1. **Access Control and Getters:**

**Create the User class with private members for username and profile picture (string).**

**Implement public member functions for the constructor and getters (accessor methods) for username and profile picture.**

#include <string>

#include <iostream>

using namespace std;

class User {

private:

string username;

string profile\_picture;

public:

User(const string& user, const string& picture)

: username(user), profile\_picture(picture) {}

string getUsername() const {

return username;

}

string getProfilePicture() const {

return profile\_picture;

}

};

int main() {

// Create a new User object

User user("mouni", "/images/mouni.png");

cout << "Username: " << user.getUsername() << endl;

std::cout << "Profile Picture: " << user.getProfilePicture() << endl;

return 0;

}

OUTPUT:

Username: mouni

Profile Picture: /images/mouni.png

**2. Post Class and Display:**

**Create the derived class Post inheriting from User.**

**Add private members for post content (string) and timestamp (date/time format of your choice).**

**Implement a public member function getPostInfo that returns a formatted string** c**ontaining username, profile picture, post content, and timestamp.**

#include <string>

#include <iostream>

using namespace std;

// User class definition

class User {

private:

string username;

string profile\_picture;

public:

User(const string& user, const string& picture)

: username(user), profile\_picture(picture) {}

string getUsername() const {

return username;

}

string getProfilePicture() const {

return profile\_picture;

}

};

class Post : public User {

private:

string content;

string timestamp;

public:

// Constructor

Post(const string& user, const string& picture, const string& postContent, const string& time)

: User(user, picture), content(postContent), timestamp(time) {}

string getPostInfo() const {

return "Username: " + getUsername() + "\n" +

"Profile Picture: " + getProfilePicture() + "\n" +

"Content: " + content + "\n" +

"Timestamp: " + timestamp + "\n";

}

};

int main() {

// Create a new Post object

Post post("mouni", "/images/mouni.png", "This is my first post!", "2024-07-02 10:00:00");

// Display the post information

cout << post.getPostInfo();

return 0;

}

OUTPUT:

Username: mouni

Profile Picture: /images/mouni.png

Content: This is my first post!

Timestamp: 2024-07-02 10:00:00

**3. Basic Interaction Function:**

**Define a friend function basicInteract that takes two User objects (or derived class objects) as a**r**guments.**

**Inside the function, simply print a generic message like "User1 interacts with User2."**

#include <string>

#include <iostream>

using namespace std;

class User {

private:

string username;

string profile\_picture;

public:

User(const string& user, const string& picture)

: username(user), profile\_picture(picture) {}

string getUsername() const {

return username;

}

string getProfilePicture() const {

return profile\_picture;

}

friend void basicInteract(const User& user1, const User& user2);

};

class Post : public User {

private:

string content;

string timestamp;

public:

Post(const string& user, const string& picture, const string& postContent, const string& time)

: User(user, picture), content(postContent), timestamp(time) {}

string getPostInfo() const {

return "Username: " + getUsername() + "\n" +

"Profile Picture: " + getProfilePicture() + "\n" +

"Content: " + content + "\n" +

"Timestamp: " + timestamp + "\n";

}

};

void basicInteract(const User& user1, const User& user2) {

cout << user1.getUsername() << " interacts with " << user2.getUsername() << "." << endl;

}

int main() {

User user1("krishna", "/images/krishna.png");

User user2("tanuja", "/images/tanuja.png");

Post post1("mouni", "/images/mouni.png", "This is my first post!", "2024-07-02 10:00:00");

Post post2("abhi", "/images/abhi.png", "Hello, world!", "2024-07-03 11:00:00");

basicInteract(user1, user2);

basicInteract(post1, post2);

return 0;

}

OUTPUT:

krishna interacts with tanuja.

mouni interacts with abhi.

**4. Overloaded Interact Functions:**

**Create overloaded versions of the interact function:**

**likePost(User& user, Post& post): This function should print a message indicating the user liked the post.**

**followUser(User& follower, User& followed): This function should print a message indicating the user started following another user.**

#include <string>

#include <iostream>

using namespace std;

class Post;

class User {

private:

string username;

string profile\_picture;

public:

User(const string& user, const string& picture)

: username(user), profile\_picture(picture) {}

string getUsername() const {

return username;

}

string getProfilePicture() const {

return profile\_picture;

}

// Friend functions declarations

friend void likePost(User& user, Post& post);

friend void followUser(User& follower, User& followed);

};

// Post class definition

class Post {

private:

string content;

string timestamp;

User& author;

public:

Post(User& user, const string& postContent, const string& time)

: content(postContent), timestamp(time), author(user) {}

string getContent() const {

return content;

}

string getTimestamp() const {

return timestamp;

}

string getAuthorUsername() const {

return author.getUsername();

}

string getAuthorProfilePicture() const {

return author.getProfilePicture();

}

friend void likePost(User& user, Post& post);

};

void likePost(User& user, Post& post) {

cout << user.getUsername() << " liked the post by " << post.getAuthorUsername() << "." << endl;

}

void followUser(User& follower, User& followed) {

cout << follower.getUsername() << " started following " << followed.getUsername() << "." << endl;

}

int main() {

// Create users

User user1("MOUNI", "/images/MOUNI.png");

User user2("Darshan", "/images/Darshan.png");

Post post(user1, "This is my first post!", "2024-07-02 10:00:00");

likePost(user2, post);

followUser(user1, user2);

return 0;

}

OUTPUT:

Darshan liked the post by MOUNI.

MOUNI started following Darshan.

**Static Data Members**

• Variable declaration preceded by keyword 'static'

• Only one copy of static variable is created. All the objects share the same copy

• Initialized to zero when first object is created. No other initialization permitted.

• Should be defined outside the class definition after declaring them inside the class in this way - datatype classname :: varname

• They are normally used to maintain values that are common to the entire class, e.g., to keep a count of number of objects created.

#include <iostream>

class MyClass {

private:

static int counter; // Static variable to keep track of object instances

public:

MyClass() {

counter++;

} // Increment counter for each object creation

static int getCount() { // Static method to access and return the counter

return counter;

}

};

// Initialize static variable outside the class (required)

int MyClass::counter = 0; // Set initial value to e

int main() {

MyClass obj1;

MyClass obj2;

MyClass obj3;

std::cout << "Number of objects created: " << MyClass::getCount() << std::endl;

return 0;

}

OUTPUT:

Number of objects created: 3

#include <iostream>

class MyClass {

private:

static int counter; // Static variable to keep track of object instances

int count; // Non-static variable to keep track of instance count

public:

MyClass() {

count = 0; // Initialize count to 0 for each instance

counter++; // Increment static counter for each object creation

count++; // Increment instance count for this object

}

static int getCounter() {

return counter; // Return static counter

}

int getCount() {

return count; // Return instance count for this object

}

};

// Initialize static variable outside the class (required)

int MyClass::counter = 0;

int main() {

MyClass obj1;

MyClass obj2;

MyClass obj3;

std::cout << "Number of objects created: " << MyClass::getCounter() << std::endl;

std::cout << "obj1 count: " << obj1.getCount() << std::endl;

std::cout << "obj2 count: " << obj2.getCount() << std::endl;

std::cout << "obj3 count: " << obj3.getCount() << std::endl;

return 0;

}

OUTPUT:

Number of objects created: 3

obj1 count: 1

obj2 count: 1

obj3 count: 1

Distance Converter:

Create a class named DistanceConverter. Include the following static methods:

**convertMilesToKm(double miles): Converts miles to kilometers (1 mile = 1.60934 kilometers).**

**convertKmToMiles(double kilometers): Converts kilometers to miles. In your main function, prompt the user for a distance and a unit (miles or kilometers). Use the appropriate static method from the DistanceConverter class to perform the conversion and display the result to the user.**

**Math Utility Class:**

**Design a class named MathUtil. Include static methods for basic mathematical operations:**

#include <iostream>

using namespace std;

class MathUtil {

public:

// Static method to add two integers

static int add(int a, int b) {

return a + b;

}

// Static method to subtract two integers

static int subtract(int a, int b) {

return a - b;

}

// Static method to multiply two integers

static int multiply(int a, int b) {

return a \* b;

}

// Static method to divide two integers with error handling for division by zero

static double divide(int a, int b) {

if (b == 0) {

cout << "Error: Division by zero." << endl;

return 0; // or handle error as needed

}

return static\_cast<double>(a) / b;

}

};

int main() {

int num1, num2;

char operation;

// Prompt user for input

cout << "Enter first number: ";

cin >> num1;

cout << "Enter second number: ";

cin >> num2;

cout << "Enter operation (+, -, \*, /): ";

cin >> operation;

// Perform calculation based on operation

switch (operation) {

case '+':

cout << "Result: " << MathUtil::add(num1, num2) << endl;

break;

case '-':

cout << "Result: " << MathUtil::subtract(num1, num2) << endl;

break;

case '\*':

cout << "Result: " << MathUtil::multiply(num1, num2) << endl;

break;

case '/':

cout << "Result: " << MathUtil::divide(num1, num2) << endl;

break;

default:

cout << "Invalid operation." << endl;

break;

}

return 0;

}

**convertToEur(double amount): Converts an amount from the base currency (USD) to EUR based on the exchange rate.**

**convertFromEur(double amount): Converts an amount from EUR to the base currency (USD). In your main function, prompt the user for an amount and a conversion direction (USD to EUR or EUR to USD). Use the appropriate static method from the CurrencyConverter class to perform the conversion and display the result.**

#include <iostream>

using namespace std;

class CurrencyConverter {

public:

// Static method to convert USD to EUR (assuming exchange rate 1 USD = 0.85 EUR)

static double convertToEur(double amount) {

return amount \* 0.85;

}

// Static method to convert EUR to USD

static double convertFromEur(double amount) {

return amount / 0.85;

}

};

int main() {

double amount;

char direction;

// Prompt user for input

cout << "Enter amount: ";

cin >> amount;

cout << "Enter conversion direction (USD to EUR: 'u', EUR to USD: 'e'): ";

cin >> direction;

// Perform conversion based on direction

switch (direction) {

case 'u':

cout << amount << " USD is " << CurrencyConverter::convertToEur(amount) << " EUR." << endl;

break;

case 'e':

cout << amount << " EUR is " << CurrencyConverter::convertFromEur(amount) << " USD." << endl;

break;

default:

cout << "Invalid conversion direction." << endl;

break;

}

return 0;

}

**Add(int a, int b): Adds two integers.**

**subtract(int a, int b): Subtracts two integers.**

**multiply(int a, int b): Multiplies two integers.**

**divide(int a, int b) (optional): Divides two integers with error handling for division by zero. In your main function, prompt the user for two numbers and an operation (+, -, \*, or /). Use the corresponding static method from the MathUtil class to perform the calculation and display the result.**

**Simple Currency Converter:**

**Create a class named CurrencyConverter. Define a static variable named exchangeRate (e.g., USD to EUR exchange rate). Implement static methods:**

#include <iostream>

using namespace std;

class DistanceConverter {

public:

// Static method to convert miles to kilometers

static double convertMilesToKm(double miles) {

return miles \* 1.60934;

}

// Static method to convert kilometers to miles

static double convertKmToMiles(double kilometers) {

return kilometers / 1.60934;

}

};

int main() {

double distance;

string unit;

// Prompt user for distance and unit

cout << "Enter distance: ";

cin >> distance;

cout << "Enter unit (miles or kilometers): ";

cin >> unit;

// Perform conversion based on the unit

if (unit == "miles") {

double km = DistanceConverter::convertMilesToKm(distance);

cout << distance << " miles is " << km << " kilometers." << endl;

} else if (unit == "kilometers") {

double miles = DistanceConverter::convertKmToMiles(distance);

cout << distance << " kilometers is " << miles << " miles." << endl;

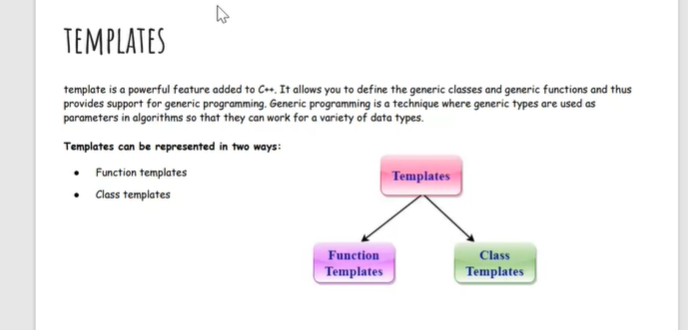
} else {

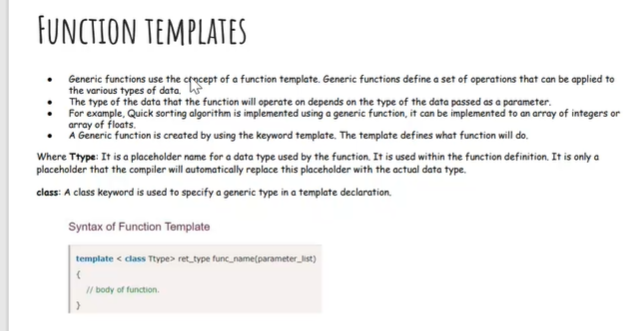
cout << "Invalid unit." << endl;

}

return 0;

}





#include<iostream>

using namespace std;

template< class T>T add(T&a,T&b)

{

T result = a+b;

return result;

}

int main()

{

int i=2;

int j=3;

float m=2.3;

float n=1.2;

cout<<"Addition of i and j is :"<<add(i,j);

cout<<'\n';

cout<<"Addition of m and n is :"<<add(m,n);

return 0;

}

OUTPUT:

Addition of i and j is :5

Addition of m and n is :3.5

**FUNCTION TEMPLATES WITH MULTIPLE PARAMETERS:**

#include<iostream>

using namespace std;

template< class x,class y>void fun(x a,y b)

{

std::cout<<"value of a is:"<<a<<std::endl;

std::cout<<"value of b is:"<<b<<std::endl;

}

int main()

{

fun(15,12.3);

return 0;

}

OUTPUT:

value of a is:15

value of b is:12.3