

*School of
Computer
Science*

ПРИНЦИПЫ ООП. ИНКАПСУЛЯЦИЯ. ВЗАИМОДЕЙСТВИЕ ОБЪЕКТОВ

ПРОГРАММИРОВАНИЕ НА PYTHON

Лекции для IT-школы



1-АЯ ТЕМА СЕГОДНЯШНЕГО ЗАНЯТИЯ

- Основные принципы ООП:
 - Инкапсуляция и абстракция данных
 - Наследование
 - Полиморфизм
- Сегодня мы попрактикуем:
 - Создание классов, определяющих функциональность объектов
 - Разработку разнотипных методов и атрибутов классов для использования в объектах
 - Создание объектов (экземпляров классов)
 - Ограничение доступа к атрибутам объекта



2-АЯ ТЕМА СЕГОДНЯШНЕГО ЗАНЯТИЯ

- **Взаимодействие объектов**
- Сегодня мы попробуем:
 - Создание множественных объектов в одной программе
 - Взаимодействие объектов друг с другом
- В ООП объекты взаимодействуют через обмен сообщениями
- На практике это значит, что объекты вызывают методы друг друга



ПРАКТИЧЕСКАЯ ЦЕЛЬ ЗАНЯТИЯ

ПРОГРАММА «МОЯ ЗВЕРЮШКА»

- Позволяет пользователю завести виртуальное домашнее животное:
 - Пользователь сам присваивает имя зверьку и должен ухаживать за ним
- Чтобы зверюшка оставалась в хорошем настроении нужно:
 - Кормить ее
 - Играть с ней
- Кроме того, можно спрашивать у зверюшки как она себя чувствует



ОСНОВЫ ОБЪЕКТНО ОРИЕНТИРОВАННОГО ПОДХОДА

- ООП позволяет представить объект реального мира как программный объект:
 - Характеристики объекта – это его **атрибуты**
 - Способы поведения объекта – **методы**
- **Объекты** создаются на основе описаний, называемых **классами**
- Другими словами, класс – это фрагмент кода, в котором объявлены атрибуты и методы, используемые объектами (**экземплярами класса**)



СОЗДАНИЕ КЛАССОВ, МЕТОДОВ И ОБЪЕКТОВ

- Объявление класса – `class Critter(): ...`
- Объявление метода экземпляра класса:
 - Метод экземпляра – это функция объекта, расположенная внутри определения класса
 - Всегда содержит первый параметр `self`
- Создание объекта – присваивание результата вызова класса переменной, например: `crit = Critter()`
- Вызов метода – `crit.talk()`
- Смотрите пример в `simple_critter.py`



ПРИМЕНЕНИЕ ИНИЦИАЛИЗИРУЮЩЕГО МЕТОДА (КОНСТРУКТОРА)

- Особый метода `__init__` вызывается сразу после создания нового объекта
- Обычно `__init__` применяется для установки начальных значений атрибутов объекта
- Добавим конструктор `__init__` в класс `Critter`
- Создадим пару экземпляров класса `Critter` и посмотрим сообщения от них
- Смотрите пример в `init_critter.py`



ПРИМЕНЕНИЕ АТТРИБУТОВ

ПРОГРАММА «ЗВЕРЮШКА С АТТРИБУТОМ»

- Создадим новый тип объекта с атрибутом `name`, присваивая его в методе `__init__`
- Созданные в `__init__` атрибуты объекта могут использоваться в любых методах
- Возможен прямой доступ к полям объекта извне, хотя это плохая практика разработки
- Осмысленный вывод информации об объекте на экран реализует метод `__str__`
- Смотрите пример в `attribute_critter.py`



ПРИМЕНЕНИЕ АТТРИБУТОВ КЛАССА И СТАТИЧЕСКИХ МЕТОДОВ

- Информация, которая относится ко всему классу, хранится в **аттрибутах класса**
- Методы, связанные с классом, называются **статическими**. Они часто применяются вместе с атрибутами класса
- Для обозначения метода статическим используется конструкция `@staticmethod`
- Создадим атрибут класса `Critter.total` – количество созданных объектов
- Смотрите пример в `classy_critter.py`



ДОСТУП К АТТРИБУТУ КЛАССА

- Через класс – `Critter.total`:
 - Внутри определения класса
 - В основной программе
- Через статическую функцию-метод в основной программе – `Critter.status()`
- Через экземпляр класса – `crit1.total` – в основной программе



СПЕЦИФИКА СТАТИЧЕСКИХ МЕТОДОВ

- Перед описанием статического метода указывается декоратор `@staticmethod`
- У них нет первого параметра `self` как у методов экземпляра класса
- Вызов статического метода принято делать через класс – `Critter.status()`



МЕТОДЫ КЛАССА. ИХ ОТЛИЧИЕ ОТ СТАТИЧЕСКИХ МЕТОДОВ

- Для обозначения метода класса используется конструкция `@classmethod`
- В методе класса присутствует ссылка на класс
- Ссылка на класс упрощает обращение к атрибутам класса
- Смотрите пример в `classy_critter2.py`



ИНКАПСУЛЯЦИЯ И АБСТРАКЦИЯ В ООП

- **Инкапсуляция** в контексте ООП – это связывание данных объекта с методами, которые оперируют этими данными
- Данные объекта при этом зачастую хочется скрыть от внешнего мира – это **сокрытие данных**
- **Инкапсуляция + сокрытие данных = абстракция**



ЗАКРЫТЫЕ АТТРИБУТЫ И МЕТОДЫ

- Чтобы ограничить прямой доступ клиентского кода к атрибутам объекта можно сделать атрибуты **закрытыми**
- **Два символа подчеркивания** в начале имени атрибута или метода говорят Python, что они закрытые
- Смотрите пример в [private_critter.py](#)



РЕКОМЕНДАЦИИ ПО ПОСТРОЕНИЮ СОБСТВЕННЫХ КЛАССОВ

- При реализации своих классов придерживайтесь двух правил:
 - создавайте методы, существование которых сделает ненужным прямой доступ из клиентского кода к атрибутам объекта
 - закрывайте лишь те атрибуты и методы, которые обслуживают внутренние операции объекта
- В работе с объектом полезно следовать таким правилам:
 - избегайте непосредственного чтения и изменения значений атрибутов
 - никогда не пытайтесь прямо обращаться к закрытым атрибутам и методам объекта



УПРАВЛЕНИЕ ДОСТУПОМ К АТТРИБУТАМ

- **Свойства** – инструмент для доступа к атрибутам экземпляра класса из клиентского кода
- Можно ограничивать такой доступ, например, разрешить только чтение атрибута
- Смотрите пример в [property_critter.py](#)
- Более подробно можно почитать [здесь](#)



КОНЕЧНАЯ ВЕРСИЯ ПРИМЕРА ПРОГРАММА «МОЯ ЗВЕРЮШКА»

- Конструктор с инициализацией имени, уровней голода и уныния
- Закрытый метод `__pass_time()` для прироста уровней голода/уныния
- Свойство `mood` с доступом по чтению
- Методы `talk()`, `eat()` и `play()` для разговора, кормления и игры
- См. скрипт в `critter_caretaker.py`



ПРАКТИЧЕСКИЕ ЗАДАНИЯ

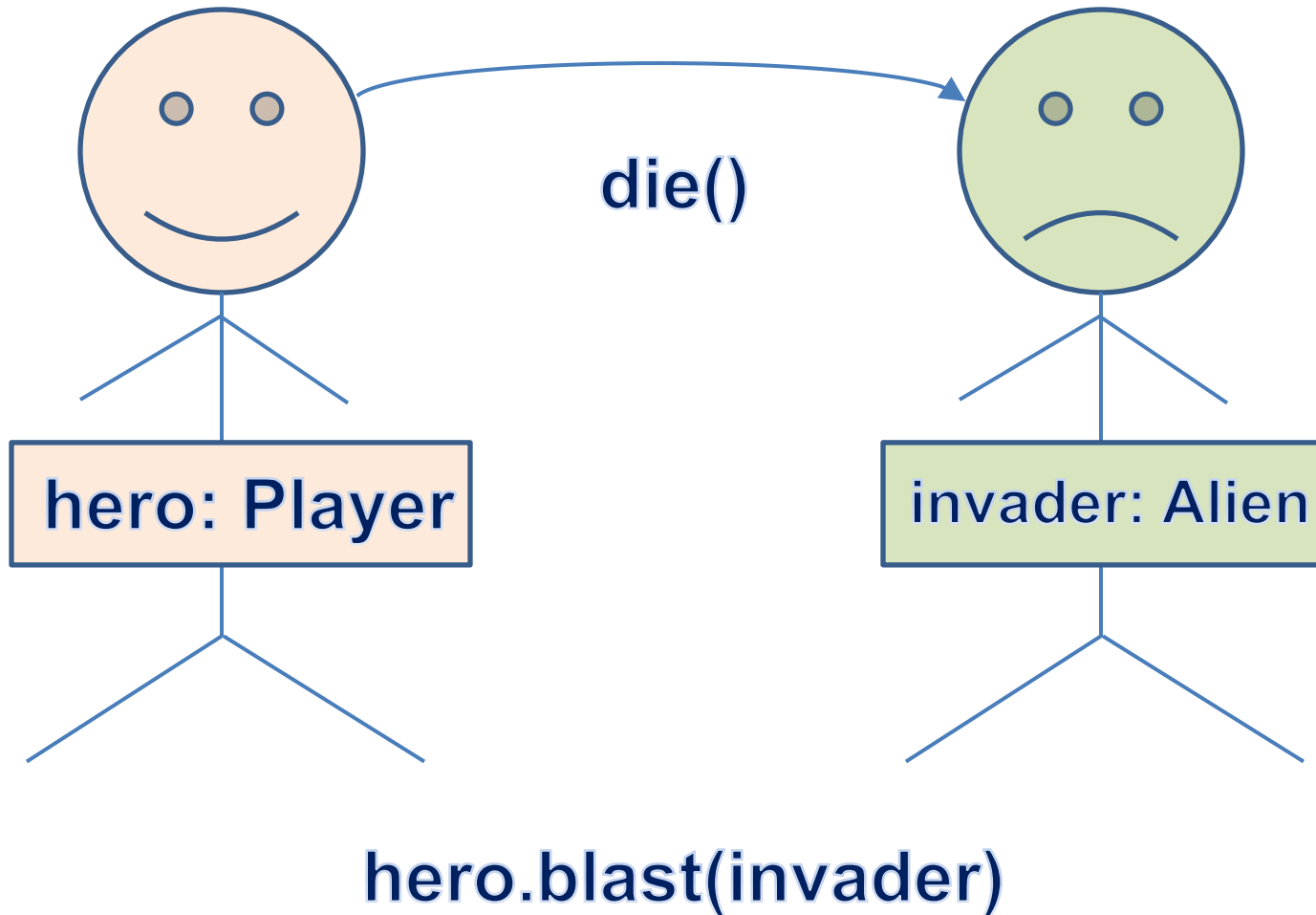
РАЗВИТИЕ ПРОГРАММЫ «МОЯ ЗВЕРЮШКА»

1. Доработайте программу, чтобы пользователь мог сам решить, сколько еды скормить зверюшке и сколько времени потратить на игру с ней:
 - в зависимости от передаваемых величин зверёк насыщается и веселеет неравномерно
 - если голод зверька превысит лимит, например становится > 10 , то животное погибает
2. Создайте в программе «Моя зверюшка» своего рода «черный ход» – способ увидеть точные значения числовых атрибутов объекта:
 - сделайте в меню секретный пункт, который подсказка не будет отражать, а, если пользователь его выберет, выводите объект на экран (для этого в классе `Critter` должен быть реализован специальный метод `__str__`)



ОБМЕН СООБЩЕНИЯМИ

ПРОГРАММА «ГИБЕЛЬ ПРИШЕЛЬЦА»





СКРИПТ «ALIEN_BLASTER.PY»

- Объект класса `Player` связывается с переменной `hero`
- Объект класса `Alien` связывается с переменной `invader`
- Командой `hero.blast(invader)` вызывается метод `blast()` объекта `hero` с аргументом `invader`
- Метод `blast()` командой `enemy.die()` вызывает метод `die()` объекта класса `Alien`



ПРАКТИКА «БИТВА ВОИНОВ»

СКРИПТ `WARRIOR_TEMPLATE.PY`

- Создайте класс `Warrior` (Воин) и два экземпляра этого класса
- У каждого объекта начальный уровень здоровья равен 100 очков
- Пусть воины бьют друг друга в случайном порядке
- Если воин бьет, то он здоровья не теряет, а если воина бьют, то его здоровье уменьшается на 20 очков от каждого удара
- После удара выводится сообщение, какой воин атаковал, и сколько у его противника осталось здоровья
- Как только у кого-то заканчивается ресурс здоровья, программа завершается сообщением о том, кто одержал победу



ДОМАШНЕЕ ЗАДАНИЕ

ПРОГРАММА «ЗВЕРОФЕРМА»

1. Напишите программу «Звероферма», в которой будет создано несколько объектов класса **Critter**, а манипулировать ими всеми можно будет с помощью списка
2. Теперь пользователь должен заботиться не об одной зверюшке, а обо всех обитателях фермы. Выбирая пункт в меню, пользователь выбирает действие, которое хотел бы выполнить со всеми зверюшками: покормить их, поиграть с ними или узнать об их самочувствии
3. Чтобы программа была интереснее, при создании каждой зверюшки следует назначать ей случайно выбранные уровни голода и уныния
4. Оголодавшие животные, с уровнем голода > 10 погибают и удаляются из списка

СПАСИБО ЗА ВНИМАНИЕ !
ВОПРОСЫ ?



*School of
Computer
Science*