Answer of the question-2

Ans:

Method overriding overriding in a feature in java that allows a subclass to provide a new implementation for a method that is already define in its superclass. How its work in inheritance.

(i) When a subclass overrides a method, the subclass version of the method gets executed even if the method called on a superclass reference holding a superclass object.

(ii) This process is called runtime polymorphism because the method call is resolved at runtime

(iii) overriding enables customized behavior for subclass object while maintaining a consistent interface

What happens when a subclass overrides a method –

1. Subclass method executes, replacing the superclass method,
2. Runtime polymorphism determines method execution at runtime
3. Superclass method is hidden unless called using super
4. Overriding rules apply:
5. super can call the overridden superclass method.

The super keyword is used to call an overridden method from the superclass. This allows the subclass to extend or modify the behaviour without completely replacing it.

1. Potential issue when overriding methods:

i) visibility Restriction - cannot reduce access

ii) Exception limitation - cannot throw broader exceptions.

iii) Final state method - Final method can't be overriden,
static method are hidden, not overridden.

2. Issue with Constructors:

i) Constructors cannot be overridden because they are not inherited.

ii) super() must be used for superclass initialization.

- if the superclass has a parameterized constructor the subclass must explicitly call super(argument);
- If no explicit super() is used, java inserts a default constructor call, which may cause an error if the superclass lacks a no argument constructor.

Answer of the question No-10

Ans:

Difference between Static and Non Static member in java -

Feature	Static Members	Non-static Members
Defination	Belong to the class and are Shared amongall object	Belong to the individual object each instance has its own copy
Access	Accessed using class name of instance	Access only through an object of class
Memory Allocation	Stored in the method area	Stored in the heap memory
Invocation	Can be called without creating an object	Requires object creation before
Usage	Used for constant, utility methods and shared properties.	Used for object specific behaviour and instance data
Example	Company name, gravity, ~~mon~~national. anthem, domain extension	Employee ID, Can number phone number

Check if a number is string is palindrome

Program:

```
import java.util.Scanner;
public class PalindromeCheck{
  static boolean Pal(String s){
     return s.equalsIgnoreCase(new StringBuilder(s)
                                .reverse().toString());
  }
  Static boolean Pal(Int n){
    int rev=0, temp=n;
    while(temp>0){
      rev = rev*10+temp%10;
      temp/=10;
    } return n == rev;}
public Static void main(String[]args){
   Scanner sc = new Scanner(System.in);
   System.out.printf("Enter a number:");
    int n = sc.nextInt();
  System.out.println( n+(Pal(n)?"is":"is not")+"a Pal");
  System.out.print ("Enter a string:");
```

```
String s= sc.next();
System.out.println(st(Pal(s))?"is":"is not" +
                                    "a Palindrome");
sc.close();}
}
```

Answer of the question no - 11

Class Abstraction:

Class abstraction is the process of simplifying complex reality by modeling only the essential attribute and behaviours of an object while hidding unnecessary details. It focus on "what" an object does rather than "how" it does it.

Example: A vehicle class may define an abstract method start(), but each subclass (eg: car bike) proviten its own implementation.

Class Encapsulation:

Encapsulation is the building of data (attributes) and methods (behaviors) that operate on that data within a single unit. It also involves controlling the access to the internal data of an object, preventing direct modification from the

outside. This is often achived through access modifiers like private, protected and public.

Example: A BankAccount class has a balance variable marked private and users can access it only through getBalance() and deposits methods. These methods can include logic to validate the operations and maintain the integrity of data.

Difference between abstract class and Interface

Abstract class	Interface
1. Abstract class can have both abstract and concrete methods	1. Interface contains only abstract methods
2. Can have instance variables with any access modifier	2. Can have only public static, final constants.
3. Can have constructors	3. Cannot have constructors.
4. Can include both implemented and non-implemented methods	4. All methods are public and abstract by default
5 Used when classes share common behaviour and need some abstraction	5 Used when different classes must follow a common contract but share no implementation.

Answer to the questions 12

Java program:

```
import java.util.Scanner;
Class BaseClass{
    void printResult(String msg, object result){
        System.out.println(msg+result);
    }
    }
Class SubClass extends BaseClass{
    double Summation(){
        double Sum=0.0;
    for(double i=1.0; i>0.1; i-=0.1){
        Sum+=i;
        }
        return Sum;
    }
    }
```

```
class DivisorMultiple class extends BaseClass{
  int gcd(int a, int b){
    while(b!=0){
  int temp=b;
  b=a%b
  a=temp;}
  return a;
  }
class NumberConversion class extends BaseClass{
String toBinary(int num){
return integer.toBinaryString(num);
}
String toHex(int num){
  return Integer.toHexString(num).toUpperCase();
}
 String toOctal(int num){
 return Integer.toOctalString(num);
 }}
```

```
Class Custom PrintClass{
  void pr(String msg)
  System.out.println(">>"+msg);
}
}
public Class MainClass{
  public Static void main(String[]args){
  Scanner sc=new scanner(System.in);
  SumClass sumobj=new SumClass();
  DivisorMultipleClass obj=new DivisorMultipleClass();
  NumberConversionClass no obj=new NumberConvertClass();
  CustomprintClass printobj=new CustomerprintClass();

  doubleSum=Sum obj.SumSeries();
  Sumobj.printResult("Sum of Series:",Sum);

  System.out.println("Enter two numbers for GCD
                      & LCM:");
```

```
in num1= Sc.nextInt(); num2=Sc.nextInt();
Sumobj.printResult("GCD:", fmobj, gcd(num1, num2));
Sumobj printResult("LCM:", fmobj, lcm(num1, num2));
Systemout.print("Enter a decimal number for
                Conversion:");
    int num= Sc.nextInt();
    printobj.pr("Binary:" nc obj to Binary(num));
    printobj pr("Hexidecimal:" nc obj to Hexa(num);
    printobj pr("octal:" neobj.to octal(num));
    Sc.clann();
  }
}
```

Answer to the question 13

Complete the program based on the provided uml diagram we need to implement the Geometricobject circle and rectangle classes along with a main class to demonstrate their functionality Below is the java code that follows the UML Scenario:

```
import.java.util.Date;
   Class Geometric object{
private String Color;
private boolean Filled;
private Date datecreated;

public Geometricobject(){
   this Color = "white";
   this.filled = "False";
   this.datecreated = new Date();

   }
```

```
public GeometricObject(String color, boolean filled){
    this.color = color;
    this.filled = filled;
    this.dateCreated = new Date();
}
public String color(){
    return color;
}
public void setColor(String color){
    this.color = color;
}
public boolean isFilled(){
    return filled;}
public void setFilled(boolean filled){
    this.filled = filled;
}
```

```
Class circle extend Geometricobject {
    private double radius;

    public circle () {
    Super ();
    this.radius = 1.0; }
    public circle (double radius) {
        Super ();
        this.radius = radius;
        }
    public double getradion (){
            return radius; }

public void set Raddius (double radius) {
        this radius = radius;

public double get area (){
        return math.PI  radius * radius;
```

```
public Class main{
    public static void main (String [ ] args) {
    Circle circle = new circle (5.0, "blue", true);
    circle.printcircle();

System.out.println("Area" + circle.getArea());
System.out.println("Perimeter:" + circle.getperimeter());
System.out.println("Diameter:" + circle.getDiameter());

Rectangle rectangle = new Rectangle (4.0, 5.0, "red", false);
System.out.println (rectangle.toString());
System.out.println ("Area" + rectangle.getArea());
System.out.println ("perimeter:" + rectangle.getperim());
}
}
```

Answer to the question No-14

The bigInteger class in java is part of the java.math package and is used to handle arbitrarily large integers. unlike primitive data type like int or long which have fixed size limits (int can store up to $2^{31}-1$ and long up to $2^{63}-1$) BigInteger can store integers of virtually any size limited only by the available memory. This makes it ideal for calculations involving very large numbers such as factorials of large integers, cryptographic algorithms or precise mathematical computations.

Below is a java program that calculates the factorial of any integer using BigInteger.

```
import java.math.BigInteger;
import java.util.Scanner;
public Class FactorialCalculator{
    public static BigInteger factorial(int n){
        BigInteger result = BigInteger.ONE;
        for (int i=2; i<n; i++){
        result = result.multiply(BigInteger.valueOf(i));
    }
```

Answer to the question No-16

Polymorphism in java

Polymorphism is the ability of an object to take on many forms. In java it allows a single method or class to operate an object of different types. It is achieved through method overriding and method overloading

- Dynamic method dispatch

It is the mechanism by which a call to an overridden method is resolved of runtime rather than compile time. It is the foundation of runtime Polymorphism in java.

Example of polymorphism using interface and method overriding

```
Class Animal {
  void Sound(){
    System.out.println("Animal make a Sound");
  }
}
Class Dog extends Animal {
  @Override
  void Sound(){
    System.out.println("Dog bark");
}
Class Cat extends Animal {
  @Override
      void Sound(){
        System.out.println("cat meow");
  }
  }
```

```
public class main4 {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();
        Animal myDog = new Dog();
        Animal myCat = new Cat();
        myAnimal.Sound();
        myDog.Sound();
        myCat.Sound();
    }
}
```