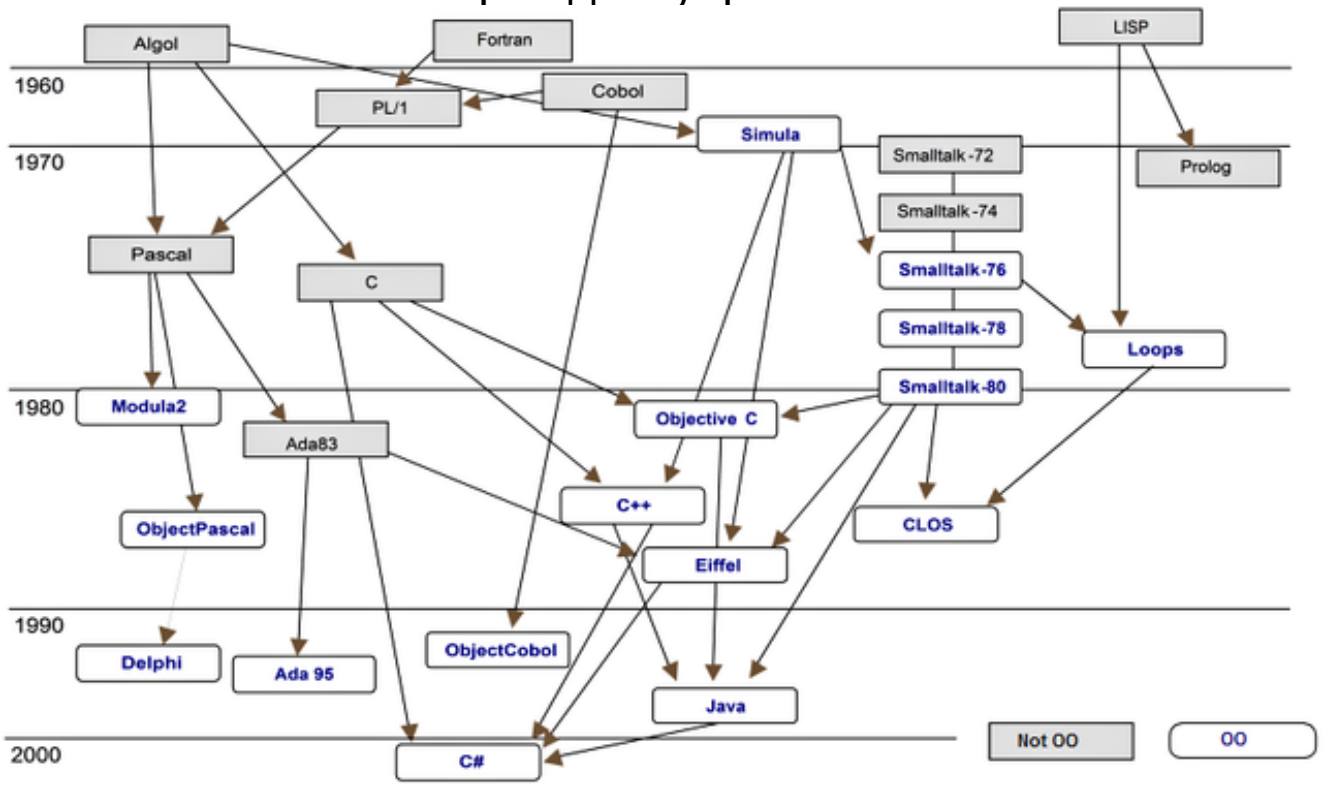
Компьютерные технологии в физике элементарных частиц.

Языки программирования и основы проектирования

Языки программирования

Язык программирования — формальная знаковая система, предназначенная для записи компьютерных программ. Язык программирования определяет набор лексических, синтаксических и семантических правил, задающих внешний вид программы и действия, которые выполнит компьютер под её управлением.



Понятие парадигмы программирования

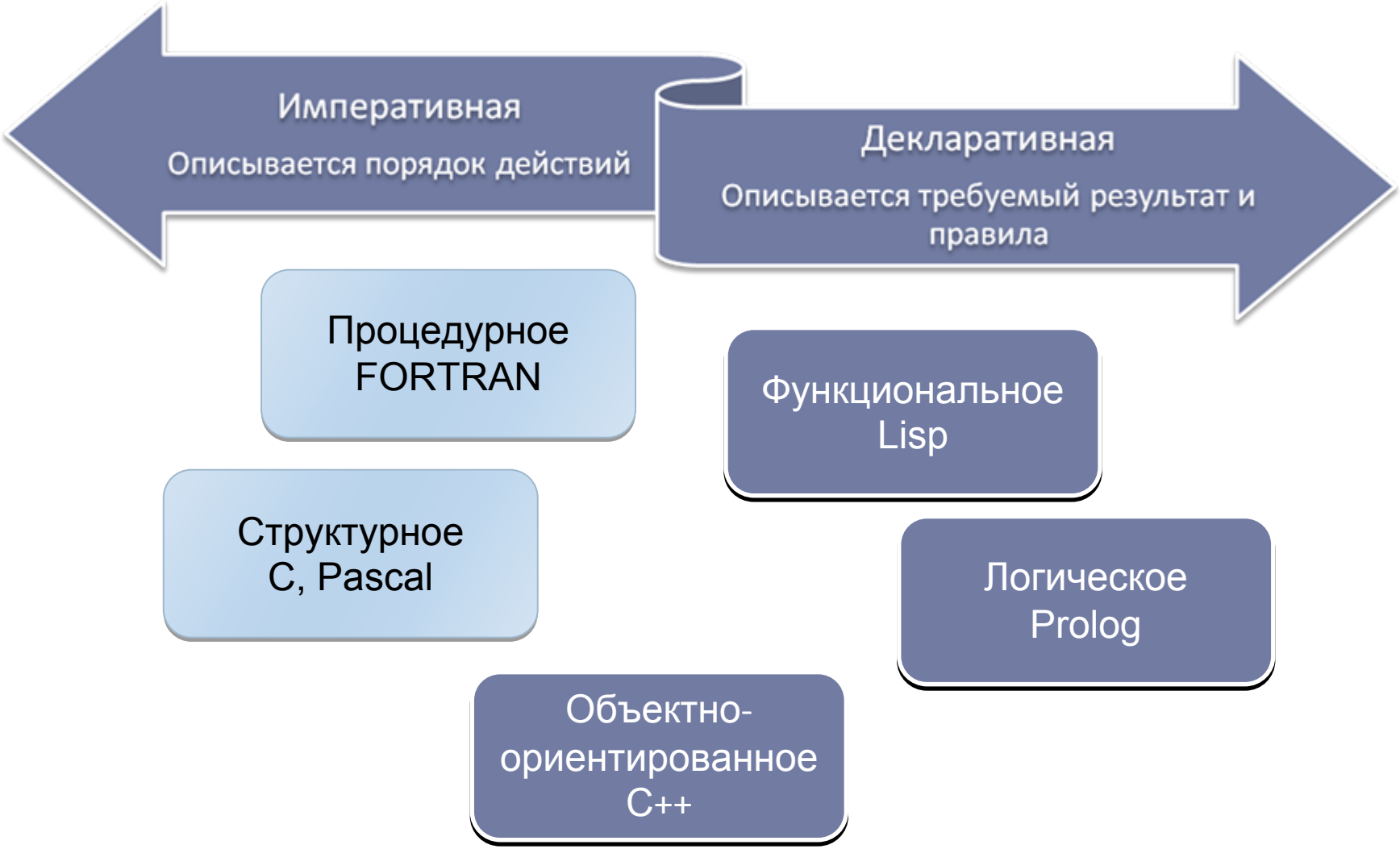
Любой язык программирования является мостом между человеком и ЭВМ. Поэтому любой язык должен быть понятен ЭВМ (т.е. позволять трансляцию в машинный код), и должен быть понятен и удобен человеку. Но человеческое мышление очень многообразно, что приводит к тому, что языки программирования (как и человеческие языки) могут кардинально отличаться.

Парадигма программирования – это совокупность идей и понятий, определяющих стиль написания программ. Парадигма определяет набор терминов (абстракций), которые отражаются в синтаксисе языка и на основе которых строится структура программы.

Парадигма программирования и язык программирования – связанные понятия, но это не одно и то же. Для каждой парадигмы программирования существует множество языков, которые ей следуют. Существуют множество языков программирования, которые позволяют писать программы в разных парадигмах.

Большинство парадигм можно отнести к одному из классов: императивные и декларативные. Наиболее распространенной на сегодня является парадигма объектно-ориентированного программирования.

Основные парадигмы программирования



Классификация языков

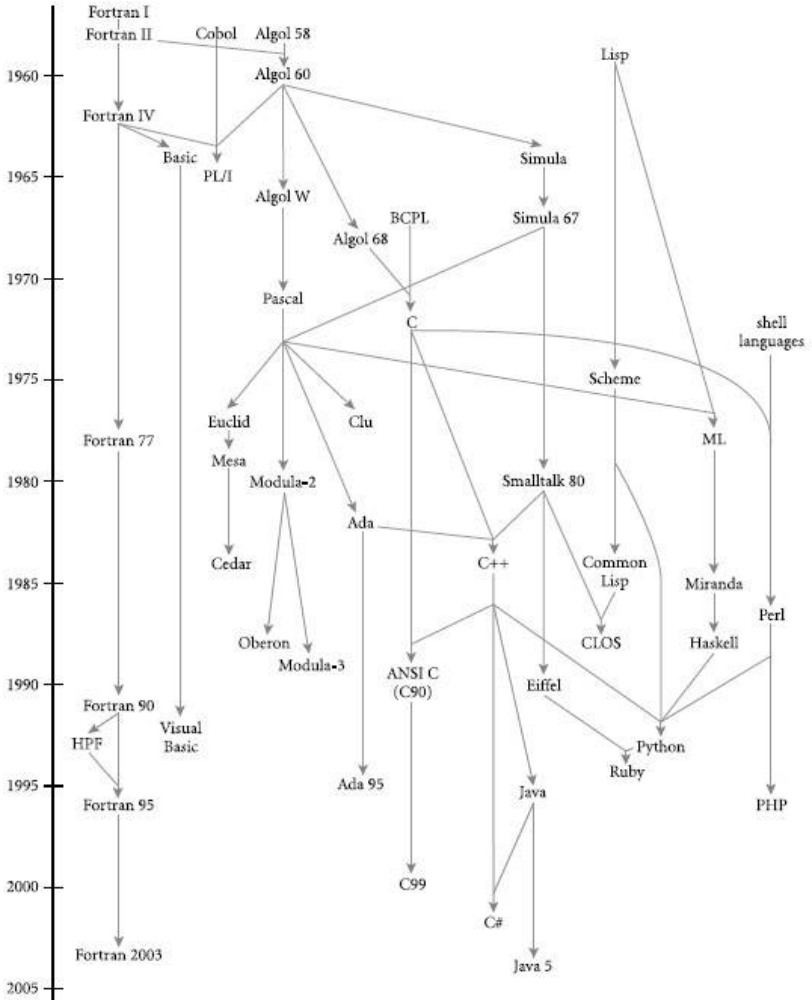
Существует несколько тысяч языков программирования. Их можно объединить в группы по различным свойствам.

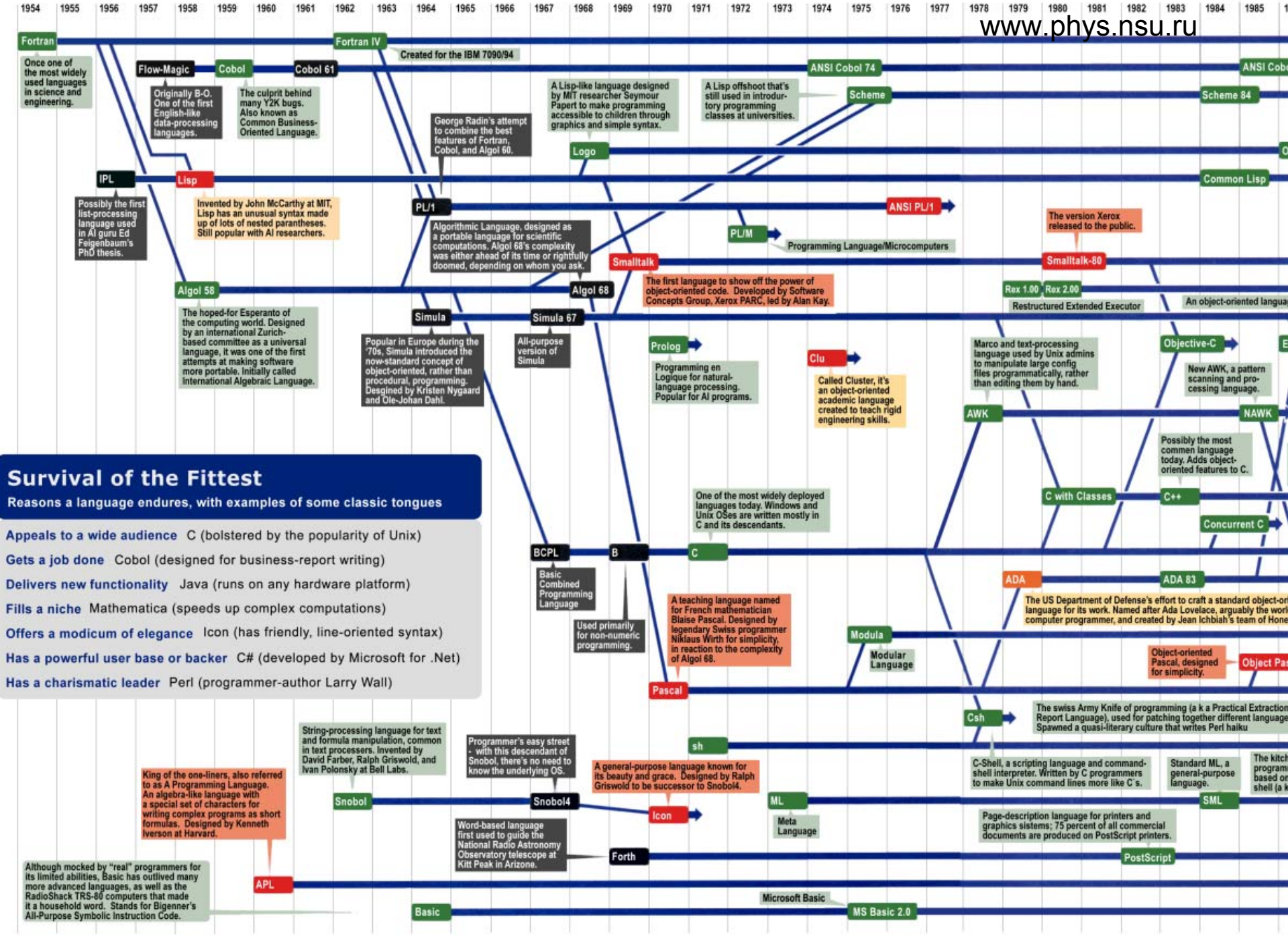
По поддерживаемым парадигмам: •Процедурные (FORTRAN) •Структурные (Pascal) •Объектно-ориентированные (C++, Python) •Функциональные (LISP, Python) •Логические (Prolog)	**По назначению:** • Универсальные (C++,Java) • Языки сценариев (sh, Perl) • Представление данных (XML) • Запросов (SQL)
По типизации: • Статическая/динамическая • Строгая/слабая	**По модели исполнения:** • Компилируемые (C++) • Интерпретируемые (Perl) • Гибридные (Java)

А также: по выразительности, по сложности, по производительности и т.п.

Языки, оказавшие большое влияние

FORTRAN	1957, научные расчеты
LISP	1958, первый язык ФП, обработка списков
ALGOL	1958, язык записи алгоритмов
PASCAL	1970, структурный, учебный
C	1969, системное программирование
Prolog	1972, логическое программирование
C++	1983, объектно-ориентированный
Java	1995, переносимый, web-сценарии
Perl	1987, популярный язык сценариев
Python	1990, современная замена Perl

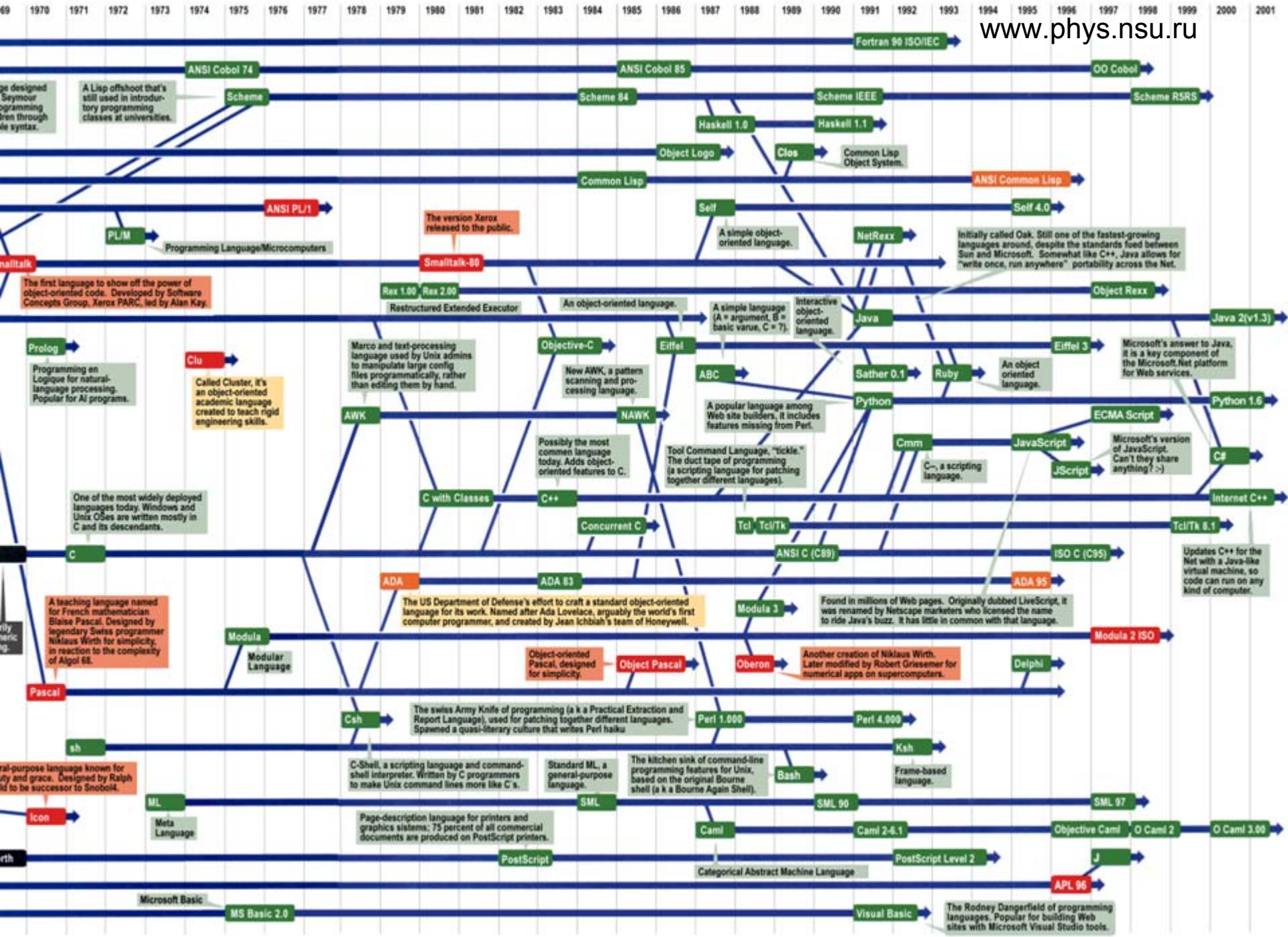




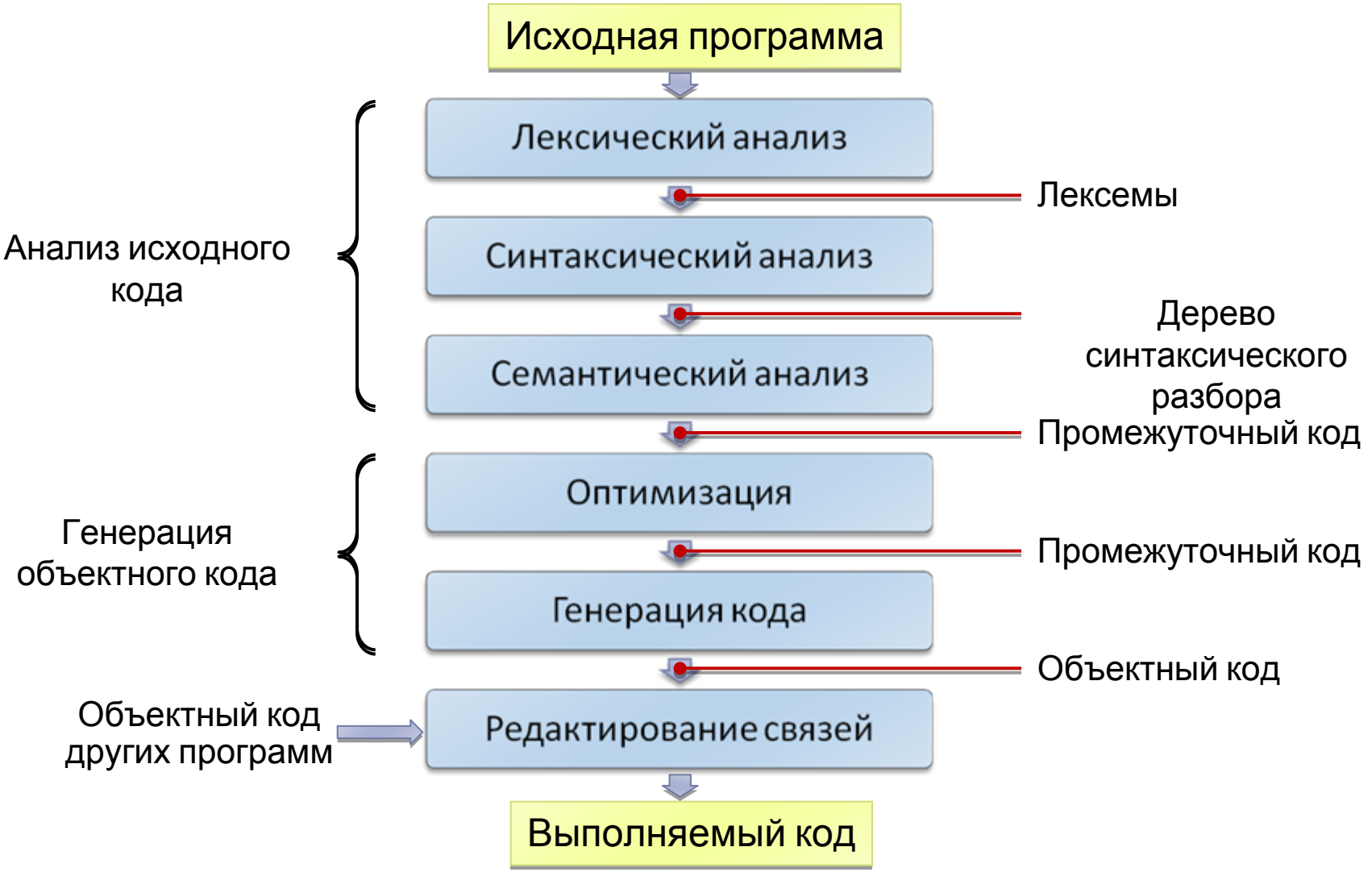
Survival of the Fittest

Reasons a language endures, with examples of some classic tongues

- **Appeals to a wide audience** C (bolstered by the popularity of Unix)
- **Gets a job done** Cobol (designed for business-report writing)
- **Delivers new functionality** Java (runs on any hardware platform)
- **Fills a niche** Mathematica (speeds up complex computations)
- **Offers a modicum of elegance** Icon (has friendly, line-oriented syntax)
- **Has a powerful user base or backer** C# (developed by Microsoft for .Net)
- **Has a charismatic leader** Perl (programmer-author Larry Wall)


www.phys.nsu.ru
1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001
Fortran 90 ISO/IEC
ANSI Cobol 74
ANSI Cobol 85
OO Cobol
A Lisp offshoot that's still used in introductory programming classes at universities.
Scheme
Scheme 84
Scheme IEEE
Scheme R5RS
Haskell 1.0
Haskell 1.1
Object Logo
Clos
Common Lisp Object System.
Common Lisp
ANSI Common Lisp
ANSI PL/1
Self
Self 4.0
The version Xerox released to the public.
PL/M
Programming Language/Microcomputers
A simple object-oriented language.
NetRexx
Initially called Oak. Still one of the fastest-growing languages around, despite the standards fued between Sun and Microsoft. Somewhat like C++, Java allows for "write once, run anywhere" portability across the Net.
Smalltalk-80
The first language to show off the power of object-oriented code. Developed by Software Concepts Group, Xerox PARC, led by Alan Kay.
Rex 1.00
Rex 2.00
Object Rexx
Restructured Extended Executor
An object-oriented language.
A simple language (A = argument, B = basic varue, C = ?).
Interactive object-oriented language.
Java
Java 2(v1.3)
Prolog
Programming en Logique for natural-language processing. Popular for AI programs.
Clu
Called Cluster, it's an object-oriented academic language created to teach rigid engineering skills.
Marco and text-processing language used by Unix admins to manipulate large config files programmatically, rather than editing them by hand.
Objective-C
Eiffel
Eiffel 3
Microsoft's answer to Java, it is a key component of the Microsoft.Net platform for Web services.
New AWK, a pattern scanning and processing language.
ABC
Sather 0.1
Ruby
An object oriented language.
AWK
NAWK
A popular language among Web site builders, it includes features missing from Perl.
Python
Python 1.6
ECMA Script
Possibly the most commen language today. Adds object-oriented features to C.
Tool Command Language, "tickle." The duct tape of programming (a scripting language for patching together different languages).
Cmm
JavaScript
Microsoft's version of JavaScript. Can't they share anything? :-)
C--, a scripting language.
JScript
C#
One of the most widely deployed languages today. Windows and Unix OSes are written mostly in C and its descendants.
C with Classes
C++
Internet C++
Concurrent C
Tcl
Tcl/Tk
Tcl/Tk 8.1
C
ANSI C (C89)
ISO C (C95)
Updates C++ for the Net with a Java-like virtual machine, so code can run on any kind of computer.
ADA
ADA 83
ADA 95
A teaching language named for French mathematician Blaise Pascal. Designed by legendary Swiss programmer Niklaus Wirth for simplicity, in reaction to the complexity of Algol 68.
The US Department of Defense's effort to craft a standard object-oriented language for its work. Named after Ada Lovelace, arguably the world's first computer programmer, and created by Jean Ichbiah's team of Honeywell.
Modula 3
Found in millions of Web pages. Originally dubbed LiveScript, it was renamed by Netscape marketers who licensed the name to ride Java's buzz. It has little in common with that language.
Modula
Modular Language
Modula 2 ISO
Object-oriented Pascal, designed for simplicity.
Object Pascal
Oberon
Another creation of Niklaus Wirth. Later modified by Robert Griesemer for numerical apps on supercomputers.
Delphi
Pascal
Csh
The swiss Army Knife of programming (a k a Practical Extraction and Report Language), used for patching together different languages. Spawned a quasi-literary culture that writes Perl haiku
Perl 1.000
Perl 4.000
sh
Ksh
C-Shell, a scripting language and command-shell interpreter. Written by C programmers to make Unix command lines more like C's.
Standard ML, a general-purpose language.
The kitchen sink of command-line programming features for Unix, based on the original Bourne shell (a k a Bourne Again Shell).
Bash
Frame-based language.
ML
Meta Language
SML
SML 90
SML 97
Icon
Page-description language for printers and graphics sistems; 75 percent of all commercial documents are produced on PostScript printers.
Caml
Caml 2-6.1
Objective Caml
O Caml 2
O Caml 3.00
PostScript
PostScript Level 2
Categorical Abstract Machine Language
J
APL 96
Microsoft Basic
MS Basic 2.0
Visual Basic
The Rodney Dangerfield of programming languages. Popular for building Web sites with Microsoft Visual Studio tools.

Основные этапы компиляции



Формальные грамматики

Лексический и синтаксический (а также, частично, семантический) анализ основан на теории формальных языков.

Формальная грамматика — способ описания формального языка, то есть выделения некоторого подмножества из множества всех слов некоторого конечного алфавита.

Терминальный символ — объект, непосредственно присутствующий в словах языка (буква, цифра, специальный символ).

Нетерминальный символ — объект, обозначающий какую-либо *сущность* языка (например: формула, арифметическое выражение, команда) и не имеющий конкретного символьного значения.

Грамматика задается:

1. Алфавитами терминальных и нетерминальных символов
2. Набором правил A→B, где в A есть хоть один нетерминальный символ
3. Начальным нетерминальным символом

Пример грамматики

Пример: следующая грамматика описывает арифметические выражения со скобками, такие как $1 + 2 * (3 + 5)$

Терминальный алфавит:{'0','1','2','3','4','5','6','7','8','9','+','-','*','/','(',')'}.

Нетерминальный алфавит:{ ФОРМУЛА, ЗНАК, ЧИСЛО, ЦИФРА }

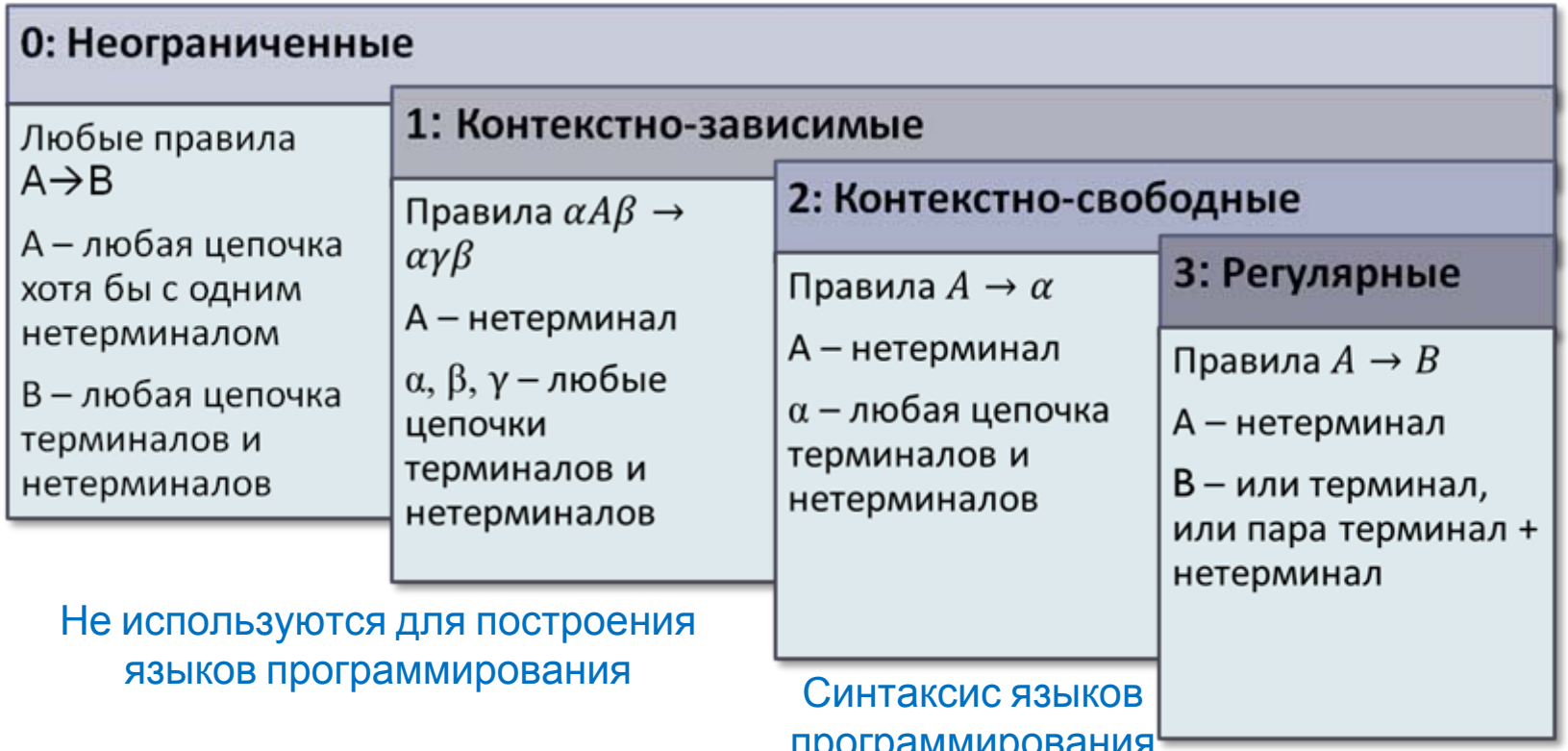
Правила:

1.ФОРМУЛА → ФОРМУЛА ЗНАК ФОРМУЛА
2.ФОРМУЛА → ЧИСЛО
3.ФОРМУЛА → (ФОРМУЛА)
4.ЗНАК → + | - | * | /
5.ЧИСЛО → ЦИФРА
6.ЧИСЛО → ЧИСЛО ЦИФРА
7.ЦИФРА → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Начальный нетерминал: ФОРМУЛА

Иерархия Хомского

Согласно классификации, предложенной Н.Хомским, все формальные языки делятся на 4 группы по сложности



Не используются для построения языков программирования

Синтаксис языков программирования

Регулярные выражения, лексический разбор

Пример: язык TLT описания третичного триггера КМД-2

Иногда для решения конкретной задачи удобно разработать специализированный язык программирования. Например, был разработан специализированный язык для описания алгоритма работы третичного триггера детектора КМД-2.

```
/* Initialize event counter */
global NEVENT = 0;

/* Create  histograms */
book   4000 'First Level Trigger Bits ' 16 1 17;
book   4005 'TF+2*NT+4*BGO+8*NTN' 16 0 16;
book   4611 'e+e- -> e+e- energy ' 50 0 2000;
book   4612 'e+e- -> e+e- phi distribution '  56 0 56;
bookpr 4110 'Average amplitude vs time, DC ' 100 0 600;
book2  4901 'Hitted BGO crystals' 22 0.5 22.5 22 0.5 22.5;

/* New variables definition */
TF_event      = FLT[1];
Neutral_event = FLT[12] && !TF_event;
SingleTrack  = NDCMAXS==1 && HDCMAXS<18 && WDCMAXS<7;
Bhabha_Ecl_Min = 300.0;
IHardCsiClusters = ITRUE(ECLCSI>300);
CsI_Bhabha = (ETOTCSI>500) && (NCLCSI>1) && (NCLCSI<4) &&
             LEN(IHardCsiclusters)==2 &&
             ALL(NHZACSI[IHardCsiClusters]>0);
TriggerCombination = ( FLT[1]  ? 1 : 0 ) + ( FLT[12] ? 2 : 0 ) +
                     ( FLT[5]  ? 4 : 0 ) + ( FTL[10] ? 8 : 0 );
```

```
/* Mark events:                                                  */
/* Bit    Event type                                             */
/*   1    Event, triggered by trackfinder                        */
/*   2    Neutral event, trigger by neutral trigger only         */
/*   3    e+e- -> e+e- event in CsI calorimeter                  */
mark ( TF_event ) in 1;
mark ( Neutral_event ) in 2;
mark ( CsI_Bhabha ) in 3;

/* Fill histograms */
fillbit 4000 by FLT;
fill 4005 by TriggerCombination;
if ( CsI_Bhabha ) {
   fill 4611 by ETOTCSI;
   fill 4612 by PHICLCSI[IHardCsiClusters];
}
if( SingleTrack ) fill 4110 by AWIREDC versus TWIREDC;
fill 4901 by YCRBGO versus XCRBGO;

/* Print trace information */
NEVENT = NEVENT + 1;
if( (NEVENT%1000)==0 ) print NEVENT,' events processed';

/* Specify return code /
if( !SingleTrack ) return 1; else return 0;
```

Особенности: С-подобный, декларативный, векторный, учитывает особенности структуры данных и время их жизни, позволяет очень эффективную реализацию

Утилиты YACC и LEX

Существуют стандартные программные утилиты для разработки лексических и синтаксических анализаторов.

Утилита lex (flex, jlex) позволяет создать лексический анализатор. На вход задается набор правил (регулярная грамматика), на выходе получается программа, которая разбивает входной поток символов на лексемы.

Утилита yacc (bison) позволяет создать синтаксический анализатор. На вход задается набор правил (контекстно-свободная грамматика), на выходе получается программа, которая анализирует входной поток лексем и идентифицирует языковые конструкции. Как правило, используется в паре с lex.

Грамматика для yacc задается в расширенной форме Бэкуса – Наура (БНФ). П

```
Число = ["+"|"-"]НатЧисло["."[НатЧисло]][("e"|"E")["+"|"-
"]НатЧисло]
НатЧисло = Цифра{Цифра}
Цифра = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
Идент = Буква{Буква|Цифра|"_«}
```

Пример: описание лексики TLT

```
skip " "|"\n"|"\t"
oper ","|"<"|">"|"("|")"|"!"|"="|":"|"?"|"["|"]"|"+"|"-"|"/"|"%"|"*"|"{"|"}"
line ";"
ident [a-zA-Z][a-zA-Z0-9_]*
fnumb [0-9]+\.[0-9]+
inumb [0-9]+
str    "'"[^']*"'"
or  ("|""|"|"."or".")
and ("&""&"|"."and".")
not "."not"."
eq  (==|"."eq".")
ne  (!=|"."ne".")
lt  "."lt"."
gt  "."gt"."
le  (<=|=<|"."le".")
ge  (>=|=>|"."ge".")
```

```
"/""*"  { ncomment++; }
"*""/"  { if( ncomment>0 ) ncomment--; else return t_ERROR; }
{str}   { … }
global  { if( ncomment==0 ) return t_GLOBAL; }
return  { if( ncomment==0 ) return t_RETURN; }
print   { if( ncomment==0 ) return t_PRINT;  }
book    { if( ncomment==0 ) return t_BOOK;   }
book2   { if( ncomment==0 ) return t_BOOK2;  }
bookpr  { if( ncomment==0 ) return t_BOOKPR; }
fill    { if( ncomment==0 ) return t_FILL;   }
fillbit { if( ncomment==0 ) return t_FILLBIT;}
versus  { if( ncomment==0 ) return t_VERSUS; }
mark    { if( ncomment==0 ) return t_MARK;   }
{fnumb} { if( ncomment==0 ) { yylval.fval = atof(yytext);
          return t_FLOAT; } }
{inumb} { if( ncomment==0 ) { yylval.ival = atoi(yytext);
          return t_INT; } }
{skip}  { }
…
```

Пример: описание синтаксиса TLT

```
…
%token <text> t_STRING
%type <pval> ivalue fvalue bvalue numb prnoper value
%type <prog> oper prn fill mark ret block assign
%type <pbl>  bstart
%right '='
%nonassoc '?'
%nonassoc ':'
%left t_AND t_OR
%left '!'
%nonassoc t_LE '<' '>'
%left '+' '-'
%left '*' '/'
%left '%'
%nonassoc t_UMINUS t_U
%nonassoc '[' ']'
```

```
err :     t_ERROR
preop :   book  t_EOL
          | debug t_EOL
          | defn  t_EOL
book :    t_BOOK t_INT t_STRING t_INT numb numb
          | t_BOOKPR t_INT t_STRING t_INT numb numb
          | t_BOOK2 t_INT t_STRING t_INT numb numb t_INT numb numb
oper :    t_EOL
          | prn  t_EOL
          | assign t_EOL
          | fill t_EOL
          | mark t_EOL
          | ret  t_EOL
          | block
          | t_IF '(' bvalue ')' oper
          | t_IF '(' bvalue ')' oper t_ELSE oper
bstart :  '{' oper
          | bstart oper
block :   bstart '}'
…
```

Объектно-ориентированное программирование

Основные понятия, методики разработки программного обеспечения, паттерны программирования

Сложные системы

Современным программным комплексам присуща как композиционная, так и функциональная сложность.

Автоматизация эксперимента	Автоматизация бизнеса
• Множество разнородных, но связанных между собой подсистем • Большие объемы и потоки данных • Надежность, отказоустойчивость • Учет особых требований эксперимента	• Множество разнородных, по переплетенных, задач и ролей: управление, финансы,… • Необходимость интеграции с внешними системами, с предыдущими версиями • Учет особых требований законодательства • Гибкость

Способ победить сложность: декомпозиция. Множество различных методик:

- Структурное проектирование
- Проектирование на основе потоков данных
- Объектно-ориентированное проектирование

Ключевые понятия объектно-ориентированного подхода

Ключевую роль в объектно-ориентированном подходе играет понятие объекта
(а не процедура).

Объект – основная структурная единица программной системы. Как правило, представляет собой некоторый блок данных и набор операций, который с ними можно делать.

Программная система представляет собой множество объектов, которые обмениваются между собой сообщениями.

Каждый объект обладает:

- индивидуальностью
- состоянием
- поведением.

Схожие объекты объединяются в классы.

Объектно-ориентированные языки должны предоставлять возможность:

- инкапсуляции;
- наследования;
- полиморфизма.

C++

Первым объектно-ориентированным языком считается Simula (1967). В нем было предложено очень много понятий ООП.

Огромное влияние на все ОО языки оказал язык Smalltalk, который развивался в течение 70-х годов. В нем наиболее полно оказалась реализована концепция «все объект».

Язык C++ был разработан в течении 80-х годов Бьерном Страуструпом.

Основные принципы, на базе которых разрабатывался язык:

•язык должен быть универсальным, не привязанный к какой то платформе или задаче;

•максимально возможная совместимость с языком C;

•полноценная поддержка процедурного и объектно-ориентированного программирования (и, позднее, обобщенного программирования);

•возможность эффективной реализации (т.е. программы, написанные на C++, должны компилироваться в эффективный машинный код).

На сегодняшний день C++ является одним из наиболее распространенных языков программирования. На нем написаны ROOT и Geant4.

Классы и объекты в C++

Я предполагаю, что C++ вам хорошо знаком. Очень краткое напоминание:

```
class Particle  {

private:
  double mass;

public:
  Particle(double);
  ~Particle();

  double mass();
  void mass(double);

  virtual bool isHadron();
  virtual void printProperties() = 0;
};
```

Класс = описание структуры + функции работы с ней

Объект = конкретный экземпляр класса

Инкапсуляция

Полиморфизм

Виртуальные функции

Наследование в C++

```
class ChargedParticle: public Particle  {
private:
  int charge;
public:
  virtual void printProperties() { … }
};

class HadronParticle: public Particle  {
private:
  std::string quarkComposition;
… }

class Proton: public HadronParticle, Charged Particle
{ … }
```

Множественное наследование часто вызывает проблемы и не поддерживается во многих ОО языках.

Безопасное решение – абстрактные классы (интерфейсы)

Исключения

C++ предоставляет механизм исключений, значительно упрощающий обработку нестандартных ситуаций по сравнению с традиционной проверкой кода возврата.
Исключения не являются средством объектно-ориентированного программирования. Большинство современных языков программирования включают аналогичные механизмы обработки исключений.

```
try {
    // do something
    throw 1;
    // throw “a”;
} catch (long b) {
    cout << "пойман тип long: " << b << endl;
} catch (string b) {
    cout << "пойман тип string: " << b << endl;
} catch (...) {
    cout << "поймано что то другое << endl;
}
```

Шаблоны

C++ предоставляет средства обобщенного программирования – шаблоны.

Шаблон класса:

```
template< class T > class List
{
public:
   void Add( const T& Element );
   bool Find( const T& Element );
…
};
List<Proton> lp;
```

Шаблон функции:

```
template< typename T > T slowest( T a, T b ) {
   return a.momentum() < b.momentum() ? a : b;
}
slowest(proton1,proton2);
```

STL

Важной частью современного C++ является стандартная библиотека шаблонов (STL). Основные компоненты STL:

Контейнеры	pair, vector, list, set, multiset, map, multimap, queue, priority_queue, stack, bitset, valarray,…
Итераторы	механизм перебора элементов в контейнере
Алгоритмы	обобщенные алгоритмы, которые можно применить к элементам контейнера (sort, search,…)
Функторы	Объекты, содержащие в себе функции (используются алгоритмами) (plus, minus, less, greater, …)

```
vector<Particle> p;
struct lighter : public binary_function<double, double, bool> {
  bool operator()(Particle x, Particle y) {
    return x.mass() < y.mass();
} };
sort(p.begin(), p.end(), lighter());
```

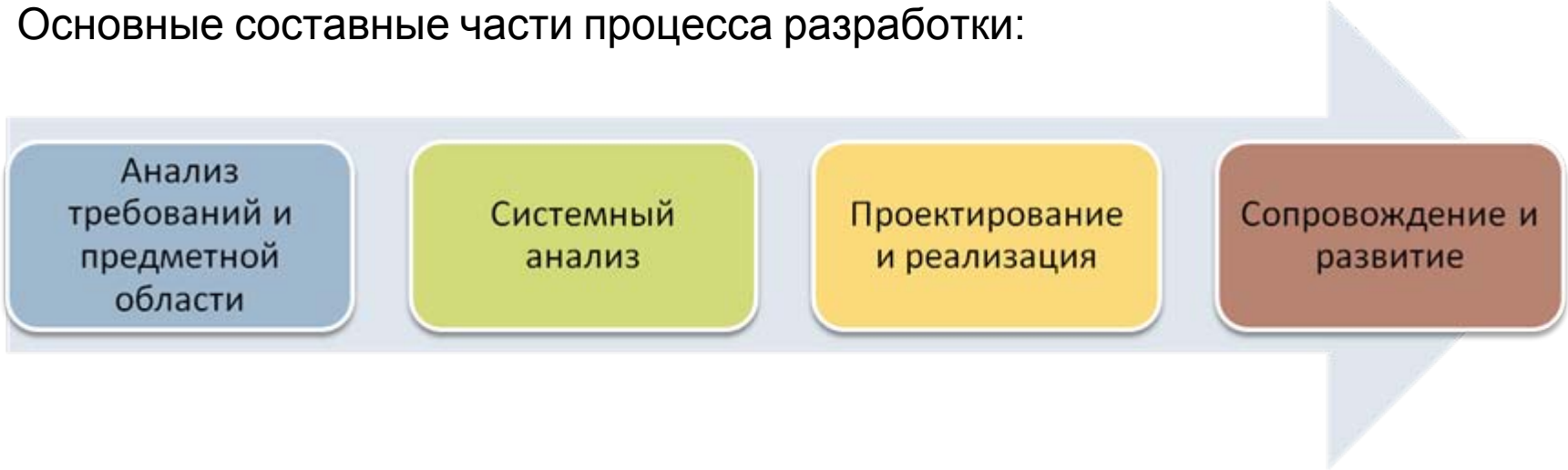
Процесс разработки ПО

Как правило, большой программный комплекс

- сложен
- разрабатывается командой программистов и других специалистов.

Поэтому разработка сложного программного комплекса требует специальной организации.

Основные составные части процесса разработки:



Существует множество моделей и методологий, которые отличаются тем, на какие подзадачи разбивается весь процесс разработки, содержанием подзадач, последовательностью выполнения подзадач,... Эти методологии можно использовать не только для разработки ПО, но и выполнении любого проекта.

Основные модели разработки

Каскадная модель (waterfall)

Последовательное выполнение всех стадий

Итеративная модель

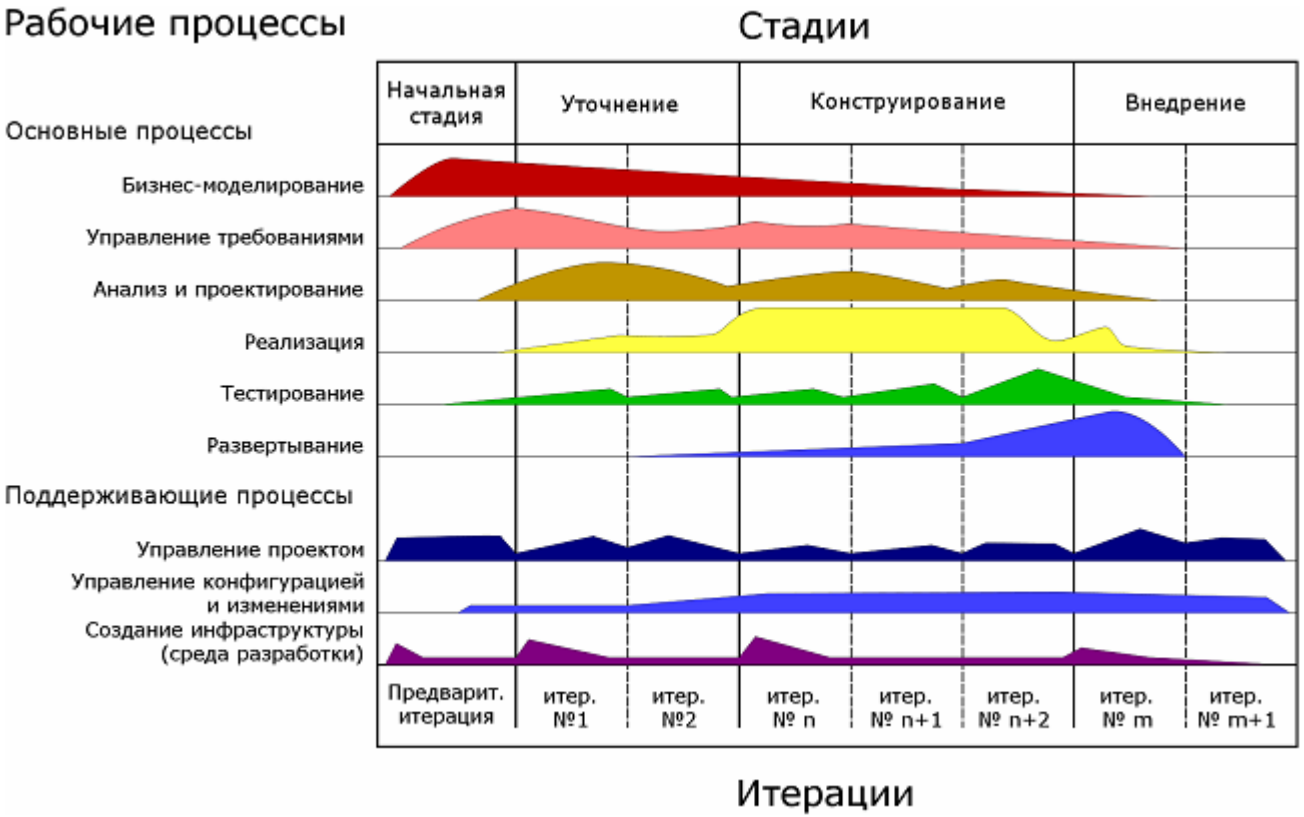
Выполнение стадий «по кругу», с тестированием и оценкой промежуточных результатов и соответствующей модификацией проекта и реализации.

Инкрементная, спиральная модель.

На каждой итерации наращивается функционал.

Rational Unified Process

Одной из самых известных методологий разработки ПО стала итеративно-инкрементная методология Rational Unified Process (RUP).



RUP довольно «тяжеловесная» и редко применяется при разработке ПО в ФВЭ. Однако для разработки Geant4 использовалась методология – предшественник RUP.

Agile Software Development

Для разработки программного обеспечения экспериментов органичнее использовать гибкие (agile) методологии.

Основные принципы agile software development («Agile Manifesto»):

1. Люди и взаимодействие важнее процессов и инструментов
2. Работающий продукт важнее исчерпывающей документации
3. Сотрудничество с заказчиком важнее согласования условий контракта
4. Готовность к изменениям важнее следования первоначальному плану

Такие методологии характеризуются относительно небольшим объемом документации, короткими итерациями, развитой инфраструктурой тестирования, постоянным взаимодействием разработчиков и «заказчика» (в эксперименте часто это одни и те же люди).

При использовании agile-подходов быстро создается работающий, но ограниченный по возможностям, программный продукт, которое потом непрерывно дорабатывается, при этом оставаясь все время функциональным.

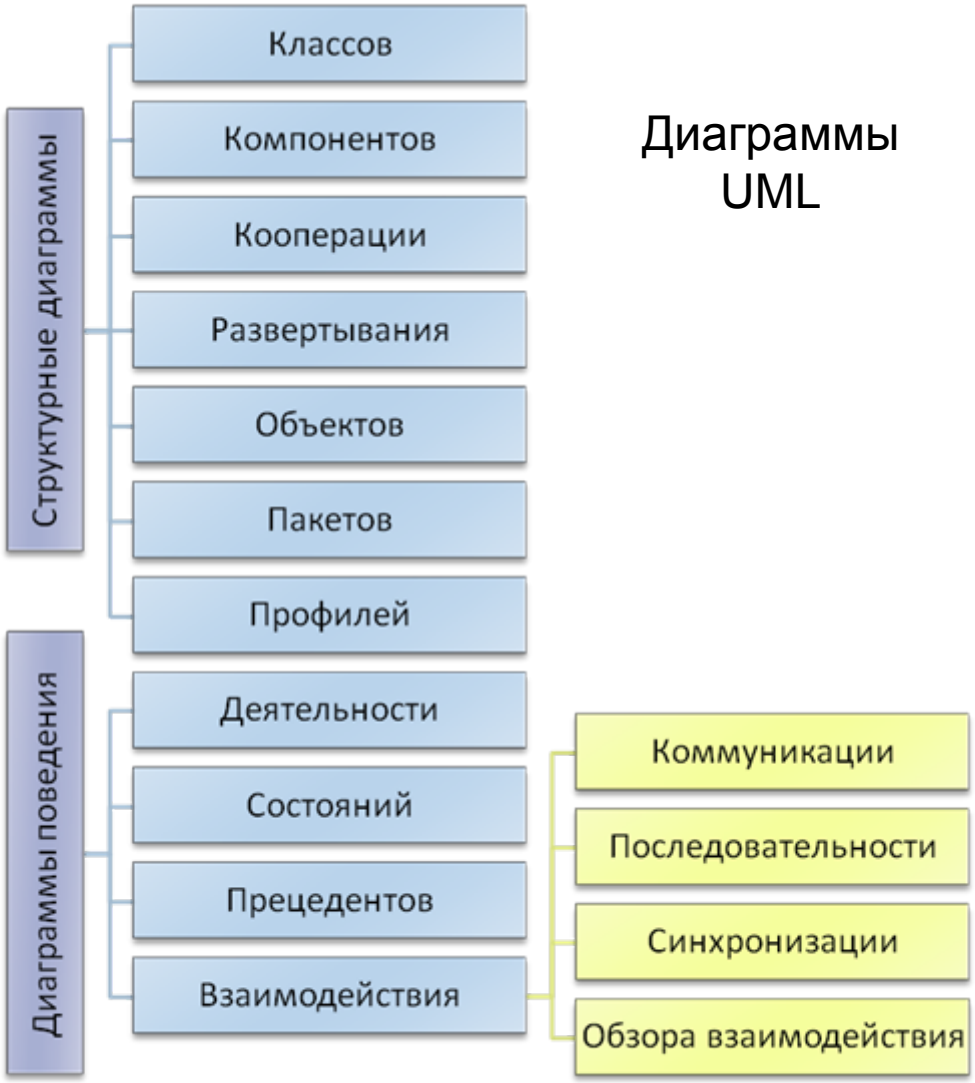
Наиболее известные agile-методологии – Extreme Programming (XP), Scrum, DSDM.

UML

Для описания объектной модели системы используется графический язык UML (Unified Modeling Language). Разработка UML была тесно связана с разработкой RUP, но область применения UML значительно шире.

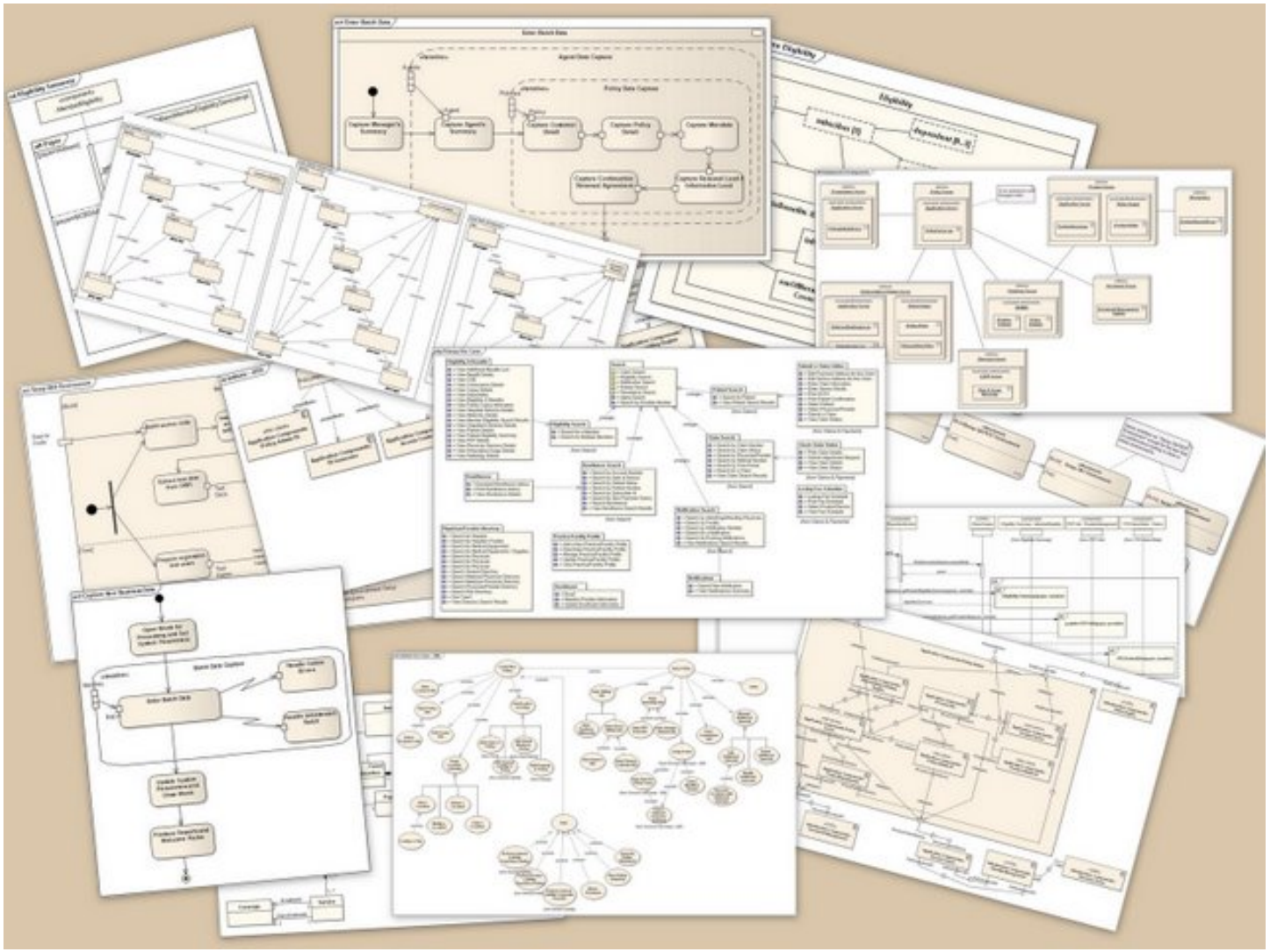
В UML система описывается с помощью различных диаграмм. С помощью специальной нотации на диаграммах показаны составные элементы системы, их связь между собой, механизмы взаимодействия, жизненный цикл системы,...

Существуют системы разработки, которые по UML-диаграмме генерируют соответствующий программный



Диаграммы UML

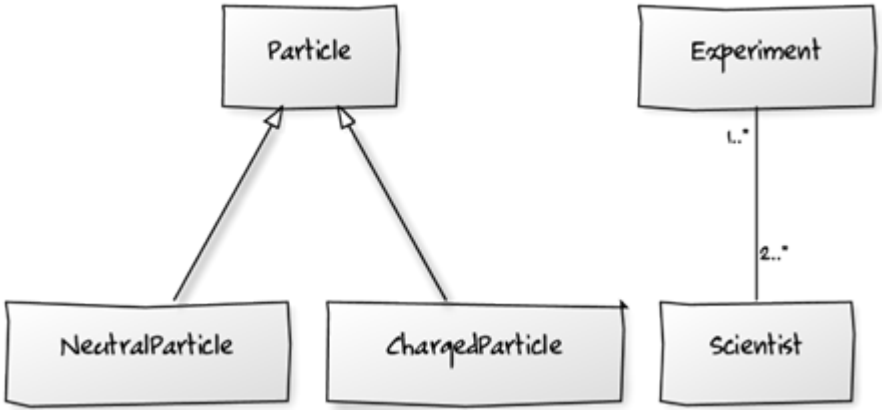
Примеры диаграмм UML



Примеры диаграмм UML

Наследование

Ассоциация



Объединение

Агрегация

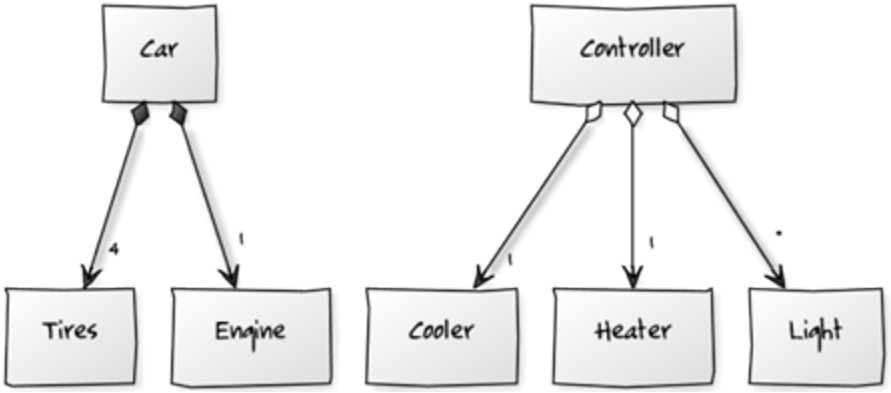


Диаграмма классов

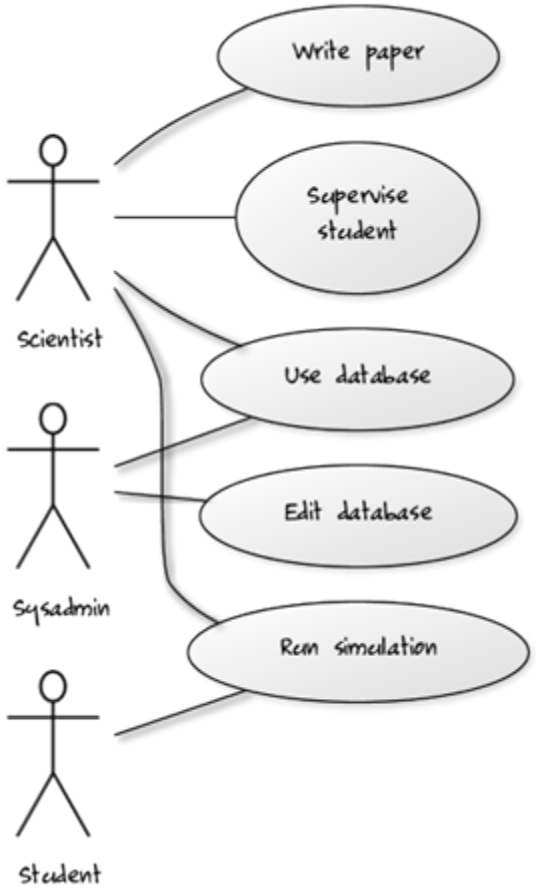
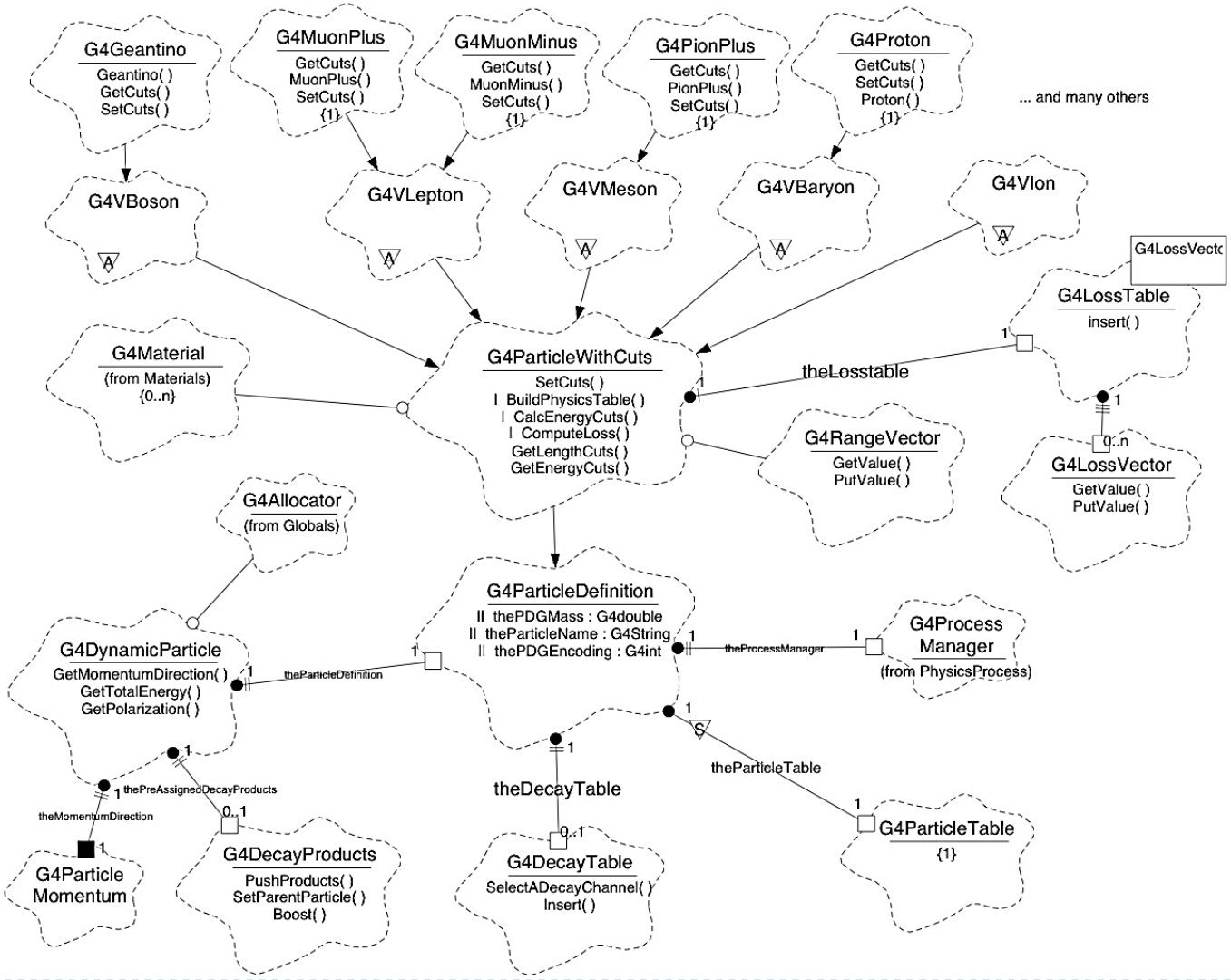


Диаграмма прецедентов

Диаграмма классов Geant4



Принципы проектирования

Опыт промышленного программирования помог сформулировать ряд принципов проектирования:

1. Выделите аспекты приложения, которые могут изменяться, и отделите их от тех, которые всегда остаются постоянными.
2. Программируйте на уровне интерфейсов, а не реализации.
3. Отдавайте предпочтение композиции, а не наследованию.
4. Стремитесь к слабой связанности взаимодействующих объектов.
5. Классы должны быть открыты для расширения, но закрыты для изменения.
6. Код должен зависеть от абстракций, а не от конкретных классов.
7. Принцип минимальной информированности: общайтесь только с близкими друзьями.
8. Класс должен иметь только одну причину для изменения.

и множество других…

Паттерны (шаблоны) программирования

Многолетний мировой опыт создания программного обеспечения позволил выделить типичные архитектурные конструкции – паттерны (шаблоны).

Паттерн — решение задачи в контексте.

Контекстом называется ситуация, в которой применяется паттерн. Ситуация должна быть достаточно типичной и распространенной.

Задачей называется цель, которой вы хотите добиться в контексте, в совокупности со всеми ограничениями, присущими контексту.

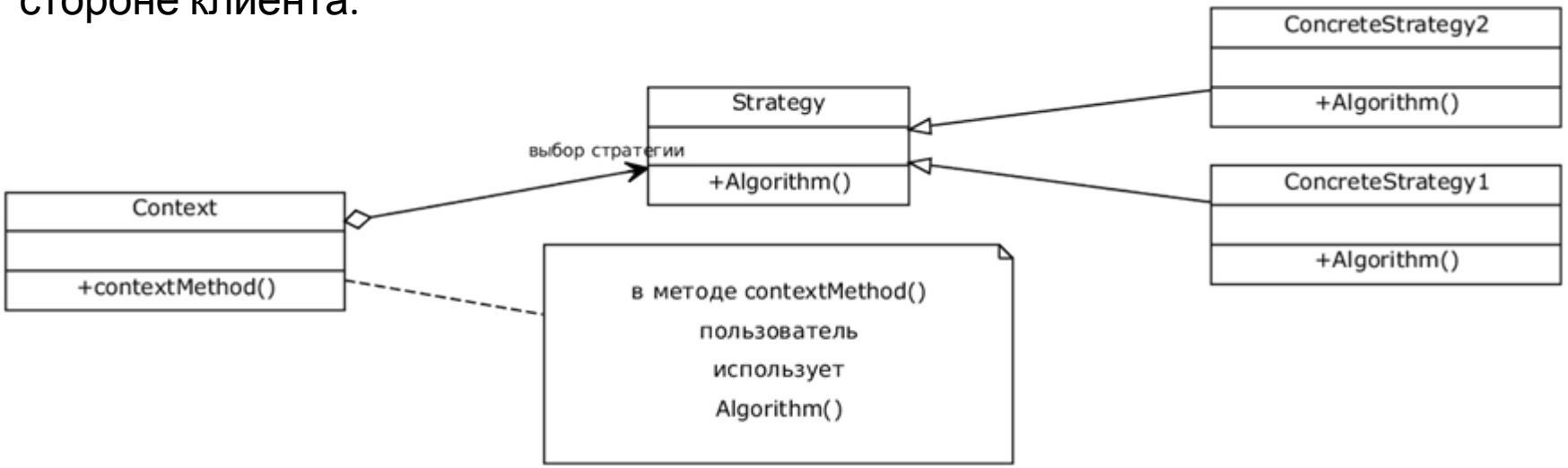
Решением называется обобщенная архитектура, которая достигает заданной цели при соблюдении набора ограничений.

Наиболее распространенные паттерны

Порождающие
Одиночка, Строитель, Фабрика, Абстрактная фабрика,…
Структурные
Заместитель, Декоратор, Компоновщик, Адаптер, Мост,…
Поведенческие
Итератор, Стратегия, Шаблонный метод, Команда, Наблюдатель, Посредник,…

Паттерн Стратегия (Strategy)

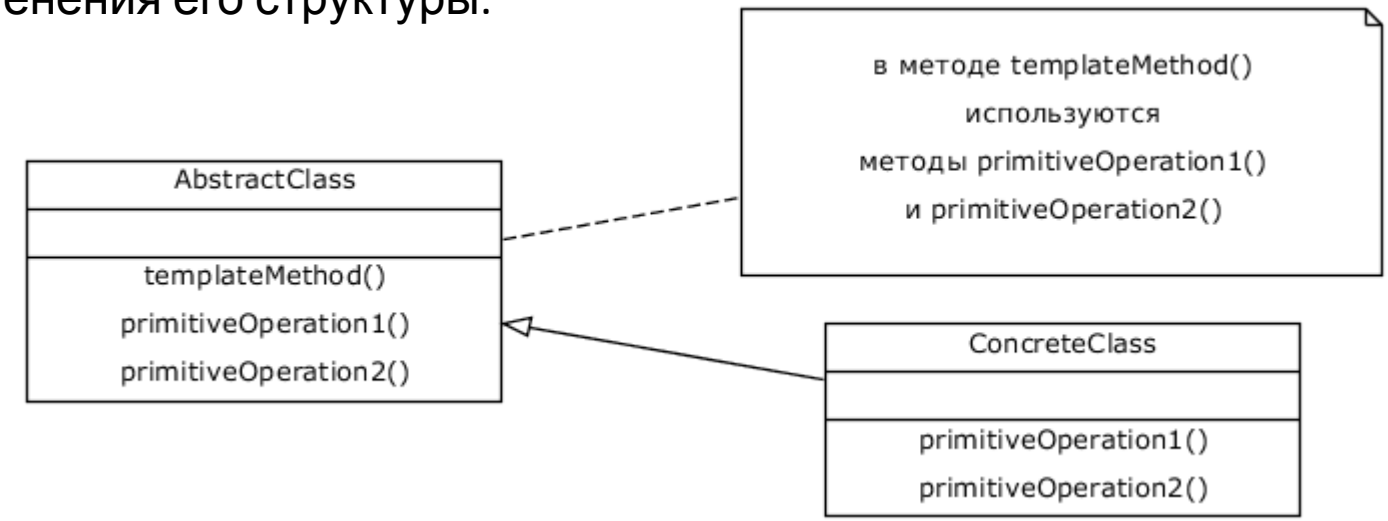
Паттерн Стратегия определяет семейство алгоритмов, инкапсулирует каждый из них и обеспечивает их взаимозаменяемость. Он позволяет модифицировать алгоритмы независимо от их использования на стороне клиента.



Пример: для заряженной частицы (Context) требуется вычислить ее движение в электромагнитном поле (concreteMethod). Для этого нужно проинтегрировать уравнения движения, что можно сделать разными методами (Algorithm). Применение паттерна Стратегия позволит гибко выбирать конкретный алгоритм численного интегрирования.

Паттерн Шаблонный Метод (Template Method)

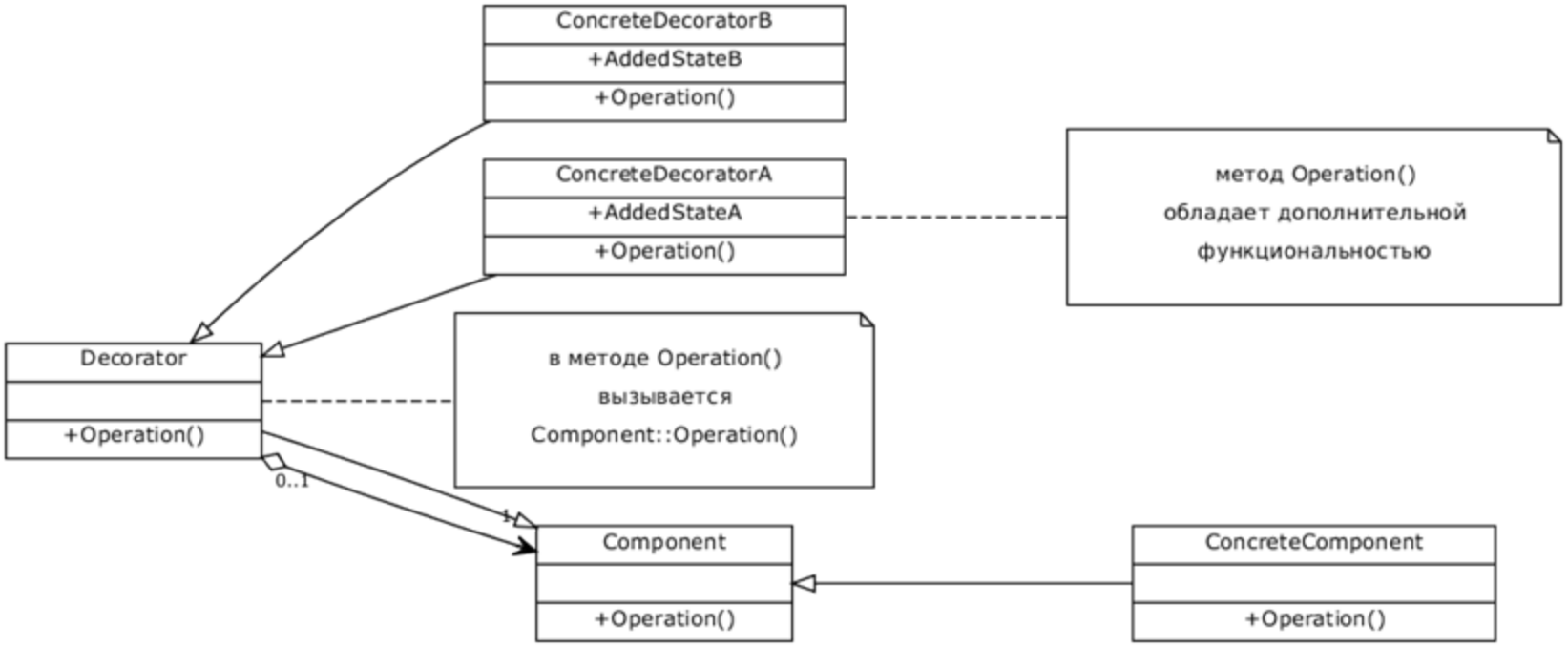
Паттерн Шаблонный Метод задает «скелет» алгоритма в методе, оставляя определение реализации некоторых шагов субклассам. Субклассы могут переопределять некоторые части алгоритма без изменения его структуры.



Пример: любой класс пользователя, в котором производится анализ данных (templateMethod), должен открыть файл, набрать гистограммы (primitiveOperation1) и закрыть файл. В templateMethod() выполняются все эти шаги, но конкретная реализация набора гистограмм производится в пользовательском классе-наследнике.

Паттерн Декоратор (Wrapper)

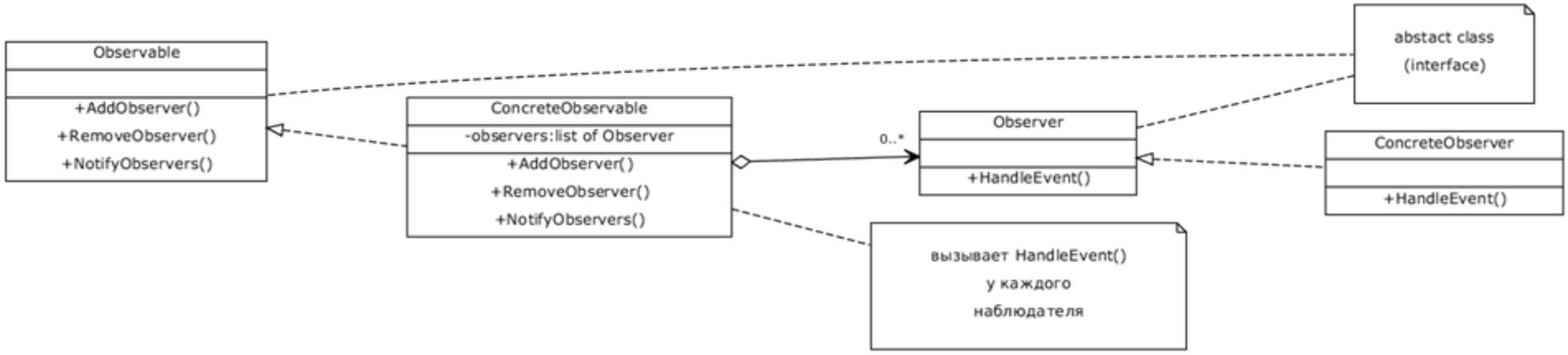
Паттерн Декоратор динамически наделяет объект новыми возможностями и является гибкой альтернативой созданию классов-наследников для расширения функциональности.



Пример: требуется журналировать все обращения к определенным методам библиотечного класса (Component). Решение: создаем декоратор для этого класса и добавляем журналирование перед вызовом нужных методов.

Паттерн Наблюдатель (Observer)

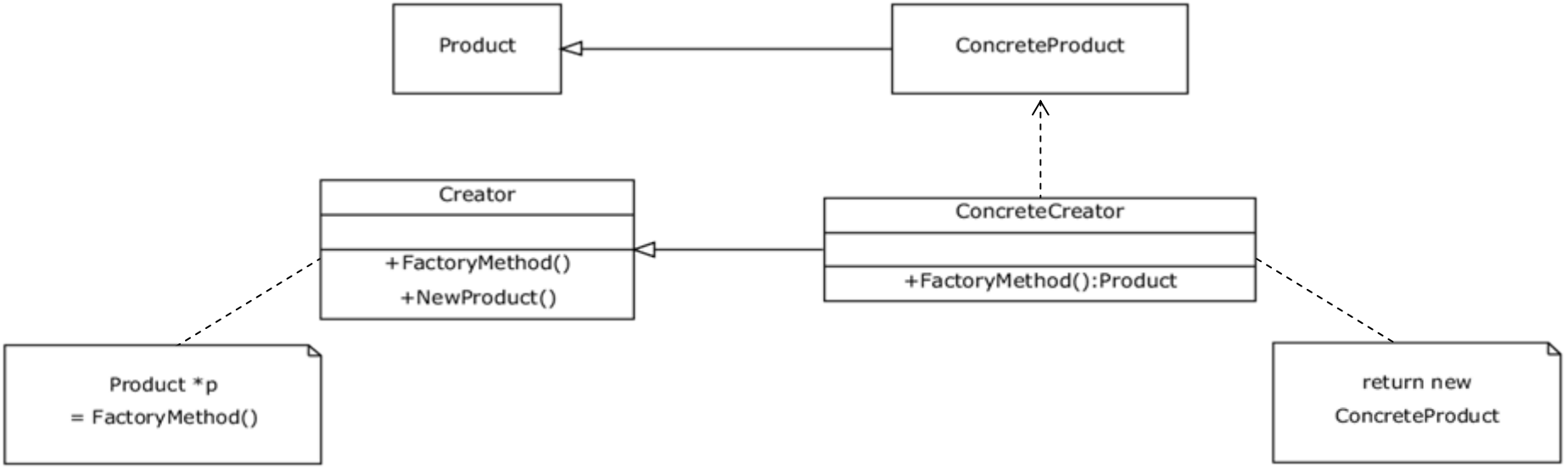
Паттерн Наблюдатель определяет отношение «один-ко-многим» между объектами таким образом, что при изменении состояния одного объекта происходит автоматическое оповещение и обновление всех зависимых объектов.



Пример: требуется, чтобы в программе обработки (Observable) при появлении событий определенного типа (например, начало захода) все модули обработки (Observer) выполняли определенные действия. Другой пример: реакция на нажатие кнопки в пользовательском интерфейсе.

Паттерн Фабричный Метод (Factory Method)

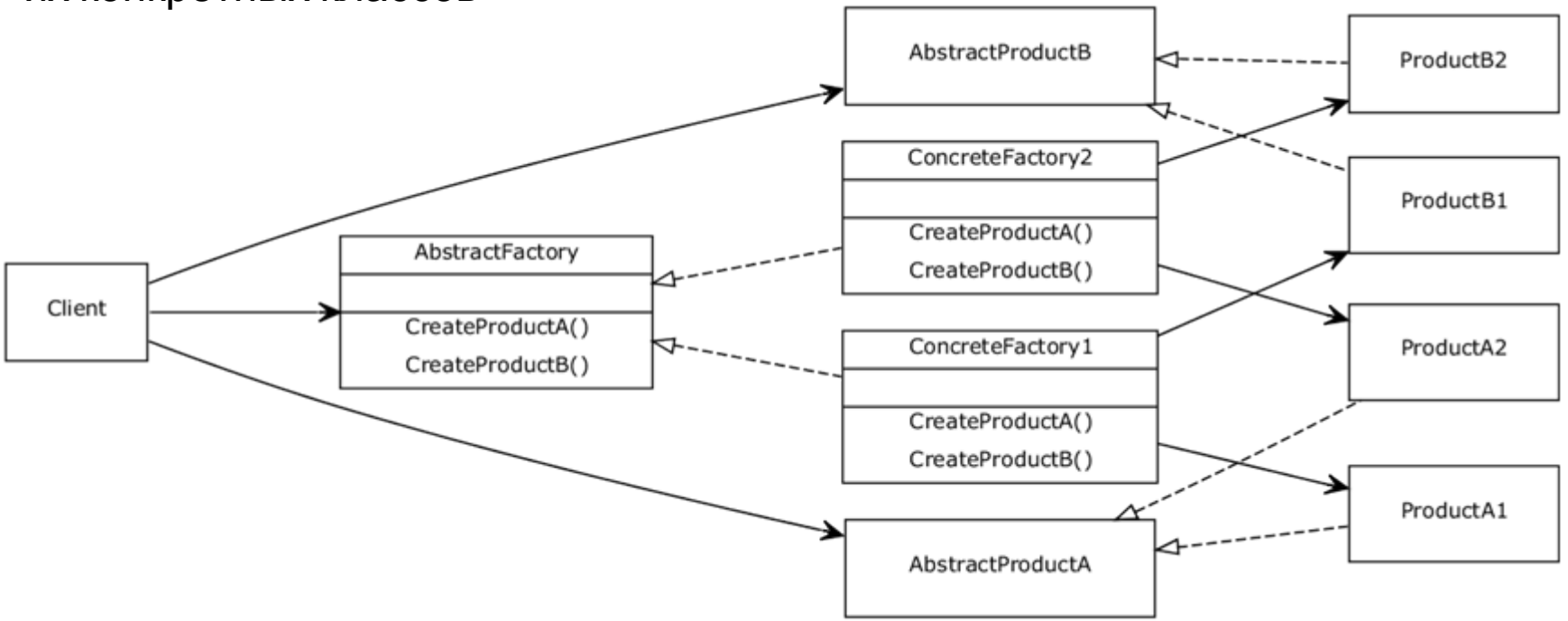
Паттерн Фабричный Метод определяет интерфейс создания объекта, но позволяет субклассам выбрать класс создаваемого экземпляра. Таким образом, Фабричный Метод делегирует операцию создания экземпляра субклассам.



Пример: требуется, чтобы программа обработки могла открывать файлы (Product) как на диске, так и через веб-сервис (варианты ConcreteProduct). В программе обработки для открытия файлов используем фабрику (Creator), и создаем две реализации этой фабрики (ConcreteCreator).

Паттерн Абстрактная Фабрика (Abstract Factory)

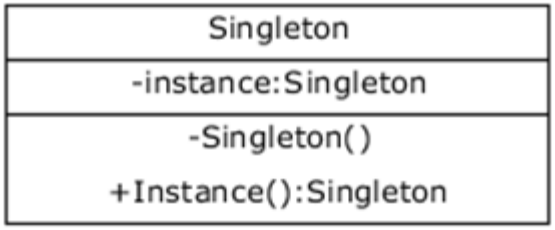
Паттерн Абстрактная Фабрика предоставляет интерфейс создания семейств взаимосвязанных или взаимозависимых объектов без указания их конкретных классов



Пример: для вычисления трека заряженной частицы нужно описать магнитное поле, уравнение движения, интегратор. Все это создаем через интерфейс абстрактной фабрики.

Паттерн Одиночка (Singleton)

Паттерн Одиночка гарантирует, что класс имеет только один экземпляр, и предоставляет глобальную точку доступа к этому экземпляру.

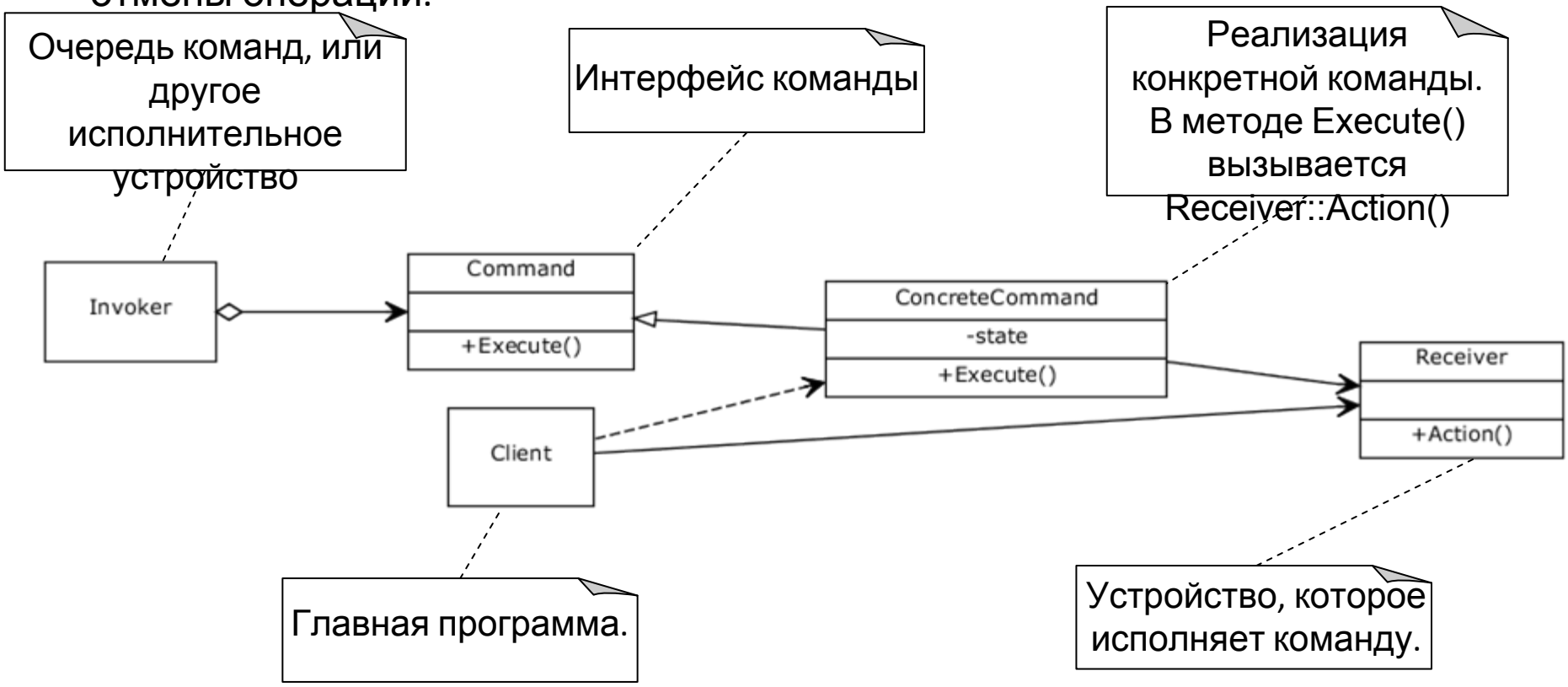



```
public class Singleton {

   private:
   static Singleton uniqueInstance;
   Singleton() {}

   public:
   static Singleton getInstance() {
     if (uniqueInstance == null) {
        uniqueInstance = new Singleton();
     }
     return uniqueInstance;
   }
}
```

Пример: таблица со свойствами всех частиц, или глобальный счетчик использования ресурсов.

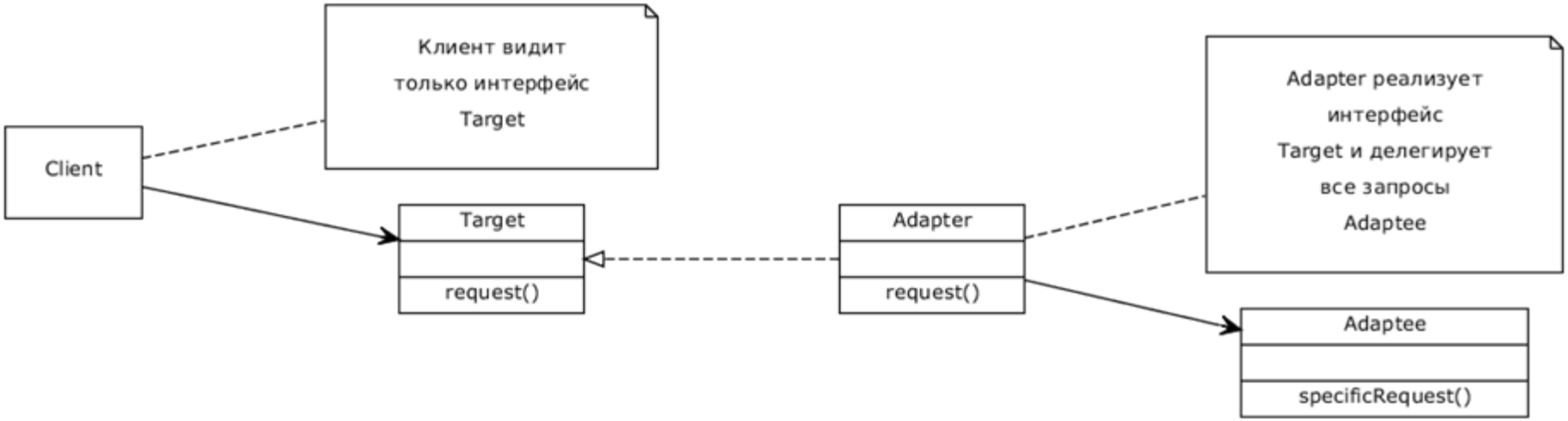
Паттерн Команда (Command)

Паттерн Команда инкапсулирует запрос в виде объекта, делая возможной параметризацию клиентских объектов с другими запросами, организацию очереди или регистрацию запросов, а также поддержку отмены операций.



Паттерн Адаптер (Adapter)

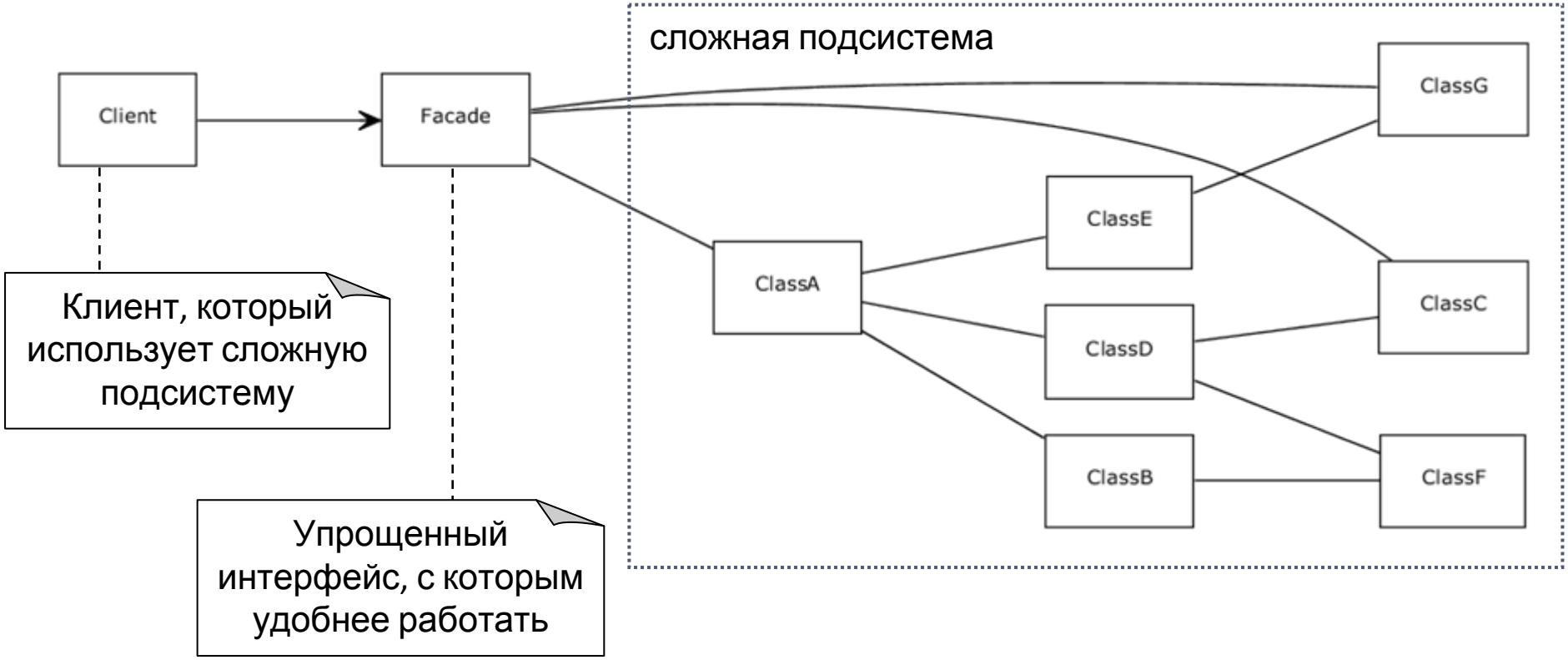
Паттерн Адаптер преобразует интерфейс класса к другому интерфейсу, на который рассчитан клиент. Адаптер обеспечивает совместную работу классов, невозможную в обычных условиях из-за несовместимости интерфейсов.



Пример: требуется написать модуль первичного генератора для программы моделирования, написанной на C++, в которой использовать библиотеку, написанную на FORTRAN. Создаем адаптер для библиотеки и используем его.

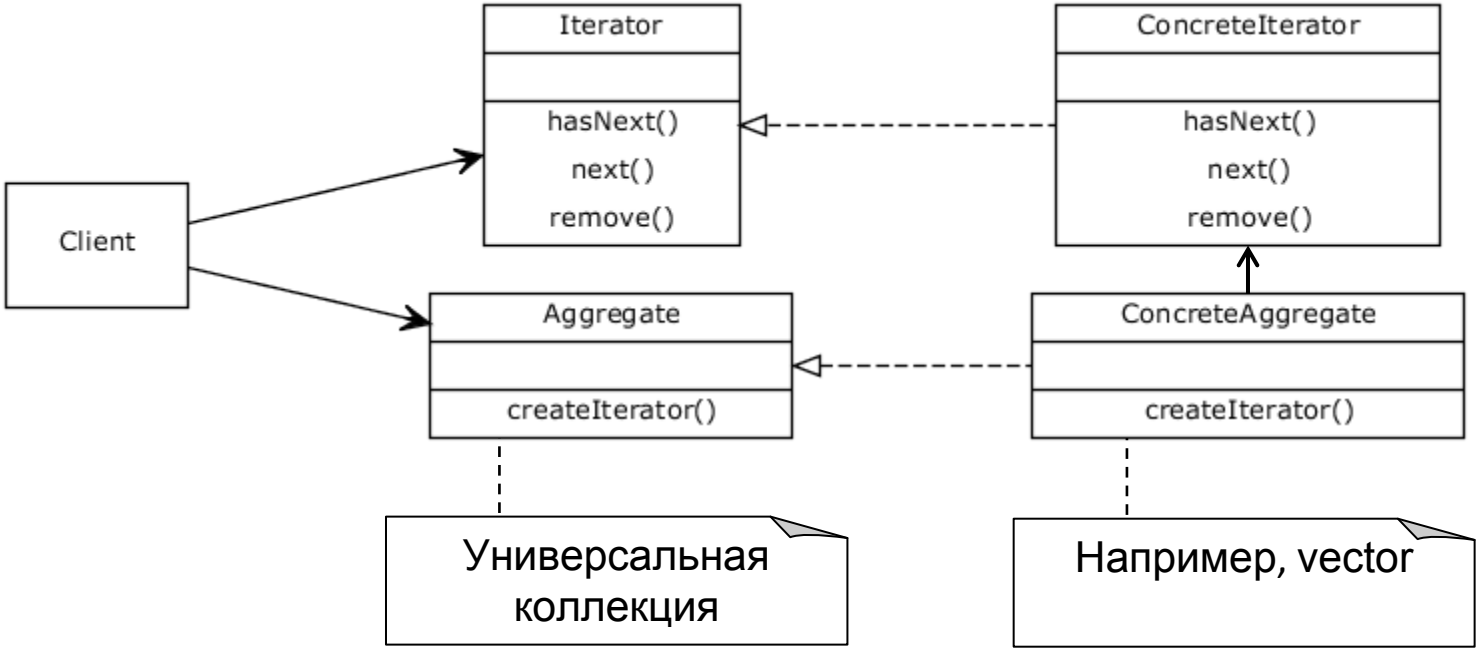
Паттерн Фасад (Facade)

Паттерн Фасад предоставляет унифицированный интерфейс к группе интерфейсов подсистемы. Фасад определяет высокоуровневый интерфейс, упрощающий работу с подсистемой.



Паттерн Итератор (Iterator)

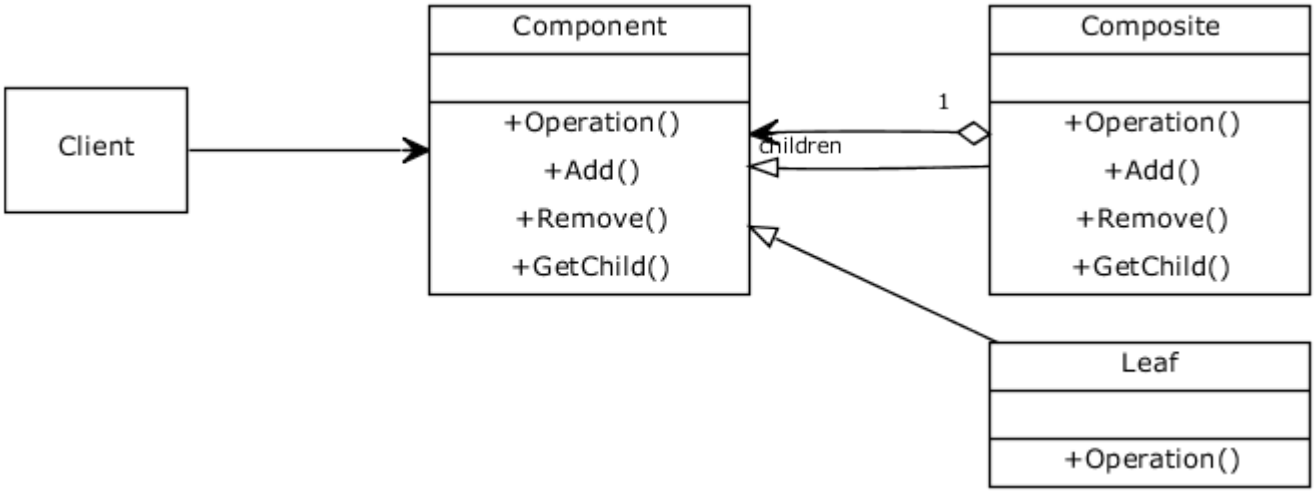
Паттерн Итератор предоставляет механизм последовательного перебора элементов коллекции без раскрытия ее внутреннего представления.




```
for(list<int>::iterator  it=mylist.begin(); it!=mylist.end(); it++)
        cout << " " << *it;
```

Паттерн Компоновщик (Composite)

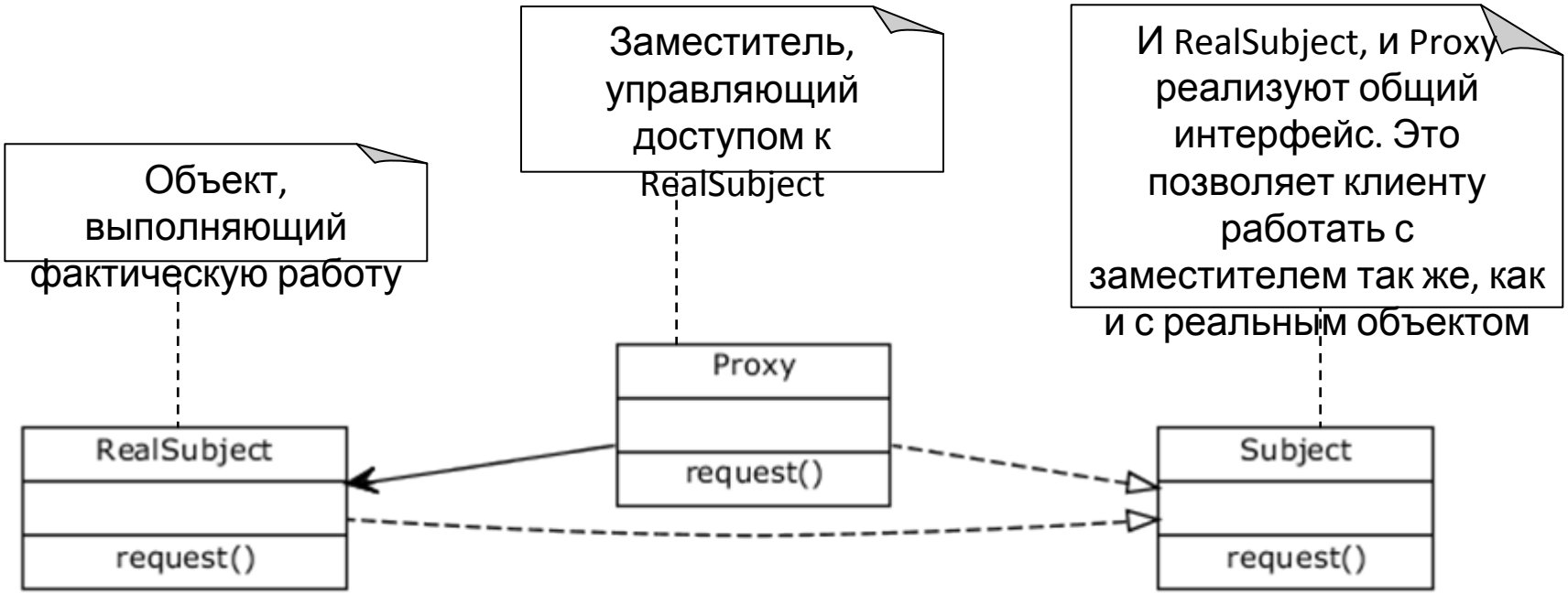
Паттерн Компоновщик объединяет объекты в древовидные структуры для представления иерархий «часть/целое». Компоновщик позволяет клиенту выполнять однородные операции с отдельными объектами и их совокупностями.



Пример: в программе требуется описать карту магнитного поля, которая может состоять из нескольких подкарт. Используем для этого паттерн Компоновщик, метод Operation() – это функция получения значения поля в точке, Composite – составная карта, Leaf – несоставная карта.

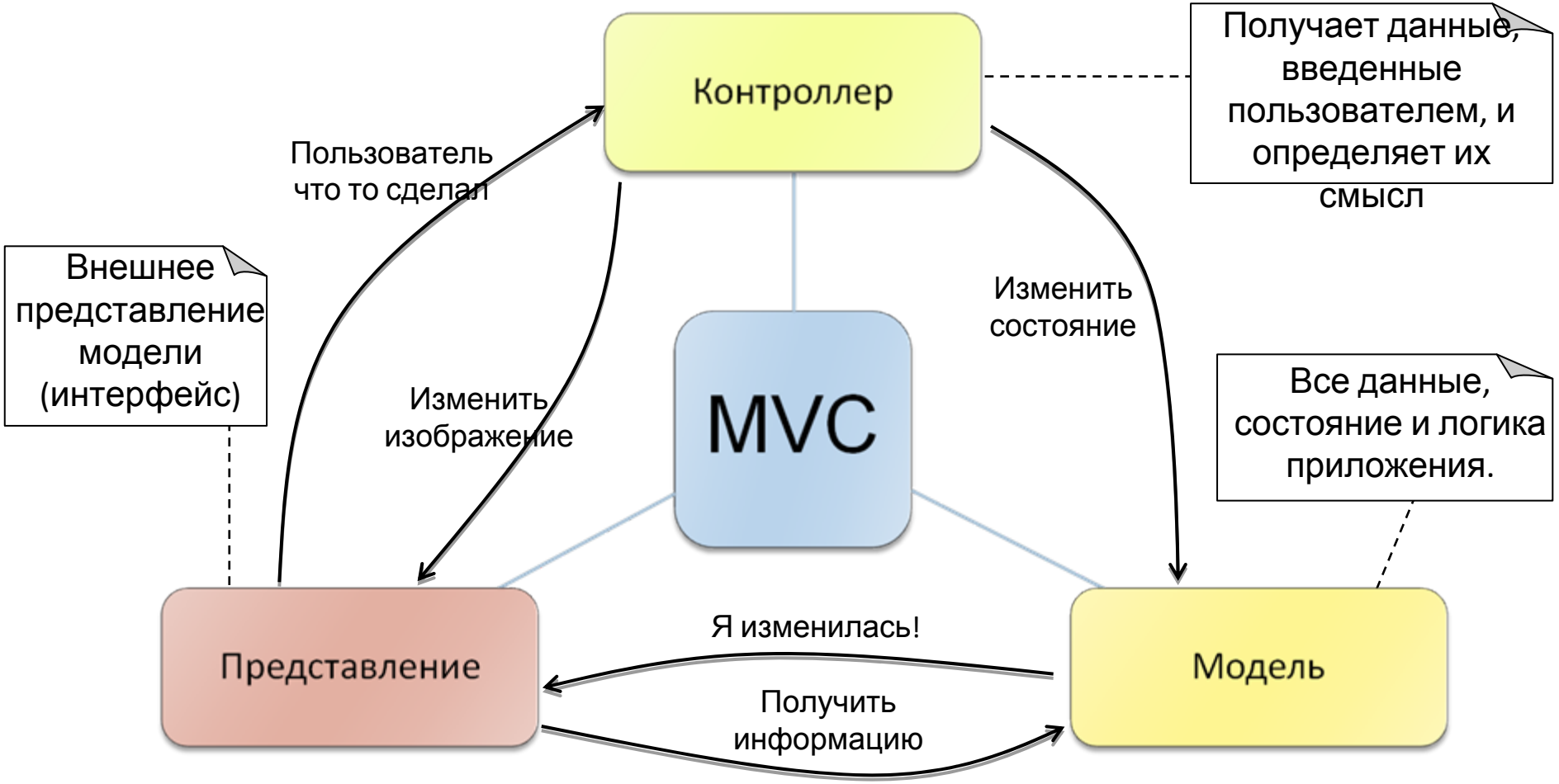
Паттерн Заместитель (Proxy)

Паттерн Заместитель предоставляет суррогатный объект, управляющий доступом к другому объекту.



Пример: часто используется для доступа к аппаратуре, базам данных, при необходимости организовать доступ с кэшированием или с ограничениями.

Составной паттерн Model-View-Controller



Пример: самый распространенный паттерн для разработки интерфейсов, веб-приложений и т.п.