Основные понятия ООП

Основные парадигмы программирования

Процедурное (структурное) программирование – программа представляет собой последовательность инструкций (**процедур**).

Объектно-ориентированное программирование – программа представляет собой совокупность взаимодействующих между собой **объектов**, каждый из которых является экземпляром определенного **класса**, а классы образуют иерархию **наследования**.

Функциональное программирование – все вычисления в программе осуществляются через вызовы **функций**.

Логическое программирование – вычисления описываются с помощью формальной логики (через набор **высказываний**), т.е. описывается сама задача, а не способ ее решения.

Классификация языков программирования

По парадигме:

Процедурные (императивные) – FORTRAN, C, Pascal

Объектно-ориентированные – C++, C#, Java, Kotlin, Python, JavaScript, Go, Swift, Ruby, PHP и др.

Функциональные – Lisp, R, Python, Kotlin

Логические (декларативные) – Prolog

Дополнительная классификация:

Языки разметки – HTML, CSS

Скриптовые – JavaScript, PHP

Универсальные (мультипарадигменные) – Python, Kotlin, F#

Концепция локальных и глобальных данных в процедурном программировании

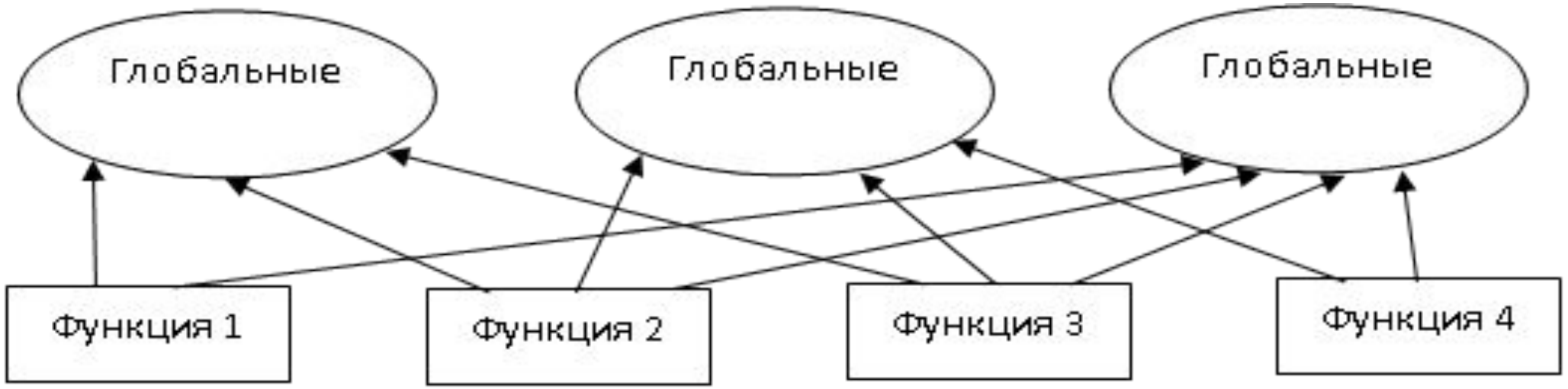
Проблема: число возможных связей между глобальными переменными и функциями может быть очень велико.

Следствия: - усложняется структура программы;

- в программу становится трудно вносить изменения.

Основные проблемы процедурного подхода

- неограниченный доступ функций к глобальным данным



- разделение данных и функций – такой подход плохо отображает картину реального мира, поскольку данные существуют сами по себе, функции - сами по себе.

Основная идея объектно-ориентированного подхода

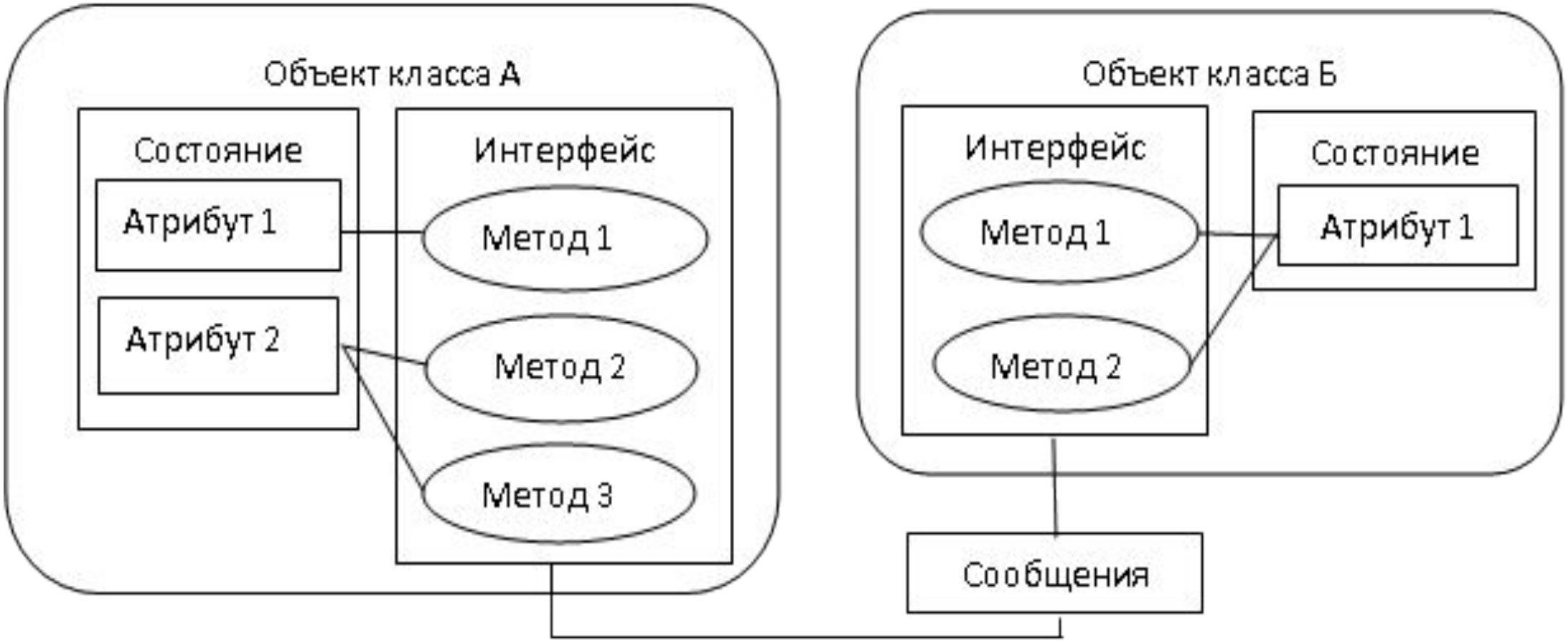
- объединение данных и функций

Результат: при решении задачи в стиле ООП вместо проблемы разбиения ее на функции программист решает проблему разбиения ее на объекты.

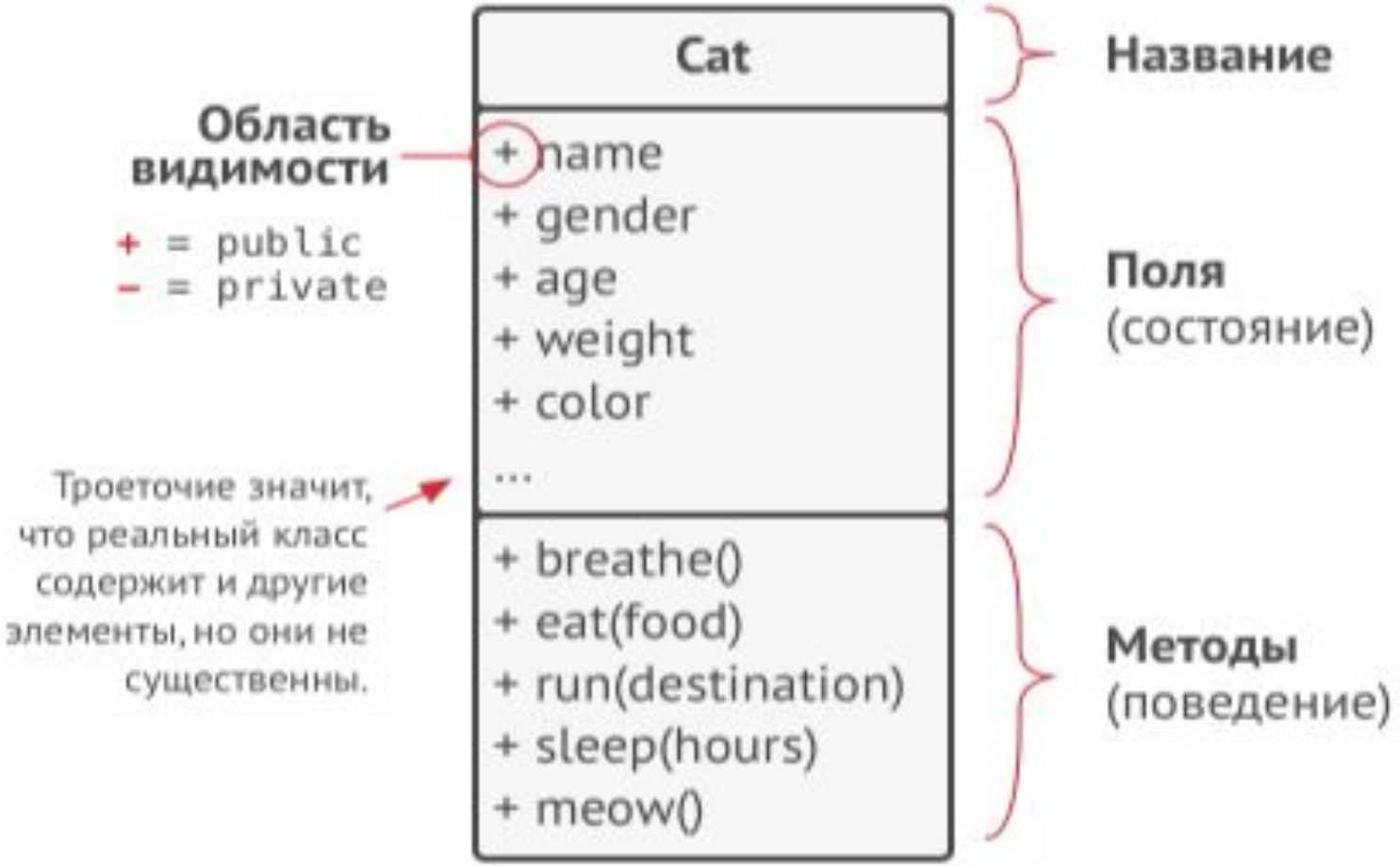
Свойства программных объектов

- каждый программный объект моделирует некоторый объект реального мира (самолет, автомобиль, файл, список сотрудников, геометрическую фигуру);
- каждый программный объект обладает собственным набором свойств (атрибутов) (автомобили одной марки могут иметь разный цвет кузова, файлы могут быть разного типа);
- все однотипные объекты принадлежат одному классу;
- все объекты одного класса обладают схожим поведением, т.е. имеют один и тот же набор функций (методов) (автомобиль любого цвета при нажатии на педаль тормоза остановится, файлы разного типа можно перемещать, копировать, переименовывать, удалять).

Общая схема объектно-ориентированного подхода



Пример: класс Кот



UML-диаграмма класса Cat

Объекты – это экземпляры классов

Pushistik: Cat

```
name   = "Pushistik"
sex    = "male"
age    =  3
weight =  5.5
color  =  gray
```

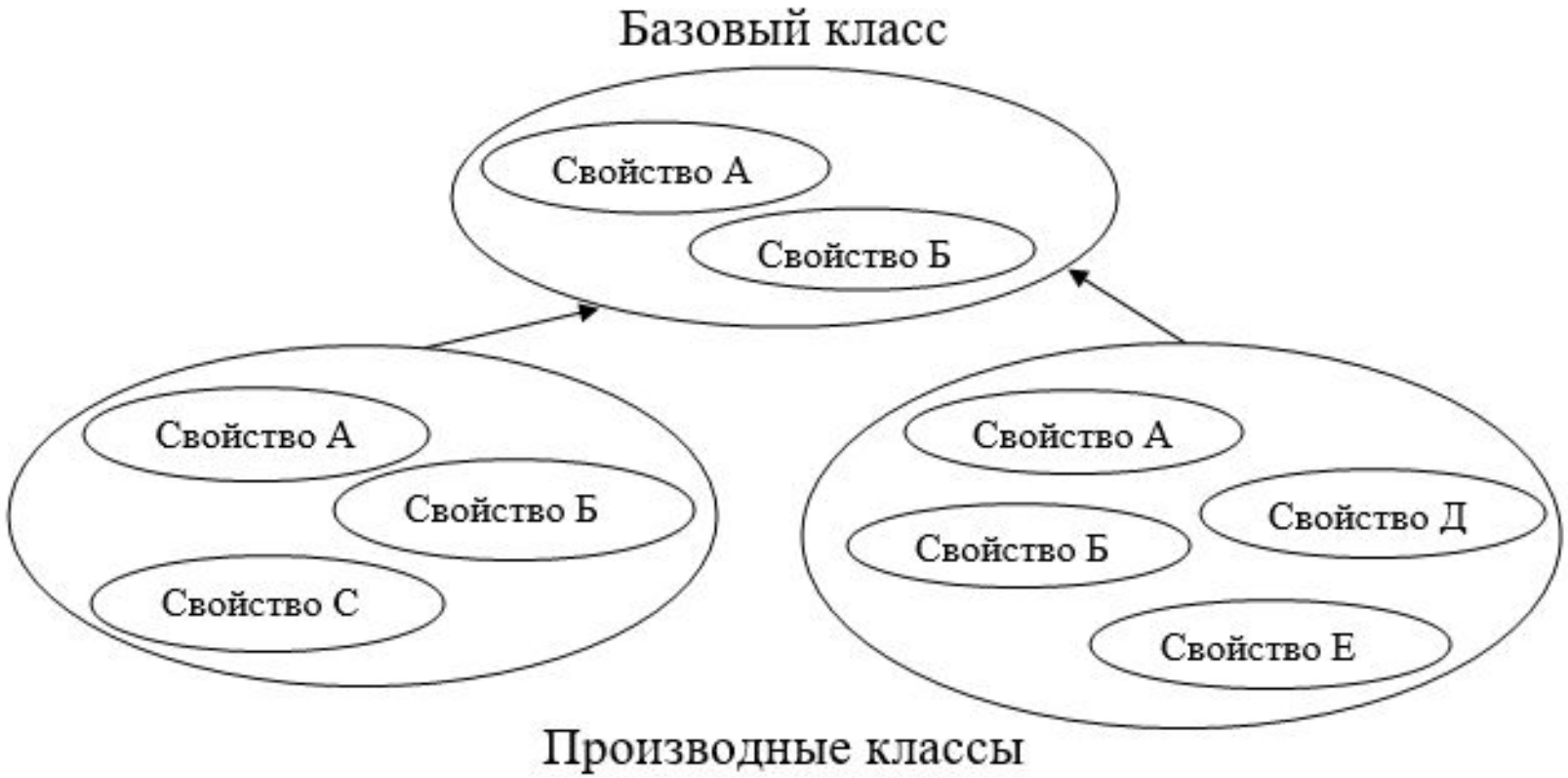
Murka: Cat

```
name   = "Murka"
sex    = "female"
age    =  1
weight =  3.5
color  =  white
```

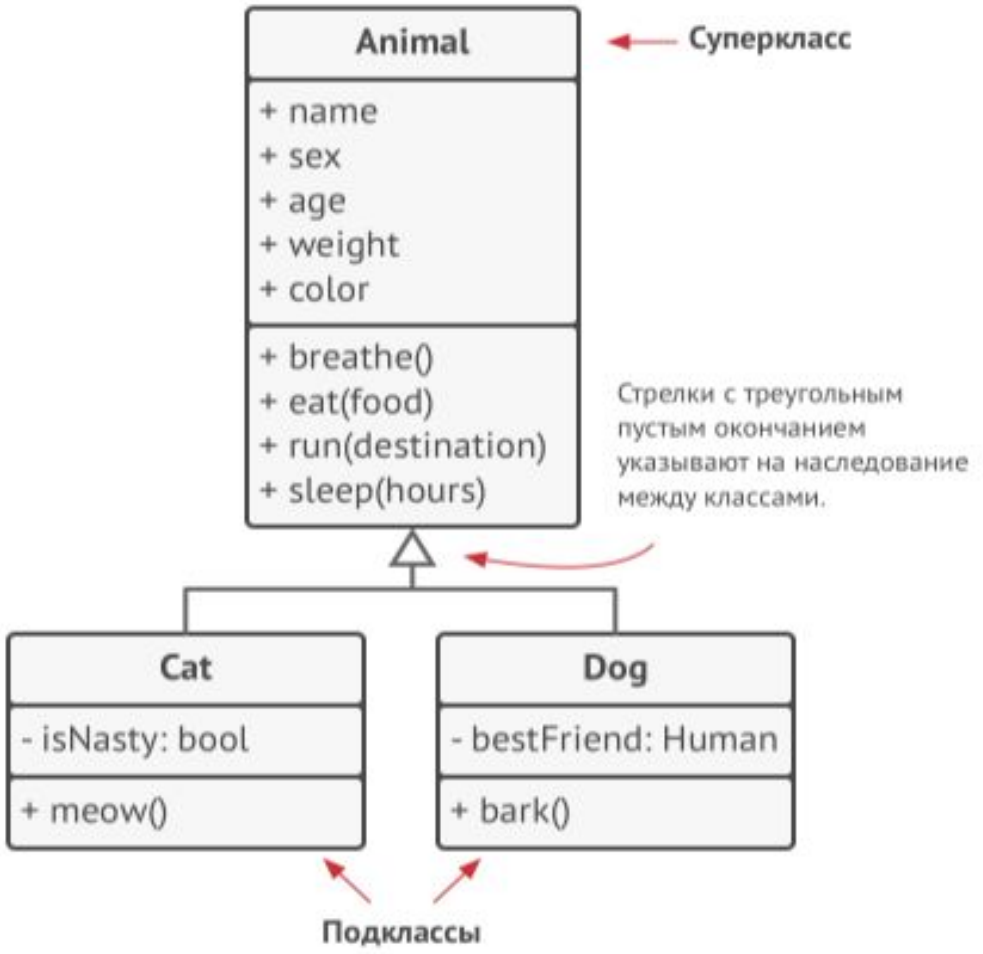
Главное: класс – это своеобразный «чертеж», по которому создаются объекты – экземпляры этого класса.

Иерархия классов

Для отработки отношений иерархии в объектном подходе используется механизм наследования: все общие свойства различных сущностей объединены в некотором базовом (родительском) классе, а производные классы наследуют все его свойства, добавляя свои собственные свойства и поведение.

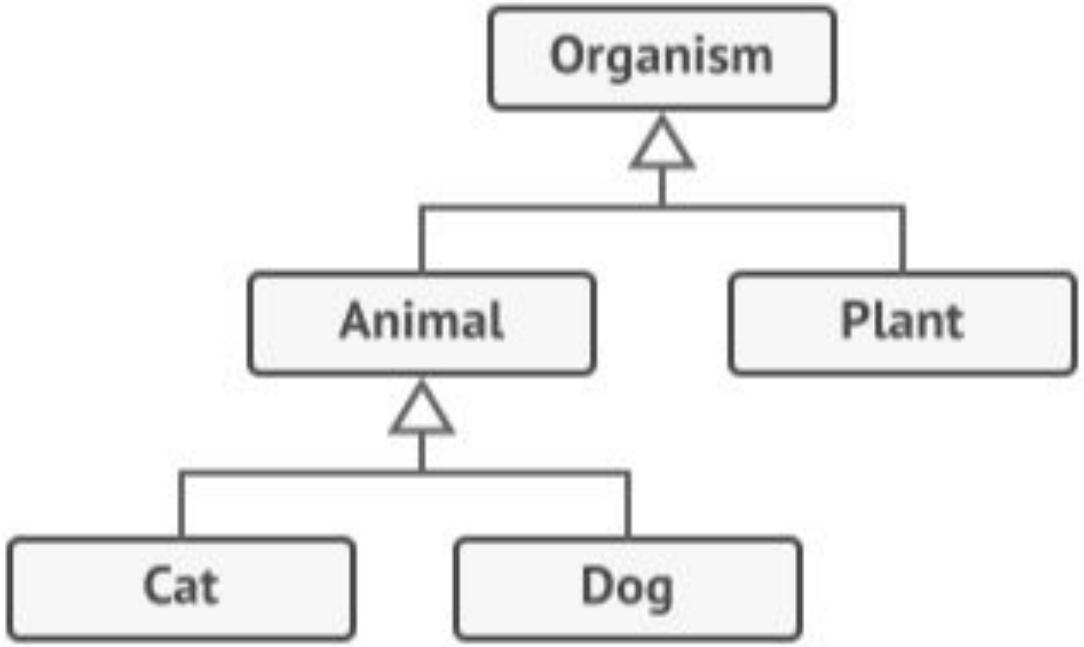


Пример наследования



UML-диаграмма иерархии классов

Пример наследования



Главное:

- наследование – это возможность создания новых классов на основе существующих;
- производные классы могут переопределять методы базовых классов; при этом они могут как полностью заменить поведение метода базового класса, так и просто что-то добавить к нему.

Особенности наследования

- механизм наследования работает только в одну сторону, а именно от производного класса к базовому. Пример: инженер – это человек. Обратное же неверно: человек – это необязательно инженер.
- назначение наследования в объектно-ориентированном программировании такое же, как и у функций в процедурном программировании – сократить размер кода и упростить связи между элементами программы.
- 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 проекты.

Плюсы и минусы наследования

Главная польза от наследования – возможность повторного использования существующего кода.

Пример. Постоянные обновления версий различного ПО напрямую связаны с использованием механизма наследования. Вносить так часто изменения в таких количествах в процедурную программу без нарушения ее работоспособности было бы очень проблематично.

Главная «расплата» за наследование – производные классы наследуют все свойства и методы базового класса, т.е. нельзя исключить из производного класса метод базового класса (если, например, он там не нужен).

Ложные аналогии

Отношения «класс – объект» и «базовый класс – производный класс» – это «две большие разницы».

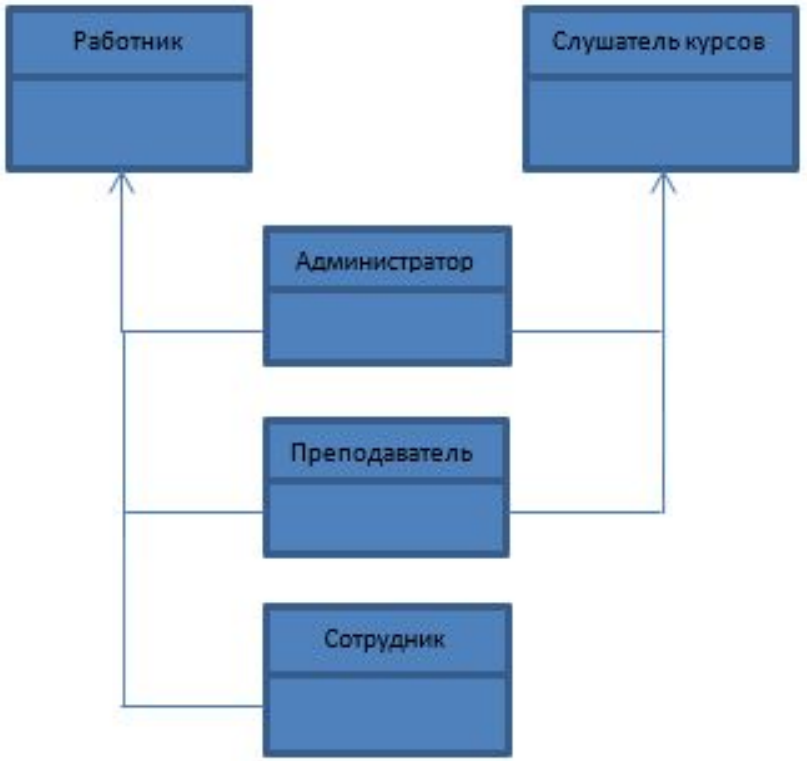
Объекты класса представляют собой воплощение свойств и методов, присущих классу, к которому они принадлежат.

Производные классы имеют свойства как унаследованные от базового класса, так и свои собственные.

Множественное наследование

Множественное наследование – это разновидность наследования, когда производный класс наследуется от нескольких базовых классов.

Важно! Концепция множественного наследования поддерживается не всеми языками программирования. Например, в Java вместо множественного наследования используется такая конструкция, как интерфейс.



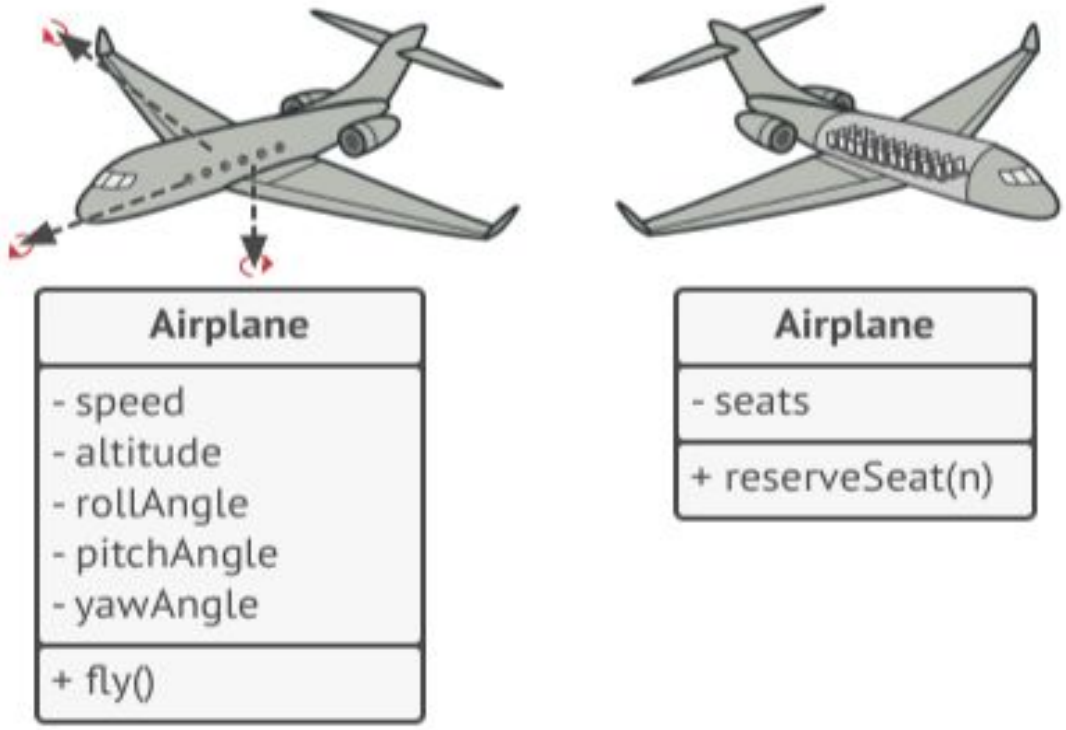
Краеугольные камни ООП

Абстракция

Программные объекты не повторяют в точности их реальные аналоги, они только ***моделируют*** свойства и поведение реальных объектов, важные в конкретном контексте, а остальные – игнорируют.

Абстракция – это модель некоего объекта или явления реального мира, в которой опущены незначительные детали, не играющие существенной роли в данном контексте.

Главная задача ООП – выбор правильного набора абстракций (классов и объектов) для заданной предметной области.



Инкапсуляция

Инкапсуляция – это способность объектов скрывать часть своего состояния и поведения от других объектов, предоставляя внешнему миру только определенный интерфейс (публичная часть объекта) для взаимодействия с собой.

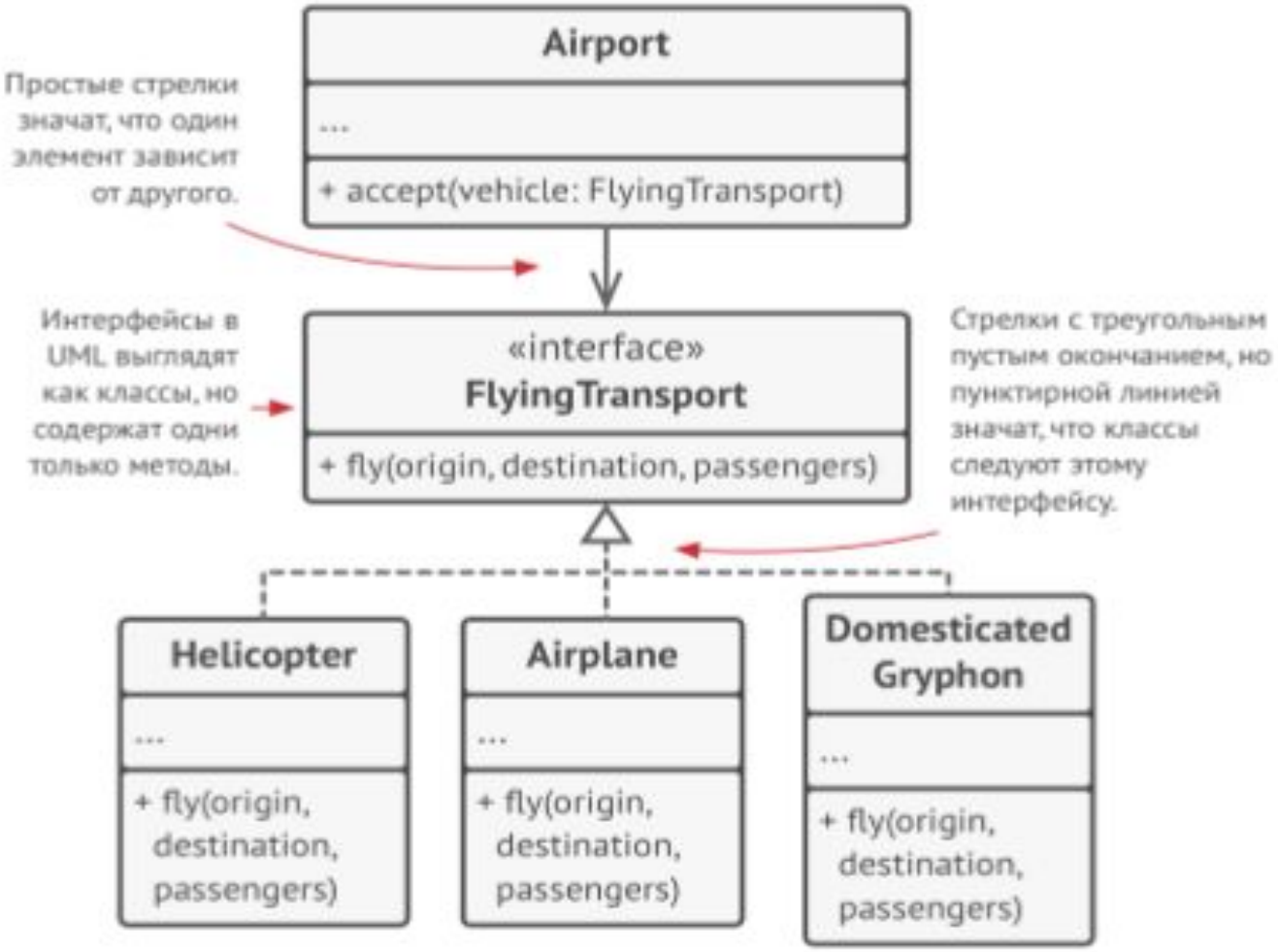
Инкапсуляция означает, что получить доступ к данным (свойствам) объекта можно только с помощью каких-либо его методов, т.е. прямой доступ к данным объекта закрыт.

Фактически, объект класса для окружающей среды (операционной системы, объектов других классов) представляет собой аналог «чёрного ящика»: он может принимать входные воздействия и выдавать в качестве реакции на них выходные, но при этом он никак не проявляет свою внутреннюю структуру.

Интерфейс интерфейсу рознь

Словом интерфейс называют как публичную часть объекта, так и конструкцию interface во многих языках программирования (например, в Java).

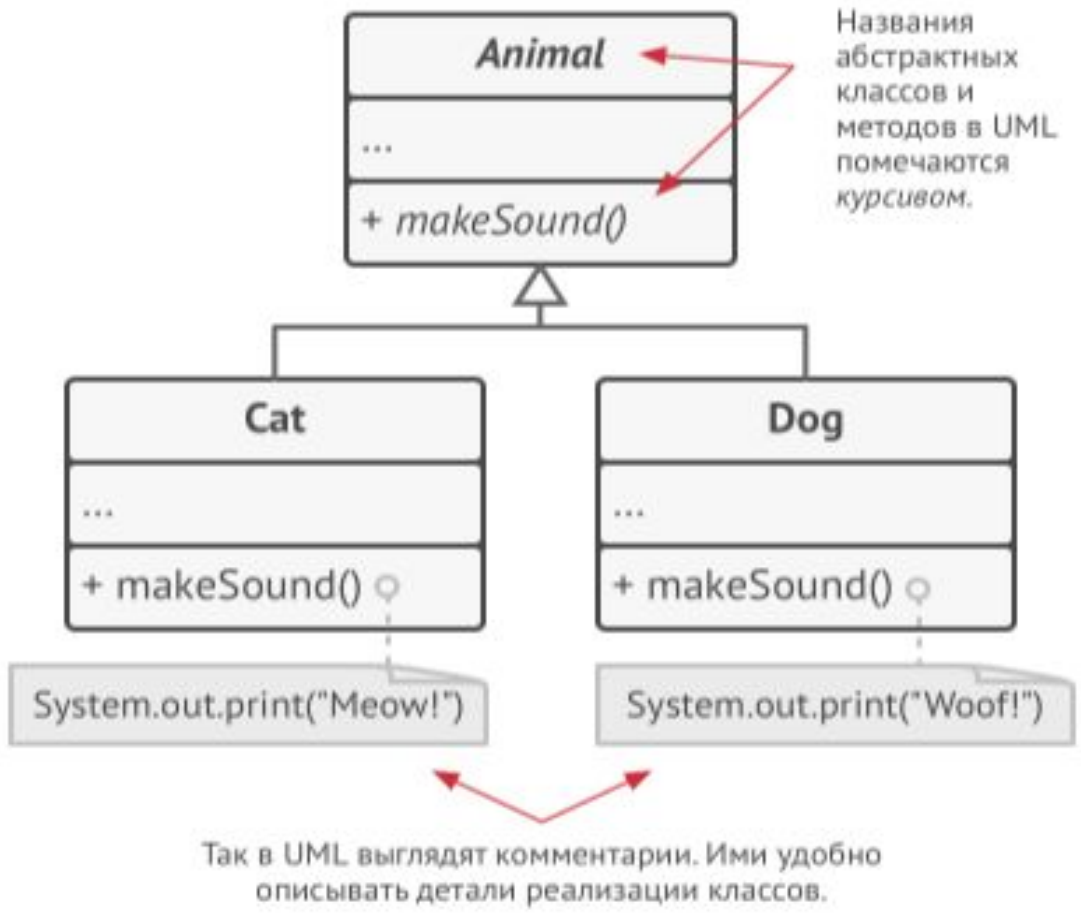
interface – это контракт, в рамках которого части программы взаимодействуют между собой. Это позволяет создавать модульные конструкции, в которых для изменения одного элемента не нужно трогать остальные.



Полиморфизм

Полиморфизм – это способность программы выбирать различные реализации при вызове методов с одним и тем же названием.

Частные случаи полиморфизма – перегрузка функций и перегрузка операций.




```
1  bag = [new Cat(), new Dog()];
2
3  foreach (Animal a : bag)
4    a.makeSound()
5
6  // Meow!
7  // Bark!
```