МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І.І. МЕЧНИКОВА
ФАКУЛЬТЕТ МАТЕМАТИКИ, ФІЗИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА МАТЕМАТИЧНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

КУРСОВА РОБОТА

з дисципліни

«Об`єктно Орентоване Програмування»

на тему:

«Турклуб»

студента І курсу

групи ___2_____

спеціальності _____КІ_____

____Зіма Микита Євгенович_____

(Прізвище, ім’я та по батькові)

Одеса - 2020

Введение

В последнее время в сфере современных технологий, а именно IT, как главном двигателе прогресса. Появляются все новые и новые методы и концепции развития, что уследить за всеми практически невозможно. Все они Они определяют последовательность действий и способы взаимодействия заказчиков, постановщиков задач, разработчиков и программистов. Но все равно, одним из главных и наиболее распространённых, являются парадигмы ООП.

Объектно-ориентированное программирование (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

Идеологически ООП — подход к программированию как к моделированию информационных объектов, решающий на новом уровне основную задачу структурного программирования: структурирование информации с точки зрения управляемости, что существенно улучшает управляемость самим процессом моделирования, что, в свою очередь, особенно важно при реализации крупных проектов.

Предметом

работы является написание программы, для выполнения поставленной задачи

Задачи, решение которых необходимо для достижения поставленной цели:

1. Собрать необходимую информацию
2. Проанализировать собранную информацию
3. Продемонстрировать а программе основные парадигмы ООП

Глава 1.
Объектно-ориентированное программирование

Языки высокого уровня (Algol 68, Fortran, PL/1 и т.д.) облегчили трудоемкую работу по созданию машинного кода, который стал делать компилятор. Программы стали короче и понятнее.

Потом задачи усложнились, и программы снова стали слишком громоздкими. Программы стали разбивать на процедуры или функции, которые решают свои задачи. Написать, откомпилировать и отладить маленькую функцию можно легко и быстро, а потом собрать все функции воедино.

Такое программирование стало процедурным программированием и стало парадигмой.

Появились библиотеки процедур и функций. Как сделать программу нагляднее, какую часть кода надо выделить в отдельную процедуру и как лучше связать их между собой, т.е. как выявить структуру программы.

Появились новые языки.

Удачная структура данных может облегчить их обработку. Некоторые удобно представить в виде массива, а другие в виде стэка или дерева. Рост сложности и размеров программ потребовал развития структурирования данных и появления новых типов данных, которые могут определяться программистом.

Идея объединения данных и всех процедур их обработки в единый модуль -- парадигма модульного программирования.

Сначала такой модуль был более или менее случайным набором данных и подпрограмм. В такие модули собирали подпрограммы, которые, как казалось, скорее всего будут изменяться совместно. Программы составлялись из отдельных модулей. Эффективность таких программ тем выше, чем меньше модули зависят друг от друга.
Необходимо четко отделить процедуры, которые будут вызываться другими модулями -- открытые процедуры. Отделять их будем от вспомогательных, которые обрабатывают данные, заключенные в модуль. Данные, занесенные в модуль также делятся на открытые и закрытые. Удобно разбить программу на модули. Таким образом, чтобы она превратилась в совокупность взаимодействующих объектов.
Так возникло ООП.

Это современная парадигма программирования.

Все программы состоят из двух частей (описание и сама программа). Любая программа может быть концептуально организована либо вокруг её кода “кодовое воздействие на данные”, либо вокруг данных “управляемый данными доступ к коду”. При первом процедурном подходе программу определяет последовательность операторов её кода. Второй подход организует программу вокруг данных, т.е. вокруг объектов и набора хорошо организованных интерфейсов.

ООП -- это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса; а классы образуют иерархию наследования.

1).ООП использует в качестве базовых элементов объекты, а не алгоритмы.

2).Каждый объект будет экземпляром какого-либо определенного класса.

3).Классы реализованы (организованы) иерархически.

Основное достоинство ООП -- сокращение числа межмодульных связей, изменение объемов информации, которая передается между модулями и возможность повторного использования кодов.

Недостатки -- снижение быстродействия из-за более сложной организации программы.

1. Инкапсуляция. Сокрытие данных. Для работы с данными определяются свойства, методы и события.

Инкапсуляция -- ограничение прав доступа. Объекты в задаче сохраняют конкретные данные, тип которых определяется полями класса. Каждый объект в задаче играет роль, определяет его “поведение”. То, что может делать каждый объект (кроме сохранения значений своих полей) задается элементами функции. Особенностью класса является инкапсуляция одной конструкции, как данных, так и методов функций, которые обрабатывают эти данные, контролируемым образом. Это защита данных и операций от неконтролируемого доступа.

Так элементы private оказываются автоматически доступными только для методов самого класса, но сокрытыми для другой части программы.

Элементы public определяют интерфейс класса с другими частями программы и другими классами.

2. Наследование. Объекты могут получать свойства и методы других объектов (предков). Класс-предок называется базовым, класс-потомок - производным. Наследники получают все свойства и методы предков, которые могут быть изменены, а также могут обладать собственными методами или свойствами Например:
В моей программе самый яркий пример наследования это классы Tourist и класс Guide, являющийся наследником вышепредставленного класса.

3.Полиморфизм. Методы различных объектов могут иметь одинаковые имена, но отличаться по своему содержанию.

Полиморфизм -- это положение теории типов, согласно которому имена (переменных) могут обозначать объекты разных, но имеющих общего родителя, классов. Следовательно, любой объект, обозначаемый полимофным именем, может по-своему реагировать на некий общий набор операций. Один интерфейс -- много методов.
Может понадобиться программа, которой требуется три типа СТЭКОВ (для хранения целых, вещественных чисел, для хранения символов).
Алгоритм, который реализует все стэки, будет один и тот же, хотя хранимые даны различны.

Полиморфизм позволяет определить общий для всех типов данных набор стэковых функций, использовать одно и то же имя. Дальнейшее -- забота компилятора -- выбрать специфический метод для использования в каждой конкретной ситуации.
Применение полиморфизма позволяет решить проблему добавления новых функциональностей. При этом существующий программный код не подвергается никаким изменениям. Мы добавим новый код к уже существующему.

4.Абстракция. Моделирование объектов в программе. Свойства и методы.
Абстрагирование -- это метод решения сложных задач. Описывая поведение сложного объекта, мы выделяем только те стороны, которые нас интересуют с точки зрения решаемых задач. Т.е. строим его приближенную модель. Модель не может описать реальный объект полностью. Мы выделяем только те характеристики, которые важны для задачи. Нам надо абстрагироваться от несущественных деталей объекта.

Уровень абстракции.

Надо выбрать правильный уровень абстракции, чтобы не получилась слишком простая модель, когда потеряется что-то важное. Нельзя выбирать слишком высокий уровень абстракции, так как он дает слишком приблизительное упрощенное описание объекта.

Слишком низкий уровень абстракции делает объект слишком сложным, перегруженным деталями.

Абстрагирование -- это взгляд на объект ни как он есть на самом деле, а с точки зрения наблюдателя и интересующих его характеристик данного объекта.

Характеристики -- это свойства объекта, т.е. то, что, касается его состояния или определяет его поведение, выделяется в единую программную единицу или некий абстрактный класс. Объектно-ориентированное проектирование основано на абстрактном объединении объектов, решении одних и тех же задач, в классы. В виде класса можно представить любую общепринятую абстракцию данных.

Абстрагирование и инкапсуляция дополняют друг друга. Абстрагирование направлено на наблюдение за объектом, а инкапсуляция занимается внутренним устройством объекта -- это сокрытие некоторых элементов абстракции (которые не затрагивают существенных характеристик объекта как целого).

Надо выделить две части в описании абстракции. Первая -- это интерфейс (взаимодействие). Это основные характеристики состояния, поведения объекта. Он описывает внешнее поведение объекта типа class. Вообще описывает абстракцию поведения всех объектов данного класса. В этой части собрано все, что касается взаимодействия этого объекта с любыми другими объектами.

Вторая -- реализация, представляет собой недоступные извне элементы реализации абстракции (внутренняя организация абстракции и механизмы реализации её поведения). Скрывает все дета

Наследование классов -- это основа ООП, позволяет увеличить масштабируемость программ, реализовать повторное использование кода. Программа, которая использует объекты базового класса может быть расширена без изменения старого кода путем построения производных классов от базового. Использование новых объектов в производных классах. Это достигается соглашением о наследовании: объекты производного класса содержат все элементы базового класса (поля и методы), как если бы эти элементы были описаны в самом производном классе.

Отпадает необходимость многократного переписывания одних и тех же определений базового класса, появляется возможность пользоваться ими, как они

есть. В базовом классе надо определить какие права доступа мы предоставляем производным класса.

Производный класс может наследовать базовый как private, как public и как protected. Это влияет на возможность будущего расширения иерархии классов и интерпретации целей самого наследования.

Глава 2

Разработка программы “Турклуб”

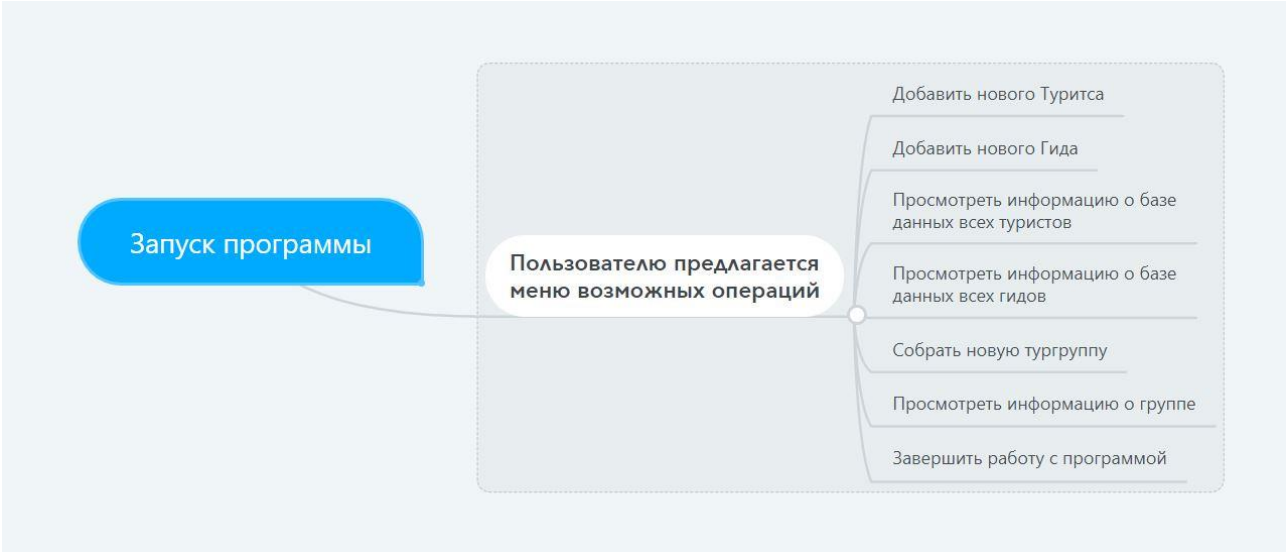
1. Требования к программе: Створення ієрархії класів на тему «Турклуб»

Создать классы: «Турист» (ФИО, возраст, пол, список интересов, время в которое человек может поехать – начальная возможная дата и конечная) и наследник «Руководитель группы» (с дополнительной информацией – какими экспедициями может руководить). Создать класс, описывающий общую турпоездку или поход, а также классы наследники с конкретными видами туризма – пеший поход, велосипедный, альпинизм и т.д. (на выбор студента). Для каждой турпоездки, хранить продолжительность (в днях) и место. Также хранить специфическую информацию для отдельных поездок – например, высота горы для альпинизма.

В основном проекте создать возможность вводить и редактировать данные о туристах и турпоездках, а также возможность собирать группы – функция должна принимать минимальное и максимальное количество человек в группе и подбирать группу туристов с одинаковым интересом к поездке, среди них должен быть руководитель с возможностью руководить такого вида поездками, а также должна быть выбрана начальная и конечная дата, подходящая всем.

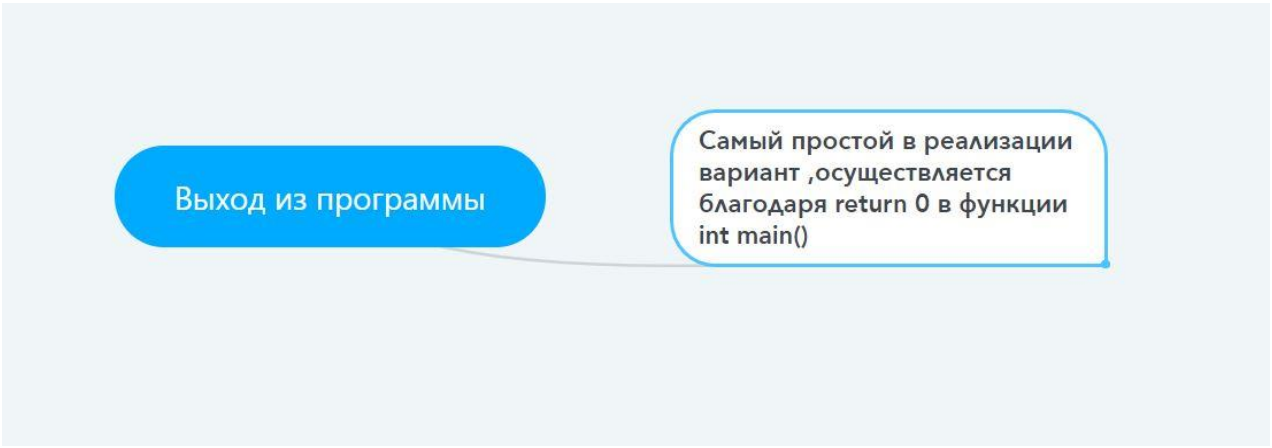
Алгоритм работы программы

Программа предназначена для хранения и обработки данных о туристах, гидах и турпоездках. В своей работе, я ориентровался на работу реально работающих программ, написанных для туристических компаний и агентств.

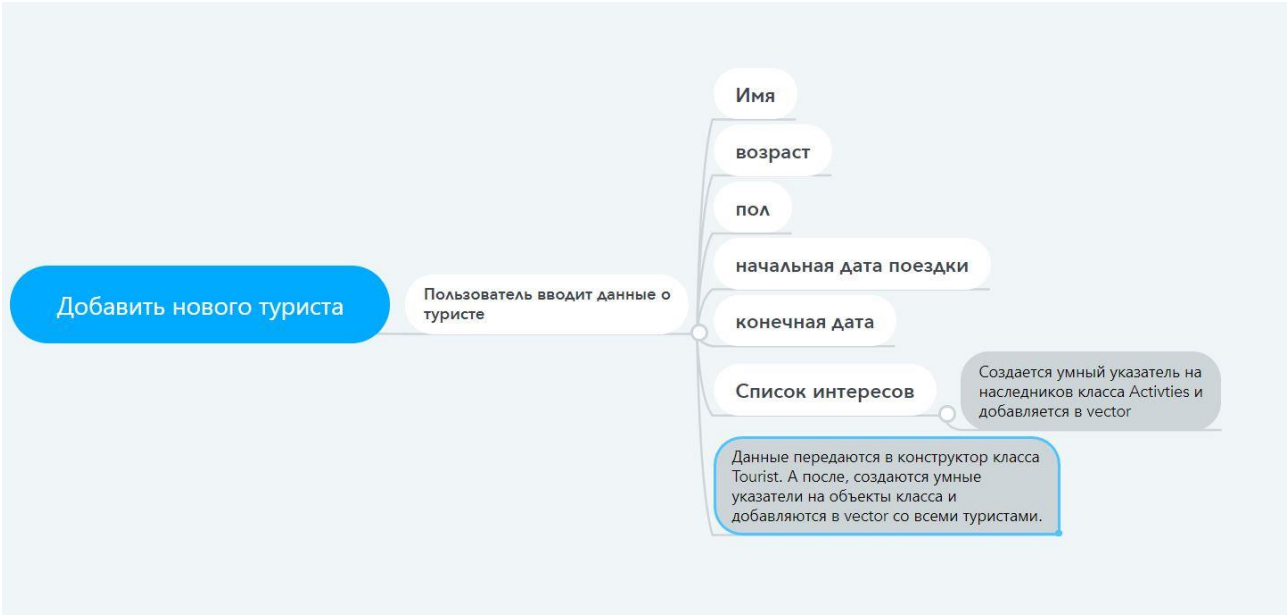


Далее рассмотрим пример работы каждого из вариантов:

1. Завершить работу с программой.

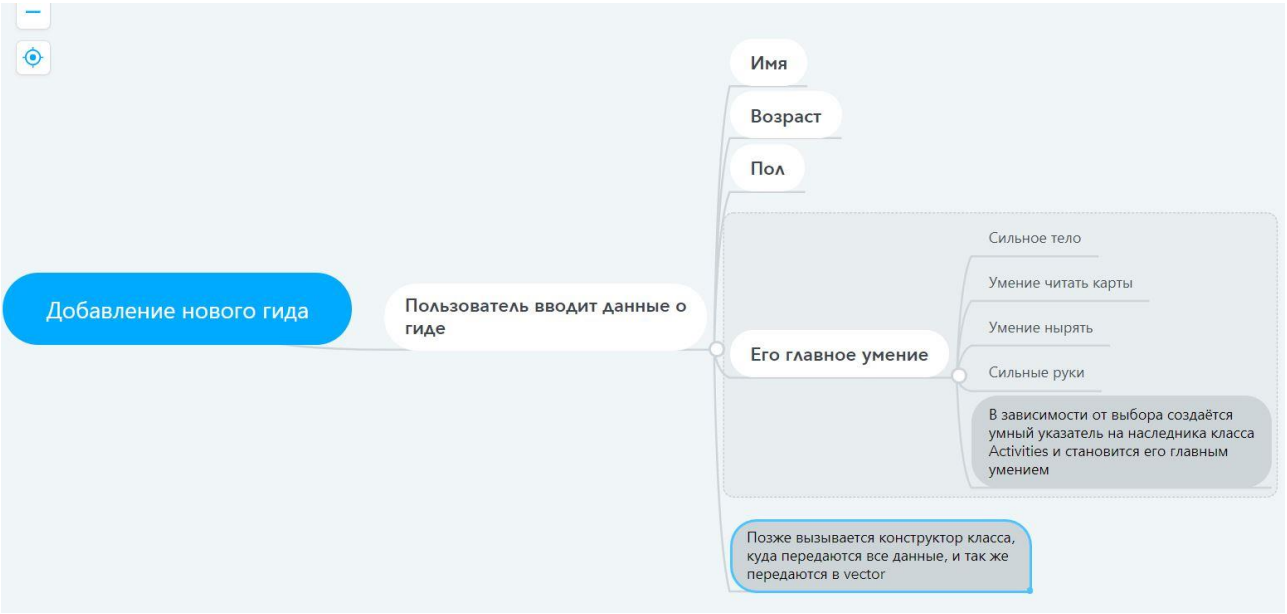


2. Добавить нового туриста

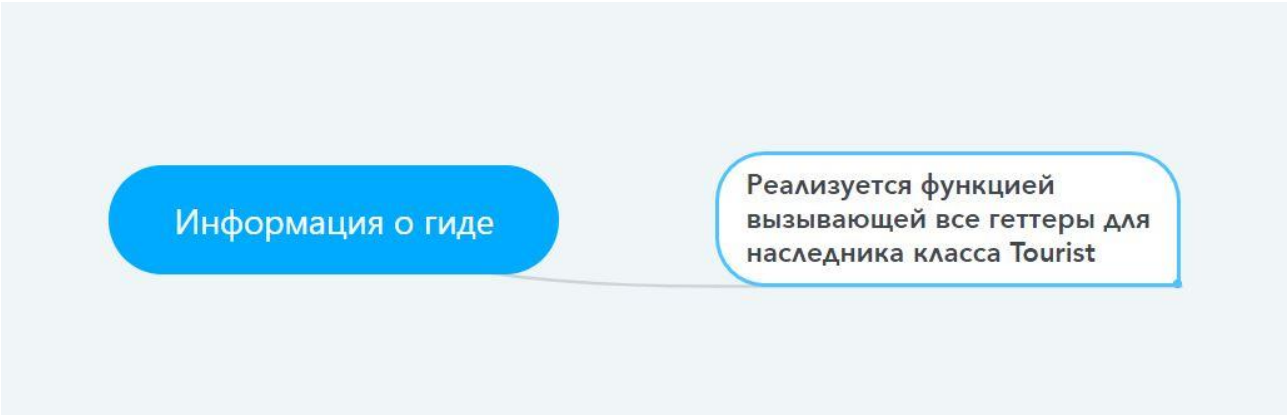


3.

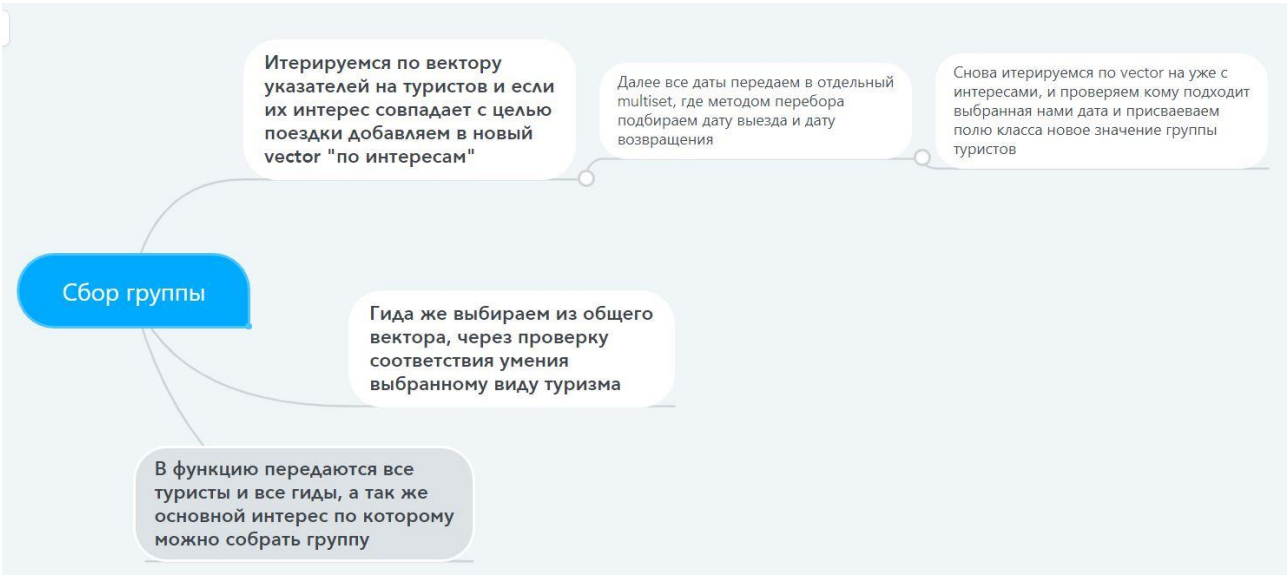
4. Добавление нового гида



5. Просмотр информации о всех гидах



6. Самая главная функция класса Group, создание группы



-Проверка по интересам

```
int c = 0;
for (auto i : group)
{
    if (i->HasInterest(activity)) {
        c++;
        this->activity = activity;
        by_inter.push_back(i);
    }
    else if (c == 0) {
        cout << "No one interested in such type of trip\n";
        return false;
    }
}
```

-Поиск подходящих дат

```
set<Date> repeats_start;
multiset<Date>::iterator start = start_dates.begin();
for (; start != start_dates.end(); ++start)
{
    if (repeats_start.count(*start) > 0) continue;
    else {
        if ((counter = start_dates.count(*start)) > 1)
        {
            repeats_start.insert(*start);
        }
    }
}
for (auto i : repeats_start)
{
    this->start = i;
    break;
}
```

-Поиск по датам

```
for (auto i : guides)
{
    if (i.getSkill()!=this->activity)
    {
        guide = i;
        this->guide = guide;
        cout << "Guide was found\n";
        break;
    }
    else {
        cout << "There are no guides with such skill\n";
        return false;
    }
}
```

-Поиск и добавление гида

```
    for (auto i : guides)
    {
        if (i.getSkill()!=this->activity)
        {
            guide = i;
            this->guide = guide;
            cout << "Guide was found\n";
            break;
        }
        else {
            cout << "There are no guides with such skill\n";
            return false;
        }
    }
```

Список источников:

-https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5

-https://devcolibri.com/%D1%87%D1%82%D0%BE-%D1%82%D0%B0%D0%BA%D0%BE%D0%B5-%D0%BE%D0%BE%D0%BF-%D0%B8-%D1%81-%D1%87%D0%B5%D0%BC-%D0%B5%D0%B3%D0%BE-%D0%B5%D0%B4%D1%8F%D1%82/

-https://tproger.ru/translations/oop-principles-cheatsheet/