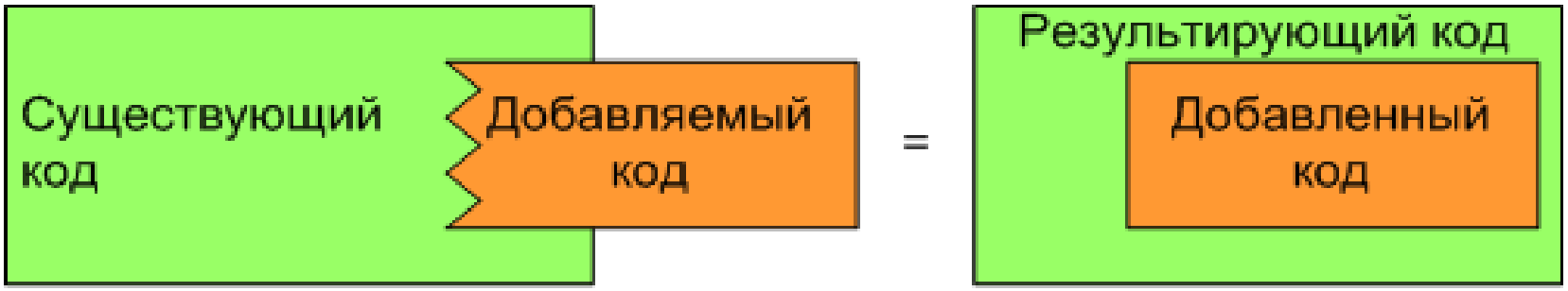
Девятая независимая
научно-практическая конференция
«Разработка ПО 2013»
23 - 25 октября, Москва

Эволюционная разработка программ с применением процедурно-параметрического программирования

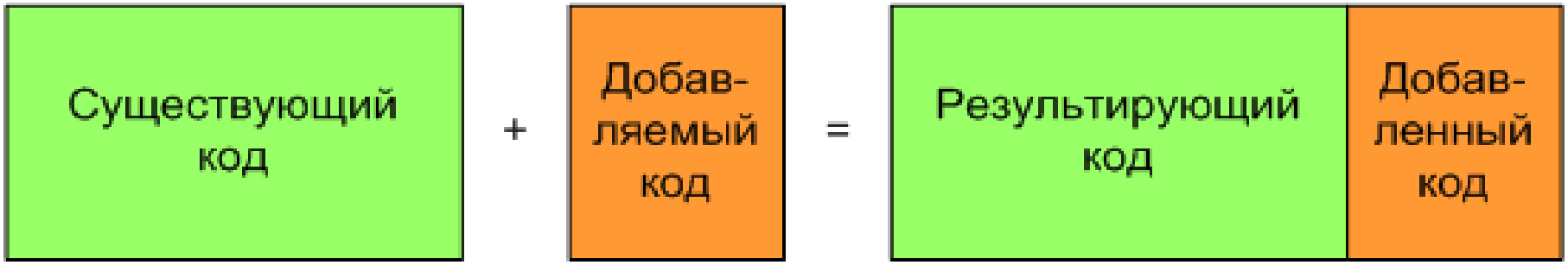
Легалов А.И., *Косов П.В.*

Сибирский федеральный университет

Специфика эволюционного расширения



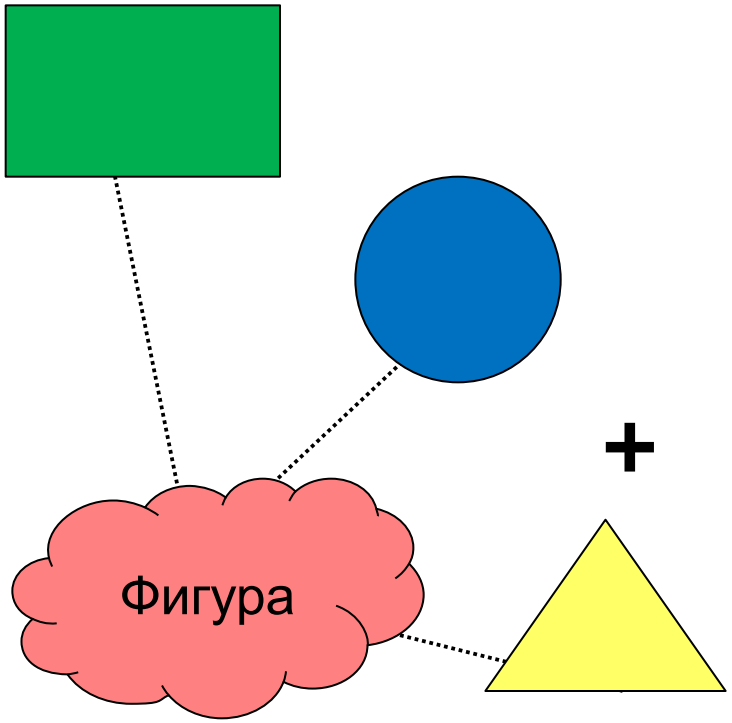
а) Изменение существующего кода



б) Эволюционное расширение

Специфика эволюционного расширения

Специализации



Обобщение

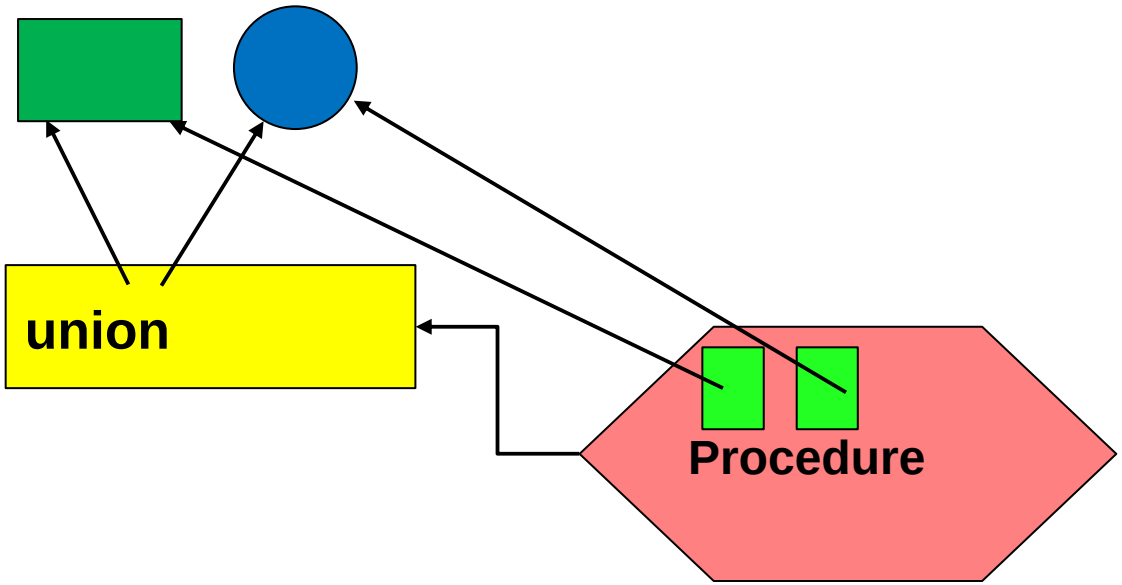
Обработчики специализаций

- Вывод прямоугольника
- Периметр прямоугольника
- Площадь прямоугольника
- Вывод круга
- Периметр круга
- Площадь круга
- Вывод треугольника
- Периметр треугольника
- Площадь треугольника
- Вывод фигуры
- Периметр фигуры
- Площадь фигуры

Обобщающие процедуры

Процедурное программирование

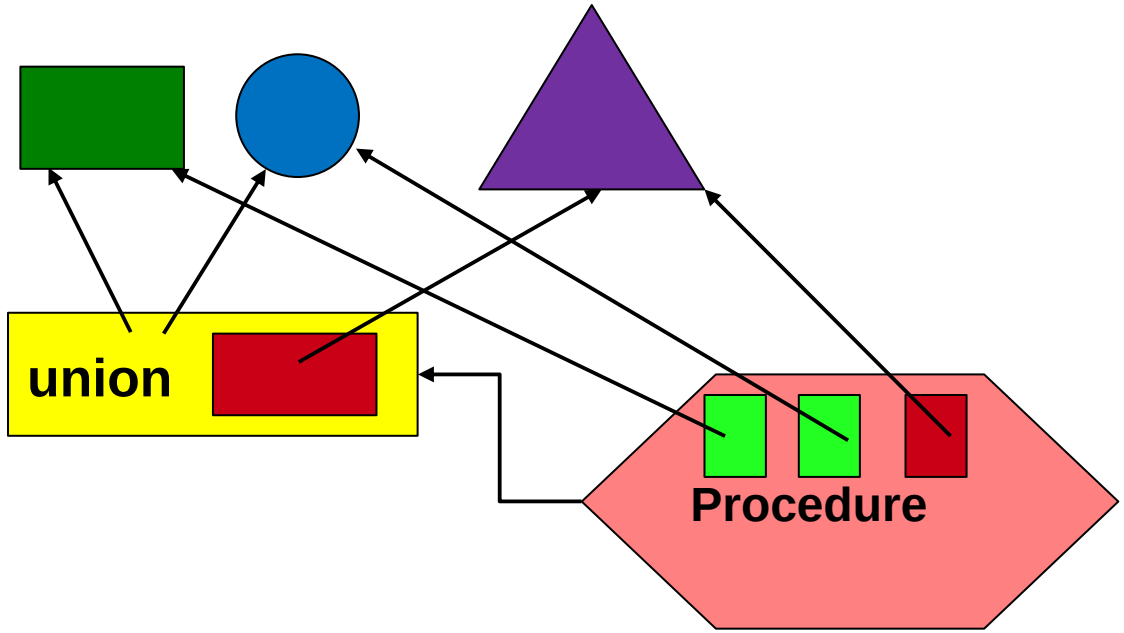
```
struct Rectangle { int x; int y; };
struct Circle { int r; };

enum key { rectangle, circle };

struct Figure {
    key k;
    union {
        Rectangle r;
        Circle c;
    };
};

void Procedue(Rectangle& r);
void Procedue(Circle& r);

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



- Безболезненное добавление новых процедур.
- Специализации можно использовать в разных обобщениях.
- Изменение кода при добавлении новых специализаций (можно избежать).
- Изменение обобщающих процедур при добавлении новых специализаций и их обработчиков.

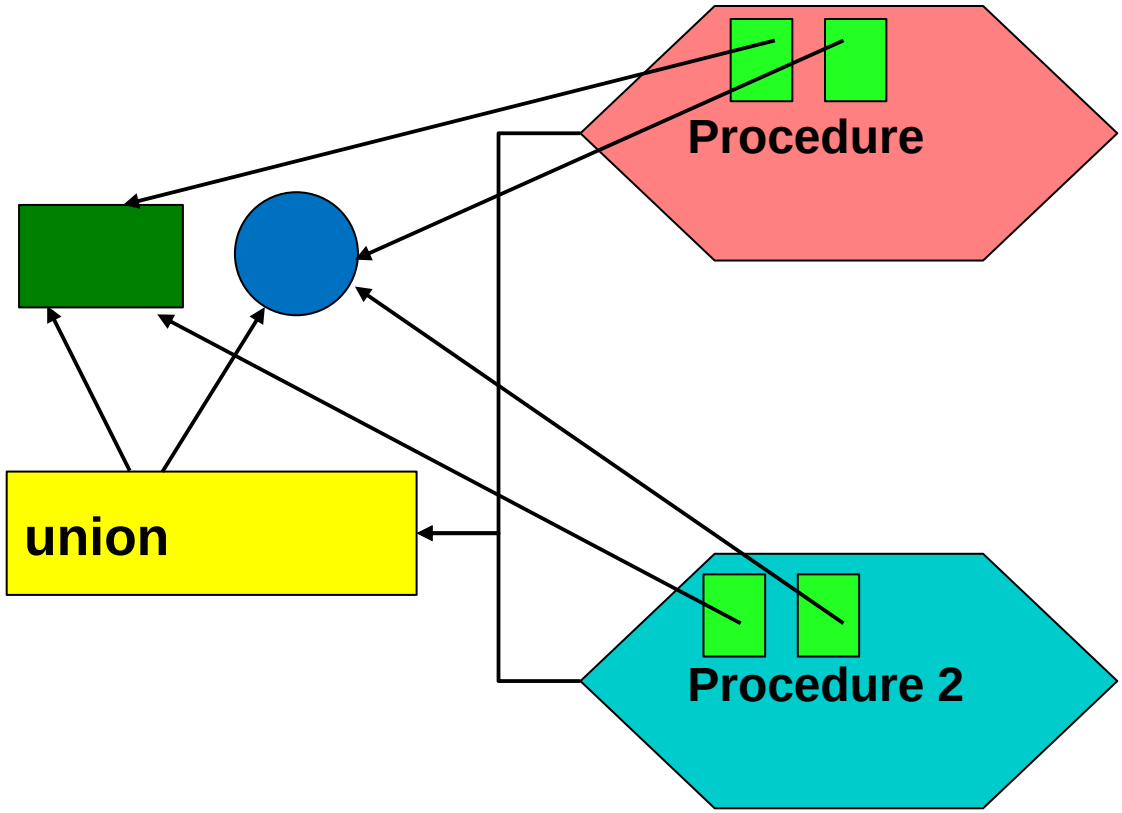
Процедурное программирование. Добавление специализаций

```
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;
    }
}
```



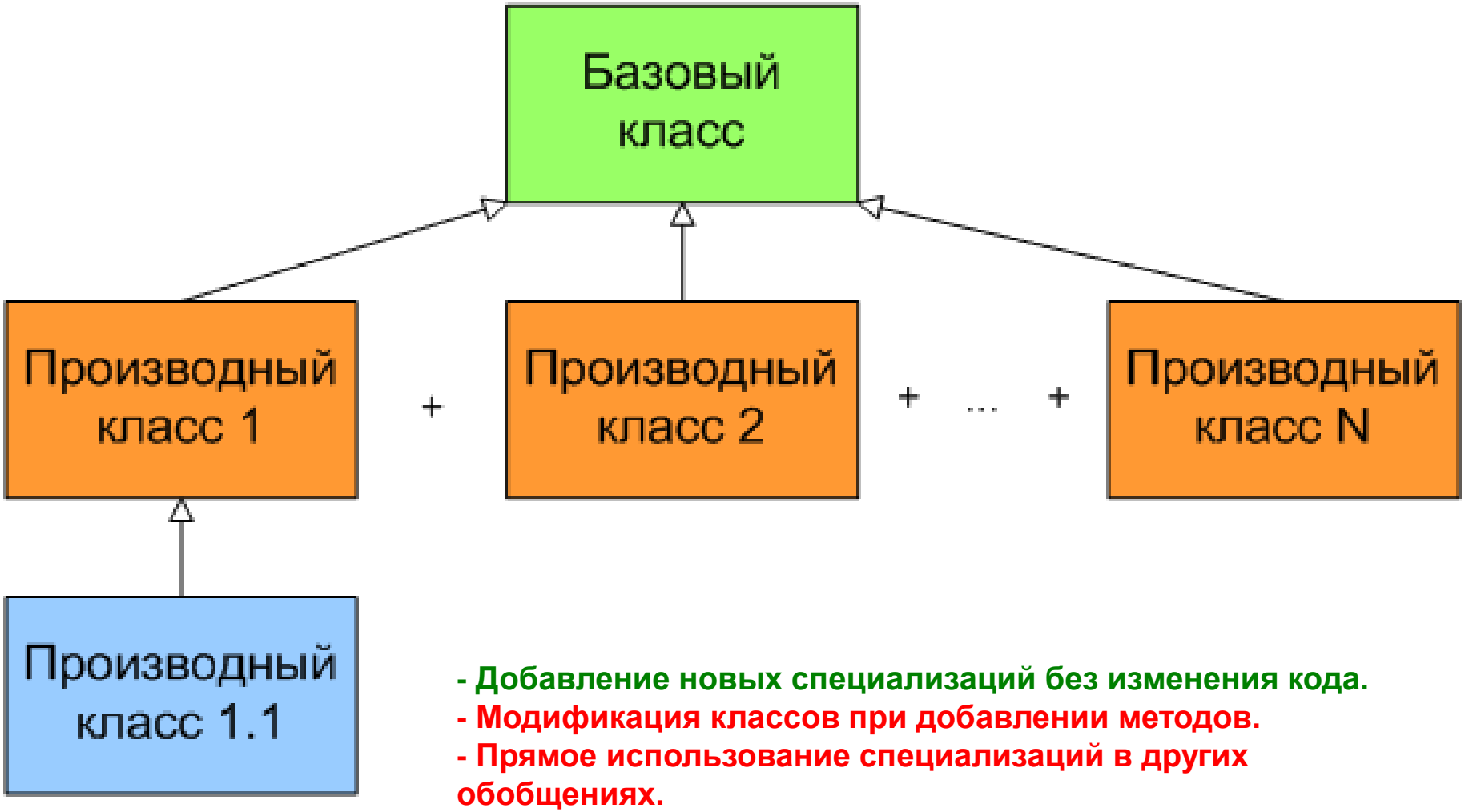
Процедурное программирование. Добавление процедуры

```
struct Rectangle { int x; int y; };
struct Circle { int r; };

enum key { rectangle, circle };

struct Figure {
     key k;
     union {
          Rectangle r;
          Circle c;
     };
};

void Procedue(Rectangle& r);
void Procedue(Circle& r);

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




```
void Procedue2(Rectangle& r);
void Procedue2(Circle& r);

void Procedure2(Figure& f) {
     switch(key) {
     case rectangle: P
          Procedure2(f.r);
     break;
     case circle:
          Procedure2(f.c);
     break;
     }
}
```

Объектно-ориентированное программирование

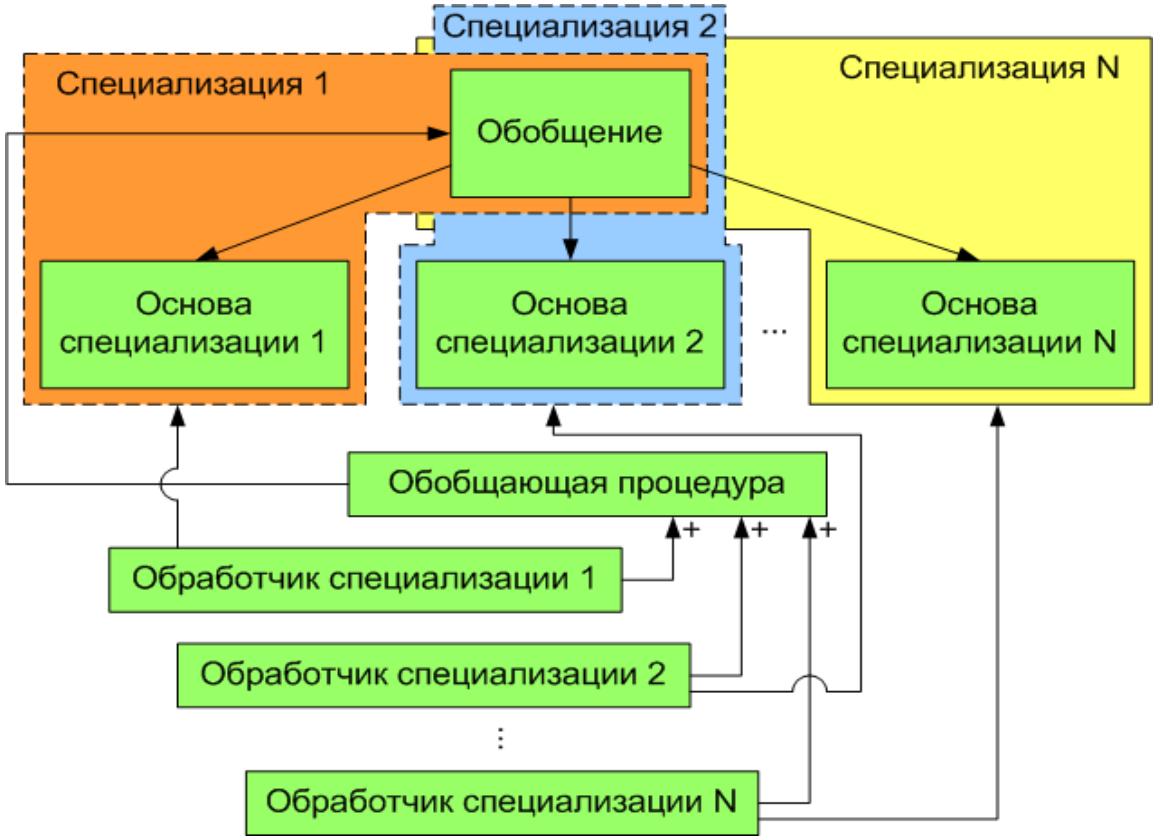


- Добавление новых специализаций без изменения кода.
- Модификация классов при добавлении методов.
- Прямое использование специализаций в других обобщениях.

Безболезненное расширение обобщений и процедур?
→ процедурно-параметрический стиль

Цели:
-Использование одних и тех же специализаций для создания различных обобщений
- Гибкое расширение обобщений за счет новых специализаций
- Гибкое расширение обобщающих процедур
- Безболезненное добавление новых процедур
- Поддержка эволюционного расширения в случае множественного полиморфизма (мультиметоды)

На этапе выполнения программы (динамическое связывание)



Обобщение

ТипОбобщение = CASE [TYPE] OF [[LOCAL]
[СписокСпециализаций]]
[ELSE Специализация] END .
СписокСпециализаций = (
СписокПризнаков «:» (Тип | NIL)) | Тип
{ «|» (СписокПризнаков «:» (Тип | NIL)) | Тип } .
СписокПризнаков = идентификатор
{ «,» идентификатор }.
Специализация = (идентификатор «:» (Тип | NIL)) | Тип .

```
// Прямоугольник со сторонами x, y
Rectangle = RECORD x, y: INTEGER END
// Треугольник со сторонами a, b, c
Triangle = RECORD a, b, c: INTEGER END

Figure = CASE OF
        rect: Rectangle |
        trian: Triangle
END
```

Расширение обобщения

Расширение = Обобщение "+=" СписокСпециализаций.

```
// Круг радиуса r
Circle = RECORD r: INTEGER END
// Расширение обобщения
Figure += circ: Circle
```

Блокировка расширения

```
// Дни недели
Day = CASE LOCAL OF
   Sunday, Monday, Tuesday, Wednesday,
   Thursday, Friday, Saturday: NIL
END;
```

Обобщенная запись

ТипОбобщеннаяЗапись = RECORD
[СписокПолей {";" СписокПолей}]
(Обобщение | [CASE ИмяОбобщения] END).

```
T0 = RECORD x: INTEGER; CASE OF END;
T0 += y: REAL;

VAR v(y): T0
    ...
v.x := 10;
v(y) := 3.14;
```

или

```
v() = 3.14;
```

Создание специализаций (круг, прямоугольник)

```
...
Circle* = RECORD
r* : INTEGER; // радиус круга
END;

// Процедура вывода
PROCEDURE Output*(VAR c: Circle);
BEGIN ... END Output;
...

Rectangle* = RECORD
x*, y* : INTEGER;
END;

// Процедура вывода
PROCEDURE Output*(VAR r: Rectangle);
BEGIN ... END Output;
```

Создание обобщения (круг + прямоугольник)

```
Figure* = CASE TYPE OF
   Rectangle |
  Triangle
END;

//Обобщенная параметрическая процедура вывода фигуры
PROCEDURE Output* {VAR f: Figure} := 0;

// Вывод обобщенного прямоугольника
PROCEDURE Output {VAR r: Figure(Rectangle)};
BEGIN MRect.Output(r); END Output;

// Вывод обобщенного треугольника
PROCEDURE Output {VAR t: Figure(Triangle)};
BEGIN MTrian.Output(t); END Output;
```

Создание записи с уже существующими обобщениями

```
ColoredFigure = RECORD
  color: INTEGER;
  CASE Figure
END;
```

В отличие от концепции базового типа, расширяемого за счет добавления в производных типах, использование параметрического обобщения предполагает сохранение исходного типа, а альтернативные специализации вводятся как его уточнения.

Внешне все специализации имеют единый тип, а их разнообразное толкование используется только внутри него.

Это позволяет убрать глобальную идентификацию типов, применяемую при расширении записей или наследовании, и обеспечивает поддержку концепции строгой типизации.

==> Возможность табличных обращений

Косвенная адресация -- Индексация

Возможности обобщенных записей

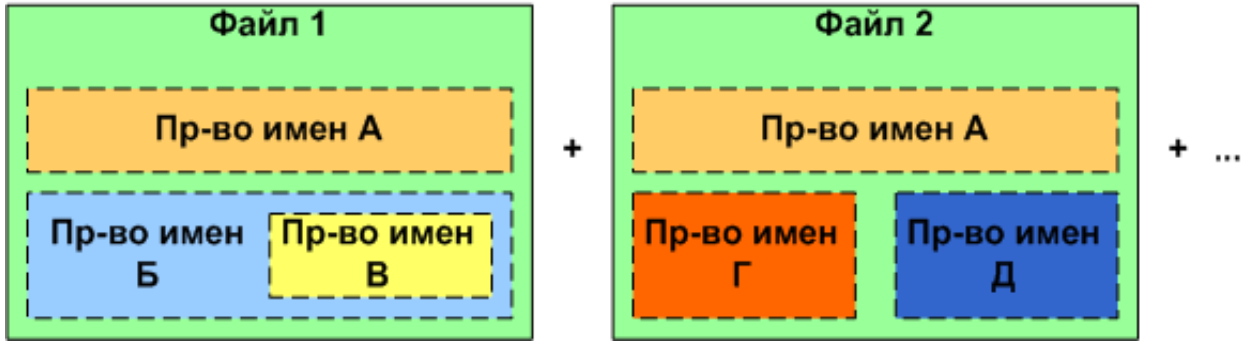
```
// Расширение исходного (базового) типа
T = RECORD x, y: INTEGER; CASE OF END;
T += t0: BOOLEAN;
T += t1: RECORD r: REAL; s: CHAR END;
// Специализированные переменные
v0: T(t0);
v1:T(t1);
// Обращение к основным и альтернативным полям
v0.x, v1.y, v0(), v1().r …
// Рекуррентное определение типа
T += t3: T;
// Рекурентные переменные
T(t3), T(t3)(t3),
T(t3)(t3)(t3)(t3)(t2)(t00), …
```

Статика вместо динамики

```
// Декоратор Точки
Decorator = CASE TYPE OF END;
PointDecorator = RECORD p:Point CASE Decorator END;
// Декоратор Цвета
ColorDecorator = RECORD c:Color CASE Decorator END;
// Декоратор угла
AngleDecorator = RECORD alpha:REAL;
                        CASE Decorator END;
// Общий декоратор
Decorator += PointDecorator | ColorDecorator
           | AngleDecorator | Figure;

// Цветная фигура на плоскости через декораторы
NewFigure = PointDecorator(AngleDecorator
                 (ColorDecorator(
                      ColorDecorator(Figure))));

VAR   circle: NewFigure(Circle);
      rectangle: NewFigure(Rectangle);
```

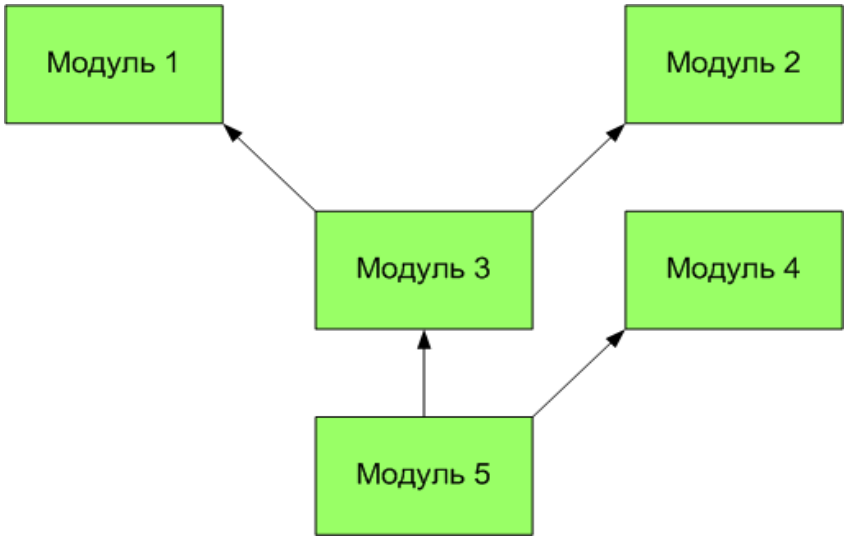
Структура программы

Пространства имен



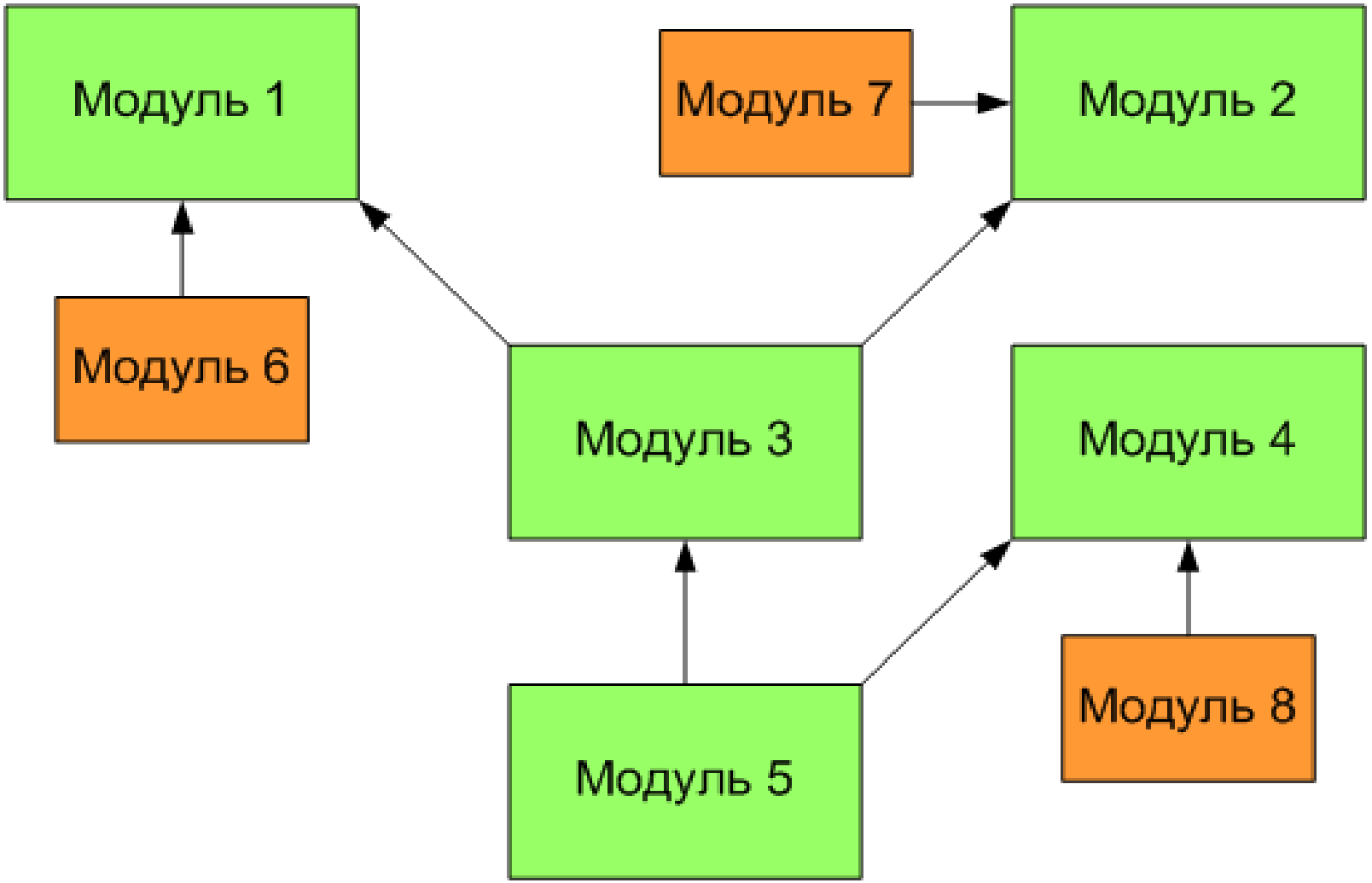
C++, C#

Традиционная модульная структура

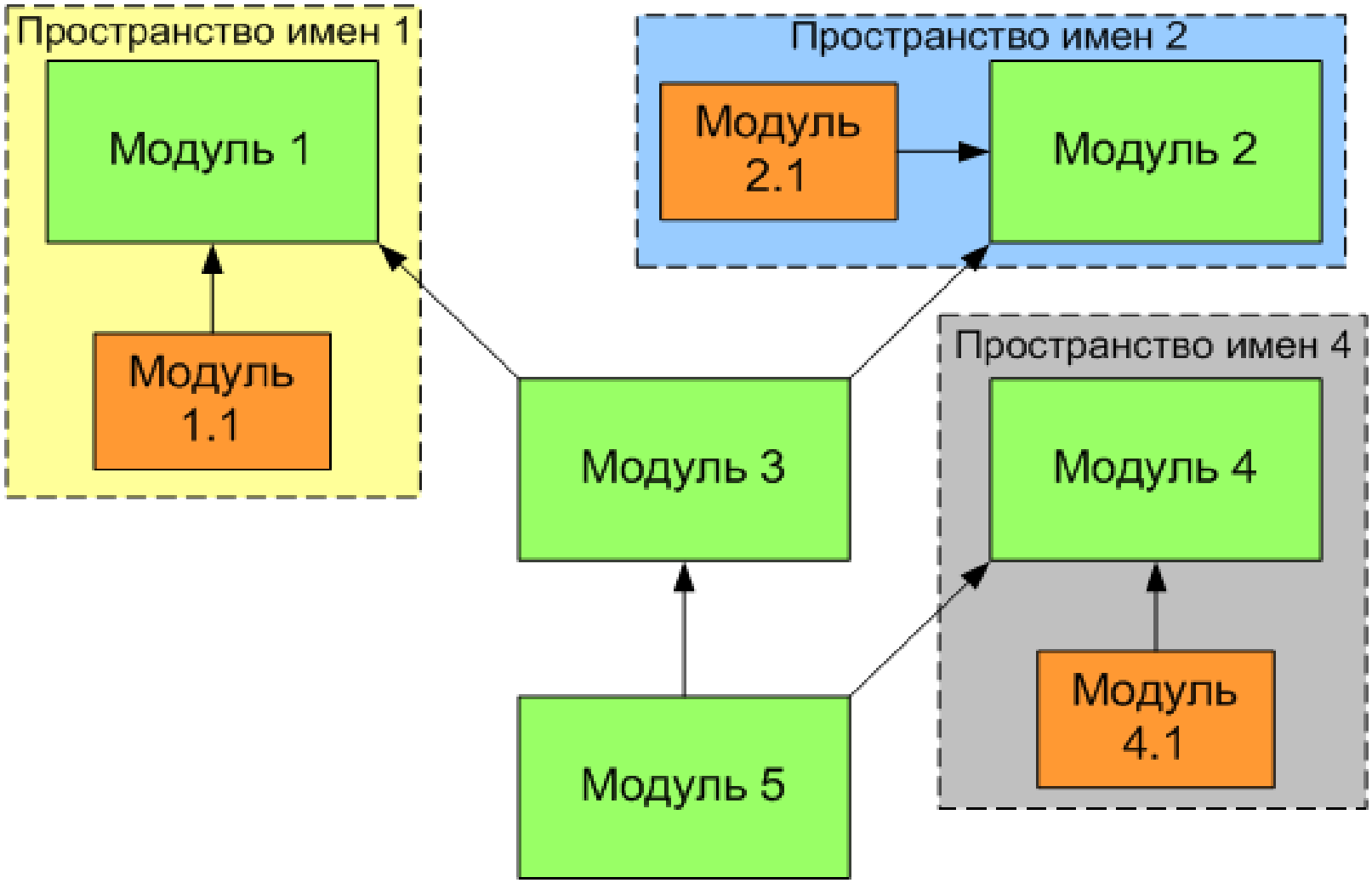


Oberon, Python

Расширение иерархической модульной структуры



Подключаемые модули



Модуль с прямоугольником - специализацией

```
// Модуль, описывающий прямоугольник
MODULE MRect;
IMPORT In, Out;
TYPE
PRectangle* = POINTER TO Rectangle;
Rectangle* = RECORD
x*, y* : INTEGER // стороны прямоугольника
END;
// Процедура вывода
PROCEDURE Output*(VAR r: Rectangle);
BEGIN
Out.String("Rectangle: x = "); Out.Int(r.x, 0);
Out.String(", y = "); Out.Int(r.y, 0);
Out.Ln;
END Output;
// Прочие процедуры
...
END MRect.
```

Модуль с треугольником - специализацией

```
// Модуль, описывающий треугольник
MODULE MTrian;
IMPORT In, Out;
TYPE
PTriangle* = POINTER TO Triangle;
Triangle* = RECORD
// стороны треугольника
a*, b*, c* : INTEGER
END;
// Процедура вывода
PROCEDURE Output*(VAR t: Triangle);
BEGIN
Out.String("Triangle: a = "); Out.Int(t.a, 0);
Out.String(", b = "); Out.Int(t.b, 0);
Out.String(", c = "); Out.Int(t.c, 0);
Out.Ln;
END Output;
// Прочие процедуры
...
END MTrian.
```

Модуль с обобщенной фигурой

```
MODULE MFig;
IMPORT In, Out, MRect, MTrian;
TYPE
//Указатель на обобщенную геометрическую фигуру
PFigure* = POINTER TO Figure;
//Обобщение геометрической фигуры
Figure* = CASE TYPE OF
   MRect.Rectangle |
   MTrian.Triangle
END;
//Обобщенная параметрическая процедура вывода фигуры
PROCEDURE Output* {VAR f: Figure} := 0;
// Обработчики специализаций
// Вывод обобщенного прямоугольника
PROCEDURE Output {VAR r: Figure(MRect.Rectangle)};
BEGIN MRect.Output(r) END Output;
// Вывод обобщенного треугольника
PROCEDURE Output {VAR t: Figure(MTrian.Triangle)};
BEGIN  MTrian.Output(t) END Output;
END MFig.
```

Модуль с процедурой, реализующей мультиметод

```
MODULE MRTMM (MFig*);
IMPORT In, Out, MRect, MTrian;
// Чистая обобщающая процедура, задающая интерфейс
PROCEDURE FirstInSecond* {VAR f1,f2: Figure}:BOOLEAN := 0;
// Обработчики специализаций
// прямоугольник разместится внутри прямоугольника
PROCEDURE FirstInSecond {VAR f1,f2: Figure(MRect.Rectangle)}: BOOLEAN;
BEGIN
    Out.String("Rectangle in Rectangle compare");
    Out.Ln;
    RETURN ((f1.x < f2.x) & (f1.y < f2.y)) OR ((f1.x < f2.y) & (f1.y < f2.x))
END FirstInSecond;
// прямоугольник разместится внутри треугольника
PROCEDURE FirstInSecond {VAR f1(MRect.Rectangle), f2(MTrian.Triangle):
Figure}: BOOLEAN;
BEGIN ... END FirstInSecond;
// треугольник разместится внутри прямоугольника
PROCEDURE FirstInSecond
       {VAR f1(MTrian.Triangle), f2(MRect.Rectangle): MFig.Figure}: BOOLEAN;
BEGIN ... END FirstInSecond;
// треугольник разместится внутри треугольника
PROCEDURE FirstInSecond
{VAR f1, f2: MFig.Figure(MTrian.Triangle)}: BOOLEAN;
BEGIN ... END FirstInSecond;
END MRTMM.
```

Клиентский модуль, использующий обобщающие процедуры

```
MODULE Mclient; // Клиентский модуль, обрабатывающий обобщения
IMPORT MRTMM, Console;
VAR
    bool: BOOLEAN; ...
    // Указатели на обобщенные фигуры формируемые
    // вне клиентского модуля
    r, t, c MFig.PFigure;
BEGIN // Собственно обработка
    ...
    // Использование обобщенного вывода
    Mfig.Output{r};
    Mfig.Output{t};
    Mfig.Output{c};
    // Использование мультиметода
    bool := MRTMM.FirstInSecond{r, r};
    bool := MRTMM.FirstInSecond{r, t};
    bool := MRTMM.FirstInSecond{r, c};
    bool := MRTMM.FirstInSecond{t, r};
    bool := MRTMM.FirstInSecond{t, t};
    bool := MRTMM.FirstInSecond{t, c};
    bool := MRTMM.FirstInSecond{c, r};
    bool := MRTMM.FirstInSecond{c, t};
    bool := MRTMM.FirstInSecond{c, c};
END MClient.
```

Добавление специализации

```
MODULE MCirc;
IMPORT In, Out;
TYPE
PCircle* = POINTER TO Circle;
   Circle* = RECORD
   r* : INTEGER // радиус
END;
// Процедура вывода
PROCEDURE Output*(VAR c: Circle);
BEGIN
   Out.String("Circle: r = "); Out.Int(c.r, 0);
   Out.Ln;
END Output;
// Прочие процедуры
...
END MCirc.
```

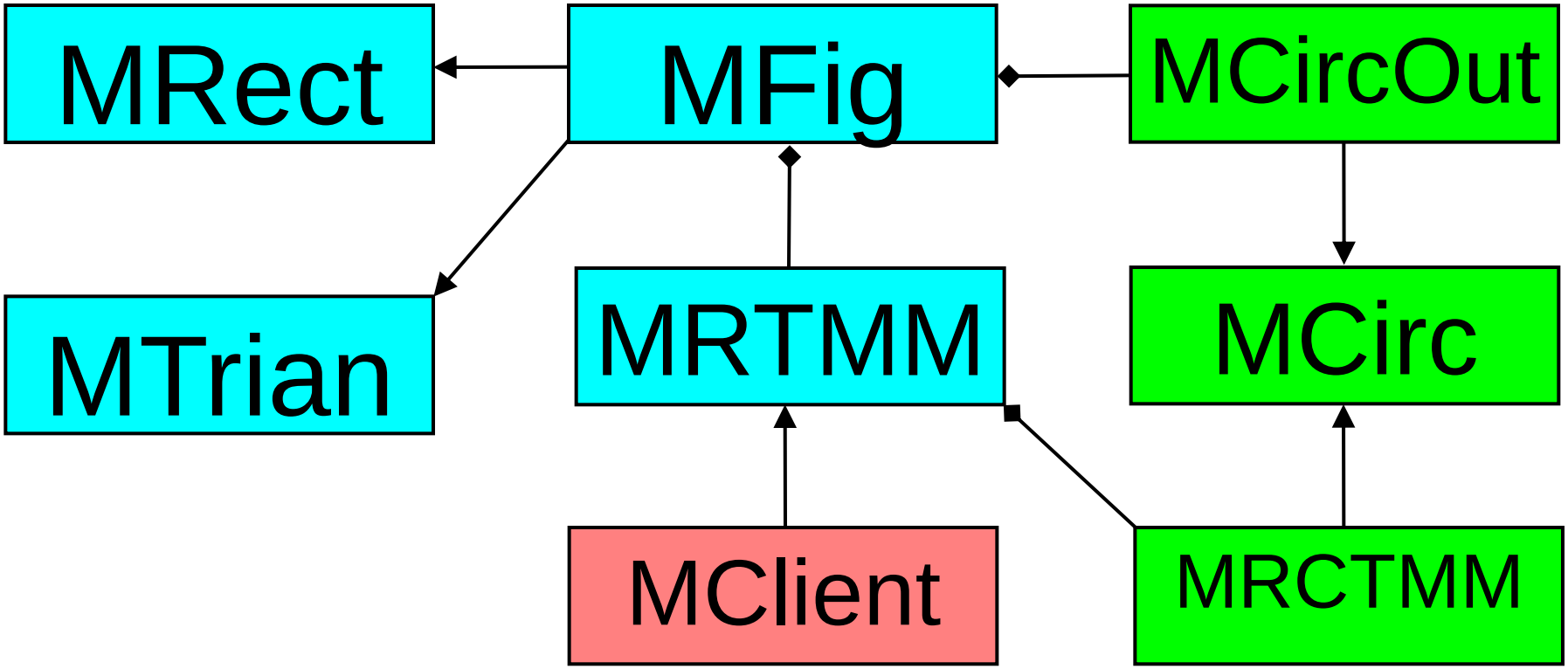
Расширение процедуры вывода после добавления специализации

```
MODULE MCircOut (MFig);
IMPORT In, Out, MCirc;
TYPE
// Расширение обобщения добавлением круга
Figure += MCirc.Circle;
// Вывод обобщенного круга
PROCEDURE Output {VAR c: Figure(MCirc.Circle)};
BEGIN MCircle.Output(c) END Output;
END MCircOut.
```

Расширение мультиметода после добавления специализации

```
MODULE MRCTMM (MRTMM);
IMPORT In, Out, MRect, MTrian, MCirc;
TYPE
// Расширение обобщения добавлением круга
Figure += MCirc.Circle;
// Дополнительные обработчики специализаций
// прямоугольник разместится внутри круга
PROCEDURE FirstInSecond {VAR f1(MRect.Rectangle), f2(MCirc.Circle): Figure}:BOOLEAN;
BEGIN
    Out.String("Rectangle in Circle compare");
    Out.Ln;
    RETURN ((f1.x*f1.x + f1.x*f1.x) < (f2.r*f2.r))
END FirstInSecond;
// треугольник разместится внутри круга
PROCEDURE FirstInSecond {VAR f1(MTrian.Triangle), f2(MCirc.Circle): Figure}:BOOLEAN;
BEGIN ... END FirstInSecond;
// круг разместится внутри прямоугольника
PROCEDURE FirstInSecond {VAR f1(MCirc.Circle), f2(MRect.Rectangle): Figure}:BOOLEAN;
BEGIN ... END FirstInSecond;
// круг разместится внутри треугольника
PROCEDURE FirstInSecond {VAR f1(MCirc.Circle), f2(MTrian.Triangle): Figure}:BOOLEAN;
BEGIN ... END FirstInSecond;
// круг разместится внутри круга
PROCEDURE FirstInSecond {VAR f1, f2: Figure(MCirc.Circle)}:BOOLEAN;
BEGIN
    Out.String("Circle in Circle compare"); Out.Ln;
    RETURN f1.r < f2.r
END FirstInSecond;
END MRTCMM.
```

Зависимости между модулями в ходе эволюционного расширения



Эволюционное расширение с поддержкой динамического связывания

Ситуация	Возможность расширения		
	ПП	**ООП**	**ППП**
1 Расширение обобщений новыми специализациями	**нет**	**есть**	**есть**
2 Добавление новых процедур	**есть**	**нет**	**есть**
3 Добавление новых полей в существующие типы данных	**косвенное, для расширяемых типов**	**косвенное, при наличии RTTI**	**косвенное, при использовании обобщенной записи**
4 Добавление процедур, осуществляющих обработку одной специализации	**есть**	**нет**	**есть процедурный и параметрический**
5 Создание нового обобщения на основе существующих специализаций	**есть**	**косвенное**	**есть**
6 Добавление мультиметодов	**есть**	**нет**	**есть**
7 Изменение мультиметодов при добавлении новых специализаций	**нет**	**нет**	**есть**

Благодарю за внимание!

Близкие по духу работы

1. Фуксман А.Л. Технологические аспекты создания программных систем. – М.: Статистика, 1979. – 184 с.

2. Горбунов-Посадов М.М. Расширяемые программы. – М.: Полиптих, 1999. – 336 с.

3. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного пректирования. Паттерны проектирования. / Пер. с англ. – СПб: Питер, 2001. – 368 с.

4. Влиссидес Дж. Применение шаблонов проектирования. Дополнительные штрихи. / Пер с англ. – М.: Издательский дом «Вильямс», 2003. – 144 с.

Наши ключевые публикации

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.

2. Легалов, А. И. Процедурный язык с поддержкой эволюционного проектирования. / А. И. Легалов, Д. А. Швец // Научный вестник НГТУ. – 2003. – № 2 (15). – С. 25-38.

3. Легалов, И. А. Применение обобщенных записей в процедурно-параметрическом языке программирования. / И. А. Легалов // Научный вестник НГТУ. – 2007. – № 3 (28). – С. 25-38.

4. 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.