**OBJECT ORIENTEDPROGRAMMING(OOPS)**

**WITH C++**

-V.BHAGYA SATYA SRI

1.Project Description:

Title: iPhone purchase System

The iPhone purchase System is a C++ program that showcases Object-Oriented Programming (OOP) principles by implementing a simple system for purchasing iPhones. The project emphasizes the use of classes, inheritance, polymorphism, and encapsulation to create a modular and extensible program.

2.OOPS used in the code:

Classes, Objects, Inheritance, Polymorphism, Encapsulation, Function overriding, Function Overloading, and Templates.

3.Source Code:

#include<iostream> #include <string>

using namespace std;

// Base class template representing a product template <typename T> class Product {

public:

string name;

T price;

// Function to display basic information about the product virtual void display() {

cout << "Model: " << name << endl;

cout << "Price: Rs." << price << endl;

}

// Virtual overloaded function to display additional information with a discount

virtual void display(double discount) {

cout << "Model: " << name << endl;

cout << "Price: Rs." << price << endl;

cout << "Discount: " << discount << "%" << endl;

cout << "Discounted Price: Rs." << price - (price \* discount / 100.0) << endl;

}

};

// Derived class representing an iPhone class IPhone :

public Product<double> {

public:

IPhone(const string& model, double price) { name = "iPhone " + model;

this->price = price;

}

// Overridden function to display iPhone-specific information

void display() override {

cout << "Company: Apple" << "\n"; // Display only the company name

Product::display(); // Call the base class display function for common information

}

// Overridden function to display additional information with a discount

void display(double discount) override {

cout << "Company: Apple" << "\n"; // Display only the company name

Product::display(discount); // Call the base class display function with a discount

}

};

int main() {

string choseniphone; double chosenPrice;

// Display available iPhone models

cout << "Available iPhone Models:\n";

cout << "1. iPhone 15 Pro Max Rs.1,79,900\n";

cout << "2. iPhone 15 Plus - Rs.99,900\n"; cout << "3. iPhone 15 - Rs.89,900\n";

// Read user input for iPhone model

cout << "\nEnter the number of the iPhone model you want to buy:"; int modelChoice;

cin >> modelChoice;

// Validate user input

while (modelChoice < 1 || modelChoice > 3) {

cout << "Invalid choice. Please enter a number between 1 and 3:";

cin >> modelChoice;

}

// Set the chosenModel and chosenPrice based on user input

switch (modelChoice) {

case 1:

choseniphone = "15 Pro Max"; chosenPrice = 179900;

break;

case 2:

choseniphone = "15 Plus"; chosenPrice = 99900;

break;

case 3:

choseniphone = "15";

chosenPrice = 89900;

break;

default:

std::cout << "Invalid choice. Exiting...\n"; return 1;

}

// Create an iPhone object using the template

IPhone myiPhone(choseniphone, chosenPrice);

// Display information about the purchased iPhone using the overridden functions

cout << "\nCongratulations! You have purchased the following iPhone:\n";

myiPhone.display(); // Calls the overridden display function in IPhone class

myiPhone.display(10.0); // Calls the overloaded display function in IPhone class with a discount return 0;

}

4.Introduction:

Object-Oriented Programming is a programming paradigm that revolves around the concept of objects, which encapsulate data and behaviour. This project serves as a practical application of OOP principles in the context of creating a purchase system for iPhones.

5.Project Overview:

* Implement a base class template (‘Product’) representing a generic product.
* Create a derived class (‘IPhone’) that inherits from the base class to represent iPhones.
* Demonstrate inheritance, polymorphism, and encapsulation.
* Allow users to choose an iPhone model, display information, and apply discounts.
* We have used C++ programming language in this code.

6. Code Structure:

Base Class (Product):

The Product class serves as a template for all products in the system.

* Attributes:
  + - name: A string representing the model name of the product.
    - price: A template type representing the price of the product.
* Methods:
  + - display(): A virtual function to display basic information about the product, such as the model name and price.
    - display(double discount): A virtual overloaded function to display additional information with a discount, including the discounted price.
* Derived Class (IPhone):
* The IPhone class inherits from the Product class and represents iPhones.
* Constructor:

IPhone(const string& model, double price):

* + - * + Initializes the iPhone object with a model name and price.
* Methods:
  + - * + display(): Overrides the base class display() to display iPhone-specific information, including the company name (“Apple”). display(double discount): Overrides the base class display(double discount) to display iPhone-specific information with a discount.
* Main Function:
  + The main function is the entry point of the program.
  + Displays available iPhones models.
  + Reads user input for the chosen iPhone model.
  + Creates an IPhone object based on the user’s choice.
  + Displays information about the purchased iPhone using the overridden display functions.

7. Project Execution:

* Design Phase:

The design phase involved identifying the classes, their attributes/methods, defining relationships, and planning the program structure. UML diagrams were used to visualize the class hierarchy and interactions.

* Implementation Phase:

The team implemented the code following the designed structure, ensuring adherence to OOP principles. Modular functions were created to enhance code readability and maintainability.

* Testing Phase:

A comprehensive testing approach was employed to verify the correct behaviour of the program. Test cases included scenarios such as valid/invalid user inputs, checking for correct information display, and verifying discount calculations.

8.Object-Oriented Concepts:

* Classes and Objects:
  + - The project follows the fundamental principles of Object-Oriented Programming (OOP) by organizing code into classes and creating objects. Classes serve as blueprints for objects, defining their attributes and behaviour. In this project:
    - The ‘Product’ class is a template representing a
    - generic product with attributes such as ‘name’ and ‘price.’
    - The ‘IPhone’ class, a derived class, inherits from ‘Product’ and represents specific iPhones with additional attributes.
    - Inheritance:
    - Inheritance is a key OOP concept allowing a class to inherit attributes and methods from another class. In this project:
    - The ‘IPhone’ class inherits from the ‘Product’ class, enabling code reuse and establishing an "is-a" relationship.
* Polymorphism:
  + - Polymorphism is demonstrated through function override and function overloading:
* Function Override:
  + - The base class ‘Product’ declares virtual functions (‘display()’ and ‘display(double discount)’).
    - The derived class ‘IPhone’ provides specific implementations for these functions, overriding the base class versions. This enables the program to dynamically call the correct function based on the object's type.
* Function Overloading:
  + - The ‘Product’ class includes an overloaded ‘display’ function that takes a discount parameter.
    - The ‘IPhone’ class also overloads the ‘display’ function, providing a specific implementation for iPhones with a discount.
    - This demonstrates the ability to define multiple functions with the same name but different parameter lists, enhancing code flexibility.
* Encapsulation:
  + - Encapsulation involves bundling data and methods that operate on the data within a single unit. In this project:
    - The attributes ‘name’ and ‘price’ in both the ‘Product’ and ‘IPhone’ classes are encapsulated, limiting direct access, and ensuring controlled interactions.
* Templates:
  + - Templates allow code to be written without specifying the data type, enabling generic programming. In this project:
    - The ‘Product’ class is a template class, making it versatile and adaptable to various data types for the ‘price’ attribute.

9. Conclusion:

This project successfully applies OOP principles to create a modular and extensible iPhone Sales System. The code structure facilitates easy extension to accommodate additional product types or features. Thorough testing ensures the reliability and robustness of the program.