IT-23008

QN 2: Differences between static fields, final fields and methods in Java:

Static fields	Final fields	Methods
Belong to the class, shared by all instances	Value cannot be changed once assigned	Define actions performed by Objects.
Accessed using className.fieldName or Object.fieldName	Must be initialized at declaration or in constructor	Called Object.methodName() or, className.methodName() (if static)
can be modified	Cannot be changed after initialization	Can' be overridden (unless final)
Stored in class memory, not per Object	Exists per instance(unless static final)	Stored in method area
Shared among all Objects	Final static makes it a constant	final method cannot be overridden.
Exists once per class	Exists per instance	Can be static, final, or overridden

4 What happens when we access a static field/Method Using an Object :

Static fields and methods belong to the class. not to individual Objects. However, Java allows accessing them through an Object, but it internally redirects the call to the class. There is no difference in execution. ~~No~~ Weather we call Obj.staticMethod(); or ClassName.Static method(); Jave treats them the same internally. ~~There is no~~

But if multiple objects exist, calling a static field / method via an object might mislead others in to thinking it's tied to the object, while it's actually shared across all instances.

Code:

```
class Example {
  static in count =10; // static field

  Static void display(){
  System.out.println("static method called");
  }
}

Public class Test Static{
  Public static void main(String[] args){
    Example obj = new Example();
  System.out.println("Count (via Object) :"+Obj.count);
    Obj.display();
  System.out.println(" Count (via class): "+Example.Count);
  Example.display();
  }
}
```

Output:

```
Count (via Object): 10
 Static method called

Count (via class): 10
 static method called.
```

IT-23008

```
3) impord Java.Util.Scanner;
Public class Factorion_3 {
      Public Static long fact(int n){
        long f= 1;
        for(int i=1; i<=n; i++){
            f *=i;
        }
        returnf;
      }
    Public Static boolean CheckFactorion(int num){
        int OrgNum = num;
        long sum = 0;
       while (num >0){
           int r= num % 10;
           sum = sum + fact(r);
           num = num /10;
        }
        return sum == OrgNum;
    }
  Public static void main(String args[ ]){
   Scanner sc = new Scanner (System.in);
   System.at.print ("Enter the lower bound of the range :");
    int lb = sc.nextInt();
   System.at.print ("Enter the Upper bound of the range : ");
   int ub = sc.nextInt();
   boolean ~~Check~~ Check = false;
```

JT-23008

```
System. out. Println ("Factorion numbers in the range : ");
    for (int i=lb ; i<=ub ; i++){
        if ( CheckFactorion (i) ){
            System.out.println(i);
            check = true;
        }
    }
    if (! Check ){
        System.out.println("No factorion numbers found in
        the given range.");
    }
    sc.close();
  }
}
```

IT-23008

4|| Difference Among class, Local and Instance Variables;

Class Variables (static)	Instance Variables	Local Variables
Declared with the static keyword inside a class.	Declared inside a class but outside methods / constructors.	Declared inside a method, constructor, or block
Shared among all objects of the class	Each object hat its own copy.	Only accessible within the method where it's declared
stored in class memory (static area)	Stored in heap memory	Stored in stack memory
Can be accessed using className.variable or an object.	Accessed using an object	Cannot be accessed outside the method.
Example: static int cnt;	Example: int age;	Example: int temp=5;

5 Significane of this keyword:

The this keyword refers to the current object of a class. It is used to differentiate instance variables from local variables when they have the same name. It improves code readability and avoids variable shadowing. this keyword makes object handling more clear and consistent in Java. It is mainly used for better object reference management inside a class.

IT23008

Exomple:

```
class Examplo {
      int n; // Instance variable
    void setX(int x) {
      this.x = n;
    }
}
```

In the example 'this.x' refers to the instance varriable, 'x' is the local variable.

5]

```
import Java.util.*;
Public class Array_Sum_5{
   Public Static int array_sum(int[] array){
        int sum = 0;
     for(int num: array){
         sum+= num;
     }
     returnng sum;
}
Public Static void main(String[] args) {
     Scanner sc = new Scanner(System.in);
     System.out.print(" Enter the sinze of the array: ");
     int n= sc.nextInt( );
    int[] a= new int[n];
fore (int i=0; i<n; i++) {
    a[i] = sc.next Int();
  }
 int result = array_sum(a);
 System.out.println("The sum of the array is : "+ result);
 sc.close();
 }
}
```

6) Access modifiers in Java define the visibility and accessibility of classes, methods and variables.

There are four type of Java access modifiers:

1. Private
2. Public
3. Default
4. protected

Comparison of public, Private and protected modifiers.

Modifier	Accessible within class	Accessible within package	Accessible by sub class	Accessible everywhere
Private	Yes	NO	NO	NO
Protected	Yes	Yes	Yes	NO
public	yes	Yes	Yes	Yes

-1923043

Didderent type of varuable in Java :

Local variable : Declared inside a method/ blokk.

Local variable : A variable defined within a block or method or constructor is called a local variabe.

The scope of these variables exists only within the block or method. The Local variabe is created at the time of declaration and destroyed after exiting from the block or when the call returns from the function.

Instance variable : Declared inside a class (but outside methods). The scope of these variables ~~exists as long as the~~ only within specific to an object. It exists as long as the object exists.

Static variable : declared inside a class with static keword. Shared among all objects. Exists as long as the class is loaded.

IT-23008

Example:

```
class Example{
  static int classVar = 10; // static

  int instanceVar = 20; // Instance

  void method(){
     int localVar = 30; // Local
     System.out.println("Local: " + localVar);
  }
}
```

7) import Java.util.*;

```
Public class Root_find_7{
 ~~Scanner(System~~
 Scanner sc = new Scanner(System.in);
 double a, b, c;
 System.out.Print("Enter coefficients a, b, and c : ");
  a = sc.nextDouble();
  b = sc.nextDouble();
  c = sc.nextDouble();
  double d = (b*d - 4*a*c);

  if(d>0){
      double root1 = (-b + Math.sqrt(d))/2;
      double root2 = (-b - Math.sqrt(d))/2;

      double res = Math.min(root1, root2);
```

IT23008

```
System.out.println("The smallest positive root is: "+ res);
else {
    System.out.println("No real root exists.");
  }
  sc.close();
 }
}
```

8)

```
import. Java.util.*;
Public class qn_8{
   Public ~~Stat~~ Static void main(String[] args) {
   Scanner sc = new Scanner (System.in);
   System.out.print ("Enter the string: ");
   String s = sc.nextLine();
   int letter = 0, digit =0, space =0;
       if(s.charAt(i) == ' '){
         Space++;
     }
     else if (s.charAt(i) >= '0' && s.charAt(i) <= '9'){
         digit++;
    }
    else
         letter++;
}
```

JT-23008

```
System.out.println("Letters :" + letters);
System.out.println("whitespace:" + space);
System.out.println("Digit:" + digit);
  sc.close();
 };
}
```

19 Difference between Static and Non-Static Members;

Static Members (Static)	Non-Static Members
1. Belong to the class, shared by all objects.	1. Belong to individual objects, each object has its own copy.
2. Stored in class memory (static area)	2. Stored in heap memory
3. Does not requir an object for access.	3. Requires an object to access.
4. Cannot use "this" keyword	4. Can use "this" keyword.
5. Cannot be overridden (but can be hidden)	5. Can be overridden in subclasses.
6. Example : static int count;	7. Examble : int age

JT-23006

Example:

```
 class Example {
      Static int staticVar = 10; // Static member.
      int nonStaticVar = 20; // NonStatic Member.

 Static void StaticMethod(){
     System.out.println("static Method "+ staticVar);
     }

Void nonStaticMethod(){
     System.out.println("Non-static method:"+ nonStaticV
 }
}
Public class Text {
   Public static void main (String[] args) {
      System.out.Println(Example.staticVar);
      Example.staticMethod();

    Example obj = new Example();
     System.out.println (Obj.nonStaticVar);
      obj.nonStaticMethod();

   }
}
```

Output:

```
      10
      Static Method : 10
      20
      Non-static Method : 20
```

IT-23008

Code:

```
import Java.Util.*;
Public class Pallindrome_10{
  Public static void main(String[] args){
   Scanner sc = new Scanner(System.in);
   System.out.print("Enter the string : ");
   String r = "";
   for (int i=s.length()-1; i>=0; i--){
       r+=s.charAt(i);
   }
   boolean check = true;
   for (int i = 0; i<s.length(); i++){
       if (s.CharAt(i)!=r.charAt(i)){
         check=false;
          break;
     } }
   if (check) System.out.println("Pallindrome");
   else System.out.println("Not Pallindrome");
   Sc.close();
 }
}
```

J7-23008

11

Abstraction : Abstraction is a process of ~~hidd~~ hiding the implementation details and Sharing only functionality to the user.

Abstract class: Java abstract class is a class that can not be instantiated by itself, it needs to be subclassed by another class to use its properties.

An abstract class is declared using the "abstract" keyword in its class definition.

Encapsulation: Encapsulation is a process of binding data and method together in a single unit, providing Controlled access to data.

Example of Abstract Classes :

```
Public  abstract class Shape {
        public abstract double area();
        public void display() {
        system.out.println("This is a shape.");
   }
}
```

IT-23008

```
Class Programmer {
    Private String name;

    Public String getName(){ return name; }

    Public void setName( String name){ this.name = name;}
 }

Public class Encap {
   Public static void main(String[] args){
   Programmer P = new Programmer();
    P.SetName("Mahmudul");
    System.out.println("Name =>" + P.getName());
  }
}
```

Abstract Class VS Interface:

Abstract class	Interface
1. Can have both abstract and Concrete method	1. Can have only abstract method
2. Supports partial abstraction	2. Supports full abstraction
3. Can have Constructors.	3. Cannot have Constructors.
4. Allows instance variables	4. variables are public Static, and final by defult.
5. A class Can extand only one abstract class	5. A Class Can implement multiple interfaces.
6. Can have Static and non-static methods	6 Can have only static methods.
7. Methods Can have any access modifiern	7. Methods are public by defult.

IT-23008

14) Significane of BigIntegerr in Java;

The BigIntegerr Class in Java is used to handle very large numbers that Cannot be stored in int or long.

It supports arithmetic operation. It is found in Java.math.BigIntegerr.

Code:

```
    import.Java.Util.*;
    import Java.math.BigIntegerr;
  Public class Factorial_bigInt_14{
     Static BigIntegerr factorial(BigIntegerr n){
       BigIntegerr fact = BigIntegerr.ONE;
  for(BigIntegerr i = BigIntegerr.ONE; i.CompareTo(n) <=0;
       i = i.add(BigIntegerr.ONE)){
           fact = fact.multiply(i);
       }
     return fact;
  }
  Public Static void main(String[] args){
     Scanner sc = new Scanner(System.in);
     System.out.print("Enter the number: ");
   BigIntegerr num = Sc.NextBigIntegerr();
 System.out.Println("Factorial of "+num+" is :\n"+factorial(num));
   Sc.close();
  }
}
```