Объектно-ориентированное программирование
Лекция 2. Парадигмы

П. А. Макаров

СЫКТЫВКАРСКИЙ
ЛЕСНОЙ
ИНСТИТУТ

22 сентября 2023 г.

Содержание

Парадигмы как явление

Слово “парадигма” заимствовано из греческого языка ($\pi\alpha\rho\alpha\delta\epsilon\iota\gamma\mu\alpha$ — пример/модель).

Определение 1

*Согласно Т. Куну, **парадигма** — это устоявшаяся система научных взглядов, в рамках которой ведутся исследования. Парадигмы сменяют друг друга, уточняя и улучшая наши представления о том или ином явлении.*

Пример 1

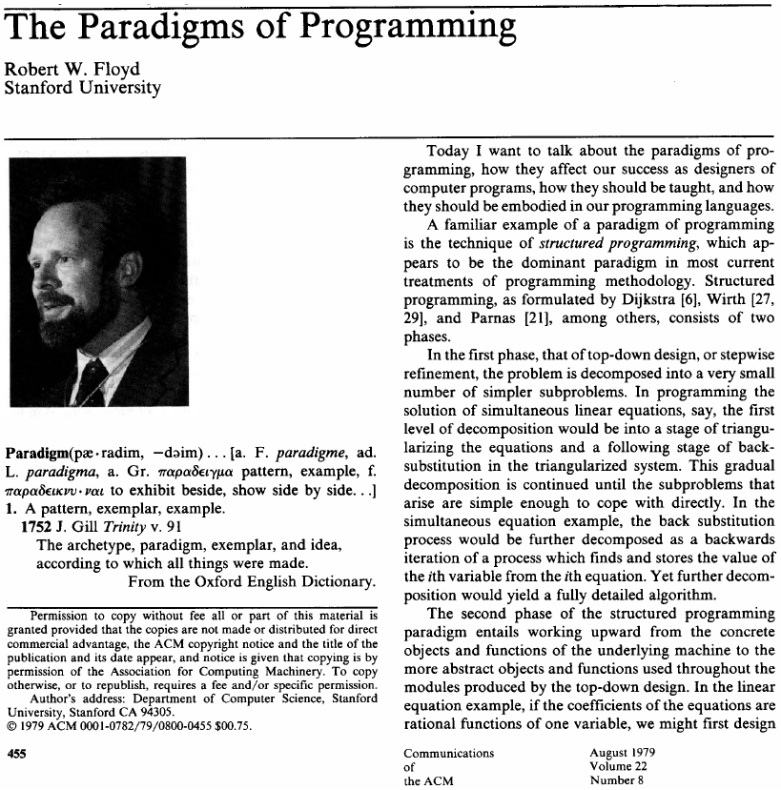
Геоцентрическая небесная механика Птолемея и гелиоцентрическая система Коперника.

Некоторые элементы философии UNIX, которую можно представлять как некую общую парадигму разработчика:

- универсальная парадигма “всё есть поток байтов”;
- всё есть текст;
- каждая программа должна решать одну задачу, и делать это хорошо;
- KISS — Keep It Simple, Stupid!

- Модульное и структурное программирование.
- Событийно-управляемое программирование.
- Параллельное программирование.
- Программирование с разделяемыми данными.
- Рекурсия как парадигма.

Рис. 1: Роберт Флойд и его статья



The Paradigms of Programming

Robert W. Floyd
Stanford University

Paradigm(pæ·radim, −dəim) . . . [a. F. *paradigme*, ad. L. *paradigma*, a. Gr. *παραδειγμα* pattern, example, f. *παραδεικνυ·ναι* to exhibit beside, show side by side. . .]
1. A pattern, exemplar, example.
1752 J. Gill *Trinity* v. 91
The archetype, paradigm, exemplar, and idea, according to which all things were made.
From the Oxford English Dictionary.

Author's address: Department of Computer Science, Stanford University, Stanford CA 94305.

Today I want to talk about the paradigms of programming, how they affect our success as designers of computer programs, how they should be taught, and how they should be embodied in our programming languages.

A familiar example of a paradigm of programming is the technique of *structured programming*, which appears to be the dominant paradigm in most current treatments of programming methodology. Structured programming, as formulated by Dijkstra [6], Wirth [27, 29], and Parnas [21], among others, consists of two phases.

In the first phase, that of top-down design, or stepwise refinement, the problem is decomposed into a very small number of simpler subproblems. In programming the solution of simultaneous linear equations, say, the first level of decomposition would be into a stage of triangularizing the equations and a following stage of back-substitution in the triangularized system. This gradual decomposition is continued until the subproblems that arise are simple enough to cope with directly. In the simultaneous equation example, the back substitution process would be further decomposed as a backwards iteration of a process which finds and stores the value of the *i*th variable from the *i*th equation. Yet further decomposition would yield a fully detailed algorithm.

The second phase of the structured programming paradigm entails working upward from the concrete objects and functions of the underlying machine to the more abstract objects and functions used throughout the modules produced by the top-down design. In the linear equation example, if the coefficients of the equations are rational functions of one variable, we might first design

Рис. 1: Роберт Флойд и его статья

Определение 2

Парадигма программирования *— комплекс концепций, используемых программистом при проектировании программы.*

Обзор основных парадигм

- Императивное программирование.
- Процедурное программирование.
- Функциональное программирование.
- Логическое программирование.
- ООП.

Базовые термины

- объект, свойства и методы,
- события,
- интерфейс,
- класс.

Ключевые идеи ООП (four pillars of Object Oriented Programming)

- инкапсуляция (encapsulation),
- абстракция данных (data abstraction),
- полиморфизм (polymorphism),
- наследование (inheritance).

Определение 3

Инкапсуляция — это механизм сокрытия реализации данных путем ограничения доступа к публичным методам. Для этого переменные данного экземпляра класса (конкретного объекта) сохраняются приватными (private), а методы-аксессоры делаются доступными (public).

```
1  class Employee {
2  private:
3      String name;
4      Date dob;
5  public:
6      String getName() { return name; }
7      void setName(String name) { this.name = name; }
8      Date getDob() { return dob; }
9      void setDob(Date dob) { this.dob = dob; }
10 };
```

Листинг 1: Пример инкапсуляции

Определение 4

Абстракция — подход, в котором концепции или идеи, отделяются от конкретного экземпляра класса. Используя абстрактный класс/интерфейс, мы выражаем "намерение" класса, а не его фактическую реализацию. В некотором смысле, один класс не должен знать внутренние детали другого, чтобы использовать его, достаточно знать интерфейсы.

Определение 5

Наследование — способность одного объекта (класса) базироваться на другом объекте (классе). Это главный механизм для повторного использования кода. Наследственное отношение классов четко определяет их иерархию.

Определение 6

Полиморфизм — реализация некоторым множеством способов различных задач с одним и тем же названием (и сходным смыслом).

- *Статический полиморфизм достигается с помощью перегрузки методов (method overloading);*
- *Динамический полиморфизм — с помощью переопределения методов (method overriding). Тесно связан с наследованием. С помощью динамического полиморфизма код, который работает на суперклассе (superclass), будет работать с любым типом подкласса (subclass).*

```
 1 #include <iostream>
 2
 3 void print(char * s) {
 4   std::cout << s << std::endl;
 5 }
 6 void print(char *a[], int N) {
 7   int i = 0;
 8   while( i < N ) {
 9     print(*a++);
10     i++;
11   }
12 }
13
14 int main(int argc, char **argv) {     // char *argv[] <--- ?
15   char string[] = "Hello, world!";
16   print(string);
17   print(++argv, --argc);     // print(argv++, argc--) <--- ?
18   return 0;
19 }
```

Листинг 2: Пример статического полиморфизма в C++

```
1 import sys
2
3 var = 'Hello, world!'
4 print(var)
5
6 var = sys.argv
7 print(var[1:])
```

Листинг 3: Пример полиморфизма в Python

- Не повторяйся (DRY — Don't repeat yourself).
- Принцип единственной обязанности.
- Принцип открытости/закрытости.
- Принцип подстановки Барбары Лисков.
- Принцип разделения интерфейсов.
- Принцип инверсии зависимостей.

Совместимость C и C++

```
1 int try;
2 long new;
```

Листинг 4: Ключевые слова

```
1 struct mystruct {
2     int a, b;
3 };
```

Листинг 5: Структуры в C++

```
1 typedef struct mystruct {
2     int a, b;
3 } mystruct;
```

Листинг 6: Структуры в C

```
1 struct point {
2     double x, y, z;
3 };
4 point A, B, C;  // struct point A, B, C;
```

Листинг 7: Использование структур в C++

Контрольные вопросы и задания

1. Изучите основные идеи философии UNIX, сформулированные Дугом Макилроем, Майком Ганцарзом и Эриком Рэймондом.
2. Ознакомьтесь с работой Роберта Флойда.
3. В чём разница между параллельным программированием и программированием с разделяемыми данными? Действительно ли это различные концепции? Почему?
4. Какие языки позволяют следовать парадигмам, перечисленным в разделе 3?
5. Есть ли существенная разница между императивной и процедурной парадигмами? Какие чисто императивные языки вам известны?
6. Определите значения всех понятий, упомянутых в разделе 4.
7. Изучите листинги 2 и 3. В чём принципиальное различие между ними? Можно ли его устранить? Что для этого потребуется?
8. Какие парадигмы позволяет использовать программисту современный C++? Python?