✅ 1 Getter & Setter (Accessor & Mutator)

🔷 Definition

Getter (Accessor):
A function that returns the value of a private data member.

Setter (Mutator):
A function that modifies or sets the value of a private data member.

🔷 Key Points

- Used to maintain data hiding
- Provide controlled access to private data
- Allow validation logic
- Improve security of class
- Follow OOP design principle

🔷 Example

```
#include <iostream>
using namespace std;

class BankAccount
{
private:
    double balance;

public:
    void setBalance(double b)   // Setter
    {
        if(b >= 0)
            balance = b;
        else
            cout << "Invalid amount!" << endl;
    }

    double getBalance() const   // Getter
    {
        return balance;
    }
};

int main()
{
```

```
    BankAccount acc;
    acc.setBalance(5000);
    cout << acc.getBalance();

    return 0;
}
```

✅ 2 THIS Pointer

🔷 Definition

`this` is a pointer that refers to the current calling object.

🔷 Why Needed?

When local variable name and data member name are same.

🔷 Example

```
class Student
{
private:
    string name;

public:
    void setName(string name)
    {
        this->name = name;
    }
};
```

Without `this`, compiler gets confused.

🔷 Method Chaining Example

```
class Test
{
private:
    int x;

public:
```

```
    Test& setValue(int x)
    {
        this->x = x;
        return *this;
    }

    void display()
    {
        cout << x;
    }
};
```

✅ 3 Const Member Functions

🔷 Definition

A function that cannot modify object data.

```
int getValue() const;
```

🔷 Key Points

- Protects object state
- Required when calling using const object
- Good programming practice

🔷 Example

```
class Example
{
private:
    int value;

public:
    Example(int v) { value = v; }

    int getValue() const
    {
        return value;
    }
};
```

✅ 4 Static Data Members

🔷 Definition

A variable shared among all objects of a class.

🔷 Example

```
class Counter
{
private:
    static int count;

public:
    Counter()
    {
        count++;
    }

    static int getCount()
    {
        return count;
    }
};

int Counter::count = 0;
```

✅ 5 Static Member Functions

🔷 Key Points

- Can access only static members
- Called using class name

```
cout << Counter::getCount();
```

✅ 6 Initialization List

🔷 Definition

Used to initialize data members before constructor body runs.

🔷 Example

```
class Student
{
private:
    string name;

public:
    Student(string n) : name(n)
    {
    }
};
```

✅ 7 Constant Data Members

Must be initialized using initialization list.

```
class Test
{
private:
    const int id;

public:
    Test(int i) : id(i)
    {
    }
};
```

✅ 8 Destructor

🔷 Definition

A special function that runs automatically when object is destroyed.

🔷 Syntax

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

🔷 Example

```
class Demo
{
public:
    ~Demo()
    {
        cout << "Destructor called";
    }
};
```

🔷 Destructor with Dynamic Memory

```
class Example
{
private:
    int* ptr;

public:
    Example(int val)
    {
        ptr = new int(val);
    }

    ~Example()
    {
        delete ptr;
    }
};
```

✅ 9 Rule of Three

If your class uses:

- Dynamic memory
- Pointer
- Resource management

You must define:

1. Destructor
2. Copy Constructor
3. Copy Assignment Operator

✅ 🔟 Association

Before inheritance, we must understand object relationships.

🔷 1. Simple Association

Two independent objects connected.

Example:
Student has Laptop

🔷 2. Aggregation

Weak relationship (can exist independently).

Example:
Department has Teachers

🔷 3. Composition

Strong ownership.

Example:
Car has Engine
If Car is destroyed → Engine destroyed

🔷 Composition Example

```
class Engine
{
public:
    void start()
    {
        cout << "Engine started";
    }
};
```

```
class Car
{
private:
    Engine engine;  // Composition

public:
    void startCar()
    {
        engine.start();
    }
};
```

🎯 Final Encapsulation Completion Checklist

- ✔ Getters & Setters
- ✔ `this` Pointer
- ✔ Const Member Functions
- ✔ Static Members
- ✔ Initialization List
- ✔ Const Data Members
- ✔ Destructor
- ✔ Rule of Three
- ✔ Association (Aggregation & Composition)