Object Oriented Analysis & Design Using C++

Minor-Project On : Furniture Enterprise System

Made Using C++ (Console -Application)

Primary Purpose : Demonstration of Object Orientation

By Aayush Sahay
Roll 02 Id : 1062
MCA-1st Sem (AKU)
(2024-2026)

Map of the Session:

1. Features of Project
2. Object Orientation Involved
3. Implementation of the Features
4. Discussion / Question -Answer Round

Features of the Project:

1.Create Bookshelves :Instantiate Mode

2.Create Chairs : Instantiate Mode

3. View Bookshelves and Chairs : View Mode

Implemented By :
->Abstract Class
->Runtime Object Instantiation
->Function Overriding (Runtime Polymormphism)

Concepts of Object Orientation Used:

Inheritance : Multiple Inheritance

-> The Chair & Bookshelf Class Inherit Furniture Class

Polymorphism : Runtime - Function overriding

-> Pure Virtual Functions Redifined in Child Classes.

Encapsulation : Use of Classes

Data Hiding : Use of Protected Variable,used by sub classes.

Abstraction: Abstract Class Furniture
(Only - Relevent Info + No implementation Logic)

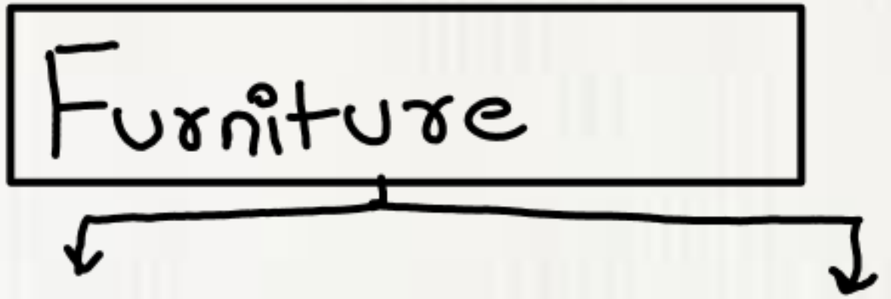
Class Digrams : [Furniture : Abstract Class]
Made up only of pure Virtual functions.
This is serving only as the Parent Class

Protected Variables:
Act as Private outside class.
Only accessible by subclasses.

Pure Virtual functions:
To Implement Dyn. Polymorphism
Must be implemented by subclass
or sub class also becomes abstract
class.

```
class Furniture {
protected:
    float length, width, height;
    float cost;
    string material, design;

public:
    virtual void setDimensions() = 0;
    virtual void getDimensions() = 0;

    virtual void setCost() = 0;
    virtual void getCost() = 0;

    virtual void setMaterialAndDesign() = 0;
    virtual void getMaterialAndDesign() = 0;

    virtual void interact() = 0;
    virtual void display() = 0;

    virtual ~Furniture() {}
};
```

Class Digram : Book Shelf and Chair




```
class Bookshelf : public Furniture {
private:
    int numShelves;
public:
    void setDimensions() override {…
    void getDimensions() override {…
    void setCost() override {…
    void getCost() override {…
    void setMaterialAndDesign() override {…
    void getMaterialAndDesign() override {…
    void interact() override {…
    void display() override {…
};
```

```
class Chair : public Furniture {
private:
    string category;
public:
    void setDimensions() override {…
    void getDimensions() override {…
    void setCost() override {…
    void getCost() override {…
    void setMaterialAndDesign() override {…
    void getMaterialAndDesign() override {…
    void interact() override {…
    void display() override {…
};
```

Notes:

```
// Arrays to hold up to 100 of each type
Bookshelf* bookshelfList[100];
Chair* chairList[100];
int bookshelfCount = 0;
int chairCount = 0;
```

To make a Global
Array of Pointers
of Both Chair and the Bookshelf class.

Purpose is to store the address of the object
as soon as possible they are created.

Intention : help us to refer when we intend
to see all of the objects.

Meanwhile , they store the address of instances
created at runtime.

Global function:

Gives a menu:
To create instance of

either one of classes.

Then Run the interact function.Which will assign properties of obj.

Assign address of obj to array of pointers.

Increase the index of array.

```
void instantiate() {
    int choice;
    do {
        cout << "\n===== Instantiate Menu =====\n";
        cout << "1. Add Bookshelf\n";
        cout << "2. Add Chair\n";
        cout << "3. Back to Main Menu\n";
        cout << "Enter your choice: ";
        cin >> choice;

        if (choice == 1 && bookshelfCount < 100) {
            Bookshelf* bs = new Bookshelf();
            bs->interact();
            bookshelfList[bookshelfCount++] = bs;
        } else if (choice == 2 && chairCount < 100) {
            Chair* ch = new Chair();
            ch->interact();
            chairList[chairCount++] = ch;
        } else if (choice == 3) {
            break;
        } else {
            cout << "Invalid choice or limit reached. Try again.\n";
        }

    } while (choice != 3);
}
```

Driver Code :

Provides Menu .

Mode { 1-> instantiate();
2-> menu to view
3-> exit
Cleanup objects after program Terminates
Problem....
Solution : Implement File Handling.

```
int main() {
    int mode;
    do {
        cout << "\n===== Main Menu =====\n";
        cout << "1. Instantiate Mode\n";
        cout << "2. View Mode\n";
        cout << "3. Exit\n";
        cout << "Enter your choice: ";
        cin >> mode;
        switch (mode) {···

    } while (mode != 3);

    // Cleanup
    for (int i = 0; i < bookshelfCount; ++i)
        delete bookshelfList[i];
    for (int i = 0; i < chairCount; ++i)
        delete chairList[i];

    return 0;
}
```

This Shows Menu:

To show the Bookshelves and chairs [View Mode]

```
switch (mode) {
    case 1:
        instantiate();
        break;
    case 2: {
        int viewChoice;
        cout << "\n--- View Mode ---\n";
        cout << "1. Show All Bookshelves\n";
        cout << "2. Show All Chairs\n";
        cout << "Enter choice: ";
        cin >> viewChoice;
        if (viewChoice == 1)
            showAllBookshelf();
        else if (viewChoice == 2)
            showAllChairs();
        else
            cout << "Invalid option.\n";
        break;
    }
    case 3:
        cout << "Exiting program...\n";
        break;
    default:
        cout << "Invalid choice. Try again.\n";
}
```

View all the

Bookshelf and Chairs

All classes Have their own Display()

by which all values of attributes are displayed on terminal.

```
void showAllBookshelf() {
    if (bookshelfCount == 0) {
        cout << "\nNo Bookshelves available.\n";
        return;
    }
    for (int i = 0; i < bookshelfCount; ++i) {
        cout << "\nBookshelf #" << (i + 1) << ":\n";
        bookshelfList[i]->display();
    }
}
void showAllChairs() {
    if (chairCount == 0) {
        cout << "\nNo Chairs available.\n";
        return;
    }
    for (int i = 0; i < chairCount; ++i) {
        cout << "\nChair #" << (i + 1) << ":\n";
        chairList[i]->display();
    }
}
```

Practical Implementation

Questions Round :

1. Reason to Make the Furniture Class Base Class?
2. The Destructor is called at the end?
   ~Furniture() virtual Destructor
   or Destructor of Bookshelf / chair Class?

Any Queries :

```
// Cleanup
for (int i = 0; i < bookshelfCount; ++i)
    delete bookshelfList[i];
for (int i = 0; i < chairCount; ++i)
    delete chairList[i];
```

Thankyou

Get the Source code at Github