2 What are the differences you have even found between static and final fields and method? Exemplify what will happen if you try to access the static method field with the object instead of class name.

Difference Between Static and final Fields and Method :

| Feature | Static | Final |
|---|---|---|
| Applies to | Methods, fields, blocks, nested classes | Method fields, class-ess. |
| Meaning | Belongs to the class rather than any instance | cannot be modified once assigned (for variables) cannot be overridden (for methods) cannot be extended (for classes |
| Memory Allocation | Stored in the class's memory (shared among all instances) | Depends on where it is used (instance-level or class-level). |

| | | |
|---|---|---|
| Method Behavior | Can be called without creating an instance | cannot be overridden if used on a method |
| Flield Behavior | Shared by all instances of the class | If applied to a variable, it's value cannot be reassigned. |

# What happend If you Access a static Field / Method using an object instead of class Name?

⇒ Although it is not recommended, Java allows accessing static fields and methods through an object reference.

1. It still reffers to the class-level variable/method rather than an instance-specific one.
2. It might create confusion, making the code harder to read.

(3) Write a Java program to find all factorion numbers lot within a given range. A number is called a factorion if the sum of the factorials of its digits equals the number it self. The program should take users input of the lower and upper bounds of the range and print all factorion numbers lol within that range.

use a method to calculate the factorial of digits efficiently.

Simple input: Enter the lower bound of the range: 1

Enter the upper bound of the range: 100000

Factorion numbers in the range:

Simple output:

1, 2, 145, 40585

```
Import java.util.scanner;
Publie Class FactorionNumbers {
    private static final int[] factorials = new int[10];
    Static {
        factorials[0] = 1;
        for(int i=1; i<10; i++) {
            factorials[i] = i * factorials[i-1];
        }
    }
private static boolean isFactorion(int num) {
    int sum=0, temp=num;
    while(temp>0) {
        Sum += factorials[temp % 10];
        temp /= 10;
    }
    return sum == num;
}
public static void main(String[] args) {
    Scanner scanner = new scanner(system.in);
    System.out.print(" Enter the lower bound of the rang:
```

```
int lower = scanner.nextInt();
System.out.print("Enter the upper bound of the range: ");
int upper = scanner.nextInt();
System.out.println("Factorion number in the range:");
boolean found = false;
for (int i = lower; i <= upper; i++) {
    if (isFactorion(i)) {
        System.out.print(i + ", ");
        found = true;
    }
}
if (!found) {
    System.out.println("None");
}
scanner.close();
}
}
```

4. Distinguish the differenceess among class, local and instance variables. What is significance of this keyword?

Class variables (static variables):

- Declaration: marked with the static keyword within a class but outside any method or constructor.
- Scope: Accessible by all instances of the class. Shared among them.
- Life time: Exist from class loading until the program terminates.
- Initialization: Automatically initialized to default values if not explicitly set.

Instance Variables:

- Declaration: Defined within a class but outside any method or constructor.

- Scope : Each instance of the class has its own copy; not shared among instance.
- Lifetime: Exist as long as the instance exists; created when an object is instantiated and destroyed when the object is garbage collected.
- Initialization: Automatically initialized to default values if not explicitly set.

Local variables:

Declaration : Declared within a method, constructor, or block.

Scope : Accessible only within the block where they're declared

Life-time: Exist only during the execution of the block, created when the block is entered and destroyed upon Exit.

Calling other constructors: Within a constructor, this() can be used to invoke another constructor in the same class.

- Returning the current object: useful in methods that need to return the object itself.

It's important to note that this cannot be used in static contexts (like static methods) because static contexts belong to the class, not to any particular instance.

⑤ Write a java code that defines a method to calculate the sum of all elements in an integer array. The method should take an integer array as a parameter and return the sum. Demonstrate this method by passing an array of integers from the main method.

```
public static
public class ArraySumCalculator {
    public static int calculateSum(int[] array) {
        int sum = 0;
        for (int num: array) {
            sum += num;
        }
        return sum;
    }
public static void main(String[] args){
        int[] numbers = {1,2,3,4,5};
        int result = calculateSum(numbers);
System.out.println("The sum of the array elements is:" + result
};
```

6 What is called Access Modifiers? Compare the accessibility of public, private and protected modifiers. Describe different types of variables in java with example.

⇒ In Java, access modifiers are keywords that set that set the accessibility or scope of classes, methods, variables, and constructors. They are fundamental to object oriented programming as they help enforce encapsulation, a core principle restricting direct access to some of an object's components.

Java provides four main types of access modifiers; public, private, protected and the default access.

Comparison of Access Modifiers;

| Modifiers | class | Package | Subclass (Same package) | Subclass (different package) | world |
|---|---|---|---|---|---|
| public | ✓ | ✓ | ✓ | ✓ | ✓ |
| protected | ✓ | ✓ | ✓ | ✓ | ✗ |
| private | ✓ | ✗ | ✗ | ✗ | ✗ |

- public : The member is accessible from any other class.
- protected: The member is accessible within its own package and, in addition, by a subclass of its class in another package.
- private : The member is accessible only within it own class.
- Default: If no access modifier is specified, the member is accessible only within it own package. This is known as package-private access.

Type of variables in Java:

Java variables are containers for storing data values.

Instance variables (Non-static fields): Technically speaking, objects store their individual states in "non-static fields", that is fields declared without the static keyword. Non-static fields are also known as instance variables because their values are unique to each instance of a class.

Class variables (static fields): A class variable is any field declared with the static modifier, this tells the compiler that there is exactly one copy of this variable in existence, regardless of how many times the class has been instantiated.

Local variables: Similar to how an object stores its state in fields, a method will often store its temporary state in local variables. The syntax for

declaring a local variable is similar to declaring a field. There is no special keyword designating a variable as local, that determination comes entirely from the location in which the variable is declared. which is between the opening and closing braces of a method.

```
Public class Example {
        static int classvariable = 10;
    int instance variable = 20;
    public void method ( ) {
        int localvariable = 30;
        System.out.println ("Local variable:" + localvariable);
    }
}
```

in this example, classvariable is a class variable instanceVariable is an instance variable and localvariable is an local variable within the method.

7' Write a program to find the smallest positive root of a quadratic equation of the form.

$$ax^2+bx+c=0$$

using formula: $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$

Simple input: Enter coefficients a, b, c : 1, -3, 2

" output: The smallest positive root is: 1.0.

```
# import java.util.Scanner;
public class QuadraticEquationSolver{
  public static void main (String[] args){
    Scanner scanner = new Scanner (System.in);
     System.out.print ("Enter coefficient a: ");
    double a = scanner.nextDouble ();
    System.out.print ("Enter coefficient b: ");
    double b = scanner.nextDouble();
    System.out.print ("Enter coefficient c: ");
     double c = scanner.nextDouble();
```

```
double discriminant = b*b-4*a*c;

if (discriminant >=0){

  double root1 = (-b - Math.sqrt(discriminant))/(2*a);

 double root2 = (-b + Math.sqrt(discriminant))/(2*a);

double SmallestpositiveRoot = Double.MAX_VALUE;

if (root1>0) {

    SmallestpositiveRoot = Root1;

  }

if (root2>0 && root2<SmallestpositiveRoot)

  {  SmallestpositiveRoot = root2;

  }

if (SmallestpositiveRoot != Double.Max_VALUE) {

 System.out.println("The smallest positive root is:

                    " + SmallestpositiveRoot);

}
```

```
        else {
            System.out.println("No real roots exist");
        }
        scanner.close();
    }
}
```

[8] Write a program that can determine the letters, whitespace and digit. How do we pass an array to a function? write a example.

```
import java.util.Scanner;
public class CharacterCounter {
    public static void main (String[] args) {
        Scanner scanner = new Scanner (System.in);
        System.out.println("Enter a String:");
        String input = scanner.nextLine();
        int letterCount = 0;
        int whitespaceCount = 0;
        int digitCount = 0;
```

```
for (char ch: input.toCharArray()){
    if (Character.isLetter(ch)){
        letterCount++;
    }
    else if (Character.isWhitespace(ch)){
        whitespaceCount++;
    }
    else if (Character.isDigit(ch)){
        digitCount++;
    }
}
System.out.println("Number of letters: "+letterCount);
System.out.println("Number of whitespace: "+whitespaceCount);
System.out.println("Number of digits: "+digitCount);
Scanner.close();
}
}
```

passing an Array to Function in Java:

In Java you can pass an array to a method by specifying the array type in the method's parameters. When you pass an array to a method, you're passing a reference to the array, meaning any changes made to the array element within the method will affect the original array.

```
public class ArrayExample {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5};
        printArray(numbers);
    }
public static void printArray(int[] array){
    for(int num: array){
    System.out.println(num);
    }
}
}
```

2 In java explain how method overriding works in the context of inheritance. what happens when a subclass overrides a method from its superclass? How does the $superclass$ keyword help in calling the superclass method, and what are the potential issues when overriding methods, especially when dealing with constructors

⇒ In java method overriding allows a subclass to provide a specific implementation for a method that is already defined in it superclass. This mechanism enables polymorphism, where a subclass can tailor or completely change the behavior of method inherited from its superclass.

How method overriding work:

- Definition: When a subclass defines a method with

the same name, return type and parameters, as a method in its superclass, it overrides the superclass's method.

• Invocation: When an overridden method is called on an instance of the subclass, the subclass's version of the method is executed, even if the reference is of the superclass type.

```
Class Animal {
    Void makesound() {
        System.out.println("Animal makes a sound");
    }
}

Class Dog extends Animal {
    Void makesound() {
        System.out.println("Dog barks");
    }
}
```

```
public class Main{
    public static class void main (String[] args){
        Animal myDog = new Dog();
        myDog.makeSound();
    }
}
```

In this example the Dog class overrides the makeSound method of the Animal class. When makeSound is called on myDog, the overriden method in Dog is executed.

Using the Supper keyword:

The Super keyword in java refers to the Superclass of the current object. It's particularly useful in method overriding Seenarios:

Accessing Overridden Method: within a subclass, you can call the overridden method of the Superclass using Super.methodName().

Example:

```
class Animal {
    void makeSound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    void makeSound() {
        super.makeSound();
        System.out.println("Dog barks");
    }
}

public class main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.makeSound();
    }
}
```

Here the Dog class's makeSound method first calls super.makeSound() to execute the makeSound method of Animal then adds its own behavior.

Overriding Methods in Constructor:

Risk: calling an overridable method from a Constructor can lead to unexpected behavior. During Object creation, the superclass constructor runs before the subclass constructor. If the superclass constructor calls a method that is overridden in the subclass, the subclass's method will execute before the subclass's constructor has completed its initialization.

```
Class parent {
    parent() {
        display();
    }
Void display() {
  System.out.println("parent display");
};
```

```
class child extends parent {
        int number = 5;

    Void display() {
            system.out.println("child .display : " + number);

    }}

public class main {
        public static void main (String [] args){
            Child child = new child();

    }
}
```

in this case when a child object is created, the parent constructor calls the overridden display method. since the child constructor hasn't finished initializing number, it prints 0 instead of 5.

method

10. Differentiate between static and non static members including necessary example. Write a program that able to check either a number or String is pallindrome or not.

Static members:

- Definition: static member (variable or method) belong to class itself rather than any particular instance

- Characteristies;

(i) Shared Acrosss instances: All instances of the class share the same static variable.

(ii) Access: Can be accessed using the class name without creating an instance.

(iii) Memory Allocation: Allocated when the class is loaded into memory.

```
class counter {
    static int count = 0;
    static void increment() {
        count++;
    }
}
public class main {
    public static void main (String [] args) {
        public static void
            counter.increment();
            counter.increment();
        System.out.println(" count: " + counter.count);
    }}
```

In this example the count variable and the increment method are static. They belong to the counter class and can be accessed without creating an instance of counter.

Non-static member

Definition: ~~As~~ Non-static members are associated with instances of the class. Each object has its own copy.

Access: Require an instance of the class to be accessed.

Memory Allocation: Allocation when an instance is created and deallocated when the instance is destroyed.

```
class counter {
    int count = 0;
    void increment() {
        count++;
    }
}
public class main {
    public static void main (String [] args) {
        Counter obj1 = new Counter();
        Counter obj2 = new Counter();
```

```
        obj1.increment();
        obj2.increment();
        obj2.increment();
    System.out.println("obj1 Count: " + obj1.count);
    System.out.println("obj2 coount: "+obj2.count);
    }
}
```

Here each counter object has its own count variable

b. Incrementing count on obj1 does not affect obj2 count.

# program to check if a Number or String is a palindrome.

```
import java.util.Scanner;
public class palindrome cheeker{
    public static boolean ispalindrome (String input){
        int Left =0;
        int right = input.length()-1;
    while (left< right){
```

```
if (input.charAt(left) != input.charAt(right)) {
    return false;
    }
    left++;
    right--;
}
return true;
}
public static boolen ispalindrome (int number)
{ return is palindrome (String.valueof (number));
}
public static void main (string [] args) {
    Scanner scanner = new scanner (system.in);
System.out.print("Enter a string:");
    String str = scanner.nextLine();
if (ispalindrome (str)) {
    system.out.println( str + " is a palindrome.");
}
else {
```

```
System.out.println (str + " is not a palindrome.");
}
System.out.println ("Enter a number: ");
int num = scanner.nextInt ();
if (ispalindrome (num)) {
System.out.println (num + " is a palindrome.");
}
else { System.out.println (num + " is not a palindrome");
}
scanner.close();
}
}
```

(11) What is called class abstraction and encapsulation? Describe with the example. what are the differences between Abstract class and interface?

Abstraction:

Abstraction focuses on exposing only the essential features of an object while hiding the complex details. It allows developers to work with concepts at a high level without needing to understand all the underlying intricacies.

```
abstract class Vehicle{
    abstract void start();
}
Class Car extends Vehicle{}
        void start() {
    System.out.println("Car starts with a key.");
};
```

```
public class main {
    public static void main (String [] args) {
        Vechicle mycar = new car();
        myCar.start();
    }
}
```

in this example Vehicle is an abstract class with an abstract method start(). The car class extends Vehicle and provides a specific implemention for the start() method.

## Encapsulation:

Encapsulation is the practice of bundling related data and method that operate on that data within a single unit, such as a class. It restricts external access to certain components of an object, promoting modularity and safeguarding data integrity. In java encapsulation is implemented using accesss modifiers

like private, protected and public.

```
public class person {
    private string name;
    public string getName() {
        return name;
    }
    public void setName (String name) {
        this.name = name;
    }
}
public class Main {
    public static void main (String[] args) {
        Person person = new person ();
        person.setName ("Alice");
        System.out.println (person.getName());
    }
}
```

Difference Between Abstract class and Interface:

| Feature | Abstract Class | Interface |
|---|---|---|
| Method implementation | can have both abstract method and concrete method. | prior to Java 8, A reference type in Java, similar to a class that can contain only constants method signatures, default methods, static method. |
| multiple Inheritance | A class can extend only one abstract class due to Java's single inheritance model. | A class can implement multiple interfaces allowing for multiple inheritance of type. |
| Constractor | Can have constractors which are called when an instance of a subclass is created | cannot have constructor. |
| Access modifier | Can have fields and method with various access modifiers (private protected public) | Fields are implicitly public, static, and final. Methods are implicity public and abstract, unless declared as default or Static. |

| | | |
|---|---|---|
| use cases | suitable for situations where classes share a common base and some default behavior | ideal for defining a contract that multiple classes can implement especially when they are unrelated |