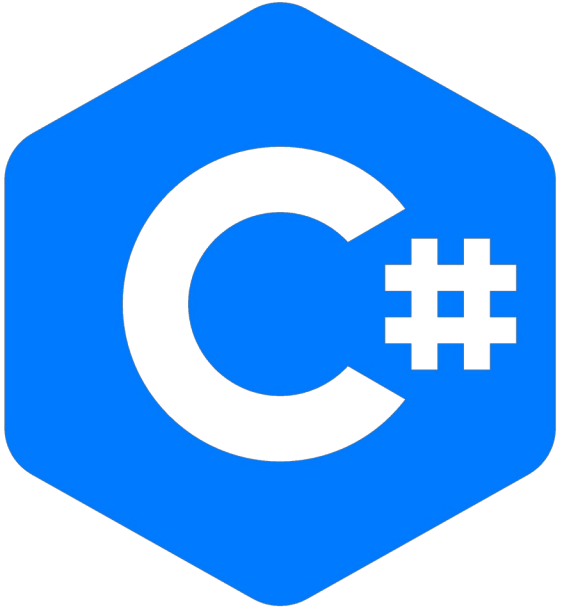
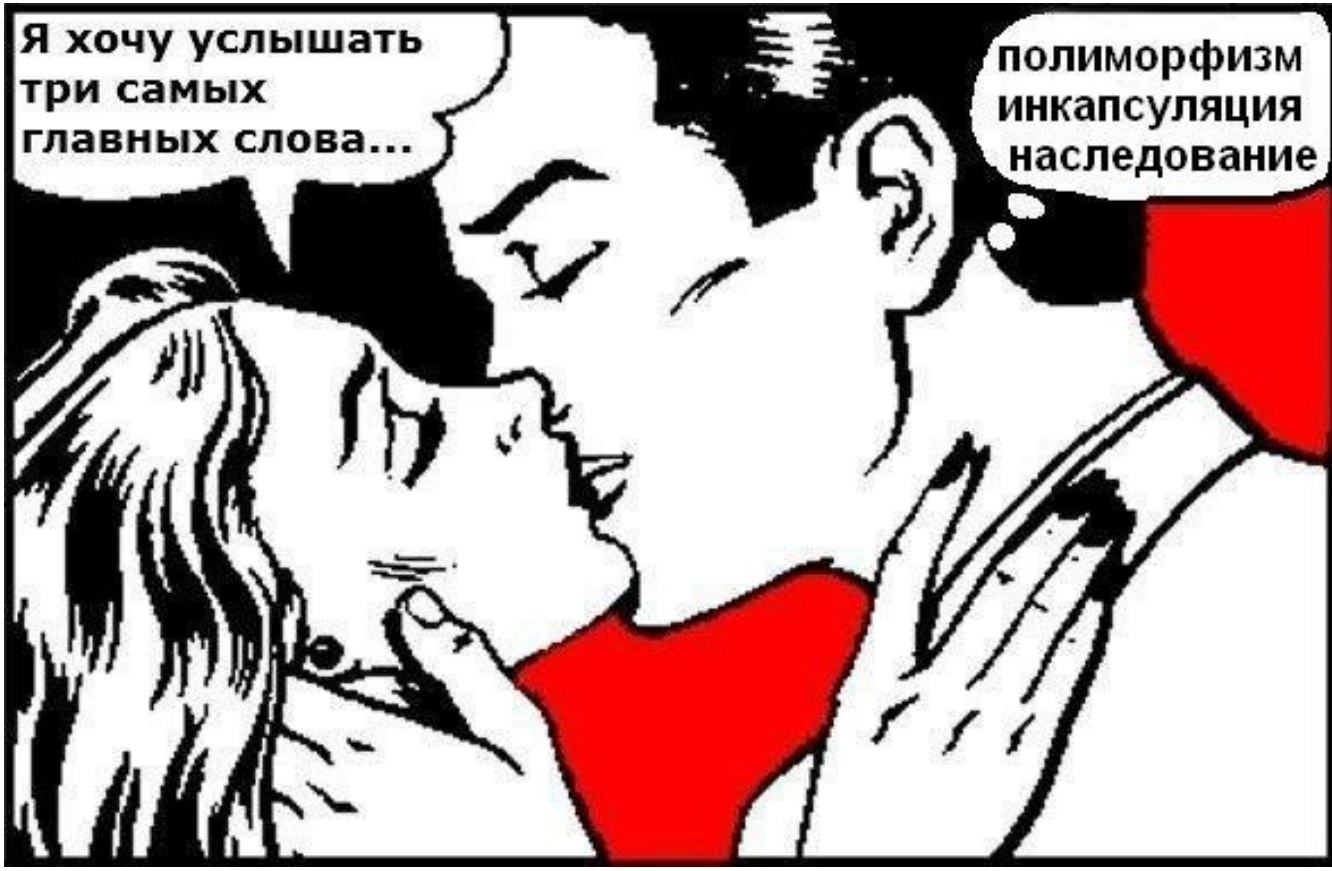
Архитектура уровня статически типизированного объектно-ориентированного языка



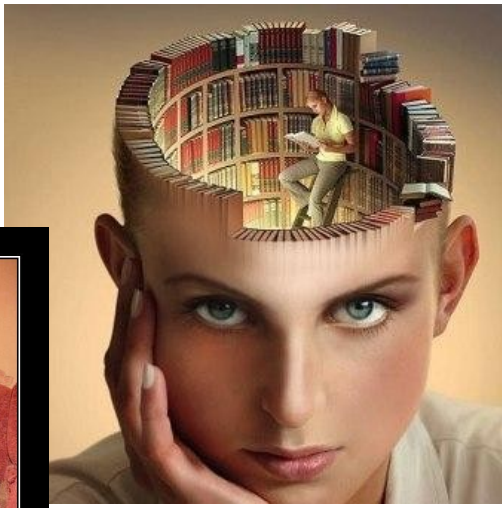
Основные термины и определения

Класс
Базовый класс
Дочерний класс
Инкапсуляция
Наследование
Виртуализация
Полиморфизм



Фундаментальные характеристики ООП (по Алану Кею):

1. Все является **объектом**
2. Вычисления путем взаимодействия между объектами
3. Объект имеет **память**, состоящую **из** других **объектов**
4. Объект - представитель класса, выражающего общие свойства объектов
5. В классе задается **поведение** (функциональность) объекта.
6. Классы организованы в **древовидную** структуру с общим корнем (**иерархия наследования**)



1. Каждый, кто ходит на двух ногах, - враг.

2. Каждый, кто ходит на четырех ногах или у кого есть крылья, - друг.

3. Животные не носят платья.

4. Животные не спят в кроватях.

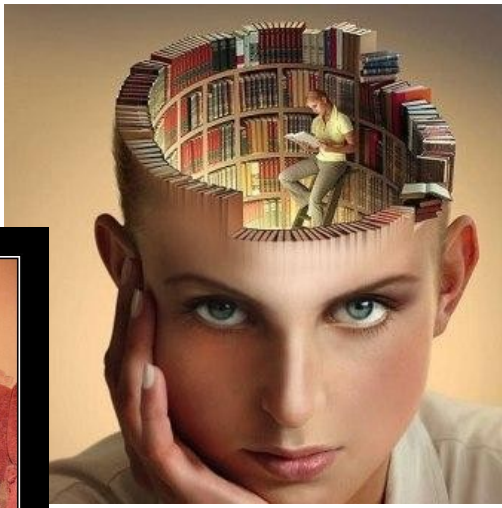
5. Животные не пьют алкоголя.

6. Животное не может убить другое животное.

7. Все животные равны.

Фундаментальные характеристики ООП (по Алану Кею):

1. Все является **объектом**
2. Вычисления путем взаимодействия между объектами
3. Объект имеет **память**, состоящую **из** других **объектов**
4. Объект - представитель класса, выражающего общие свойства объектов
5. В классе задается **поведение** (функциональность) объекта.
6. Классы организованы в **древовидную** структуру с общим корнем (**иерархия наследования**)



Принцип подстановки

Если есть два класса **A** и **B** такие, что класс **B** является **подклассом** класса **A** (возможно, отстоя в иерархии на несколько ступеней), то мы должны иметь возможность **подставить** представителя класса **B вместо** представителя класса **A** в любой ситуации, причем **без видимого проявления эффекта подстановки**.

Агрегаты данных и действий построенные с использованием объектно-ориентированного подхода

```
// Простейший контейнер на основе одномерного массива
class Container {
public:
    Container();
    ~Container();

    // Ввод содержимого контейнера из указанного потока
    void In(ifstream &ifst);
    // Случайный ввод содержимого контейнера
    void InRnd(int size);
    // Вывод содержимого контейнера в указанный поток
    void Out(ofstream &ofst);
    // Вычисление суммы периметров всех фигур в контейнере
    double Perimeter();
private:
    void Clear();    // Очистка контейнера от данных
    int len; // текущая длина
    Shape* storage[10000];
};
```

```
// Вычисление суммы периметров всех фигур в контейнере
double Container::Perimeter() {
    double sum = 0.0;
    for(int i = 0; i < len; i++) {
        sum += storage[i]->Perimeter();
    }
    return sum;
}
```

Свобода для маневра. Динамическое задание размера массива

```
// Простейший контейнер на основе одномерного массива
class Container {
public:
   Container(int s);
   ~Container();

   // Ввод содержимого контейнера из указанного потока
   void In(ifstream &ifst);
   // Случайный ввод содержимого контейнера
   void InRnd(int size);
   // Вывод содержимого контейнера в указанный поток
   void Out(ofstream &ofst);
   // Вычисление суммы периметров всех фигур в контейнере
   double Perimeter();
private:
   void Clear();   // Очистка контейнера от данных
   int len; // текущая длина
   Shape** storage;
   int size;
};
```

```
// Вычисление суммы периметров всех фигур в контейнере
double Container::Perimeter() {
   double sum = 0.0;
   for(int i = 0; i < len; i++) {
      sum += storage[i]->Perimeter();
   }
   return sum;
}
```

Сопоставление процедурного и объектно-ориентированного подходов

Агрегирование

Разнорукое программирование
(http://www.softcraft.ru/paradigm/dhp/index.shtml)

F(X) или X.F()

?

Что лучше?

Основа обобщения в ООП (агрегат с изюминкой)

```
// структура, обобщающая все имеющиеся фигуры
class Shape {
protected:
    static Random rnd20;
    static Random rnd2;
public:
    virtual ~Shape() {};
    // Ввод обобщенной фигуры
    static Shape *StaticIn(ifstream &ifdt);
    // Ввод обобщенной фигуры
    virtual void In(ifstream &ifdt) = 0;
    // Случайный ввод обобщенной фигуры
    static Shape *StaticInRnd();
    // Виртуальный метод ввода случайной фигуры
    virtual void InRnd() = 0;
    // Вывод обобщенной фигуры
    virtual void Out(ofstream &ofst) = 0;
    // Вычисление периметра обобщенной фигуры
    virtual double Perimeter() = 0;
};
```

Специализации (агрегаты - расширители)

```
// прямоугольник
class Rectangle: public Shape {
public:
    virtual ~Rectangle() {}
    // Ввод параметров прямоугольника из файла
    virtual void In(ifstream &ifst);
    // Случайный ввод параметров прямоугольника
    virtual void InRnd();
    // Вывод параметров прямоугольника в форматируемый поток
    virtual void Out(ofstream &ofst);
    // Вычисление периметра прямоугольника
    virtual double Perimeter();
private:
    int x, y; // ширина, высота
};
```

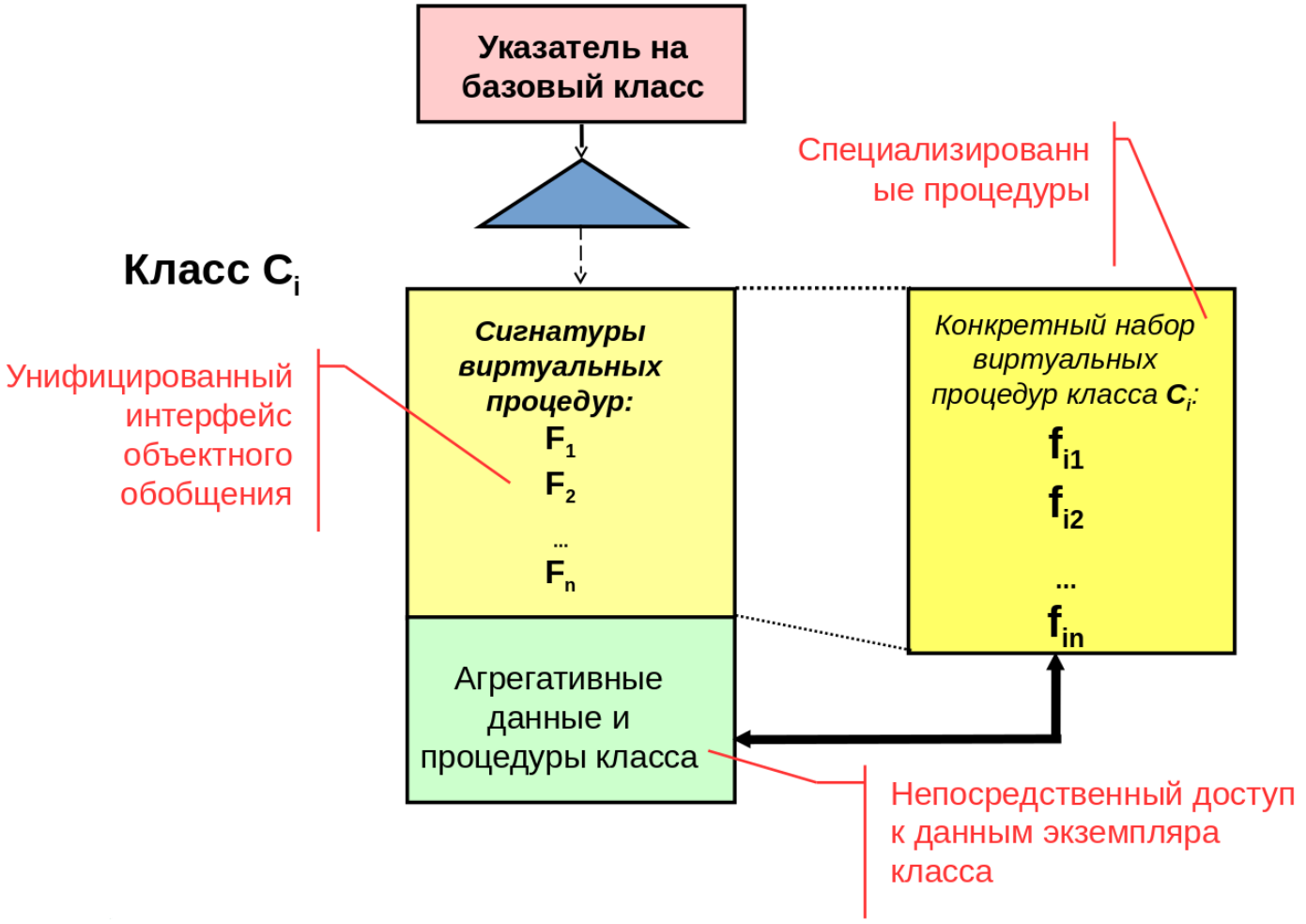
```
// треугольник
class Triangle: public Shape {
public:
    virtual ~Triangle() {}
    // Ввод параметров треугольника из файла
    virtual void In(ifstream &ifst);
    // Случайный ввод параметров треугольника
    virtual void InRnd();
    // Вывод параметров треугольника в форматируемый поток
    virtual void Out(ofstream &ofst);
    // Вычисление периметра треугольника
    virtual double Perimeter();
private:
    int a, b, c; // стороны
};
```

Обработчики обобщения (децентрализация)

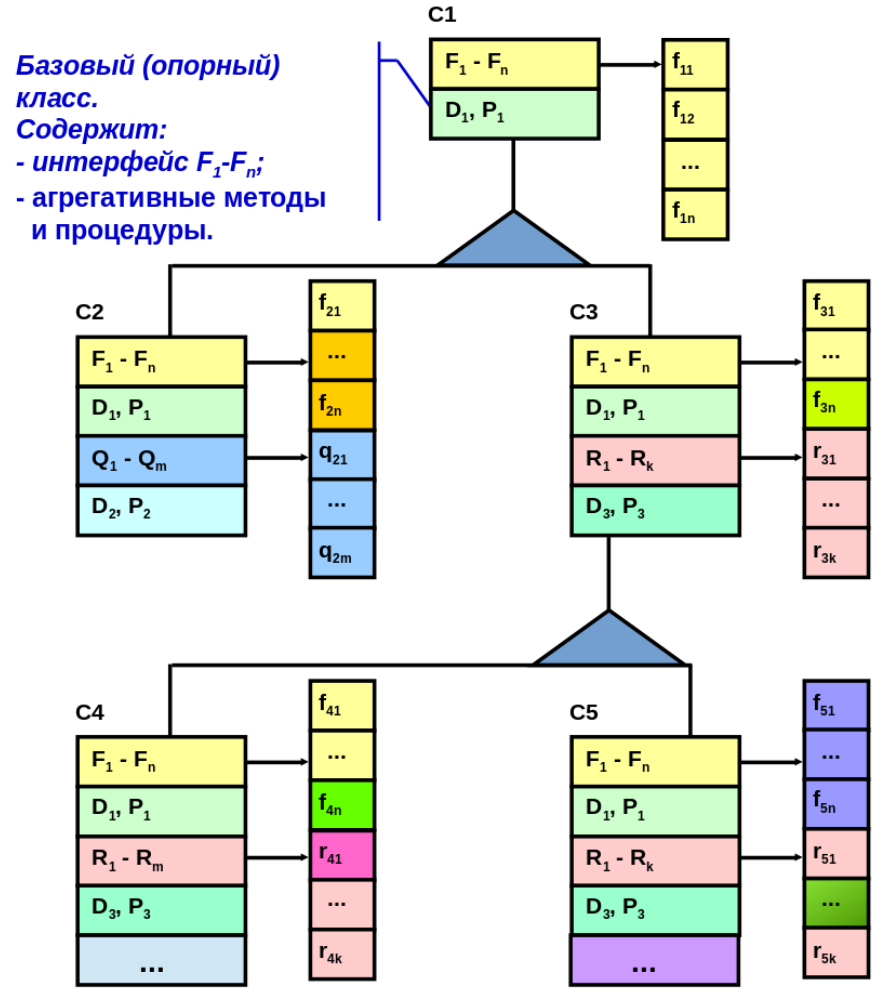
```
// Вычисление периметра прямоугольника
double Rectangle::Perimeter( ) {
    return 2.0 * (x + y);
}
```

```
// Вычисление периметра треугольника
double Triangle::Perimeter( ) {
    return double(a + b + c);
}
```

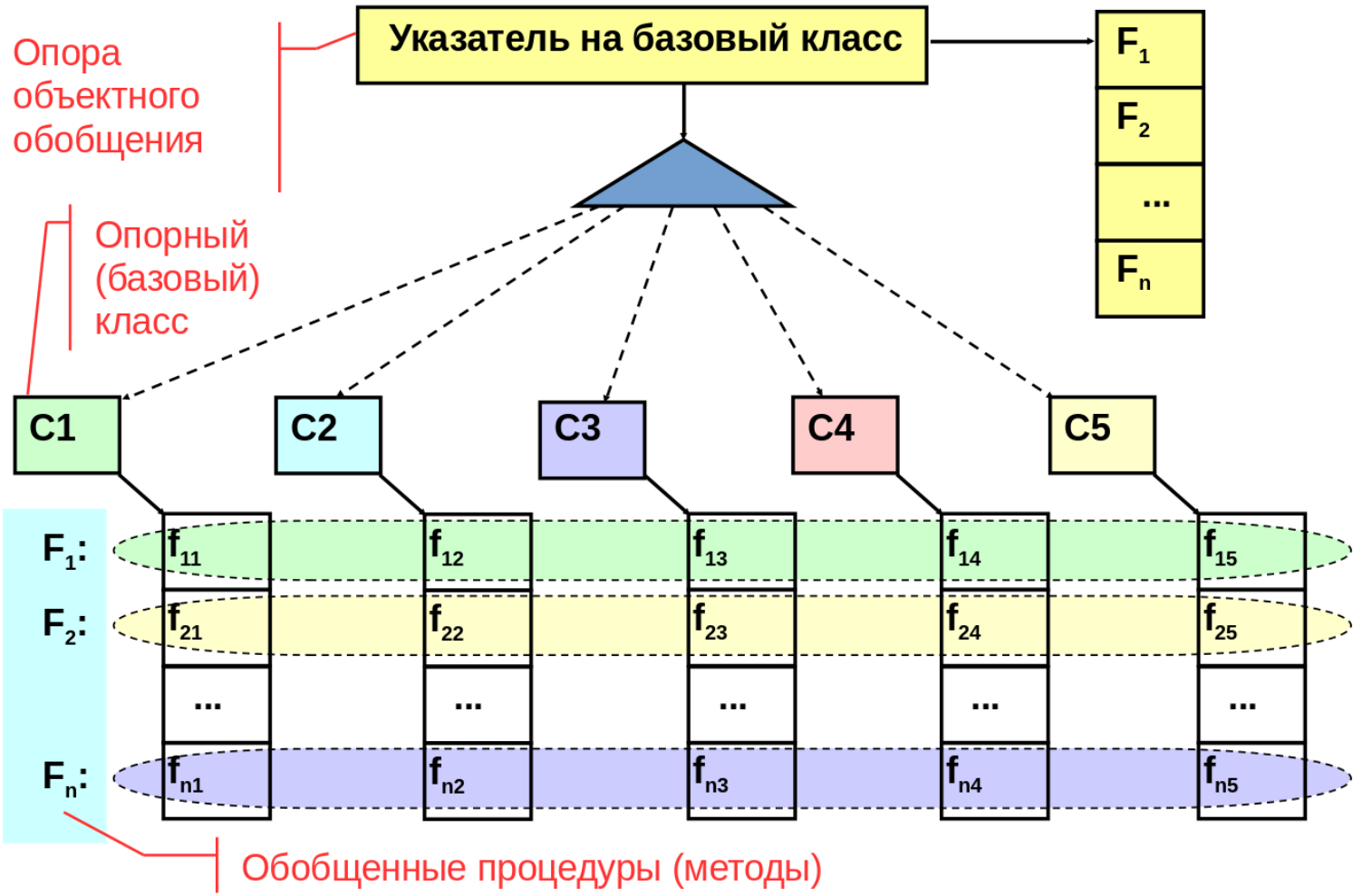
Двойное связывание, обеспечивающее трактовку класса как альтернативы объектного обобщения



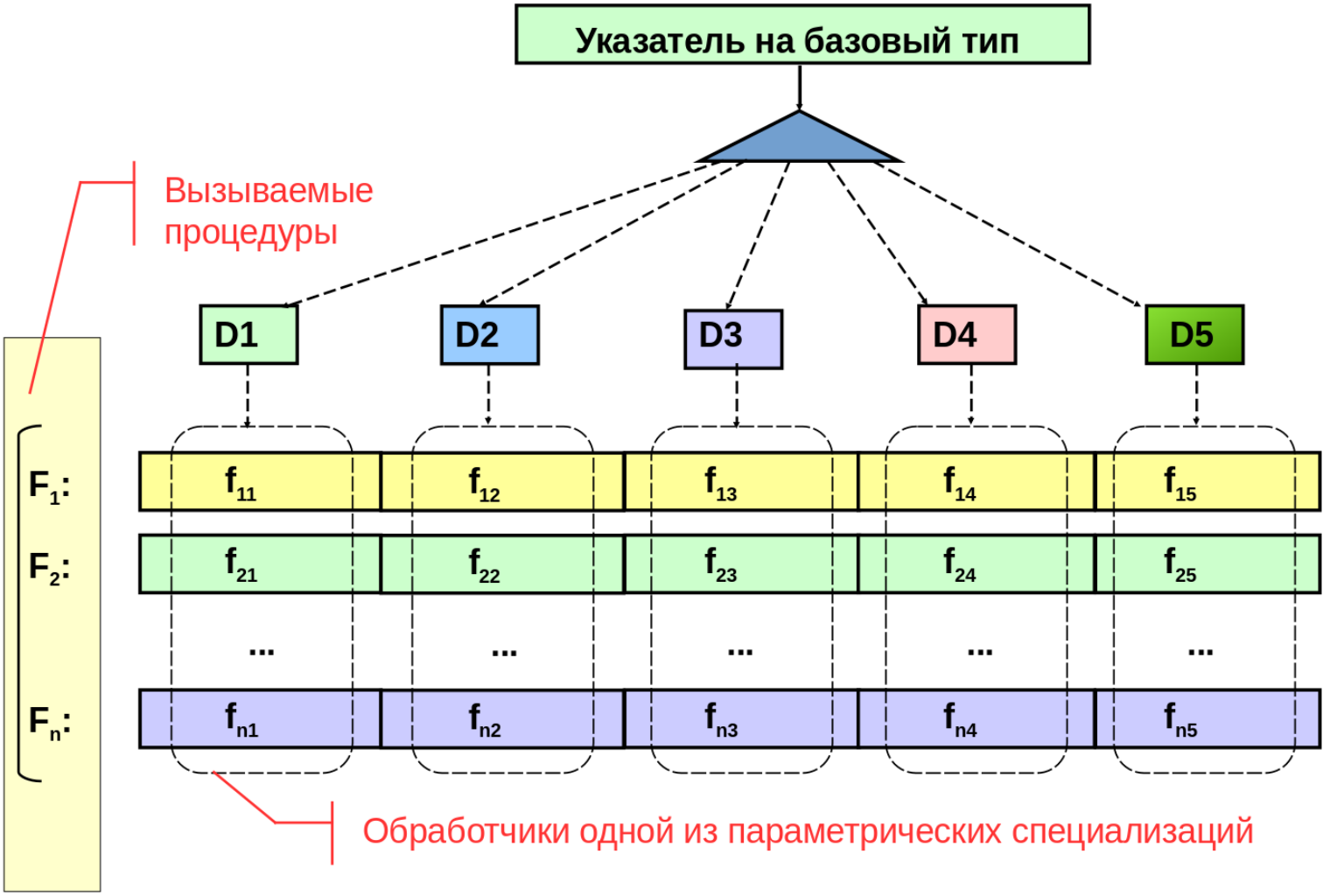
Иерархия классов, построенная с применением наследования и виртуализации

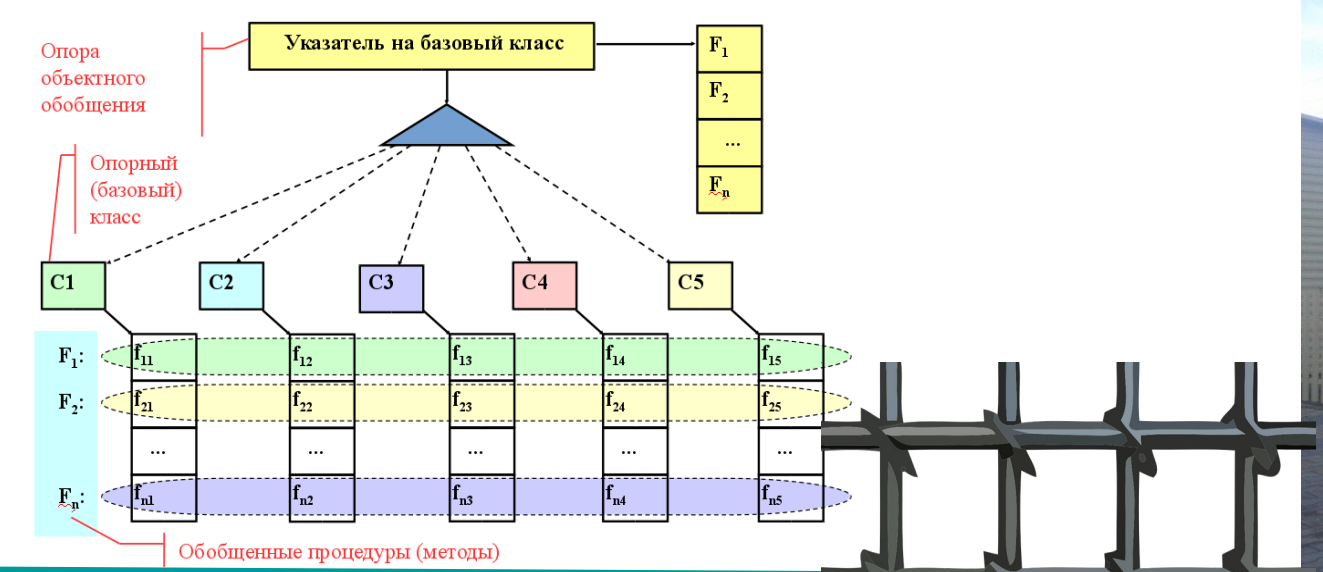


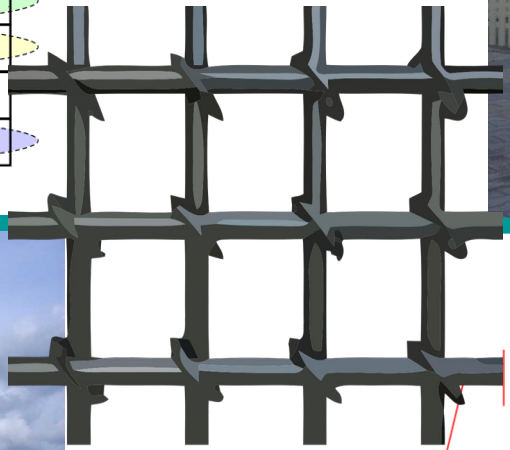
Динамическое подключение альтернатив в объектном обобщении

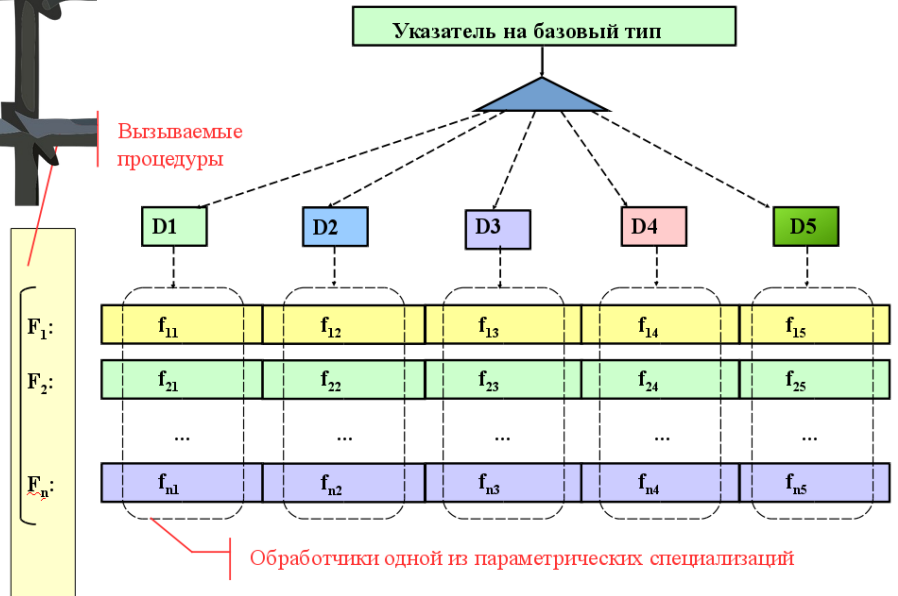


Организация альтернатив при процедурном подходе




Опора объектного обобщения
Указатель на базовый класс
F1
F2
...
Fn
Опорный (базовый) класс
C1
C2
C3
C4
C5
F1:
F2:
Fn:
f11 f12 f13 f14 f15
f21 f22 f23 f24 f25
fn1 fn2 fn3 fn4 fn5
Обобщенные процедуры (методы)




Указатель на базовый тип
Вызываемые процедуры
D1
D2
D3
D4
D5
F1:
F2:
Fn:
f11 f12 f13 f14 f15
f21 f22 f23 f24 f25
fn1 fn2 fn3 fn4 fn5
Обработчики одной из параметрических специализаций

Прямая имитация обобщения

```
//-----------------------------------------------------------
// структура, обобщающая все имеющиеся фигуры
struct shape {
   // Ввод обобщенной фигуры
   void (*In)(shape* self, ifstream &ifdt);
   // Случайный ввод обобщенной фигуры
   // Виртуальный метод ввода случайной фигуры
   void (*InRnd)(shape* self);
   // Вывод обобщенной фигуры
   void (*Out)(shape* self, ofstream &ofst);
   // Вычисление периметра обобщенной фигуры
   double (*Perimeter)(shape* self);
   // Удаление обобщенной фигуры
   void (*Delete)(shape* self);
};

// Ввод параметров обобщенной фигуры из файла
shape* In(ifstream &ifst);

// Случайный ввод обобщенной фигуры
shape *InRnd();
```

Формирование специализаций, имитирующих наследников

```
// Ввод параметров обобщенной фигуры из файла
shape* In(ifstream &ifst) {
  shape *sp;
  int k;
  ifst >> k;
  switch(k) {
    case 1:
      sp = createRectangleAsShape();
      break;
    case 2:
      sp = createTriangleAsShape();
      break;
    default:
      return 0;
  }
  sp->In(sp, ifst);
  return sp;
}
```

Использование основы специализации для формирования прямоугольника как фигуры

```
// прямоугольник
struct rectangle {
   int x, y; // ширина, высота
};
```

```
// прямоугольник как фигура
struct rectangleAsShape {
   shape base;
   rectangle spec;
};
```

Инициализация указателей на функции, имитирующих объектно-ориентированные методы

```
// Инициализация прямоугольника как фигуры
void init(rectangleAsShape &ts) {
    ts.base.In = inRectangleAsShape;
    ts.base.InRnd = inRndRectangleAsShape;
    ts.base.Out = outRectangleAsShape;
    ts.base.Perimeter = perimeterRectangleAsShape;
    ts.base.Delete = deleteRectangleAsShape;
}
```

Имитация виртуализации для прямоугольника-фигуры с использованием ранее разработанного прямоугольника

```
// Вывод прямоугольника как фигуры
void outRectangleAsShape(shape* self, ofstream &ofst) {
   rectangleAsShape* ptr = (rectangleAsShape*)self;
   Out(ptr->spec, ofst);
}

//-----------------------------------------------------------
// Вычисление периметра прямоугольника как фигуры
double perimeterRectangleAsShape(shape* self) {
   rectangleAsShape* ptr = (rectangleAsShape*)self;
   return Perimeter(ptr->spec);
}
```

Контейнер остается неизменным

```
// Простейший контейнер на основе одномерного массива
struct container {
   enum {max_len = 10000}; // максимальная длина
   int len; // текущая длина
   shape *cont[max_len];
};
```

```
// Вычисление суммы периметров всех фигур в контейнере
double PerimeterSum(container &c) {
   double sum = 0.0;
   for(int i = 0; i < c.len; i++) {
      sum += c.cont[i]->Perimeter(c.cont[i]);
   }
   return sum;
}
```

Имитация таблицы виртуальных методов

```
// таблица виртуальных методов
struct VT {
   // Ввод обобщенной фигуры
   void (*In)(shape* self, ifstream &ifdt);
   // Случайный ввод обобщенной фигуры
   // Виртуальный метод ввода случайной фигуры
   void (*InRnd)(shape* self);
   // Вывод обобщенной фигуры
   void (*Out)(shape* self, ofstream &ofst);
   // Вычисление периметра обобщенной фигуры
   double (*Perimeter)(shape* self);
   // Удаление обобщенной фигуры
   void (*Delete)(shape* self);
};
```

Имитация абстрактного класса с таблицей виртуальных методов

```
// структура, обобщающая все имеющиеся фигуры
struct shape {
   // Указатель на таблицу виртуальных методов
   VT* vtPtr;
};
```

Имитация производного класса и его методов

```
// прямоугольник как фигура
struct rectangleAsShape {
    shape base;
    rectangle spec;
};
```

```
// Вычисление периметра прямоугольника как фигуры
double perimeterRectangleAsShape(shape* self) {
    rectangleAsShape* ptr = (rectangleAsShape*)self;
    return Perimeter(ptr->spec);
}

//-------------------------------------------------------
-
// Удаление прямоугольника как фигуры
void deleteRectangleAsShape(shape* self) {
    rectangleAsShape* ptr = (rectangleAsShape*)self;
    delete ptr;
}
```

Формирование таблицы виртуальных методов прямоугольника

```
// Таблица виртуальных методов прямоугольника
static VT vtRectangle = {
    inRectangleAsShape,
    inRndRectangleAsShape,
    outRectangleAsShape,
    perimeterRectangleAsShape,
    deleteRectangleAsShape
};
```

Изменения в обработке элементов контейнера

```
// Простейший контейнер на основе одномерного массива
struct container {
   enum {max_len = 10000}; // максимальная длина
   int len; // текущая длина
   shape *cont[max_len];
};
```

```
// Вычисление суммы периметров всех фигур в контейнере
double PerimeterSum(container &c) {
   double sum = 0.0;
   for(int i = 0; i < c.len; i++) {
      sum += c.cont[i]->vtPtr->Perimeter(c.cont[i]);
   }
   return sum;
}
```