1. Introduction

- **Programming paradigm** in which a program is designed using **objects** and **classes**.
- Tries to model real-world entities (student, car, bank account) as objects having **data** and **functions** together.
- Common OOP languages: C++, Java, C#, Python, etc.

---

2. Basic Terms

2.1 Class

- A **class** is a *user-defined data type* or *blueprint* that defines **data members** (variables) and **member functions** (methods).
- Example idea:
  `class Student { roll, name; getData(); showData(); };`

2.2 Object

- An **object** is a **runtime instance** of a class.
- It occupies memory and can use the data and functions of its class.
- Example idea:
  `Student s1;` → `s1` is an object of class `Student`.

2.3 Data members and Member functions

- **Data members**: variables inside a class that store object's state (e.g. `roll`, `balance`).
- **Member functions**: functions defined inside a class that operate on its data (e.g. `deposit()`, `withdraw()`).

---

3. Main Features / Principles of OOP

(These are the standard NEB points: abstraction, encapsulation, inheritance, polymorphism, etc.)

3.1 Data Abstraction

- Process of **showing only essential features** of an object and hiding background details.
- Example: a `BankAccount` object shows operations like `deposit()` and `withdraw()`, but not the internal formula or file operations.

3.2 Encapsulation

- **Binding of data and functions** that operate on that data into a single unit (class).
- Data is protected from outside interference; it can only be accessed through member functions.
- Often implemented using **access specifiers** like `private`, `public`, `protected`.

3.3 Data Hiding

- Restricting direct access to internal data of an object.
- Only authorized member functions can read or modify data.
- In C++ this is achieved by keeping data members `private`.

3.4 Inheritance

- Mechanism by which a **new class (derived class)** is created from an **existing class (base class)**.
- Derived class **inherits** data and functions of base class and can add its own.
- Supports **code reuse** and hierarchical classification.
- Example: `class SavingAccount : public BankAccount { … };`

Types Of Inheritance

1. Single Inheritance

- A derived class has **only one** direct base class.

Example (idea):
```
class A { … };
class B : public A { … };
```

---

2. Multiple Inheritance

- A derived class has **two or more** base classes.

Example:
```
class A { … };
class B { … };
class C : public A, public B { … };
```

---

3. Multilevel Inheritance

- Inheritance occurs in **levels**: a class is derived from another derived class (chain).

Example:

```
class A { … };
class B : public A { … };
class C : public B { … };
```

---

4. Hierarchical Inheritance

- **One base class** is inherited by **two or more** derived classes.

Example:

```
class A { … };
class B : public A { … };
class C : public A { … };
```

---

5. Hybrid Inheritance

- **Combination** of two or more of the above types in a program (e.g. multilevel + multiple).
- Used to design complex relationships among classes.

3.5 Polymorphism

- Means "many forms".
- Ability to use **same name** for functions or operators but behave **differently** in different situations.
- Types:
  - **Compile-time polymorphism** – function overloading, operator overloading.
  - **Run-time polymorphism** – virtual functions, late binding.

3.6 Message Passing

- Objects **communicate** with each other by sending messages (calling member functions).
- Example: `acc1.transfer(acc2, 5000);` → object `acc1` sends a message to transfer money to `acc2`.

3.7 Dynamic Binding (Late Binding)

- The code to be executed in response to a function call is decided **at run time**, not at compile time.
- Achieved using **virtual functions** and supports run-time polymorphism.

---

4. Advantages of OOP (write any 4–6)

1. **Modularity** – program is divided into classes and objects; easier to understand and manage.
2. **Reusability** – classes and objects can be reused in other programs; inheritance promotes reuse.
3. **Maintainability** – changes in one part of the system have less effect on others.
4. **Security** – encapsulation and data hiding protect data from unauthorized access.
5. **Extensibility** – new features can be added by creating new classes or inheriting existing ones.
6. **Better mapping to real world** – objects represent real-world entities, making design natural.

---

5. OOP vs Procedure-Oriented Programming (POP)

(Short comparison for exam answers.)

Basis	Procedure-Oriented (C)	Object-Oriented (C++, Java)
Approach	Focus on **functions** and sequence of actions	Focus on **objects** containing data and functions
Data	Data is **global** and free; less protected	Data is **hidden** inside objects (encapsulation)
Reusability	Through functions only	Through classes, objects, inheritance
Security	Lower (global data can be accessed anywhere)	Higher (private/protected members, controlled access)
Suitable for	Simple, small or math-oriented problems	Large, complex, real-world modeling applications