Roll No. _____

Pandit Deendayal Energy University

Mid Semester Examination – (September, 2023)

B. Tech. (Computer Science & Engg.)

Semester - III

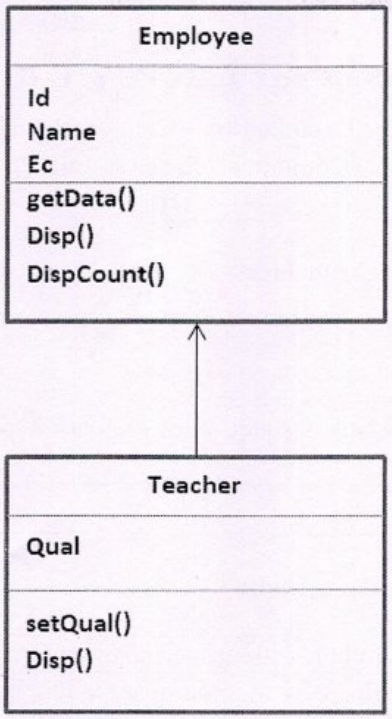
Course Name : Object Oriented Programming with Java
Course Code : 20CP204T

Date: 15/09/2023
Time: 2 hours
Max. Marks: 50

Instructions:

1. Do not write anything other than your roll number on question paper.
2. Assume suitable data wherever essential and mention it clearly.
3. Writing appropriate units, nomenclature, and drawing neat sketches/schematics wherever required is an integral part of the answer.

Question No.	Description	Marks	Course Outcomes (CO's)
1.	Java is platform independent. Explain with proper justification.	4	Co1
2.	WAP in java to overload searchInArray method to search the given integer number from int array, search given String name from String array. If searched value available then return index of the same else return **-1**. Demonstrate the method overloading in main method.	6	Co1
3.	Differentiate between **this** and **super** keyword.	4	Co2
4.	**Consider the following problems** Write a program for Student class having attributes such as Rollno, Name, University name and Student count. All students have unique Rollno, Name, common University name and Student count. The Student count gets incremented after each student enrolment. Use constructor method to initialize each data. Create Disp() and SDisp() method to display all instance and static variables.	6	Co2
5.	Create a class Employee having attributes Id, name and a count to keep accountability on the number of Employees created. An employee can set their class attributes using getData() method, display the data using disp() method. Also create a static method DispCount() to display the number of Employees created. Create a Derived class Teacher having attribute Qualification. It is also containing methods setQual() to set qualification and disp() to display the qualification. Use default and parameterized constructor in both the classes to initialize the attributes and override the method disp() in both the classes.	6	Co3

			
6.	Analyze different usage of final keyword.	4	Co3
7.	**Find the Output and Justify the reason**		

a.	class Test1{
	public static void main(String[] args){
	String[] array = new String[2];
	array[0] = "Java";
	array[0].concat("Skill");
	System.out.println(array[0].length());
	}
		2.5	Co1

b.	class Test2{
	public static void dataPrinting(float f1, float f2){
	System.out.print("float, float "); }
	public static void dataPrinting(double d1, double d2){
	System.out.print("double, double "); }
	public static void main(String[] args){
	dataPrinting(15.0, 25.0f);
		2.5	Co1

	dataPrinting(10, 20); }}		

c.	Find the output if no input is entered during execution of the program.
	public class ABC
	{
	public static void main(String args[])
	{
	System.out.println("Hello"+args[0]);
	}
	}
		2.5	Co1

d.	class Base {
	final public void show() {
	System.out.println("Base::show() called");
	}
	}
	
	class Derived extends Base {
	public void show() {
	System.out.println("Derived::show() called");
	}
	}
	
	public class Main {
	public static void main(String[] args) {
	Base b = new Derived();
	b.show();
	}
	}
		2.5	Co3

8.	**Find the write option**		
a.	Which option is false about the *final* keyword? a. A *final* method cannot be overridden in its subclasses. b. A *final* class cannot be extended. c. A *final* class cannot extend other classes.	2	Co2
b.	In Java, the keyword used to declare a class is ___. a. Class b. Java c. class d. java	2	Co2
c.	Creating an object from a class is also called ____. a. Initializing b. Instantiating c. Interfacing d. None of the above	2	Co2

d.	class Base {
	private int a;
	public void foo() { System.out.println("Base"); }
	}
	
	class Derived extends Base {
		2	Co3

	private void foo() { System.out.println("Derived"); }
	}
	
	public class Main {
	public static void main(String args[]) {
	Base b = new Derived();
	b.a=5;
	System.out.println(b.a);
	b.foo();
	}
	}
	a)5
	Base
	b)Garbage value
	Base
	c)Compiler Error
	d)5
	Derived		

e.	class Base {
	final public void show() {
	System.out.println("Base");
	}
	}
	
	class Derived extends Base {
	public void show() {
	System.out.println("Derived");
	}
	}
	
	class Main {
	public static void main(String[] args) {
	Base b = new Derived();
	b.show();
	}
	}
	a)Base
	b)Derived
	c) Base
	Derived
	d)Error
		2	Co3

Copilot

Sure, let's go through each question from your mid-semester exam and solve them with explanations in Java:

1. Java is platform independent. Explain with proper justification.

Java is platform-independent because of its bytecode and the Java Virtual Machine (JVM). When Java code is compiled, it is converted into bytecode, which is a platform-independent code. This bytecode can be executed on any platform that has a JVM, making Java programs platform-independent.

2. Method Overloading in Java

Write a program to overload `searchInArray` method to search for an integer in an int array and a string in a string array. Return the index if found, else return -1.

```
public class SearchArray {
    public static int searchInArray(int[] array, int value) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == value) {
                return i;
            }
        }
        return -1;
    }

    public static int searchInArray(String[] array, String value) {
        for (int i = 0; i < array.length; i++) {
            if (array[i].equals(value)) {
                return i;
            }
        }
        return -1;
    }

    public static void main(String[] args) {
        int[] intArray = {1, 2, 3, 4, 5};
        String[] strArray = {"Java", "Python", "C++"};

        System.out.println("Index of 3: " + searchInArray(intArray, 3));
        System.out.println("Index of 'Python': " + searchInArray(strArray, "Python"));
    }
}
```

3. Differentiate between `this` and `super` keyword.

- **`this` keyword**: Refers to the current instance of the class. It is used to access class variables and methods.
- **`super` keyword**: Refers to the parent class instance. It is used to access variables and methods of the parent class.

4. Student Class Example

Create a `Student` class with attributes `Rollno`, `Name`, `University name`, and `Student count`. Use a constructor to initialize data and methods to display instance and static variables.

```
public class Student {
    private int rollno;
    private String name;
    private static String universityName = "PDEU";
    private static int studentCount = 0;

    public Student(int rollno, String name) {
        this.rollno = rollno;
        this.name = name;
```

```
        studentCount++;
    }

    public void disp() {
        System.out.println("Rollno: " + rollno + ", Name: " + name + ", University: " + universityName);
    }

    public static void sDisp() {
        System.out.println("Total Students: " + studentCount);
    }

    public static void main(String[] args) {
        Student s1 = new Student(1, "Alice");
        Student s2 = new Student(2, "Bob");

        s1.disp();
        s2.disp();
        Student.sDisp();
    }
}
```

5. Employee and Teacher Classes

Create an `Employee` class with attributes `Id`, `name`, and a count of employees. Create a `Teacher` class derived from `Employee` with an additional attribute `Qualification`.

```
class Employee {
    private int id;
    private String name;
    private static int count = 0;

    public Employee() {
        count++;
    }

    public Employee(int id, String name) {
        this.id = id;
        this.name = name;
        count++;
    }

    public void getData(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public void disp() {
        System.out.println("ID: " + id + ", Name: " + name);
    }

    public static void dispCount() {
        System.out.println("Total Employees: " + count);
    }
}

class Teacher extends Employee {
    private String qualification;

    public Teacher() {
        super();
    }

    public Teacher(int id, String name, String qualification) {
        super(id, name);
        this.qualification = qualification;
    }

    public void setQual(String qualification) {
        this.qualification = qualification;
    }

    @Override
    public void disp() {
```

```
        super.disp();
        System.out.println("Qualification: " + qualification);
    }

    public static void main(String[] args) {
        Teacher t1 = new Teacher(1, "John", "MSc");
        t1.disp();
        Employee.dispCount();
    }
}
```

6. Usage of `final` keyword

- **final variable**: Cannot be changed once initialized.
- **final method**: Cannot be overridden by subclasses.
- **final class**: Cannot be subclassed.

7. Find the Output and Justify

a.

```
class Test1 {
    public static void main(String[] args) {
        String[] array = new String[2];
        array[0] = "Java";
        array[0].concat("Skill");
        System.out.println(array[0].length()); // Output: 4
    }
}
```

Explanation: `concat` does not change the original string, so `array[0]` remains “Java”.

b.

```
class Test2 {
    public static void dataPrinting(float f1, float f2) {
        System.out.print("float, float ");
    }

    public static void dataPrinting(double d1, double d2) {
        System.out.print("double, double ");
    }

    public static void main(String[] args) {
        dataPrinting(15.0, 25.0f); // Output: double, float
        dataPrinting(10, 20); // Output: double, double
    }
}
```

Explanation: Method overloading resolution is based on the closest match.

c.

```
public class ABC {
    public static void main(String args[]) {
        System.out.println("Hello" + args[0]); // Output: ArrayIndexOutOfBoundsException
    }
}
```

Explanation: No input is provided, so accessing `args[0]` throws an exception.

d.

```
class Base {
    final public void show() {
        System.out.println("Base :: show() called");
    }
}

class Derived extends Base {
```

```
    public void show() {
        System.out.println("Derived :: show() called");
    }
}

public class Main {
    public static void main(String[] args) {
        Base b = new Derived();
        b.show(); // Output: Base :: show() called
    }
}
```

Explanation: `final` method cannot be overridden.

8. Multiple Choice Questions

a. Which option is false about the final keyword?

- c. A final class cannot extend other classes.

b. In Java, the keyword used to declare a class is

- c. class

c. Creating an object from a class is also called

- b. Instantiating

d.

```
class Base {
    private int a;
    public void foo() {
        System.out.println("Base");
    }
}

class Derived extends Base {
    private void foo() {
        System.out.println("Derived");
    }
}

public class Main {
    public static void main(String args[]) {
        Base b = new Derived();
        b.a = 5; // Compiler Error
        System.out.println(b.a); // Compiler Error
        b.foo(); // Output: Base
    }
}
```

Explanation: `a` is private and cannot be accessed outside the class.

e.

```
class Base {
    final public void show() {
        System.out.println("Base");
    }
}

class Derived extends Base {
    public void show() {
        System.out.println("Derived");
    }
}

class Main {
    public static void main(String[] args) {
        Base b = new Derived();
```

```
        b.show(); // Output: Base
    }
}
```

Explanation: `final` method cannot be overridden.

Feel free to ask if you have any questions or need further explanations!

Roll No. _____

Pandit Deendayal Energy University

Mid Semester Examination - September 2022

B. Tech. Computer Science and Engineering

Semester – III

Course Name : Object Oriented Programming with Java
Course Code : 20CP204T

Date: 28/9/2022
Time: 2 hours
Max. Marks: 50

Instructions:

1. Do not write anything other than your roll number on question paper.
2. Assume suitable data wherever essential and mention it clearly.
3. Writing appropriate units, nomenclature, and drawing neat sketches/schematics wherever required is an integral part of the answer.

		CO Mapped	BL	Marks
Q- 1	(a) Whether Java is Platform Independent? Justify your answer with proper arguments.	CO1	L1	[05]
	(b) List primitive data types available in Java and explain Integral Promotion rules.	CO1	L1	[05]
Q –2	(a) WAP in Java to overload the add method to find the addition of 2 integers, addition of 2 double numbers, and concatenate 2 strings. Demonstrate the method overloading in the main method.	CO2	L2	[06]
	OR			
	(a) Write a program to create a complex number with two int properties to store the real and imaginary components. Create one default constructor (zero-argument constructor), one parameterized constructor, and one copy constructor for this class. Demonstrate the instantiation of the objects in the main method.	CO2	L2	[06]
	(b) From the code snippets shown on page no. 2, find if there is any error (mention the type and place of error); if no error then find the output **(Any 5. Each carries two marks).**	CO2	L2	[10]
	(c) Explain with examples the usage of ***new*** and ***this***.	CO2	L1	[04]
Q –3	(a) Write a program to implement the object-oriented design shown in Figure - 1 (page no. 4). Use the concept of Inheritance wherever applicable. Do method implementation and overriding wherever applicable. Create the objects of the designed classes in the main() method and call the implemented methods. Show the output of your program.	CO3	L3	[10]
	OR			
	(a) Write a program to implement the object-oriented design shown in Figure - 2 (page no. 4). Use the concept of Interface implementation wherever applicable. Do method implementation wherever applicable. Create the object of the designed classes in the main() method and call the implemented methods. Show the output of your program.	CO3	L3	[10]
	(b) From the code snippets shown on page no. 3, find if there is any error (mention the type and place of error); if no error then find the output **(Any 3. Each carries two marks).**	CO3	L2	[06]
	(c) Explain the usage of ***super*** and ***final*** keywords in Java.	CO3	L1	[04]

Find Error/Output for Q-2 (b) (Any Five)

1.

```
// filename Main.java
class Test {
    protected int x, y;
}

class Main {
public static void main(String args[])
 {
  Test t = new Test();
  System.out.println(t.x + " " + t.y);
 }
}
```

2.

```
// filename Main.java
class Main {
 public static void main(String args[])
 {
  Main obj = new Main();
  System.out.println(obj.fun());
 }
int fun()
 {
  return 20;
  }
}
```

3.

```
//filename Test.java
public class Test
{
  public int getData()
   {
    return 0;
   }
 public long getData()
   {
   return 1;
   }
public static void main(String[] args)
   {
    Test obj = new Test();
   System.out.println(obj.getData());
   }
}
```

4.

```
//filename Test.java
public class Test
 {
   private String function()
    {
      return ("Computer");
    }
  static
   {
     System.out.println("Best Of Luck");
   }
   public static void main(String[] args)
    {
     Test obj = new Test();
     System.out.println(obj.function());
   }
 }
```

5.

```
//filename Test. Java
class Test {
public static void main(String args[])
{
  int arr[2];
 System.out.println(arr[0]);
  System.out.println(arr[1]);
}
}
```

6.

```
//filename Test. java
class Test {
    int i;
  }
class Main
{
  public static void main(String args[])
{
     Test t = new Test();
     System.out.println(t.i);
}
}
```

Q1 (a).

Yes, Java is considered platform-independent, and this characteristic is one of its fundamental features. Java's platform independence is achieved through a combination of factors and technologies. Here are the key arguments justifying Java's platform independence:

1. **Bytecode**: Java source code is compiled into an intermediate form known as bytecode by the Java compiler. Bytecode is not specific to any particular operating system or architecture; it's a low-level representation of the code that can be executed on any platform that has a Java Virtual Machine (JVM) implementation.

2. **Java Virtual Machine (JVM)**: Each platform (Windows, macOS, Linux, etc.) has its own JVM implementation. When you run a Java program, the JVM interprets the bytecode and executes it natively on the host system. Since the JVM is available for various platforms, Java applications can run on multiple operating systems without modification.

3. **Write Once, Run Anywhere (WORA)**: Java's famous motto is "Write Once, Run Anywhere." This means that once you've written and compiled your Java code, you can run it on any platform that supports Java, without needing to modify the code for each specific platform. This portability is a significant advantage for developers and businesses, as it reduces the effort required to support multiple platforms.

4. **Platform-Specific Libraries**: While the core Java language and libraries are platform-independent, Java also provides ways to interact with platform-specific features when needed. This is typically achieved using platform-specific libraries or APIs. Developers can use these libraries through Java's native interface mechanisms (e.g., Java Native Interface - JNI) when they need to access hardware or OS-specific functions.

5. **Security**: Java's platform independence also contributes to its security model. Since Java bytecode runs within the JVM, it can be isolated from the underlying system, making it difficult for malicious code to interfere with the host system. This has made Java a popular choice for secure applications, such as web applets and mobile applications.

In summary, Java's platform independence is primarily due to its bytecode compilation and execution through the JVM. This architecture allows Java applications to run on various platforms without modification, making it a versatile and widely-used programming language for cross-platform development.

Q1 (b).

In Java, primitive data types represent simple, built-in data types that are used to store values directly in memory. Here's a list of primitive data types in Java:

1. **byte**: This is a 1-byte signed integer type that can store values in the range of -128 to 127.

2. **short**: This is a 2-byte signed integer type that can store values in the range of -32,768 to 32,767.

3. **int**: This is a 4-byte signed integer type that can store values in the range of -2^31 to 2^31 - 1.

4. **long**: This is an 8-byte signed integer type that can store values in the range of -2^63 to 2^63 - 1.

5. **float**: This is a 4-byte floating-point type that can store decimal numbers with single-precision.

6. **double**: This is an 8-byte floating-point type that can store decimal numbers with double-precision.

7. **char**: This is a 2-byte Unicode character type that represents a single character.

8. **boolean**: This represents a Boolean value, which can be either `true` or `false`.

Now, let's discuss Integral Promotion rules in Java. Integral promotion is the process by which smaller integer types are automatically promoted to larger integer types when they are involved in expressions. This helps in avoiding loss of data or precision in calculations. Here are the key rules:

1. **Smaller Types Promote to int**: If an expression involves smaller integer types such as `byte` or `short`, these values are automatically promoted to `int` before the operation is performed. For example:

```
byte a = 10;
byte b = 20;
int result = a + b; // Both 'a' and 'b' are promoted to int before addition.
```

2. **Unary Operators**: Unary operators like `++`, `--`, and `~` do not perform integral promotion. If you increment or decrement a `byte` or `short`, the result will still be of the same type.

```
byte x = 5;
x++; // No integral promotion, 'x' remains a byte.
```

3. **Binary Operators**: Binary operators like `+`, `-`, `*`, `/`, and `%` promote operands to at least `int` or `long`, depending on the type of operation. If both operands are `int`, the

result is also `int`. If one operand is `long`, the other is promoted to `long`, and the result is `long`.

```
  byte a = 10;
   short b = 20;
   int result = a + b; // Both 'a' and 'b' are promoted to int before addition.
```

4. **Casting**: If you need the result of an expression to be a smaller type, you can explicitly cast it. However, be careful of potential data loss or overflow.

```
   int bigValue = 1000;
   byte smallValue = (byte) bigValue; // Explicit casting to byte (potential data
loss).
```

Integral promotion ensures that arithmetic operations with smaller integer types do not lose precision, and it helps prevent unexpected behavior in Java programs.

Q2.

(a).

```
class overloadAdd{
        public static int add(int x, int y){
                return x+y;
        }

        public static double add(double x, double y){
                return x+y;
        }

        public static String add(String x , String y){
                return x+y;
        }

        public static void main(String[] main){
                int intResult = add(5,10);
                double doubleResult = add(5.0,10.0);
                String stringResult = add("Hello","World");

                System.out.println("Int Result "+intResult);
                System.out.println("double result "+doubleResult);
                System.out.println("String Result "+stringResult);
        }
}
```

OR

```
class complexNumbers{

        public static class complex{

                private int real,imaginary;
                public complex(){
                        System.out.println("Complex Numbers");
                }

                public complex(int real, int imaginary){
                        this.real=real;
                        this.imaginary=imaginary;
```

```
                }

                public complex(complex C){
                        this.real=C.real;
                        this.imaginary=C.imaginary;
                        System.out.println("COPY");
                }

        }

        public static void main(String[] args){
                complex C1= new complex();
                complex C2= new complex(3,4);
                complex C3= new complex(C2);

                System.out.println("Complex Number" + C2.real+"
"+C2.imaginary+"i");
        }
}
```

(b).

1. 0,0
2. 20
3. Error ; method overloading not possible with different return type
4. BOL
   Computer
5. Error; array not initialized.
6. Error; Main function outside scope of File, filename should be Main.java

(c).

In Java, both `new` and `this` are important keywords used in different contexts.

1. **`new` Keyword**:

The `new` keyword is used for object creation. It is followed by a constructor call to create an instance of a class. Here's an example:

```
class MyClass {
      int x;

      MyClass(int value) { // Constructor
          x = value;
      }
  }

  public class Main {
      public static void main(String[] args) {
          MyClass obj = new MyClass(42); // Using 'new' to create an
instance of MyClass
          System.out.println(obj.x); // Accessing the instance variable 'x'
of obj
      }
  }
```

In this example, the `new` keyword is used to create an instance of the `MyClass` class by calling its constructor `MyClass(42)`. The `obj` variable holds the reference to the newly created object.

2. **`this` Keyword**:

The `this` keyword is used within a class to refer to the current instance of the class. It is often used to reduce ambiguity between instance variables and parameters or to call another constructor within the same class. Here are examples of its usage:

- Reducing Ambiguity between instance variables and parameters:

```
class MyClass {
       int x;

       MyClass(int x) { // Constructor with a parameter
           this.x = x; // Using 'this' to refer to the instance variable
'x'
       }
    }
```

- Calling another constructor within the same class:

```
class MyClass {
       int x;

       MyClass() { // Default constructor
           this(0); // Calling another constructor with 'this'
       }

       MyClass(int x) { // Constructor with a parameter
           this.x = x; // Using 'this' to set the instance variable 'x'
       }
    }
```

In both cases, `this` helps distinguish between the instance variable `x` and the constructor parameter `x`.

The usage of `new` and `this` is crucial for object creation and maintaining clarity and correctness in your Java code.

Q3.

(a). ABSTRACT FUNCTION NOT IN SYLLABUS FOR MIDSEM

OR

INTERFACE NOT IN SYLLABUS FOR MIDSEM

(b)

1. Error; variable s called without using super() in constructor of child class B
2. Class A
   Class B
   Class C

3. Test class Derived class
4. 10

(c)

In Java, the `super` and `final` keywords serve different purposes:

1. `super` Keyword:
- The `super` keyword is primarily used to access members (fields or methods) of the superclass (parent class) in the context of a subclass (child class).
- It is often used in the following ways:
- To call a superclass constructor: You can use `super()` in a subclass constructor to call a constructor of the superclass. This is typically used to initialize inherited fields or perform additional setup in the superclass.
- To access superclass methods or fields: You can use `super.methodName()` or `super.fieldName` to explicitly access a method or field from the superclass, even if the subclass has overridden or hidden them.

Example of calling a superclass constructor:

```
class Parent {
      Parent() {
          System.out.println("Parent constructor");
      }
  }

  class Child extends Parent {
      Child() {
          super(); // Calls the Parent constructor
          System.out.println("Child constructor");
      }
  }
```

2. `final` Keyword:
- The `final` keyword is used to declare that an entity (class, method, or variable) cannot be changed or overridden after its initial declaration.
- Its usage varies depending on the context:
- Final variable: A final variable's value cannot be modified once it is assigned. It is often used for constants.
- Final method: A final method in a class cannot be overridden by subclasses, providing a level of method immutability.
- Final class: A final class cannot be extended by other classes, ensuring that it cannot serve as a superclass.

Example of a final variable:

```
final int MAX_VALUE = 100;
  // MAX_VALUE cannot be changed once assigned
```

Example of a final method:

```
class Parent {
```

```
    final void display() {
        System.out.println("Parent's display method");
    }
}

class Child extends Parent {
    // Cannot override the final method display
    // Attempting to do so would result in a compilation error
}
```

Example of a final class:

```
final class FinalClass {
    // Class body
}

// Cannot extend FinalClass with a subclass
```

In summary, `super` is used for accessing superclass members and constructors, while `final` is used for indicating immutability or preventing further extension or overriding of methods and classes.