded like append(char), append(boolean), append(int), append(float),
append(double) etc.
StringBuffer sb=new StringBuffer("Hello ");
sb.append("Java");//now original string is changed
System.out.println(sb);//prints Hello Java
insert(int offset, String s)method: is used to insert the specified string with this string at the specified position. The insert()
method is overloaded like insert(int, char),insert(int, boolean),
insert(int, int), insert(int, float), insert(int, double)
etc.StringBuffer sb=new StringBuffer("Hello
");sb.insert(1,"Java");//now original string is changed
System.out.println(sb);//prints HJavaello replace(int startIndex, int endIndex, String str): is used to replace the string from
specified startIndex and endIndex.
StringBuffer sb=new StringBuffer("Hello");
sb.replace(1,3,"Java");
System.out.println(sb);//prints HJavalo
delete(int startIndex, int endIndex):is used to delete the string
from specified startIndex and endIndex.
StringBuffer sb=new StringBuffer("Hello");
sb.delete(1,3);
System.out.println(sb);//prints Hlo
reverse():is used to reverse the string.
StringBuffer sb=new StringBuffer("Hello");
sb.reverse();
System.out.println(sb);//prints olleH
public intcapacity():is used to return the current capacity.The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.
StringBuffer sb=new StringBuffer();
System.out.println(sb.capacity());//default 16
public intlength():is used to return the length of the string i.e. total number of characters.
StringBuffer s = new StringBuffer("Hello");
 int p = s.length();
 System.out.println("Length "+ p); //length 5
public Stringsubstring(int beginIndex):is used to return the substring from the specified beginIndex.
System.out.println("SubSequence = "+ str.substring(3)); // lo

35.What is this operator?

In java, this keyword is a reference variable that refers to the current object of a method or a constructor. The main purpose of using this keyword in Java is to remove the confusion between class attributes and parameters that have same names.
Following are various uses of 'this' keyword in Java:
• It can be used to refer instance variable of current class
• It can be used to invoke or initiate current class constructor
• It can be passed as an argument in the method call
• It can be passed as argument in the constructor call
• It can be used to return the current class instance

```
class Student{
int rollno;
String name;
Student(int rollno,String name){
this.rollno=rollno;
this.name=name; }
```

36.What is finalize method in java?

It is a method that the Garbage Collector always calls just before the deletion/destroying the object which is eligible for Garbage Collection, so as to perform clean-up activity. Clean-up activity means closing the resources associated with that object like Database Connection, Network Connection or we can say resource de-allocation. Remember it is not a reserved keyword.
Once the finalize method completes immediately Garbage Collector destroy that object. finalize method is present in Object class and its syntax is:
protected void finalize throws Throwable{}

37.What is a final variable?

final variables are nothing but constants. We cannot change the value of a final variable once it is initialized.A final variable can be explicitly initialized only once. A reference variable declared final can never be reassigned to refer to a different object.

```
public class Test {
 public static void main(String args[]) {
 final int i = 10;
i = 30; // Error because i is final.
 }
}
```

38.What are the uses of super keyword?

The super keyword in Java is a reference to the object of the parent/superclass. Using it, you can refer/call a field, a method or, a constructor of the immediate superclass.super can be used to refer immediate parent class instance variable.

```
class Animal{
String color="white";
}
class Dog extends Animal{
String color="black";
void printColor(){
System.out.println(color);//prints color of Dog class
System.out.println(super.color);//prints color of Animal class
}
}
```

In the above example, Animal and Dog both classes have a common property color. If we print color property, it will print the color of current class by default. To access the parent property, we need to use super keyword.super can be used to invoke immediate parent class method.

```
class Animal{void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
void bark(){System.out.println("barking...");}
void work(){
super.eat(); //invokes Animal class eat() method
bark();
}
}
```

In the above example Animal and Dog both classes have eat() method if we call eat()
method from Dog class, it will call the eat() method of Dog class by default because priority is given to local.To call the parent class method, we need to use super keyword.super() can be used to invoke immediate parent class constructor.

```
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
super(); //invokes Animal class constructor
System.out.println("dog is created");
}
}
```

39.Explain static variables and methods with an example.

In java, static belongs to class. You can create static variables and static methods. You can call these directly by using class name, without creating instance.
Java static variables:
Static variables are belongs to the class and not to the object. These are only once, at the starting of the execution. Static variables are not part of object state, means there is only one copy of the values will be served to all instances. You can call static variable with reference to class name without creating an object. Below example shows how to create and call static variables.
Java static methods:
Static methods are also similar to static variables, you can access them with reference to class name, without creating object. Inside static methods, you cannot access instance variables or instance methods. You can only access static variables or static methods.

```
/Java Program to demonstrate the use of a static method.
class Student{
 int rollno;  String name; //instance variable
 static String college = "ITS"; //static variable
static void change(){ //static method to change the value of static variable
 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();
 }
}
```

40.Explain abstract classes and its characteristics with example.

Abstraction is a process of hiding the implementation details and showing only functionality to the user.A class which is declared with the abstract keyword is known as an abstract class in Java.
It can have abstract and non-abstract methods. abstract classes is a type of class in Java, that declare one or more abstract methods. A normal class cannot have abstract methods.
It needs to be extended and its method implemented. It cannot be instantiated.
Example of abstract class
abstract class A{}
Abstract Method in JavaA method which is declared as abstract and does not have implementation is known as an abstract method.
Example of abstract method
abstract void printStatus();//no method body and abstract
Example

```
abstract class Shape{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end user
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}
class Circle extends Shape{
void draw(){System.out.println("drawing circle");}
}
//In real scenario, method is called by programmer or user
class TestAbstraction1{
public static void main(String args[]){
Shape s=new Circle();
s.draw();
}
}
```

41.What the purpose is of extends keyword?

The extends keyword extends a class (indicates that a class is inherited from another class).In Java, it is possible to inherit attributes and methods from one class to another.
• The extends is a Java keyword, which is used in inheritance process of Java. It specifies the super class in a class declaration using extends keyword. It is a keyword that indicates the parent class that a subclass is inheriting from. In Java, every class is a subclass ofjava.lang.Object.
• Extends keyword is also used in an interface declaration to specify one or more super interfaces.

```
public class A{
public int number;}
class B extends A {
public void increment() {
number++; }}
```

In this example, we inherit from class A, which means that B will also contain a field called number. Two or more classes can also be inherited from the same parent class.

42.Explain inheritance in java.

OR

43.Explain the different forms of inheritance with examples.

OR

44.Describe multiple and multilevel inheritance.

OR

45.Write a java program to implement multilevel inheritance

OR

46.Explain single inheritance with an example.

inheritance

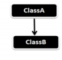
It is the process of deriving a new class from the existing one in such a way that new class inherits all the members of the existing class.The class that is inherited by other classes is called a base class or superclass or parent class.
The class which inherits the properties of base class is called derived class or subclass or child class.
The subclass inherits all the instance variables and methods of the superclass and also has its own members.
Defining a Subclass
The name of the superclass is specified in the subclass. A subclass is defined by using the extends keyword.
Syntax to define a subclass is as follows:

```
class subclass_name extends superclass_name
{
//variables declaration
//methods declaration
}
```

subclass_name=> name of the subclass
superclass_name=> name of the superclass
extends=> it is a keyword that indicates the superclass properties have been extended to the subclass.
Types of inheritance
Inheritance allows the creation of a logical relationship between two or more classes.
Depending on the number of classes involved and the way the classes are inherited,
inheritance may take different forms, namelySingle inheritance
Multilevel inheritance
Hierarchical inheritance
Multiple inheritance
In java it is not possible to implement multiple inheritance directly, there is a
concept known as interface through which multiple inheritance implemented.

Single Inheritance:

Single Inheritance is the simple inheritance of all, When a class extends another class(Only one class) then we call it as Single inheritance. The below diagram represents the single inheritance in java where Class B extends only one class Class A. Here Class B will be the Sub class and Class A will be one and only Super class.



Syntax:

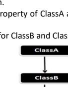
```
class classA
{
// members
}
class classB extends classA
{
//members
}
```

Example

```
class Shape{
 void draw()
 {
 System.out.println("Draw shape");
 }
}
class Circle extends Shape
{
 void drawCircle(){
 System.out.println("Draw Circle");
 }
public static void main(String args[])
{ Circle c = new Circle();
c.draw();
c.drawCircle();}}
```

Multilevel Inheritance:

In Multilevel Inheritance a derived class will be inheriting a parent class and as well as
the derived class act as the parent class to other class. As seen in the below diagram.
ClassB inherits the property of ClassA and again ClassB act as a parent for ClassC. In
Short ClassA parent for ClassB and ClassB parent for ClassC.



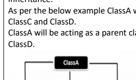
Syntax:

```
class classA{
// members}
class classB extends classA{
//members}
class classC extends classB{
//members}
```

Example

```
public class ClassA{
 public void dispA(){
 System.out.println("disp() method of ClassA");}}
public class ClassB extends ClassA
{ public void dispB(){
 System.out.println("disp() method of ClassB");}}
public class ClassC extends ClassB{
 public void dispC() { System.out.println("disp() method of ClassC"); } public static void main(String args[]){
 //Assigning ClassC object to ClassC reference
ClassC c = new ClassC();
 //call dispA() method of ClassA
c.dispA();
 //call dispB() method of ClassB
c.dispB();
 //call dispC() method of ClassC
c.dispC();
 }
}
```

Hierarchical Inheritance:
When one single class is inherited by multiple subclasses is known as hierarchical
inheritance.
As per the below example ClassA will be inherited by ClassB, ClassC and ClassD.
ClassA will be acting as a parent class for ClassB, ClassC and ClassD.



Syntax

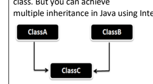
```
class classA{
// members}
class classB extends classA
{//members}
class classC extends classA{
//members}
```

Example

```
class Laptop
{
 void display()
 {
 System.out.println("Working...");
 }
class Dell extends Laptop
{
 void print()
 {
 System.out.println("Dell Inspiron");
 }
}
class Lenovo extends Laptop
{
 void show()
 {
 System.out.println("Lenovo YOGA");
 }
}
class TestDemo
{
 public static void main(String args[]){
 Dell d = new Dell();
d.print();
d.display();
 Lenovo l = new Lenovo();
l.show();
l.display();
 }
}
```

Multiple Inheritance

Multiple Inheritance is nothing but one class extending more than one class. Multiple
Inheritance is basically not supported by many Object Oriented Programming languages
such as Java, Small Talk, C# etc.. (C++ Supports Multiple Inheritance). As the Child
class has to manage the dependency of more than one Parent class. But you can achieve
multiple inheritance in Java using Interfaces.



47.What is an interface? How interface helps to implement multiple inheritance
in java.
OR
48.How multiple inheritance can be implemented in java?
we cannot achieve multiple inheritance in the case of class because of the ambiguity problem. but using the interface we can achieve multiple inheritance in java.An interface in java is syntactically similar to classes, but they do not have instance variable and their method are declared without any body.
The methods declared inside an interface are by default abstract(only method signature, not implementation) and the variables declared inside an interface are public, static, and final by default. The interface in java is a mechanism to achieve abstraction. Once an interface is defined, any number of classes can implement this interface. Also, one class can implement any number of interfaces.
define an Interface.
An interface is defined much like a class. It can have methods and variables like class. But in Interface, all the methods are by default abstract and all the variables are by default public static, and final. This is the general form of an interface:
interface name{
return-type method-name1(parameter-list);
return-type method-name2(parameter-list);
type varname1 = value;
type varname2 = value;
//----
return-type method-nameN(parameter-list);
type varnameN = value;
}to implement an Interface.
Once an interface has been defined, one or more classes can implement that interface. To implement an interface, include the implements clause in a class definition, and create the methods defined by the interface. The general form of a class that includes the implements clauses look like this:
class classname implements interface-name{
// class body
}
Example
interface Area
{
float p=3.14;
void compute();
}
class circle implements Area
{
float r=2.2F;
public void compute()
{double c=p*r*r;
System.out.println("area of circle="+c);
}
}
class sphere implements Area
{
float r=3.7;
public void compute()
{double s=4*p*r*r;
system.out.println("area of sphere="+s);
}
}
class inter

Each package in Java has its unique name and organizes its classes and interfaces into a separate namespace, or name group.
Although interfaces and classes with the same name cannot appear in the same package, they can appear in different packages. This is possible by assigning a separate namespace to each Java package.
Syntax: -
package nameOfPackage;
The steps of creating a package.
• Choose the name of the package
• Include the package command as the first line of code in your Java Source File.
• The Source file contains the classes, interfaces, etc you want to include in the package
• Compile to create the Java packages
Step 1) Consider the following package program in Java:
package p1;class c1(){
public void m1(){System.out.println("m1 of c1");}}
public static void main(String args[]){
c1 obj = new c1();
obj.m1();} }
Here,To put a class into a package, at the first line of code define package p1.
Next create class c1 with method m1 and call this method using object obj .
Step 2) In next step, save this file as demo.java
Step 3) In this step, we compile the file.
javac demo.java
The compilation is completed. A class file c1 is created. However, no package is created?
Next step has the solution
Step 4) Now we have to create a package, use the command
javac –d . demo.java
This command forces the compiler to create a package.
The "." operator represents the current working directory.
Step 5) When you execute the code, it creates a package p1.
When you open the java package p1 inside you will see the c1 class file.
Step 6) Compile the same file using the following code
javac –d .. demo.java
Here ".." indicates the parent directory.
Step 7) Now let's say you want to create a sub package p2 within our existing java
package p1. Then we will modify our code as
package p1.p2;
class c1{
public void m1() {System.out.println("m1 of c1");}}
to Import Package
Syntax
import packageName;
Once imported, you can use the class without mentioning its fully qualified name.
Step 1) Copy the code into an editor.
package p3;
import p1.*; //imports classes only in package p1 and NOT in the sub-package p2
class c3{
public void m3(){ System.out.println("Method m3 of Class c3");
}
public static void main(String args[]){
c1 obj1 = new c1();
obj1.m1();}}
Step 2) Save the file as Demo2.java. Compile the file using the command
javac –d .. Demo2.java
Step 3)Execute the code using the command java p3.c3

52.Write a note on java API packages
Java API(Application Program Interface) provides a large numbers of classes grouped into different packages according to functionality. Most of the time we use the packages available with the the Java API. Java System Packages and Their Classes

java.lang	Language support classes. They include classes for primitive types, string, math functions, thread and exceptions.
java.util	Language utility classes such as vectors, hash tables, random numbers, data, etc.
java.io	Input/output support classes. They provide facilities for the input and output of data.
java.applet	Classes for creating and implementing applets.
java.net	Classes for networking. They include classes for communicating with local computers as well as with internet servers.
java.awt	Set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on.

53.Differentiate method overloading and method overriding with examples for each.
OR
54.With suitable examples explain the difference between method overriding and method overloading

Method Overloading	Method Overriding
Method overloading is used *to increase the readability of the program.*	Method overriding is used *to provide the specific implementation of the method* that is already provided by its super class.
Method overloading is performed *within class.*	Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
In case of method overloading, *parameter must be different.*	In case of method overriding, *parameter must be same.*
Method overloading is the example *of compile time polymorphism.*	Method overriding is the example *of run time polymorphism.*
In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter.	*Return type must be same or covariant* in method overriding.

There are many differences between method overloading and method overriding in java.
A list of differences between method overloading and method overriding are given
below:
Java Method Overloading example
class OverloadingExample{
static int add(int a,int b){return a+b;}
static int add(int a,intb,int c){return a+b+c;} }
Java Method Overriding example
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
}

3

1. What is file class?
The File class is an abstract representation of file and directory pathname. A pathname can be either absolute or relative.The file class package is present in the java.io package. The abstract pathname of the file or a directory is a java.io.File object whether the pathname string is actually a String object.The File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc.
2. List any four methods in java.io.file class.
Methods of File Class in Java
booleancanExecute(): This function returns true if the abstract pathname points to a file that is executable. If the file is not executable, it returns false.
booleancanRead() : This function returns true, if the program can read the file. This file is actually an abstract pathname. If it cannot read, it returns false.
booleancanWrite(): This boolean function returns true if it can write over the contents of the file. If not, it returns false.
booleancreateNewFile(): This function creates a new file that is empty following the abstract pathname. If it cannot then it returns false.
booleandelete(): This function deletes the directory or the file denoted by the abstract pathname. It returns false if an exception occurs.
booleanequals(Object obj): This function checks whether the pathname is equal to the given object.
booleanexists(): This method checks whether the file or directory is present or not. Returns true if found, else returns false.
booleanisFile(): This boolean method returns true if the pathname points to a file and false otherwise.
String getPath(): This method generates a pathname string from the abstract pathname.
booleanisHidden(): This method checks if the path points to a hidden file and false otherwise.If it is hidden it returns true, else false.
3. Explain different stream classes in java .
Java provides I/O Streams to read and write data where, a Stream represents an input source or an output destination which could be a file, i/o device, other program etc.In general, a Stream will be an input stream or, an output stream
InputStream: This is used to read data from a source.
OutputStream: This is used to write data to a destination.
Based on the data they handle there are two types of streams –
Byte Streams: These handle data in bytes (8 bits) i.e., the byte stream classes read/write data of 8 bits. Using these you can store characters, videos, audios, images etc.
Character Streams: These handle data in 16 bit Unicode. Using these you can read and write text data only.Byte Stream Classes Byte stream is defined by using two abstract class at the top of hierarchy, they are InputStream and OutputStream.
These two abstract classes have several concrete classes that handle various devices such as disk files, network connection etc
Some important Byte stream classes.

Stream class	Description
BufferedInputStream	Used for Buffered Input Stream.
BufferedOutputStream	Used for Buffered Output Stream.
DataInputStream	Contains method for reading java standard datatype
DataOutputStream	An output stream that contain method for writing java standard data type
FileInputStream	Input stream that reads from a file
FileOutputStream	Output stream that write to a file.
InputStream	Abstract class that describe stream input.
OutputStream	Abstract class that describe stream output.
PrintStream	Output Stream that contain print() and println() method

These classes define several key methods. Two most important are
read() : reads byte of data.
write() : Writes byte of data.
Character Stream Classes
Character stream is also defined by using two abstract class at the top of hierarchy, they are Reader and Writer.These two abstract classes have several concrete classes that handle unicode character.
Some important Character stream classes

Stream class	Description
BufferedReader	Handles buffered input stream.
BufferedWriter	Handles buffered output stream.
FileReader	Input stream that reads from file.
FileWriter	Output stream that writes to file.
InputStreamReader	Input stream that translate byte to character
OutputStreamReader	Output stream that translate character to byte.
PrintWriter	Output Stream that contain print() and println() method.
Reader	Abstract class that define character stream input
Writer	Abstract class that define character stream output

4. What is a byte stream?
Byte Streams: These handle data in bytes (8 bits) i.e., the byte stream classes read/write data of 8 bits. Using these you can store characters, videos, audios, images etc.
5. What is stream class?
The Stream class defines objects which accepts a sequence of characters. The InputStream is used to read data from a source and the OutputStream is used for writing data to a destination. InputStream and OutputStream are the basic stream classes in Java.
6. Explain Buffered writer class.
BufferedWriter is a sub class of java.io.Writer class.
BufferedWriter writes text to character output stream, buffering characters so as to provide for the efficient writing of single characters, arrays, and strings.The buffer size may be specified, or the default size may be used.
Class constructors
BufferedWriter(Writer out): Creates a buffered character-output stream that uses a default
sized output buffer with specified Writer object.
BufferedWriter(Writer out, int sz): Creates a buffered character-output stream that uses an output buffer of specified size with specified Writer object.
Class methods

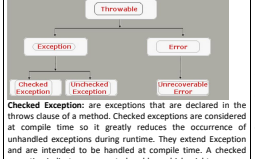
void newLine()	It is used to add a new line by writing a line separator.
void write(int c)	It is used to write a single character.
void write(char[ ] cbuf, int off, int len)	It is used to write a portion of an array of characters.
void write(String s, int off, int len)	It is used to write a portion of a string.
void flush()	It is used to flushes the input stream.
void close()	It is used to closes the input stream

7. What is a thread? Explain different states of thread.
OR
8. Explain in details the life cycle of a thread.
A thread, in the context of Java, is the path followed when executing a program. It is a sequence of nested executed statements or method calls that allow multiple activities within a single process. All Java programs have at least one thread, known as the main threadwhich is created by the Java Virtual Machine (JVM) at the program's start, when the main method is invoked with the main thread.Each thread comes with its own local variables, program counter, and stack, and is used to perform background processing that makes graphical user interface (GUI) application smoother.
states of thread
A java thread can be in any of following thread states during its life cycle i.e. New, Runnable, Blocked, Waiting, Timed Waiting or Terminated. These are also called life cycle events of a thread in java.
New
As soon as, you create new thread, it's in NEW state. It remains in this state until the program starts the thread using it'start() method.At this point, thread is not alive and it's a state internal to Java programming.
Runnable
Calling start() method on thread put it in RUNNABLE state. A thread that is ready to run is moved to runnable state. In this state, a thread might actually be running or it might be ready run at any instant of time. It is the responsibility of the thread scheduler to give the thread, time to run.
A multi-threaded program allocates a fixed amount of time to each individual thread. Each and every thread runs for a short while and then pauses and relinquishes the CPU to another thread, so that other threads can get a chance to run. When this happens, all such threads that are ready to run, waiting for the CPU and the currently running thread lies in runnable state.
Blocked
A RUNNABLE thread transitions to the BLOCKED state when it attempts to perform a task that cannot be completed immediately and it must temporarily wait until that task completes.For example, when a thread issues an input/output request, the operating system blocks the thread from executing until that I/O request completes—at that point, the blocked **thread**
transitions to the RUNNABLE state, so it can resume execution. A blocked thread cannot use a processor, even if one is available.
Waiting
A thread can be put in waiting state for various reasons e.g. calling it's wait() method. Usually, program put a thread in WAIT state because something else needs to be done prior
to what current thread is doing.Once the thread wait state is over or it is, its state is changed to RUNNABLE and it's moved back to thread pool.

Timed Waiting:
A thread lies in timed waiting state when it calls a method with a time out parameter. A thread lies in this state until the timeout is completed or until a notification is received. For example, when a thread calls sleep or a conditional wait, it is moved to a timed waiting state.
Terminated
A thread enters the TERMINATED state (sometimes called the dead state) when it successfully completes its task or otherwise terminated due to any error or even it was forcefully killed.
9. How can we create a thread in java? Explain with example.
OR
10.With an example, explain how to create a child thread by implementing
runnable interface.
OR
11.Explain Runnable interface for creating a thread.
Thread implementation in java can be achieved in two ways:
Extending the java.lang.Thread class
Implementing the java.lang.Runnable Interface
The Thread and Runnable are available in the java.lang.* package
By extending thread class
• The class should extend Java Thread class.
• The class should override the run() method.• The functionality that is expected by the Thread to be executed is written in the run()
method.
void start(): Creates a new thread and makes it runnable.
void run(): The new thread begins its life inside this method.
Example:
public class MyThread extends Thread {
public void run(){
System.out.println("thread is running...");
}
public static void main(String[] args) {
MyThread obj = new MyThread();
obj.start();
}
By Implementing Runnable interface
steps involved in implementing the Runnable interface in Java:
• The first step is to create a class that implements the Runnable interface.
• Now, you need to override the run method in the Runnable class.
• Next, you need to pass the Runnable object as a parameter to the constructor of the
Thread class object while creating it. Now, this Thread object is capable of executing
our Runnable class.
• Finally, you need to invoke the Thread object's start method.
Example:
public class MyThread implements Runnable {
public void run(){
System.out.println("thread is running..."); }
public static void main(String[] args) {
Thread t = new Thread(new MyThread());
t.start();
}
12.What is synchronization and how it is implemented in java?
It is a Java feature that restricts multiple threads from trying to access the commonly shared resources at the same time. Here

shared resources refer to external file contents, class variables or database records.The main purpose of synchronization is to avoid thread interference. At times when more than one thread try to access a shared resource, we need to ensure that resource will be used by only one thread at a time. The process by which this is achieved is called synchronization. The "Synchronized" keyword in java creates a block of code referred to as critical section.Synchronization is widely used in multithreaded programming. We can synchronize our code in either of two ways. Both involve the use of the synchronized keyword.
• Using Synchronized Statement Or Block
• Using Synchronized Methods
Using Synchronized Statement Or Block
Synchronized blocks in Java are marked with the synchronized keyword. A synchronized block in java is synchronized on some object. All synchronized blocks synchronized on the same object can only have one thread executing inside them at a time. All other threads attempting to enter the synchronized block are blocked until the thread inside the synchronized block exits the block.
General Syntax:
synchronized (objRef)
{
//statement to be synchronized)
Here, objRef is a reference to the object being synchronized. A synchronized block ensures that a call to a synchronized method that is a member of objRef's class occurs only after the current thread has successfully entered objRef's monitor.This synchronization is implemented in Java with a concept called monitors. Only one thread can own a monitor at a given time. When a thread acquires a lock, it is said to have entered the monitor. All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor.
Using Synchronized Methods
To make a method synchronized, simply add the synchronized keyword to its declaration:
If you declare any method as synchronized, it is known as synchronized method.Synchronized method is used to lock an object for any shared resource.When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.
public class SynchronizedCounter
{
private int c = 0;
public synchronized void increment() {c++;}public synchronized void decrement() { c--; }
public synchronized int value() {return c; }}
it is not possible for two invocations of synchronized methods on the same object to interleave. When one thread is executing a synchronized method for an object, all other threads that invoke synchronized methods for the same object block (suspend execution) until the first thread is done with the object.
13.What is a daemon thread?
A Daemon thread is a background service thread which runs as a low priority thread and performs background operations like garbage collection. Its life depends on the mercy of user threads i.e., when all the user threads dies, JVM terminates this thread automatically.There are many java daemon threads running automatically e.g.,gc, finalizer etc.The setDaemon() method of the Thread class is used to mark/set a particular thread as either a daemon thread or a user thread.This method must be called before the thread is started.
14.Write the importance of try, catch block in exception handling?
OR
15.Explain try-catch-finally statement.
Try block
The try block contains set of statements where an exception can occur. A try block is always followed by a catch block, which handles the exception that occurs in associated try block. A try block must be followed by catch blocks or finally block or both.
Syntax of try block
try{
//statements that may cause an exception
}
While writing a program, if you think that certain statements in a program can throw a exception, enclosed them in try block and handle that exception
Catch block
A catch block is where you handle the exceptions, this block must follow the try block. A single try block can have several catch blocks associated with it. You can catch different
exceptions in different catch blocks. When an exception occurs in try block, the corresponding catch block that handles that particular exception executes. For example if an arithmetic exception occurs in try block then the statements enclosed in catch block for arithmetic exception executes.
try {
// java code which might cause exception}catch(ExceptionName e1) { // it can be any type of exception
{ //Catch block for handling the exception }
finally { //The finally block always executes irrespective of the exception. }
finally block
Java finally block is a block that is used to execute important code such as closing connection, stream etc.Java finally block is always executed whether exception is handled or not.Java finally block follows try or catch block.
16.Explain the syntax of throws statement.
Syntax of java throws
return_typemethod_name(parameters) throws exception_list{
//method code }
throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception using a try-catch block.exception_list is aexception_class_name comma separated list of all the exceptions which a method might throw.
// Java program to demonstrate working of throws
class ThrowsExcep{
static void fun() throws IllegalAccessExcepton{
System.out.println("Inside fun(). ");
throw new IllegalAccessExcepton("demo");}
public static void main(String args[]){
try{
fun();
}
catch(IllegalAccessExcepton e)
{
System.out.println("caught in main.");} } }

17.Explain exception handling mechanism in java.
OR
18.Explain how exceptions are implemented in java.
Exception handling is one of the most important feature of java programming that allows us to handle the runtime errors caused by exceptions.In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.There can be several reasons that can cause a program to throw exception. For example: Opening a non-existing file in your program, Network connection problem, bad input data provided by user etc.Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOExcepton, SQLExcepton, RemoteExcepton, etc.
The core advantage of exception handling is to maintain the normal flow of the application.
Java Exception hierarchy
The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:



Checked Exception: are exceptions that are declared in the throws clause of a method. Checked exceptions are considered at compile time so it greatly reduces the occurrence of unhandled exceptions during runtime. They extend Exception and are intended to be handled at compile time. A checked exception indicates an expected problem which might occur during normal application execution.
e.g. IOExcepton, SQLExcepton, ClassNotFoundExcepton .
Unchecked Exception: Runtime Exceptions are also known as Unchecked Exceptions. These exceptions are not checked at compile-time so compiler does not check whether the programmer has handled them or not but it's the responsibility of the programmer to handle these exceptions and provide a safe exit.
e.g.ArithmeticExcepton,NullPointerExcepton,
ArrayIndexOutOfBoundsExcepton etc.
Error:
Error is irrecoverable e.g.OutOfMemoryError, VirtualMachineError, AssertionError etc.
Default Exception Handling/How JVM handle an Exception: Whenever inside a method, if an exception has occurred, the method creates an Object known as Exception Object and hands it off to the run-time system(JVM). The exception object contains name and description of the exception, and current state of the program where exception has occurred.
Creating the Exception Object and handling it to the run-time system is called throwing an Exception.There might be the list of the methods that had been called to get to the method where exception was occurred. This ordered list of the methods is called Call Stack.Now the following procedure will happen.
• The run-time system searches the call stack to find the method that contains block of code that can handle the occurred exception. The block of the code is called Exception handler.
{ System.out.println("Parent Exception occurs"); }
• The run-time system starts searching from the method in which exception occurred, proceeds through call stack in the reverse order in which methods were called.
• If it finds appropriate handler then it passes the occurred exception to it. Appropriate handler means the type of the exception object thrown matches the type of the exception object it can handle.
• If run-time system searches all the methods on call stack and couldn't have found the appropriate handler then run-time system handover the Exception Object to default exception handler, which is part of run-time system. This handler prints the exception information in the following format and terminates program abnormally.
Customized Exception Handling/How Programmer handles an exception:
Java exception handling is managed via five keywords: try, catch, throw, throws, and finally. Briefly, here is how they work.
• Program statements that you think can raise exceptions are contained within a try block. If an exception occurs within the try block, it is thrown.
• Your code can catch this exception (using catch block) and handle it in some rational manner. System-generated exceptions are automatically thrown by the Java run-time system.
• To manually throw an exception, use the keyword throw.
• Any exception that is thrown out of a method must be specified as such by a throws clause.
• Any code that absolutely must be executed after a try block completes is put in a finally block.

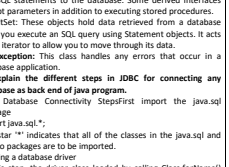
```
public class JavaExceptionExample{
public static void main(String args[]){
try{
  //code that may raise exception
  int data=100/0;
}catch(ArithmeticExcepton e){System.out.println(e);}
  //rest code of the program
  System.out.println("rest of the code...");
 } }
```

19.Explain multiple catch classes in exception handling.
Multi-catch block
A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.
• At a time only one exception occurs and at a time only one catch block is executed.
• All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.
Example

```
public class MultipleCatchBlock3 {
 public static void main(String[] args) {
try{
 int a[]=new int[5];
a[5]=30/0;
 System.out.println(a[10]);
}
catch(ArithmeticExcepton e)
{ System.out.println("ArithmeticExcepton occurs"); }
catch(ArrayIndexOutOfBoundsExcepton e)
{  System.out.println("ArrayIndexOutOfBounds  Exception occurs"); }
catch(Excepton e)
```

4

1. What is JDBC?
JDBC (Java Database Connectivity) is an SQL-based API created by Sun Microsystems to enable Java applications to use SQL for database access.The JDBC API defines a set of Java interfaces that encapsulate major database functionality, such as running queries, processing results, and determining configuration information.These API provide communications between an application residing on a client machine and a data source residing on the same client machine or on another server computer.
2. Briefly explain the JDBC components.
OR
Explain JDBC Architecture.
To communicate with the Database using JDBC need the following components.JDBC DriverManager: The DriverManager class of the java.sql package manages different types of JDBC drivers. This class loads the driver classes. In addition to this,
whenever a new connection establishes it chooses and loads the suitable driver from the previously loaded ones.
JDBC API: It is a Java abstraction which enables applications to communicate with relational databases. It provides two main packages namely, java.sql and javax.sql. It provides classes and methods to connect with a database, create statements (quires), execute statements and handle the results.
JDBC-ODBC Bridge driver: This is a bridge driver which translates the JDBC method calls to ODBC function calls using this package you can communicate with the databases which uses ODBC drivers.

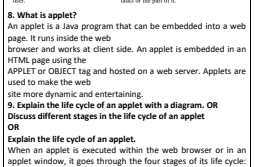


Driver: This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.Connection: This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
Statement: You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
ResultSet: These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
SQLException: This class handles any errors that occur in a database application.
3. Explain the different steps in JDBC for connecting any database as back end of java program.
Java Database Connectivity StepsFirst import the java.sql package
import java.sql.*;
The star '*' indicates that all of the classes in the java.sql and java.io packages are to be imported.
Loading a database driver
In this step, the driver class loaded by calling Class.forName() with the Driver class.
Class is a class in java.lang package.
Syntax: Class.forName("com.mysql.jdbc.Driver");
forName() is static method of the "Class" class . This loads the driver class and returns Class instance. It takes string type value as an argument and matches the class with string.Creating a jdbc Connection
The objects defined by DriverManager class to establish the applications with the JDBC driver. This class manages the JDBC driverswhich is installed on the system. getConnection() is the method by which it can connect. It uses the username, password, and a jdbcurl to make the connection with database.url - Database url where stored or created your database
userName - User name of MySQL
password -Password of MySQL
try{Connection
con=DriverManager.getConnection(url,"userName","password");
}
catch(SQLException x){
System.out.println("e.printStackTrace");
}Creating a jdbc Statement object
When an connection is established then we can interact with the database. Connection interface defines methods for interacting with the database. It used to instantiate a Statement by using the createStatement() method.
Statement st = con.createStatement();
Executing a statement with the Statement object
This interface defines methods which is used to communicate with database. This class has three methods to execute the statements- executeQuery(), executeUpdate(), and execute(). For SELECT statements, executeQuery() method will be used. For create or modify tables, the executeUpdate() method is used. executeQuery() method returns the result of the query in the form of ResultSet object and executeUpdate() method returns the number of rows affected from execution of the query.
ResultSetrs = st.executeQuery("SELECT * from Employee");
Getting ResultSet object Executing the executeQuery() method returns the result in the form of ResultSet object.
We can now operate on this object to extract the rows values returned from the execution of the query. Its next() method sets the pointer to the first row and getX() methods can be used to get different types of data from the result set.
while(rs.next()) {
String data = rs.getString(1);
System.out.println(data);
}
con.close()
This method of Connection interface is used for closing the connection. All the resources will be free occupied by the database.
4. Explain SQL exception class.
The SQLExcepton Class and its subtypes provide information about errors and warnings that occurs when the data source is accessed.
This class is an extension of java.lang.exception and provides information related to failure in database connectivity.
The following information is available from an SQLExcepton: Text Description, SQLState, Error Code, A reference to any other exception that also occur.
An SQLExcepton can occur both in the driver and the database. When such an exception occurs, an object of type SQLExcepton will be passed to the catch clause.catch (SQLExcepton){
System.out.println("SQLExcepton exception: ");
System.out.println("Message:....." + e.getMessage());
System.out.println("SQLState:....." + e.getSQLState());
System.out.println("Vendor Code:." + e.getErrorCode());
e.printStackTrace(); }
The passed SQLExcepton object has the following methods available for retrieving additional information about the exception –
Method Description
getErrorCode( ) Gets the error number associated with the exception.
getMessage( ) Gets the JDBC driver's error message for an error, handled
by the driver or gets the Oracle error number and message for a database error.
getSQLState( ) Gets the XOPEN SQLstate string. For a JDBC driver error,
no useful information is returned from this method. For a database error, the five-digit XOPEN SQLstate code is returned. This method can return null.For example:
08001 where 08 is the class and 001 is the subclass. 08 class means connection exception and 001 means client unable to establish connection.
getNextException( ) Gets the next Exception object in the exception chain.
printStackTrace( ) Prints the current exception, or throwable, and it's backtrace to a standard error stream.5. Explain Statement interface.
The Statement interface provides the method to execute the database queries. After making a connection, Java application can interact with database. The Statement interface contains the ResultSet object.
Creating Statement ObjectStatement st = con.createStatement();
ResultSetrs = st.executeQuery("select * from student");
The PreparedStatementinterface extends the Statement interface. It represents precompiled SQL statements and stores it in a PreparedStatement object.It increases the performance of the application because the query is compiled only once.The PreparedStatement is easy to reuse with new parameters.
Creating PreparedStatement Object
String sql = "Select * from Student where rollNo= ?";
PreparedStatementps = con.prepareStatement(sql); (or)
PreparedStatementps = con.prepareStatement("insert into Student values(?, ?, ?)");
ps.setInt(1, 101);
ps.setString(2, "Suraj");
ps.setString(3, "MCA");
ps.executeUpdate();
All the parameter are represented by "?" symbol and each parameter is referred to by its origin position.
The CallableStatementinterface is used to execute the SQL stored procedure in a database. The JDBC API provides stored procedures to be called in a standard way for all RDBMS.
A stored procedure works like a function or method in a class. The stored procedure makes the performance better because these are precompiled queries.Creating CallableStatement Interface
The instance of a CallableStatement is created by calling prepareCall() method on a Connection object.
For example: CallableStatement cs = con.prepareCall("{call procedures(?,?)}");
6. ExplainResultSetMetaData?
The ResultSetMetaData provides information about the obtained ResultSet object like, the number of columns, names of the columns, datatypes of the columns, name of the table etc...Following are some methods of ResultSetMetaData class.
Method Description
getColumnCount() Retrieves the number of columns in the current ResultSet object.
getColumnLabel() Retrieves the suggested name of the column for use.getColumnName() Retrieves the name of the column.
getTableName() Retrieves the name of the table.
The getMetaData() method of ResultSet interface returns the object of ResultSetMetaData.
The following code fragment creates the ResultSet object rs,creates the ResultSetMetaData object rsmd, and uses rsmd to find out how many columns rs has,column name and columntype.

Statement ps=con.Statement("select * from emp");
ResultSetrs=ps.executeQuery();
ResultSetMetaDatarsmd=rs.getMetaData();
System.out.println("Total columns: "+rsmd.getColumnCount());
System.out.println("Column Name of 1st column: "+rsmd.getColumnName(1));
System.out.println("Column Type Name of 1st column: "+rsmd.getColumnTypeName(1));
7. Differentiate java application and applet.

Java Application	Java Applet
Applications are just like a Java programs that can be execute independently without using the web browser.	Applets are small Java programs that are designed to be included with the HTML web document. They require a Java-enabled web browser for execution.
Application program requires a main function for its execution.	Applet does not require a main function for its execution.
Java application programs have the full access to the local file system and network.	Applets don't have local disk and network access.
Applications can access all kinds of resources available on the system.	Applets can only access the browser specific services. They don't have access to the local system.
Applications can executes the programs from the local system.	Applets cannot execute programs from the local machine.
Java Application	Java Applet
An application program is needed to perform some task directly for the user.	An applet program is needed to perform small tasks or the part of it.

8. What is applet?
An applet is a Java program that can be embedded into a web page. It runs inside the web
browser and works at client side. An applet is embedded in an HTML page using the
APPLET or OBJECT tag and hosted on a web server. Applets are used to make the web
site more dynamic and entertaining.
9. Explain the life cycle of an applet with a diagram. OR
Discuss different stages in the life cycle of an applet
OR
Explain the life cycle of an applet.
When an applet is executed within the web browser or in an applet window, it goes through the four stages of its life cycle: Born, Running, Idle and Dead. These stages correspond to the applet methods init(), start(), stop() and destroy() respectively. All these methods defined in the Applet class which is called automatically by the browser or the applet viewer controlling the applet.The life cycle of an applet is as shown in the figure below:



As shown in the above diagram, the life cycle of an applet starts with init() method and ends with destroy() method. Other life cycle methods are start(), stop() and paint(). The methods to execute only once in the applet life cycle are init() and destroy(). Other methods execute multiple times.
init(): The init() method is the first method to execute when the applet is executed. Variable declaration and initialization operations are performed in this method.
start(): The start() method contains the actual code of the applet that should run. The start() method executes immediately after the init() method. It also executes whenever the applet is restored, maximized or moving from one tab to another tab in the browser.
stop(): The stop() method stops the execution of the applet. The stop() method executes when the applet is minimized or when moving from one tab to another in the browser.
destroy(): The destroy() method executes when the applet window is closed or when the tab containing the webpage is closed. stop () method executes just before when destroy() method is invoked. The destroy () method removes the applet object from memory.
paint() The paint() method is used to redraw the output on the applet display area. The paint() method executes after the execution of start() method and whenever the applet or browser is resized.
10. Write about applet skeleton.
An Applet SkeletonMost applets override these four methods. These four methods forms Applet lifecycle.
init() : init() is the first method to be called. This is where variable are initialized. This method is called only once during the runtime of applet.
start() : start() method is called after init(). This method is called to restart an applet after it has been stopped.
stop() : stop() method is called to suspend thread that does not need to run when applet is not visible.
destroy() : destroy() method is called when your applet needs to be removed completely from memory.
Example of an Applet Skeleton

```
import java.awt.*;
import java.applet.*;
public class AppletTest extends Applet{
public void init(){
System.out.println("Applet initialized");}
public void start (){
 //start or resume execution
System.out.println("Applet execution started");}
public void stop(){
 //suspend execution
System.out.println("Applet execution stopped");}{
public void destroy(){
 //perform shutdown activity
{ System.out.println("Applet destroyed");}
public void paint (Graphics g){
 //display the content of window
System.out.println("Painting...");}}
```

The method execution sequence when an applet is executed is:
•init()
•start()
•paint()
The method execution sequence when an applet is closed is:
•stop() •destroy()
11. What is an applet? Explain its working with examples.
An applet is a special kind of Java program that runs in a Java enabled browser. This is the first Java program that can run over the network using the browser. Applet is typically embedded inside a web page and runs in the browser.Example of an Applet program

```
import java.applet.*;
```

```
import java.awt.*;
public class MyApplet extends Applet {
int height, width;
public void init() {
 height = getSize().height;
 width = getSize().width;
setName("MyApplet");}
public void paint(Graphics g)
{
g.drawRoundRect(10, 30, 120, 120, 2, 3);
}
}
```

Every Applet application must import two packages - java.awt and java.applet.• java.awt.* imports the Abstract Window Toolkit (AWT) classes. Applets interact with the user (either directly or indirectly) through the AWT. The AWT contains support for a window-based, graphical user interface.

• java.applet.* imports the applet package, which contains the class Applet. Every applet that you create must be a subclass of Applet class.

• The class in the program must be declared as public, because it will be accessed by code that is outside the program.Every Applet application must declare a paint() method. This method is defined by AWT class and must be overridden by the applet. The paint() method is called each time when an applet needs to redisplay its output.

• An Applet class does not have any main() method. It is viewed using JVM. The JVM can use either a plug-in of the Web browser or a separate runtime environment to run an applet application.

• JVM creates an instance of the applet class and invokes init() method to initialize an Applet.

12. Discuss the steps involved in developing and running a local applet.

There are two ways to run an applet

• By html file.

• By appletViewer tool (for testing purpose).

Simple example of **Applet by html file:**

To execute the applet by html file, create an applet program and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{
 public void paint(Graphics g){
g.drawString("welcome",150,150);
} }
```

myapplet.html

```
<html>
<body>
<applet code="First.class" width="300" height="300"></applet>
</body>
</html>
```

Simple example of **Applet by appletviewer tool:**

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{
public void paint(Graphics g){
g.drawString("welcome to applet",150,150); }}
```

```
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java
c:\>appletviewer First.java
```

13. What is the use of paint() method?

The paint() method is used to redraw the output on the applet display area. The paint() method executes after the execution of start() method and whenever the applet or browser is resized.

The method paint() gives us access to an object of type Graphics class. Using the object of the Graphics class, we can call the method of the Graphics class to write a text or draw various shapes etcin the applet window.

14. Explain how you will pass parameters to applets with an example.

ORHow will you pass parameters to applets?

Parameters specify extra information that can be passed to an applet from the HTML page.

Parameters are specified using the HTML's param tag.

Parameters are passed on an applet When it is loaded. we can define the init( )

Method in the applet to get hold of the parameter defined in the tags.

Param Tag

The <param> tag is a sub tag of the <applet> tag.

The <param> tag contains two attributes: name and value which are used to specify the name of the parameter and the value of the parameter respectively.

For example, the param tags for passing name and age parameters looks as shown below:

```
<param name="name" value="john" />
<param name="age" value="25" />
```

getParameter() Method

The getParameter() method of the Applet class can be used to retrieve the parameters passed from the HTML page. The syntax of getParameter() method is as follows:

```
String getParameter(String param-name)
```

```
import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet
{
String n;
String a;
public void init()
{
n = getParameter("name");
a = getParameter("age");
}
public void paint(Graphics g)
{
g.drawString("Name is: " + n, 20, 20);
g.drawString("Age is: " + a, 20, 40);
}}
```

```
/*
<applet code="MyApplet" height="300" width="500">
<param name="name" value="John" />
<param name="age" value="25" />
</applet>
*
```

5

1. What is event handling?

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events.

2. What do you mean by delegation event model?OR

Briefly explain the delegation event model.

Event model is based on the concept of an 'Event Source' and 'Event Listeners'.

Any object that is interested in receiving messages (or events) is called an Event Listener Any object that generates these messages (or events) is called an Event Source The Event Delegation model is based on – The Event Classes, The Event Listeners, Event Objects.

There are three participants in event delegation model in Java;

• Event Source – the class which broadcasts the events.

• Event Listeners – the classes which receive notifications of events

• Event Object – the class object which describes the event.

This model is referred to as the Delegation Event Model which defines a logical approach to handle events. It is based on the concept of source and listener. A source generates an

event and sends it to one or more listeners. On receiving the event, listener processes the event and returns it. The notable feature of this model is that the source has a registered list of listeners which will receive the events as they occur. Only the listeners that have been registered actually receive the notification when a specific event is generated.

For example,

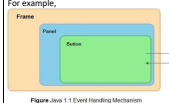


Figure Java 1.1 Event Handling Mechanism

as shown in Figure, when the mouse is clicked on Button, an event is generated. If the Button has a registered listener to handle the event, this event is sent to Button, processed and the output is returned to the user. However, if it has no registered listener the event will not be propagated upwards to Panel or Frame.

3. How event handling is done in java?

In Java when works with the AWT components like button, textbox, etc (except panel, label) generates an event. This event is handled by the listener. Event listener listens the event generated on components and performs the corresponding action.In Java event handling may comprised the following four classes :

Event Sources:Sources for generating an event may be the components. In Java java.awt.Component specifies the components that may or may not generate events.

These components classes are the subclass of the above class. These event sources may be the button, combobox, textbox etc.

Event Classes:Event Classes in Java are the classes defined for almost all the components that may generate events. These events classes are named by giving the specific name such as for the component source button the event class is ActionEvent.

Following are the list of Event Classes:ActionEvent : Button, TextField, List, Menu

WindowEvent : Frame

ItemEvent : Checkbox, List

AdjustmentEvent : Scrollbar

MouseEvent : MouseKeyEvent : Keyboard

Event Listeners:Event Listeners are the Java interfaces that provides various methods to use in the implemented class. Listeners listens the event generated by a component. In Java almost all components have its own listener that handles the event generated by the component. For example, there is a Listener named ActionListener handles the events generated from button, textfield, list, menus.Event Adapters:Event Adapters classes are abstract class that provides some methods used for avoiding the heavy coding. Adapter class is defined for the listener that has more than one abstract methods.

4. What is an event listener?

Event Listeners are the Java interfaces that provides various methods to use in the implemented class. Listeners listens the event generated by a component. In Java almost all components have its own listener that handles the event generated by the component.

For example, there is a Listener named ActionListener handles the events generated from button, textfield, list, menus.

5. Define adapter classes?

An adapter class provides the default implementation of all methods in an event listener interface. Adapter classes are very useful when you want to process only few of the events that are handled by a particular event listener interface.If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So, it saves code. The adapter classes are found in java.awt.event packages.

6. Explain various event classes and event listener interfaces in java.

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc.

The java.awt.event package provides many event classes and Listener interfaces for event handling. Some commonly used Event Classes and Listener InterfacesActionEvent ActionListener MouseEvent MouseListener and MouseMotionListener

KeyEvent KeyListener

ItemEvent ItemListener

TextEvent TextListener

WindowEvent WindowListener

ContainerEvent ContainerListener

ActionEvent: Represents a graphical element is clicked, such as a button or item in a list.

Related listener: ActionListener.

ItemEvent: Represents an event which indicates that an item was selected or deselected.

For example, click on checkbox. Related listener: ItemListener

TextEvent: event which indicates that an object's text changed.

For example when a value in textfield or textarea is entered or edited. Related listener: TextListener

ContainerEvent: Represents an event that occurs to the GUI's container itself, for example, if a user adds or removes an object from the interface. Related listener:

ContainerListener.

KeyEvent: Represents an event in which the user presses, types or releases a key. Related

listener: KeyListener.

WindowEvent: Represents an event relating to a window, for example, when a window is closed, activated or deactivated. Related listener: WindowListener.

MouseEvent: Represents any event related to a mouse, such as when a mouse is clicked

or pressed. Related listener: MouseListener.

7. Mention the list of commonly used containers while designing GUI and AWT. Also explain any one of the containers with an example.

The Container class is a subclass of Component class. It is used to contain various Component objects as well as other Container objects within it.

The Container class object groups, manages, and positions components and treats them as a unit.

Two immediate subclasses of Container class are:

Panel: It is used for grouping components. An Applet is a subclass of Panel.

Window: It is used for creating and handling windows. Two subclasses of Window are Dialog and Frame.

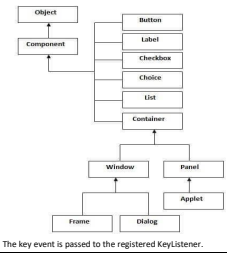
simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

```
import java.awt.*;
class First extends Frame{
First(){
Button b=new Button("click me"); b.setBounds(30,100,80,30);//
setting button position
add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300 height
setLayout(null);//no layout manager
setVisible(true);//now frame will be visible, by default not visible
}
public static void main(String args[]){
First f=new First();
}}
```

8. Explain the structure of AWT.

Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java.The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

The hierarchy of Java AWT classes are given below.



The key event is passed to the registered KeyListener.

Component: Component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. For examples buttons, checkboxes, list and scrollbars of a graphical user interface.Container: The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.Window: The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel: The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame: The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

9. Explain choice control.

Choice control is used to show pop up menu of choices. Selected choice is shown on the top of the menu. It inherits Component class.

When user clicks on its list of choices appears. The user can select only one of the items contained in the list and only the selected item is displayed.To add items to the choice list , add() method is used which takes the name of the item to be added as argument.

Choice constructors:

Choice() Creates a new choice menu.

Choice methods:

void add(String item) Adds an item to this Choice menu.

void addItemListener(ItemListener l) Adds the specified item listener to receive item events from this Choice menu.

String getItem(int index) Gets the string at the specified index in this Choice menu.

int getItemCount() Returns the number of items in this Choice menu.

void remove(int position) Removes an item from the choice menu at the specified position.

Example:

```
import java.awt.*;
public class ChoiceExample {
 ChoiceExample(){
 Frame f= new Frame();
 Choice c=new Choice();
 c.setBounds(100,100, 75,75);
 c.add("Item 1");
 c.add("Item 2");
 c.add("Item 3");
 c.add("Item 4");
 c.add("Item 5"); f.add(c);
 f.setSize(400,400);
 f.setLayout(null);
 f.setVisible(true); }
public static void main(String args[]) {
 new ChoiceExample(); }
```

10. Write short note on TextField and TextArea classes in java.awt.package

Text Field

The textField component allows the user to edit single line of text.

When the user types a key in the text field the event is sent to the TextField.

The key event may be key pressed, Key released or key typed.

Example: na=new TextField(20);

Class constructors

TextField() Constructs a new text field.

TextField(int columns) Constructs a new empty text field with the specified number of columns.

TextField(String text) Constructs a new text field initialized with the specified text.

TextField(String text, int columns) Constructs a new text field initialized with the

specified text to be displayed, and wide enough to hold the specified number of columns.

TextField Methods

String getText(): It is used to obtain the string currently contained in the text field.

void setText(String str): It is used to set the text. Here, str is the new String.

void select(int startIndex, int endIndex): It is used to select a portion of the text under program control. It selects the characters beginning at startIndex and ending at endIndex-1.

String getSelectedText(): It returns the currently selected text.

boolean isEditable(): It is used to determine editability. It returns true if the text may be changed and false if not.

Text Area

A text area is used for the editing of multiple lines of text.

The only difference between the Text field and Text area is that Text Field is used for editing a single line of text within the user interface while a Text Area is used for editing multiple lines of text.

Example:

```
TextArea area=new TextArea("Welcome to the universe");
area.setBounds(10,30, 300,300);Class constructors
```

TextArea() Constructs a new text area with the empty string as text.

TextArea(int rows, int columns) Constructs a new text area with the specified number of rows and columns and the empty string as text.

TextArea(String text) Constructs a new text area with the specified text.

TextArea(String text, int rows, int columns) Constructs a new text area with the specified

text, and with the specified number of rows and columns.

TextArea(String text, int rows, int columns, int scrollbars) Constructs a new text area

with the specified text, and with the rows, columns, and scroll bar visibility as specified.

Scrollbars can take one of these values- SCROLLBARS_BOTH, SCROLLBARS_NONE, SCROLLBARS_HORIZONTAL_ONLY, SCROLLBARS_VERTICAL_ONLY.

TextArea Methods

TextArea is a subclass of TextComponent. Therefore, it supports the getText( ), setText( ), getSelectedText( ), select( ), isEditable( ), and setEditable( ) methods described in the

TextField section. TextArea adds the following methods:

void append(String str): It appends the string specified by str to the end of the current text.

void insert(String str, int index): It inserts the string passed in str at the specified index.

voidReplaceRange(String str, int startIndex, int endIndex): It is used to replace the text. It replaces the characters from startIndex to endIndex-1.

11. Explain the AWT controls Button and TextField with their constructors.

TextField explained in above question 10

Button is a control component that has a label and generates an event when pressed.

When a button is pressed and released, AWT sends an instance of ActionEvent to the button, by calling processEvent on the button.

If an application wants to perform some action based on a button being pressed and released, it should implement ActionListener and register the new listener to receive

events from this button, by calling the button's addActionListener method.

Example:

```
a1=new Button("submit");
a2=new Button("cancel");
```

Class constructors

Button() Constructs a button with an empty string for its label.

Button(String text) Constructs a new button with specified label.

Button Methods :

void setLabel(String str): You can set its label by calling setLabel(). Here, str is the new Label for the button.

String getLabel(): You can retrieve its label by calling getLabel() method.12. Explain various AWT controls.

The Components that allow a user to interact with a GUI based application are called controls.

The various controls supported by AWT are Button,textField, TextArea, Choice, Checbox, List, Label, Scroll bars, Menu bar etc.

Button - Explained in question 11

TextField - Explained in question 10

TextArea - Explained in question 10

Choice - Explained in question 9

Checkbox

It is a control that consists of a combination of a small box and a label.

The label provides the description of the box with which it is associated.

It is a two state control having states true(checked) and false(unchecked).

The state of a checkbox can be changed by clicking on it.

Checkbox is an object of Chechbox class.

Class constructors

Checkbox() Creates a check box with an empty string for its label.

Checkbox(String label) Creates a check box with the specified label.

Checkbox(String label, boolean state) Creates a check box with the specified label and sets the specified state.

Checkbox(String label, boolean state, CheckboxGroup group) Constructs a Checkbox with the specified label, set to the specified state, and in the specified check box group.

Checkbox(String label, CheckboxGroup group, boolean state) Creates a check box with the specified label, in the specified check box group, and set to the specified state.

Example

```
Checkbox chkApple = new Checkbox("Apple");
Checkbox chkMango = new Checkbox("Mango");
f.add(chkApple);
f.add(chkMango);
```

Methods of Checkbox

boolean getState(): To retrieve the present state of a checkbox

void setState(boolean on): To line its state, call setState(). Here, if one is true, the box is checked. If it's false, the box is cleared.

String getLabel(): you'll obtain the present label related to a checkbox by calling getLabel().

void setLabel(String str): To line the label, call setLabel(). The string passed in str becomes the new label related to the invoking checkbox.

ListIt is a control which displays a list of items.

It allows the user to make multiple selections from the given list of items.

Using list any number of items can be displayed and multiple selection can be made.

List is an object of List class.

To add items to the list, add() method is used.

Example

```
List l1=new List(4);
l1.setBounds(100,100, 75,75);
```

Class constructors

List() Creates a new scrolling list.

List(int rows) Creates a new scrolling list initialized with the specified number of visible lines.

List(int rows, boolean multipleMode) Creates a new scrolling list initialized to display the specified number of rows.

Method of Listsvoid add(String name): To add a selection to the list, use add(). Here, the name is that the name of the item being added. It adds items to the end of the list.

void add(String name, int index): It also adds items to the list but it adds the items at the index specified by the index.

String getSelectedItem(): It determines which item is currently selected. It returns a string containing the name of the item. If more than one item is selected, or if no selection has been made yet, null is returned.

int getSelectedIndex(): It determines which item is currently selected. It returns the index of the item. The first item is at index 0. If more than one item is selected, or if no selection has yet been made, -1 is returned.

Label

It is a simple control which is used to display text(non-editable) on the window.

Labels are passive controls that do not support any interaction with the user.

Creating Label : Label l = new Label(String);

Label Constructors:

Label() : It creates a blank label.

Label(String str) : It creates a label that contains the string specified by str.

Label(String str, int how): It creates a label that contains the string specified by str

using the alignment specified by how.

The value of how must be one of these three constants: LEFT, Label.RIGHT,

Label.CENER.

Label Methods:void setText(String str): It is used to set or change the text in a label by using the setText() method. Here, str specifies the new label.

String getText(): It is used to obtain the current label by calling getText() method.

Here, the current label is returned.

void setAlignment(int how): It is used to set the alignment of the string within the label by calling setAlignment() method. Here, how is one of the alignment

constants?

int getAlignment(): It is used to obtain the current alignment, getAlignment() is called.

Scroll bars

Scrollbars are used to select continuous values between a specified minimum and maximum. Scroll bars may be oriented horizontally or vertically. A scroll bar is really a composite of several individual parts.

Each end has an arrow that you simply can click to get the present value of the scroll bar one unit within the direction of the arrow. The current value of the scroll bar relative to its minimum and maximum values is indicated by the slider box for the scroll bar.

Scrollbars are encapsulated by the Scrollbar class.

Creating Scrollbar : Scrollbar sb = new Scrollbar();

Scrollbar Constructor

Scrollbar(i) It creates a vertical scrollbar.

Scrollbar(int style) It allows you to specify the orientation of the scrollbar. If style is Scrollbar.VERTICAL, a vertical scrollbar is created. If a style is Scrollbar.HORIZONTAL, the scroll bar is horizontal.

Scrollbar(int style, int initialValue, int thumbSize, int min, int max) Here, the initial value of the scroll bar is passed in initialValue. The number of units represented by the peak of the thumb is passed in thumbSize. The minimum and maximum

values for the scroll bar are specified by min and max.

Scrollbar Methods

void setValues(int initialValue, int thumbSize, int min, int max): It is used to set the parameters of the constructors.

int getValue(): It is used to obtain the current value of the scroll bar. It returns the current setting.

void setValue(int newValue): It is used to set the current value. Here, newValue specifies the new value for the scroll bar. When you set a worth, the slider box inside the scroll bar is going to be positioned to reflect the new value.int getMaximum(): It is used to retrieve the minimum values. They return the requested quantity. By default, 1 is the increment added to the scroll bar.

int getMaximun(): It is used to retrieve the maximum value. By default, 1 is the increment subtracted from the scroll bar.

Menu bar

A menu bar can be created using MenuBar class.

A menu bar may contain one or multiple menus, and these menus are created using Menu class.

A menu may contain one of multiple menu items and these menu items are created using MenuItem class.

Simple constructors of MenuBar, Menu and MenuItem

public MenuBar() Creates a menu bar to which one or many menus are added.

public Menu(String title) Creates a menu with a title.

public MenuItem(String title) Creates a menu item with a title.

Class methods

Menu add(Menu m) Adds the specified menu to the menu bar.

int getMenuCount() Gets the number of menus on the menu bar.

void remove(MenuComponent m) Removes the specified menu component from this menu bar.

13. Name the various layout managers.

OR

14. What is layout manager in AWT? Explain.

OR

15. Differentiate between Flow layout, Border layout and Grid layout.

OR

16. Explain four layout managers.

Layout manager are used to arrange components within the container. It automatically places the control at a particular position within window.

LayoutManager is an interface implemented by all the classes of layout managers.

Following are the different classes of Layout manager in AWT

1. BorderLayout
2. CardLayout
3. FlowLayout
4. GridLayout

The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. It is used add method to add the component at specified region. Components of the BorderLayout Manager:

- BorderLayout.NORTH• BorderLayout.SOUTH
- BorderLayout.EAST
- BorderLayout.WEST
- BorderLayout.CENTER

BorderLayout Constructor: - BorderLayout()It is used to create a new border layout with no gaps between components.

The CardLayout manages the components in form of stack and provides visibility to only one component at a time. It treats each component as a card that is why it is known as CardLayout.

CardLayoutConstructors

- CardLayout(): creates a card layout with zero horizontal and vertical gap.
- CardLayout(int hgap, int vgap): creates a card layout with the given horizontal and vertical gap.

The FlowLayout is used to arrange the components in a line, one after another (in a flow).

It is the default layout of applet or panel.

Following are the possible values in FlowLayout manager.

LEFT, RIGHT, CENTER, LEADING, TRAILING

FlowLayout Constructors

- FlowLayout(): creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
- FlowLayout(int align): creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
- FlowLayout(int align, int hgap, int vgap): creates a flow layout with the given alignment and the given horizontal and vertical gap.The GridLayout manager is used to arrange the components in the two-dimensional grid. Each component is displayed in a rectangle.

GridLayout Constructors

- GridLayout(): creates a grid layout with one column per component in a row.
- GridLayout(int rows, int columns): creates a grid layout with the given rows and columns but no gaps between the components.
- GridLayout(int rows, int columns, int hgap, int vgap): creates a grid layout with the given rows and columns along with given horizontal and vertical gaps.17. Which are the graphics methods to draw a line and rectangle?

java.awt.Graphics class

The Graphics class is the abstract super class for all graphics contexts which allow an application to draw onto components. Graphics methods for drawing line, rectangle etc., are shownbelow. For each drawing method that requires a width and height parameter, the width and height must benonnegative values.

public void drawLine(int x1, int y1, int x2, int y2)

Draws a line between the point (x1, y1) and the point (x2, y2).

public void drawRect(int x, int y, int width, int height)

Draws a rectangle of the specified width and height. The top-left corner of the rectangle has the coordinates (x, y). Only the outline of the rectangle is drawn using the Graphics object's colorthe body of the rectangle is not filled with this color.

public void fillRect(int x, int y, int width, int height)

Draws a filled rectangle with the specified width and height. The top-left corner of the rectangle has the coordinate (x, y). The rectangle is filled with the Graphics object's color.

public void drawOval(int x, int y, int width, int height)

Draws an oval in the current color with the specified width and height. The bounding

rectangle's top-left corner is at the coordinates (x, y). The oval touches all four sides of

the bounding rectangle at the center of each side . Only the outline of the shape is drawn.

public void fillOval(int x, int y, int width, int height)

Draws a filled oval in the current color with the specified width and height. The bounding rectangle's top-left corner is at the coordinates (x, y). The oval touches all four sides of

the bounding rectangle at the center of each side.

public void drawRoundRect(int x, int y, int width, int height, int arcWidth,

int arcHeight)

Draws a filled rectangle with rounded corners in the current color with thespecified width and height. The arcWidth and arcHeight determine the rounding of the corners . Only the outline of the shape is drawn.

public void fillRoundRect(int x, int y, int width, int height, int arcWidth,

int arcHeight)

Draws a filled rectangle with rounded corners in the current color with thespecified width and height. The arcWidth and arcHeight determine the rounding of the corners18. Write a java program to drawoval, rectangle etc.

```
import java.applet.*;
import java.awt.*;
public class Shapes extends Applet {
public void paint(Graphics g) {
g.drawLine(30,30,200,10);
g.drawOval(150,50,100,100);
g.drawRect(400,50,200,100);
}
```