C++

# Урок 16

# Что такое виртуальная функция?

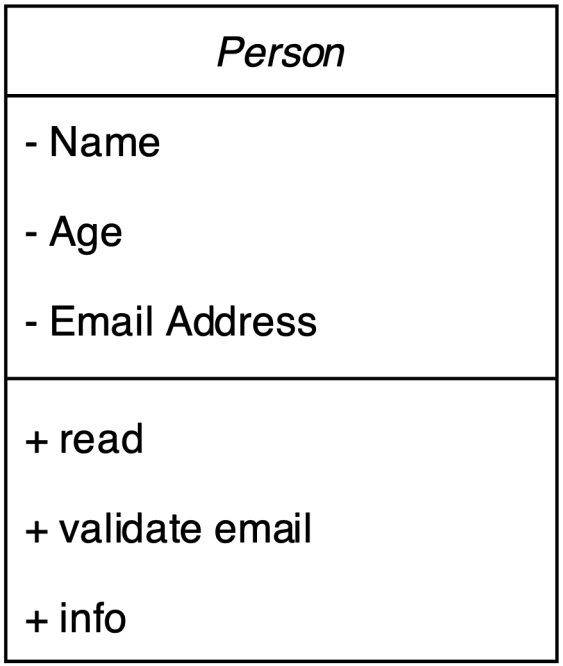
# Когда она применяется?

# Что такое абстракция?

# За счет чего она работает?

# Задача

**Задача:** Реализуйте класс Person по диаграмме на след странице.




## Особенности:

- Поле age сделайте динамическим.
- read - функция ввода email
- validate email - функция валидации email. Учтите, что внутри строки должны быть символы "@" и ”.” ,а также то, что точка стоит после @.
- info - функция вывода полей

| *Person* |
| --- |
| - Name<br>- Age<br>- Email Address |
| + read<br>+ validate email<br>+ info |

## Особенности:

- Поле age сделайте динамическим.
- read - функция ввода email
- validate email - функция валидации email. Учтите, что внутри строки должны быть символы "@" и ”.” ,а также то, что точка стоит после @.
- info - функция вывода полей

```
int main()
{
    Person Alex("Alex", 18);
    Alex.info();
}
```

# Не забыли про деструктор?

# Что он должен удалять?

```
int main()
{
    Person Alex("Alex", 18);
    Person Melmon = Alex;
    Alex.info();
}
```

Напишем такой main()

# Какие ошибки выдает консоль?

```
int main()
{
    Person Alex("Alex", 18);
    Person Melman = Alex;
    Alex.info();
    Melman.info();
}
```

Напишем такой main()

# Ошибки еще есть?

```
int main()
{
    Person Alex("Alex", 18);
    shN(Alex);
    Alex.info();
}
```

Напишем такой main()

```cpp
int main()
{
    Person Alex("Alex", 18);
    shN(Alex);
    Alex.info();
}
```

```cpp
string getN()
{
    return name;
}
```

```cpp
void shN(Person test){
    cout <<test.getN()<<endl;
}
```

Добавим геттер и функцию

# Будут ли ошибки?

# С чем они связвны?

# Конструктор копирования

# конструктор копирования

**Конструктор копирования** – это специальный конструктор, который позволяет получить идентичный к заданному объект. То есть, с помощью конструктора копирования можно получить копию уже существующего объекта.

**Конструктор копирования** еще называется инициализатором копии (copy initializer). Конструктор копирования должен получать входным параметром константную ссылку (**&**) на объект такого же класса.

# конструктор копирования

Случаи вызова конструктора копирования:

# конструктор копирования

Случаи вызова конструктора копирования:

- В момент объявления нового объекта и его инициализации данными другого объекта с помощью оператора **=**

# конструктор копирования

Случаи вызова конструктора копирования:

- В момент объявления нового объекта и его инициализации данными другого объекта с помощью оператора **=**
- Когда нужно передать объект в функцию как параметр-значение. В этом случае создается полная копия объекта.

# конструктор копирования

Случаи вызова конструктора копирования:

- В момент объявления нового объекта и его инициализации данными другого объекта с помощью оператора **=**
- Когда нужно передать объект в функцию как параметр-значение. В этом случае создается полная копия объекта.
- Когда нужно вернуть объект из функции по значению. В этом случае также создается полная копия объекта.

# конструктор копирования

Целесообразное использование:

**Конструктор копирования** необходимо использовать в тех классах, где осуществляется динамическое выделение памяти для данных.

Если в классе нету динамического выделения памяти для данных, то конструктор копирования можно не использовать. В этом случае побитового копирования (по умолчанию) достаточно для корректной работы класса.

**Исключение:** если при инициализации объекта другим объектом нужно установить некоторые специальные условия копирования.

# конструктор копирования

```
Person(const Person &test)
{
    //тело конструктора
}
```

Синтаксис

# конструктор копирования

```
Person(const Person &test)
{
    this->name = test.name;              //копируем значение поля
    this->email = test.email;            //копируем значение поля
    this->age = new int(*test.age); //выделили память под переменную и занесли значение
}
```

Синтаксис

# Почему константная ссылка?

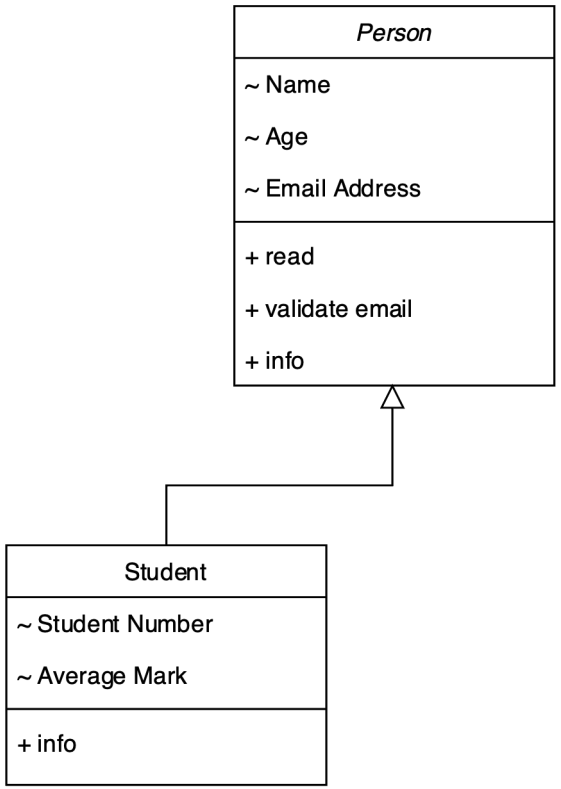
# конструктор копирования

Особенности копирования:

- Если в классе **не объявлен** конструктор копирования, то используется конструктор копирования, который автоматически генерируется компилятором. Этот конструктор копирования реализует побитовое копирование для получения копии объекта.
- **Побитовое копирование** подойдет для классов, в которых нет динамического выделения памяти. Однако, если в классе есть динамическое выделение памяти (класс использует указатели), то побитовое копирование приведет к тому, что указатели обоих объектов будут указывать на один и тот же участок памяти.

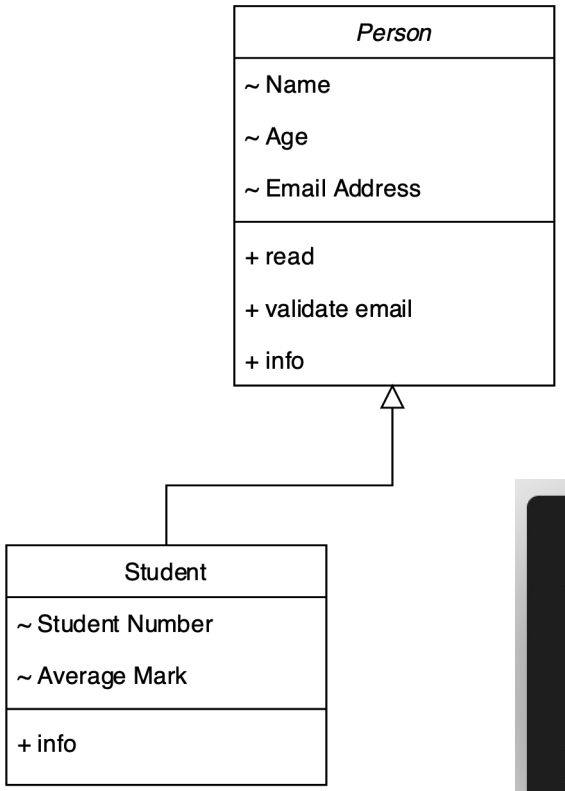
# Задача

**Задача:** Реализуйте класс Student по диаграмме на след странице.




## Особенности:

- Student Number – целое число, которое > 0
- Average Mark – вещественное число > 0
- info - функция вывода полей




## Особенности:

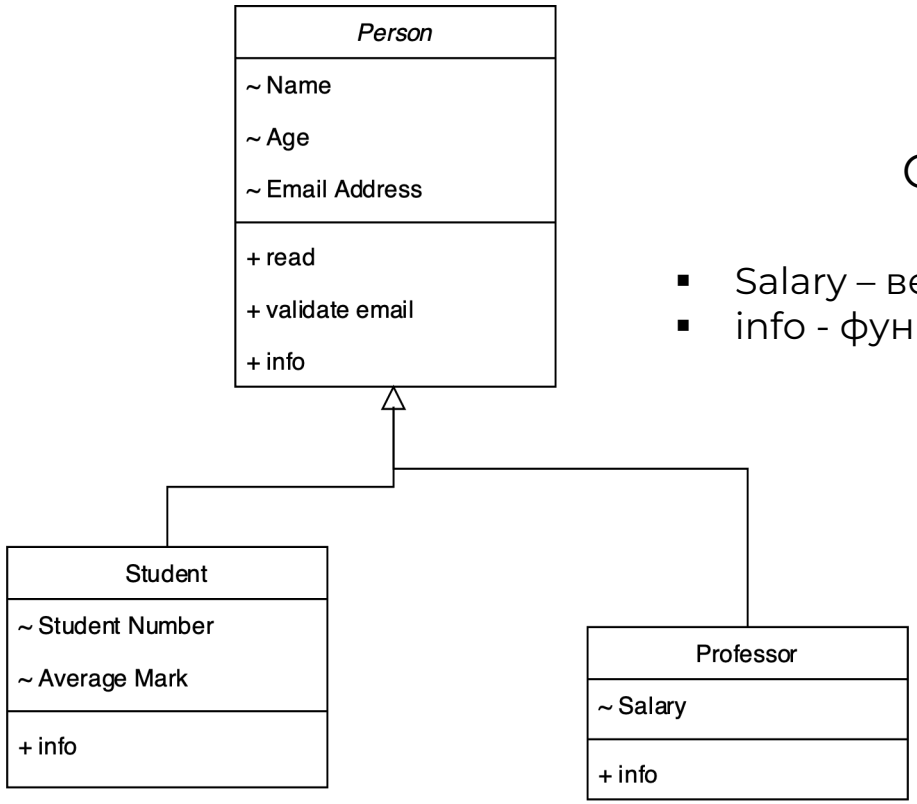
- Student Number – целое число, которое > 0
- Average Mark – вещественное число > 0
- info - функция вывода полей

```
int main()
{
    Person *Melman = new Student;
    Melman->info();
    delete Melman;
}
```

# Задача

**Задача:** Реализуйте класс Professor по диаграмме на след странице.

В main реализуйте вектор указателей на объекты и вводите пока не введено сообщение “exit”.




## Особенности:

- Salary – вещественное число, которое > 0
- info - функция вывода полей

# Задача

**Задача:** напишите ф-ию count, которая считает кол-во студентов и преподавателей.

# Перегрузка операторов

# Какие операторы бывают?

# Какие операторы бывают?

# Дружественная ф-ия

# Дружественные функции

**Дружественные функции** - это функции, которые не являются членами класса, однако имеют доступ к его закрытым членам - переменным и функциям, которые имеют спецификатор private/protected.

Для определения дружественных функций используется ключевое слово **friend**.

# Дружественные функции

**Дружественные функции** - это функции, которые не являются членами класса, однако имеют доступ к его закрытым членам - переменным и функциям, которые имеют спецификатор private/protected.

Для определения дружественных функций используется ключевое слово **friend**.

```
friend тип результата имя функции(параметры) {}
```