Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**о работе по информатике**

Семестр: 2

На тему: «ООП. Классы и объекты. Инкапсуляция»

Выполнил студент ИВТ-22-2б:

Казанцев Антон Васильевич

(дата, подпись)

Проверила:

Полякова Ольга Андреевна

(дата, подпись)

Пермь 2023

**Задание**

1. Реализовать определение нового класса. Для демонстрации работы с объектами написать главную функцию. Продемонстрировать разные способы создания объектов и массивов объектов.

2. Структура-пара – структура с двумя полями, которые обычно имеют имена first и second. Требуется реализовать тип данных с помощью такой структуры. Во всех заданиях должны присутствовать:

a. Метод инициализации Init (метод должен контролировать значения аргументов на корректность);

b. Ввод с клавиатуры Read;

c. Вывод на экран Show.

3. Реализовать внешнюю функцию make\_тип(), где тип – тип реализуемой структуры. Функция должна получать значения для полей структуры как параметры функции и возвращать структуру как результат. При передаче ошибочных параметров следует выводить сообщение и заканчивать работу.

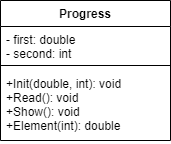
*Вариант 15:*

Элемент арифметической прогрессии aj вычисляется по формуле: aj=a0·rj Реализовать вычисление j-го элемента прогрессии.

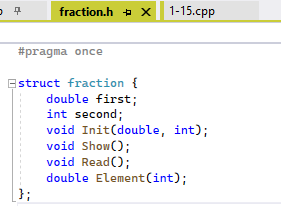
**Анализ задачи**

1. Метод Init – инициализация переменных
2. Метод Show – вывод значений переменных
3. Метод Read – ввод значений переменных с клавиатуры
4. Метод Power – вычисление j-го элемента прогрессии

**UML-Диаграмма**



**Программное решение**





#include "fraction.h"

#include <iostream>

using namespace std;

fraction makeFraction(double, int);

int main() {

system("chcp 1251 > NULL");

//

fraction A;

A.Init(3.0, 2);

A.Show();

cout << A.Element(5) << '\n';

/\*другие способы:

fraction B;

B.Read();

B.Show();

int j1;

cout << "Введите номер какого элемента вы хотите получить " << '\n';

cout << "j1: ";

cin >> j1;

cout << B.Element(j1) << '\n';

fraction\* X = new fraction;

X->Init(2.0, 5);

X->Show();

fraction mas[3];

for (int i = 0; i < 3; i++) {

mas[i].Read();

}

for (int i = 0; i < 3; i++) {

mas[i].Show();

}

for (int i = 0; i < 3; i++) {

cout << " Степень? " << '\n';

cout << "j:";

cin >> j1;

mas[i].Element(j1);

}

fraction\* p\_mas = new fraction[3];

for (int i = 0; i < 3; i++) {

p\_mas[i].Read();

}

for (int i = 0; i < 3; i++) {

p\_mas[i].Show();

}

for (int i = 0; i < 3; i++) {

cout << " Степень? " << '\n';

cout << "j:";

cin >> j1;

cout << mas[i].Element(j1) << '\n';

}

double y; int z;

cout << "first:";

cin >> y;

cout << "second:";

cin >> z;

fraction F = makeFraction(y, z);

F.Show();

cout << F.Element(j1) << '\n';\*/

return 0;

}

fraction makeFraction(double f, int s) {

fraction t;

t.Init(f, s);

return t;

}



**Контрольные вопросы**

*1.Что такое класс?*

**Класс** – абстрактным тип данных, определяемый пользователем. Класс представляет собой модель реального объекта в виде данных и функций для работы с ними. Позволяет представить часть кода в виде одного объекта. В объектно-ориентированном программировании программа представляется в виде множества объектов, что позволяет регулировать отношения между ними.

Пример:

class Pair

{

private:

int first;

double second;

public:

int get\_first() { return first; }

double get\_second() { return second; }

};

*2. Что такое объект (экземпляр) класса?*

**Объект(экземпляр)** – отдельный элемент класса, который характеризуется полями класса.

Пример:

int main() {

Pair h; //Экземпляр класса (тип данных - название класса)

return 0;

}

*3. Как называются поля класса?*

**Поля** класса или **атрибуты**, составляют структуру класса.

Элементы int first; double second; из вопроса 1).

*4. Как называются функции класса?*

**Методы** класса – функции для работы с объектами класса.

Элементы int get\_first() { return first; } double get\_second() { return second; } из вопроса 1).

*5. Для чего используются спецификаторы доступа?*

Спецификаторы доступа **управляют видимостью элементов класса**. Действие любого спецификатора распространяется до следующего спецификатора или до конца класса. За счёт спецификаторов доступа осуществляется инкапсуляция.

Спецификаторы бывают private, public из вопроса 1), а также protected.

*6. Для чего используется спецификатор public?*

**Открытый тип доступа.** Поля и методы имеют глобальную область видимости и могут использоваться в любой части программы. Время жизни до конца исполнения программы.

Пример:

public:

//К методам можно обращаться в любой части программы

int get\_first() { return first; }

double get\_second() { return second; }

*7. Для чего используется спецификатор private?*

**Закрытый тип доступа.** Локальная область видимости (доступ к полям и методам только внутри класса) и время жизни - во время компиляции кода описания класса. Доступ вне класса через методы типа public.

Пример:

private:

//Обращение к полям только с помощью public-методов

int first;

double second;

*8. Если описание класса начинается со спецификатора class, то какой спецификатор доступа будет использоваться по умолчанию?*

Если при описании класса используется слово class, то по умолчанию все методы и поля класса будут скрытыми – **private**.

Пример:

class Pair

{ //По умолчанию private

int first;

double second;

int get\_first() { return first; }

double get\_second() { return second; }

};

*9. Если описание класса начинается со спецификатора struct, то какой спецификатор доступа будет использоваться по умолчанию?*

Если при описании класса используется слово struct, то все поля и методы по умолчанию будут общедоступными – **public**.

Пример:

struct Pair

{ //По умолчанию public

int first;

double second;

int get\_first() { return first; }

double get\_second() { return second; }

};

*10. Какой спецификатор доступа должен использоваться при описании интерфейса класса? Почему?*

Интерфейс класса описывается после спецификатора **public** для того, чтобы его методы были доступны для использования в других частях кода, и класс не был изолированной структурой данных.

Пример:

class Pair

{

private:

int first;

double second;

public: //Интерфейс

int get\_first() { return first; }

double get\_second() { return second; }

};

*11. Каким образом можно изменить значения атрибутов экземпляра класса?*

Если атрибуты имеют доступ типа public, то изменить поля можно, обратившись напрямую **по имени**, **с помощью указателя** или **по ссылке**.

Пример:

class Pair

{

public:

int first;

double second;

};

int main() {

Pair p;

p.first = 20; //Доступ по имени

Pair\* p1 = new Pair;

p1->second = 18.7; //Доступ с помощью указателя

Pair& p2 = p;

p2.first = 25; //Доступ по ссылке

return 0;}

Изменить информацию о содержимом полей, описанных после спецификатора private можно только с помощью специальных public-методов которые называются **модификаторами** (сеттерами).

Пример:

class Pair

{

private:

int first;

double second;

public:

//Модификатора (сеттеры)

void set\_first(int f) { first = f; }

void set\_second(double s) { second = s; }

};

int main() {

Pair p;

p.first = 20; //Доступ по имени невозможен

p.set\_first(20); //Возможен доступ с помощью модификатора

return 0;

}

*12. Каким образом можно получить значения атрибутов экземпляра класса?*

Если атрибуты имеют доступ типа public, то получить значение поля можно, обратившись напрямую **по имени**, **с помощью указателя** или **по ссылке**.

Пример:

class Pair

{

public:

int first;

double second;

};

int main() {

Pair p;

cout << p.first;//Доступ по имени

Pair\* p1 = new Pair;

cout << p1->second; //Доступ с помощью указателя

Pair& p2 = p;

cout << p2.first; //Доступ по ссылке

return 0;

}

Получить информацию о содержимом полей, описанных после спецификатора private можно только с помощью специальных методов, которые называются **селекторами** (геттерами).

Пример:

class Pair

{

private:

int first;

double second;

public:

//Селекторы (геттеры)

int set\_first() { return first; }

double set\_second() { return second; }

};

int main() {

Pair\* p1 = new Pair;

cout << p1->second; //Доступ с помощью указателя невозможен

cout << p1->get\_first();

return 0;

}

*13. Класс описан следующим образом:*

struct Student

{

string name;

int group;

. . . . . .

};

*Объект класса определен следующим образом:*

Student\* s = new Student;

*Как можно обратиться к полю name объекта s?*

Так как у типа данных struct по умолчанию public доступ, то к полям можно обратиться напрямую. Так как экземпляр класса задан через указатель, то к нему можно обратиться следующим образом

int main() {

Student\* s = new Student;

s->name = "Ivan";

}

*14. Класс описан следующим образом:*

struct Student

{

string name;

int group;

. . . . . .

};

*Объект класса определен следующим образом:*

Student s;

*Как можно обратиться к полю name объекта s?*

Так как у типа данных struct по умолчанию public доступ, то к полям можно обратиться напрямую. Так как экземпляр класса задан по имени, то к нему можно обратиться следующим образом

int main() {

Student s;

s.name = "Ivan";

}

*15. Класс описан следующим образом:*

class Student

{

string name;

int group;

. . . . . .

};

*Объект класса определен следующим образом:*

Student\* s = new Student;

*Как можно обратиться к полю name объекта s?*

Так как у типа данных class по умолчанию private доступ, то напрямую к полям обратиться нельзя. Требуются специальные методы типа public, но они здесь не описаны и область public вообще отсутствует.

int main() {

Student\* s = new Student;

s->name = "Ivan"; //Ошибка при обращении с помощью указателя

}

*16. Класс описан следующим образом:*

class Student

{

string name;

int group;

public:

. . . . . .

};

*Объект класса определен следующим образом:*

Student s;

*Как можно обратиться к полю name объекта s?*

Так как у типа данных class по умолчанию private доступ, то к полям обратиться нельзя. Требуются специальные методы типа public (например, модификаторы), которые можно описать в области public:

class Student

{

string name;

int group;

public:

void set\_name(string s) { name = s; }

};

int main() {

Student s;

s.set\_name("Ivan"); //Обращение к методы по имени

}

*17. Класс описан следующим образом:*

class Student

{

public:

char\* name;

int group;

. . . . . .

};

*Объект класса определен следующим образом:*

Student\* s = new Student;

*Как можно обратиться к полю name объекта s?*

Так как у типа данного класса доступ к полям public, то к ним можно обратиться напрямую. Так как экземпляр класса задан через указатель, то к нему можно обратиться следующим образом:

int main() {

Student\* s = new Student;

char arr1[] = "Ivan";

s->name = arr1;

}