Лекция 1. Принципы объектно-ориентированного программирования

Казахский национальный университет имени аль-Фараби

Кафедра компьютерных наук

Объектно-ориентированное программирование

Лектор: старший преподаватель, PhD, Кенжебек Ержан Ғалымжанұлы

Цель лекции

Цель лекции

Сформировать представление о причинах появления объектно-ориентированного программирования, недостатках процедурного подхода и основных понятиях ООП: объект, класс, наследование, полиморфизм.

Ожидаемые результаты

1. Объяснить недостатки структурного программирования
2. Описать объектно-ориентированный подход к построению программ
3. Назвать основные характеристики объектно-ориентированных языков
4. Понимать назначение языка UML

План лекции

- Для чего нужна ООП
- Деление на функции и недостатки структурного программирования
- Глобальные данные и функции
- Моделирование реального мира с помощью ООП
- Характеристики (основные элементы) объектно-ориентированных языков (объекты, классы, полиморфизм)
- Универсальный язык моделирования (UML)

Для чего нужна ООП?

- Развитие объектно-ориентированного метода обусловлено **ограниченностью других методов программирования**, разработанных ранее. Чтобы лучше понять и оценить значение ООП, необходимо разобраться, в чем состоит эта ограниченность и каким образом она проявляется в традиционных языках программирования.

Деление на функции

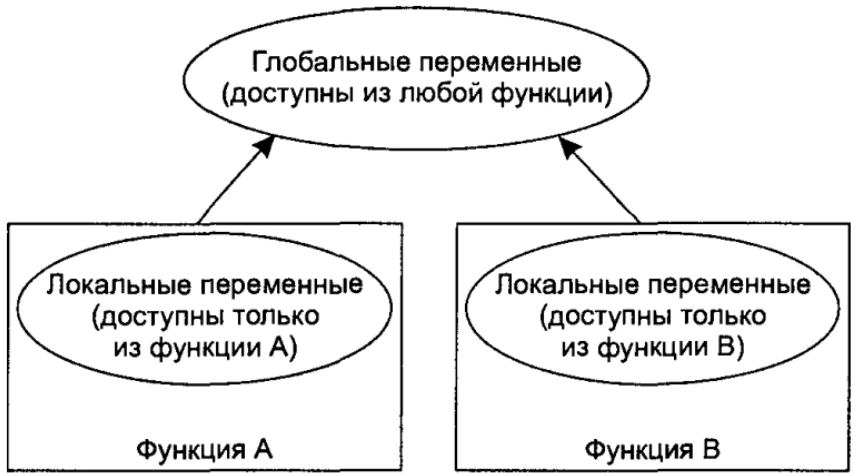
- Когда размер программы велик, список команд становится слишком громоздким. Очень небольшое число программистов способно удерживать в голове более 500 строк программного кода, если этот код не разделен на более мелкие логические части (функции). Идея: объединение несколько функции в модуль.

- Деление программы на функции и модули является основой структурного программирования. Структурное программирование представляет собой нечто не вполне определенное, однако в течение нескольких десятков лет, пока не была разработана концепция объектно-ориентированного программирования, оно оставалось важным способом организации программ

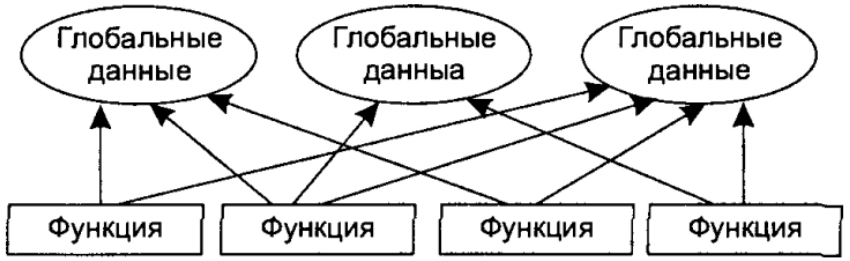
Недостатки структурного программирования

- Как бы эффективно ни применялся структурный подход, он не позволяет в достаточной степени упростить большие сложные программы (неудачные проекты в производстве)
- Существует две основные проблемы.
  - Первая заключается в **неограниченности доступа функций к глобальным данным**.
  - Вторая состоит в том, что разделение данных и функций, являющееся основой структурного подхода, плохо отображает картину реального мира.
- Неконтролируемый доступ к данным
  - **Локальные данные** находятся внутри функции и предназначены для использования исключительно этой функцией. Локальные данные функции недоступны никому, кроме самой функции, и не могут быть изменены другими функциями.
  - Если существует необходимость совместного использования одних и тех же данных несколькими функциями, то данные должны быть объявлены как **глобальные**.

Глобальные данные и функции



Большие программы обычно содержат множество функций и глобальных переменных. Проблема процедурного подхода заключается в том, что число возможных связей между глобальными переменными и функциями может быть очень велико, как показано на след рисунке:



Большое число связей между функциями и данными, в свою очередь, также порождает несколько проблем.

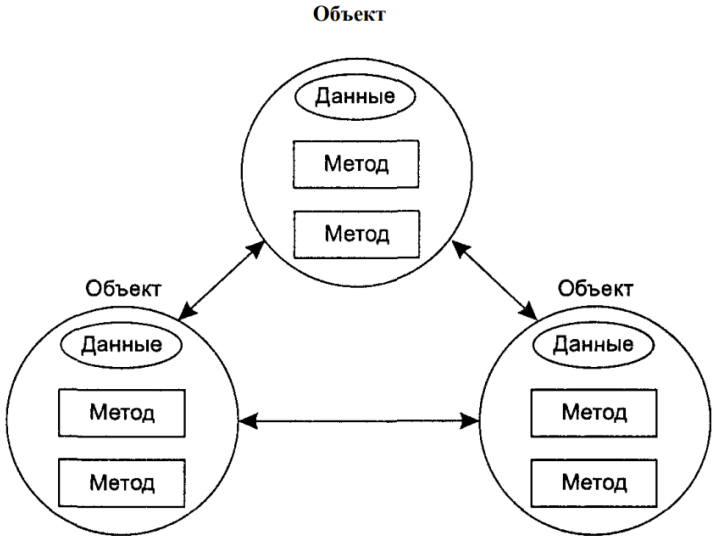
- Во-первых, усложняется структура программы.
- Во-вторых, в программу становится трудно вносить изменения. **Изменение структуры глобальных данных** может потребовать переписывания всех функций, работающих с этими данными. (**пример с изменением типами данных {short->long}**)
- Из-за многочисленных связей между функциями и данными исправленные функции начинают некорректно работать с другими глобальными данными

Моделирование реального мира

- Вторая, более важная, проблема процедурного подхода заключается в том, что отделение данных от функций оказывается малопригодным для отображения картины реального мира. **В реальном мире нам приходится иметь дело с физическими объектами**, такими, например, как люди или машины. Эти объекты нельзя отнести ни к данным, ни к функциям, поскольку реальные вещи представляют собой совокупность **свойств и поведения**.
- Свойства (также называют характеристиками). Приведем примеры для людей и машин (цвет глаз, количество дверей и т.д.)
- Поведение — это некоторая реакция объекта в ответ на внешнее воздействие. **Пример:** если вы нажмете на тормоз автомобиля, это повлечет за собой его остановку. Поведение сходно с функцией (совершает действие)
- **Итог:** Таким образом, ни отдельно взятые данные, ни отдельно взятые функции не способны адекватно отобразить объекты реального мира.

Объектно-ориентированный подход

- Основополагающей идеей объектно-ориентированного подхода является объединение данных и действий, производимых над этими данными, в единое целое, которое называется объектом.
- Функции объекта, называемые в C++ **методами** или **функциями-членами**, обычно предназначены для доступа к данным объекта. Если необходимо считать какие-либо данные объекта, нужно вызвать соответствующий метод, который выполнит считывание и возвратит требуемое значение. **Прямой доступ к данным невозможен**. Данные сокрыты от внешнего воздействия, что защищает их от случайного изменения. Говорят, что данные и методы **инкапсулированы**. Термины сокрытие и инкапсуляция данных являются ключевыми в описании объектно-ориентированных языков
- Если необходимо изменить данные объекта, то, очевидно, это действие также будет возложено на методы объекта. Никакие другие функции не могут изменять данные класса. Такой подход облегчает написание, отладку и использование программы.



Пример

- Сотрудники каждого отдела производят операции только с теми данными, которые относятся к данному отделу. Подобное разделение обязанностей позволяет легче следить за деятельностью компании и контролировать ее, а также поддерживать целостность информационного пространства компании



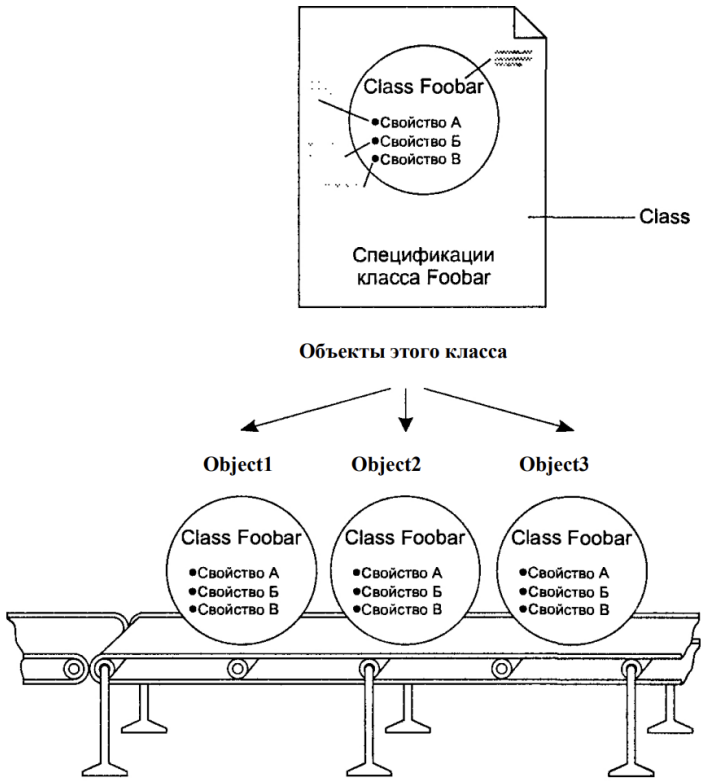
Характеристики объектно-ориентированных языков

- **Объекты**. Когда вы подходите к решению задачи с помощью объектно-ориентированного метода, то вместо проблемы разбиения задачи на функции вы сталкиваетесь с проблемой разбиения ее на объекты. Мышление в терминах объектов оказывается гораздо более простым и наглядным, чем в терминах функций, поскольку программные объекты схожи с объектами реального мира. Более подробно данный вопрос рассмотрен в последующих уроках.
- **Классы.** Когда мы говорим об объектах, мы говорим, что они являются экземплярами классов. Что это означает? Рассмотрим следующую аналогию. Практически все компьютерные языки имеют стандартные типы данных; например, в C++ есть целый тип int. Мы можем определять переменные таких типов в наших программах:

```
int day;
int count;
int divisor;
int answer;
```

- Класс является своего рода формой, определяющей, какие данные и функции будут включены в объект класса. При объявлении класса не создаются никакие объекты этого класса, по аналогии с тем, что существование типа int еще не означает существование переменных этого типа.

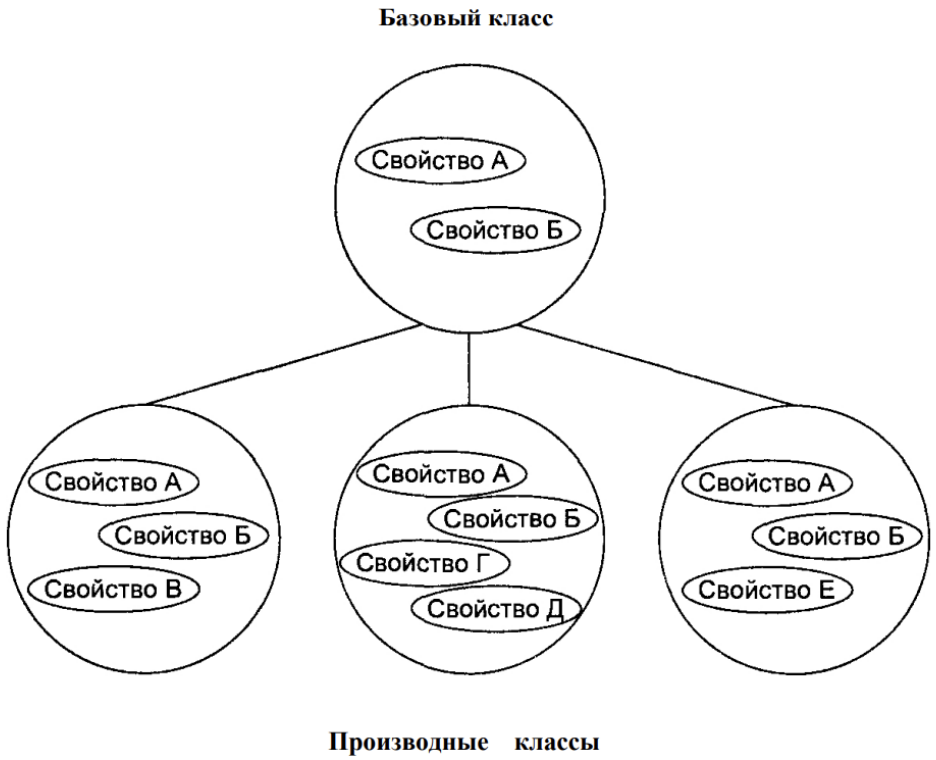
Пример класса



- Класс является описанием совокупности сходных между собой объектов (**называют экземпляром класса**). Это соответствует нестрогому в техническом смысле пониманию термина «класс»: например, Micheal Jackson, The Beatles и Rihanna относятся к классу поп-музыкантов. (пример: музыканты как объекты)

Наследование

- Понятие класса приводит нас к понятию наследования. В повседневной жизни мы часто сталкиваемся с разбиением классов на подклассы: например, класс животные можно разбить на подклассы млекопитающие, земноводные, насекомые, птицы и т. д. Класс наземный транспорт делится на классы автомобили, грузовики, автобусы, мотоциклы и т.д.
- Принцип, положенный в основу такого деления, заключается в том, что каждый подкласс обладает свойствами, присущими тому классу, из которого выделен данный подкласс.
- Не проводите ложных аналогий между отношениями «объект—класс» и «базовый класс — производный класс»! Объекты, существующие в памяти компьютера, являются воплощением свойств, присущих классу, к которому они принадлежат. Производные классы имеют свойства как унаследованные от базового класса, так и свои собственные



Повторное использование кода

- Разработанный класс может быть использован в других программах. Это свойство называется возможностью повторного использования кода. Аналогичным свойством в процедурном программировании обладают библиотеки функций, которые можно включать в различные программные проект.
- Программист может взять существующий класс, и, ничего не изменяя, добавить в него свои элементы. Все производные классы унаследуют эти изменения, и в то же время каждый из производных классов можно модифицировать отдельно.
- **Пример** с созданием графического пользовательского интерфейса (GUI) Microsoft Windows. вы создаете новый класс, наследующий все свойства исходного класса, и добавляете в него необходимый код.
- Легкость повторного использования кода уже написанных программ является важным достоинством ООП. Многие компании утверждают, что возможность включать в новые версии программного обеспечения коды программ более старых версий благоприятно сказывается на прибыли, приносимой

Полиморфизм и перегрузка.

- Представьте себе, что вам необходимо работать с объектами, имеющими две координаты, например x и y:

```
position_obj1 = position_obj2 + origin
```

- Обратите внимание на то, что операции присваивания = и сложения + для типа position должны выполнять действия, отличающиеся от тех, которые они выполняют для объектов стандартных типов, например int. Объекты **position1** и прочие не являются стандартными, поскольку определены пользователем как принадлежащие классу **position**.
- Использование операций и функций различным образом в зависимости от того, с какими типами величин они работают, называется полиморфизмом. Когда существующая операция, например **=** или **+**, наделяется возможностью совершать действия над операндами нового типа, говорят, что такая операция является перегруженной. Перегрузка представляет собой частный случай полиморфизма и является важным инструментом ООП.

Универсальный язык моделирования (UML)

- UML можно условно называть графическим языком, предназначенным для моделирования компьютерных программ. Под моделированием понимается создание наглядной визуальной интерпретации чего-либо. UML позволяет создавать подобную интерпретацию программ высокоуровневой организации.
- **Почему UML необходим?**
  - Во-первых, потому, что бывает трудно установить взаимоотношение частей большой программы между собой посредством анализа ее кода.
  - Проблема изучения кода состоит в том, что он очень детализован. Гораздо проще было бы взглянуть на его общую структуру, отображающую только основные части программы и их взаимодействие. UML обеспечивает такую возможность
- Наиболее важным средством UML является набор различных видов диаграмм. Диаграммы классов иллюстрируют отношения между различными классами, диаграммы объектов — между отдельными объектами, диаграммы связей отражают связь объектов во времени и т. д. Все эти диаграммы, по сути, отражают взгляды на программу и ее действия с различных точек зрения
- Не забывайте, что UML является лишь средством для иллюстрирования разрабатываемого проекта.

Выводы

1. Структурный подход плохо справляется с большими программами из-за неограниченного доступа функций к глобальным данным и разделения данных и функций.
2. ООП объединяет данные и действия над ними в объекте; доступ к данным осуществляется через методы объекта.
3. Класс описывает совокупность однотипных объектов, а объект является экземпляром класса.
4. Наследование обеспечивает повторное использование кода, а полиморфизм позволяет по-разному применять одни и те же операции и функции к данным разных типов.
5. UML — графический язык, позволяющий наглядно описать структуру программы до написания кода.

Контрольные вопросы

1. Какие недостатки процедурного (структурного) программирования привели к появлению ООП?
2. Чем плохо неограниченный доступ функций к глобальным данным?
3. В чём состоит основная идея объектно-ориентированного подхода?
4. Чем класс отличается от объекта? Приведите пример.
5. Что такое наследование и какие преимущества оно даёт?
6. Что такое полиморфизм и перегрузка? Для чего нужен язык UML?

Список литературы

Основная литература

1. Лафоре Р. Объектно-ориентированное программирование в C++. 4-е изд. — СПб.: Питер, 2004.
2. Страуструп Б. Язык программирования C++. Специальное издание. — М.: Бином, 2011.
3. Буч Г., Рамбо Дж., Джекобсон А. Язык UML. Руководство пользователя. 2-е изд. — М.: ДМК Пресс, 2007.

Дополнительная литература

4. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приёмы объектно-ориентированного проектирования. Паттерны проектирования. — СПб.: Питер, 2007.
5. Страуструп Б. Программирование: принципы и практика с использованием C++. 2-е изд. — М.: Вильямс, 2016.
6. Stroustrup B. A Tour of C++. 3rd ed. — Addison-Wesley, 2022.

Электронные ресурсы

7. Учебник по C++ (learncpp): www.learncpp.com
8. Справочник по C++ и стандартной библиотеке: en.cppreference.com

- Спасибо за внимание!