

Объектно-ориентированное программирование. Лекция 2. Парадигмы

Макаров П. А.

19 сентября 2024 г.

Содержание

1	Парадигмы как явление	1
2	Парадигмы программирования	2
2.1	Парадигмы разработки как философия	2
2.2	Системное и прикладное программирование	2
2.3	Современный смысл термина	3
3	Обзор основных парадигм	3
4	Объектно-ориентированное программирование	3
4.1	Базовые термины	3
4.2	Ключевые идеи ООП	4
4.3	Основные принципы ООП	5
5	Совместимость С и С++	6
6	Контрольные вопросы и задания	7

1 Парадигмы как явление

У термина парадигма нет строгого определения, различные авторы определяют его по-разному. Само слово “парадигма” заимствовано из греческого языка (*παράδειγμα* — пример/модель), но свое современное значение в научно-технической литературе получило в работе “Структура научных революций” философа Томаса Куна.

Определение 1. Согласно 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.

Чаще всего слово парадигма употребляется тогда, когда речь идёт о различных способах восприятия людьми одного и того же явления.

2 Парадигмы программирования

2.1 Парадигмы разработки как философия

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!

2.2 Системное и прикладное программирование

В системном программировании выбор парадигм ограничивается требованиями, предъявляемыми к программам и очень сильно связан с особенностями применяемой программно-аппаратной архитектуры. Прикладное программирование в этом смысле значительно свободнее, и использование специфических парадигм в особых случаях способно в разы сократить трудозатраты на разработку.

- Модульное и структурное программирование.
- Событийно-управляемое программирование. В программе имеется *главный цикл*, одна его итерация соответствует наступлению некоего события из определённого множества, а все действия программы — это реакции на событие. Тело главного цикла делится на две стадии: *выборка события* и *обработка события*.

- Параллельное программирование.
- Программирование с разделяемыми данными.
- Рекурсия как парадигма.

2.3 Современный смысл термина

Роберт Флойд первым ввёл термин “парадигма программирования” в своей работе “Парадигмы программирования” 1979 года. Он опирался на идею Томаса Куна, но не исключал различные парадигмы, а напротив, указывал на то, что они могут сочетаться, обогащая инструментарий программиста.

Определение 2. Парадигма программирования — комплекс концепций, используемых программистом при проектировании программы.

3 Обзор основных парадигм

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

4 Объектно-ориентированное программирование

4.1 Базовые термины

- объект, свойства (атрибуты) и методы,
- события,
- интерфейс,
- класс.

4.2 Ключевые идеи ООП

- 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 чётко 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 // Polymorphic print function
4 void print(char * s) {
5     std::cout << s << std::endl;
6 }
7 void print(char *a[], int N) {
8     int i = 0;
9     while( i < N ) {
10        print(*a++);
11        i++;
12    }
13 }
14
15 int main(int argc, char **argv) {    // char *argv[] <--- ?
16     // Passing a String
17     char string[] = "Hello, world!";
18     print(string);
19
20     // Passing an Array of Strings
21     print(++argv, --argc);    // print(argv++, argc--) <--- ?
22
23     return 0;
24 }
```

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

4.3 Основные принципы ООП

- Не повторяйся (DRY — Don't repeat yourself). Избегайте повторного написания кода, вынося в абстракции часто используемые задачи и

данные. Каждая часть вашего кода или информации должна находиться в единственном числе в единственном доступном месте. Это один из принципов читаемого кода.

- Принцип единственной обязанности. Для каждого класса должно быть определено единственное назначение. Все ресурсы, необходимые для его осуществления, должны быть инкапсулированы в этот класс и подчинены только этой задаче.
- Принцип открытости/закрытости. Программные сущности должны быть открыты для расширения, но закрыты для изменений.
- Принцип подстановки Барбары Лисков. Методы, использующие некий тип, должны иметь возможность использовать его подтипы, не зная об этом.
- Принцип разделения интерфейсов. Предпочтительнее разделять интерфейсы на более мелкие тематические, чтобы реализующие их классы не были вынуждены определять методы, которые непосредственно в них не используются.
- Принцип инверсии зависимостей. Система должна конструироваться на основе абстракций “сверху вниз”: не абстракции должны формироваться на основе деталей, а детали должны формироваться на основе абстракций.

5 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++

6 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?