Q1. Define what an object is in the context of object-oriented programming. Explain how an object represents an instance of a class and its importance in OOP.

Ans: In OOP, an object is an instance of a class that encapsulates data (attributes) and behaviour (methods). Objects represent real world entities and are key in OOP for interacting with data and functionality.

Q2. Discuss the key characteristics of objects, including their state, behaviour and identity. How do these characteristics contribute to the design and functionality of a software system?

Ans: State: The state of an object refers to its data or attributes at a particular moment.

Behaviour: Behaviour defines what an object can do and how it can interact with other objects.

Identity: Identity is what distinguishes one object from another, even

if they have the same state and behaviour. Each object have unique address of reference to memory

→ contribution to software design and functionality

- State allows objects to hold information relevant to their role.
- Behaviour enable objects to perform actions and interact with other objects.
- Identity ensures that objects are distinct and can be uniquely referenced, which is crucial for managing object lifecycle and relationships.

Q3) Explain the process of ~~in~~ creating and initiating objects from a class. what role do constructor play in the process?

Ans 1) creating a class

A class is a blueprint for creating objects. It defines the attributes and methods that the objects will have.

2) Instantiating an object

Instantiation is the process of creating an object from a class using the new keyword.

e.g.

3)

Q4)

Ans

e.g. public class car{
    String mode;
    String color;
    void start(){
        System.out.println("car started");
    }
}
car mycar = new car();

3) Role of constructor
A constructor is a special method used to initialize an object. It is called automatically when an object is created.

Q4) Provide an example of a class with attributes and methods. Show how changes to an object's state affect its behaviour.

Ans: class definition:
public class BankAccount{
    double balance;
    // constructor
    public BankA(double initialB){
        this.balance = initialB;
    }
    // method to deposit money
    public void deposit(double amount){
        balance += amount;
    }
}

2) object instantiation and state change

```
public class main {
    public static void main (String[] args){
        BankAccount account = new
                              BankAccount (1000.0);
        account.deposit (500.0);
    }
}
```

Here, depositing money changes the object's balance attribute (state) which in turn affects its behaviour as seen in the updated balance after each operation.

Q5. Define encapsulation and discuss its significance in object.

Ans: Encapsulation is the principle of bundling data (attributes) and method (functions) that operate on the data into a single unit or class. It also involves restricting direct access to some of the objects components, typically by making attributes private, and providing public setter & getter methods.

Q4.

Ans:

02

Q4. Design a class Employee with attributes to employee ID, name and salary. Include methods for updating the salary, displaying employee details, and calculating annual salary.

Ans:
```
class Employee{
    private int id;
    private string name;
    private double salary;
    public Employee( int id, string name,
        double salary){
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
    public void updateSalary(double salary){
        this.salary = salary;
    }
    public void displayDetails(){
        System.out.println(id + " " +
        name + " " + salary);}
    }
    public double annualSalary(){
        return salary * 12;
    }
}
```

Q2. Design a class Department that manages a collection of Employee objects. Include methods to add employees,

05. Explain how access modifiers (eg. private, protected, public) are used to implement encapsulation. Provide an example illustrating these concepts.

Ans

```
class Student{
    private int id;
    protected string name;
    public double gradie;
    public int getId(){
        return id;
    }
    public void setId(int id){
        this.id = id;
    }
}
```

Q1. Define what a class is in the context of object-oriented programming. Explain the role of a class in designing and structing a software application.

Ans: A class is a blueprint or template for creating objects in object-oriented programming (OOP). It defines properties (attributes) and behaviours (methods) that the objects created from the class can have.

→ Role in software design:

- Encapsulation: Bundles data and methods that operate on the data within one unit.
- Abstraction: Represents essential features without including complex background details.
- Reusability: classes can be reused accross different parts of a program or in different projects.

Q2. Explain the process of creating and initializing a class. What are constructors, and how do they facilitate the instantiation of objects?

Ans: Creating a class:

```
class Person{
    string name;
    int age;
}
```

=> constructor

Role:

- Automatically called when an object is created.
- Ensures that objects are initialized with valid data.

```
class person{
    string name;
    int age;
    public person(string name, int age){
        this.name = name;
        this.age = age;
    }
}
```

Q3) Define encapsulation and data hiding. How do these principles enhance the security and integrity of the class's data?

Ans: Encapsulation - It is the concept of wrapping data (attributes) and methods(functions) into a single unit, i.e., a class. It restricts direct access to some components, which is known as data hiding.

Data Hiding - It involves using access modifiers like private to restrict access to class members. It prevents external classes from modifying internal state directly, ensuring data integrity and protecting the internal implementation.

Q4) Explain the concepts of inheritance and composition in relation to classes. How do these mechanisms support code reuse and the creation of complex systems?

Ans: Inheritance - Inheritance allows a new class (subclass) to inherit properties and behaviours from an existing class (super class). Common code is placed in the superclass and reused in subclass.

composition - composition involves building complex classes using objects of other classes. Existing classes can be used as components in new classes.

Q1. Design a class Book with attributes for title, author and publication year. Include methods to set and get the book's details, and a method to display the book information.

Ans:
```
class Book{
    private string title;
    private string author;
    private int publicationYear;

    public void setDetails(String title,
        String author, int year){
        this.title = title;
        this.author = author;
        this.publicationYear = year;
    }

    public String getDetails(){
        return title+", "+author+", "+
            publicationYear;
    }

    public void displayInfo(){
        System.out.println("Title: "+
        title+"\nAuthor:"+author);
    }
}
```

Q2. Provide examples illustrating inheritance and composition. Describe how these concepts allow for extending or combining class functionalities.

Ans: Inheritance Example

```
class vehicle {
    public void start(){
        System.out.println("vehicle
            started");
    }
class car extends vehicle{
    public void driver(){
        System.out.println("
            car driving");
    }
}
```

composition Example:

```
class Engine{
    public void ignite(){
        System.out.println("Engine
            ignited");
    }
class car{
    privae Engine engine = new
                Engine();
    public void startCar(){
        engine.ignite();
        System.out.println("car
            started");
    }
}
```

03. Discuss the use of access modifiers (e.g. privae, protected, public) ~~the~~ in

encapsulation. Provide examples of how different access levels affect data visibility and manipulation.

Ans: Private: Visible only within class. Used for sensitive data.

Protected: Visible within the class, package and subclasses.

Public: Visible from any class

```
class Account{
    private double balance;
    protected void setBalance(double
        Balance){
            this.balance = Balance;
    }
    public double getBalance(){
        return balance;
    }
}
```

Q4) Provide an example of a class with a constructor. Describe how the constructor initializes an object's state and how this process contributes to object creation.

Ans.
```
class student{
    private String name;
    private int rollnum;
    public student(String name,
                   int rollnum){
```

}

Explan
→ const
roll
roll
stud

→ Initi
obje
stat
the
inst

```
        this.name = name;
        this.rollnum = rollnum;
    }
    public void displayInfo(){
        System.out.println("name: "+
        name + "\nRoll num: " +
                rollnum);
    }
}
```

Explanation:

→ constructor: student (String name, int rollnum) initializes the name and rollnum attributes when creating a student object.

→ Initialization - Ensures every student object starts with a defined state with a defined state, making the object ready for use immediately upon creation.

01. Define what methods are in the context of object-oriented programming. Explain how methods define the behaviour of objects and how they are used to interact with an object's state.

Ans: Methods are functions defined inside a class that describe the behaviour of objects. They operate on an object's attributes and define what actions an object can perform. Methods allow interaction with object's state, enabling tasks like ~~modify~~ modifying data, performing calculations or generating output.

Role in behaviour:

- Define Actions - Specify what an object can do.
- State Interaction - Access or modify attributes.
- Encapsulation - Hide the internal workings of the objects exposing only necessary behaviors.

02) Explain the concept of messages in OOP. How do messages facilitate communication between objects?

Ans:

Ans: In OOP, messages refer to the calls made to methods of an objects. Sending a message to an object means invoking a method on that object.

e.g.
```
class lamp{
    private boolean isOn = false;
    public void turnOn(){
        isOn = true;
        System.out.println("
            lamp is on");
    }
}
public class main {
    public static void main(String[]
        args){
        lamp newlamp = new lamp();
        way
        newlamp.turnOn();
    }
}
```

03) Define abstraction in object-oriented programming. How does abstraction help in ~~mag~~ managing complexity by focusing on high-level functionalities and hiding implementation details?

Ans: Abstraction is the concept of hiding the complex implementation details of a system and exposing only

the essential features. It focuses on high-level functionalities, making it easier to manage complexity by providing a simplified interface to interact with.

Benefits:

- simplifies use - Users interact with simple interfaces without needing to understand internal workings.
- Reduces complexity - By showing only necessary details, it helps manage larger systems more efficiently.
- Improved modularity - Implementation can be changed without affecting the external interface.

Q4) Provide an example of a class that uses abstraction, describe how the class abstracts away implementation details and exposes only necessary functionality through its methods.

Ans:

```
abstract class Shape{
    abstract double area();
    public void displayArea(){
        System.out.println("Area: "+
            area());
    }
}
```

class

}

Expla

- A
- d
- .

```
class circle extends shape{
        private double radius;
        public circle( double radius){
            this.radius = radius;
        }
        double area(){
            return Math.PI* radius*
                    radius;
        }
}
```

Explanation:

- Abstracts the concept of a shape by defining an abstract method area().
- Implements the area() method, providing specific details for calculating a circle's area.
- Users can use Shape class focus on the area() method without needing to know the actual formula used in the circle class.

Q1. Define encapsulation and discuss its significance in OOP. How does encapsulation contribute to data protection and controlled access to an object's internal state?

Ans: Encapsulation is the principle of bundling an object's data (attributes) and methods (functions) into a single ~~node~~ unit, i.e., a class, and restricting direct access to some of the object's components.

Significance:

- Data Protection: Prevents unauthorized access and modifications.
- Controlled Access - Provides methods (getters / setters) to access or modify the data, ensuring it remains valid.
- Code Modularity - Keeps data and methods that operate on it together.

Q2) Explain how access modifiers (e.g. private, protected, public) are used to implement encapsulation. Provide an example demonstrating how encapsulation is achieved through these access controls.

Ans: • Private - Only accessible within the same place.
• Protected - Accessible within the same package and subclass
• Public - Accessible from any class

```
class Student{
    private String name;
    public String getName(){
        return name;
    }
    public void setName(String name){
        if(!name.isEmpty()){
            this.name = name;
        }
    }
}
```

03) compare and contrast abstraction & encapsulation. How do these concepts differ, and how do they complement each other in OOP?

Ans

Abstraction	Encapsulation
• Hides complex implementation details and shows only the necessary features.	• Hides the internal state and implementation details of an object and protects its data.
• Simplifies the interface for the user.	• Ensures controlled access and data protection.

- focus on "what" an object does.
- An abstract class or interface providing a general way to interact with objects.

- focuses on "how" the objects hides its data and internal details
- using private attributes with public getter / setter methods.

Q4) Discuss how both abstraction and encapsulation contribute to the design of a robust and maintainable object-oriented system. explain in short.

Ans: **Abstraction:**

- Simplified complexity: By exposing only necessary details, it reduces the complexity users need to handle.
- Modularity: Allows the creation of reusable and interchangeable components.

Encapsulation:

- Data Integrity - Protects data from being altered in unexpected ways.
- code maintainability - Encapsulated

classes can be modified internally without affecting external code.

Combined Impact:

- Encapsulation - enforces strict control over the data and behaviour of objects.
- Abstraction provides clear, simplified interfaces to interact with complex systems. Both principles create a system that is easier to understand, maintain, and extend, leading to more robust and maintainable software design.