|  |  |
| --- | --- |
| ас | Министерство образования и науки Российской Федерации  Федеральное государственное автономное образовательное учреждение высшего образования «Уральский федеральный университет имени первого Президента России Б.Н.Ельцина» (УрФУ)  Институт экономики и управления  Кафедра Анализа систем и принятия решений |

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель междисциплинарного

проектирования Чернильцев А.Г.

Члены комиссии \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к междисциплинарному проекту

**Основы объектно-ориентированного программирования**

Студент: Корнев Дмитрий Евгеньевич \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(ФИО) (Подпись)

Группа: ЭУ-293631 \_

Екатеринбург

2021

|  |  |
| --- | --- |
|  | Министерство образования и науки Российской Федерации  Федеральное государственное автономное образовательное учреждение высшего образования «Уральский федеральный университет имени первого Президента России Б.Н.Ельцина» (УрФУ)  Институт экономики и управления  Кафедра Анализа систем и принятия решений |

**Задание**

**на междисциплинарный проект**

Студент Корнев Дмитрий Евгеньевич

Группа ЭУ-293631

специальность/направление подготовки 38.03.05 Бизнес-информатика

1. Тема междисциплинарного проекта

Основы объектно-ориентированного программирования

2. Содержание проекта, в том числе состав графических работ и расчетов

Введение

1. Обзор понятий, средств и методов объектно-ориентированного анализа и программирования

2. Объектно-ориентированный анализ и моделирование с помощью UML

3. Разработка приложения. Создание и использование коллекции List<T>

Заключение

Список использованных источников

3. Дополнительные сведения

Оформлять пояснительную записку в соответствии с требованиями «Положения о выпускной

квалификационной работе (уровень бакалавр) по направлениям подготовки»

4. План выполнения проекта:

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование элементов  проектной работы | Сроки | Примечания | Отметка о выполнении |
| Выполнение 1 раздела |  |  |  |
| Выполнение 2 раздела |  |  |  |
| Выполнение 3 раздела |  |  |  |
| Заключение |  |  |  |
| Список использованных источников |  |  |  |
| Оформление, выполнение презентации |  |  |  |
| Нормоконтроль |  |  |  |
| Защита |  |  |  |

Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ А. Г. Чернильцев /

# Введение

# 1. Обзор понятий, средств и методов объектно-ориентированного анализа и программирования

Темой теоретической части моего проекта по модулю был обзор основ объектно-ориентированного подхода к программированию, освещение его базовых принципов и особенностей. В этой главе мной будут описаны такие черты ООП, как базовый и дочерний класс, наследование, конструкторы классов, абстрактные классы и методы.

## Основные принципы объектно-ориентированного программирования

Объектно-ориентированное программирование (ООП) задумывалось как подход к программированию, где программные единицы представляют собой объекты классов – абстрактных типов данных, что позволяет гораздо легче реализовывать процесс моделирования систем любой сложности. Такой подход с новой стороны открывает решение задачи структурного программирования – построения программ с помощью отдельных независимых модулей.

ООП позволило разработчикам глубже погружаться в задачу, которая перед ними ставится при создании программного продукта, с необходимой точностью декомпозируя предметную область на отдельные классы и работая уже с их объектами. Отсюда вытекает удобство управления проектом – чётко выстроенная структура с помощью объектно-ориентированного анализа позволяет практически независимо от каких-либо других программных модулей вносить изменения в бизнес-логику приложений и делать исправления в коде.

Если речь идёт об объектно-ориентированном подходе к программированию, следует использовать свойственную этому подходу терминологию. Например, переменные, относящиеся к некоторому классу, в разных ситуациях называются его полями или свойствами. Функции, также относящиеся к классу и использующие (чаще всего, но необязательно) в своей логике его поля/свойства, называются его методами.

Поскольку ООП – это целая философия восприятия программы как информационной модели, безусловно, у него существуют свои основополагающие принципы.

* Абстракция – формализация предмета из предметной области в виде программного класса;
* Инкапсуляция – безопасное сокрытие большей части бизнес-логики и внутренних данных приложения «под капотом», внутри закрытых полей и методов классов. Для внешнего воздействия разработчиком оставляется лишь набор методов и свойств, отвечающих за запуск выполнения определённой задачи. Порождает такие понятия, как «публичный» («открытый»), «приватный» («закрытый») метод и другие модификаторы доступа;
* Наследование – возможность создавать на основе определенного класса новые классы, с переносом всех ранее описанных свойств и методов. Этот принцип позволяет избежать дублирования данных и порождает такие понятия, как «базовый класс» и «дочерний класс»;
* Полиморфизм – черта ООП, позволяющая обрабатывать одной точкой входа разные типы данных при разных условиях.

## Особенности наследования

Наследование в объектно-ориентированном программировании позволяет повторно использовать в дочернем классе (подклассе) компоненты родительского класса (суперкласса), избегая дублирования их объявлений и реализаций. Все поля, свойства и методы суперкласса, указанные как наследуемые, переносятся в подкласс с необходимым уровнем доступа.

Интерфейсы – это структуры, состоящие из абстрактных методов (в некоторых языках, например, C#, могут также состоять из свойств). Интерфейсы представляют собой своего рода «контракт» с дочерним классом, «обязывающий» программиста реализовать внутри него описанные в интерфейсе методы. Таким образом, классы, представляющие собой абсолютно разные предметы из предметной области, но реализующие методы одного и того же родительского интерфейса, могут быть вызваны в коде, обращающемся к этим методам, наравне друг с другом.

Пример наследования класса и двух интерфейсов на языке C#:

interface IShootable {

    int Damage { get; set; }

    int Shoot();

    int Reload();

}

interface IStorable {

    int TakenSpace { get; set; }

    int Store();

}

class Item {

    public string Name { get; private set; }

    public Player Owner { get; set; }

}

class Gun : Item, IStorable, IShootable {

    public int Damage { get; set; }

    public int TakenSpace { get; set; }

    public Gun(int damage, Player owner) {

        Name = "Gun";

        Owner = owner;

    }

    int IShootable.Shoot() { /\* ...  \*/ }

    int IShootable.Reload() { /\* ...  \*/ }

    int IStorable.Store() { /\* ...  \*/ }

}

В разных языках программирования определены разные условия наследования. Например, языки C++ и Python допускают множественное наследование, то есть порождение дочернего класса от нескольких «предков». В это же время C# позволяет унаследовать дочерний класс только от одного базового, но при этом допускается множественное наследование интерфейсов.

В некоторых языках существуют ключевые слова, уведомляющие о том, что от класса невозможно наследовать подклассы – например, sealed в C# и final в Java.

## Конструкторы базовых, производных классов

Конструктор класса – это метод, вызываемый при создании объекта класса. Если он имеет входные параметры, то данные из них могут быть обработаны во время создания. В C-подобных языках конструктор чаще всего объявляется без типа и с тем же именем, что и имя класса (если класс называется A, то его конструктор в С# должен быть объявлен как public A()), в JavaScript для объявления используется ключевое слово constructor, а в Python, например, для конструктора в каждом создаваемом классе зарезервирован встроенный метод \_\_init\_\_(self).

В большинстве случаев дочерние классы имеют отличные от «предков» входные параметры конструктора, так как с расширением функционала класса требуется обрабатывать новые данные. Объектно-ориентированный подход был бы бесполезен, если бы в конструкторе подкласса невозможно было передать данные в суперкласс. У каждого языка программирования, поддерживающего ООП, свои способы сделать это, хоть в большинстве своём и схожие друг с другом.

В языке C++ передача данных в родительский конструктор может быть объявлена прямо в заголовке дочернего конструктора:

class Parent {

public:

    Parent(int a) { /\* ... \*/}

};

class Child : public Parent {

public:

    Child(int a, int b) : Parent(a) { /\* ... \*/ }

}

На C# есть возможность передавать данные в родительский класс через ключевое слово base:

class Parent {

    public Parent(int a) { /\* ... \*/}

}

class Child : Parent {

    public Child(int a, int b) { base(a) }

}

Дочерний класс может не иметь своего конструктора, и тогда он унаследуется из родительского.

## Абстрактные классы и методы

В некоторых случаях у разработчика нет, и не может быть возможности определить функционал некоторых методов у базового класса, так как его абсолютно точно должны будут определять подклассы. Оставлять тело метода пустым в таких ситуациях было бы нерационально, так как это негативно скажется на производительности и в целом нарушит принципы объектно-ориентированного подхода. Также в ряде ситуаций уже существующий функционал методов необходимо переопределить, так как подклассы могут нуждаться в изменении алгоритма их работы. Для решения этой проблемы в языках, поддерживающих ООП, реализованы различные средства абстракции для классов и методов.

Абстрактный метод (также иногда называется «чистый виртуальный» метод) – метод класса, реализация которого на момент объявления отсутствует. Класс, содержащий такие методы, принято считать также абстрактным. Абстрактный метод может быть реализован в дочерних классах.

Абстрактные методы входят в подмножество виртуальных методов, то есть, не каждый виртуальный метод – абстрактный. Виртуальным методом в целом называется такой метод, который подлежит переопределению в подклассах, а так как отсутствие реализации для компилятора/интерпретатора также является реализацией (но пустой, нулевой), то абстрактный метод также считается виртуальным.

Наглядным примером «нулевой реализации» абстрактного метода может послужить способ его объявления в С++:

virtual void AbstractMethod() = 0;

В C# абстрактный метод не имеет тела, а содержащий его класс, как и сам метод, должен быть помечен ключевым словом abstract. В то же время реализация метода в дочернем классе помечается ключевым словом override:

public abstract class AbstractClass {

    /\* ... \*/

    public abstract int ReturnSomething();

}

public class ImplementingClass : AbstractClass {

    /\* ... \*/

    public override int ReturnSomething() { return 0; }

}

# 2. Объектно-ориентированный анализ и моделирование с помощью UML

# 3. Разработка приложения. Создание и использование коллекции List<T>

# Заключение

# Список использованных источников