Complete OOPs (Object-Oriented Programming) Theory in C++

1. Encapsulation

Definition: Wrapping data and the functions that operate on that data into a single unit (class). It hides the internal state of the object from the outside world.

Example:

```
class Person {
private:
    string name;
    int age;
public:
    void setName(string n) { name = n; }
    string getName() { return name; }
};
```

2. Abstraction

Definition: Hiding internal implementation and showing only necessary details.

Example:

```
class Animal {
public:
    virtual void sound() = 0;
};
class Dog : public Animal {
public:
    void sound() override { cout << "Bark" << endl; }
};
```

3. Inheritance

Definition: One class (child) acquires properties of another class (parent).

Example:

```
class Vehicle {
public:
    void start() { cout << "Vehicle started" << endl; }
};
class Car : public Vehicle { };
```

4. Polymorphism

A. Compile-Time Polymorphism (Function Overloading)

```
class Print {
public:
    void show(int x) { cout << x; }
    void show(string s) { cout << s; }
};
```

B. Run-Time Polymorphism (Function Overriding)

```
class Base {
public:
    virtual void show() { cout << "Base"; }
};
class Derived : public Base {
public:
    void show() override { cout << "Derived"; }
};
```

5. Static Keyword

Static Variable:

```
class Counter {
public:
    static int count;
    Counter() { count++; }
};
int Counter::count = 0;
```

Static Function:

```
class Test {
public:
    static void show() { cout << "Static Function"; }
};
```

6. Const Keyword

- const int x = 5;
- void show() const;
- const int* ptr;

7. Virtual Function

Definition: Allows function to be overridden in derived class.

```
class Base {
public:
    virtual void display() { cout << "Base"; }
};
```

Pure Virtual Function:

```
class Interface {
public:
    virtual void operation() = 0;
};
```

8. Shallow vs Deep Copy

Shallow Copy:

```
class A {
public:
    int* ptr;
    A(int val) { ptr = new int(val); }
    A(const A& a) { ptr = a.ptr; }
};
```

Deep Copy:

```
A(const A& a) { ptr = new int(*a.ptr); }
```

9. Friend Function & Class

Friend Function:

```
class A {
private:
    int x = 10;
    friend void show(A);
};
void show(A a) { cout << a.x; }
```

Friend Class:

```
class B;
class A {
    friend class B;
};
```

10. Exception Handling

```
try {
    throw "Error!";
} catch (const char* e) {
    cout << e;
}
```

11. Association, Aggregation, Composition

Concept	Ownership	Lifetime Dependency	Example
Association	No	No	Teacher - Student
Aggregation	Yes	No	Dept - Student (ptr)
Composition	Yes	Yes	House - Room (obj)

Summary Table

Concept	Type	Purpose
Inheritance	OOP Pillar	Reuse code
Polymorphism	OOP Pillar	Many forms (overload/override)
Encapsulation	OOP Pillar	Data hiding

| Abstraction | OOP Pillar | Show only essential |

| Static | Keyword | Shared across instances |

| Const | Keyword | Prevent modification |

| Virtual | Function | Enable overriding |

| Friend | Class/Func | Access private members |

| Copy Constructors | Shallow/Deep | Object copying |