Объектно-ориентированное программирование

ст. преподаватель Пелипас Всеволод Олегович

ст. преподаватель Сметанина Татьяна Ивановна

https://github.com/pelipas/OOP_2017

План курса

- Вводная лекция. Основные понятия OOA/D/P.
- C++ - язык ООП
- UML – язык OOA/OOD
- OOA и OOD на примере
- Принципы SOLID
- Основные шаблоны объектно-ориентированного проектирования (как пример применения ОО подхода к решению практических задач)

Литература

- Общее понимание ООП и OOD:
  - Гради Буч - Объектно-ориентированный анализ и проектирование с примерами приложений
- Любой учебник по C++
  - Дейтел Х. М. Как программировать на C++ (полное издание) —2008
- Практика проектирования и шаблоны:
  - Крэг Ларман - Применение UML 2.0 и шаблонов проектирования. Практическое руководство.
  - Эрик Фримен, Элизабет Фримен - Паттерны проектирования (Head First O'Reilly).
  - Э.Гамма и др. - Приемы объектно-ориентированного проектирования. Паттерны проектирования.

Почему это важно?

Объектно-ориентированная технология - **основной** метод промышленной разработки программного обеспечения

Что такое программа?

Ваши версии?

Что такое программа?

- ISO/IEC/IEEE 24765:2010
  - комбинация компьютерных инструкций и данных, позволяющая аппаратному обеспечению вычислительной системы выполнять вычисления или функции управления
- ISO/IEC 2382-1:1993
  - синтаксическая единица, которая соответствует правилам определённого языка программирования, состоящая из определений и операторов или инструкций, необходимых для определённой функции, задачи или решения проблемы.
- ГОСТ 19781—90
  - данные, предназначенные для управления конкретными компонентами системы обработки данных в целях реализации определённого алгоритма
- ГК РФ
  - представленная в объективной форме совокупность данных и команд, предназначенных для функционирования ЭВМ и других компьютерных устройств с целью получения определённого результата
- Сильно упрощая – это способ объяснить компьютеру, чего мы от него хотим ☺️.

Парадигмы программирования (сильно упрощая)

- **Парадигма** — это совокупность идей и понятий, которые мы используем для описания или познания чего-либо.
- Простыми словами: **парадигма программирования** определяет то, опираясь на какие понятия мы программируем, т.е. «объясняем компьютеру, что мы от него хотим».
- Основные парадигмы программирования:
  - **Императивная** – как делать?
  - **Декларативная** – что делать?
  - **Объектная** – кто делает?

ОО-парадигма

- Ключевые идеи:
  - Программа - это совокупность объектов, способных взаимодействовать друг с другом посредством сообщений;
  - Каждый объект является экземпляром определенного класса;
  - Классы образуют иерархию наследования.
- Фактически, ОО-программа – это работающая модель.
  - Как реализовано поведение элементов этой модели – императивно или декларативно – значения не имеет.
  - Хотя большинство популярных ОО-языков имеют императивные корни (C#, Java, C++), есть и функциональные ОО-языки (OCaml, ОО-диалекты Haskell)

Steve Jobs, 1994, в интерью журналу Rolling Stone:

- **Не могли бы вы в нескольких словах объяснить, что же такое объектно-ориентированное программное обеспечение?**
- Объекты – они как люди. Они живые вещи, у которых есть разум, позволяющий им знать, как сделать ту или иную вещь, у них есть память. Вы взаимодействуете с ними на очень высоком уровне абстракции, словно с людьми.
- Например, я – ваш объект, занимающийся чисткой ваших вещей. Вы можете дать мне грязную одежду и послание “доставь мои вещи в прачечную”. Я знаю, где в Сан-Франциско лучшая прачечная, я говорю по-английски и у меня есть деньги в кармане. Я выхожу, ловлю такси, посещаю прачечную и возвращаюсь с вашими вещами и словами: “Вот, ваша чистая одежда”.
- Вы не знаете, как я это сделал. Не знаете, где эта прачечная или вы говорите по-французски, а может у вас нет денег, чтобы поймать такси. Однако, я знал, как все это сделать, а вам – это знать необязательно. Вся сложность процесса скрыта внутри меня, но наше с вами общение было простым – в этом вся суть объектов. **Сложности – внутри, но интерфейс – доступен каждому.**

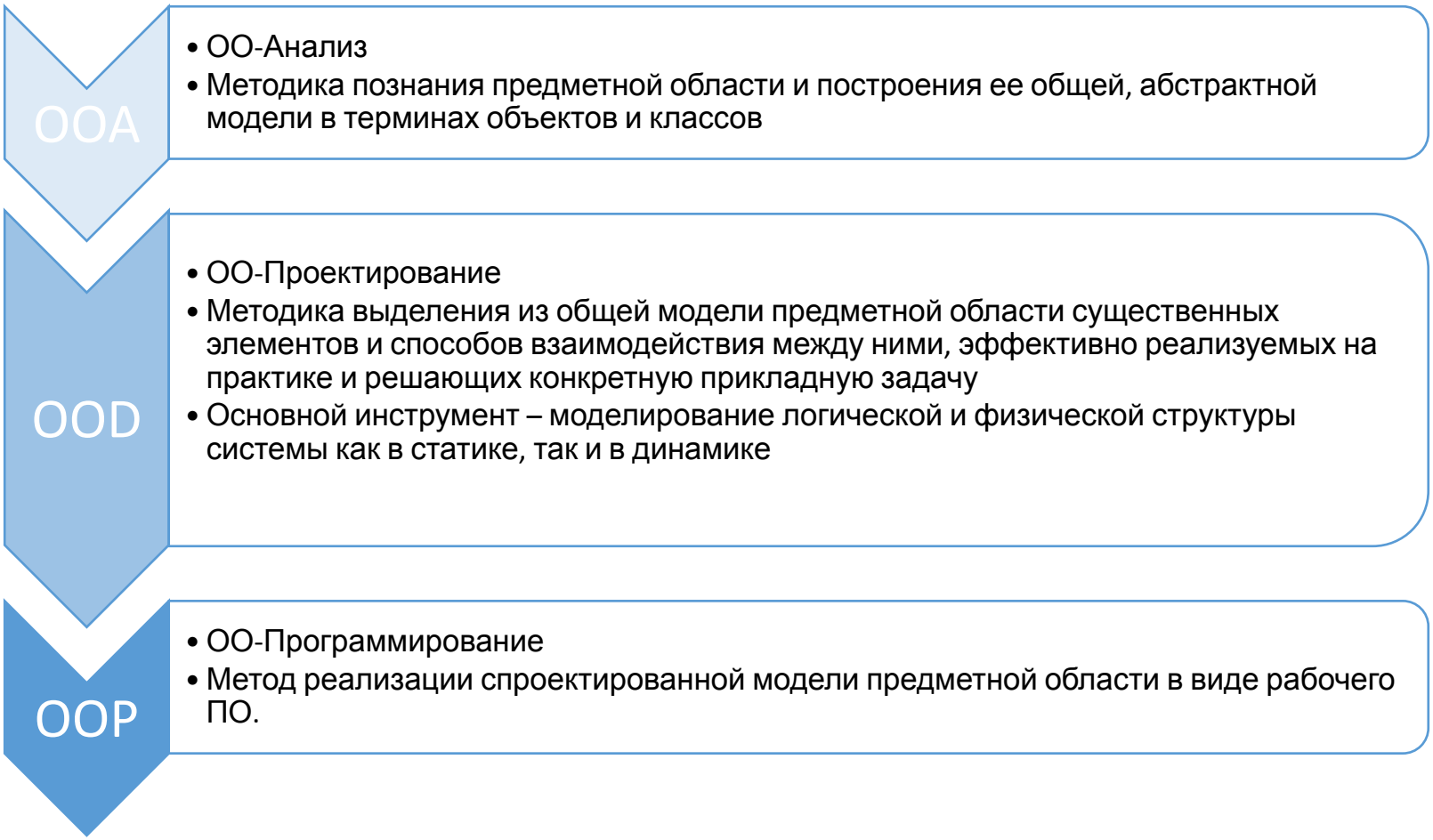
Вопросы?

- Дальше – углубимся в базовые понятия ОО-подхода

Объекты и классы

- Определение Г. Буча: **объект** – это некая сущность в нашей модели, обладающая:
  - Состоянием
  - Поведением
  - Индивидуальностью
- Структура и поведение схожих объектов определяется в общем для них классе.
- **Класс** - описание структуры объекта и методов работы с ним.
- **Класс –** тип данных, определяемый программистом, **объект –**переменная класса

Объектный подход



ООА - пример

- Постановка задачи (л/р №1):
  - Динамическая структура—очередь. В списке хранится информация о событиях: день (1-31), месяц (1-12), год (целое число) и название (строка). Предусмотреть функции добавления элементов в список и удаления из него, а также поиска введенной даты.
- Объекты и операции с ними:
  - Набор событий
    - Создать/удалить набор
    - Добавить/удалить из набора событие
    - Найти событие в наборе по дате
  - Событие
    - Создать/удалить событие
    - Получить/изменить данные события

OOD - пример

- Класс Событие
  - Поля для данных
  - **Указатель на следующий элемент очереди**
  - Конструктор для создания
  - Деструктор для уничтожения (*если строку распределяем динамически*)
  - Функции получения и установки значений полей
- Класс Очередь
  - Указатели на голову и хвост очереди (на элемент Событие)
  - Конструктор для создания
  - Деструктор для уничтожения (перебор и уничтожение Событий)
  - Функции добавления и изъятия События из очереди
  - Функция поиска элемента в очереди (перебор очереди)
- А нужен ли класс Очередь? Не редуцировать ли его? ☺

ООР - пример

```
class Event
{
        int day, month, year;
        char* name;
        Event* next;

        public:
        Event(int d, int m, int y, char* n) {…};
        ~Event() {…};
        Event* GetNextEvent() {…};
        char* GetName() {…};
        void SetName(char* n) {…};
        ….
}
```

Ну а полный исходный код самого решения на C++ вы напишете самостоятельно ☺

Предпосылки ОО подхода

- Сложность – неотъемлемая часть ПО
- История роста сложности:
  - машинные коды
  - математика
  - алгоритмы
  - интерактивные системы
  - графический интерфейс
  - распределенные (облачные) системы.
- Проблемы сложности: сроки (бюджеты) и надежность

Борьба со сложностью при проектировании ПО

- Абстракция – концентрация на существенных деталях, отбрасывая несущественные
- Декомпозиция – «разделяй и властвуй»:
  - Алгоритмическая
    - Проблема разделяемых и неразделяемых данных
  - Объектная
    - Принцип разделения ответственности – код и данные объединены в одну сущность

Вопросы?

- Дальше – рассмотрим элементы ОО-подхода

Элементы объектного подхода

- Основные элементы:
  - Абстрагирование;
  - Инкапсуляция;
  - Модульность;
  - Иерархия.
- Дополнительные элементы:
  - Типизация;
  - Полиморфизм;
  - Параллелизм;
  - Персистентность (сохраняемость).

Абстракция.

Абстракция выделяет существенные характеристики некоторого объекта, отличающие его от всех других видов объектов и, таким образом, четко определяет его концептуальные границы с точки зрения наблюдателя.

Абстракция.

Виды абстракций:

Абстракция сущности	Объект представляет собой полезную модель некой сущности в предметной области
Абстракция поведения	Объект состоит из обобщенного множества операций
Абстракция виртуальной машины	Объект группирует операции, которые либо вместе используются более высоким уровнем управления, либо сами используют некоторый набор операций более низкого уровня
Произвольная абстракция	Объект включает в себя набор операций, не имеющих друг с другом ничего общего

Абстракция - пример

- Тема разрабатываемой системы
  - По традиции – компьютерная игра ☺️. Например, платформер.
  - Наша предметная область: персонаж, враги, вертикальные и горизонтальные блоки и все такое.
- Какие сущности выделим?
  - Какие из них нам действительно важны, а какие нет?
  - Какие свойства этих сущностей нам нужны, а какие нет?
- Какие связи между сущностями выделим?
  - Статика (структура данных)
  - Динамика (взаимодействие объектов)
- Результат – модель.

Инкапсуляция

- **Инкапсуляция** - это процесс отделения друг от друга элементов объекта, определяющих его устройство и поведение:
  - **Интерфейс** отражает внешнее поведение объекта, описывая абстракцию поведения всех объектов данного класса.
  - Внутренняя **реализация** описывает представление этой абстракции и механизмы достижения желаемого поведения объекта.
- Инкапсуляция служит для того, чтобы изолировать контрактные обязательства абстракции от их реализации.
- Инкапсуляция скрывает детали реализации объекта.

Инкапсуляция - пример

- Та же модель, рассмотрим Персонажа
- Что может делать объект (интерфейс):
  - Идти
  - Прыгать
- А что внутри?
  - Надо хранить и менять координату по определенному закону
  - Надо проверять на коллизии с другими объектами
  - Надо перерисовывать изображение в соответствии с изменением координаты

Модульность

- **Модульность** - это свойство системы, которая была разложена на внутренне связные, но слабо связанные между собой модули.
- Принципы абстрагирования, инкапсуляции и модульности являются взаимодополняющими.
  - Объект логически определяет границы определенной абстракции,
  - а инкапсуляция и модульность делают их физически незыблемыми.

Модульность - пример

- Та же модель игры
- Группируем классы
  - Модуль с классами модели игры
  - Модуль с классами визуальных моделей
  - Модуль с классами общего игрового интерфейса
  - И т.д.

Иерархия

- **Иерархия** - это упорядочение абстракций, расположение их по уровням.
- Основными видами иерархических структур применительно к сложным системам являются:
  - структура классов (наследование, иерархия «является», "is-a")
  - структура объектов (композиция/агрегация, иерархия «содержит», "part of").
- **Наследование** - такая иерархия абстракций, в которой подклассы наследуют **строение** и **поведение** от одного или нескольких суперклассов.
  - Простое наследование – когда подкласс создается только из одного суперкласса.
  - Множественное наследование, когда подкласс создается из нескольких суперклассов.

Иерархия – пример композиции/агрегации

- Снова модель игр, но чуть посложнее
- Группа юнитов – агрегация объектов
  - Управляем классом Отряд – он принимает общие решения (например, прокладка пути отряда) и делегирует управление деталями классам Юнит (отрисовка действий, например).
  - Отряд можно распустить и пересобрать.
- Сложные составные объекты – композиция объектов
  - Моделируем танк – включаем в него модули Двигатель, Орудие, Башня, Гусеницы и т.п.
  - Модуль не существует отдельно от Танка, а Танк – без Модуля.

Иерархия – пример наследования

- Возвращаемся к модели платформера
- Игровой персонаж и враги
  - много общего
    - Свойства – координаты, состояние здоровья, сила атаки
    - Поведение – перемещение, атака и т.п.
  - но есть и разница – в поведении, возможностях и т.п.
- Решение
  - вынести общие свойства и поведение в суперкласс (общего предка) Персонаж
  - частности реализовать в классах ИгровойПерсонаж и NPC (неигровой персонаж), наследующих от Персонажа общие свойства и поведение

Типизация

- **Типизация** - это способ защититься от использования объектов одного класса вместо другого, или, по крайней мере, управлять таким использованием
- Объектно-ориентированные языки программирования могут быть строго типизированными, нестрого и совсем не типизированными.
- По времени проверки типизации существует статическая типизация (статическая связь, на этапе компиляции) и динамическая типизация (динамическая связь, на этапе исполнения).

Типизация - пример

- Та же игровая модель
- От класса NPC порождаем 2 производных подкласса
  - Лучник – имеет поведение Стрелять
  - Мечник – имеет поведение Бить
- Когда мы создаем экземпляры этих объектов – мы создаем переменные нужного типа*
  - Проверка типов компилятором позволяет убедиться, что мы не пытаемся Мечника заставить Стрелять и наоборот – это вызовет ошибку компиляции.

** чаще – указатели на нужный тип, но об этом будем говорить позже*

Типизация и наследование

- Продолжаем работать с той же моделью
  - Классы Лучник и Мечник унаследовали от предка NPC поведение Перемещаться.
- В таком случае, мы можем заставлять Перемещаться и экземпляры (переменные) класса Лучник и класса Мечник.
  - Пусть Вася – Лучник
  - А Петя – Мечник
- Мы можем заставить Перемещаться и Васю, и Петю по отдельности:
  - Можем, как бы, сказать Васе :«ты Лучник, а значит NPC, а значит умеешь Перемещаться. Переместись сюда». Аналогично с Петей.
- А теперь объединим их в Отряд:
  - Лучник (потомок НеигровогоПерсонажа) Вася
  - Мечник (потомок НеигровогоПерсонажа) Петя
  - Как заставить их шагать в ногу?

Типизация и наследование

- Т.к. поведение и структура, определенные суперклассом, наследуются всеми потомками, значит у любого потомка они гарантированно присутствуют.
  - И Лучник и Мечник унаследовали от NPC способность Перемещаться
- Значит, все свойства и поведение предка можно использовать, обращаясь к ним, независимо от того, какого типа на самом деле потомок.
  - Можно заставить Перемещаться любого NPC,мы это видели ранее
- Значит, работать с экземпляром любого подкласса можно через переменную типа суперкласса, от которого унаследовано нужное поведение
  - Если я хочу заставить Перемещаться любого NPC – мне все равно, Лучник это или Мечник, я могу, как бы, обратиться к нему: «так, я знаю, ты - NPC, и умеешь Перемещаться, Перемещайся туда-то»

Типизация и наследование

- Итак, смоделировать отряд можно как массив объектов типа NPC
  - Каждый из этих объектов на самом деле может быть либо Лучником, либо Мечником
  - Компилятор разрешит нам присваивать переменной типа NPC экземпляры подтипов Лучник и Мечник
  - Естественно, работая с ними как просто с NPC мы не сможем ни Бить, ни Стрелять – базовый NPC этого не умеет, и компилятор нам такого не позволит.
- Тогда алгоритм перемещения отряда выглядит элементарно:
  1. ТекущийNPC = первый попавшийся NPC из Отряда
  2. заставить ТекущегоNPC Переместиться в нужную сторону
  3. ТекущийNPC = следующий НПЦ из Отряда
  4. Если Отряд еще не закончился – идем к п. 2.

Полиморфизм

- **Полиморфизм** – обеспечение множественности вариантов реализации однотипного поведения различными классами-потомками общего предка.
- Т.е. все потомки умеют делать то, что умел делать предок, но каждый потомок может делать это по-своему

Полиморфизм - пример

- Научим наш отряд драться вместе
- Добавим в иерархию класс БоевойNPC между классами NPC и Лучник/Мечник.
  - БоевойNPC наследует от NPC Перемещение, а от него – его наследуют Лучник/Мечник
  - БоевойNPC вводит поведение Атаковать, но не определяет его пока никак.
- В классах Лучник и Мечник переопределим поведение Атаковать:
  - Для лучника – как вызов поведения Стрелять
  - Для мечника – как вызов поведения Бить
- Заставляем отряд Атаковать вместе по аналогии с Перемещением

Полиморфизм - пример

- Переопределим Отряд как как массив объектов типа БоевойNPC
- Алгоритм атаки отряда:
  1. ТекущийБоевойNPC = первый попавшийся БоевойNPC из Отряда
  2. заставить ТекущегоБоевогоNPC Атаковать нужный объект
  3. ТекущийБоевойNPC = следующий БоевойNPC из Отряда
  4. Если Отряд еще не закончился – идем к п. 2.
- И тут нас ждет суровая встреча с реальностью:
  - Компилятор может повести себя НЕПОЛИМОРФНО – и вызовет метод Атаковать класса БоевойNPC (по непосредственному типу) – и ничего не произойдет (т.к. мы его не определяли)
  - Компилятор может повести себя ПОЛИМОРФНО – и вызовет переопределенный метод Атаковать класса Лучник/Мечник (понял настоящий тип объекта) – произойдет атака соответствующего типа (удар или стрельба).
  - Тем, как именно поведет себя компилятор – мы можем управлять (смотрите в следующих сериях ☺)

Параллелизм

- **Параллелизм** - это свойство, отличающее активные объекты от пассивных.
- В OOA/D/P каждый объект (как абстракция реальности) может представлять собой отдельный канал управления (абстракцию процесса). Такой объект называется активным.

Параллелизм - пример

- Распределенные и многопоточные системы – истинный параллелизм
- Наша любимая предметная область – игры:
  - Многие объекты должны быть активными – действовать, не только реагируя на действия игрока, но и самостоятельно.
  - Обычно объекты реализуют некоторое поведение как реакцию на приходящее сообщение (вызов функции объекта)
  - Общепринятый подход - вводится понятие игрового цикла – некоего основного цикла, который по очереди перебирает объекты игрового мира, и дает им возможность «сделать ход» – некое подобие коллективной многозадачности.

Персистентность

- **Персистентность(сохраняемость)** - способность объекта существовать во времени, переживая породивший его процесс, и (или) в пространстве, перемещаясь из своего первоначального адресного пространства.
- Спектр персистентности объектов охватывает:
  - Промежуточные результаты вычисления выражений.
  - Локальные переменные в вызове процедур.
  - Глобальные переменные и динамически создаваемые данные.
  - Данные, сохраняющиеся между сеансами выполнения программы.
  - Данные, сохраняемые при переходе на новую версию программы.
  - Данные, которые вообще переживают программу

Персистентность - пример

- Та же предметная область
  - Реализуем сохранение и загрузку состояния игрового мира и персонажей – повышаем уровень персистентности ряда объектов
  - Часть объектов – летящие снаряды, например – нет смысла не то что сохранять в файл, а сохранять и в памяти после того, как они пролетят через игровое поля – их необходимо уничтожать или переиспользовать, чтобы не засорять память.
  - У каждого объекта – свой жизненный цикл и уровень персистентности, который надо определоить на стадии проектирования.

Выводы

- В программировании существует несколько парадигм, ориентированных на процедуры, объекты, логику, правила и ограничения.
- Развитие программной индустрии привело к созданию методов объектно-ориентированного анализа, проектирования и программирования, которые служат для программирования сложных систем.
- Абстракция определяет существенные характеристики некоторого объекта, которые отличают его от всех других видов объектов и, таким образом, абстракция четко очерчивает концептуальную границу объекта с точки зрения наблюдателя.
- Инкапсуляция - это процесс разделения устройства и поведения объекта; инкапсуляция служит для того, чтобы изолировать контрактные обязательства абстракции от их реализации.
- Модульность - это состояние системы, разложенной на внутренне связные и слабо связанные между собой модули.
- Иерархия - это ранжирование или упорядочение абстракций.
- Типизация - это способ защититься от использования объектов одного класса вместо другого, или по крайней мере способ управлять такой подменой.
- Полиморфизм – обеспечение множественности вариантов реализации однотипного поведения различными классами-потомками общего предка.
- Параллелизм - это свойство, отличающее активные объекты от пассивных.
- Сохраняемость (персистентность) - способность объекта существовать во времени и (или) в пространстве.