-----ANSWERS--------

1. **Differentiate ArrayList and LinkedList**:

   - ArrayList is implemented as a dynamically resizable array, which means it's like a dynamic array that can grow and shrink as needed. LinkedList is implemented as a doubly-linked list, which means each element points to both the next and the previous elements.

   - Access time for ArrayList is faster than LinkedList, as ArrayList uses an index-based system for accessing elements. In LinkedList, you have to traverse the list from the beginning to reach a specific element.

   - Insertion and deletion operations are faster in LinkedList compared to ArrayList, especially when it comes to frequent insertions and deletions at the middle of the list.

----

2. **Differentiate abstract classes and interfaces**:

   - Abstract classes can have both abstract and concrete methods, whereas interfaces can only have abstract methods.

   - A class can extend only one abstract class, but it can implement multiple interfaces.

   - Abstract classes can have instance variables, constructors, and non-abstract methods. Interfaces can only have constant (static final) variables.

----

3. **Explain decision making statements**:

   Decision-making statements allow the program to make decisions and execute different blocks of code based on certain conditions. In Java, this is typically done using if-else, switch, and ternary operator.

   For example, an if-else statement checks a condition and executes a block of code if the condition is true, otherwise it executes a different block of code.

~~~
   ```java
   if (condition) {
       // Code to be executed if condition is true
   } else {
       // Code to be executed if condition is false
   }
~~~

----

4. **Program to calculate percentage**:

```java
import java.util.Scanner;

public class PercentageCalculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter marks in subject 1: ");
        double subject1 = scanner.nextDouble();

        System.out.print("Enter marks in subject 2: ");
        double subject2 = scanner.nextDouble();

        System.out.print("Enter marks in subject 3: ");
        double subject3 = scanner.nextDouble();

        double totalMarks = subject1 + subject2 + subject3;
        double percentage = (totalMarks / 300) * 100;

        System.out.println("Percentage: " + percentage + "%");
    }
}
```

This program takes marks in three subjects as input and calculates the percentage.

5. **Hierarchy of the Collection framework**:

   - Collection (Interface)
     - List (Interface)
       - ArrayList
       - LinkedList
     - Set (Interface)
       - HashSet
       - TreeSet
     - Queue (Interface)
       - PriorityQueue
     - Map (Interface)
       - HashMap
       - TreeMap

The Collection framework in Java provides a unified architecture for manipulating and storing groups of objects.

----

6. **Program to perform arithmetic operations using method overloading**:

```java
public class ArithmeticOperations {
    public int add(int a, int b) {
        return a + b;
    }

    public double add(double a, double b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }

    public double subtract(double a, double b) {
        return a - b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }

    public double multiply(double a, double b) {
        return a * b;
    }

    public int divide(int a, int b) {
        return a / b;
    }

    public double divide(double a, double b) {
        return a / b;
    }
}
```

----

7. **Packages and their types with examples**:

   - A package is a way to organize related classes and interfaces into a single unit, making it easier to manage and locate them.

   - Types of packages:
     1. **Built-in Packages**: These are the packages that are already available in Java, such as `java.lang`, `java.util`, etc.

     2. **User-defined Packages**: These are the packages created by users to organize their classes. For example, if you have a set of utility classes, you might put them in a package like `com.example.utilities`.

     Example of creating a user-defined package:

```java code:-
// File: MyPackageExample.java
package com.example;
```

```
    public class MyPackageExample {
        // ...
    }
```

----

8. **Difference between Abstraction and Encapsulation**:

   - **Abstraction**:
     - Abstraction focuses on hiding the implementation details and showing only the essential features of an object.
     - It is achieved through abstract classes and interfaces.
     - It is more about designing and planning the structure of the system.

   - **Encapsulation**:
     - Encapsulation is about wrapping the data (variables) and code acting on the data (methods) together as a single unit.
     - It helps in controlling the access to certain components, preventing unintended interference.
     - It is implemented using access specifiers (public, private, protected) and getter/setter methods.

----

9. **String and how to declare a string in Java**:

   - In Java, a string is an object that represents a sequence of characters.
   - You can declare a string in Java using either of the following methods:

~~~
   ```java code :-
   String str1 = "Hello"; // Using string literal
   String str2 = new String("World"); // Using the String class
constructor
~~~

----

10. **Five String Methods in Java**:

    - `length()`: Returns the length (number of characters) of the string.
    - `charAt(int index)`: Returns the character at the specified index.
    - `substring(int beginIndex)`: Returns a substring from the specified index to the end of the string.
    - `substring(int beginIndex, int endIndex)`: Returns a substring within the specified range.
    - `indexOf(String str)`: Returns the index of the first occurrence of a specified substring.

----

11. **Difference between 'throw' and 'throws'**:

   - `throw` is a keyword used to explicitly throw an exception.
   - `throws` is a keyword used in method declarations to indicate that
      the method may throw certain exceptions.

   Example of `throw`:
   ```java code:-
   if (condition) {
       throw new SomeException("This is an error message");
   }
   ```

   Example of `throws`:
   ```java
   public void someMethod() throws SomeException {
       // Method code that may throw SomeException
   }
   ```
----

12. **Why does the Java array index start with 0? Explain 3D arrays**:

   - The array index starts with 0 because of the way elements are stored
      in memory. In languages like C and C++, arrays are implemented as
      pointers to memory locations. Starting with index 0 allows for
      simpler pointer arithmetic.

   - A 3D array is an array of arrays of arrays. It is like a collection
      of 2D arrays. Here is an example:

   ```java code:-
   int[][][] threeDArray = new int[3][4][2];
   ```
----

13. **Enum in Java**:

   - Enums in Java are a special data type that allow a variable to take
      a predefined set of values.

   - Example:

   ```java
   public enum Day {
       SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
   }
   ```

   - Enums can have methods and properties like a regular class.
----

14. **Java program to check if a given number is a perfect square**:

~~~
   ```java
   public class PerfectSquareChecker {
       public static boolean isPerfectSquare(int num) {
           int sqrt = (int) Math.sqrt(num);
           return sqrt * sqrt == num;
       }

       public static void main(String[] args) {
           int number = 25;
           if (isPerfectSquare(number)) {
               System.out.println(number + " is a perfect square.");
           } else {
               System.out.println(number + " is not a perfect square.");
           }
       }
   }
  ------------------------------------------------------------------------
-----------------

15. **Array declaration and initialization**:

   -  Declaration:

   ```java
   int[] arr; // Declaration of an int array
   ```

   -  Initialization:

   ```java
   arr = new int[5]; // Initializing the array with size 5
  ------------------------------------------------------------------------
-----------------

16. **Program to initialize an array from the user and print the sum of
its elements**:

   ```java
   import java.util.Scanner;

   public class ArraySum {
       public static void main(String[] args) {
           Scanner scanner = new Scanner(System.in);

           System.out.print("Enter the size of the array: ");
           int size = scanner.nextInt();

           int[] arr = new int[size];

           System.out.println("Enter the elements of the array:");
           for (int i = 0; i < size; i++) {
               arr[i] = scanner.nextInt();
           }
~~~

~~~
            int sum = 0;
            for (int num : arr) {
                sum += num;
            }

            System.out.println("Sum of the elements: " + sum);
        }
    }
  ------------------------------------------------------------------------
----------------
~~~

17. **try, catch, and finally with examples**:

~~~
   ```java
   try {
       // Code that may throw an exception
   } catch (ExceptionType1 e1) {
       // Handle ExceptionType1
   } catch (ExceptionType2 e2) {
       // Handle ExceptionType2
   } finally {
       // Code to be executed regardless of exceptions
   }
--------------------------------------------------------------------------
-----------------
~~~

18. **Defining and running a thread using the Runnable interface**:

~~~
   ```java
   public class MyRunnable implements Runnable {
       public void run() {
           System.out.println("Thread is running");
       }

       public static void main(String args[]) {
           Thread thread = new Thread(new MyRunnable());
           thread.start();
       }
   }
--------------------------------------------------------------------------
-----------------
~~~

19. **Storing numbers in an ArrayList and finding their average**:

~~~
   ```java code:-
   import java.util.ArrayList;

   public class AverageCalculator {
       public static double calculateAverage(ArrayList<Integer> numbers) {
           int sum = 0;
           for (int num : numbers) {
               sum += num;
           }
           return (double) sum / numbers.size();
~~~

```java
       }

       public static void main(String[] args) {
           ArrayList<Integer> numbers = new ArrayList<>();
           numbers.add(10);
           numbers.add(20);
           numbers.add(30);

           double average = calculateAverage(numbers);
           System.out.println("Average: " + average);
       }
   }
```
----

20. **Difference between while and do-while loop**:

   - `while` loop: Checks the condition first before executing the loop body. It may not execute the loop body at all if the condition is false from the start.

   - `do-while` loop: Executes the loop body at least once and then checks the condition.

   Example of a `while` loop:
   ```java
   while (condition) {
       // Code to be executed
   }
   ```

   Example of a `do-while` loop:
   ```java
   do {
       // Code to be executed
   } while (condition);
   ```
----

jump Statements in Java
21.Jump statements allow you to change the flow of control in a program. There are three main jump statements in Java:

break: The break statement is used to exit a loop or switch statement early. When encountered, it terminates the current loop or switch statement and transfers control to the statement immediately after it.

continue: The continue statement is used to skip the current iteration of a loop and proceed with the next iteration. It is often used to skip certain iterations based on a condition.

return: The return statement is used to explicitly return from a method. It can be used to provide a value back to the caller of the method.

----

22. **Program to find the sum of each row of a 2D int array**:

~~~
   ```java code:-
   public class RowSum {
       public static void main(String[] args) {
           int[][] arr = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

           for (int i = 0; i < arr.length; i++) {
               int sum = 0;
               for (int j = 0; j < arr[i].length; j++) {
                   sum += arr[i][j];
               }
               System.out.println("Sum of row " + i + ": " + sum);
           }
       }
   }
~~~

----

23. Effect of Defining Variables, Methods, and Classes as Final:
Final Variables: When a variable is declared as final, its value cannot be changed once it is assigned. This is useful when you want to create constants.

Final Methods: When a method is declared as final, it cannot be overridden by subclasses. This can be useful to prevent modification of a method's behavior.

Final Classes: When a class is declared as final, it cannot be subclassed or extended. This is often used to prevent further inheritance.

----

24. **Static and Dynamic Binding**:

- **Static Binding**: Also known as early binding, static binding happens at compile time. The binding of a method to its body is determined by the compiler based on the reference type.

- **Dynamic Binding**: Also known as late binding, dynamic binding happens at runtime. The binding of a method to its body is determined by the JVM based on the actual object type.

----

25. **Thread Priority**:
In Java, threads can have different priority levels ranging from 1 (lowest priority) to 10 (highest priority). The default priority is 5. Higher priority threads get more CPU time compared to lower priority threads. However, it's important to note that thread priority behavior can vary across different operating systems.

----

26. **Interfaces in Java**:

An interface in Java is a reference type, similar to a class, that can contain only abstract methods, default methods, static methods, and constant (final) variables. It provides a blueprint for classes to implement.

Example:

~~~
   ```java code:-
   interface Shape {
       void draw(); // Abstract method
   }
~~~

````
interface Shape {
    void draw(); // Abstract method
}
````

Any class that implements the Shape interface must provide an implementation for the draw method.

----

27. **Program to delete duplicate values in an ArrayList**:

~~~
   ```java code;-
   import java.util.ArrayList;
   import java.util.HashSet;

   public class RemoveDuplicates {
       public static void main(String[] args) {
           ArrayList<Integer> list = new ArrayList<>();
           list.add(1);
           list.add(2);
           list.add(2);
           list.add(3);
           list.add(4);
           list.add(4);

           HashSet<Integer> set = new HashSet<>(list);
           list.clear();
           list.addAll(set);

           System.out.println("ArrayList after removing duplicates: " +
list);
       }
   }
~~~

----

28. **Program to check if a given string is a palindrome**:

~~~
   ```java
   public class PalindromeCheck {
       public static boolean isPalindrome(String str) {
~~~

```
        String reversed = new StringBuilder(str).reverse().toString();
        return str.equals(reversed);
    }

    public static void main(String[] args) {
        String str = "racecar";
        if (isPalindrome(str)) {
            System.out.println(str + " is a palindrome.");
        } else {
            System.out.println(str + " is not a palindrome.");
        }
    }
}
```

----

29. **Types of Inheritance in Java**:

   - **Single Inheritance**: A subclass extends a single superclass.

   - **Multilevel Inheritance**: A subclass extends a class which is itself a subclass of another class.

   - **Hierarchical Inheritance**: Multiple subclasses extend the same superclass.

   - **Multiple Inheritance** (Not supported in Java): A class inherits from multiple classes.

----

30. **Overloading vs. Overriding**:

   - **Overloading**: It involves having multiple methods with the same name but different parameters within the same class. It is determined at compile time.

   - **Overriding**: It occurs in a subclass that provides a specific implementation of a method that is already provided by one of its superclasses. It is determined at runtime.

