1. Introduction to OOPs

Object-Oriented Programming (OOP) is a programming paradigm that organizes software design around **objects** rather than functions and logic. An object represents a real-world entity and contains both **data** and **functions** that operate on that data.

C++ is an object-oriented language that supports both **procedural programming** and **object-oriented programming**, making it very powerful and flexible.

---

2. Need for OOPs

Traditional procedural programming becomes difficult to manage for large and complex programs. OOPs helps to overcome these problems by:

- Improving code organization
- Reducing complexity
- Increasing reusability
- Making programs easier to understand and maintain

OOPs is especially useful in large software projects such as banking systems, games, operating systems, and web applications.

---

3. Basic Terminologies in OOPs

Object

An object is a real-world entity such as a car, student, or employee. In programming, an object is an instance of a class.

Class

A class is a blueprint or template for creating objects. It defines the data members and member functions.

Example

```
class Student {
  int roll;
  char name[20];
};
```

---

4. Features of OOPs

The main features of Object-Oriented Programming are:

1. Encapsulation

2. Abstraction
3. Inheritance
4. Polymorphism
5. Data Hiding
6. Message Passing

---

5. Encapsulation

Encapsulation is the process of **binding data and functions together** into a single unit called a class. It protects data from unauthorized access.

Advantages of Encapsulation

- Improves data security
- Enhances modularity
- Makes code easy to manage

Example

```
class Account {
  private:
    int balance;
  public:
    void setBalance(int b) {
      balance = b;
    }
};
```

---

6. Data Hiding

Data hiding is closely related to encapsulation. It restricts direct access to data members using **access specifiers**:

- private
- protected
- public

Data hiding ensures that sensitive data cannot be accessed directly from outside the class.

---

7. Abstraction

Abstraction means showing only essential features and hiding unnecessary details. It focuses on **what an object does**, not how it does it.

Example

When using a car, we only know how to drive it, not the internal engine mechanism.

In C++, abstraction is achieved using:

- Classes
- Access specifiers
- Abstract classes

---

8. Inheritance

Inheritance allows one class to acquire the properties of another class. The existing class is called the **base class**, and the new class is called the **derived class**.

Advantages

- Code reusability
- Reduced redundancy
- Easy maintenance

Example

```
class Animal {
  public:
    void eat() {}
};

class Dog : public Animal {
};
```

---

9. Types of Inheritance

1. **Single Inheritance**
2. **Multiple Inheritance**
3. **Multilevel Inheritance**
4. **Hierarchical Inheritance**
5. **Hybrid Inheritance**

Each type provides flexibility in code reuse.

---

10. Polymorphism

Polymorphism means **many forms**. It allows the same function name to behave differently based on context.

Types of Polymorphism

1. Compile-time Polymorphism
2. Run-time Polymorphism

---

11. Compile-Time Polymorphism

Also called **static polymorphism**, achieved using:

- Function Overloading
- Operator Overloading

Function Overloading Example

```
int add(int a, int b);
float add(float a, float b);
```

---

12. Run-Time Polymorphism

Also called **dynamic polymorphism**, achieved using:

- Function overriding
- Virtual functions

Example

```
class Base {
  public:
    virtual void show() {}
};
```

---

13. Constructors

A constructor is a special member function that is automatically called when an object is created. It has the same name as the class.

Types

- Default constructor
- Parameterized constructor
- Copy constructor

---

14. Destructor

A destructor is a special function that is called when an object is destroyed. It is used to release resources.

Syntax

```
~ClassName() {}
```

15. Access Specifiers

Access specifiers define the scope of class members:

- public – accessible everywhere
- private – accessible only inside class
- protected – accessible in derived classes

16. Message Passing

Objects communicate with each other by calling member functions. This is known as message passing.

Example

```
obj.function();
```

17. Advantages of OOPs

- Code reusability
- Easy debugging
- Better security
- Improved maintainability
- Real-world modeling

18. Limitations of OOPs

- Requires more memory
- Slower execution compared to procedural programs
- Complex design for small programs

19. Applications of OOPs

OOPs is widely used in:

- Game development
- Banking software
- GUI applications
- Web development

- Embedded systems

---

20. Conclusion

Object-Oriented Programming is a powerful programming approach that helps in designing efficient, secure, and reusable software. C++ provides full support for OOPs concepts, making it suitable for both small and large-scale applications. Understanding OOPs concepts is essential for becoming a skilled C++ programmer.