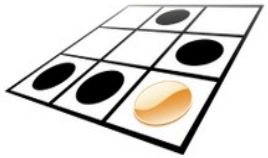


XXI конференция
разработчиков свободных
программ

НИУ ВШЭ
Департамент программной
инженерии

Расширение языка C для поддержки процедурно–параметрического программирования

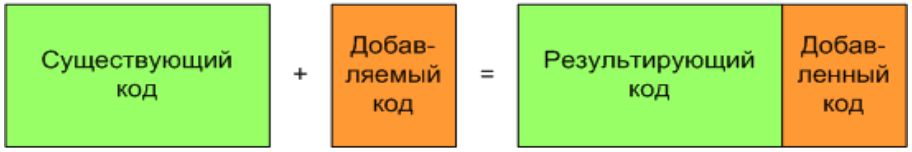
http://www.softcraft.ru/ppp/

Sep 2025	Sep 2024	Change	Programming Language	Ratings	Change
1	1		Python	25.98%	+5.81%
2	2		C++	8.80%	-1.94%
3	4	▲	C	8.65%	-0.24%
4	3	▼	Java	8.35%	-1.09%
5	5		C#	6.38%	+0.30%

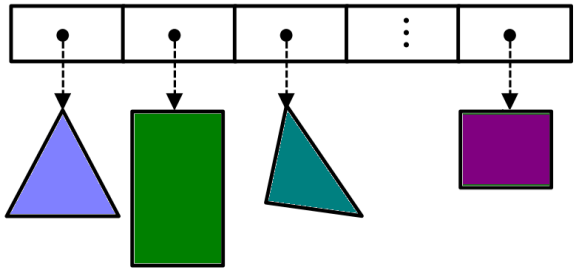
Александр Легалов
Павел Косов
Александр Сухих

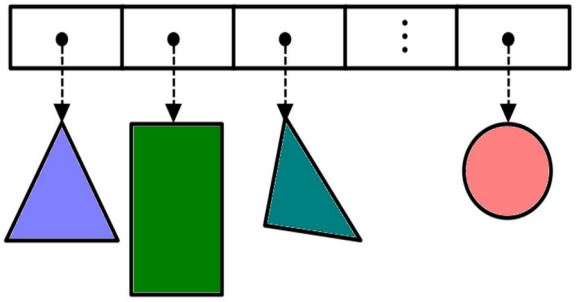
Динамическая проверка типов во время выполнения и гибкая разработка ПО

а) Изменение существующего кода

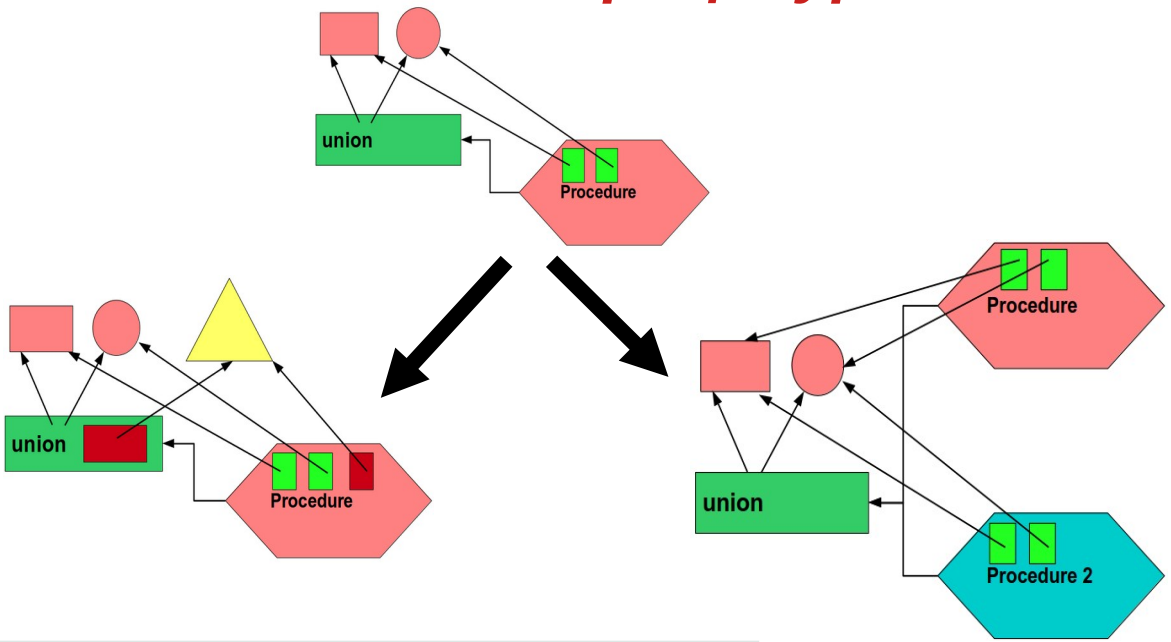


б) Эволюционное расширение



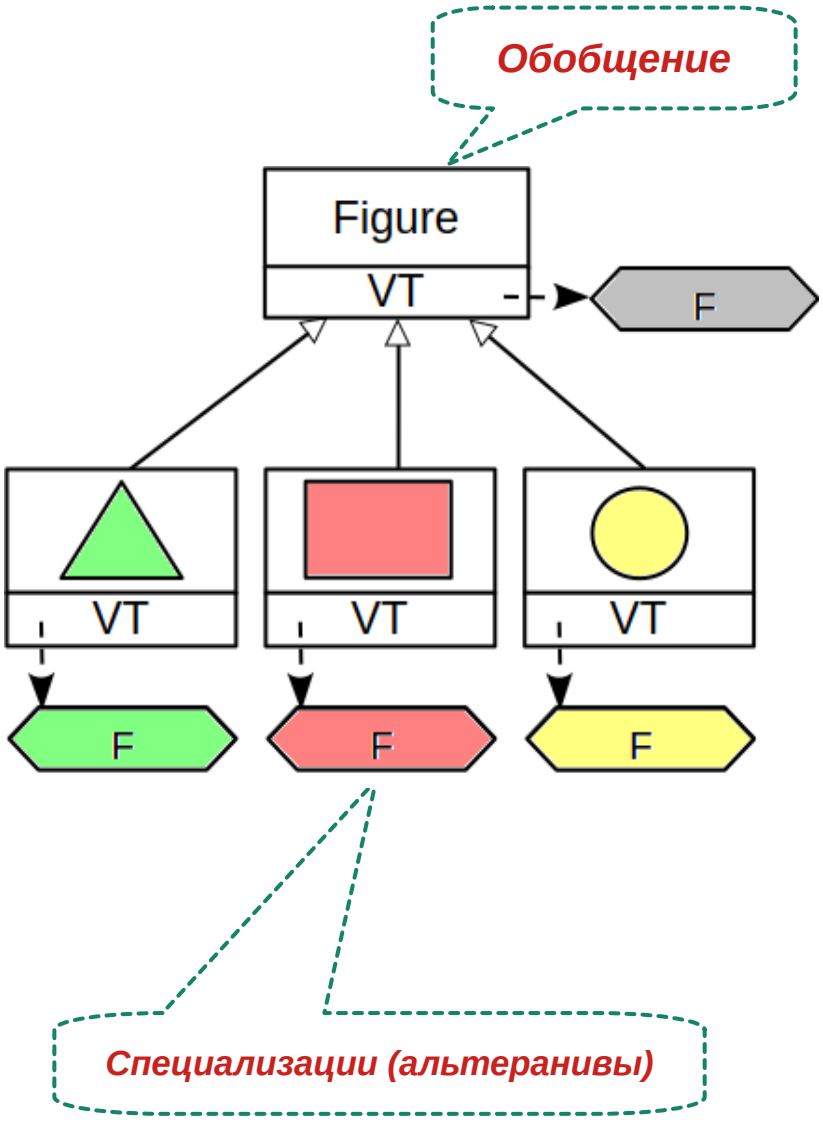


Динамическая проверка типов в процедурных и ОО программах



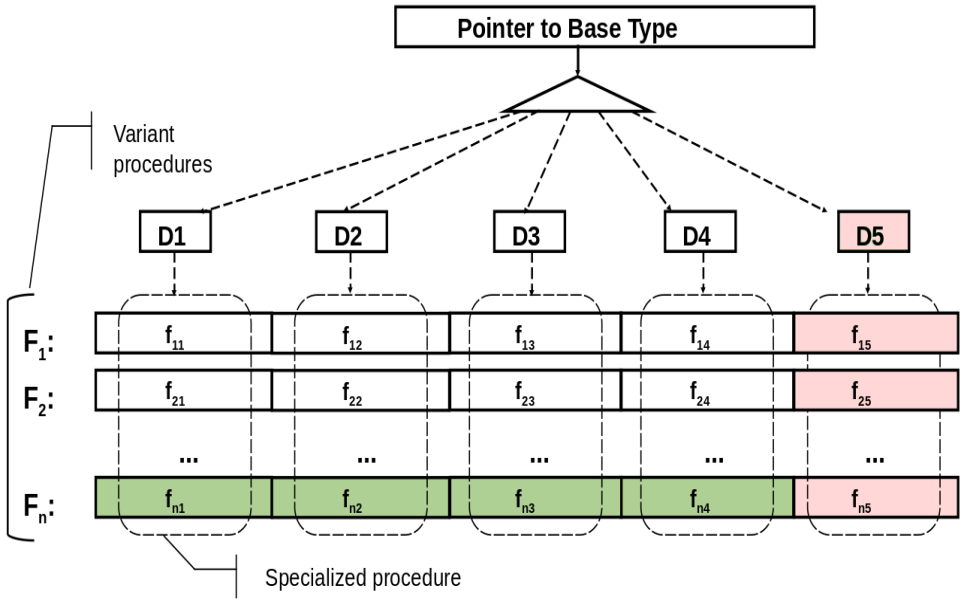

```
struct Rectangle { int x; int y; };
struct Circle { int r; };
struct Triangle {
     int a; int b; int c;
};
void Procedue(Rectangle& r);
void Procedue(Circle& r);
void Procedue(Triangle& r);
enum key {
    rectangle, circle, triangle
};
struct Figure {
     key k;
     union {
          Rectangle r;
          Circle c;
          Triangle t;
     };
};
```

```
void Procedure(Figure& f)
{
     switch(key) {
     case rectangle: P
          Procedure(f.r);
     break;
     case circle:
          Procedure(f.c);
     break;
     case triangle:
          Procedure(f.t);
     break;
     }
}
```



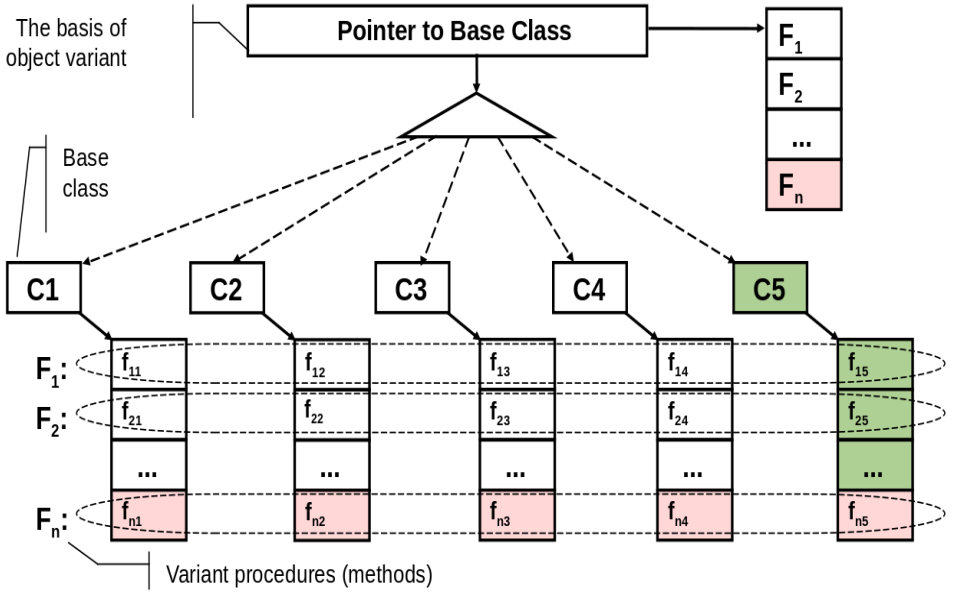
Процедурный стиль

F(X)
F(X, Y, Z)



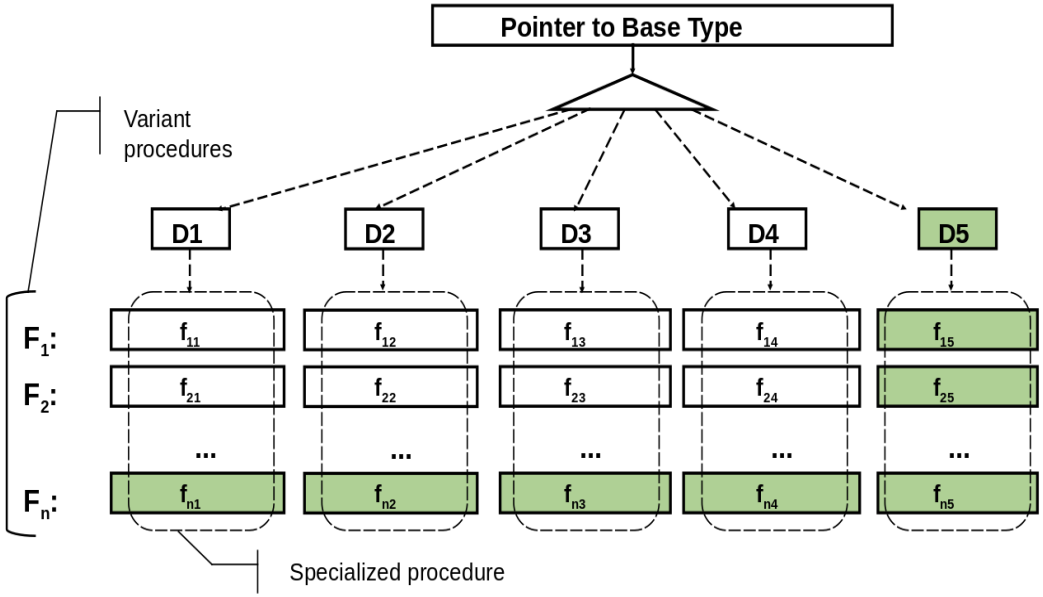
Объектно-ориентированный стиль

X.F()
X.F(Y,Z)



Процедурно-параметрический стиль

F<X>()
F<X,Y,Z>()
F<X,Y>(Z)

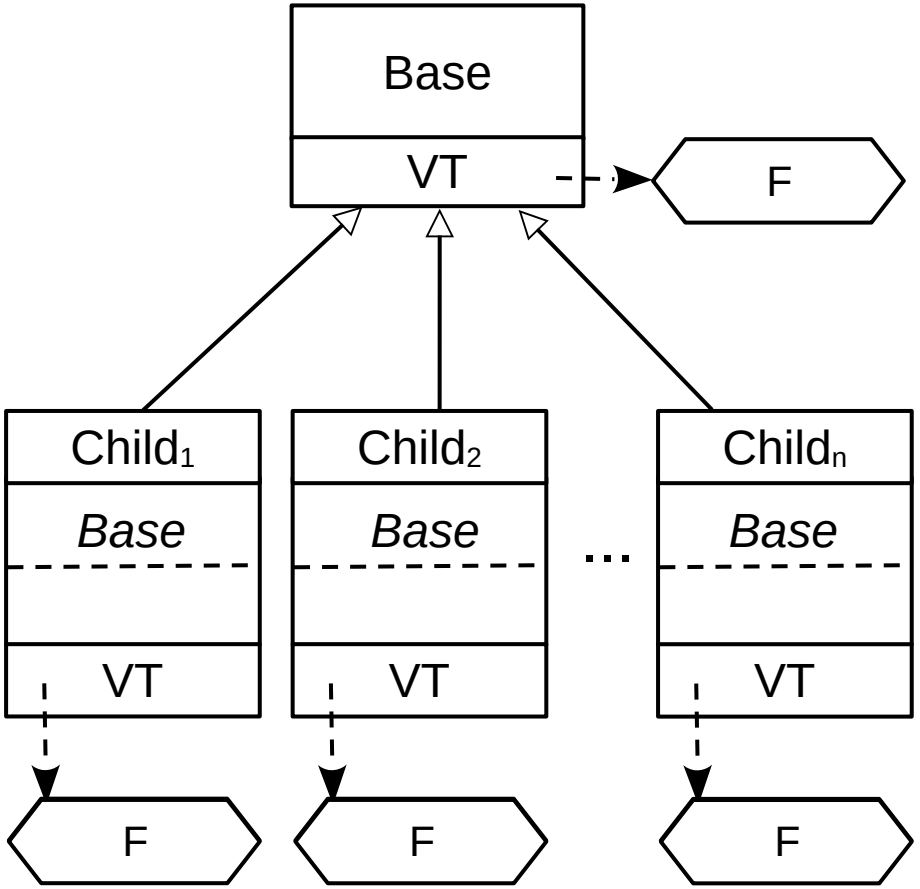


Особенности процедурно-параметрической парадигмы программирования (4П)

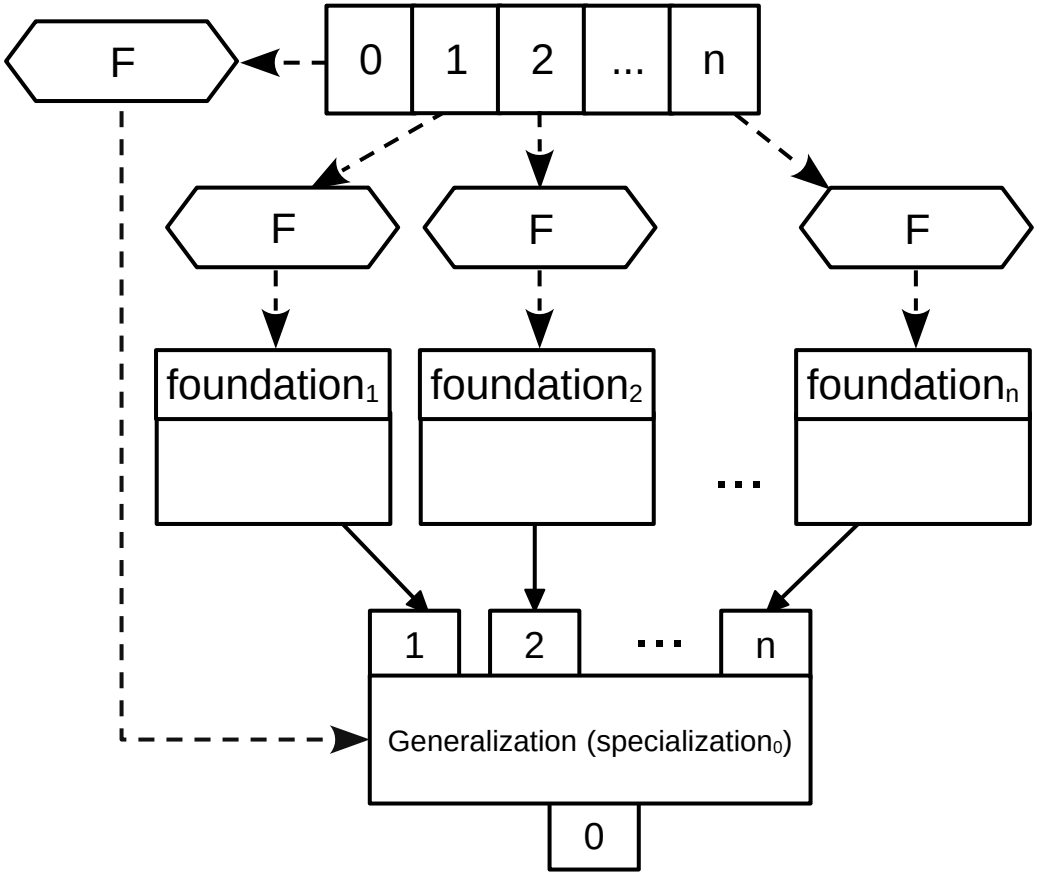
1. Иная разновидность динамического полиморфизма.
2. Перекрытие многих возможностей ОО парадигмы (монометодов).
3. Непосредственная инструментальная поддержка мультиметодов (множественного полиморфизма).
4. При формировании альтернатив (специализаций) можно использовать именованные типы, указатели на них, неименованные структуры.
5. Относительная независимость формирования данных и функций, позволяющая использовать 4П с другими парадигмами и, как следствие, возможность ее встраивания даже в уже существующие языки программирования (есть ли смысл в дублировании иного полиморфизма?).
6. По сравнению с известными подходами более гибкое эволюционное расширение программ без изменения ранее написанного кода как при нисходящем, так и при восходящем проектировании.
7. Много обобщений от одних и тех же основ специализаций.

Иная разновидность динамического полиморфизма

C++



PPC



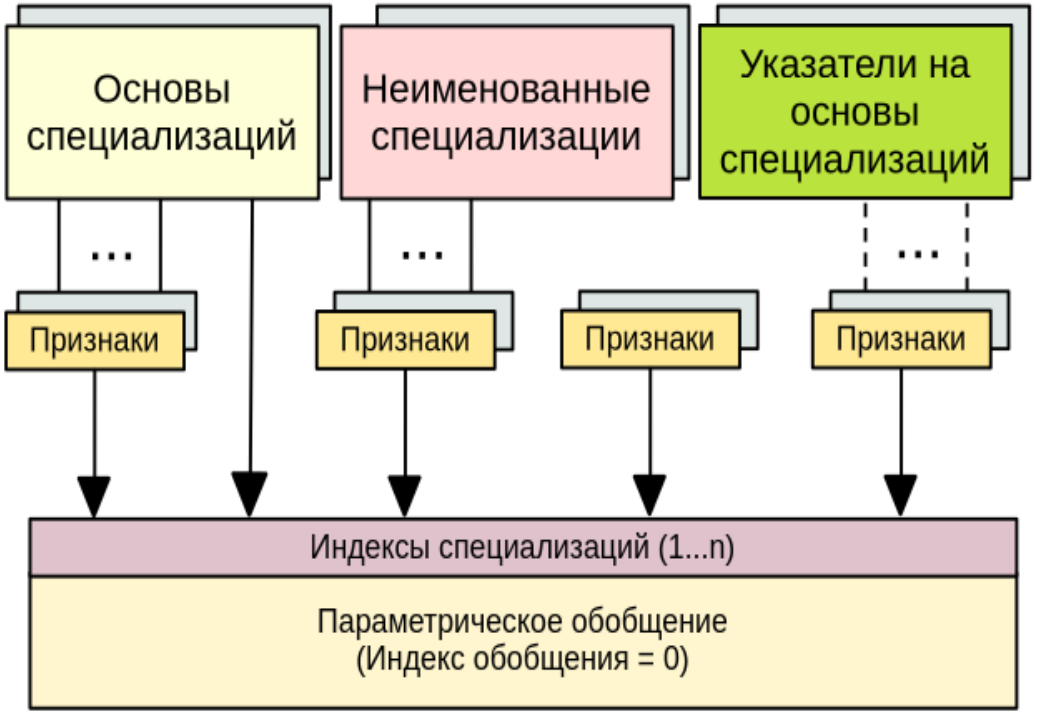
Разновидности динамического полиморфизма в статически типизированных языках:

1) ОО полиморфизм (C++, Java, C#…)

2) Статическая утиная типизация (Go, Rust)

3) Процедурно-параметрический полиморфизм (O2M, PPC) – иное техническое решение для поддержки динамического полиморфизма

Варианты построения специализаций




```
// Основа до обобщения
typedef struct Rectangle {
  int x, y;
} Rectangle;
// Обобщение
typedef struct Figure {} <> Figure;
// Figure.rect – Явное использование признака
Figure + < rect: Rectangle; >;
// Основа после обобщения
typedef struct Triangle {
  int a, b, c;
} Triangle;
// Figure.Triangle – Использовани имени типа
Figure + < Triangle; >;
// Figure.Point – Использование только признака
// (моделирование эволюционно расширяемого enum
Figure + < Point: void; >;
// Рекурсия (Непосредственное применение)
typedef struct Decorator {
    unsigned int color;
} < Figure; > Decorator;
// Figure.decor
Figure + < decor: Decorator; > ;
// Рекурсия
// (Использование указателя как специализации)
Decorator + < fig: Figure*; > ;
```

Компилятор ppclang – расширение clang:
https://gitverse.ru/kpdev/llvm-project

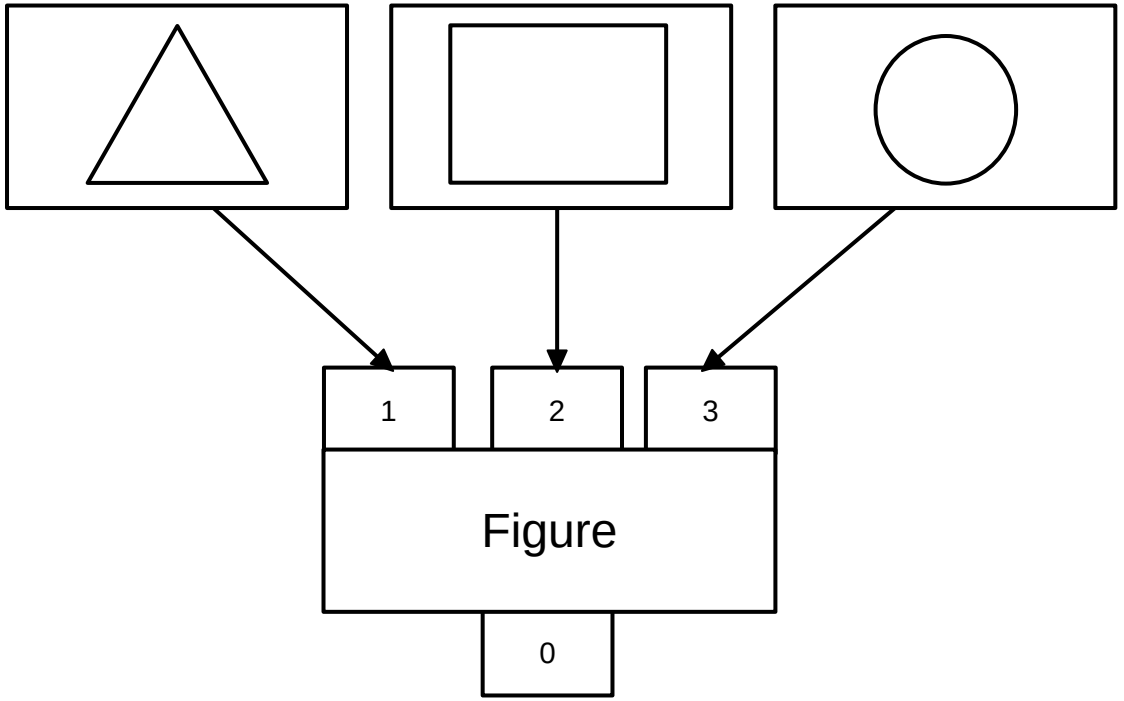
Procedural Parametric C (PPC)

иная техника написания кода

```
typedef struct Figure {} <> Figure;
...
// Prototype of a generalized function
double FigurePerimeter<Figure *f>()

typedef struct Rectangle {
  int x, y;
} Rectangle;

typedef struct Triangle {
  int a, b, c;
} Triangle;

Figure + < rect: Rectangle; >;
Figure + < trian: Triangle; >;

...
```

```
// Обобщенная функция вычисления периметра
double FigurePerimeter<Figure *f>() = 0;

// Обработчики специализаций (альтернатив)

double RectanglePerimeter(Rectangle *r);
// Вычисление периметра для Figure.rect
double FigurePerimeter<Figure.rect *f>() {
  return RectanglePerimeter(&(f->@));
  // или: return (double)(2*(f->@x + f->@y));
}

double TrianglePerimeter(Triangle *t);
// Вычисление периметра для Figure.trian
double FigurePerimeter<Figure.trian *f>() {
  return TrianglePerimeter(&(f->@));
  // или: return (double)(f->@a + f->@b + f->@c);
}

// Обработчики для отдельных основ специализаций
// в других единицах компиляции

double RectanglePerimeter(Rectangle *r) {
  return (double)(2*(r->x + r->y));
}

double TrianglePerimeter(Triangle *t) {
  return (double)(t->a + t->b + t->c);
}
```

Эволюционная поддержка множественного полиморфизма



	0	1	2	3
0	F00	F01	F02	F03
1	F10	F11	F12	F13
2	F20	F21	F22	F23
3	F30	F31	F32	F30

Формируются многомерные таблицы для обработчиков специализаций

Добавление мультиметода при процедурном подходе

явная централизованная проверка типа альтернативы (через ее ключ)

```
//-----------------------------------------------------------------------------
void Multimethod(Figure* f1, Figure* f2, FILE* ofst) {
  switch(f1->k) {
    case RECTANGLE:
      switch(f2->k) {
        case RECTANGLE:
          MMRR((Rectangle*)f1, (Rectangle*)f2, ofst);
          break;
        case TRIANGLE:
          MMRT((Rectangle*)f1, (Triangle*)f2, ofst);
          break;
        case CIRCLE:
          MMRC((Rectangle*)f1, (Circle*)f2, ofst);
          break;
        default:
          fprintf(ofst, "1st is RECTANGLE. Incorrect key of figure 2 = %d\n", f2->k);
      }
      break;
    case TRIANGLE:
      switch(f2->k) {
        case RECTANGLE:
          MMTR((Triangle*)f1, (Rectangle*)f2, ofst);
          break;
        case TRIANGLE:
          MMTT((Triangle*)f1, (Triangle*)f2, ofst);
          break;
        case CIRCLE:
          MMTC((Triangle*)f1, (Circle*)f2, ofst);
          break;
        default:
          fprintf(ofst, "1st is TRIANGLE. Incorrect key of figure 2 = %d\n", f2->k);
      }
      break;
    case CIRCLE:
      switch(f2->k) {
        case RECTANGLE:
          MMCR((Circle*)f1, (Rectangle*)f2, ofst);
          break;
        case TRIANGLE:
          MMCT((Circle*)f1, (Triangle*)f2, ofst);
          break;
        case CIRCLE:
          MMCC((Circle*)f1, (Circle*)f2, ofst);
          break;
        default:
          fprintf(ofst, "1st is CIRCLE. Incorrect key of figure 2 = %d\n", f2->k);
      }
      break;
    default:
      fprintf(ofst, "Incorrect key of figure 1 = %d\n", f1->k);
  }
}
```

Расширение мультиметода при ОО подходе

диспетчеризация

```
class Figure {
public:
  // идентификация, порождение и ввод фигуры из потока
  static  Figure* In(std::ifstream &ifst);
  virtual void InData(std::ifstream &ifst) = 0;  // ввод данных из потока
  virtual void Out(std::ofstream &ofst) = 0;     // вывод данных в стандартный поток
  virtual void Multimethod(Figure& fig2, std::ofstream &ofst) = 0; // мультиметод
  virtual void FirstRectangle(Rectangle& rect, std::ofstream &ofst) = 0;
  virtual void FirstTriangle(Triangle& trian, std::ofstream &ofst) = 0;
  virtual void FirstCircle(Circle& circ, std::ofstream &ofst) = 0;
};
//--------------------------------------------------------------------------
// прямоугольник
class Rectangle: public Figure {
  int x, y; // ширина, высота
public:
  // переопределяем интерфейс класса
  virtual void InData(std::ifstream &ifst);  // ввод данных из потока
  virtual void Out(std::ofstream &ofst);    // вывод данных в стандартный поток
  virtual void Multimethod(Figure& fig2, std::ofstream &ofst); // мультиметод
  virtual void FirstRectangle(Rectangle& rect, std::ofstream &ofst);
  virtual void FirstTriangle(Triangle& trian, std::ofstream &ofst);
  virtual void FirstCircle(Circle& circ, std::ofstream &ofst);
  Rectangle(): x{0}, y{0} {} // создание без инициализации.
};
//--------------------------------------------------------------------------
// треугольник
class Triangle: public Figure {
  int a, b, c; // стороны
public:
  // переопределяем интерфейс класса
  void InData(std::ifstream &ifst);  // ввод данных из потока
  void Out(std::ofstream &ofst);     // вывод данных в стандартный поток
  virtual void Multimethod(Figure& fig2, std::ofstream &ofst); // мультиметод
  virtual void FirstRectangle(Rectangle& rect, std::ofstream &ofst);
  virtual void FirstTriangle(Triangle& trian, std::ofstream &ofst);
  virtual void FirstCircle(Circle& circ, std::ofstream &ofst);
  Triangle(): a{0}, b{0}, c{0} {} // создание без инициализации.
};
```

Расширение мультиметода при ПП подходе

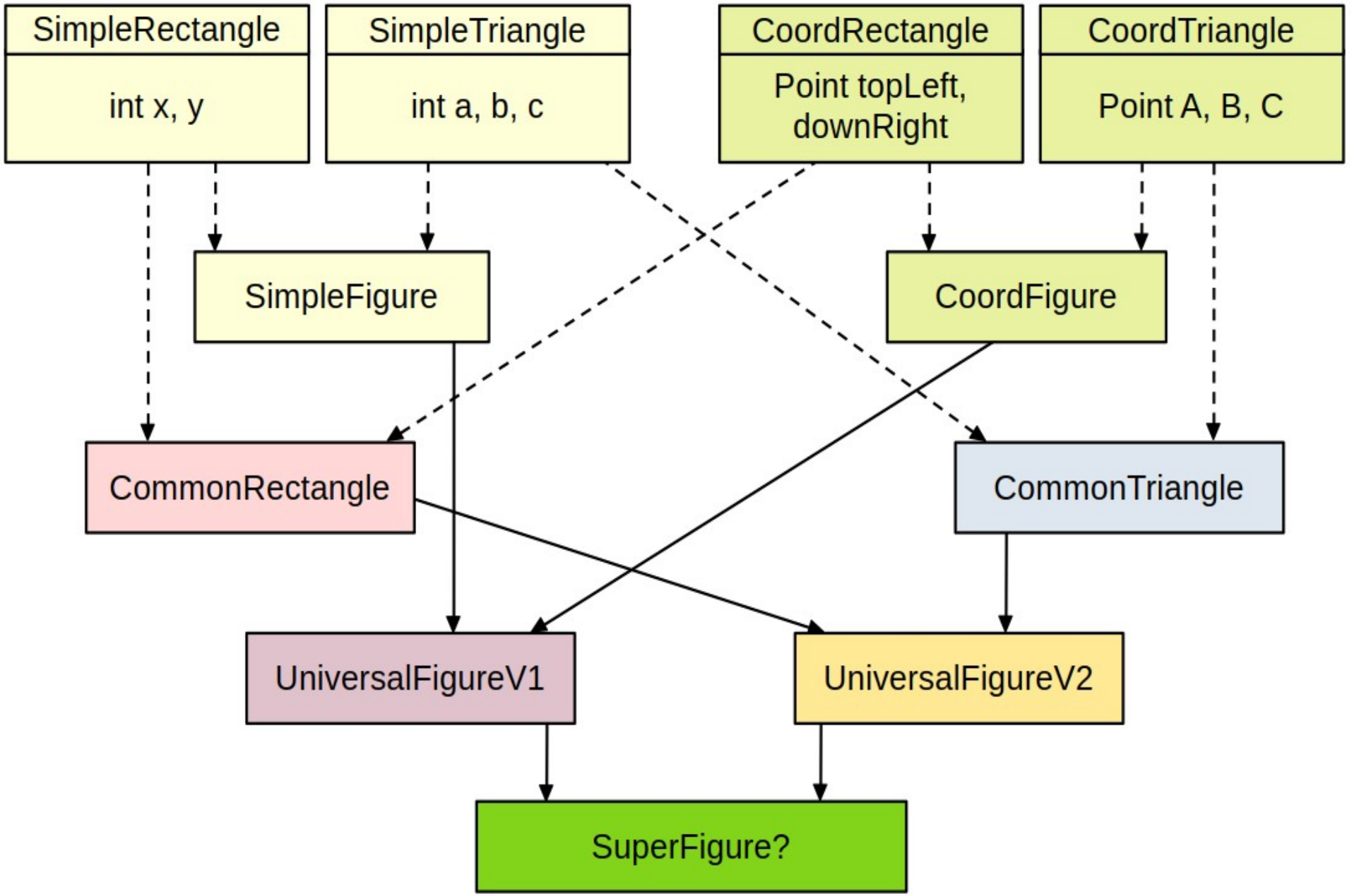
раздельное определение обработчиков специализаций

```
//-----------------------------------------------------------------------
// Обобщающая функция, задающая абстрактный мультиметод
void Multimethod<Figure* f1, Figure* f2>(FILE* ofst) {} //= 0;


//-----------------------------------------------------------------------
// Обработчик специализации для двух прямоугольников
void Multimethod<Figure.Rectangle* r1, Figure.Rectangle* r2>(FILE* ofst) {
  fprintf(ofst, "Rectangle - Rectangle Combination\n");
}
//-----------------------------------------------------------------------
// Обработчик специализации для прямоугольника и треугольника
void Multimethod<Figure.Rectangle* r1, Figure.Triangle* t2>(FILE* ofst) {
  fprintf(ofst, "Rectangle - Triangle Combination\n");
}
//-----------------------------------------------------------------------
// Обработчик специализации для треугольника и прямоугольника
void Multimethod<Figure.Triangle* t1, Figure.Rectangle* r2>(FILE* ofst) {
  fprintf(ofst, "Triangle - Rectangle Combination\n");
}
//-----------------------------------------------------------------------
// Обработчик специализации для двух треугольников
void Multimethod<Figure.Triangle* t1, Figure.Triangle* t2>(FILE* ofst) {
  fprintf(ofst, "Triangle - Triangle Combination\n");
}
```

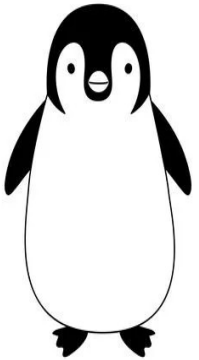
Расширение мультиметода при ПП подходе

```
//-------------------------------------------------------------------------
// Прототип обобщающей функции, задающей абстрактный мультиметод
void Multimethod<Figure* f1, Figure* f2>(FILE* ofst);

//-------------------------------------------------------------------------
// Обработчик специализации для прямоугольника и круга
void Multimethod<Figure.Rectangle* r1, Figure.Circle* c2>(FILE* ofst) {
  fprintf(ofst, "Rectangle - Circle Combination\n");
}
//-------------------------------------------------------------------------
// Обработчик специализации для треугольника и круга
void Multimethod<Figure.Triangle* r1, Figure.Circle* c2>(FILE* ofst) {
  fprintf(ofst, "Triangle - Circle Combination\n");
}
//-------------------------------------------------------------------------
// Обработчик специализации для круга и прямоугольника
void Multimethod<Figure.Circle* c1, Figure.Rectangle* r2>(FILE* ofst) {
  fprintf(ofst, "Circle - Rectangle Combination\n");
}
//-------------------------------------------------------------------------
// Обработчик специализации для круга и треугольника
void Multimethod<Figure.Circle* c1, Figure.Triangle* t2>(FILE* ofst) {
  fprintf(ofst, "Circle - Triangle Combination\n");
}
//-------------------------------------------------------------------------
// Обработчик специализации для двух кругов
void Multimethod<Figure.Circle* c1, Figure.Circle* c2>(FILE* ofst) {
  fprintf(ofst, "Circle - Circle Combination\n");
}
```

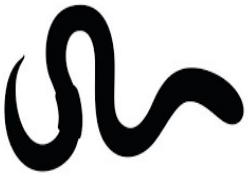
Много обобщений от одних и тех же основ специализаций



Наш зоопарк



- ***Слон***
  - Ходит: "Топ-топ"
  - Издает звуки: "Ду-ду"
- ***Собака***
  - Ходит: "Чап-чап"
  - Издает: звуки "Гав-гав"
- ***Пингвин***
  - Ходит: непонятно как (по умолчанию так, как ходят те, о ком нет знаний)
  - Издает звуки: "Линукс рулит"
  - Плавает: "Буль буль" (каждый, кто плавает, делает это уникально)
- ***Червяк***
  - Ходит: "Ползает"



https://rutube.ru/video/4b80586896390df235cb6a94316bc956/
https://www.youtube.com/watch?v=Nz_jCQ8rmFI

https://github.com/kreofil/evo-situations/tree/main/other/animals

Варианты развития программы...

1. Мы забыли, что собака может плавать

Предлагается сформировать для собаки функционал, обеспечивающий поддержку плавания по собачьи для каждого из рассматриваемых вариантов кода.

2. Мы вдруг вспомнили, что пингвин может еще и нырять

Возникают вопросы:

- Куда определить ныряние? Стоит ли включать его в другой интерфейс или создать новый (не все ныряют)?
- Как оно будет интегрироваться с другой функциональностью?

3. Расширение общего свойства еще одной альтернативной функциональностью

Можно, например, рассмотреть вариант, когда каждое из животных обладает таким свойством как поедание определенной пищи. Оно есть у всех и определяется уникально для каждого (не делать обработчик по умолчанию).

4. Добавление такой общей функциональности, как принадлежность к классу животных

Отношение к млекопитающим, птицам и прочим - это общее свойство. Но оно повторяется. Притом для разных животных. Разделение на отдельные категории вряд ли рационально. Поэтому интересна реализация как подмножество категорий животного.

5. Мультиметод или множественный полиморфизм

Для каждого варианта отношений придумать свой функционал. Это просто, достаточно выводить какую-то нестандартную фразу для каждой из комбинации животных. Например для двух аргументов:

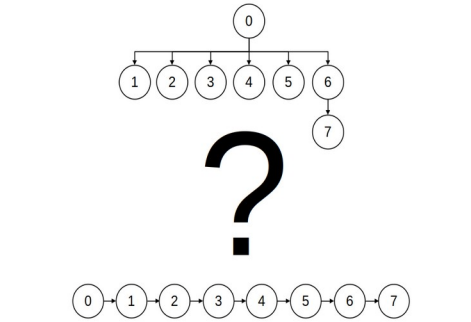
- Слон - собака. А он себе идет и лаю твоего не замечает.
- Собака - слон. Ай Моська, знать она сильна, что лает на слона.

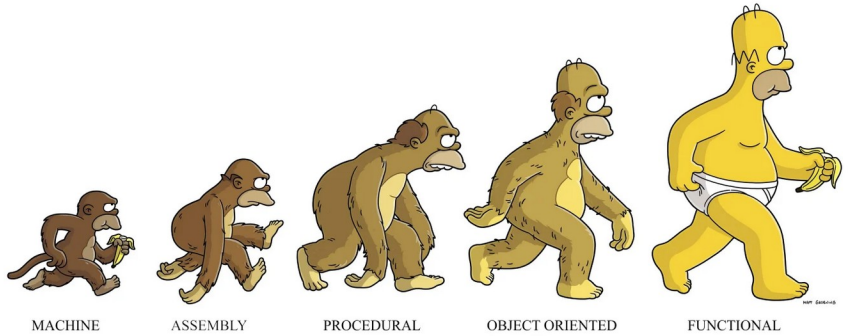
6. Добавление нового животного

Можно добавить, например, комара, рыбу, орла и т.д. С появлением комара или орла появится необходимость ввода функционала полета. Ну а далее могут быть муха и прочие.

Гибкое эволюционное расширение программ

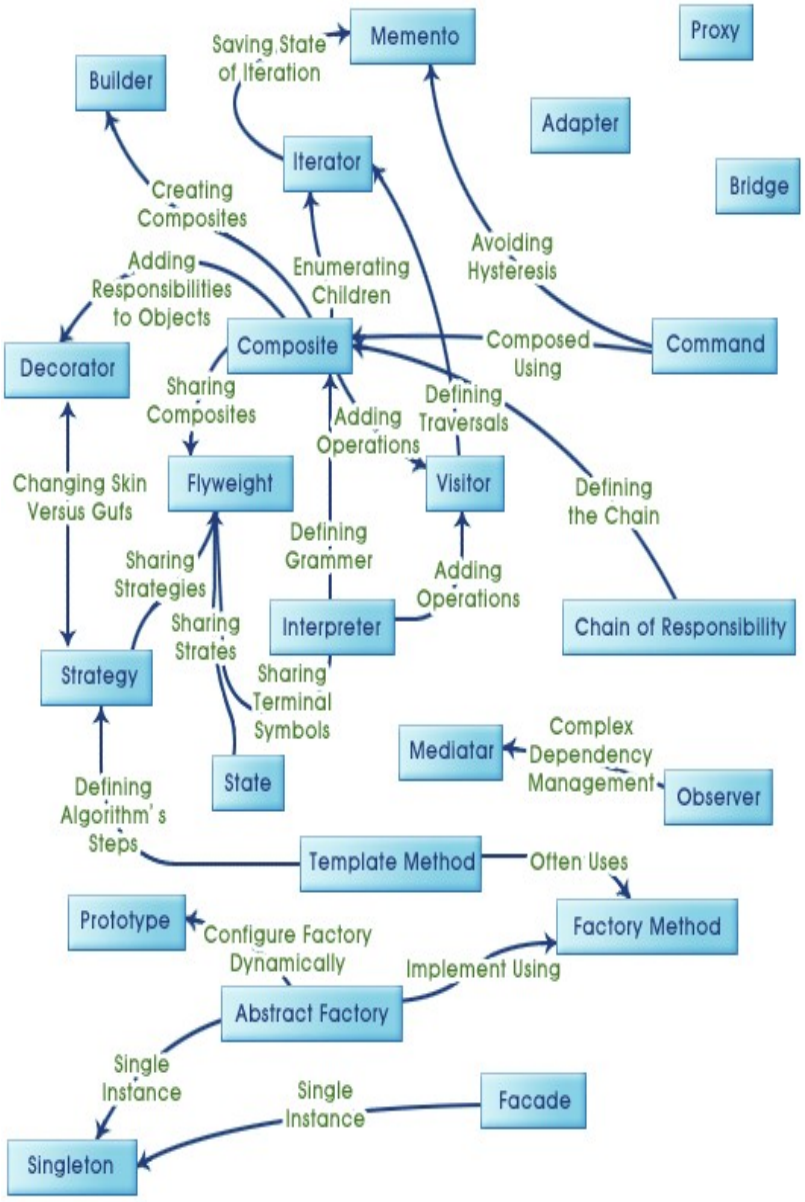
Ситуация	Подходы			
	Процедурный	ОО	Go-поход	ПП
1 Добавление специализации и ее обработчиков				
2 Добавление процедур с дополнительной функциональностью				
3 Добавление новых полей в существующий тип				
4 Добавление новых процедур для обработки конкретных специализаций внутри существующих обобщений				
5 Создание обобщения на основе существующих специализаций				
6 Добавление в программу мультиметода				
7 Изменение мультиметода при добавлении специализации				

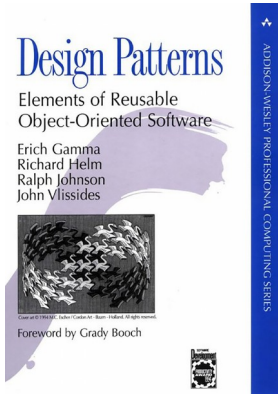




Ситуация	Подходы			
	Процедурный	ОО	Go-поход	ПП
1 Добавление специализации и ее обработчиков	*нет*	*есть*	*есть*	*есть*
2 Добавление процедур с дополнительной функциональностью	*есть*	*нет*	*есть*	*есть*
3 Добавление новых полей в существующий тип	*косвенное, для расширяемых типов*	*косвенное, при наличии динамической проверки типов во время выполнения*	*нет*	*косвенное, при использовании обобщенной записи*
4 Добавление новых процедур для обработки конкретных специализаций внутри существующих обобщений	*есть*	*нет*	*есть*	*есть процедурный и параметрический*
5 Создание обобщения на основе существующих специализаций	*есть*	*косвенное*	*есть*	*есть*
6 Добавление в программу мультиметода	*есть*	*нет*	*есть*	*есть*
7 Изменение мультиметода при добавлении специализации	*нет*	*нет*	*нет*	*есть*

Паттерны проектирования





Реализованы практически все классические паттерны

Дополнительные возможности по сравнению с ОО паттернами проектирования:

- Эволюционное расширение функциональности с поддержкой полиморфизма для уже сформированных структур за счет обобщающих функций.
- Другие подходы к реализации паттернов, обеспечивающие разнообразные специализированные варианты (мультиметод, монолитные декораторы).
- Ряд паттернов уже оказались реализованными при реализации эволюционно расширяемых программ (декоратор, посетитель)
- Зачастую формируются более простые технические решения

https://github.com/kreofil/evo-situations/tree/main/gof-patterns
https://github.com/kreofil/evo-situations/tree/main/oo-patterns-shvets

http://softcraft.ru/ppp/patterns/intro/

SOLID

Single Resposibility Principle
A class should have only a single responsibility (i.e. only one potential change in the software's specification should be able to affect the specification of the class)

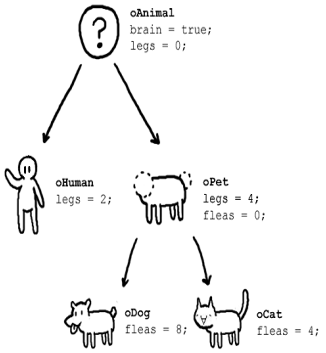
Open / Closed Principle
A software module (it can be a class or method) should be open for extension but closed for modification.

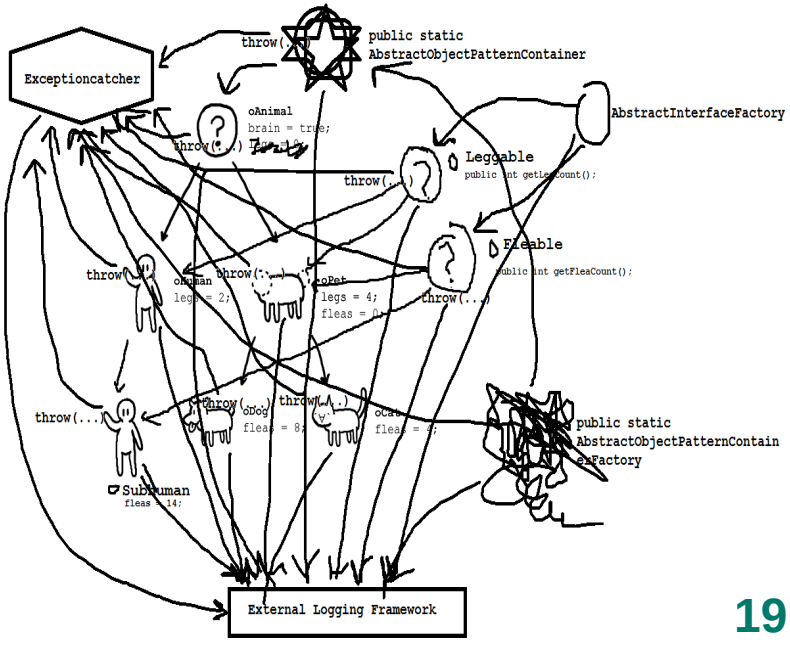
Liskov Substitution Principle
Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.

Interface Segregation Principle
Clients should not be forced to depend upon the interfaces that they do not use.

Dependency Inversion Principle
Program to an interface, not to an implementation.

- **Отсутствуют проблемы в следовании принципам SOLID.**
- **Многие принципы реализуются с использованием паттернов проектирования, которые полностью реализованы с использованием ППП**
- **Разделение данных и полиморфных функций облегчает следование принципам SOLID.**





https://rutube.ru/video/c8fb779074c41e792552a10b871f5a4e/

ППП и процесс разработки

иная техника проектирования?

ООП:

1)Прецеденты – это функции. Зачем их отображать в классы? Искусственная привязка к конструкциям, ограничивающим взаимодействие. Классы не всегда эффективно отображают сложные отношения.

2)Прямое отображение не приветствуется, так как не позволяет достичь требуемых критериев качества. Приходится трансформировать в абстракции, не связанные с предметной областью. Например, паттерны...

ППП:

1)Функциональность предметной области представлена напрямую.

2)Многие отображения в код реализуются напрямую или более гибко.

3)Что мешает использовать ПП язык вместо ОО языка в ОО проектах? (только отсутствие нормального языка?)

4)Возможное дополнительное влияние на процесс проектирования?

Основные свойства и возможности 4П

1. Иная разновидность полиморфизма с непосредственной инструментальной поддержкой мультиметодов.
2. Гибкое эволюционное расширение программ без изменения ранее написанного кода как при нисходящем, так и при восходящем проектировании.
3. Более мелкие фракции данных и функций, используемые в процессе инкрементального наращивания кода.
4. Относительная независимость процедурно-параметрического механизма, позволяющая использовать ППП с другими парадигмами и, как следствие, возможность ее встраивания даже в уже существующие языки программирования (есть ли смысл в дублировании иного полиморфизма?).
5. Возможность независимого использования основ специализаций <вместо шаблонов?>.
6. Обобщение можно формировать как до создания основ специализаций (как и в ООП), так и после, обобощая уже существующие основы специализаций.
7. Много обобщений от одних и тех же основ специализаций.
8. Повышение надежности и качества кода для ненадежных языков за счет использования ППП в качестве обертки ненадежных конструкций (в качестве примера можно привести язык С).
9. Прямая поддержка мультиметодов как свойство ПП полиморфизма.
10. Дополнительные возможности при замене ОО подхода (альтернативная реализация паттернов ОО проектирования).
11. В качестве специализаций можно использовать различные программные объекты (именованные типы, указатели на них, неименованные структуры).
12. Параметризация не только типами, но и значениями за счет специальных функций.
13. Если в вызове обработчика специализаций подставлены конкретные специализации, то можно убрать из параметричесой таблицы соответствующие измерения. Вплоть до непосредственной подстановки нужного обработчика.
14. Возможность гибкой эквивалентной трансформации и оптимизации структуры ПП программ, что позволяет легко заменить параметрические таблицы на другие методы реализации, включая использование конструкций, применяемых в традиционном процедурном (монолитном) программировании.

Дополнительная информация

1. Легалов А.И. Процедурно-параметрическая парадигма программирования. Возможна ли альтернатива объектно-ориентированному стилю? - Красноярск: 2000. Деп. рук. № 622-В00 Деп. в ВИНИТИ 13.03.2000. - 43 с. <http://www.softcraft.ru/ppp/pppfirst/>
2. Легалов А.И. ООП, мультиметоды и пирамидальная эволюция (2002) <http://www.softcraft.ru/coding/evo/>
3. Легалов А.И. Эволюция мультиметодов при процедурном подходе (2002) <http://www.softcraft.ru/coding/evp/>
4. Легалов И.А. Применение обобщенных записей в процедурно-параметрическом языке программирования. / И.А. Легалов // Научный вестник НГТУ. – 2007. – № 3 (28). – С. 25-38. <http://www.softcraft.ru/ppp/genrecords/>
5. Легалов А.И., Бовкун А.Я., Легалов И.А. Расширение модульной структуры программы за счет подключаемых модулей. / Доклады АН ВШ РФ, № 1 (14). – 2010. – С. 114-125. <http://www.softcraft.ru/ppp/inclmodules/>
6. Легалов А.И., Легалов И.А., Солоха А.Ф. Эволюционное расширение программ при различных парадигмах программирования. / Труды XVI Байкальской Всероссийской конференции «Информационные и математические технологии в науке и управлении». Часть III. - Иркутск: ИСЭМ СО РАН, 2011. ISBN 978-5-93908-094-1. - С. 42-49. <http://www.softcraft.ru/ppp/simplesituations/>
7. Легалов А.И., Косов П.В. Эволюционное расширение программ с использованием процедурно-параметрического подхода // Вычислительные технологии. 2016. Т. 21. № 3. С. 56-69. <http://www.ict.nsc.ru/jct/content/t21n3/Legalov_n.pdf>
8. Легалов А.И., Косов П.В. Расширение языка С для поддержки процедурно-параметрического полиморфизма. Моделирование и анализ информационных систем. 2023;30(1):40-62. <https://doi.org/10.18255/1818-1015-2023-1-40-62>
9. Легалов А.И., Косов П.В. Процедурно-параметрическое расширение языка программирования С. Синтаксис и семантика. <http://www.softcraft.ru/ppp/ppc/>
10. Размещение проекта с процедурно–параметрической версией языка программирования С на Gitverse. <https://gitverse.ru/kpdev/llvm-project>
11. Примеры на Гитхаб, написанные с использованием процедурно–параметрической версии языка С. <https://github.com/kreofil/evo-situations>
12. Прототип семантической модели (промежуточного представления), разрабатываемый с использованием ППП. <https://github.com/kreofil/ppp-semantic-model>

SoftCraft

разноликое программирование

- ОТПРАВНАЯ ТОЧКА
- ПРОЕКТИРОВАНИЕ
- ПАРАДИГМЫ
- СИСТЕМЫ ПРОГРАММИРОВАНИЯ
- ТЕХНИКА КОДИРОВАНИЯ
- ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ
- ТЕОРИЯ
- УЧЕБНЫЙ ПРОЦЕСС
- РАЗНОЕ
- ОБ АВТОРЕ

Последние изменения

2025

12.07.2025 А.И. Легалов, П.В. Косов. Процедурно-параметрическое расширение языка программирования С. Синтаксис и семантика

18.06.2025 **Начал формирование материалов по** процедурно-параметрической парадигме и объектно-ориентированным паттернам проектирования. В настоящий момент сформированы следующие разделы:

- Общие рассуждения
- Демонстрационный пример
- Фабричный метод (Factory Method)
- Абстрактная фабрика (Abstract Factory)
- Строитель (Builder)
- Прототип (Prototype)
- Декоратор (Decorator)
- Стратегия (Strategy)
- [illegible]

03.06.2025 А.И. Легалов Об объектно-ориентированной парадигме программирования (путешествие в ООП и обратно)

02.06.2025 Выложено новое видео на rutube.ru: Животный мир и процедурно-параметрическое программирование. На простом примере, описывающем формирование свойств животных, рассматривается использование процедурно-параметрической парадигмы для гибкой и эволюционной разработки приложений.

04.04.2025 **Создан плейлист на rutube.ru по** процедурно-параметрической парадигме программирования
Выложены лекции, записанные на спецкурсе по парадигмам и языкам программирования:

- Композиция программных объектов
- Эволюционное расширение программ и парадигмы программирования
- Принципы SOLID и процедурно-параметрическое программирование
- Процедурно-параметрическое программирование. Текущие результаты и пути дальнейшего развития

20.01.2025 А.И. Легалов, П.В. Косов. *Технические приемы процедурно-параметрического программирования.* Доклад на семинаре STEP-2024 13.01.2025 (в продолжение предыдущего доклада, сделанного 29.05.2024).
Видеозапись
Презентация

2024

31.05.2024 А.И. Легалов, П.В. Косов. *Процедурно-параметрический полиморфизм и его интеграция с языком программирования С.* Доклад на семинаре STEP-2024 29.05.2024
Видеозапись
Презентация

Благодарю за внимание!

http://www.softcraft.ru/ppp/