Explanation in Prog-1

Class and objects

1. Class Definition (Step 1): The class Book is defined to encapsulate the properties and behaviors of a book.
2. Private Data Members (Step 2): The private section contains the data members (title, author, publicationYear) that represent the state of the object. These members are accessible only within the class.
3. Public Member Functions (Step 3): Public member functions include the constructor, getter methods, setter methods, and a display method. These functions define how the outside world interacts with the class.
4. Constructor (Step 4): The constructor initializes the object when it is created. It takes initial values for the book's title, author, and publication year.
5. Getter Methods (Step 5): Getter methods allow access to the private data members from outside the class.
6. Setter Methods (Step 6): Setter methods allow modification of the private data members from outside the class.
7. Display Method (Step 7): The displayBookDetails method prints the details of a book to the console.
8. Main Function (Step 8): The main function demonstrates the usage of the Book class by creating two book objects and modifying one of them.
9. Object Creation and Modification (Step 9): Two book objects, book1 and book2, are created, and their details are displayed. The title of book1 is then updated, and the modified details are displayed again.

C++ is an object-oriented programming language.

Everything in C++ is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has attributes, such as weight and color, and methods, such as drive and brake.

Attributes and methods are basically variables and functions that belongs to the class. These are often referred to as "class members".

A class is a user-defined data type that we can use in our program, and it works as an object constructor, or a "blueprint" for creating objects.

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

// Step 1: Define the class
class Book {
private:
    // Step 2: Declare private data members
    std::string title;
    std::string author;
    int publicationYear;

public:
    // Step 3: Declare public member functions (methods)

    // Constructor (Step 4)
    Book(const std::string& t, const std::string& a, int year) {
        title = t;
        author = a;
        publicationYear = year;
    }

    // Getter methods (Step 5)
    std::string getTitle() const {
        return title;
    }

    std::string getAuthor() const {
        return author;
    }

    int getPublicationYear() const {
        return publicationYear;
    }

    // Setter methods (Step 6)
    void setTitle(const std::string& t) {
        title = t;
    }

    void setAuthor(const std::string& a) {
        author = a;
    }

    void setPublicationYear(int year) {
        publicationYear = year;
    }

    // Display method (Step 7)
    void displayBookDetails() const {
        std::cout << "Title: " << title << std::endl;
        std::cout << "Author: " << author << std::endl;
        std::cout << "Publication Year: " << publicationYear << std::endl;
    }
};

int main() {
    // Step 8: Create objects of the class
    Book book1("The Catcher in the Rye", "J.D. Salinger", 1951);
    Book book2("To Kill a Mockingbird", "Harper Lee", 1960);

    // Step 9: Access and modify object properties using methods
    book1.displayBookDetails();

    std::cout << "\nUpdating book1 title...\n";
    book1.setTitle("New Title for Book1");

    std::cout << "\nUpdated book1 details:\n";
    book1.displayBookDetails();

    return 0;
}
```

Explanation in Prog-2 C++ Constructors

1. Class Definition (Step 1): The class Car is defined to encapsulate the properties and behaviors of a car.
2. Private Data Members (Step 2): The private section contains the data members (model, manufacturer, year) that represent the state of the object. These members are accessible only within the class.
3. Constructors (Step 3):
    - Default Constructor (Step 4): The default constructor initializes the object with default values for the car model, manufacturer, and year.
    - Parameterized Constructor (Step 5): The parameterized constructor allows the user to provide initial values for the car model, manufacturer, and year when creating an object.
4. Member Functions (Step 6): The displayCarDetails method is a member function that prints the details of a car to the console.
5. Main Function (Step 7): The main function demonstrates the usage of constructors by creating two car objects: one using the default constructor and the other using the parameterized constructor.
6. Object Creation and Display (Step 8): Two car objects, defaultCar and customCar, are created using different constructors. The details of each car are then displayed using the displayCarDetails method.

Constructors

A constructor in C++ is a special method that is automatically called when an object of a class is created.

To create a constructor, use the same name as the class, followed by parentheses ():

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

// Step 1: Define the class
class Car {
private:
    // Step 2: Declare private data members
    std::string model;
    std::string manufacturer;
    int year;

public:
    // Step 3: Constructors

    // Default constructor (Step 4)
    Car() {
        model = "Unknown";
        manufacturer = "Unknown";
        year = 0;
    }

    // Parameterized constructor (Step 5)
    Car(const std::string& carModel, const std::string& carManufacturer, int carYear) {
        model = carModel;
        manufacturer = carManufacturer;
        year = carYear;
    }

    // Step 6: Member functions

    // Display method
    void displayCarDetails() const {
        std::cout << "Model: " << model << std::endl;
        std::cout << "Manufacturer: " << manufacturer << std::endl;
        std::cout << "Year of Manufacture: " << year << std::endl;
    }
};

int main() {
    // Step 7: Create objects of the class using constructors

    // Creating a car using the default constructor
    Car defaultCar;

    // Creating a car using the parameterized constructor
    Car customCar("Mustang", "Ford", 2022);

    // Step 8: Access and display object properties
    std::cout << "Default Car Details:\n";
    defaultCar.displayCarDetails();

    std::cout << "\nCustom Car Details:\n";
    customCar.displayCarDetails();

    return 0;
}
```

Explanation in Prog-2 C++ Constructors

Class Methods

Methods are functions that belongs to the class.

There are two ways to define functions that belongs to a class:

- Inside class definition
- Outside class definition

Method to Update Car Details (Step 1): The updateCarDetails method allows for changing the details of a car, such as the model, manufacturer, and year.

Method to Calculate Car Age (Step 2): The calculateCarAge method computes the age of the car based on the assumption that the current year is 2024. You can adjust this value according to the actual current year.

Main Function (Step 3): The main function demonstrates the usage of the updated Car class by creating two car objects and using the new methods to update details and calculate the age.

Object Creation and Display (Step 4): Two car objects, defaultCar and customCar, are created using different constructors. The details of each car are then displayed using the displayCarDetails method.

Use of Class Methods (Step 5): The updateCarDetails method is used to change the details of customCar, and the calculateCarAge method is used to determine its age. The updated details and calculated age are then displayed.

```cpp
#include <iostream>
#include <string>
#include <ctime>

// Step 1: Define the class
class Car {
private:
    // Step 2: Declare private data members
    std::string model;
    std::string manufacturer;
    int year;

public:
    // Step 3: Constructors

    // Default constructor (Step 4)
    Car() {
        model = "Unknown";
        manufacturer = "Unknown";
        year = 0;
    }

    // Parameterized constructor (Step 5)
    Car(const std::string& carModel, const std::string& carManufacturer, int carYear) {
        model = carModel;
        manufacturer = carManufacturer;
        year = carYear;
    }

    // Step 6: Member functions

    // Display method
    void displayCarDetails() const {
        std::cout << "Model: " << model << std::endl;
        std::cout << "Manufacturer: " << manufacturer << std::endl;
        std::cout << "Year of Manufacture: " << year << std::endl;
    }

    // Method to update car details (Step 7)
    void updateCarDetails(const std::string& newModel, const std::string& newManufacturer, int newYear) {
        model = newModel;
        manufacturer = newManufacturer;
        year = newYear;
    }

    // Method to calculate the age of the car (Step 8)
    int calculateCarAge() const {
        // Assuming the current year is 2024
        int currentYear = 2024;
        return currentYear - year;
    }
};

int main() {
    // Step 9: Create objects of the class using constructors

    // Creating a car using the default constructor
    Car defaultCar;

    // Creating a car using the parameterized constructor
    Car customCar("Mustang", "Ford", 2022);

    // Step 10: Access and display object properties
    std::cout << "Default Car Details:\n";
    defaultCar.displayCarDetails();

    std::cout << "\nCustom Car Details:\n";
    customCar.displayCarDetails();

    // Step 11: Use class methods to update details and calculate age
    std::cout << "\nUpdating Custom Car Details...\n";
    customCar.updateCarDetails("Civic", "Honda", 2019);

    std::cout << "\nUpdated Custom Car Details:\n";
    customCar.displayCarDetails();

    std::cout << "\nAge of Custom Car: " << customCar.calculateCarAge() << " years\n";

    return 0;
}
```

Access specifiers

in C++ determine the visibility and accessibility of class members (data members and member functions) from outside the class. There are three access specifiers in C++:

1. Public: Members declared as public are accessible from anywhere, including outside the class.
2. Private: Members declared as private are not accessible from outside the class. Only the class itself and its friend functions can access private members.
3. Protected: Members declared as protected are similar to private members, but they are accessible in derived classes. Outside the class and its derived classes, protected members are not accessible.

Explanation:

1. Public Members and Methods (Step 1): The class MyClass has a public member variable (publicMember), a public member function (publicMethod), a private member variable (privateMember), a private member function (privateMethod), a protected member variable (protectedMember), and a protected member function (protectedMethod).
2. Access in Main Function (Step 2 and 3): In the main function, an object of MyClass is created (myObject). Public members and methods can be accessed directly from outside the class.
3. Private and Protected Access (Note): Attempting to access private and protected members and methods from outside the class will result in compilation errors, as indicated in the comments. They are only accessible within the class itself and its friend functions.

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

// Step 1: Define the class with public, private, and protected members
class MyClass {
public:
    // Public member
    int publicMember;

    // Public member function
    void publicMethod() {
        std::cout << "Public method called.\n";
    }

private:
    // Private member
    int privateMember;

    // Private member function
    void privateMethod() {
        std::cout << "Private method called.\n";
    }

protected:
    // Protected member
    int protectedMember;

    // Protected member function
    void protectedMethod() {
        std::cout << "Protected method called.\n";
    }
};

int main() {
    // Step 2: Create an object of the class
    MyClass myObject;

    // Step 3: Access public members and methods
    myObject.publicMember = 42;
    std::cout << "Public member value: " << myObject.publicMember << std::endl;
    myObject.publicMethod();

    // Note: Uncommenting the following lines will result in compilation errors
    // because private and protected members are not accessible from outside the class.

    // myObject.privateMember = 10;  // Error: 'int MyClass::privateMember' is private
    // myObject.privateMethod();     // Error: 'void MyClass::privateMethod()' is private

    // myObject.protectedMember = 20;  // Error: 'int MyClass::protectedMember' is protected
    // myObject.protectedMethod();     // Error: 'void MyClass::protectedMethod()' is protected

    return 0;
}
```

Encapsulation

is one of the fundamental principles of Object-Oriented Programming (OOP). It refers to the bundling of data (attributes) and the methods (functions) that operate on the data into a single unit, known as a class. The class controls access to its members by using access specifiers (public, private, protected). Encapsulation helps in hiding the implementation details of a class and allows for a clear separation between the internal workings of the class and the external code that uses it.

In this example, we'll create a BankAccount class with private data members (account number and balance) and public member functions for deposit, withdrawal, and displaying the account details.

1. BankAccount Class (Encapsulation): The BankAccount class encapsulates the account details by having private data members (accountNumber and balance). These members can only be accessed and modified through the public member functions.
2. Constructor (Initialization): The constructor initializes the accountNumber and balance when a BankAccount object is created.
3. Deposit and Withdrawal Methods: The public member functions deposit and withdraw allow external code to interact with the object in a controlled manner. They perform validation checks and update the account balance accordingly.
4. Display Method: The displayAccountDetails method allows external code to view the account details without accessing the private members directly.
5. Main Function (Usage): In the main function, a BankAccount object is created, and its public member functions are used to deposit, withdraw, and display account details. The encapsulation ensures that the internal state of the BankAccount object is controlled and manipulated through well-defined interfaces.

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

class BankAccount {
private:
    // Private data members
    std::string accountNumber;
    double balance;

public:
    // Public member functions

    // Constructor
    BankAccount(const std::string& accNumber, double initialBalance) {
        accountNumber = accNumber;
        balance = initialBalance;
    }

    // Deposit method
    void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            std::cout << "Deposit of $" << amount << " successful.\n";
        } else {
            std::cout << "Invalid deposit amount.\n";
        }
    }

    // Withdrawal method
    void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            std::cout << "Withdrawal of $" << amount << " successful.\n";
        } else {
            std::cout << "Invalid withdrawal amount or insufficient funds.\n";
        }
    }

    // Display method
    void displayAccountDetails() const {
        std::cout << "Account Number: " << accountNumber << std::endl;
        std::cout << "Balance: $" << balance << std::endl;
    }
};

int main() {
    // Create a BankAccount object
    BankAccount myAccount("12345", 1000.0);

    // Access public member functions to perform operations
    myAccount.displayAccountDetails();

    myAccount.deposit(500.0);
    myAccount.displayAccountDetails();

    myAccount.withdraw(200.0);
    myAccount.displayAccountDetails();

    myAccount.withdraw(1500.0); // Attempting to withdraw more than the balance
    myAccount.displayAccountDetails();

    return 0;
}
```

ENCAPSULATION PROG 2

1. The program starts by taking user input for the initial account number and balance.
2. A BankAccount object is created using the user-provided values.
3. A menu-driven loop allows the user to choose operations like deposit, withdrawal, and displaying account details.
4. The program takes user input for the chosen operation and performs the corresponding action using the BankAccount object's methods.
5. The loop continues until the user chooses to exit (enters 0). The program then prints a message and exits.

This modified program allows users to interactively perform operations on the BankAccount class by entering their choices and input values during runtime.

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

class BankAccount {
private:
    std::string accountNumber;
    double balance;

public:
    BankAccount(const std::string& accNumber, double initialBalance) {
        accountNumber = accNumber;
        balance = initialBalance;
    }

    void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            std::cout << "Deposit of $" << amount << " successful.\n";
        } else {
            std::cout << "Invalid deposit amount.\n";
        }
    }

    void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            std::cout << "Withdrawal of $" << amount << " successful.\n";
        } else {
            std::cout << "Invalid withdrawal amount or insufficient funds.\n";
        }
    }

    void displayAccountDetails() const {
        std::cout << "Account Number: " << accountNumber << std::endl;
        std::cout << "Balance: $" << balance << std::endl;
    }
};

int main() {
    // Get initial values from user
    std::string accountNumber;
    double initialBalance;

    std::cout << "Enter account number: ";
    std::getline(std::cin, accountNumber);

    std::cout << "Enter initial balance: $";
    std::cin >> initialBalance;

    // Create a BankAccount object
    BankAccount myAccount(accountNumber, initialBalance);

    // Perform operations based on user input
    int choice;
    double amount;

    do {
        std::cout << "\n1. Deposit\n";
        std::cout << "2. Withdraw\n";
        std::cout << "3. Display Account Details\n";
        std::cout << "0. Exit\n";
        std::cout << "Enter your choice: ";
        std::cin >> choice;

        switch (choice) {
            case 1:
                std::cout << "Enter deposit amount: $";
                std::cin >> amount;
                myAccount.deposit(amount);
                break;
            case 2:
                std::cout << "Enter withdrawal amount: $";
                std::cin >> amount;
                myAccount.withdraw(amount);
                break;
            case 3:
                myAccount.displayAccountDetails();
                break;
            case 0:
                std::cout << "Exiting program.\n";
                break;
            default:
                std::cout << "Invalid choice. Try again.\n";
        }
    } while (choice != 0);

    return 0;
}
```

Inheritance

In C++, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:

- derived class (child) - the class that inherits from another class
- base class (parent) - the class being inherited from

To inherit from a class, use the : symbol.

Inheritance is a fundamental concept in Object-Oriented Programming (OOP) that allows a class to inherit properties and behaviors from another class. In C++, a derived class can inherit members (data members and member functions) from a base class. Let's create a simple program that demonstrates inheritance with a BankAccount class as the base class and a SavingsAccount class as the derived class.

1. Base Class (BankAccount): The BankAccount class has a constructor, deposit, withdraw, and displayAccountDetails methods.
2. Derived Class (SavingsAccount): The SavingsAccount class is derived from BankAccount and adds an interestRate and applyInterest method. It overrides the withdraw method to include a penalty for withdrawals.
3. Inheritance: The SavingsAccount class inherits the members (constructor, deposit, withdraw, and displayAccountDetails) from the BankAccount class using the public access specifier.
4. Constructor Initialization: The constructor of the SavingsAccount class initializes its own members and calls the constructor of the base class (BankAccount) using the member initializer list.
5. Overriding Methods: The withdraw method is overridden in the SavingsAccount class to provide a different implementation than the one in the base class.
6. Usage in main Function: The main function creates a SavingsAccount object and performs operations using both the inherited and overridden methods. The object can be treated as both a BankAccount and a SavingsAccount.

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

class BankAccount {
protected:
    std::string accountNumber;
    double balance;

public:
    BankAccount(const std::string& accNumber, double initialBalance) {
        accountNumber = accNumber;
        balance = initialBalance;
    }

    void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            std::cout << "Deposit of $" << amount << " successful.\n";
        } else {
            std::cout << "Invalid deposit amount.\n";
        }
    }

    virtual void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            std::cout << "Withdrawal of $" << amount << " successful.\n";
        } else {
            std::cout << "Invalid withdrawal amount or insufficient funds.\n";
        }
    }

    void displayAccountDetails() const {
        std::cout << "Account Number: " << accountNumber << std::endl;
        std::cout << "Balance: $" << balance << std::endl;
    }
};

class SavingsAccount : public BankAccount {
private:
    double interestRate;

public:
    SavingsAccount(const std::string& accNumber, double initialBalance, double rate)
        : BankAccount(accNumber, initialBalance), interestRate(rate) {}

    void applyInterest() {
        double interestAmount = balance * (interestRate / 100.0);
        deposit(interestAmount);
        std::cout << "Interest applied: $" << interestAmount << std::endl;
    }

    void withdraw(double amount) override {
        const double penalty = 10.0;

        if (amount > 0 && amount <= balance) {
            balance -= (amount + penalty);
            std::cout << "Withdrawal of $" << amount << " with penalty. Total deducted: $" << (amount + penalty) << std::endl;
        } else {
            std::cout << "Invalid withdrawal amount or insufficient funds.\n";
        }
    }
};

int main() {
    // Get initial values from user
    std::string accountNumber;
    double initialBalance, interestRate;

    std::cout << "Enter account number: ";
    std::getline(std::cin, accountNumber);

    std::cout << "Enter initial balance: $";
    std::cin >> initialBalance;

    std::cout << "Enter interest rate: ";
    std::cin >> interestRate;

    // Create a SavingsAccount object using user input
    SavingsAccount mySavings(accountNumber, initialBalance, interestRate);

    // Perform operations based on user input
    int choice;
    double amount;

    do {
        std::cout << "\n1. Deposit\n";
        std::cout << "2. Withdraw\n";
        std::cout << "3. Apply Interest\n";
        std::cout << "4. Display Account Details\n";
        std::cout << "0. Exit\n";
        std::cout << "Enter your choice: ";
        std::cin >> choice;

        switch (choice) {
            case 1:
                std::cout << "Enter deposit amount: $";
                std::cin >> amount;
                mySavings.deposit(amount);
                break;
            case 2:
                std::cout << "Enter withdrawal amount: $";
                std::cin >> amount;
                mySavings.withdraw(amount);
                break;
            case 3:
                mySavings.applyInterest();
                break;
            case 4:
                mySavings.displayAccountDetails();
                break;
            case 0:
                std::cout << "Exiting program.\n";
                break;
            default:
                std::cout << "Invalid choice. Try again.\n";
        }
    } while (choice != 0);

    return 0;
}
```