CMR
INSTITUTE OF
TECHNOLOGY

USN

Internal Assessment Test 1 – Aug 2024

Sub:	**Object Oriented Programming with Java**							**Sub Code:**	**22MCA22**
Date:	**12.08.24**	**Duration:**	**90 min's**	**Max Marks:**	**50**	**Sem:**	**II**	**Branch:**	**MCA**

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

	PART I	MARKS	OBE	
			CO	RBT
1	a) Explain the object oriented principles of Java b) What is for-each loop? Write a program to implement it. **OR**	[5+5]	CO1	L1
2	Write a program in Java for String handling which performs the following: i. Take two string input and check if they are same using equals() method. ii. Append the two strings and print the result. iii. Check the capacity of StringBuffer objects before and after the append.	[3+3+4]	CO1	L3
3	**PART II** What is a constructor? What are the differences between constructor and method? Write a program to implement parameterized constructor. **OR**	[2+3+5]	CO1	L2
4	What is the use of 'this' keyword? Demonstrate with examples	[10]	CO1	L2

5	**PART III** Differentiate method overloading and method overriding. Explain with help of program **OR**	[5+5]	CO2	L2
6	What is super keyword? Explain the three uses of super with example.	[10]	CO1	L2
7	**PART IV** Explain the use of bitwise operators in Java. **OR**	[10]	CO1	L2
8	What are the uses of final keyword in Java? Explain each with examples.	[10]	CO1	L2
9	**PART V** What is inheritance? Explain the types of inheritance in Java. **OR**	[10]	CO2	L1
10	Create a class with a method that prints "This is parent class" and its subclass with another method that prints "This is child class". Now, create an object for each of the class and call 1 - method of parent class by object of parent class 2 - method of child class by object of child class 3 - method of parent class by object of child class	[10]	CO2	L4

1a. OOPs Principles

Encapsulation, Inheritance and Polymorphism are the basic principles of any object oriented programming language.

Encapsulation is the mechanism to bind the data and code working on that data into a single entity. It provides the security for the data by avoiding outside manipulations. In Java, encapsulation is achieved using classes. A class is a collection of data and code. An object is an instance of a class. That is, several objects share a common structure (data) and behavior (code) defined by that class. A class is a logical entity (or prototype) and an object is a physical entity. The elements inside the class are known as members. Specifically, the data or variables inside the class are called as member variables or instance variables or data members.

Inheritance allows us to have code re-usability. It is a process by which one object can acquire the properties of another object. It supports the concept of hierarchical classification. For example, consider a large group of animals having few of the abstract attributes like size, intelligence, skeletal structure etc. and having behavioral aspects like eating, breathing etc. Mammals have all the properties of Animals and also have their own specific features like type of teeth, mammary glands etc. that make them different from Reptiles. Similarly, Cats and Dogs have all the characteristics of mammals, yet with few features which are unique for themselves. Though Doberman, German-shepherd, Labrador etc. have the features of Dog class, they have their own unique individuality.

Polymorphism can be thought of as one interface, multiple methods. It is a feature that allows one interface to be used for a general class of actions. The specific action is determined by the exact nature of the situation. Consider an example of performing stack operation on three different types of data viz. integer, floating-point and characters. In a non-object-oriented programming, we write functions with different names for push and pop operations though the logic is same for all the data types. But in Java, the same function names can be used with data types of the parameters being different.

1b.for each loop:

The for-each loop is used to traverse array or collection in java.
• It is easier to use than simple for loop because we don't need to incrementvalue and use subscript notation.
• It works on elements basis not index. It returns element one by one in thedefined variable.

Syntax:
```
for(type var:array)
{
//code to be executed
}
```

Example
```
class ForEach
{
public static void main(String[] args)
{
String rnsit[]={"raghu", "mca","ec","mba"};
for(String k:rnsit)
{
System.out.println(k);
}
}
}
```

2.

```
import java.util.*;
public class example{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		StringBuffer str1=new StringBuffer();
		System.out.println(str1.capacity());
		str1.append(“Hello”);
		StringBuffer str2=new StringBuffer(sc.nextLine());
		String s1=str1.toString();
		String s2=str2.toString();
		System.out.println(s1);
		System.out.println(s2);
		System.out.println(s1==s2);
		System.out.println(s1.compareTo(s2));
		sc.close();
		System.out.println(“Capacity before appending”+str1.capacity());
		System.out.println(“The appended string”+str1.append(str2));
		System.out.println(“Capacity after appending”+str1.capacity());
	}
}
```

3. Constructors

• A constructor initializes an object immediately upon creation.
• It has the same name as the class in which it resides and is syntactically similar to a method.
• Once defined, the constructor is automatically called immediately after the object is created, before the new operator completes.

diff b/w constructor and method

Constructor

* A block of code that initialize at the time of creating a new object of the class is called constructor.
* The constructor name will always be the same as the class name.
*A class can have more than one parameterized constructor. But constructors should have different parameters.

Method

*A set of statements that performs specific task with and without returning value to the caller is known as method.
*We can use any name for the method name, such as addRow, addNum and subNumbers etc.
*A class can also have more than one method with the same name but different in arguments and datatypes.

Program for Parameterized Constructor

```
class Student4
{
int id;
String name;
Student4(int i,String n)
{
id = i;
name = n;
}
void display()
{
System.out.println(id+" "+name);
}
public static void main(String args[])
{
Student4 s1 = new Student4(111,"Karan");
Student4 s2 = new Student4(222,"Aryan");
s1.display();
s2.display();
}
}
```

4. this keyword in java uses

• There can be a lot of usage of java this keyword.
• In java, this is a reference variable that refers to the current object.
• this keyword can be used to refer current class instance variable.
• this keyword can be used to invoke current class method (implicitly)
• this can be passed as an argument in the method call.
• this can be passed as argument in the constructor call.

EX:1 to invoke current class method

```
class A
{
void m()
{
System.out.println("cmrit");
}
void n()
{
System.out.println("bangalore");
this.m();
}
}
class TestThis4
{
public static void main(String args[])
{
A a=new A();
a.n();
}
}
```

EX2:to invoke current class constructor

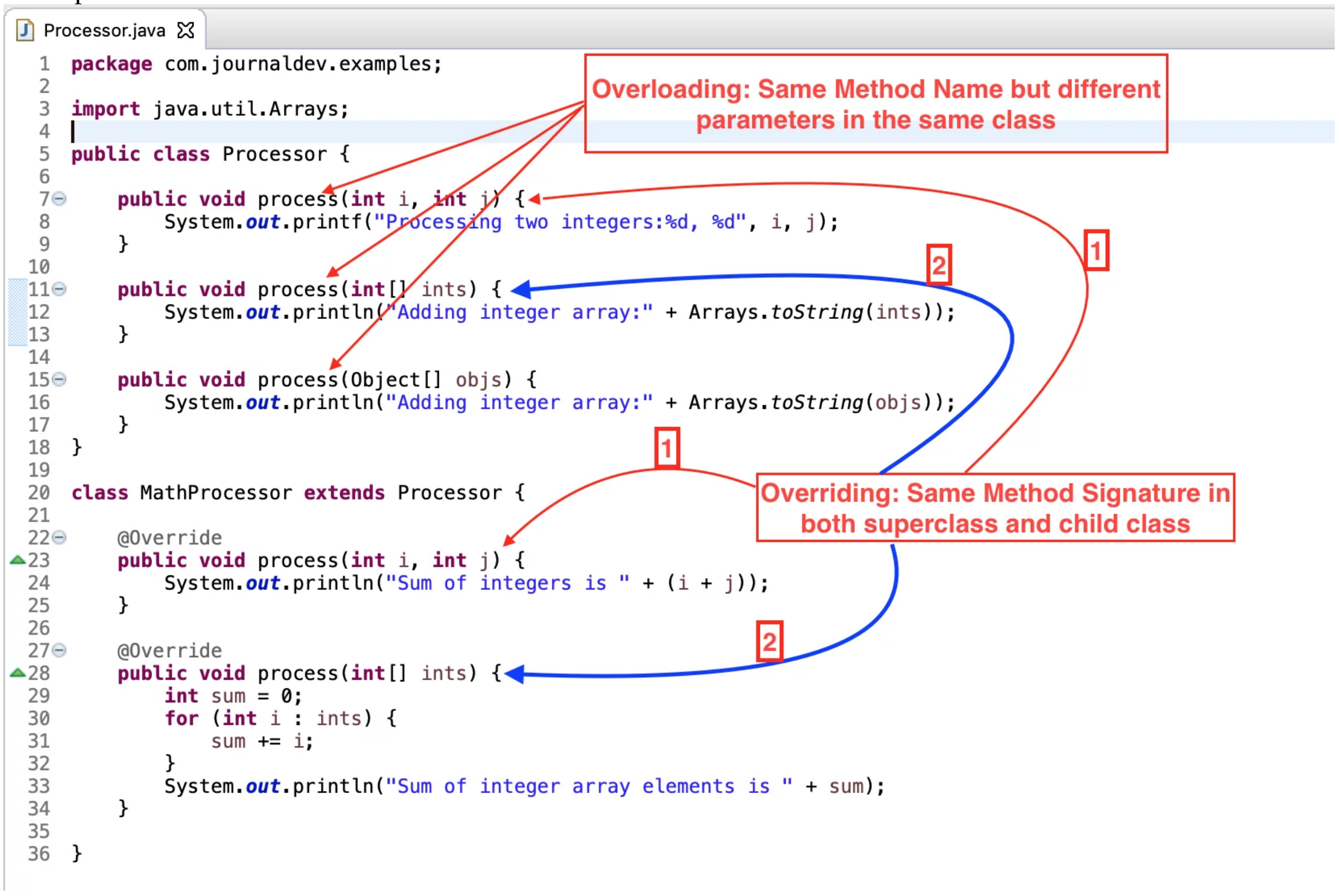
```
class A
{
A()
{
System.out.println("hello a");}
```

A(int x)
{
this();
System.out.println(x);
}
}
class TestThis5
{
public static void main(String args[])
{
A a=new A(10);
}
}

5. Differences between method overloading and overriding:

- Overriding implements **Runtime Polymorphism** whereas Overloading implements **Compile time polymorphism**.
- The method Overriding occurs between **superclass and subclass**. Overloading occurs between the methods in the **same class**.
- Overriding methods have the **same signature** i.e. same name and method arguments. Overloaded method **names are the same but the parameters are different**.
- With Overloading, the method to call is determined at the compile-time. With overriding, the method call is determined at the runtime based on the object type.
- If overriding breaks, it can cause serious issues in our program because the effect will be visible at runtime. Whereas if overloading breaks, the compile-time error will come and it's easy to fix.

Example:

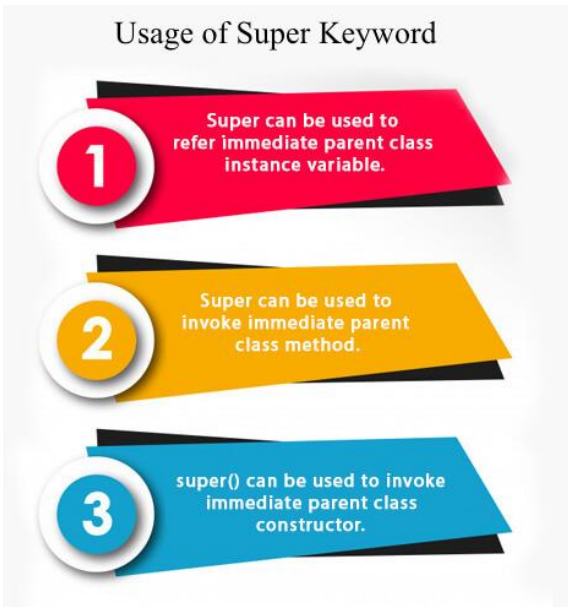

Processor.java
Overloading: Same Method Name but different parameters in the same class
Overriding: Same Method Signature in both superclass and child class
1
2
1
2

```
package com.journaldev.examples;

import java.util.Arrays;

public class Processor {

    public void process(int i, int j) {
        System.out.printf("Processing two integers:%d, %d", i, j);
    }

    public void process(int[] ints) {
        System.out.println("Adding integer array:" + Arrays.toString(ints));
    }

    public void process(Object[] objs) {
        System.out.println("Adding integer array:" + Arrays.toString(objs));
    }
}

class MathProcessor extends Processor {

    @Override
    public void process(int i, int j) {
        System.out.println("Sum of integers is " + (i + j));
    }

    @Override
    public void process(int[] ints) {
        int sum = 0;
        for (int i : ints) {
            sum += i;
        }
        System.out.println("Sum of integer array elements is " + sum);
    }

}
```

6. super keyword

A Superclass Variable Can Reference a Subclass Object: * A reference variable of a superclass can be assigned a reference to any subclass derived from that superclass.

```
class Base
{
void dispB(){
System.out.println("Super class " );
}
}
class Derived extends Base
{
void dispD()
{
System.out.println("Sub class ");
}
}
class Demo
{
public static void main(String args[])
{
Base b = new Base();
Derived d=new Derived();
b=d;
b.dispB();
//b.dispD(); error!!
}
}
```


Usage of Super Keyword
1
Super can be used to refer immediate parent class instance variable.
2
Super can be used to invoke immediate parent class method.
3
super() can be used to invoke immediate parent class constructor.

Using Super
• Sometimes, we may need to initialize the members of super class while creating subclass object.
• Writing such a code in subclass constructor may lead to redundancy in code.

```
class Box
{
double w, h, b;
Box(double wd, double ht, double br)
{
w=wd; h=ht; b=br;
}
}
class ColourBox extends Box
{
int colour;
ColourBox(double wd, double ht, double br, int c)
{
w=wd; h=ht; b=br; //code redundancy
colour=c;
}
}
```

• Also, if the data members of super class are private, then we can’t even write such a code in subclass constructor.
• To avoid such problems, Java provides a keyword called super.
• Whenever a subclass needs to refer to its immediate superclass, it can do so by use of the keyword super.

• super has two general forms.
– The first calls the superclass' constructor.
– The second is used to access a member of the
superclass that has been hidden by a member of a subclass.
• If we use super( ) to call superclass constructor, then it must be the first statement executed inside a subclass constructor.

```
class Box
{
double w, h, b;
Box(double wd, double ht, double br)
{
w=wd; h=ht; b=br;
}
}
class ColourBox extends Box
{
int colour;
ColourBox(double wd, double ht, double br, int c)
{
super(wd, ht, br);
colour=c;
}
}
class Demo
{
public static void main(String args[])
{
ColourBox b=new ColourBox(2,3,4, 5);
}
}
```

• The super keyword can also be used toaccess superclass member (variable or method).
• This second form of super is most applicable to situations in which member names of a subclass hide members by the same name in the superclass.

```
class A
{
int a;
}
class B extends A
{
int a;//this a hides a in A
B(int x, int y)
{
super.a=x;
a=y;
}
void disp()
{
System.out.println("super class a: "+ super.a);
System.out.println("sub class a: "+ a);
}
}
```

7.

Operators	**Symbol**	**Uses**
Bitwise AND	&	op1 & op2
Bitwise exclusive OR	^	op1 ^ op2
Bitwise inclusive OR	|	op1 | op2
Bitwise Compliment	~	~ op
Bitwise left shift	<<	op1 << op2
Bitwise right shift	>>	op1 >> op2
Unsigned Right Shift Operator	>>> op >>>	number of places to shift

```
public class BitwiseAndExample
{
public static void main(String[] args)
{
int x = 9, y = 8;
// bitwise and
// 1001 & 1000 = 1000 = 8
System.out.println("x & y = " + (x & y));
}
}
Output:
x & y = 8
```

```
public class BitwiseXorExample
{
public static void main(String[] args)
{
int x = 9, y = 8;
// bitwise XOR
// 1001 ^ 1000 = 0001 = 1
System.out.println("x ^ y = " + (x ^ y));
}
}
Output:
x ^ y = 1
```

```
public class BitwiseInclusiveOrExample
{
public static void main(String[] args)
{
```

```
int x = 9, y = 8;
// bitwise inclusive OR
// 1001 | 1000 = 1001 = 9
System.out.println("x | y = " + (x | y));
}
}
```

Output:

```
x | y = 9
```

```
public class BitwiseComplimentExample
{
public static void main(String[] args)
{
int x = 2;
// bitwise compliment
// ~0010= 1101 = -3
System.out.println("~x = " + (~x));
}
}
```

Output:

```
~x = -3
```

```
public class SignedRightShiftOperatorExample
{
public static void main(String args[])
{
int x = 50;
//b=a>>n   =>   b=a/2^n
System.out.println("x>>2 = " + (x >>2));
}
}
```

Output:

```
x>>2 = 12
```

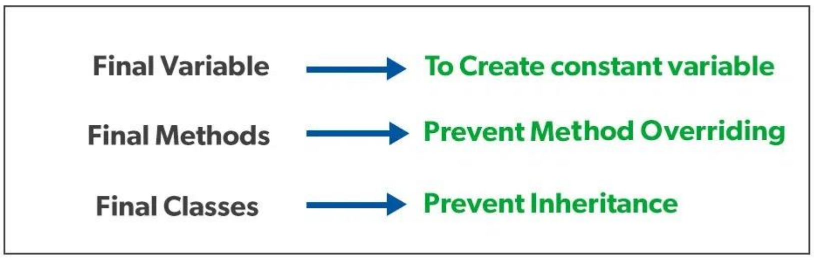
```
public class SignedLeftShiftOperatorExample
{
public static void main(String args[])
{
int x = 12;
//b=a>>n  =>  b=a*(2^n)
System.out.println("x<<1 = " + (x << 1));
}
}
```

Output:

```
x<<1 = 24
```

8. The final method in Java is used as a non-access modifier applicable only to a variable, a method, or a class. It is used to restrict a user in Java.
The following are different contexts where the final is used:

- Variable
- Method
- Class



Characteristics of final keyword in Java:

In Java, the final keyword is used to indicate that a variable, method, or class cannot be modified or extended. Here are some of its characteristics:

Final variables: When a variable is declared as final, its value cannot be changed once it has been initialized. This is useful for declaring constants or other values that should not be modified.

Final methods: When a method is declared as final, it cannot be overridden by a subclass. This is useful for methods that are part of a class's public API and should not be modified by subclasses.

Final classes: When a class is declared as final, it cannot be extended by a subclass. This is useful for classes that are intended to be used as is and should not be modified or extended.

Initialization: Final variables must be initialized either at the time of declaration or in the constructor of the class. This ensures that the value of the variable is set and cannot be changed.

Performance: The use of a final can sometimes improve performance, as the compiler can optimize the code more effectively when it knows that a variable or method cannot be changed.

Security: The final can help improve security by preventing malicious code from modifying sensitive data or behavior.

Overall, the final keyword is a useful tool for improving code quality and ensuring that certain aspects of a program cannot be modified or extended. By declaring variables, methods, and classes as final, developers can write more secure, robust, and maintainable code.

Java Final Variable

When a variable is declared with the final keyword, its value can't be changed, essentially, a constant. This also means that you must initialize a final variable.

If the final variable is a reference, this means that the variable cannot be re-bound to reference another object, but the internal state of the object pointed by that reference variable can be changed i.e. you can add or remove elements from the final array or final collection.

It is good practice to represent final variables in all uppercase, using underscore to separate words.

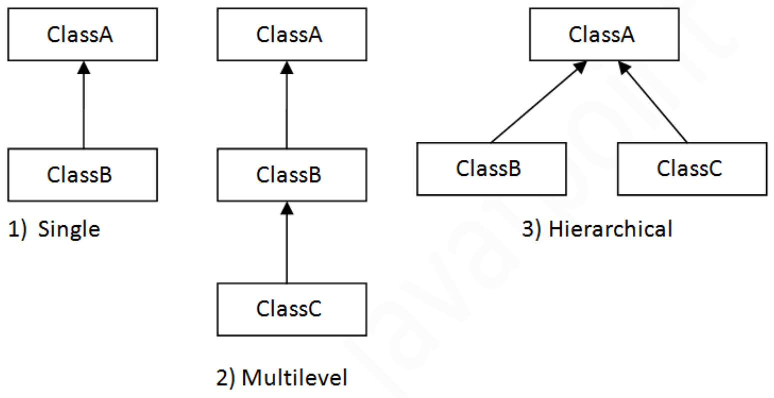
Example:

```
public class ConstantExample {

    public static void main(String[] args) {

        // Define a constant variable PI

        final double PI = 3.14159;



        // Print the value of PI

        System.out.println("Value of PI: " + PI);

    }

}
```

9. Inheritance

•Inheritance allows re-usability of the code.
•In Java, we use the terminology as super class and sub class.
•Inheritance is achieved using the keyword extends.
•Java does not support multiple inheritance.

Types of Inheritance



Single Inheritance Example:

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

Multilevel Inheritance Example:

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Hierarchical Inheritance Example:

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

10.

```
class Parent
{
        Print()
        {
                System.out.println("This is parent class");
        }
}
class Child extends Parent
{
        Show()
        {
                System.out.println("This is child class");
        }
}
class main
{
        public static void main(String[] args)
        {
                Parent p = new Parent();
                Child c = new Child();
                p.print();
                c.show();
                c.print();
        }
}
```