Лабораторная работа №5
Наследование и полиморфизм

Задания

Наследование

5.1.1 Создайте класс Person со строковыми полями имени, фамилии, отчества. Создайте соответствующие сеттеры, геттеры и конструктор класса.

5.1.2 Создайте дочерний от класса Person класс Student с дополнительными полями номера зачетной книжки и года поступления. Создайте соответствующие сеттеры, геттеры и конструктор класса. Конструктор дочернего класса должен быть основан на наследовании от конструктора базового класса.

5.1.3 Создайте дочерний от класса Person класс Teacher с дополнительными полем должности. Создайте соответствующие сеттеры, геттеры и конструктор класса. Конструктор дочернего класса должен быть основан на наследовании от конструктора базового класса.

5.1.4 Создайте функцию ShowName(Person* person), принимающую указатель на базовый класс. Функция принимает на вход объект типа Person (или объект производного класса), и выводит на экран фамилию, имя и отчество. Пример вывода:

```
Безбородов Николай Александрович
```

5.1.5 Создайте функцию main, в которой создайте объект базового класса Person. Вызовите функцию ShowName() с экземпляром базового класса. Создайте объект дочернего класса Student и вызовите функцию ShowName() с экземпляром дочернего класса. Создайте объект дочернего класса Teacher и вызовите функцию ShowName() с экземпляром дочернего класса. Убедитесь, что функция одинаково работает с объектами всех трёх классов и корректно выводит данные на экран.

5.1.6 Убедитесь, что нет утечек памяти.

5.1.7 Нарисуйте UML-диаграмму классов для созданных классов

Рефакторинг с выделением базового класса

Работа разработчика неразрывно связана с умением читать чужой код, находить в нём ошибки проектирования или ошибки работы, и исправлять их.

5.2.1 Избавьтесь от дублирования кода с помощью выделения базового класса в следующем коде:

```
////////////////////// Post - пост в блоге пользователя с платным аккаунтом

class Post
{
    string _title;
    string _text;

public:
    void SetTitle(string title);
    void SetText(string text);

    string GetTitle();
    string GetText();

    Post(string title, string text);
};

void Post::SetTitle(string title)
{
    _title = title;
}
```

```
void Post::SetText(string text)
{
    _text = text;
}

string Post::GetTitle() { return _title; }
string Post::GetText() { return _text; }

Post::Post(string title, string text)
{
    SetTitle(title);
    SetText(text);
}


////////////////////// User - обычный пользователь

class User
{
    int _id;
    string _login;
    string _password;

    void SetId(int id);

public:
    void SetLogin(string login);
    void SetPassword(string password);

    int GetId();
    string GetLogin();
    string GetPassword();

    User(int id, string login, string password);
    bool IsCorrectPassword(string password);
};

void User::SetId(int id)
{
    _id = id;
}

void User::SetLogin(string login)
{
    _login = login;
}

void User::SetPassword(string password)
{
    _password = password;
}

int User::GetId() { return _id; }
string User::GetLogin() { return _login; }
string User::GetPassword() { return _password; }

User::User(int id, string login, string password)
{
    SetId(id);
    SetLogin(login);
```

```
    SetPassword(password);
}

bool User::IsCorrectPassword(string password)
{
    return (password == _password);
}

////////////////////// Paid User - пользователь с платным аккаунтом

class PaidUser
{
    int _id;
    string _login;
    string _password;
    Post* _posts;
    int _postsCount;

    void SetId(int id);

public:
    void SetLogin(string login);
    void SetPassword(string password);
    void SetPosts(Post* posts, int postsCount);

    int GetId();
    string GetLogin();
    string GetPassword();
    Post* GetPosts();
    int GetPostsCount();

    PaidUser(int id, string login, string password, Post* posts, int postsCount);
    PaidUser(int id, string login, string password);
    bool IsCorrectPassword(string password);
};

void PaidUser::SetId(int id)
{
    _id = id;
}

void PaidUser::SetLogin(string login)
{
    _login = login;
}

void PaidUser::SetPassword(string password)
{
    _password = password;
}

void PaidUser::SetPosts(Post* posts, int postsCount)
{
    if (postsCount < 0)
    {
        throw exception("Posts count must be more than 0");
    }
    _posts = posts;
    _postsCount = postsCount;
}
```

```
int PaidUser::GetId() { return _id; }
string PaidUser::GetLogin() { return _login; }
string PaidUser::GetPassword() { return _password; }
Post* PaidUser::GetPosts() { return _posts; }
int PaidUser::GetPostsCount() { return _postsCount; }

PaidUser::PaidUser(int id, string login, string password, Post* posts, int postsCount)
{
    SetId(id);
    SetLogin(login);
    SetPassword(password);
    SetPosts(posts, postsCount);
}

PaidUser::PaidUser(int id, string login, string password):
           PaidUser(id, login, password, nullptr, 0)
{
}

bool PaidUser::IsCorrectPassword(string password)
{
    return (password == _password);
}
```

5.2.2 Избавьтесь от дублирования кода в следующих функциях с помощью создания одной общей функции, работающей с объектами обоих классов через указатель на базовый класс.

```
// Метод принимает массив всех зарегистрированных бесплатных пользователей,
// ищет пользователя с введенным именем и паролем.
// Если такого пользователя нет, возвращается nullptr.
// Если пользователь есть, но пароль не подошел, то генерируется исключение.
User* Login(User** users, int usersCount, string enteredLogin, string enteredPassword)
{
    for(int i = 0; i < usersCount; i++)
    {
           if (users[i]->GetLogin() == enteredLogin)
           {
                  if (users[i]->IsCorrectPassword(enteredPassword))
                  {
                         return users[i];
                  }
                  else
                  {
                         throw exception("Uncorrect password");
                  }
           }
    }
    return nullptr;
}

// Метод принимает массив всех зарегистрированных платных пользователей,
// ищет пользователя с введенным именем и паролем.
// Если такого пользователя нет, возвращается nullptr.
// Если пользователь есть, но пароль не подошел, то генерируется исключение.
PaidUser* Login(PaidUser** paidUsers, int paidUsersCount, string enteredLogin, string
enteredPassword)
{
    for (int i = 0; i < paidUsersCount; i++)
```

```
    {
        if (paidUsers[i]->GetLogin() == enteredLogin)
        {
            if (paidUsers[i]->IsCorrectPassword(enteredPassword))
            {
                return paidUsers[i];
            }
            else
            {
                throw exception("Uncorrect password");
            }
        }
    }
    return nullptr;
}
```

5.2.3 Замените в функции main() использование ранее существовавших функций на вызов новой общей функции Login(). Массив пользователей программы при этом объедините в единый массив всех пользователей через массив указателей на базовый класс:

```
void main()
{
    User** users = new User * []
    {
        new User(100000, "morkovka1995", "1995morkovka"),
        new User(100001, "ilon_mask", "X æ A-12"),
        new User(100002, "megazver", "password"),
        new User(100003, "yogurt", "ksTPQzSu"),
    };

    PaidUser** paidUsers = new PaidUser * []
    {
        new PaidUser(200000, "TheKnyazz", "JHPzPGFG"),
        new PaidUser(200001, "system_exe", "UgfkDGmU"),
        new PaidUser(200002, "RazorQ", "TBgRnbCP"),
        new PaidUser(200003, "schdub", "CetyQVID"),
    };

    string login = "megazver";
    string password = "password";
    User* loginedUser = Login(users, 4, login, password);

    cout << "Signed in as: " << loginedUser->GetLogin() << endl;

    login = "system_exe";
    password = "UgfkDGmU";
    PaidUser* loginedPaidUser = Login(paidUsers, 4, login, password);

    cout << "Signed in as: " << loginedPaidUser->GetLogin() << endl;

    for (int i = 0; i < 4; i++)
    {
        delete users[i];
    }
    delete[] users;

    for (int i = 0; i < 4; i++)
    {
        delete paidUsers[i];
```

```
    }
    delete[] paidUsers;
}
```

5.2.4 Убедитесь, что после выделения базового класса и общей функции, код работает верно. ***Если код работает неправильно, найдите ошибку и исправьте её***. Результатом работы функции main() должен быть следующий текст:

```
Signed in as: megazver
Signed in as: system_exe
```

Также убедитесь в отсутствии утечек памяти.

5.2.5 Подсчитайте, сколько строк кода было до рефакторинга (устранения дублирования) и после. Рассчитайте на сколько процентов уменьшился исходный код программы благодаря наследованию.

5.2.6 Добавьте в базовый класс в метод SetLogin() проверку, чтобы логин не содержал знаки пунктуации `{`, `}`, `<`, `>`, `@`, `#`, `$`, `%`, `^`, `:`, `*`. Убедитесь, что проверка логина работает как для базового класса, так и для дочернего – то есть, механизм наследования работает.

5.2.7 Нарисуйте UML-диаграмму классов для созданных классов.

Полиморфизм

Следующая программа должна реализовать систему скидок в приложении для магазинов. Скидки действуют на товары определенной категории. Есть скидки процентные (скидка предоставляется в заданном размере от стоимости товара определенной категории), и есть скидки сертификатные (скидка предоставляется на заданную в сертификате сумму, но не превышающую стоимость товара).

5.3.1 Создайте класс товара Product с полями названия, категории и стоимости. Сделайте соответствующие геттеры, сеттеры и конструктор класса. Для поля категории создайте перечисление CategoryType. Стоимость не может быть отрицательной, а также превышать 100 000.

5.3.2 Создайте абстрактный базовый класс DiscountBase. В классе создайте:

- закрытое поле _category (категория товара, на которую предоставляется скидка).
- закрытый сеттер SetCategory() для категории товара.
- открытый геттер GetCategory() для категории товара.
- чисто виртуальный открытый метод Calculate(Product* product): double – который в будущем для дочерних классов будет реализовывать расчет стоимости товара после применения скидки.
- защищенный конструктор по категории товара, на которую предоставляется скидка.

5.3.3 Создайте дочерний класс PercentDiscount. Помимо отнаследованных от DiscountBase полей и методов, создайте в классе:

- закрытое поле _percent (размер скидки в процентах).
- открытый сеттер SetPercent() для размера скидки – значение должно быть в диапазоне от 0 до 100.
- открытый геттер GetPercent() для размера скидки.
- открытый конструктор класса по категории товара и размеру скидки – реализация конструктора дочернего класса должна обращаться к защищенному конструктору базового класса.
- переопределение открытого метода Calculate(Product* product): double. Метод должен провести сравнение категории товара с категорией скидки. Если категории совпадают, то производится расчет стоимости товара с учетом скидки в процентах. Метод возвращает новую стоимость товара (то есть, если скидка составляет 15%, то метод должен вернуть 85% стоимости товара). Если категории товара и скидки не совпадают, то метод возвращает полную стоимость товара.

5.3.4 Создайте второй дочерний класс CertificateDiscount. Помимо отнаследованных от DiscountBase полей и методов, создайте в классе:

- закрытое поле _amount (размер сертификата в рублях).

- открытый сеттер SetAmount() для размера сертификата – значение должно быть в диапазоне от 0 до 10 000.
- открытый геттер GetAmount() для размера сертификата.
- открытый конструктор класса по категории товара и размеру сертификата – реализация конструктора дочернего класса должна обращаться к защищенному конструктору базового класса.
- переопределение открытого метода Calculate(Product* product): double. Метод должен провести сравнение категории товара с категорией скидки. Если категории совпадают, то производится расчет стоимости товара с учетом скидки по сертификату. Метод возвращает новую стоимость товара. То есть, если сертификат превышает стоимость товара, то метод вернет 0 – товар полностью оплачен сертификатом, при этом размер сертификата уменьшается на стоимость товара (чтобы сертификат нельзя было применить дважды к разным товарам). Если сертификат меньше стоимости товара, то метод возвращает разницу стоимости товара и размера сертификата, при этом размер сертификата уменьшается до 0. Если категории товара и скидки не совпадают, то метод возвращает полную стоимость товара.

5.3.5 Создайте функцию void ShowCheckWithDiscount(DiscountBase* discount, Product* products, int productsCount), принимающую указатель на базовый класс скидки, а также массив товаров. Функция должна перебрать в цикле все товары массива, и рассчитать для них скидку через экземпляр discount. Для каждого товара на экран выводится его название, старая стоимость и новая стоимость. После цикла программа выводит на экран общую стоимость всех товаров с учетом скидки. Пример вывода на экран (скидка применена для первых двух товаров в размере 25%):

```
TV LG49N000    Old Cost: 40000    New Cost: 30000
Micromax Q1    Old Cost: 2000     New Cost: 1500
Pantum M650    Old Cost: 8000     New Cost: 8000
HP LasetJet    Old Cost: 11000    New Cost: 11000
Full Cost with Discount: 50500
```

5.3.6 Создайте функцию main, в которой создайте объекты обоих дочерних классов. Вызовите функцию ShowCheckWithDiscount() для каждого из созданных объектов дочерних классов. Убедитесь, что для каждого объекта вызывается собственная реализация полиморфного метода соответствующего класса. В частности, убедитесь, что при применении скидки по сертификату после её применения к одному товару, её размер уменьшается на стоимость товара.

Убедитесь, что в программе нет утечек памяти.

5.3.7 Убедитесь, что компилятор не позволяет создать экземпляры абстрактного класса DiscountBase, а только указатели на него.

5.3.8 Нарисуйте UML-диаграмму классов для созданных классов

Список литературы

1. **Наследование** / Metanit.com: сайт о программировании // https://metanit.com/cpp/tutorial/5.10.php
2. **Виртуальные функции и их определение** / Metanit.com: сайт о программировании // https://metanit.com/cpp/tutorial/5.11.php
3. **Абстрактные классы** / Metanit.com: сайт о программировании // https://metanit.com/cpp/tutorial/5.12.php
4. **Наследование в C++: beginner, intermediate, advanced** / Хабр: площадка блогов о программировании // https://habr.com/ru/post/445948/
5. Лафоре Р. **Объектно-ориентированное программирование на Си++.** Глава 9 Наследование
6. Лафоре Р. **Объектно-ориентированное программирование на Си++.** Глава 11 Виртуальные функции