1. Explain in detail the features of Java Programming.

- **Simple** – Java is designed to be easy for the professional programmer to learn and use.
- **Object-oriented**: a clean, usable, pragmatic approach to objects, not restricted by the need for compatibility with other languages.
- **Robust**: restricts the programmer to find the mistakes early, performs compile-time (strong typing) and run-time (exception-handling) checks, manages memory automatically.
- **Multithreaded**: supports multi-threaded programming for writing program that perform concurrent computations
- **Architecture-neutral**: Java Virtual Machine provides a platform independent environment for the execution of Java byte code
- **Interpreted and high-performance**: Java programs are compiled into an intermediate representation – byte code:
  a) can be later interpreted by any JVM
  b) can be also translated into the native machine code for efficiency.

- **Distributed**: Java handles TCP/IP protocols, accessing a resource through its URL much like accessing a local file.
- **Dynamic**: substantial amounts of run-time type information to verify and resolve access to objects at run-time.
- **Secure**: programs are confined to the Java execution environment and cannot access other parts of the computer.
- **Portability**: Many types of computers and operating systems are in use throughout the world—and many are connected to the Internet.
- For programs to be dynamically downloaded to all the various types of platforms connected to the Internet, some means of generating portable executable code is needed. The same mechanism that helps ensure security also helps create portability.
- Indeed, Java's solution to these two problems is both elegant and efficient.

2. ***What are the command line arguments used in Java? Write a java program,To demonstrate the use of command line arguments.***

Command Line Arguments

- Passing information into a program when we run it.
- The task is accomplished by passing command line arguments.
- A command line argument is the information that directly follows the programs name on the command-line when it is executed.

Usage of Command Line Arguments

- To Access Command-Line Arguments inside a java program is quite easy.
- They are stored as Strings in the String array passed to main()
- We need to convert it into desired format, using wrapper classes.

```
class CommandLineExample{
public static void main(String args[]){
System.out.println("Your first argument is: "+args[0]);
}
}
```

3. ***Explain the building blocks of Java Programming.***
   Basic Building Blocks of Java

   Statements – *A statement is some action or sequence of actions, given as a command in code. A statement ends with a semi-colon (;).*
   Blocks – A block is a set of statements enclosed in set braces { }. Blocks can be nested.
   Classes – *A class is a blueprint for building objects in Java.*
   Every Java program has at least one class.
   Programmers can define new classes
   There are many pre-built classes in the Java SDK
   Methods – *A method is a function (i.e. subroutine) that belongs to a class. In Java, all functions are methods, meaning they are always contained in some class*
   A Java program can be made up of multiple classes, spread across multiple code files, and it will typically make use of some SDK libraries as well
   The main method – *Every Java application must have a main method, which defines where the program begins. In Java, the main method belongs to a class. Any class can have a main method. The main method looks like this:*

```
Public static void main(String[] args)
{
  // statements
}
```

5 the following: (i) super (ii) final (iii) Garbage Collection

Super

The super keyword in Java is a reference variable which is used to refer immediate parent class object.

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

Final

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

Garbage Collection

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

6 Write a java program to perform matrix multiplication Program

```
public class MatrixMultiplicationExample{
public static void main(String args[]){
//creating two matrices
int a[][]={{1,1,1},{2,2,2},{3,3,3}};
int b[][]={{1,1,1},{2,2,2},{3,3,3}};

//creating another matrix to store the multi
int c[][]=new int[3][3];  //3 rows and 3 columns

//multiplying and printing multiplication of 2 matrices
for(int i=0;i<3;i++){
for(int j=0;j<3;j++){
c[i][j]=0;
for(int k=0;k<3;k++)
{
c[i][j]+=a[i][k]*b[k][j];
}//end of k loop
System.out.print(c[i][j]+" ");  //printing matrix element
}//end of j loop
System.out.println();//new line
}
}}
```

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

7 What is a string buffer class? Explain various methods associated with Strings and string buffer.

StringBuffer Class

- Java StringBuffer class is used to create mutable (modifiable) String objects.
- The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed.
- StringBuffer class creates strings of flexible length that can be modified in terms of both length and content.
- StringBuffer may have characters and substrings inserted in the middle or appended to the end.
- StringBuffer automatically grows to make room for such additions

Modifier and Type	Method	Description
public synchronized StringBuffer	append(String s)	It 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.
public synchronized StringBuffer	insert(int offset, String s)	It 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.
public synchronized StringBuffer	replace(int startIndex, int endIndex, String str)	It is used to replace the string from specified startIndex and endIndex.
public synchronized StringBuffer	delete(int startIndex, int endIndex)	It is used to delete the string from specified startIndex and endIndex.
public synchronized StringBuffer	reverse()	is used to reverse the string.
public int	capacity()	It is used to return the current capacity.
public void	ensureCapacity(int minimumCapacity)	It is used to ensure the capacity at least equal to the given minimum.
public char	charAt(int index)	It is used to return the character at the specified position.

public int	length()	It is used to return the length of the string i.e. total number of characters.
public String	substring(int beginIndex)	It is used to return the substring from the specified beginIndex.
public String	substring(int beginIndex, int endIndex)	It is used to return the substring from the specified beginIndex and endIndex.

8 Write a Java program to implement constructor and constructor overloading.

Constructor

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

```
Bike is created
```

Constructor overloading

```java
public class Marvel{
    Marvel(){
        System.out.println("Welcome to Marvel");
    }
    Marvel(String name){
        System.out.println("My name is "+name);
    }
    Marvel(String name, int age){
        System.out.println(name+" is "+age +" Years old");
    }
    public static void main(String args[]){
        Marvel obj = new Marvel();
        Marvel a = new Marvel("Ironman");
        Marvel b = new Marvel("Thor", 24);
    }
}
```

```
Welcome to Marvel
My name is Ironman
Thor is 24 Years old
PS C:\Users\Sandeep\Desktop\JAVA>
```

9. What are the benefits of inheritance? Explain various forms of inheritance.

Benefits of Inheritance

- Software Reusability (among projects)
- Increased Reliability (resulting from reuse and sharing of well-tested code)
- Code Sharing (within a project)
- Consistency of Interface (among related objects)
- Software Components
- Rapid Prototyping (quickly assemble from pre-existing components)
- Polymorphism and Frameworks (high-level reusable components)
- Information Hiding

Forms of Inheritance

- Inheritance is used in a variety of way and for a variety of different purposes .
  - Inheritance for Specialization
  - Inheritance for Specification
  - Inheritance for Construction
  - Inheritance for Extension
  - Inheritance for Limitation
  - Inheritance for Combination
- One or many of these forms may occur in a single case.

Forms of Inheritance (- ***Inheritance for Specialization*** -)

- Most commonly used inheritance and sub classification is for specialization.
- Always creates a subtype, and the principles of substitutability is explicitly upheld.

(- ***Inheritance for Specification*** -)

- This is another most common use of inheritance. Two different mechanisms are provided by Java, ***interface*** and ***abstract***, to make use of *subclassification for specification*. Subtype is formed and substitutability is explicitly upheld.
- Mostly, not used for refinement of its parent class, but instead is used for definitions of the properties provided by its parent.

(- ***Inheritance for Construction*** -)

- Child class inherits most of its functionality from parent, but may change the name or parameters of methods inherited from parent class to form its interface.
- This type of inheritance is also widely used for code reuse purposes. It simplifies the construction of newly formed abstraction but is not a form of subtype, and often violates substitutability.

(- ***Inheritance for Extension*** -)

- Subclassification for extension occurs when a child class only adds new behavior to the parent class and does not modify or alter any of the inherited attributes.
- Such subclasses are always subtypes, and substitutability can be used.

10. Differentiate between method overriding and method overloading

Method overloading	Method overriding
1. More than one method with same name, different prototype in same scope is called method overloading.	1. More than one method with same name, same prototype in different scope is called method overriding.
2. In case of method overloading, parameter must be different.	2. In case of method overriding, parameter must be same.
3. Method overloading is the example of compile time polymorphism.	3. Method overriding is the example of run time polymorphism.
4. Method overloading is performed within class.	4. Method overriding occurs in two classes.
5. In case of method overloading, Return type can be same or different.	5. In case of method overriding Return type must be same.
6. Static methods can be overloaded which means a class can have more than one static method of same name.	6. Static methods cannot be overridden, even if you declare a same static method in child class it has nothing to do with the same method of parent class.
7. Static binding is being used for overloaded methods	7. dynamic binding is being used for overridden/overriding methods.

11. Discuss primitive data types in JAVA

1. boolean type

- The `boolean` data type has two possible values, either `true` or `false`.
- Default value: `false`.

2. byte type

- The `byte` data type can have values from **-128** to **127** (8-bit signed two's complement integer).
- Default value: 0

3. short type

- The `short` data type in Java can have values from **-32768** to **32767** (16-bit signed two's complement integer).
- Default value: 0

4. int type

- The `int` data type can have values from **-2^{31}** to **$2^{31}-1$** (32-bit signed two's complement integer).
- Default value: 0

5. long type

- The `long` data type can have values from **-2^{63}** to **$2^{63}-1$** (64-bit signed two's complement integer).
- Default value: 0

6. double type

- The `double` data type is a double-precision 64-bit floating-point.
- Default value: 0.0 (0.0d)

7. float type

- The `float` data type is a single-precision 32-bit floating-point. Learn more about single-precision and double-precision floating-point if you are interested.
- Default value: 0.0 (0.0f)

8. char type

- It's a 16-bit Unicode character.
- Default value: `'\u0000'`

12. Illustrate String class methods with an example

Sno	Method	Description
1	char charAt(int index)	It returns char value for the particular index
2	int length()	It returns string length
3	static String format(String format, Object... args)	It returns a formatted string.
4	String substring(int beginIndex)	It returns substring for given begin index.
5	String substring(int beginIndex, int endIndex)	It returns substring for given begin index and end index.
6	boolean contains(CharSequence s)	It returns true or false after matching the sequence of char value.
7	static String join(CharSequence delimiter, CharSequence... elements)	It returns a joined string.
8	static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	It returns a joined string.
9	boolean equals(Object another)	It checks the equality of string with the given object.
10	boolean isEmpty()	It checks if string is empty.
11	String concat(String str)	It concatenates the specified string.
12	String replace(char old, char new)	It replaces all occurrences of the specified char value.
13	String replace(CharSequence old, CharSequence new)	It replaces all occurrences of the specified CharSequence.
14	static String equalsIgnoreCase(String another)	It compares another string. It doesn't check case.
15	int indexOf(int ch)	It returns the specified char value index.
16	int indexOf(int ch, int fromIndex)	It returns the specified char value index starting with given index.

13. Explain various types of inheritance supported by JAVA with an example

1) Single Inheritance

Single inheritance is damn easy to understand. When a class extends another one class only then we call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a **parent class** of B and B would be a **child class** of A.

```
Class A
{
   public void methodA()
   {
     System.out.println("Base class method");
   }
}

Class B extends A
{
   public void methodB()
   {
     System.out.println("Child class method");
   }
   public static void main(String args[])
   {
     B obj = new B();
     obj.methodA(); //calling super class method
     obj.methodB(); //calling local method
  }
}
```

2) Multiple Inheritance

"**Multiple Inheritance**" refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with "multiple inheritance" is that the derived class will have to manage the dependency on two base classes.

Note: Multiple inheritance is not supported in Java through class.

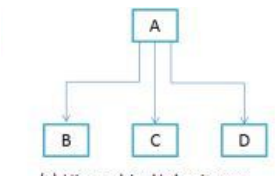
3) Multilevel Inheritance

Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A.

```
Class X
{
   public void methodX()
   {
     System.out.println("Class X method");
   }
}
Class Y extends X
{
public void methodY()
{
System.out.println("class Y method");
}
}
Class Z extends Y
{
   public void methodZ()
   {
     System.out.println("class Z method");
   }
   public static void main(String args[])
   {
     Z obj = new Z();
     obj.methodX(); //calling grand parent class method
     obj.methodY(); //calling parent class method
     obj.methodZ(); //calling local method
  }
}
```

4) Hierarchical Inheritance

In such kind of inheritance one class is inherited by many **sub classes**. In below example class B,C and D **inherits** the same class A. A is **parent class (or base class)** of B,C & D.



class A

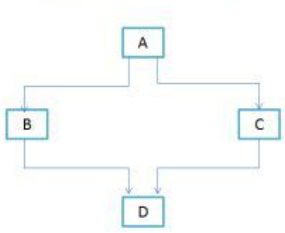
```
{
   public void methodA()
    {
       System.out.println("method of Class A");
    }
}
class B extends A
{
   public void methodB()
    {
       System.out.println("method of Class B");
    }
}
class C extends A
{
  public void methodC()
   {
      System.out.println("method of Class C");
   }
}
class D extends A
{
  public void methodD()
   {
      System.out.println("method of Class D");
   }
}
class JavaExample
{
  public static void main(String args[])
   {
      B obj1 = new B();
      C obj2 = new C();
      D obj3 = new D();
      //All classes can access the method of class A
      obj1.methodA();
      obj2.methodA();
      obj3.methodA();
   }
}
```

Output:

```
method of Class A
method of Class A
method of Class A
```

5) Hybrid Inheritance

In simple terms you can say that Hybrid inheritance is a combination of **Single** and **Multiple inheritance.** A typical flow diagram would look like below. A hybrid inheritance can be achieved in the java in a same way as multiple inheritance can be!! Using interfaces. yes you heard it right. By using **interfaces** you can have multiple as well as **hybrid inheritance** in Java




```
class C
{
   public void disp()
   {
       System.out.println("C");
   }
}

class A extends C
{
   public void disp()
   {
       System.out.println("A");
   }
}

class B extends C
{
   public void disp()
   {
       System.out.println("B");
   }

}

class D extends A
{
   public void disp()
   {
       System.out.println("D");
   }
   public static void main(String args[]){

       D obj = new D();
       obj.disp();
   }
}
```

Output:

```
D
```

14. Explain the use of 'super' keyword with an example

Uses of super keyword

1. To call methods of the superclass that is overridden in the subclass.
2. To access attributes (fields) of the superclass if both superclass and subclass have attributes with the same name.
3. To explicitly call superclass no-arg (default) or parameterized constructor from the subclass constructor.

Super Keyword

- 'super' is a keyword used to refer to hidden variables of super class from sub class.

  super.a=a;

- It is used to call a constructor of super class from constructor of sub class which should be first statement.

  super(a,b);

- It is used to call a super class method from sub class method to avoid redundancy of code

  super.addNumbers(a, b);

15. Explain the use of static keyword in java.

Java static keyword

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

Static Methods

Static methods are also called class methods. It is because a static method belongs to the class rather than the object of a class.

Static Variables

In Java, when we create objects of a class, then every object will have its own copy of all the variables of the class.

16. Explain what nested classes are. Explain different types of nested classes.

Nested Classes

- Nested class refers to a class that is inside of another class.
- Java allows us to create nested classes in java.
- A nested class is one of the members of its outer class.
- It can be declared as public, private, protected or default as well.
- The nested class has access to the other member of the outer class, while vice versa is not possible.
- It means the outer class doesn't have access to a nested class member as the nested class is a member of its enclosing outer class, so dot (.) used to access the nested class & its members.
- The nested class doesn't have any static keywords.

Nested Classes

- **Static Nested Class:** Nested class declared with the keyword static is known as static nested class. Static Nested classes are accessible by referencing the outer class. The nested class which has a static class cannot access the nonstatic variables & methods of the outer class.
- **Inner Class:** Inner class, which is not statically declared known as a non-static nested class. No static nested classes would have access to all the static & non-static variables & methods even if it declared private.

Nested Classes

- In Java, you can define a class within another class. Such class is known as nested class.
- Example:

```
class OuterClass
{
    // ... class NestedClass
        {
            // ...
        }
}
```

4 What is the role and responsibility of JVM in program execution?

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

The role of JVM has two momentous capacities:

- To permit Java programs to run on operating systems and any device.
- To manage and enhance program memory.

JVM performs the following function:

- Verifies code
- Executes code
- Provides runtime environment
- Loads code