Assignment Questions 3

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

```
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("Deposit of $" + amount + " successful.");
    }

    public void withdraw(double amount) {
        if (balance >= amount) {
            balance -= amount;
```

```
            System.out.println("Withdrawal of $" + amount + "
successful.");
        } else {
            System.out.println("Insufficient funds. Withdrawal
failed.");
        }
    }

    public void displayAccountInfo() {
        System.out.println("Account Holder Name: " +
accountHolderName);
        System.out.println("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("\n-- Banking System Menu --");
System.out.println("1. Deposit");
System.out.println("2. Withdraw");
System.out.println("3. Display Account Info");
System.out.println("4. Exit");

int choice;
do {
    System.out.print("\nEnter your choice (1-4): ");
    choice = scanner.nextInt();

    switch (choice) {
        case 1:
            System.out.print("Enter deposit amount: $");
            double depositAmount = scanner.nextDouble();
            account.deposit(depositAmount);
            break;
        case 2:
            System.out.print("Enter withdrawal amount: $");
            double withdrawalAmount = scanner.nextDouble();
            account.withdraw(withdrawalAmount);
            break;
        case 3:
            account.displayAccountInfo();
            break;
        case 4:
```

```
                System.out.println("Thank you for using the Banking
System. Goodbye!");
                break;
            default:
                System.out.println("Invalid choice. Please enter a
valid option (1-4).");
                break;
        }
    } while (choice != 4);

    scanner.close();
  }
}
```

Output:

Let's consider an example scenario where the user enters the following inputs and performs various operations in the Banking System program:

Enter Account Holder Name: Vinay kumar
Enter Initial Balance: $1000

After entering these inputs, the program displays the menu:

-- Banking System Menu --
1. Deposit
2. Withdraw
3. Display Account Info
4. Exit

Now, let's simulate some interactions with the program:

Deposit: User chooses option 1 and enters the deposit amount as $500

Enter your choice (1-4): 1
Enter deposit amount: $500
Deposit of $500.0 successful.

Withdraw: User chooses option 2 and enters the withdrawal amount as $200.

Enter your choice (1-4): 2
Enter withdrawal amount: $200
Withdrawal of $200.0 successful.

Display Account Info: User chooses option 3 to display the account information.

Enter your choice (1-4): 3

Account Holder Name: vinay kumar
Balance: $1300.0

Exit: User chooses option 4 to exit the program.

Enter your choice (1-4): 4

Thank you for using the Banking System. Goodbye!

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

```
class ParentClass {
    public void showMessage() {
        System.out.println("This is the parent class.");
    }
}

class ChildClass extends ParentClass {
    @Override
    public void showMessage() {
        System.out.println("This is the child class.");
    }
}

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

```
        // Creating objects of the classes
        ParentClass parent = new ParentClass();
        ChildClass child = new ChildClass();

        // Calling the showMessage() method on the parent
object
        parent.showMessage();  // Output: This is the parent
class.

        // Calling the showMessage() method on the child object
        child.showMessage();  // Output: This is the child class.
    }
}
```

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

```
class Animal {
  public void makeSound() {
    System.out.println("The animal makes a sound");
  }
}

class Dog extends Animal {
  public void makeSound() {
    System.out.println("The dog barks");
  }
```

```
}

public class Main {
  public static void main(String[] args) {
    Animal myAnimal = new Dog();
    myAnimal.makeSound();
  }
}
```

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

```
class Addition {
  public int add(int num1, int num2) {
    return num1 + num2;
  }
  public int add(int num1, int num2, int num3) {
    return num1 + num2 + num3;
  }
}

public class Main {
  public static void main(String[] args) {
    Addition obj = new Addition();
    System.out.println("Sum of two integers: " + obj.add(10,
20));
```

```
 }
}
```

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

Loose coupling is a situation when classes or modules have minimal dependencies on each other , changes in one class or module are unlikely to affect the other. This makes our code modular and reduces the risk of introducing bugs. This makes our code easier to maintain and extend over time.

In Java, we can achieve loose coupling using OOPs concepts such as abstraction, encapsulation, inheritance, and polymorphism.

Abstraction: Abstraction is the process of hiding implementation details while showing only the necessary information to the user. It helps in reducing complexity and achieving loose coupling.

Encapsulation: Encapsulation is the process of wrapping data and code that operates on data into a single unit. It helps in achieving loose coupling by hiding implementation details.

Inheritance: Inheritance is the process of creating a new class from an existing class. It helps in achieving loose coupling by

reusing code.Polymorphism: Polymorphism is the ability of an object to take on many forms.

Q6. What is the benefit of encapsulation in java?

Encapsulation is one of the four fundamental OOP concepts. It is the process of wrapping data and code that operates on data into a single unit. Encapsulation helps in achieving loose coupling by hiding implementation details. It also provides several benefits such as:

Data Hiding: Encapsulation allows us to hide the implementation details of a class from other classes. This helps in preventing accidental modification of data.

Modularity: Encapsulation helps in creating modular code. It allows us to break down a large program into smaller, more manageable modules.

Flexibility: Encapsulation provides flexibility by allowing us to change the implementation details of a class without affecting other parts of the program.

Code Reusability: Encapsulation promotes code reusability by allowing us to reuse code without worrying about its implementation details.

Testing and Debugging: Encapsulation makes testing and debugging easier by providing a clear separation between the interface and implementation of a class.

Q7.Is java a t 100% Object oriented Programming language? If no why ?

Java is often considered to be a pure object-oriented programming language because it supports all the features of OOP such as inheritance, encapsulation, abstraction, and polymorphism. However, Java does not support some of the features that are required for a language to be considered 100% object-oriented. These features include:

Multiple Inheritance: Java does not support multiple inheritance. This means that a class cannot inherit from more than one class at a time.

Operator Overloading: Java does not support operator overloading. This means that we cannot overload operators such as +, -, *, /, etc.

Pointers: Java does not support pointers. This means that we cannot use pointers to manipulate memory directly.

Despite these limitations, Java is still considered to be one of the most popular and widely used programming languages in the world.

Q8.What are the advantages of abstraction in java?

Advantage:

- **Reduced Complexity:** Abstraction helps in reducing complexity by hiding implementation details from the user. This makes it easier for the user to understand and use the code.

Flexibility: Abstraction provides flexibility by allowing us to change the implementation details of a class without affecting other parts of the program.

Code Reusability: Abstraction promotes code reusability by allowing us to reuse code without worrying about its implementation details.

Modularity: Abstraction helps in creating modular code. It allows us to break down a large program into smaller, more manageable modules.

Security: Abstraction helps in improving security by hiding sensitive information from the user.

Q9.What is an abstraction explained with an Example?

Abstraction is the process of hiding implementation details while showing only the necessary information to the user. It helps in reducing complexity and achieving loose coupling.

Example of abstraction in Java:

```
abstract class Shape {
   abstract void draw();
}

class Rectangle extends Shape {
   void draw() {
      System.out.println("Drawing Rectangle");
   }
}

class Circle extends Shape {
   void draw() {
      System.out.println("Drawing Circle");
   }
}

public class vinaykumar {
   public static void main(String[] args) {
      Shape s = new Circle();
      s.draw();
   }
}
```

Q10.What is the final class in Java?

In Java, the final keyword is used to restrict the user from modifying the value of a variable, method, or class. When we declare a class as final, it means that the class cannot be extended by any other class.

Example:

```
final class MyClass {
   // Class implementation
}

class MySubClass extends MyClass {
   // This will result in a compile-time error
}
```

In this example, we have declared the MyClass as final. This means that no other class can extend this class. We have also declared a subclass called MySubClass that tries to extend the MyClass. This will result in a compile-time error because we cannot extend a final class.

Here's an example program that demonstrates the use of the final keyword with a method:

```
class MyClass {
   final void myMethod() {
      // Method implementation
```

```
    }
}

class MySubClass extends MyClass {
    void myMethod() {
        // This will result in a compile-time error
    }
}
```

In this example, we have declared the myMethod() as final. This means that no other subclass can override this method. We have also declared a subclass called MySubClass that tries to override the myMethod(). This will result in a compile-time error because we cannot override a final method.