

## Парадигмы программирования (часть 1)

Алексей Островский

Физико-технический учебно-научный центр НАН Украины

28 ноября 2014 г.

# Определение парадигмы

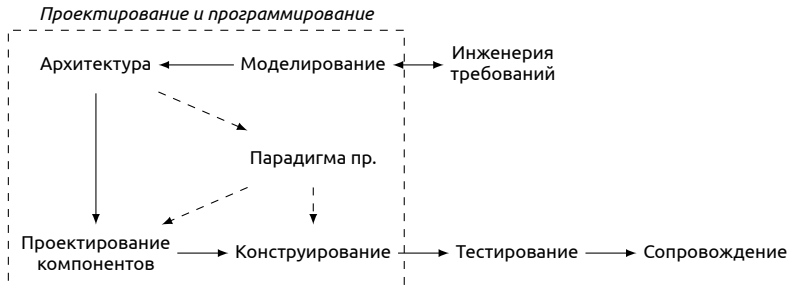
## Определение

**Парадигма программирования** (англ. *programming paradigm*) — совокупность идей и понятий, которые определяют общий стиль написания компьютерных программ, построения их структуры и отдельных элементов программной системы.

### Цель парадигмы программирования:

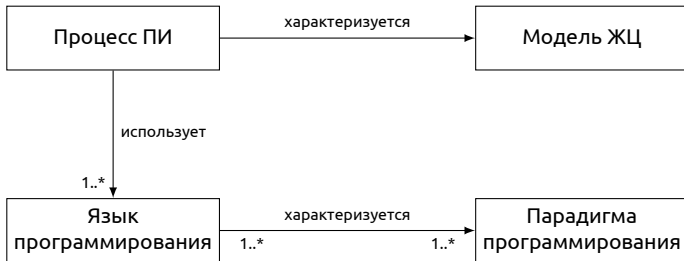
- ▶ разделение программы на базовые составные элементы (напр., функции или объекты);
- ▶ определение модели преобразования данных;
- ▶ внедрение ограничений на используемые конструкции.

## Место парадигмы в разработке



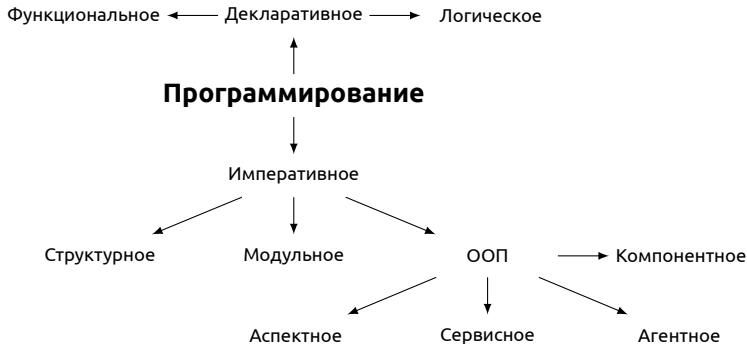
Роль парадигм программирования в разработке ПО. Парадигма выбирается, исходя из архитектуры системы и влияет на построение и реализацию ее компонентов

## Место парадигмы в разработке



Связь парадигм с языками программирования подобна связи процесса программной инженерии с выбранной моделью разработки

# Классификация парадигм программирования



# Декларативное программирование

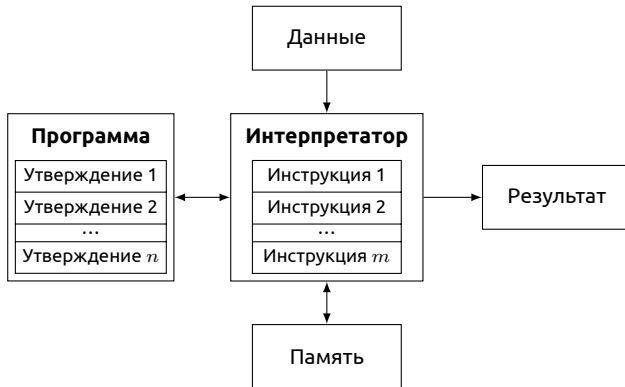
## Определение

**Декларативное программирование** (англ. *declarative programming*) — парадигма, согласно которой программа представляет логику вычислений без описания прямой последовательности действий (действия определяются компилятором или интерпретатором).

### Области использования:

- ▶ математическое моделирование;
- ▶ искусственный интеллект;
- ▶ анализ данных;
- ▶ наука.

## Выполнение декларативной программы



Декларативная программа не взаимодействует напрямую с памятью, поручая эту работу интерпретатору

# Императивное программирование

## Определение

**Императивное программирование** (англ. *imperative programming*) — парадигма, согласно которой программа представляет собой последовательность действий, изменяющих *состояние программы*.

## Определение

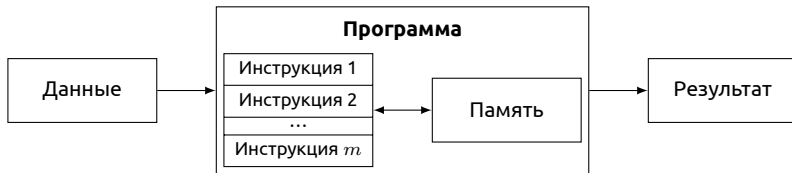
**Состояние программы** (англ. *program state*) — совокупность данных, связанных со всеми используемыми программой переменными в конкретный момент времени.

### Область использования:

- ▶ системные программы;
- ▶ прикладные программы.



## Выполнение императивной программы



Императивная программа использует именованные области памяти (переменные) для хранения состояния вычислений

# Функциональное программирование

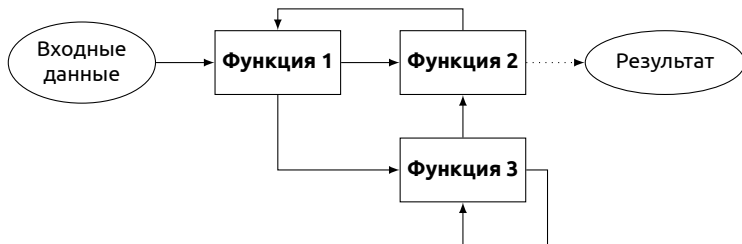
## Определение

**Функциональное программирование** (англ. *functional programming*) — парадигма, согласно которой процесс исполнения программы представляется последовательностью вычислений значений для математических функций.

### Особенности:

- ▶ отказ от явного хранения переменных (функции без побочных эффектов);
- ▶ ⇒ встроенная поддержка параллелизации, оптимизации и кэширования без необходимости действий со стороны программиста.

## Функциональное программирование (продолжение)



Функции возвращают результат, иначе не меняя состояние программы (напр., через переменные).

**Ключевые слова:** чистая функция, прозрачность ссылок.

## Функциональное программирование (продолжение)

### Концепции:

- ▶ функции высших порядков — функции, которые возвращают другие функции или принимают функции в качестве аргументов;
- ▶ замыкание (англ. *closure*) — сохранение контекста функции при ее создании;
- ▶ рекурсия для создания циклов;
- ▶ ленивые вычисления (англ. *lazy evaluation*) — вычисление аргументов функций по мере необходимости (не при задании).

**Языки программирования:** Lisp, Scheme, Clojure, Erlang, Haskell, F#.

# Логическое программирование

## Определение

**Логическое программирование** (англ. *logic programming*) — парадигма, согласно которой программа представляет собой вывод с помощью правил формальной логики.

### Правило вывода:

$$H :- B_1, \dots, B_n, \quad \text{т.е.} \quad H, \text{ если } B_1 \text{ и } \dots \text{ и } B_n.$$

### Особенности:

- ▶ данные представляются как факты (безусловные утверждения, аксиомы);
- ▶ результат выполнения — данные, для которых истинно некоторое утверждение.

## Логическое программирование (продолжение)

### Концепции:

- ▶ процедурная интерпретация правил:

$$H :- B_1, \dots, B_n;$$

чтобы вычислить  $H$ , нужно вычислить  $B_1, \dots, B_n$ .

- ▶ контроль над стратегией доказательства утверждений (параллельный или последовательный поиск, перебор с возвратом, ...).

**Языки программирования:** Prolog и диалекты, Oz.

# Структурное программирование

## Определение

**Структурное программирование** (англ. *structured programming*) — парадигма, в основе которой лежит представление программы в виде иерархии *блоков*.

## Определение

**Блок инструкций** — логически связанный набор последовательных инструкций, предназначенный для:

- ▶ ограничения области видимости;
- ▶ обращения к блоку как к одной инструкции.

### Особенности:

- ▶ каждый блок должен иметь строго один вход и выход;
- ▶ ⇒ отказ от безусловных переходов (*goto*).

## Структурное программирование (продолжение)

### Элементы программ:

Конструкция	Псевдокод
последовательность	$S_1; S_2; S_3; \dots$
ветвление	if $C$ then $S_1$ else $S_2$
цикл	while $C$ do $S$
подпрограммы	function $F(A_1, A_2, \dots, A_n)$ $B$

### Теорема (Бём, Якопини)

Программы, использующие управляющие конструкции последовательности, ветвления и цикла, способны вычислить любую вычислимую функцию.



## Структурное программирование (продолжение)

### Концепции:

- ▶ выделение повторяющегося кода в подпрограммы (процедуры и функции);
- ▶ объединение логически связанных подпрограмм в модули;
- ▶ использование блоков инструкций для контроля области видимости переменных и функций;
- ▶ проектирование программ сверху вниз;
- ▶ разделение интерфейсов и имплементаций подпрограмм в модулях.

**Языки программирования:** Algol, C, Pascal, Ada, FORTRAN, PL/I.

# Объектно-ориентированное программирование

## Определение

**Объектно-ориентированное программирование** (англ. *object-oriented programming*) — парадигма, согласно которой программа представляется в виде взаимодействующих объектов.

## Определение

**Объект** — сильная связь между структурами данных и методами ( $\simeq$  функциями), обрабатывающими эти данные. Составляющие объекта:

- ▶ идентификатор;
- ▶ свойства;
- ▶ методы.

**Особенность:** развитие структурного программирования с объединением в объектах данных и поведения.

## Концепции ООП

- ▶ **объекты**;
- ▶ **инкапсуляция** — скрывание информации от внешних (по отношению к системе или объекту) сущностей;
- ▶ **наследование** — повторное использование методов работы с данными в различных условиях; дополнение функциональности объектов;
- ▶ **полиморфизм подтипов**<sup>1</sup> — возможность использования наследованных от объекта потомков в том же контексте, что и сам объект;
- ▶ **динамическая диспетчеризация** — выбор, какую реализацию виртуального метода следует вызывать, во время исполнения программы;
- ▶ **открытая рекурсия** — переменная `this/self`, позволяющая вызывать методы объекта из других методов.

<sup>1</sup> Полиморфизм есть и в структурном программировании, но [другой](#).

## Виды ООП

- ▶ ООП на основе **классов** — вид ООП, в котором поведение объектов и наследование определяется с помощью классов (начальный набор данных + поведение).

### Языки программирования:

- ▶ C++;
  - ▶ Java;
  - ▶ C#;
  - ▶ Python.
- ▶ ООП на основе **прототипов** — вид ООП, в котором наследование поведения осуществляется с помощью клонирования существующих объектов (прототипов).

### Языки программирования:

- ▶ JavaScript;
  - ▶ Lua.

## Виды ООП (продолжение)

Характеристика	Вид ООП	
	На основе классов	На основе прототипов
Наследование	создание подклассов	клонирование прототипа
Содержимое объектов	определяется классом	определяется конкретным объектом
Связь между данными и поведением	сильная	слабая
Интерфейсы	подвид классов	duck typing

# Императивное и декларативное программирование

## Вычисление чисел Фибоначчи — императивная программа (Python):

```
1  def fib(n):
2      """ Вычисляет n-е число Фибоначчи (с отсчетом от нуля). """
3      if (n == 0):
4          return 0;
5      a, b = 0, 1;
6      for i in range(1, n):
7          a, b = b, a + b;
8      return b;
```

### Вызов:

```
>>> fib(10)
55
```

# Императивное и декларативное программирование

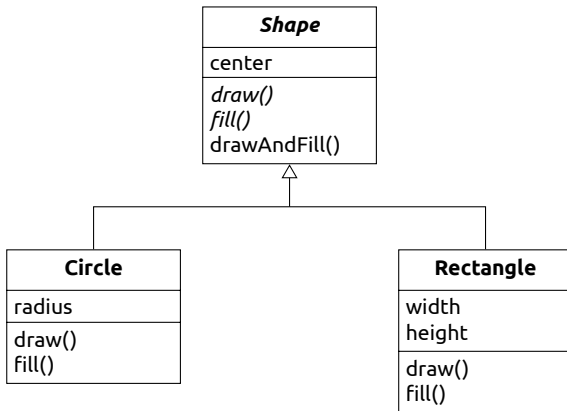
## Вычисление чисел Фибоначчи — декларативная программа (Prolog):

```
1  % Вычисление чисел Фибоначчи.
2  %  $\text{fib}(n, X) \Leftrightarrow X = F_n$ .
3  fib(N, X) :-
4      fibr(0, 1, N, X).
5
6  fibr(A, _, 0, X) :- X is A.
7  fibr(_, B, 1, X) :- X is B.
8  fibr(A, B, N, X) :-
9      N_1 is N - 1,
10     Sum is A + B,
11     fibr(B, Sum, N_1, X).
```

### Вызов:

```
?- fib(10, X).
X = 55.
```

## Процедурное программирование и ООП



Структурная модель предметной области



# Процедурное программирование и ООП

## Описание ПрО — процедурный стиль (C):

```

1  struct Rectangle {
2      Point center;
3      double width;
4      double height;
5  };
6  void drawRect(struct Rectangle rectangle) { /* ... */ }
7  void fillRect(struct Rectangle rectangle) { /* ... */ }
8
9  struct Circle {
10     Point center;
11     double radius;
12 };
13 void drawCircle(struct Circle circle) { /* ... */ }
14 void fillCircle(struct Circle circle) { /* ... */ }

```

## Процедурное программирование и ООП

### Описание ПрО — ООП (Java):

```

1  public abstract class Shape {
2      private Point center;
3      protected Shape(Point center) { /* ... */ }
4
5      public abstract void draw();
6      public abstract void fill();
7      public void drawAndFill() {
8          this.draw();
9          this.fill();
10     }
11 }
12
13 public class Circle extends Shape {
14     private double radius;
15     public Circle(Point center, double radius) { /* ... */ }
16
17     public void draw() { /* ... */ }
18     public void fill() { /* ... */ }
19 }
    
```

# Процедурное программирование и ООП

## Концепции ООП (Java):

```

1  /* Полиморфизм подтипов: Circle можно рассматривать как Shape. */
2  Shape s = new Circle(new Point(1, 1), 5);
3
4  /* Динамическая диспетчеризация: вызывается Circle.draw(), а не Shape.draw(). */
5  s.draw();
6
7  public abstract class Shape {
8      /* ... */
9
10     public void drawAndFill() {
11         /* Открытая рекурсия: при вызове s.drawAndFill() будут вызваны */
12         /* методы Circle.draw() и Circle.fill(). */
13         this.draw();
14         this.fill();
15     }
16 }
```

# Классы и прототипы

## Описание ПрО — прототипы (JavaScript):

```
1  function Shape(center) {  
2      this.center = center;  
3  };  
4  Shape.prototype.drawAndFill = function() {  
5      this.draw();  
6      this.fill();  
7  };  
8  
9  function Circle(center, radius) {  
10     Shape.call(this, center);  
11     this.radius = radius;  
12 }  
13 /* Клонирование прототипа. */  
14 Circle.prototype = Object.create(Shape.prototype);  
15 Circle.prototype.draw = function() { /* ... */ };  
16 Circle.prototype.fill = function() { /* ... */ };
```

# Классы и прототипы

## Особенности прототипов (JavaScript):

```
1  var shape = new Circle({'x': 1, 'y': 1}, 5);
2
3  /* Работает, несмотря на отсутствие методов .draw() и .fill() в Shape. */
4  shape.drawAndFill();
5  /* Заменяет метод для конкретного объекта. */
6  shape.draw = function() { /* ... */ };
7  /* Заменяет метод для всех объектов, создаваемых с помощью new Circle(...). */
8  Circle.prototype.draw = function() { /* ... */ };
9
10 var otherShape = {
11     'draw': function() { alert('Drawn!'); }
12     'fill': function() { alert('Filled!'); }
13     'drawAndFill': Shape.prototype.drawAndFill
14 };
15 /* Duck typing: работает, несмотря на отсутствие явного наследования. */
16 otherShape.drawAndFill();
```

## Выводы

1. Парадигма программирования определяет общий стиль написания программ. По своей сути парадигмы играют в проектировании и конструировании ту же роль, что и модель жизненного цикла в планировании разработки ПО.
2. Выделяют две основных группы парадигм программирования: декларативные (определяют цель, но не метод ее достижения) и императивные (определяют и то, и другое).
3. Объектно-ориентированное программирование — основная парадигма в разработке современных прикладных приложений. Ее особенность — представление предметной области в виде объектов, которые сочетают в себе данные и поведение.

# Материалы



**Abelson, Hal; Sussman, Gerald Jay.**

**Structure and Interpretation of Computer Programs.**

<http://mitpress.mit.edu/sicp/>

(функциональное программирование)



**Н. Вирт.**

**Алгоритмы + структуры данных = программы.**

(структурное программирование)



**Meyer, Bertrand.**

**Object-Oriented Software Construction.**

(ООП)

Спасибо за внимание!