JAVA Assignment

Sunday, April 16, 2023 9:03 AM ~ LalitDumka

PART 1

Q1a. How is portability achieved in java ? (focus on bytecode and JVM) ***

Java achieves portability through its bytecode and the Java Virtual Machine (JVM). When a Java program is compiled, it is converted into bytecode, which is a platform-independent representation of the code. This bytecode is then executed by the JVM, which is available on different platforms and operating systems.
The JVM is responsible for interpreting the bytecode and executing the instructions on the local machine. It provides an abstract layer between the Java program and the underlying hardware, which allows the Java program to run on different machines without modification.
Since the JVM is available on different platforms and operating systems, Java bytecode can be executed on any machine that has a JVM installed. This makes Java programs highly portable and eliminates the need for developers to write platform-specific code.
In summary, Java achieves portability through its bytecode and the JVM. The bytecode is platform-independent and can be executed on any machine that has a JVM installed, making Java programs highly portable.

Q1b. Write short notes on (with diagrams): ***

JDK, JRE, JVM, Just in time Compiler, Bytecode

1. JDK (Java Development Kit): JDK is a software development kit used to develop Java applications. It includes a set of tools, libraries, and documentation to develop, compile, and debug Java applications. JDK also includes the JRE (Java Runtime Environment) and the Java compiler.

2. JRE (Java Runtime Environment): JRE is a software environment used to run Java applications. It includes the JVM and the necessary class libraries. JRE does not include the Java compiler or the development tools.

3. JVM (Java Virtual Machine): JVM is an abstract machine that provides a runtime environment for Java bytecode to execute. It interprets the bytecode and executes the instructions on the local machine. JVM is available on different platforms and operating systems, making Java programs highly portable.

4. Just-in-Time Compiler (JIT): JIT is a component of the JVM that improves the performance of Java applications. JIT compiles the bytecode into machine code at runtime, which is then executed by the local machine. This compilation process eliminates the need for interpreting the bytecode repeatedly, resulting in faster execution.

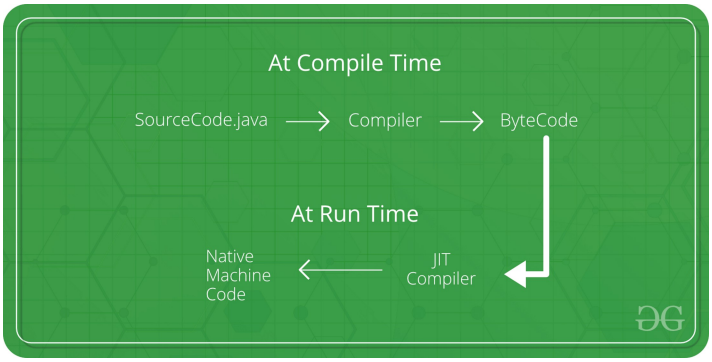
   #### Advantages of JIT Compiler

   - It requires less memory usages.
   - The code optimization is done at run time.
   - It uses different levels of optimization.
   - It reduces the page faults.

   #### Disadvantages of JIT Compiler

   - It increases the complexity of the program.
   - The program with less line of code does not take the benefit of the JIT compilation.
   - It uses lots of cache memory.

   a.



- Bytecode: Bytecode is a platform-independent representation of Java code. When a Java program is compiled, it is converted into bytecode, which can be executed on any machine that has a JVM installed. Bytecode is a highly optimized code that is easy to interpret, making Java programs faster and more efficient.

  In summary, JDK, JRE, JVM, Just-in-Time Compiler, and Bytecode are essential components of the Java platform that work together to make Java

programs portable, efficient, and easy to develop.

Q2a. Write all built in data types in java.

The eight primitive data types supported by the Java programming language are:

- **byte**: The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive).
- **short**: The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive).
- **int**: By default, the int data type is a 32-bit signed two's complement integer, which has a minimum value of -2^{31} and a maximum value of $2^{31}-1$. In Java SE 8 and later, you can use the int data type to represent an unsigned 32-bit integer, which has a minimum value of 0 and a maximum value of $2^{32}-1$.
- **long**: The long data type is a 64-bit two's complement integer. The signed long has a minimum value of -2^{63} and a maximum value of $2^{63}-1$. In Java SE 8 and later, you can use the long data type to represent an unsigned 64-bit long, which has a minimum value of 0 and a maximum value of $2^{64}-1$.
- **float**: The float data type is a single-precision 32-bit IEEE 754 floating point.
- **double**: The double data type is a double-precision 64-bit IEEE 754 floating point.
- **boolean**: The boolean data type has only two possible values: true and false. Use this data type for simple flags that track true/false conditions.
- **char**: The char data type is a single 16-bit Unicode character. It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).

In addition to the eight primitive data types listed above, the Java programming language also provides special support for character strings via the java.lang.String class. Enclosing your character string within double quotes will automatically create a new String object; for example, String s = "this is a string";. String objects are *immutable*

From <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

Q2b. Derive the range of int (32 bits).

The int data type in Java is a 32-bit signed two's complement integer.
In binary representation, 32 bits can represent a maximum of 2^32 different values. However, since the int data type is signed, one bit is used to represent the sign of the number. Therefore, the range of int is divided between positive and negative values, with zero as a neutral point.

The range of int in Java is from -2,147,483,648 to 2,147,483,647 (inclusive). This can be derived as follows:

- The maximum positive value of an int is (2^31) - 1, which is 2,147,483,647 in decimal representation.
- The minimum negative value of an int is -2^31, which is -2,147,483,648 in decimal representation.
- Zero occupies one value in the range.

Therefore, the range of int in Java is -2,147,483,648 to 2,147,483,647.

Q3. Write features of java. Write differences between C++ and java.

Features of Java:

a. Simple and Easy to Learn: Java was designed to be easy to learn and use, with a simple syntax and a wide range of libraries and tools.
b. Platform Independent: Java code is compiled into bytecode, which can be run on any platform that has a Java Virtual Machine (JVM) installed.
c. Object-Oriented: Java is an object-oriented programming language, which means it is based on the concept of objects and classes.
d. Robust: Java is designed to be robust and reliable, with features like automatic garbage collection and exception handling.
e. Secure: Java runs inside virtual machine and has built-in security features to protect against common vulnerabilities like buffer overflows and memory leaks.
f. Multithreaded: Java supports multithreading, allowing multiple threads of execution to run concurrently within a single program.
g. High Performance: Java has a Just-In-Time (JIT) compiler that can optimize code at runtime for high performance.
h. Rich API: Java provides a rich set of APIs for everything from GUI programming to networking to database access.

i. Distributed: because it facilitates users to create distributed applications in java. RMI and EJP are used to create distributed applications.

Differences between C++ and Java:

a. Memory Management: C++ provides manual memory management using pointers, while Java has automatic memory management through garbage collection.
b. Platform Independence: Java is platform-independent because of its bytecode, whereas C++ code is compiled into machine-specific code, making it platform-dependent.
c. Object-Oriented Programming: Both languages support object-oriented programming, but Java is purely object-oriented, while C++ is a hybrid language that supports both object-oriented and procedural programming.
d. Compilation: C++ code is compiled directly into machine code, while Java code is compiled into bytecode that is executed by the JVM.
e. Performance: C++ is generally faster than Java because it allows for direct memory access and has less overhead.
f. Library Support: Java has a large standard library and a wide range of third-party libraries, while C++ has a smaller standard library and fewer third-party libraries.
g. Error Handling: Java has built-in exception handling, while C++ relies on error codes and error handling functions.

Q4. Explain the working of Garbage Collector **

In Java, Garbage Collector is responsible for automatically freeing up memory that is no longer in use by the program.

The Garbage Collector works as follows:

a. Marking: The Garbage Collector starts by marking all the objects that are still reachable by the program. It does this by starting from a set of root objects (such as local variables, static variables, and the main object), and then following all the object references to identify all the objects that are still in use.
b. Tracing: After marking the reachable objects, the Garbage Collector traces all the object references in memory, marking any additional objects that are still reachable.
c. Finalization: Once all the reachable objects have been marked, the Garbage Collector performs finalization, which involves calling the finalize() method on any objects that have one. This method can be used to perform any necessary cleanup operations before the object is garbage collected.
d. Sweeping: After finalization, the Garbage Collector sweeps through the memory, identifying any objects that were not marked as reachable and releasing the memory occupied by those objects.

The above process is repeated periodically in the background as the program runs, ensuring that memory is freed up as soon as it is no longer needed. The Garbage Collector uses various algorithms to optimize the performance of the process

Q5. What is constructor and finalize? What is constructor overloading?

a. Constructor: A constructor is a special method in Java that is used to initialize objects. It is called when an object of a class is created using the "new" keyword. Constructors have the same name as the class and do not have a return type. They can be used to initialize the values of the object's member variables, allocate memory, and perform other initialization tasks.

b. Finalize: Finalize is a method in Java that is called by the Garbage Collector before an object is garbage collected. It can be used to perform any necessary cleanup operations before the object is destroyed. However, it is not recommended to rely on the finalize() method for critical cleanup operations, as it is not guaranteed to be called in a timely manner or at all.

c. Constructor Overloading: Constructor overloading is a feature in Java that allows a class to have multiple constructors with different parameter lists. This allows objects to be created with different initialization values, depending on the needs of the program. Constructor overloading is similar to method overloading, in that it allows a class to have multiple methods with the same name but different parameter lists.

For example, consider the following class:

```
public class Person {
   private String name;
   private int age;

   public Person() {
      name = "";
      age = 0;
   }
```

```
    public Person(String name) {
        this.name = name;
        age = 0;
    }

    public Person(int age) {
        name = "";
        this.age = age;
    }

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

Q6. What is function overloading?

Function overloading is a feature in many programming languages, including Java, that allows a class to have multiple methods with the same name but different parameter lists. This means that a single function name can be used to perform different tasks depending on the types and number of parameters passed to it.

```
public class MathOperations {
    public int add(int x, int y) {
        return x + y;
    }

    public double add(double x, double y) {
        return x + y;
    }

    public int add(int x, int y, int z) {
        return x + y + z;
    }
}
```

Q7. Why are Strings immutable? *** How to create mutable strings ?

String is immutable for several reasons:

- **Security**: parameters are typically represented as String in network connections, database connection urls, usernames/passwords etc. If it were mutable, these parameters could be easily changed.
- **Synchronization** and **concurrency**: making String immutable automatically makes them thread safe thereby solving the synchronization issues.
- **Caching**: when compiler optimizes your String objects, it sees that if two objects have same value (a="test", and b="test") and thus you need only one string object (for both a and b, these two will point to the same object).
- **Class loading**: String is used as arguments for class loading. If mutable, it could result in wrong class being loaded (because mutable objects change their state).

To create mutable strings in Java, the StringBuilder or StringBuffer class can be used. These classes provide methods for appending and modifying strings.

```
StringBuilder sb = new StringBuilder("Hello");
sb.append(" world!");
String mutableString = sb.toString();
```

Q8. Differentiate between String, StringBuilder, StingBuffer classes. *

	String	StringBuilder	StringBuffer
Mutability	Immutable	Mutable	Mutable
Thread-safety	Thread-safe	Not thread-safe	Thread-safe
Synchronization	No synchronization	No synchronization	Synchronized
Performance	Fast for read operations, slower for write operations	Fast for write operations, slower for read operations	Slower due to synchronization

Usage	For situations where immutability is required	For situations where a large number of string manipulation operations are required	For multi-threaded environments where thread-safety is a concern

Q9. Create a student class with members name, roll, eng_marks, hin_marks. Create these functions: ***

Parameterized input ()

Non parameterized input()

Paramaterized constructor()

Non parameterized constructor()

Display()

```
public class Student {
    private String name;
    private int roll;
    private int engMarks;
    private int hinMarks;

    // Parameterized constructor
    public Student(String name, int roll, int engMarks, int hinMarks) {
        this.name = name;
        this.roll = roll;
        this.engMarks = engMarks;
        this.hinMarks = hinMarks;
    }

    // Non-parameterized constructor
    public Student() {
        this.name = "";
        this.roll = 0;
        this.engMarks = 0;
        this.hinMarks = 0;
    }

    // Parameterized input method
    public void parameterizedInput(String name, int roll, int engMarks, int hinMarks) {
        this.name = name;
        this.roll = roll;
        this.engMarks = engMarks;
        this.hinMarks = hinMarks;
    }

    // Non-parameterized input method
    public void nonParameterizedInput() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter name: ");
        this.name = scanner.nextLine();
        System.out.print("Enter roll number: ");
        this.roll = scanner.nextInt();
        System.out.print("Enter English marks: ");
        this.engMarks = scanner.nextInt();
        System.out.print("Enter Hindi marks: ");
        this.hinMarks = scanner.nextInt();
    }

    // Display method
    public void display() {
        System.out.println("Name: " + this.name);
        System.out.println("Roll: " + this.roll);
        System.out.println("English Marks: " + this.engMarks);
        System.out.println("Hindi Marks: " + this.hinMarks);
    }
```

}

PART -2

Scanner, arrays, type wrappers, cmd line arguments, packages

1. Write the syntax for creating an obj of scanner class and reading a string from input.
2. Write the example syntax for (do not write the complete code)
   a. printing the first and last character of string using charAt()

```
String str = "Hello";
char firstChar = str.charAt(0); // firstChar will have value 'H'
char lastChar = str.charAt(str.length() - 1); // lastChar will have value 'o'
System.out.println("First Character: " + firstChar);
System.out.println("Last Character: " + lastChar);
```

   b. finding length of a string

```
String str = "Hello";
int length = str.length(); // length will have value 5
System.out.println("Length: " + length);
```

   c. checking whether two strings are equal using equals( )

```
String str1 = "Hello";
String str2 = "hello";
boolean isEqual = str1.equals(str2); // isEqual will have value false
System.out.println("Strings are equal: " + isEqual);
```

   d. compare two string using compareTo( )

```
String str1 = "Apple";
String str2 = "Banana";
int result = str1.compareTo(str2); // result will be negative, as "Apple" comes before
"Banana" in dictionary order
System.out.println("Result: " + result);
```

   e. converting a string to upper case and lower case using toUpperCase( ) and toLowerCase( )

```
String str = "Hello";
String upperCaseStr = str.toUpperCase(); // upperCaseStr will have value "HELLO"
String lowerCaseStr = str.toLowerCase(); // lowerCaseStr will have value "hello"
System.out.println("Upper case string: " + upperCaseStr);
System.out.println("Lower case string: " + lowerCaseStr);
```

   f. converting a integer to string

```
int num = 123;
String str = Integer.toString(num); // str will have value "123"
System.out.println("String: " + str);
```

   g. converting a string to integer

```
String str = "123";
int num = Integer.parseInt(str); // num will have value 123
System.out.println("Integer: " + num);
```

   h. replace all characters in a string using replace( )

```
String str = "Hello";
String replacedStr = str.replace('l', 'x'); // replacedStr will have value "Hexxo"
System.out.println("Replaced string: " + replacedStr);
```

i. check whether a string is present in a given string using contains( )

```
String str = "Hello, World!";
boolean isPresent = str.contains("World"); // isPresent will have value true
System.out.println("String is present: " + isPresent);
```

j. find the index of a substring in a given string using indexOf( )

```
String str = "Hello, World!";
int index = str.indexOf("World"); // index will have value 7
System.out.println("Index: " + index);
```

3. Write the syntax to create a string of StringBuilder class and to replace the character at a particular index.

```
StringBuilder strBuilder = new StringBuilder("Hello");
strBuilder.setCharAt(1, 'i'); // replaces the character 'e' at index 1 with 'i'
String newStr = strBuilder.toString(); // converts StringBuilder object to String
System.out.println(newStr); // prints "Hillo"
```

4. Which of the following is true about arrays :

Arrays are dynamic	False
Memory is allocated using new	True
Array are reference variables	True
Arrays are immutable	False
Bounds checking is performed in arrays	True
Size of arr is determined using as arr.length( )	True

5. What are jagged arrays? Give example.

A jagged array is a two-dimensional array in Java where each row can have a different length. In other words, a jagged array is an array of arrays, where each sub-array can have a different length.
Here's an example of a jagged array:

```
int[][] jaggedArr = new int[3][];
jaggedArr[0] = new int[] {1, 2, 3};
jaggedArr[1] = new int[] {4, 5};
jaggedArr[2] = new int[] {6, 7, 8, 9};
```

Jagged arrays are useful when you need to represent data in a table-like structure, but the number of columns is not fixed for each row.

6. What are anonymous arrays, and objects ? give example

Anonymous arrays and objects are Java entities that do not have a name and are created and used on the fly. They are useful in cases where we need to perform some operation using an array or object, but do not need to keep a reference to the array or object beyond the operation.

```
int sum = 0;
sum = sumArr(new int[]{1, 2, 3, 4, 5});

public int sumArr(int[] arr) {
   int sum = 0;
   for(int i = 0; i < arr.length; i++) {
      sum += arr[i];
```

```
    }
    return sum;
}


int sum = 0;
sum = new Calculator().add(10, 20);

class Calculator {
    public int add(int num1, int num2) {
        return num1 + num2;
    }
}
```

In the above example, we create an anonymous object of the Calculator class and call its add method to compute the sum of two numbers. The anonymous object is created on the fly and is used only for the duration of the method call.

7. What are type wrappers / wrapper classes? What is boxing, unboxing, auto boxing, auto unboxing? Give example *

Wrapper classes in Java are a way to represent primitive data types as objects. The wrapper classes provide a mechanism to convert primitive data types into objects and vice versa. The following are the wrapper classes for the primitive data types in Java:

- Byte
- Short
- Integer
- Long
- Float
- Double
- Boolean
- Character

Boxing and unboxing are the processes of converting primitive data types to their corresponding wrapper classes (boxing) and converting wrapper classes to their corresponding primitive data types (unboxing). Auto boxing and auto unboxing are automatic conversions between primitive data types and their corresponding wrapper classes. Auto boxing and unboxing are performed automatically by the Java compiler.

```
// boxing
int num = 10;
Integer integer = Integer.valueOf(num);

// unboxing
Integer integerObj = new Integer(10);
int num1 = integerObj.intValue();

// auto boxing
int num2 = 20;
Integer integerObj2 = num2;

// auto unboxing
Integer integerObj3 = new Integer(30);
int num3 = integerObj3;
```

8. What is the use of wrapper classes? or Why do we need to convert primitive data type to objects?

Wrapper classes in Java provide a way to represent primitive data types as objects. There are several reasons why we might need to convert primitive data types to objects:

1. To use them in collections: Collections in Java can only store objects, so if we need to store primitive data types in a collection, we must first convert them to their

corresponding wrapper classes.

2. To pass them to methods that accept objects: Some methods in Java only accept objects as arguments, so we must first convert the primitive data types to their corresponding wrapper classes before passing them to the method.
3. To use them in object-oriented programming: Object-oriented programming is based on the concept of objects. If we want to treat a primitive data type as an object, we can convert it to its corresponding wrapper class.
4. To use them in reflection: Reflection is a feature in Java that allows us to inspect and manipulate the runtime behavior of an application. Reflection works with objects, so if we need to reflect on a primitive data type, we must first convert it to its corresponding wrapper class.

Overall, wrapper classes provide a way to bridge the gap between primitive data types and objects, and allow us to use primitive data types in situations where objects are required.

9. What are command line arguments? Why are they used? Give example.
10. What are packages in java? Why are they used?

In Java, a package is a way to organize related classes and interfaces into a single namespace. A package can contain both classes and sub-packages, and provides a mechanism for access control and naming conflicts.

Packages are used for a number of reasons:

1. Organization: Packages allow related classes and interfaces to be grouped together in a logical manner, making it easier to find and understand the code.
2. Access control: Packages can be used to control the visibility of classes and interfaces, allowing them to be made public or private as needed.
3. Naming conflicts: Packages provide a way to avoid naming conflicts between classes and interfaces that have the same name but are defined in different parts of the application.
4. Modularity: Packages can be used to create modular applications, where different parts of the application are isolated from each other and can be developed and maintained independently.
5. Reusability: Packages can be reused in other applications, providing a way to share code between different projects.

Overall, packages are an important feature of Java that provide a way to organize, control, and modularize code, making it easier to develop and maintain large-scale applications.

11. What are the steps to create a package?

PART 3

Inheritance, super, this, **,final, overriding, interface , abstract class, access specifiers**

1. **What is inheritance? Types of inheritance ? Purpose/ benefits of inheritance? *****

Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a class to inherit properties and behaviors from a parent class. Inheritance enables us to create new classes that are built upon existing classes, thus reusing code and reducing redundancy.

There are several types of inheritance in Java:

i. Single inheritance: A class can inherit from a single parent class.
ii. Multiple inheritance: A class can inherit from multiple parent classes (not supported in Java).
iii. Multilevel inheritance: A class can inherit from a parent class, which itself inherits from another parent class.
iv. Hierarchical inheritance: Multiple classes can inherit from a single parent class.
v. Hybrid inheritance: A combination of multiple and multilevel inheritance.

2. Why is multiple inheritance not allowed in java? *** How is it achieved?

Multiple inheritance is a feature in object-oriented programming where a class can inherit properties and behavior from multiple parent classes. However, Java does not allow multiple inheritance because it can create a problem called the "diamond problem".
The diamond problem occurs when two parent classes have a method with the same name and signature, and a child class inherits from both parent classes. In this case, it is not clear which method should be called by the child class, and it can lead to ambiguity and code complexity.
Instead of allowing multiple inheritance, Java provides interfaces which are similar to classes but contain only abstract methods and constants. A class can implement multiple interfaces, but it can only inherit from one parent class. This allows for code reuse and flexibility while avoiding the problems caused by multiple inheritance.

3. What is constructor chaining? Why is it done? Give example. *

Constructor chaining is the process of calling one constructor from another constructor within the same class or from a subclass. This allows for code reuse and helps to avoid duplicate code.

Constructor chaining is done by using the this() and super() keywords to call another constructor with the appropriate parameters. The this() or super() call must be the first statement in the constructor body.

```
public class Person {
   private String name;
   private int age;

   // Constructor with two parameters
   public Person(String name, int age) {
      this.name = name;
      this.age = age;
   }

   // Constructor with one parameter that calls the two-parameter constructor using this()
   public Person(String name) {
      this(name, 0);
   }
}

public class Student extends Person {
   private int rollNumber;

   // Constructor with three parameters that calls the two-parameter constructor in the
parent class using super()
   public Student(String name, int age, int rollNumber) {
      super(name, age);
      this.rollNumber = rollNumber;
   }
}
```

4. **What is super ( ) method? What is super keyword? *****
5. **What is this ( ) method? What is this keyword ?**
6. **Differentiate between*****
   a. **Super( ) and super**
   b. **This ( ) and this**
   c. **Super ( ) and this( )**
   d. **Super and this (super keyword and this keyword)**

a. Super( ) and super:
super() is a constructor call that invokes the parent class constructor with no arguments. It is used to call the parent class constructor explicitly from the child class constructor. super is a reference variable that refers to the parent class object.
b. This ( ) and this:
this() is a constructor call that invokes the same class constructor with no arguments. It is used to call the constructor from another constructor in the same class. this is a reference variable that refers to the current class object.
c. Super ( ) and this( ):
super() is used to call the parent class constructor explicitly from the child class constructor. this() is used to call the same class constructor from another constructor in the same class.

7. **What is function overriding? What is the importance of function overriding ? *****

Function overriding is a feature of object-oriented programming (OOP) that allows a subclass or derived class to provide a specific implementation of a method that is already defined and implemented in its parent class or superclass. In function overriding, a method in a subclass has the same name, return type, and parameters as a method in its superclass, but with a different implementation.

Importance of function overriding:

- Allows the derived class to provide its own implementation of a method that is already defined in the base class.
- Helps achieve runtime polymorphism, where the object of the derived class is referred to by the reference of the base class.

```
class Animal {
  public void move() {
    System.out.println("Animal can move");
  }
}

class Dog extends Animal {
  public void move() {
    System.out.println("Dog can run and walk");
  }
}

public class Test {
  public static void main(String args[]) {
    Animal animal = new Animal();
    Animal dog = new Dog();

    animal.move();
    dog.move();
  }
}
```

8. Differentiate between overriding and overloading.

a. Definition: Overloading is when you have multiple methods with the same name in the same class, but with different parameter lists. Overriding is when you have a method in a subclass with the same name, return type, and parameter list as a method in its superclass.
b. Scope: Overloading happens within the same class, while overriding happens between a superclass and a subclass.
c. Inheritance: Overloading does not involve inheritance, whereas overriding requires inheritance.
d. Runtime behavior: Overloading is determined at compile-time, based on the parameter types. Overriding is determined at runtime, based on the actual object type.
e. Purpose: Overloading is used to provide different versions of a method to handle different parameter types. Overriding is used to provide a specific implementation of a method in a subclass that is different from its superclass.

Overloading:

```
public class Example {

  public void print(int num) {
    System.out.println("The number is: " + num);
  }

  public void print(String text) {
    System.out.println("The text is: " + text);
  }

}
```

overriding

```
public class Animal {

  public void makeSound() {
    System.out.println("The animal makes a sound");
  }

}

public class Cat extends Animal {

  @Override
  public void makeSound() {
    System.out.println("The cat meows");
  }

}
```

9. Write about final keyword.

Here are some common uses of the final keyword:

1. Final variables: If a variable is declared as final, its value cannot be changed after initialization. The final variables are usually declared using the keyword final, and they are written in uppercase letters.
2. Final methods: If a method is declared as final, then it cannot be overridden in the subclass. The final methods are useful when a method's functionality is critical and should not be altered by the subclasses.
3. Final classes: If a class is declared as final, then it cannot be subclassed. Final classes are usually used to create utility classes that provide a set of methods that can be used across the application.

```
public class Example {
  private final int id; // final variable

  public Example(int id) {
    this.id = id;
  }
```

```
  public final void printId() { // final method
    System.out.println("Id: " + id);
  }
}

public final class UtilityClass { // final class
  public static final double PI = 3.141592653589793;

  public static int add(int a, int b) {
    return a + b;
  }
}
```

10. Write about : ***

a. Static variable

A static variable or class variable is a variable that belongs to the class and not to any instance of the class. It is declared using the static keyword and is shared among all instances of the class. Static variables are initialized only once, at the start of the execution, and they retain their value until the end of the program.

```
public class MyClass {
  static int count = 0;
  public MyClass() {
    count++;
  }
}
```

b. Static function

A static function or class method is a function that belongs to the class and not to any instance of the class. It is declared using the static keyword and can be called directly on the class itself, without the need for an instance of the class. Static functions can only access other static members of the class and cannot access non-static members.

```
public class MyClass {
  static int multiply(int x, int y) {
    return x * y;
  }
}
```

c. Static block

A static block is a block of code that is executed only once, when the class is loaded into the memory. It is declared using the static keyword and is executed before the constructor and main method of the class. Static blocks are used to initialize static variables or to perform some other one-time initialization tasks.

```
public class MyClass {
  static int num;

  static {
    num = 42;
  }
}
```

d. Static class

we can use the static keyword to make an inner class static. A static inner class is a nested class that belongs to the outer class, but can be instantiated without instantiating the outer class. It cannot access non-static members of the outer class.

```
public class MyClass {
  public static class InnerClass {
    // code for inner class
  }
}
```

e. Static import

A static import is a feature that allows us to import static members of a class directly into our code, without having to qualify them with the class name every time. This feature can be used to reduce the verbosity of our code and make it more readable. Static imports are declared using the static keyword and the import statement.

```
import static java.lang.Math.PI; // import static member PI of Math class

public class Circle {
   public static void main(String[] args) {
      double radius = 10.0;
      double circumference = 2 * PI * radius; // no need to qualify PI with Math class
      System.out.println("Circumference = " + circumference);
   }
}
```

11. What is abstract class and abstract function? How is abstract class created? Features of abstract class. Why is it used?***

In Java, an abstract class is a class that cannot be instantiated. It is marked with the abstract keyword. An abstract class can have both abstract and non-abstract methods. An abstract method is a method without a body or implementation, and it is marked with the abstract keyword.

```
abstract class Shape {
  abstract void draw();
  void display() {
    System.out.println("This is a shape");
  }
}
```

Features of an abstract class:

- It cannot be instantiated
- It can contain both abstract and non-abstract methods
- It can have constructors and non-static blocks
- It can have final, non-final, static, and non-static variables
- It can be extended by a non-abstract class
- A non-abstract class that extends an abstract class must implement all the abstract methods of the abstract class

An abstract class is used when you want to provide a base implementation for the derived classes. It is used to provide a template or a blueprint for the derived classes to follow. The derived classes can provide their own implementation for the abstract methods defined in the abstract class.

12. What is an interface? Why is it used? **

An interface in Java is a blueprint of a class that consists of a collection of abstract methods (methods with no body) and constants. It is similar to a class in that it defines a set of behaviors or functionalities, but it does not contain any implementation details. Instead, it only defines the methods and constants that must be implemented by the classes that implement the interface.

Interfaces are used in Java to achieve abstraction and provide a way to define a contract or set of rules that must be followed by the classes that implement them. This allows for more flexibility in designing systems, as different classes can implement the same interface, allowing them to be used interchangeably in the code. Interfaces also help to enforce consistency and standardization in code development, as they define a common set of behaviors that must be supported by all implementing classes.

```
public interface MyInterface {
  public void doSomething();
  public int getSomething();
  public static final int MAX_VALUE = 100;
}
```

13. Differentiate between abstract class and interface. ***

a. Implementation: An abstract class can have both abstract and non-abstract methods with an implementation, whereas an interface can only have abstract methods that don't have an implementation.
b. Multiple inheritance: A class can only extend one abstract class, but can implement multiple interfaces.
c. Accessibility: Abstract class can have different access modifiers (public, protected, private) for its members, while all members of an interface are public by default.
d. Constructors: Abstract classes can have constructors, while interfaces cannot.
e. Variables: Abstract classes can have instance variables, while interfaces cannot have any instance variables. They can only have constants (static final variables).

14. Demonstrate how multiple inheritance can be achieved using interface **

```
interface A {
  void methodA();
}

interface B {
  void methodB();
}

class C implements A, B {
  public void methodA() {
    System.out.println("Method A");
  }

  public void methodB() {
    System.out.println("Method B");
  }
}

public class Main {
  public static void main(String[] args) {
    C obj = new C();
    obj.methodA();
    obj.methodB();
  }
}
```

15. What is the role of these access specifiers?**

a. Public
b. Private
c. Protected
d. Default

a. Public: When a class, method or variable is declared as public, it can be accessed from any part of the program. This means that it has the widest level of accessibility.

b. Private: When a class, method or variable is declared as private, it can only be accessed within the same class. This means that it has the narrowest level of accessibility.

c. Protected: When a class, method or variable is declared as protected, it can

be accessed within the same class, within any class in the same package, and within any subclass in any package. This means that it has a wider level of accessibility than private, but narrower than public.

d. Default: When a class, method or variable is not declared with any access specifier (i.e. no access specifier is used), it is considered to have default access. This means that it can be accessed within the same package only.

Access specifiers help to ensure that the right level of access is granted to classes, methods and variables. This can help to improve the security and maintainability of code, as well as prevent accidental or unintended access to certain parts of the program.