# JAVA ASSIGNMENT 03

**Q1.Write a simple Banking System program by using OOPs concept where you can get account Holder name balance etc?**

**A)**

```
import java.util.Scanner;

class BankAccount {
    private String accountHolderName;
    private double balance;

    public BankAccount(String accountHolderName, double initialBalance) {
        this.accountHolderName = accountHolderName;
        this.balance = initialBalance;
    }

    public void deposit(double amount) {
        balance += amount;
        System.out.println("Amount deposited successfully. Current balance: " +
balance);
    }

    public void withdraw(double amount) {
        if (balance >= amount) {
            balance -= amount;
            System.out.println("Amount withdrawn successfully. Current balance: " +
balance);
        } else {
            System.out.println("Insufficient balance. Withdrawal failed.");
        }
    }

    public void checkBalance() {
        System.out.println("Account Holder Name: " + accountHolderName);
        System.out.println("Current balance: " + balance);
    }
}

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

```
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter Account Holder Name: ");
        String accountHolderName = scanner.nextLine();

        System.out.print("Enter Initial Balance: ");
        double initialBalance = scanner.nextDouble();

        BankAccount account = new BankAccount(accountHolderName,
initialBalance);

        System.out.println("----- Banking System -----");
        System.out.println("1. Deposit");
        System.out.println("2. Withdraw");
        System.out.println("3. Check Balance");
        System.out.println("0. Exit");

        int choice;
        do {
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter amount to deposit: ");
                    double depositAmount = scanner.nextDouble();
                    account.deposit(depositAmount);
                    break;
                case 2:
                    System.out.print("Enter amount to withdraw: ");
                    double withdrawAmount = scanner.nextDouble();
                    account.withdraw(withdrawAmount);
                    break;
                case 3:
                    account.checkBalance();
                    break;
                case 0:
                    System.out.println("Exiting the Banking System...");
                    break;
                default:
```

```
                System.out.println("Invalid choice. Please try again.");
            }
        } while (choice != 0);

        scanner.close();
    }
}
```

**Q2. Write a Program where you inherit method from parent class and show method Overridden Concept?**

**A)**

```
class Vehicle
{
    public void start()
{
        System.out.println("Vehicle started.");
    }
}

class Car extends Vehicle
{
    @Override
    public void start()
 {
        System.out.println("Car started.");
    }
}

public class InheritanceExample {
    public static void main(String[] args) {
        Vehicle vehicle = new Vehicle();
        vehicle.start(); // Output: Vehicle started.

        Car car = new Car();
        car.start(); // Output: Car started.

        // Upcasting
        Vehicle vehicle2 = new Car();
        vehicle2.start(); // Output: Car started.
    }}
```

**Q3.Write a program to show run time polymorphism in java?**

**A)**
```
class Animal {
    public void makeSound() {
        System.out.println("Animal is making a sound.");
    }
}

class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Dog is barking.");
    }
}

class Cat extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Cat is meowing.");
    }
}

public class PolymorphismExample {
    public static void main(String[] args) {
        Animal animal1 = new Animal();
        Animal animal2 = new Dog();
        Animal animal3 = new Cat();

        animal1.makeSound(); // Output: Animal is making a sound.
        animal2.makeSound(); // Output: Dog is barking.
        animal3.makeSound(); // Output: Cat is meowing.
    }
}
```

**Q4.Write a program to show Compile time polymorphism in java?**

**A)**
```
public class PolymorphismExample {
    public static int add(int a, int b) {
        return a + b;
    }

    public static double add(double a, double b) {
```

```
        return a + b;
    }

    public static String add(String a, String b) {
        return a + b;
    }

    public static void main(String[] args) {
        int sum1 = add(5, 10);
        double sum2 = add(3.14, 2.56);
        String sum3 = add("Hello", " World");

        System.out.println("Sum 1: " + sum1); // Output: Sum 1: 15
        System.out.println("Sum 2: " + sum2); // Output: Sum 2: 5.7
        System.out.println("Sum 3: " + sum3); // Output: Sum 3: Hello World
    }
}
```

**Q5. Achieve loose coupling in java by using OOPs concept?**

**A)**Loose coupling is a design principle in object-oriented programming that promotes the independence and flexibility of components. It aims to minimize dependencies between classes or modules, allowing them to be easily modified, replaced, or reused without affecting other parts of the system.

Here are some OOPs concepts and techniques that can help achieve loose coupling in Java:

1)Encapsulation
2)Abstraction
3)Dependency Injection (DI)
4)Interface-based Programming
5)Composition over Inheritance

**Q6. What is the benefit of encapsulation in java?**

**A)**Encapsulation is a fundamental principle of object-oriented programming that combines data and methods within a class and provides control over their access.

THERE ARE SOME BENEFITS:

1)Data Hiding
2)Encapsulation Protects the Internal Implementation

3)Modularity and Code Organization
4)Flexibility and Evolvability
5)Information Hiding and Security

**Q7.Is java a t 100% Object oriented Programming language? If no why ?**
**A)**No, Java is not considered a 100% object-oriented programming language. While Java is primarily an object-oriented language and follows many object-oriented principles, it also incorporates some non-object-oriented features. Here are a few reasons why Java is not considered a fully pure object-oriented programming language:

- **Primitive Data Types:** Java includes primitive data types such as int, float, boolean, etc., which are not objects. These primitive types do not have associated methods or inheritance hierarchies like objects do. However, Java provides wrapper classes (e.g., Integer, Float, Boolean) to wrap these primitive types into objects, allowing them to be used in an object-oriented manner.

- **Static Members:** Java allows the declaration of static variables and static methods. Static members are associated with the class itself rather than individual instances. They can be accessed without creating an instance of the class, which goes against the object-oriented principle of encapsulating behavior within objects.

- **Procedural Programming:** Java supports procedural programming paradigms alongside object-oriented programming. It allows the use of procedural constructs like loops, conditional statements, and procedural methods that are not tied to objects or classes. These features make Java a multi-paradigm language, supporting both procedural and object-oriented programming styles.

- **Lack of Multiple Inheritance:** Java does not support multiple inheritance, where a class can inherit from multiple parent classes. This limitation is imposed to avoid complexities and ambiguities that can arise due to conflicts between inherited methods and state from multiple sources. Instead, Java allows for multiple interface implementation, which provides a form of multiple inheritance for defining contracts.

**Q8.What are the advantages of abstraction in java?**

**A)Abstraction is a fundamental concept in object-oriented programming that allows you to represent the essential features of an object while hiding unnecessary details.**
**In detail.,**
1)Simplifies Complexity
2)Provides a High-Level View
3)Promotes Code Reusability
4)Supports Polymorphism
5)Enhances Modularity
6)Facilitates Future Enhancements:

**Q9.What is an abstraction explained with an Example?**
**A)**Abstraction is a concept in object-oriented programming that allows you to represent the essential features of an object while hiding unnecessary details. It focuses on what an object does rather than how it does it.

Example:

```
abstract class Animal {
    private String name;

    public Animal(String name) {
        this.name = name;
    }

    public abstract void makeSound();

    public void sleep() {
        System.out.println(name + " is sleeping.");
    }
}

class Dog extends Animal {
    public Dog(String name) {
        super(name);
    }

    @Override
    public void makeSound() {
        System.out.println("Dog is barking.");
    }
```

```
}

class Cat extends Animal {
    public Cat(String name) {
        super(name);
    }

    @Override
    public void makeSound() {
        System.out.println("Cat is meowing.");
    }
}

public class AbstractionExample {
    public static void main(String[] args) {
        Animal dog = new Dog("Tommy");
        dog.makeSound(); // Output: Dog is barking.
        dog.sleep(); // Output: Tommy is sleeping.

        Animal cat = new Cat("Kitty");
        cat.makeSound(); // Output: Cat is meowing.
        cat.sleep(); // Output: Kitty is sleeping.
    }
}
```

**Q10.What is the final class in Java?**

**A)**

**Final keyword: It is mainly takes place for variables ,methods and classes.**

When we mention the final keyword which makes it constant, it can't be changed until the execution.
When mention final keyword it makes constant can't be changed until execution.
When final keyword to methods it prevents mentor overriden.
When we mention final keyword to class it prevents from subclasses.