Code Reusability	Code reuse is limited to functions.	Encourages the reuse of classes and objects through inheritance and composition.
Maintainabillity	Managing complexity becomes challenging as the program grows in size.	Provides better organization and easier maintenance through modularization and encapsulation.
Encapsulation	Not a primary concern; data and functions are loosely connected.	Encourages bundling data and methods into a single unit (class), promoting data hiding and abstraction.
Inheritance	Not supported or limited (e.g., through library functions).	Supports inheritance, allowing new classes to inherit properties and behaviors from existing classes, promoting code reusability.
Polymorphism	Achieved through functions with the same name but different parameters.	Achieved through method overriding and dynamic dispatch, allowing objects to be treated as instances of their parent class.
Abstraction	Limited; focuses more on procedural logic.	Promotes abstraction by simplifying complex systems through modeling classes based on essential features.
Example Languages	C, Pascal, BASIC	Java, C++, Python

1.2 Basic Concept of OOP

A. Classes and Objects:

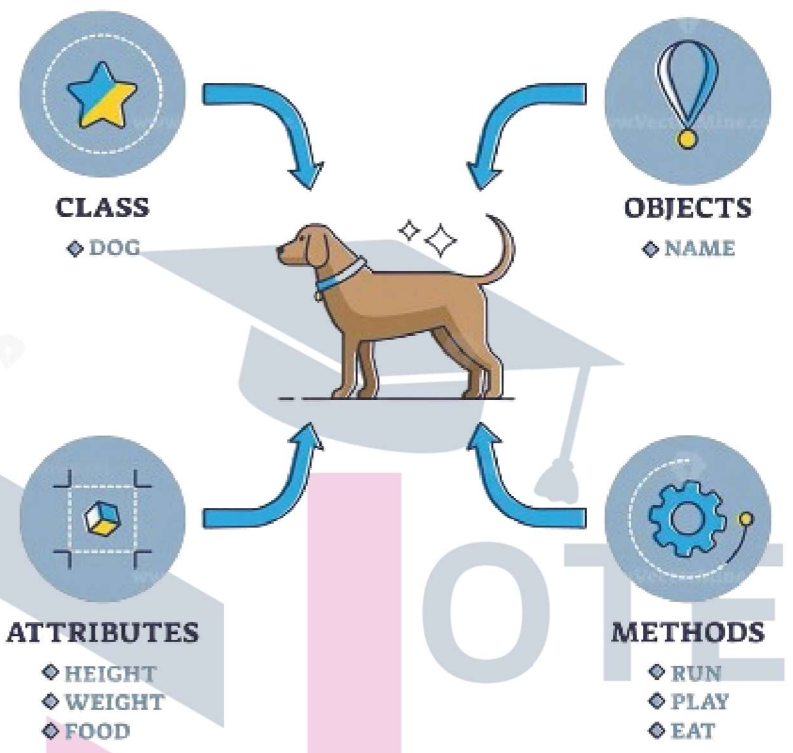
Classes:

- A class is a blueprint or template for creating objects. It defines the properties (attributes) and behaviors (methods) that objects of that class will have.
- A class encapsulates data members (attributes) and member functions (methods) that operate on the data.
- Classes facilitate code reusability and promote modularity by organizing related data and functions into a single unit.
- Classes act as user-defined data types.
- In C++, a class is declared using the class keyword, followed by the class name and a pair of curly braces containing class members.
- **Syntax:**

```
class MyClass {
    // Class members (attributes and methods)
};
```

Objects:

- Objects are instances of classes. They represent specific instances of the class, each with its own unique state.
- An object encapsulates data and behaviour, and provides an interface (through its methods) to interact with that data.
- Multiple objects can be created from the same class, each with its own independent state.
- Object attributes can be initialized using the dot (.) operator.



B. Attributes and Methods:

Attributes:

- Attributes are the data members or variables associated with a class.
- They represent the state of an object and define its characteristics or properties.
- Attributes are declared within the class and are usually private to enforce encapsulation, but they can also be public or protected based on the desired access level.
- Data members can be accessed using the dot (.) operator.

Methods:

- Methods are functions associated with a class.
- They define the behavior or actions that objects of the class can perform.
- Member functions are called using the dot (.) operator
- Methods operate on the object's data (attributes) and can manipulate its state.
- Methods can be public, private, or protected, determining their accessibility from outside the class.

Example:

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

// Class declaration
class Student {
public:
    // Data members
    int studentID;
    string name;

    // Member function to display student information
    void displayInfo() {
        cout << "Student ID: " << studentID << ", Name: " << name << endl;
    }
};

int main() {
    // Creating objects of class Student
    Student student1, student2;

    // Initializing object attributes
    student1.studentID = 101;
    student1.name = "Deepak Modi";

    student2.studentID = 102;
    student2.name = "Sumit Modi";

    // Accessing member functions to display student information
    cout << "Student 1: ";
    student1.displayInfo();

    cout << "Student 2: ";
    student2.displayInfo();

    return 0;
}
```

C. Defining Classes:

Defining a Class without a Constructor

- When a class is defined without a constructor, the compiler generates a default constructor implicitly. This default constructor initializes the data members of the class with default values (zero for numeric types, empty string for string types, etc.).

- However, if any constructor is explicitly defined within the class, the compiler does not generate the default constructor.

Example:

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

// Class declaration
class Person {
public:
    // Data members
    string name;
    int age;

    // Member function to display person's information
    void displayInfo() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    // Creating an object of class Person
    Person p1;

    // Initializing object attributes
    p1.name = "Sanjay";
    p1.age = 30;

    // Accessing member function to display person's information
    p1.displayInfo();

    return 0;
}
```

Defining a Class with a Constructor

- A constructor is a special member function of a class that is automatically called when an object of that class is created.
- It is used to initialize object properties.
- It has the same name as the class. It has no return type.

Example:

```
#include <iostream>
#include <string>
```

```
using namespace std;

// Class declaration
class Person {
public:
    // Data members
    string name;
    int age;

    // Constructor declaration
    Person(string n, int a) {
        name = n;
        age = a;
    }

    // Member function to display person's information
    void displayInfo() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    // Creating an object of class Person with constructor
    Person p1("Sanjay", 30);

    // Accessing member function to display person's information
    p1.displayInfo();

    return 0;
}
```

Defining a Class with this Keyword

- The **"this"** keyword in C++ is a pointer that refers to the current instance of a class. It is used within class methods to refer to the current object on which the method is being invoked.
- ***this->member*** is used to access the members (attributes or methods) of the current object.
- It resolves the scope and ensures that the member being accessed belongs to the current object.

Example:

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

```cpp
class MyClass {
private:
    int value;

public:
    // Constructor with parameter
    MyClass(int value) {
        // Using 'this' to distinguish between member and parameter
        this->value = value;
    }

    // Member function to display value
    void displayValue() {
        // Accessing 'value' using 'this'
        cout << "Value: " << this->value << endl;
    }
};

int main() {
    // Creating object of MyClass
    MyClass obj(10);

    // Calling member function
    obj.displayValue();

    return 0;
}
```

D. Constructor and its Types

Constructors are special member functions that are automatically called when an object is created.

Default constructors (with no parameters)

- A default constructor is a constructor that does not take any parameters. It is called implicitly when an object of the class is created without any arguments.
- The default constructor initializes the data members with default values.
- If no constructor is explicitly defined within the class, the compiler generates a default constructor automatically.

Parameterized constructors (with parameters)

- A parameterized constructor is a constructor that takes one or more parameters. It allows for initializing object attributes with specified values at the time of object creation.
- The parameterized constructor takes the arguments and initializes the data members with the provided values.

Example:

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

// Class declaration
class Car {
private:
    string brand;
    string model;
    int year;

public:
    // Default constructor
    Car() {
        brand = "";
        model = "";
        year = 0;
    }

    // Parameterized constructor
    Car(string brand, string model, int year) {
        this->brand = brand;
        this->model = model;
        this->year = year;
    }

    // Member function to display car information
    void displayInfo() {
        cout << "Brand: " << brand << ", Model: " << model << ", Year: "
<< year << endl;
    }
};

int main() {
    // Creating objects of Car class using default and parameterized
constructors
    Car car1;                                  // Using default constructor
    Car car2("Toyota", "Camry", 2022);  // Using parameterized constructor

    // Displaying information about the cars
    cout << "Car 1: ";
    car1.displayInfo();

    cout << "Car 2: ";
```