6. Introduction to Object-Oriented Programming

1. Explain the key concepts of Object-Oriented Programming (OOP).

- Object-Oriented Programming (OOP) centers around the concept of objects, which encapsulate data (attributes) and methods (functions) that operate on that data. Key principles of OOP include encapsulation, inheritance, polymorphism, and abstraction. These principles promote modularity, reusability, and maintainability in software development.

    1. **Object:**

       A fundamental unit in OOP, representing a real-world entity or concept. It has attributes (data) and methods (behavior) that define its state and actions.

    2. **Class:**
       A blueprint or template for creating objects. It defines the structure and behavior of a set of objects that share common characteristics.
    3. **Encapsulation:**
       Bundling the data (attributes) and methods that operate on that data within a single unit (the class). This hides internal implementation details and protects data from accidental modification.
    4. **Inheritance:**
       A mechanism where a new class (subclass or derived class) can inherit properties and methods from an existing class (superclass or base class). This promotes code reuse and reduces redundancy.
    5. **Polymorphism:**
       The ability of objects of different classes to respond to the same method call in their own way. This allows for generic code that can work with various object types.
    6. **Abstraction:**
       Presenting only the essential features of an object to the user, hiding unnecessary complexity and implementation details. This simplifies interaction with objects and promotes modularity.

2. What are classes and objects in C++? Provide an example.

- In C++, a class acts as a blueprint or template for creating objects, while an object is a specific instance of that class. Classes encapsulate data (attributes) and functions (methods) that operate on that data, defining the structure and behavior of objects. Think of a class as a cookie cutter, and objects as the cookies themselves.

  - Example:

```
#include <iostream>
#include <string>
// Define a class named "Car"
class Car {
public:
   // Attributes (data members)
   std::string model;
   std::string color;
   int year;

   // Method (member function)
   void displayCarInfo() {
      std::cout << "Model: " << model << std::endl;
      std::cout << "Color: " << color << std::endl;
      std::cout << "Year: " << year << std::endl;
   }
};
int main() {
   // Create an object (instance) of the Car class
   Car myCar;
   // Assign values to the object's attributes
   myCar.model = "Toyota Camry";
   myCar.color = "Blue";
   myCar.year = 2023;
  // Call the object's method to display its information
myCar.displayCarInfo();
  return 0;
}
```

3. What is inheritance in C++? Explain with an example.

- Inheritance in C++ is a core concept of Object-Oriented Programming (OOP) that allows a new class to inherit properties and behaviors (methods) from an existing class. The existing class is referred to as the base class (or parent class), and the new class is called the derived class (or child class). This mechanism promotes code reusability and establishes an "is-a" relationship between classes.

    - Example:

      Consider a scenario where we have a base class Animal and a derived class Dog. A Dog "is an" Animal, so it makes sense to use inheritance.

```
#include <iostream>
// Base class
class Animal {
public:
   void eat() {
      std::cout << "This animal eats food." << std::endl;
   }
   void sleep() {
      std::cout << "This animal sleeps." << std::endl;
   }
};
// Derived class
class Dog : public Animal { // Dog inherits publicly from Animal
public:
   void bark() {
      std::cout << "The dog barks: Woof woof!" << std::endl;
   }
};
int main() {
   Dog myDog;
   // Accessing inherited methods from the base class
   myDog.eat();
   myDog.sleep();
```

```
 // Accessing method specific to the derived class
 myDog.bark();
return 0;
}
```

4. What is encapsulation in C++? How is it achieved in classes?

- Encapsulation in C++ is the mechanism of bundling the data (attributes) and methods (functions) that operate on that data into a single unit, which is a class. It's a core concept of Object-Oriented Programming (OOP) that helps in achieving data hiding and controlling access to the class members.

❖ **Encapsulation is Achieved in Classes:**

1. **Bundling Data and Methods:**
   Encapsulation involves grouping related data members (variables) and member functions (methods) within a class. This creates a cohesive unit, making the class a self-contained entity.

2. **Access Modifiers:**

   C++ provides access specifiers (public, private, protected) to control the visibility and accessibility of class members.
   - **Private:** Members declared as private are only accessible from within the class itself.
   - **Public:** Members declared as public are accessible from anywhere, both inside and outside the class.
   - **Protected:** Members declared as protected are accessible from within the class and its derived classes.

3. **Data Hiding:**

   Encapsulation enables data hiding by making data members private, preventing direct access from outside the class.

4. **Controlled Access:**

Access to the private data members is provided through public member functions, often called "getters" and "setters". These functions act as an interface for interacting with the data.

- **Example**:

```
class BankAccount {
private:
   double balance;  // Data member (private)

public:
   // Constructor
   BankAccount(double initialBalance) : balance(initialBalance) {}

   // Public methods to access and modify the balance
   double getBalance() {
      return balance;
   }

   void deposit(double amount) {
      balance += amount;
   }

   void withdraw(double amount) {
      if (amount <= balance) {
         balance -= amount;
      } else {
         // Handle insufficient funds
      }
   }
};
```

❖ **Benefits of Encapsulation:**

1. **Data Protection:**
   Prevents accidental or unauthorized modification of data.
2. **Code Organization:**

Improves code readability and maintainability by grouping related data and functions.

3. **Code Reusability:**
   Encapsulated components can be easily reused in different parts of the code or in other projects.
4. **Maintainability:**
   Reduces the impact of changes as modifications to internal data are localized within the class.