OOPs Question Set

Context: eCommerce Project (Product, Order, Payment, Customer Modules)

1. Core Concept Questions

Question	Model Answer
What is Object-Oriented Programming?	OOP is a programming paradigm based on real-world entities like objects, which combine data and behavior. It helps structure code into reusable, modular components.
What are the four pillars of OOP?	Encapsulation, Inheritance, Polymorphism, and Abstraction. These principles make software modular, maintainable, and secure.
What is a Class and what is an Object?	A class is a blueprint that defines properties and behaviors. An object is a real instance created from that blueprint. For example, Product is a class, and "Wireless Mouse" is an object.
What is a Constructor?	A constructor is a special method automatically invoked when an object is created. It initializes object data.
What is the use of the super() keyword?	It's used by a child class to call the parent class constructor or method. For example, in ElectronicProduct, super().__init__() calls the parent Product initializer.

2. Encapsulation

Question	Model Answer
Define Encapsulation.	Encapsulation means bundling data and methods into a single unit and restricting direct access to internal data.
How do you achieve Encapsulation?	By using private variables and public methods (getters/setters) to control access.
How did you use Encapsulation in your project?	In my eCommerce app, I made Customer details like name and email private to secure sensitive data, and provided get_email() to access it safely.
What is the benefit of Encapsulation?	It protects the integrity of data and prevents unauthorized access or modification.

Code Example (Encapsulation):

```
class Customer:
    def __init__(self, name, email):
        self.__name = name
        self.__email = email

    def get_email(self):
        return self.__email
```

3. Inheritance

Question	Model Answer
Define Inheritance.	Inheritance allows a child class to use and extend the functionality of a parent class.
How did you apply Inheritance in your project?	In my eCommerce project, ElectronicProduct inherited from the parent Product class to reuse name and price attributes while adding warranty.
Why do we use the super() keyword in Inheritance?	It ensures the parent constructor runs, initializing inherited variables before child-specific attributes are added.
What is Multiple Inheritance?	It means a class can inherit from more than one class. Python supports it; Java does not (but achieves it via interfaces).

Code Example (Inheritance):

```
class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price

class ElectronicProduct(Product):
    def __init__(self, name, price, warranty):
        super().__init__(name, price)
        self.warranty = warranty
```

4. Polymorphism

Question	Model Answer
What is Polymorphism?	It means “many forms.” The same method name behaves differently based on the object type.
Difference between Overloading and Overriding.	Overloading: Same method name, different parameters (compile-time). Overriding: Same method in parent and child, different implementation (runtime).
How did you implement Polymorphism in your project?	My eCommerce app used a Payment parent class with overridden pay() methods in CardPayment and UpiPayment subclasses.
Why is Polymorphism important?	It increases flexibility — lets different objects respond differently to the same function call.

Code Example (Polymorphism):

```
class Payment:
    def pay(self): pass

class CardPayment(Payment):
    def pay(self): return "Paid via Card"

class UpiPayment(Payment):
    def pay(self): return "Paid via UPI"
```

5. Abstraction

Question	Model Answer
What is Abstraction?	Hiding complex implementation details and showing only necessary features to the user.
How do you achieve Abstraction?	Using abstract classes and interfaces in Java or abstract base classes in Python.
How did you use Abstraction in your project?	I created an abstract Payment class that defined the pay() method, and subclasses like WalletPayment implemented it differently.
What is the benefit of Abstraction?	Simplifies development and testing by hiding complexity and reducing dependency between components.

Code Example (Abstraction):

```
from abc import ABC, abstractmethod
class Payment(ABC):
    @abstractmethod
    def pay(self): pass

class WalletPayment(Payment):
    def pay(self): return "Payment through wallet"
```

6. Real-World & SME Application Questions

Question	Model Answer
How do these OOPs concepts relate to your project architecture?	Each module in my eCommerce system follows OOP design: Product and Order classes are encapsulated, inheritance connects Product types, and abstraction is used in the Payment module.
What happens if you don't use OOP?	The code becomes repetitive, harder to maintain, and less scalable — changes in one part might break others.
Can you explain a scenario where you used super()?	In ElectronicProduct, super() called the Product constructor to initialize shared attributes like name and price.
Why is OOP preferred for enterprise projects like Cognizant's?	Because it supports modularity, reusability, and easy debugging — essential for scalable enterprise systems.

7. Trick & Confidence Questions

Question	Model Answer
Can constructors be overloaded in Python/Java?	In Java – yes (by changing parameters). In Python – indirectly, by using default arguments or *args.
What happens if we don't define a constructor?	A default constructor runs automatically, initializing the object with default values.
Can a class exist without objects?	Yes, but it's only a template until instantiated.
Can we create an object without using a class?	In OOP, no — class defines the structure for objects. (Except built-ins.)

8. Integration with Other Modules

Question	Model Answer
How did OOPs help you integrate your modules?	It made it easier to connect Customer, Product, and Order modules since each had its own class structure and defined methods.
How does OOP support security?	Through encapsulation and access modifiers that protect data from direct modification.
How did you test your OOP modules?	By creating unit tests for each class, ensuring each method returned the expected output (e.g., order total or payment status).

9. Quick Recap Table

Focus Area	Keywords	eCommerce Anchor	SME Intent
Encapsulation	Private, Secure	Customer Info	Data Protection
Inheritance	Reuse, Hierarchy	Product → ElectronicProduct	Code Reuse
Polymorphism	Flexibility	Payment: Card vs UPI	Dynamic Behavior
Abstraction	Simplicity	Payment Interface	Design Thinking
Class/Object	Blueprint/Instance	Product, Order	Fundamentals
Constructor	Initialization	Auto-setup	Code Flow
super()	Parent Access	Product Base Call	Logic Clarity