Name: V. Vijaya lakshmi
Regd No: 19BQ1A0508
Section: CSE - D

1. List and explain java buzzwords. Which factors are making java famous language

Java is the most popular object-oriented programming language. Java has many advanced features, a list of key features is known as Java Buzz words. The Java team has listed the following terms as java buzz words.

Simple:

Java programming language is very simple and easy to learn, understand and code. Most of the syntaxes in java follow basic programming language "C" and object oriented programming concepts are similar to c++, In a java programming language many complicated features like pointers, operator overloading, structures, unions, etc have been removed. One of the most useful features is the garbage collector it makes java more simple.

Secure:

Java is said to be more secure programming language because it does not have pointers concept. Java provides a feature "applet" which can be embedded into a web

application. The applet in Java does not allow access to other parts of the computer, which keeps away from harmful programs like viruses and unauthorized access.

Portable:

Portability is one of the core features of Java which enables the Java programs to run on any computer or operating system. For example, an applet developed using Java runs on a wide variety of CPUs, operating System and browsers connected to the internet.

Object-oriented:

Java is said to be a pure object-oriented programming language. In Java, everything is an object. It supports all the features of the object-oriented programming paradigm. The primitive data types Java also implemented as objects using wrapper classes, but still, it allows primitive data types to archieve high-performance.

Robust:

Java is more robust because the Java code can be executed on a variety of environments, Java has a Strong memory management mechanism (garbage collector). Java is a strictly typed language, it has a strong set of exception handling mechanism and many more.

Architecture-neutral (or) platform Independent:
Java has invented to archieve "write once; run any where, any time, forever". The Java provides JVM (Java virtual machine) to archieve architectural-neutral or platform-independent. The JVM allows the java program created using one operating system can be executed on any other operating system.

Multi-threaded:
Java supports multi-threading programming, which allows us to write programs that do multiple operations simultaneously.

Interpreted:
Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode. The byte code is interpreted to any machine code so that it runs on the native machine.

High performance:
Java provides high performance with the help of features like JVM, interpretation, and its simplicity.

Distributed:
Java programming language supports TCP/IP protocols which enable the java to support the distributed environment of the Internet. Java also supports Remote

Method Invocation (RMI), this feature enables a program to invoke methods across a network.

Dynamic :

Java is said to be dynamic because the Java byte code may be dynamically updated on a running system and it has a dynamic memory allocation and deallocation (objects and garbage collector).

Which factors are making Java famous language.

Here is the list of reasons to tell Java is the best programming language in terms of opportunities, development and community support.

1) Java is Easy to learn

Java has fluent English like syntax with minimum magic characters e.g., Generics angle brackets, which makes it easy to read Java program and learn quickly. Once a programmer is familiar with initial hurdles with installing JDK and setting up PATH and understand How classpath works, it's pretty easy to write a program in Java.

2) Java has Rich API

One more reason for Java programming language's huge success is its Rich API, and most importantly, it's highly visible because it comes with Java installation.

Java provides API for I/O, networking, utilities, XML parsing, database connection and almost everything. Whatever left is covered by open source libraries like Apache Commons, Google Guava, Jackson, Gson, Apache POI and others.

3) Powerful development tools (Eclipse, Netbeans):
Coding in IDE is a pleasure, especially if you have coded in DOS editor or Notepad. They not only help in code completion but also provides a powerful debugging capability, which is essential for real-world development. Integrated Development Environment (IDE) made Java development much more comfortable, faster and fluent. Apart from IDE, Java platform also has several other tools like "Maven" and ANT for building Java applications, "Jenkins" for continuous Integration and delivery, decompilers, JConsole, visual VM for monitoring Heap Usage, etc.

4) Great collection of open source libraries
Apache, Google and other organization have contributed a lot of great libraries, which makes Java development easy, faster and cost-effective.
There are frameworks like Spring, struts, Maven, which ensures that Java development follows best practices of

software craftsmanship, promotes the use of design patterns and assisted Java developers in getting there job done.

5) Wonderful Community Support

A strong and thriving community is the biggest strength of Java programming language and platform. Java has lots of active forums, stack overflow, open-source organizations and several Java user groups to help everything. Java actually promotes taking and giving back to community habit.

6) Java is FREE

Since Java is free from the start, you don't need to pay anything to create Java application. This FREE thing also helped Java to become popular among individual programmers and among large organizations.

7) Excellent documentation support - Javadocs

It's a great piece of documentation, which tells a lot of things about Java API. Without Java documentation, Java wouldn't be as popular. Java doc made learning easy and provide an excellent reference while coding in Java.

8) Java is Everywhere

Java is everywhere, it's on the desktop, on mobile, on card, almost everywhere and so is Java programmers.

2. What are the benefits of inheritance? Explain various forms of inheritance with suitable code segments.

Inheritance:

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

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class). "extends" keyword is used to inherit the properties of a class.

Benefits of Inheritance:

1) one of the key benefits of inheritance is to minimize the amount of duplicate code in an application by sharing common code amongst several subclasses. Where equivalent code exists in two related classes, the hierachy can usually be refactored to move the common code up to a mutual superclass.

2) Inheritance can also make application code more flexible to change because classes that inherit from a common superclass can be used interchangeably.

3) Reusability: Facility to use public methods of base class without rewriting the same.

4) Extensibility: Extending the base class logic as per business logic of the derived class.

5) Data hiding: Base class can decide to keep some data private so that it cannot be altered by the derived class.

6) Overriding: With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class.

7) Inheritance makes the sub classes follow a standard interface.

8) Inheritance helps to reduce code redundancy and supports code extensibility.

9) Inheritance facilitates creation of class libraries.

Various forms of Inheritance:

Below are the various types of inheritance in Java.

1) Single Inheritance

Single Inheritance is very 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.

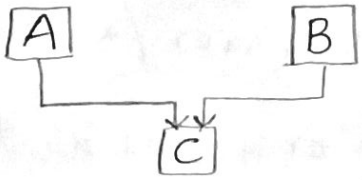
Single Inheritance.

Single Inheritance Example

```
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.



Multiple Inheritance

Multiple inheritance is very rarely used in software projects. Using multiple inheritance leads to problems and complexity when further extending class.

"Java" does not support Multiple Inheritance.

Multiple Inheritance is supported in C++.

Multiple Inheritance program

```
class Parent1
{
  void fun()
  {
   System.out.println("parent 1");
  }
}
class Parent2
{
  void fun()
  {
   System.out.println("parent 2");
  }
}
// Error: Test is inheriting from multiple classes.
class Test extends Parent1, Parent2
{
  Public static void main(String args[])
  {
    Test t = new Test();
      t.fun();
  }
}
```

Output: Compiler Error /* Complications as whether to call Parent1's fun() or Parent2's fun() method */.

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.

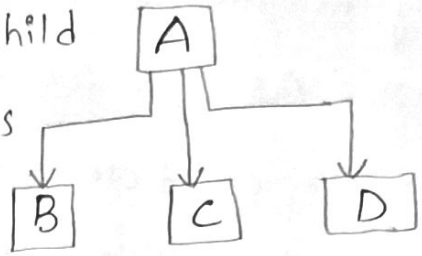
Multi level Inheritance.

Multilevel Inheritance example program

```
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 and D. When more than one child classes have the same parent class then this type of inheritance is known as hierarchical Inheritance.



Hierarchical Inheritance.

Example program

```
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()
 {
```

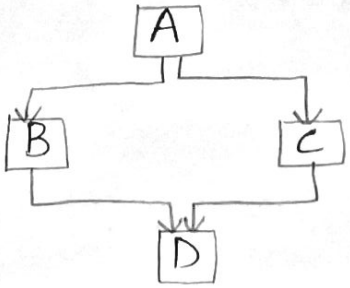
```
System.out.println("method of class D");
}
}
class Javaexample
{
 public static void main(string args[])
 {
   B obj1 = new B();
   C obj2 = new C();
   D obj3 = new D();
   // All classes can access the method of class A
      obj1.methodA();
      obj2.methodA();
      obj3.methodA();
 }
}
```

Output:

```
method of class A
method of class A
method of class A.
```

5) Hybrid Inheritance

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. By using interfaces you can have multiple as well as hybrid inheritance in java.



Hybrid Inheritance

Hybrid Inheritance program

```
class A
{
  Public void disp()
  {
    system.out.println(" A");
  }
}
class B extends A
{
  Public void disp()
  {
    system.out.println(" B");
  }
}
class C extends A
{
  Public void disp()
  {
    system.out.println(" C");
  }
}
class D extends B
{
  Public void disp()
  {
    system.out.println(" D");
  }
  public static void main(string args[])
  {
     D obj = new D();
      obj.disp();
  }
}
```

Output:

D

3. Define a class named movieMagic with the following description:

Instance variables / data members:

int year - to store the year of release of a movie.

String title - to store the title of the movie.

float rating - to store the popularity rating of the movie (minimum rating = 0.0 and maximum rating = 5.0)

Member methods:

i) movieMagic() Default Constructor to initialize numeric data members to 0 and String data member to " ";

ii) void accept() to input and store year, title and rating

iii) Void display() to display the title of a movie and a message based on the rating as per the table below.

Rating	Message to be displayed
0.0 to 2.0	Flop
2.1 to 3.4	Semi-hit
3.5 to 4.5	Hit
4.6 to 5.0	Super Hit

Write a main method to create an object of the class and call the above member methods.

```
Public class movieMagic {
        privateint year;
        private String title;
        private float rating;
        public movieMagic() {
            this.year = 0;
```

```
    this. title = " ";
    this. rating = 0;
}
Public void accept (int year, String title, float rating)
  {
    this. year = year;
    this. title = title;
    this. rating = rating;
  }
Public void display()
 {
 if (rating >=0.0 && rating=<2.0)
        System. out. println (" Flop");
 elseif (rating >= 2.1 && rating <= 3.4)
        System. out println (" Semi-Hit");
 else if (rating >= 3.5 && rating <= 4.5)
        System. out. println (" Hit");
 else if (rating >= 4.6 && rating <= 5.0)
        System. out. println (" Super Hit");

  else
    system. out. println (" please enter a valid number");
}
Public static void main (String args[]) {
   movieMagic obj = new movieMagic();
     Obj. accept (2000, "ABCD", 4.7f);
     Obj. display();
  }
}
```

Output: Super Hit.

4. Write a class to overload a function num_calc() as follows:

i) void num_calc(int num, char ch) with one integer argument and one character argument, computes the square of integer argument if choice ch is 's' otherwise finds its cube.

ii) void main num_calc(int a, int b, char ch) with two integer arguments and one character argument. It computes the product of integer arguments if ch is 'p' else add the integers.

iii) void num_calc(string s1, string s2) with two string arguments, which prints whether the strings are equal or not.

```
class OverloadDemo{
    void num_calc(int num, char ch){
        if(ch == 's')
            System.out.println(num * num);
        else
            System.out.println(num*num*num);
    }
    Void num_calc(int a, int b, char ch){
        if (ch == 'p')
            system.out.println(a*b);
        else
            system.out.println(a+b);
    }
```

```
Void num-calc (String s1, String s2) {
    if (s1 == s2)
        System.out.println ("strings are equal");
    else
        System.out.println ("strings are not equal");
    }
}
class Overload {
    public static void main (String args[]) {
        OverloadDemo obj = new OverloadDemo ();
        obj.num-calc (2, 'c');
        obj.num-calc (7, 8, 'p');
        obj.num-calc ("abcd", "abcd");
    }
}
```

Output:

```
8
56
strings are equal.
```