Концепции ООП по Гради Бучу

Объектно-ориентированный анализ и проектирование

- 7 элементов объектной модели
- ООА, ООД, ООП
- Практические примеры на C++

Александра Волосова,
к.т.н., доцент кафедры
ИУ5

Презентация разработана в рамках гранта «Грант на обучение студентов по образовательным программам высшего образования для топ-специалистов в сфере информационных технологий. Договор № 70-2025-000850 с АНО «Аналитический центр при Правительстве Российской Федерации»

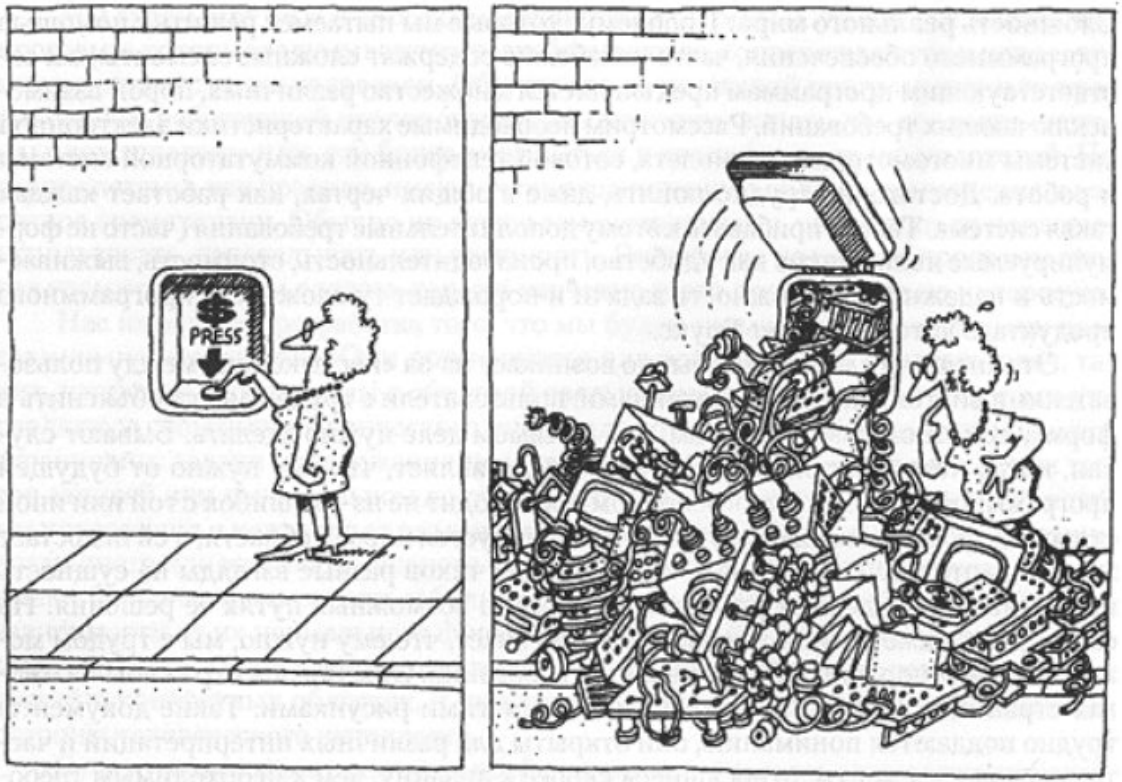
Причины появления объектно-ориентированного подхода (ООП)

- Прогресс в области архитектуры ЭВМ
- Развитие языков программирования: Simula, Smalltalk, CLU, ADA
- Развитие методологии проектирования, включая принцип модульности и принцип скрытия данных
- Развитие теории БД
- Исследования в области ИИ
- Достижения философии и теории познания

Причины появления ООП

- Сложность программных систем
- Разработчик хочет создать простую в использовании систему
- Сложные структуры часто принимают форму иерархии
- Усиление познавательных способностей через декомпозицию, выделение иерархий и абстракцию
- Сложные системы удобно исследовать, концентрируясь на объектах и процессах. Предметная область (мир) может рассматриваться, как упорядоченная совокупность объектов, которые в процессе взаимодействия определяют поведение системы
- Объектно-ориентированный анализ и проектирование – метод использующий объектную декомпозицию

Трудности управления процессом разработки: Разработчик хочет создать простую в использовании систему



5 признаков сложной системы

- Иерархичность и наличие подсистем, которые также могут быть структурированы
- Выбор компонентов системы в качестве элементарных осуществляется на усмотрение исследователя
- Внутрикомпонентная связь сильнее, чем связь между компонентами. Это позволяет разделить функции между частями системы и относительно изолированно изучать каждую часть
- Иерархические системы обычно состоят из немногих типов подсистем, по-разному скомбинированных и организованных
- Любая работающая сложная система является результатом развития работавшей более простой системы

Методы проектирования сложных систем к моменту появления ООП

- Метод структурного проектирования сверху вниз
- Метод структурного проектирования снизу вверх
- Метод потоков данных
- Объектно-ориентированное проектирование

Объект

- Понятие «объект» - попытка (20-е годы) найти компромисс между высоким уровнем программной абстракции и низким уровнем вычислительного устройства

Объектно-ориентированный подход к проектированию:

- Программную систему необходимо проектировать, как совокупность взаимодействующих друг с другом объектов
- Каждый объект рассматривается, как экземпляр определенного класса
- Классы образуют иерархию

Объектно-ориентированная модель

Эволюция объектной модели

Тенденции в проектировании:

- Смещение от программирования деталей к программированию компонентов
- Развитие языков высокого уровня
- Переход от императивных языков к декларативным
- Появление объектных и объектно-ориентированных языков

Поколения языков программирования

Первое поколение (1954-1958):

- FORTRAN I, ALGOL-58, Flowmatic
- Математические формулы

Второе поколение (1959-1961):

- ALGOL-60, COBOL, Lisp
- Подпрограммы, типы данных

Третье поколение (1962-1970):

- Pascal, Simula, PL/I
- Классы, абстрактные данные

Поколения языков программирования

- **1950-1960-е**: Концепции, которые позже войдут в ООП, зреют в других парадигмах (структуры данных в ALGOL, идеи инкапсуляции).
- **1967:** Simula 67 — первый язык с ключевыми понятиями: классы, объекты, наследование, виртуальные методы . Это истинный "первопроходец".
- **1970-е:** Smalltalk (Alan Kay и др.) — создает цельную, "чистую" объектно-ориентированную среду, вводит термины "объектно-ориентированное программирование", "полиморфизм", "динамическая типизация". Это основа классического ООП.
- **1980-е:** Появление языков, сочетающих ООП с другими популярными парадигмами. C++ (Бьёрн Страуструп) добавляет классы и объекты к C. Objective-C , Delphi (Object Pascal) . Eiffel (Бертран Мейер) развивает идеи с акцентом на надежность и контракты.
- **1990-е — 2000-е**: Java и C# становятся промышленными стандартами, предлагая "управляемую" модель ООП с виртуальной машиной и сборкой мусора. Python , Ruby — динамические языки с мощной ОО-моделью. В этот период ООП становится ведущей парадигмой для индустрии.
- **2010-е — настоящее время:** Мультипарадигмальность. Новые языки редко бывают "чисто ОО". Они заимствуют лучшие идеи из функционального программирования, например:
  - Scala — гибрид ООП и ФП на JVM.
  - Kotlin — более современная и лаконичная альтернатива Java.
  - Swift — сочетает ООП, протоколы (как в ООП), элементы ФП.
  - Rust — делает акцент на безопасности памяти и владении, используя некоторые ОО-концепты (типажи, структуры) неклассическим образом.
  - Go — имеет очень минималистичную модель ООП (интерфейсы, встраивание структур) без классического наследования.

Современные ООЯ:
Java, C#, C++, Python, Kotlin, Swift, TypeScript, Ruby, PHP (с версии 5+), Dart и многие другие.
Большинство из них — мультипарадигмальные .

Объектно-ориентированные языки

Особенности:

- Поддержка наследования
- Полиморфизм (возможен благодаря наличию иерархии и типизации)
- Инкапсуляция

Полиморфизм

- Возможность объектов с одинаковым интерфейсом иметь разную реализацию
- Позволяет обращаться с объектами разных классов через общий интерфейс
- Включает перегрузку методов и операторов
- Реализуется через виртуальные методы и динамическое связывание

7 элементов объектной модели

Основные элементы*:

1. **Абстрагирование**
2. **Инкапсуляция**
3. **Модульность**
4. **Иерархия**

Дополнительные элементы:

5. **Типизация**
6. **Параллелизм**
7. **Сохраняемость**

1. Абстрагирование

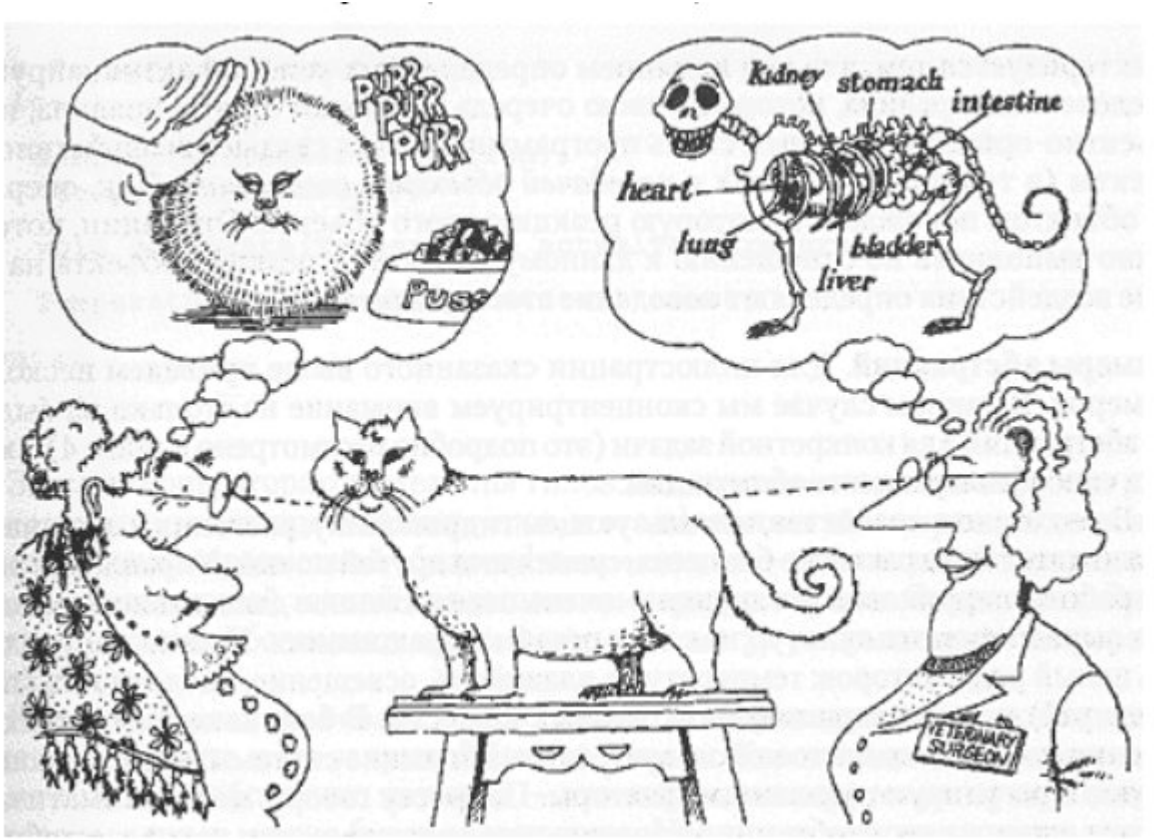
Выделение существенных характеристик объекта, отличающих его от всех других видов объектов

Принципы:

- Барьер абстракции
- Минимизация связей
- Принцип наименьшего удивления

Пример: Датчик температуры, План выращивания

Абстракция



фокусируется на существенных с точки зрения наблюдателя характеристиках объекта

2. Инкапсуляция

Сокрытие реализации объекта

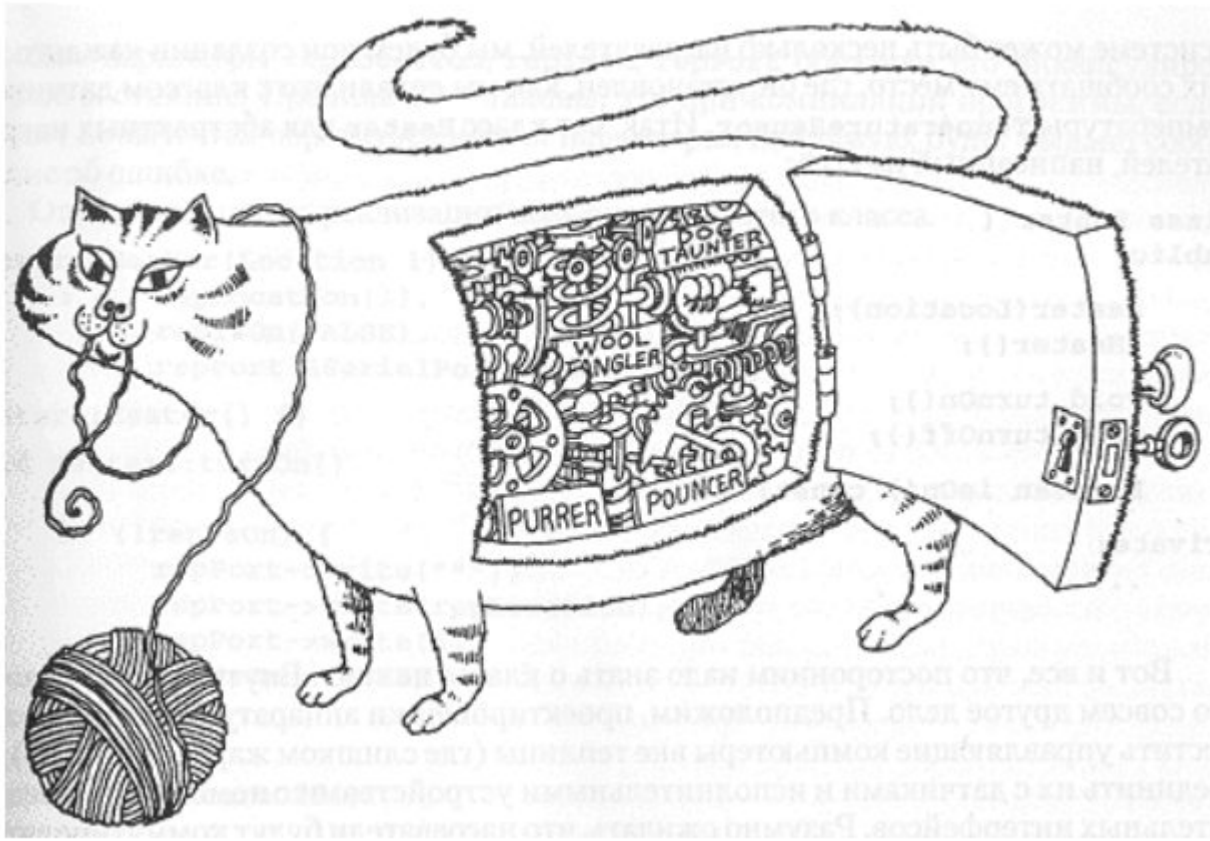
Ключевые аспекты:

- Разделение интерфейса и реализации
- Защита данных
- Предоставление интерфейса

В C++:

- private: только для класса
- protected: для класса и подклассов
- public: для всех

Инкапсуляция



скрывает детали реализации объекта

3. Модульность

Физическое разделение программы на фрагменты, которые компилируются отдельно.

Реализация в языках:

- C++: .h и .cpp файлы
- Ada: package (спецификация + тело)
- Object Pascal: unit

Правило Парнаса:

«Связи между модулями — это их представления друг о друге»

(**«Каждый модуль должен скрывать от остальных некоторую хорошо определенную тайну»**)

Модульность

Модульность позволяет хранить абстракции раздельно

	Основное обозначение	Альтернативы	Модули
Microsoft/Windows	.cpp	.cxx, .cc	.ixx
Google/Chromium	.cc	.cpp	статус эксперимента
LLVM/Clang	.cpp, .cc	.cxx	.cppm
Unix/GNU	.cc, .cpp	.C, .cxx	.cppm
CMake	.cpp	-	.ixx, .cppm
Embedded/IoT	.cpp	.cc	редко

Эволюция понятия "Модуль"

1. До 1960-х: Модуль - набор процедур
FORTRAN (1957) - нет модулей в современном смысле

```
SUBROUTINE CALCULATE(A, B, RESULT)
    REAL A, B, RESULT
    RESULT = A + B * 2.0
END
```

2. 1970-е: Модуль - unit (Паскаль, Modula)
// Pascal Unit (1970) - уже близко к ООП

```
unit StringUtils;
interface
   function StrLength(s: string): integer;
implementation function StrLength(s: string):
integer;
   begin // Реализация скрыта
   end;
end.
```

3. 1980-е: Модуль - класс (ООП)
// C++ (1983) - модуль становится классом

```
class String {
public:  // Интерфейс модуля
   size_t length() const;
private: // Скрытая реализация
    char* data;
    size_t size;
};
```

4. 2020-е: Модуль - единица компиляции (C++20)
// C++20 - возвращение к концепции 1970-х, но на новом уровне

```
export module string_lib; // Современный модуль
export class String {
 size_t length() const;
private: // ...
};
```

4. Иерархия

Упорядочение абстракций по уровням.

Два вида иерархий:

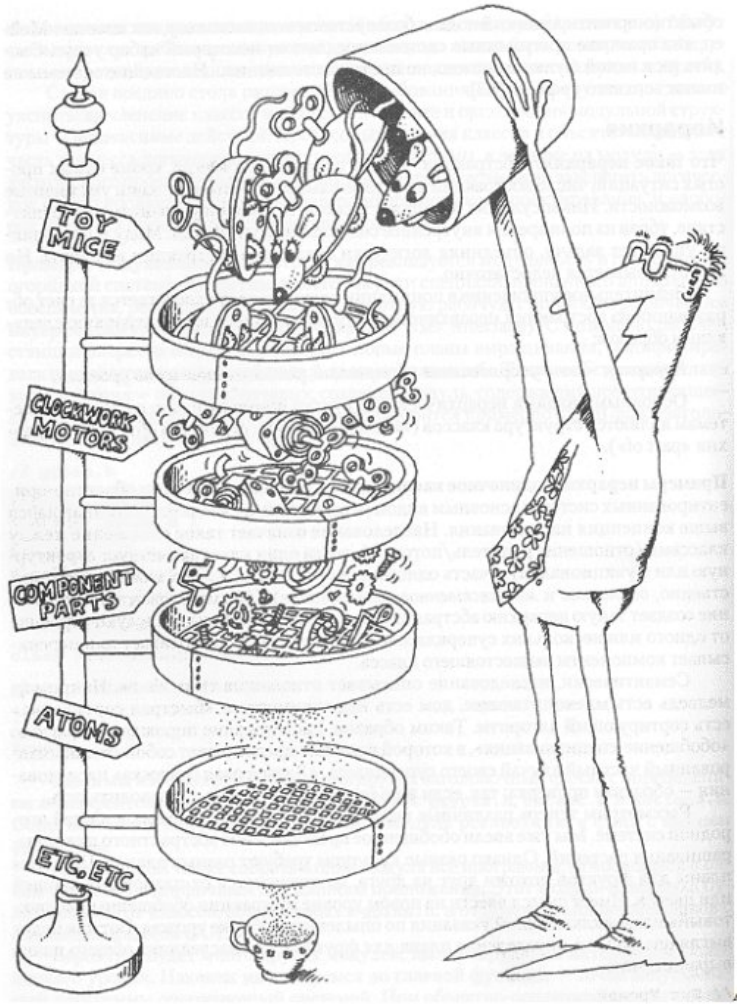
1. Классов **(is-a):**
   - Наследование
   - **ПланВыращиванияФруктов is-a ПланВыращивания**
2. Объектов **(part-of):**
   - Агрегация
   - **Растение part-of Сад**

Наследование:

- Позволяет создавать новые классы на основе существующих
- Обеспечивает повторное использование кода
- Создает иерархию классов "родитель-потомок"
- Поддерживает полиморфизм через переопределение методов
- Может быть множественным в некоторых языках программирования

Иерархия

Абстракции образуют иерархию



5. Типизация

Обеспечение корректного использования объектов через систему типов

Виды типизации:

• **Сильная**

(TypeScript, Rust, Kotlin, Go, Swift) / слабая (JavaScript, PHP)

```
// Сильная типизация – обнаруживаются ошибки типов
 // TypeScript
 let x: number = 10;
x = "текст"; // Ошибка компиляции
// Слабая типизация  - неявные преобразования
//JavaScript
let y = 10;
y = "текст"; // строка заменит число
```

• **Статическая / динамическая**

(Rust, TypeScript, Java, C# / Python, Ruby, JavaScript

```
// RUST, Статическая типизация - проверка при компиляции
fn add(a: i32, b: i32) -> i32 {
    a + b }
add(5, "текст"); //Ошибка компиляции
// Python, Динамическая типизация - проверка в runtime
def add(a, b):
      return a + b
add(5, "текст") # Скомпилируется, ошибка # - при запуске
```

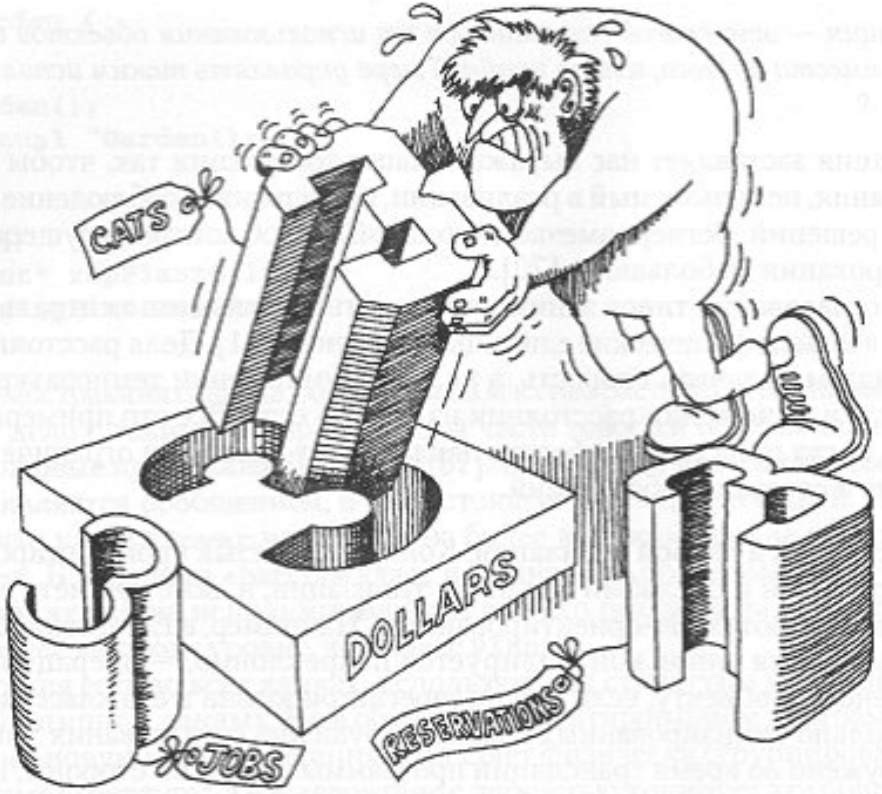
Преимущества:

• Раннее обнаружение ошибок

• Улучшенная документация

• Более эффективный код

Типизация



Строгая типизация предотвращает смешивание абстракций

5. Типизация

Обеспечение корректного использования объектов через систему типов

Виды типизации:

- **Явная / неявная**

(*Java, C* **) / *Rust, Kotlin, Swift***

```
// Java, Явная - тип указывается программистом
String name = "Анна";
int age = 25;
// kotlin, Неявная - компилятор определяет
val name = "Анна"
val age = 25
```

- **Современные виды:**

Градуальная (TypeScript, Python, PHP 8+),

Структурная (Go) и т.д.

6. Параллелизм

Способность различных объектов действовать одновременно.

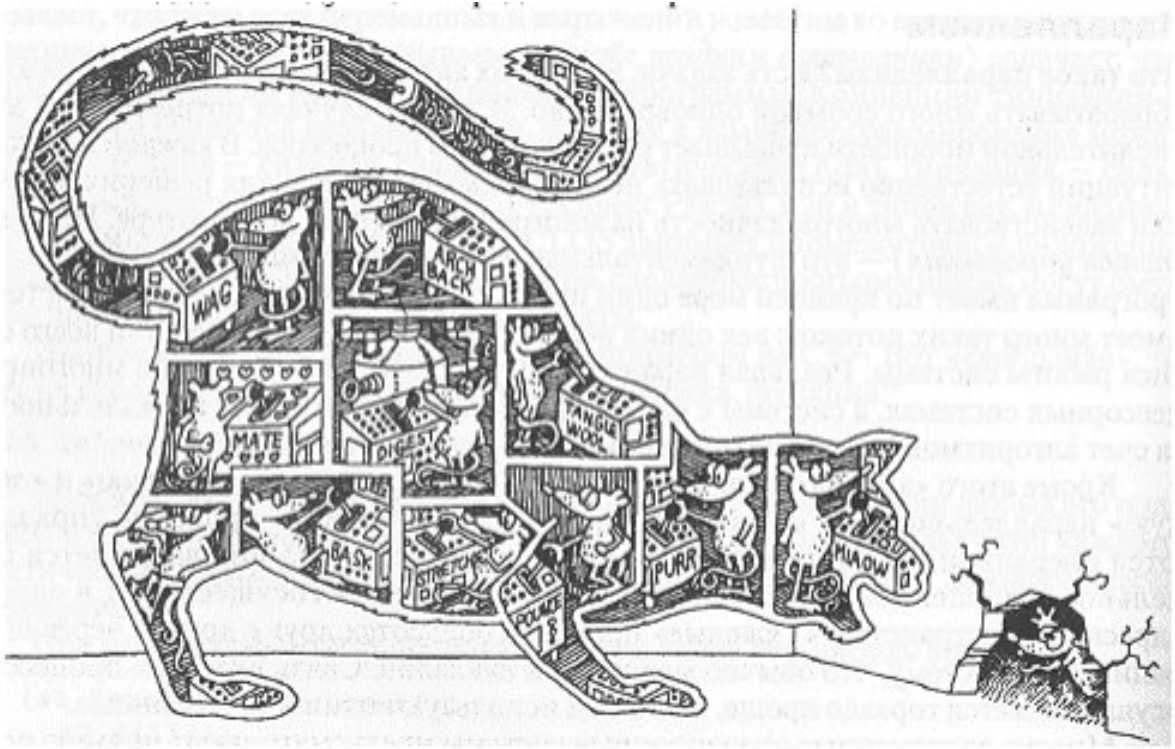
Ключевые понятия:

- Активные объекты (имеют поток управления)
- Пассивные объекты
- Взаимодействие через сообщения

Пример: **Датчик температуры**

- Периодически измеряет температуру
- Уведомляет другие объекты
- Использует обратные вызовы

Параллелизм



позволяет различным объектам действовать одновременно

7. Сохраняемость

Способность объекта существовать:

- Во времени (переживая процесс)
- В пространстве (перемещаясь между системами)

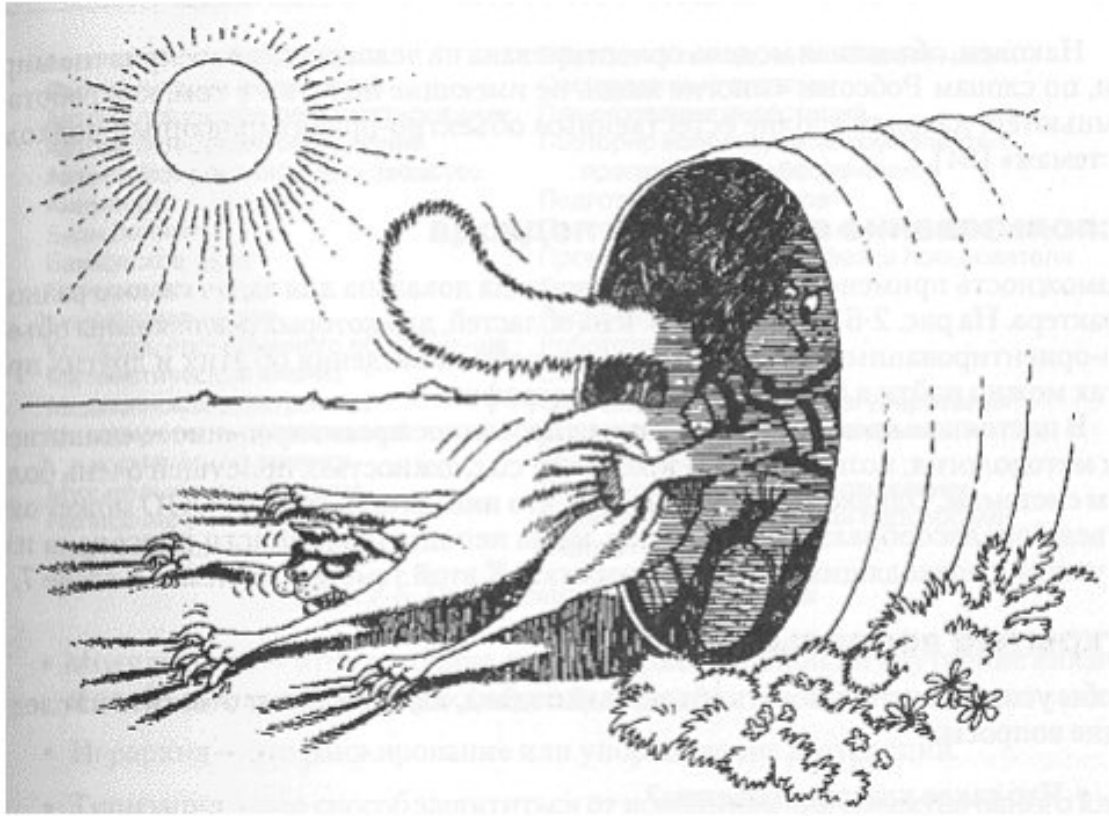
Применение:

- Объектно-ориентированные БД
- Распределенные системы
- Сохранение состояния

Проблемы:

- Сохранение классов и данных
- Изменение схемы
- Миграция объектов

Сохраняемость



поддерживает состояние и класс объекта в пространстве и во времени

ООА, ООД, ООП

Объектно-ориентированный анализ (ООА):

- Выявление классов и объектов
- Моделирование реальности
- Пример: система управления теплицей

Объектно-ориентированное проектирование (ООД):

- Проектирование архитектуры
- Использование диаграмм
- Статические и динамические аспекты

Объектно-ориентированное программирование (ООП):

- Реализация в языках
- Создание экземпляров
- Взаимодействие объектов

Пример системы (теплица)

Ключевые классы:

1. TemperatureSensor - датчик температуры
2. Heater - нагреватель
3. GrowingPlan - план выращивания
4. FruitGrowingPlan - план для фруктов
5. Plant - растение
6. Garden - огород
7. WaterTank - емкость для воды
8. Inventory - инвентарный список

Все классы взаимодействуют через четко определенные интерфейсы

Преимущества объектной модели

1. Полное использование возможностей ОО языков

2. Повышение унификации разработки и повторного использования

3. Построение на основе стабильных промежуточных описаний

4. Снижение риска разработки сложных систем

5. Естественность моделирования (ориентация на человеческое восприятие)

Области применения

- Авиационное оборудование
- Базы данных
- Банковское дело
- Верификация ПО
- Операционные системы
- Робототехника
- Телекоммуникации
- Управление воздушным движением
- Системы управления и регулирования
- Экспертные системы

Выводы

- ООП = анализ + проектирование + программирование

- 7 элементов объектной модели:
  абстрагирование, инкапсуляция, модульность,
  иерархия, типизация, параллелизм, сохраняемость

- Объект: состояние + поведение + идентичность

- Иерархии: is-a (наследование) и part-of (агрегация)