Три кита ООП.

1. Кратко об ООП
   1. Вопрос – что такое объектно-ориентированное программирование? Своими словами.
   2. До появления ООП основной подход был – функциональный. (Показать как выглядел процедурный код)
      1. Данные и код содержатся на самом верхнем уровне исполнения программы.
      2. Программа представляет собой последовательность выполняемых инструкций.
   3. Но он оказался неудобным для крупных проектов.
      1. Сложно было отделить код от данных. Мы работаем с каждой переменной как с отдельной сущностью, и теряется взаимосвязь между одной переменной и другой. Скажем мы задаем в расчете крутящий момент на вал, а материал задаем другой переменной – нужно держать этот нюанс в голове.
      2. Периодически встречались одни и те же участки кода
      3. Росла кодовая база. Привести пример программы vvvvvv.
   4. Причины, по которым родилось ООП:
      1. Мир привычнее видеть с помощью объектов.
      2. Плюсом ООП относительно функционального программирования, стало заключение функций в отдельные сущности, которые были близки в своем выполнении по бизнес-логике в рамках какого-либо объекта.
   5. Основные концепции ООП:
      1. Инкапсуляция
      2. Наследование
      3. Абстракция
      4. Полиморфизм
   6. Показать слайд и спросить – что НЕ является принципом ООП?
   7. (Шуточная иллюстрация основных принципов) – не стоит показывать.
2. Инкапсуляция
   1. Что такое инкапсуляция?
      1. Спросить – что такое инкапсуляция?
      2. Ответ: Инкапсуляция заключается в объединении данных и методов, которые с ними работают, внутри одного объекта. При этом доступ к данным объекта ограничивается (через модификаторы доступа), чтобы скрыть внутреннюю реализацию и предоставить только необходимый интерфейс.
   2. Содержимое класса
      1. Показать пример**WhatIsEncapsulation**
      2. Показать код класса
      3. Класс Coder обладает:
         1. Именем и зарплатой
         2. Поведением, которое было для него обозначено
      4. Показать клиентский код
      5. Клиентский код:
         1. Создает экземпляр класса Coder
         2. Может обратиться к имени через свойство
         3. Может вызвать метод, в частности – WriteCode
   3. Права доступа
      1. Спросить –как вы думаете: зачем нужны права доступа?
      2. Ответ:
         1. «Защита от дурака». Избежание ошибок: Например, если изменение поля доступно только через метод, вы можете контролировать его изменения в этом методе.  
            Пример: Вы содержите в объекте класса делимое, а через метод передаете делитель. Делитель не может быть равен 0. Если Вы работаете через метод, то можно сразу обработать данную исключительную ситуацию.
         2. Нет доступа в те области объекта, изменение которых может привести к неблагоприятным последствиям.   
            Пример: есть поле с id, но оно носит исключительно информативный характер.
         3. Есть доступ только к тому, что необходимо видеть и не думать о неиспользуемых частях.   
            Пример: расчет значения по формуле – очень большой и содержится в функции. Один из шагов расчета содержится во вложенной функции. Ее не нужно делать публичной, так как шаг расчета отдельно использован никогда не будет.
      3. Пролистать слайды с бытовыми примерами инкапсуляции
      4. Показать пример **AccessRightsExample**
   4. Решить задачи
3. Наследование
   1. Спросить - что такое наследование своими словами?
   2. Ответ: наследование -это механизм, который позволяет использовать возможности других классов.
   3. Показать синтаксис наследования на слайде.
   4. Показать пример синтаксиса наследования на слайде.
   5. Что наследуется:
      1. Рассказать, что наследуется на слайде.
      2. Показать синтаксис наследования в проекте **WhatIsInheritance**
         1. Рассказать о классе Animal
            1. Перейти в исходный код и рассказать, что будет унаследовано.
         2. Показать, что в Programвыполняется метод GetName() у всех наследников.
   6. Что не наследуется:
      1. Рассказать, что не наследуется с нюансами на слайде.
         1. Sealed
            1. Раскомментироватьsealed у класса**NotInheritance**
            2. Раскомментироватьsealedу метода **SomeClass. SealedMethodTest – ошибка компиляции.**
         2. Конструкторы
            1. Показать ключевое слово base. Явный вызов. Оно требуется только для передачи параметров
            2. Даже если не указан явный вызов конструктора, код конструктора родительского класса все равно выполнится.
            3. Показать в коде **Program.cs**
         3. Статичныеданные не выдадут ошибку компиляции, если попытаться их использовать, но обращаться они будут все равно к родительскому методу.
         4. При попытке обращения к приватному полю/методу, компилятор будет ругаться и скомпилировать код не получится. Однако получить доступ к приватному члену класса все равно можно, через методы в классе предка. Главное, чтобы прав доступа хватало.
   7. Что такое множественное наследование и почему оно запрещено в C#?
      1. Показать пример что такое множественное наследование.
      2. Показать UMLcмножественным наследованием и прочитать вопрос со слайда.
   8. Транзитивное наследование. Просто прочитать со слайда.
   9. Ключевое слово base
      1. Зачитать слайд.
      2. Показать наследование метода. Проект **Base**
      3. Показать наследование конструктора. Проект **Base**
   10. Ключевое слово virtual
       1. Зачитать слайд virtual.
       2. Зачитать слайд override.
       3. Показать наследование метода. Проект **virtual**
       4. Показать наследование конструктора. Проект **virtual**
   11. Вопросы, на которые отвечает наследование
   12. Решение задач
4. Абстракция
   1. Спросить что такое абстракция своими словами?
      1. Зачитать текст со слайда: **Что такое абстракция?**
      2. Показать слайд: **Суть абстракции.** Привести также в пример «космическое тело». Кто-то представляет звезду, кто-то планету, кто-то метеорит, кто-то комету. И все варианты подходят.
      3. Наиболее яркий бытовой пример абстракции – армейский. Командир говорит солдату: сделай что-то и мне неважно как ты это сделаешь.
      4. Override заменить код в слайде.
   2. Реализация абстракции в C# состоит в использовании абстрактных классов и интерфейсов.Проект**SimpleExample(*Переработать код.)***
      1. Ключевое слово abstract – проект **SimpleExample**.Запустить проект и показать:
         1. Невозможность создать объекты интерфейса и абстрактного класса через ключевое слово new.
         2. Создать объект сапера и вызвать метод ToDoинтерфейса.
         3. Вызвать методSubmitRapport, реализованный в интерфейсе.
         4. Вызвать методSubmitRapport, который переопределен в классе сапера.
         5. Вызвать методJustGo, определенный в классе Human.
      2. Обратите внимание, что свойства и методы объекта будут представлены в зависимости от того, к какому типу был приведен этот объект.
5. Полиморфизм
   1. Вопрос студентам: Что такое полиморфизм? Своими словами
      1. Полиморфизм – это свойство системы использовать объекты без информации о реализации.
      2. Примеры:
         1. Транспорт. Начальник нам сказал добраться из точки А в точку Б. С точки зрения бизнеса – все равно как устроены велосипед, автобус, такси или личный автомобиль. Необходимо просто знать, что это транспорт. Слайд с транспортом.
         2. Допустим, вы - фермер. У нас есть животные. Все животные кушают и имеют какой-то рацион питания. Все остальное – детали реализации.
         3. Показать пример: проект **Polymorphism**.
      3. Параметрический полиморфизм. По сути это работа с дженериками. Мы имеем какой-то универсальный тип, к которому мы обращаемся не зная деталей реализации. Проект **ParametricPolyMorphism**
      4. Под аdhoc-полиморфизмом обычно понимается перегрузка методов. Проект **AdHocPolymorphism**
      5. Рассказать пример из практики.
      6. Показать пример: проект **CounterStrike**.
   2. **Прорешать задачи.**
6. UML
   1. Кратко об UML
   2. Диаграммы классов

<https://skillbox.ru/media/code/oop_chast_1_chto_takoe_klassy_i_obekty/>

<https://blog.skillfactory.ru/glossary/polimorfizm/>

1. Переработать слайд и код(упростить):
   1. «Что не наследуется»
   2. «Что наследуется»