МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

УНИВЕРСИТЕТ ИМЕНИ СУЛЕЙМАНА ДЕМИРЕЛЯ

ИНЖЕНЕРНЫЙ ФАКУЛЬТЕТ

SULEYMAN DEMIREL UNIVERSITY

Кафедра «Компьютерная инженерия»

УДК 374.31

АЙТАСОВ РАСУЛ

ОБУЧЕНИЕ ОСНОВАМ ПРОГРАММИРОВАНИЯ СТУДЕНТОВ, ОБУЧАЮЩИХСЯ ПО СПЕЦИАЛЬНОСТИ “МАТЕМАТИКА”

Магистерская диссертация, представленная на соискание академической степени магистра технических наук по специальности 6М070400 – «Вычислительная техника и программное обеспечение»

Каскелен, 2013

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

УНИВЕРСИТЕТ ИМЕНИ СУЛЕЙМАНА ДЕМИРЕЛЯ

ИНЖЕНЕРНЫЙ ФАКУЛЬТЕТ

SULEYMAN DEMIREL UNIVERSITY

Кафедра «Компьютерная инженерия»

«Допущен к защите»
Заведующий кафедрой
д.т.н., профессор Амиргалиев Е.Н.
_____
«___»_____2013 г.

Заведующий департаментом
послевузовского образования
к.ф.н., Айдогду Шабан
_____
«___»_____2013 г.

Магистерская диссертация

ОБУЧЕНИЕ ОСНОВАМ ПРОГРАММИРОВАНИЯ СТУДЕНТОВ, ОБУЧАЮЩИХСЯ ПО СПЕЦИАЛЬНОСТИ “МАТЕМАТИКА”

Специальность: 6М070400 – «Вычислительная техника и программное обеспечение»

Магистрант _____ Айтасов Расул

Научный руководитель _____ Амиргалиев Е.Н.
д.т.н., профессор

Каскелен, 2013

РЕЗЮМЕ

Данная исследовательская работа заключает в себе разбор языков програмирования и выбор наиболее оптимального силабуса дальнейшего обучения студентов на специальности «Математика» инженерного факультета университета имени Сулеймана Демиреля. В докладе предлагается своеобразное новое направление курса «Языки програмирования для математиков», а именно выбран подходящий формат обучения програмированию с точки зрения математического мышления.

ABSTRACT

This research includes the analysis of programming languages and choice of the most optimal syllabus for further education of students of specialty "Mathematics", faculty of Engineering, Suleyman Demirel University. The report proposes a кind of a new direction of the course "Programming Languages for mathematicians," namely, to choose the appropriate format of education of programming from the point of view of mathematical thinкing.

ТҮЙІН

Айтылмыш зертте-жұмыс бағдарламалық тілді талдайды және Сулейман Демирел атындағы университетінің инженерлік факультетінің "математика" мамандығында оқитын студенттерге сол тілді үйрету үшін ең үйлесімді силабусты таңдайды. Баяндамада "Математиктерге арналған бағдарламалық тілдер" курсының жаңа бағыты ұсынылады, ал тап математикалық ойдың тұрғысынан тәлім-тәрбиесінің лайық форматы таңдалған.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Во-первых я хочу объяснить почему я выбрал именно эту тему. В период моего обучения на бакалвриате по специальности вычислительная техника и програмное обеспечение, со мной вместе учились студенты из нашего же факультета, но только по другой специальности – математики. Так как, мы были будущими програмистами, то у нас был сильный уклон в изучение языков програмирования, а именно язык Java. Но что было интересно математикам также преподавался урок програмирования (Java), по той же программе, что и нам. То есть, мы будущие програмисты и математики, которые в будущем может даже и не будут програмировать, проходили этот предмет на одном уровне. Я хочу обратить внимание на тот момент, что даже нам с трудом давался этот предмет, а математики без особой практики и начального фундамента вообще многое не понимали. Продолжается это и по сей день. А происходит это из-за того что математикам должен объясняться этот курс совсем по другой программе, то есть так как им это будет понятно, то е́сть чисто математически. Ввиду всего этого я заметил, что эта проблема остается все также нерешенной на нашем факультете, поэтому я и решил посвятить свою дипломную работу разрешению данной проблемы. В своей диссертации я провел небольшой анализ многих языков програмирования для определения наиболее подходящих для специальности “Математика”, выбрал три из них, и детально рассмотрел каждый язык, сопоставляя их преимущества и недочеты.

1 ЯЗЫКИ ПРОГРАММИРОВАНИЯ В МАТЕМАТИКЕ

Программирование — это не математика, программирование — это дизайн. Задача математика состоит в том чтобы найти наиболее простую и универсальную формулу или систему аксиом, описывающую некоторое явление. Программисты занимаются построением полезных систем, которые в большинстве своем сложны настолько, что даже их создатели с трудом справляются с пониманием собственных детищ. Задача программиста — создать работающую вещь; борьба со сложностью и стремление к простоте это всего лишь побочное следствие ограниченности его умственных ресурсов. Ограниченность же машинных ресурсов играет с программистом злую шутку и добавляет ему работы, которая как правило приводит к усложнению конечного результата. Из естественных наук программированию ближе всего биология. У человеческого организма оптимальный дизайн? Почти. Он идельно работает? Более-менее. Он сложен? Запредельно. И тем не менее, при всей своей запредельной сложности организм, по большей части — невербятное нагромождение груд мусора с огромнейшим, просто необъятным прицепом хлама для «обратной совместимости», накопившемся за миллиарды лет. Даже элементарные, математически точно сформулированные задачи можно превратить в сто разных программ. Когда дело доходит до реальных вещей математическая точность отходит даже не на второй план — в реальности она вообще никому не нужна. Даже если система полностью хаотична — главное, что она более-менее нормально работает. Не нужно относиться к программированию и написанию программы как к построению аксиоматической системы. Достаточно лишь чтобы каждый новый кусок кода хорошо (но не идеально!) подходил к уже существующим, и не приводил к разрушению программы сейчас или в обозримом будущем. С другой стороны, к дизайну языков программирования нужен математический подход. Чем ниже уровень языка, тем более строгое определение он должен иметь. Взять к примеру оригинальный Лисп: он определен чисто математически — как система из десятка функций для работы со словами и списками, с лямбда-исчислением в качестве вычислительного движка. Это самый нижний уровень. Идеальная математическая точность. С такой же точностью определена таблица кодирования аминокислот ДНК. Однако, слои систем возведенных поверх

основы — безнадежно сложны, хаотичны, подвержены сбоям, но тем не менее стабильны и работают. Чем дальше в лес — тем больше дров. При всей простоте основополагающих конструкций в современных языках, будь то класс или лямбда, никто не в силах предвидеть и осознать те возможности, которые в них скрыты. Мы выбираем самые простые и полезные, называем их «шаблонами проектирования» и думаем что они — те кирпичи, из которых можно собрать все что угодно; ан нет, за горизонтом всегда остается бездна возможностей. В одной «лямбде» заложена вся сложность мира вычислений. Она как числа 1, 2, 3,... нет ничего проще? Но отношения между числами непостижимы.

1.1 Matlab

Matlab (сокращение от англ. *«Matrix Laboratory»*, в русском языке произносится как Матла́б) — пакет прикладных программ для решения задач технических вычислений и одноимённый язык программирования, используемый в этом пакете. Matlab используют более 1 000 000 инженерных и научных работников, он работает на большинстве современных операционных систем, включая Linux, Mac OS, Solaris (начиная с версии R2010b поддержка Solaris прекращена) и Microsoft Windows.

MATLAB как язык программирования был разработан Кливом Моулером (англ. Cleve Moler) в конце 1970-х годов, когда он был деканом факультета компьютерных наук в Университете Нью-Мексико. Целью разработки служила задача дать студентам факультета возможность использования программных библиотек Linpack и EISPACK без необходимости изучения Фортрана. Вскоре новый язык распространился среди других университетов и был с большим интересом встречен учёными, работающими в области прикладной математики. До сих пор в Интернете можно найти версию 1982 года, написанную на Фортране, распространяемую с открытым исходным кодом. Инженер Джон Литтл (англ. John N. (Jack) Little) познакомился с этим языком во время визита Клива Моулера в Стэндфордский университет в 1983 году. Поняв, что новый язык обладает большим коммерческим потенциалом, он объединился с Кливом Моулером и

Стивом Бангертом (англ. Steve Bangert). Совместными усилиями они переписали MATLAB на C и основали в 1984 компанию The MathWorks для дальнейшего развития. Эти переписанные на C библиотеки долгое время были известны под именем JACKPAC. Первоначально MATLAB предназначался для проектирования систем управления (основная специальность Джона Литтла), но быстро завоевал популярность во многих других научных и инженерных областях. Он также широко использовался и в образовании, в частности, для преподавания линейной алгебры и численных методов.

Язык MATLAB является высокоуровневым интерпретируемым языком программирования, включающим основанные на матрицах структуры данных, широкий спектр функций, интегрированную среду разработки, объектно-ориентированные возможности и интерфейсы к программам, написанным на других языках программирования.

Программы, написанные на MATLAB, бывают двух типов — функции и скрипты. Функции имеют входные и выходные аргументы, а также собственное рабочее пространство для хранения промежуточных результатов вычислений и переменных. Скрипты же используют общее рабочее пространство. Как скрипты, так и функции не компилируются в машинный код и сохраняются в виде текстовых файлов. Существует также возможность сохранять так называемые pre-parsed программы — функции и скрипты, обработанные в вид, удобный для машинного исполнения. В общем случае такие программы выполняются быстрее обычных, особенно если функция содержит команды построения графиков.

Основной особенностью языка MATLAB являются его широкие возможности по работе с матрицами, которые создатели языка выразили в лозунге «думай векторно» (англ. Think vectorized).

1.1.1 Применение

Математика и вычисления

MATLAB предоставляет пользователю большое количество (несколько сотен) функций для анализа данных, покрывающие практически все области математики, в частности:

- Матрицы и линейная алгебра — алгебра матриц, линейные уравнения, собственные значения и вектора, сингулярности, факторизация матриц и другие.
- Многочлены и интерполяция — корни многочленов, операции над многочленами и их дифференцирование, интерполяция и экстраполяция кривых и другие.
- Математическая статистика и анализ данных — статистические функции, статистическая регрессия, цифровая фильтрация, быстрое преобразование Фурье и другие.
- Обработка данных — набор специальных функций, включая построение графиков, оптимизацию, поиск нулей, численное интегрирование (в квадратурах) и другие.
- Дифференциальные уравнения — решение дифференциальных и дифференциально-алгебраических уравнений, дифференциальных уравнений с запаздыванием, уравнений с ограничениями, уравнений в частных производных и другие.
- Разреженные матрицы — специальный класс данных пакета MATLAB, использующийся в специализированных приложениях.
- Целочисленная арифметика — выполнение операций целочисленной арифметики в среде MATLAB.

Разработка алгоритмов

MATLAB предоставляет удобные средства для разработки алгоритмов, включая высокоуровневые с использованием концепций объектно-ориентированного программирования. В нём имеются все необходимые средства интегрированной среды разработки, включая отладчик и профайлер.

1.1.2 Преимущества

Основные преимущества MATLAB, выгодно выделяющие ее среди существующих ныне математических систем и пакетов (MathCad, Mathematica и др.), заключаются в следующем:

- система MATLAB специально создана для проведения именно инженерных расчетов: математический аппарат, используемый ею, предельно приближен к современному математическому аппарату инженера и ученого и опирается на вычисления с матрицами, векторами и комплексными числами; графическое представление функциональных зависимостей здесь организовано в форме, требуемой именно инженерной документацией;
- язык программирования системы MATLAB очень прост, близок к языку Basic, посилен любому начинающему; он содержит всего несколько десятков операторов; незначительное количество операторов здесь компенсируется большим числом процедур и функций, смысл которых понятен пользователю с соответствующей математической и инженерной подготовкой;
- в отличие от большинства математических систем, MATLAB является открытой системой; это означает, что практически все процедуры и функции MATLAB доступны не только для использования, но и для коррекции и модификации; MATLAB - система, которую пользователь может расширять по своему усмотрению созданными им программами и процедурами (подпрограммами); ее легко приспособить к решению нужных классов задач;
- очень удобна возможность, как составлять, собственные отдельные программы с целью многократного их использования для исследований, так и применять практически все вычислительные возможности системы в режиме чрезвычайно мощного научного калькулятора; это делает MATLAB незаменимым средством проведения научных и инженерных расчетных исследований; Сумматор предназначен для сложения двух чисел, заданных в двоичном коде. Число входов и выходов сумматора определяется разрядностью слагаемых. Одноразрядный двоичный сумматор характеризуется таблицей истинности (табл.10.3). Такой сумматор называется полным, т.к. обрабатывает сигнал переноса с предыдущего разряда Pi и выдает сигнал переноса на последующий разряд Pi+1.
- последние версии MATLAB позволяют легко интегрировать ее с текстовым редактором Word, что дает возможность использовать при

составлении текстовых документов вычислительные и графические средства MATLAB.

- Возможности системы огромны, а по скорости выполнения задач она опережает многие другие подобные системы. Все эти особенности делают систему MATLAB весьма привлекательной для использования в учебном процессе высших учебных заведений.

1.2 Ruby

Ruby (англ. ruby — рубин, произносится ['ru:b□] — руби) — динамический, рефлективный, интерпретируемый высокоуровневый язык программирования для быстрого и удобного объектно-ориентированного программирования. Язык обладает независимой от операционной системы реализацией многопоточности, строгой динамической типизацией, сборщиком мусора и многими другими возможностями. Ruby близок по особенностям синтаксиса к языкам Perl и Eiffel, по объектно-ориентированному подходу — к Smalltalk. Также некоторые черты языка взяты из Python, Lisp, Dylan и CLU (англ.).

Кроссплатформенная реализация интерпретатора языка является полностью свободной. Создатель Ruby — Юкихиро Мацумото (Matz) — интересовался языками программирования, ещё будучи студентом, но идея о разработке нового языка появилась позже. Ruby начал разрабатываться 23 февраля 1993 года и вышел в свет в 1995 году. Название навеяно языком Perl, многие особенности синтаксиса и семантики из которого заимствованы в Ruby: англ. pearl — «жемчужина», ruby — «рубин». Целью разработки было создание «настоящего объектно-ориентированного», лёгкого в разработке, интерпретируемого языка программирования. Из письма автора:

"Ruby родился 23 февраля 1993 года. В тот день я беседовал со своим коллегой о возможности существования объектно-ориентированного сценарного языка. Я знал Perl (Perl4, а не Perl5), но он мне не нравился — был в нём некий привкус игрушечного языка (да и поныне есть). А объектно-ориентированный интерпретируемый язык казался многообещающим. В то время я знал Python. Но он мне не нравился потому, что я не считал его настоящим объектно-ориентированным языком. Его ОО свойства казались надстройкой над языком. Мне, как языковому маньяку и фанату объектно-ориентированного программирования с пятнадцатилетним стажем, очень,

очень хотелось, чтобы был истинно объектно-ориентированный, простой в использовании язык. Я пытался найти такой язык, но его не было.

Тогда я решил его создать. Прошло несколько месяцев, прежде чем интерпретатор заработал. Я добавил в мой язык то, что мне хотелось — итераторы, обработку исключений, автоматическую сборку мусора. Затем я переорганизовал свойства Perl и реализовал их как библиотеку классов. В декабре 1995 года я опубликовал Ruby 0.95 в японских новостных группах. С тех пор появились сайты, списки рассылок. В списках рассылок идут жаркие обсуждения. Самый старый список сейчас содержит 14 789 писем."

В Японии Ruby стал популярным с момента появления первой общедоступной версии в 1995 году, однако наличие документации только на японском языке сдерживало его дальнейшее распространение. Лишь в 1997 году появилось описание Ruby на английском языке, а в 1998 году открылся форум «ruby-talk». Это положило начало росту известности языка в остальном мире. Издано несколько книг на различных языках, в том числе на русском. Сейчас Ruby входит в большинство дистрибутивов ОС Linux, поставляется вместе с Mac OS X, доступен пользователям других операционных систем.

Хронология выхода версий

В списке ниже перечислены лишь наиболее крупные обновления.

Название версии	Дата выхода	Примечания
0.06	7 января 1994	Первая версия, указанная в Changelog'ах
1.0-961225	25 декабря 1996	Данная версия следовала сразу за версией 0.99.4-961224, выпущенной накануне. Номер после числа 1.0 — дата выпуска версии. Новые версии линейки 1.0 выходили ещё год (до 1.0-971225).
1.1 alpha0	13 августа 1997	Альфа-версии выходили вплоть до 7 октября 1997 (1.1 alpha9)
1.1b0	4 декабря 1997	Следующая версия после 1.1 alpha9. 27 февраля 1998 вышла версия 1.1b9, затем вплоть до середины 1998 выходили экспериментальные выпуски с обозначением вида 1.1b9_31 (версия 1.1b9_31 была выпущена, но в документации не отмечена).

1.1c0	17 июля 1998	Данная версия следовала за версией 1.1b9_31. Модификации этой версии выходили вплоть до 26 ноября 1998 (1.1c9).
1.1d0 (pre1.2)	16 декабря 1998	Данная версия следовала за версией 1.1c9. 22 декабря 1998 была выпущена экспериментальная версия 1.1d1, завершившая данную линейку.
1.2 (stable)	25 декабря 1998	В дальнейшем выходили модификации данной версии вплоть до версии 1.2.5, выпущенной 13 апреля 1999 года. 21 июня 1999 года была выпущена версия 1.2.6, объявленная как финальная версия 1.2 (1.2 final). 15 июля 1999 года вышла переупакованная (repacked) версия 1.2.6.
1.3 (development)	24 декабря 1998	Отдельная ветка модификаций, разрабатываемая независимо от линейки 1.2 (по аналогии с ядром ОС Linux). Первая версия была объявлена как версия для разработки (development version) и следовала за версией 1.1d1. В дальнейшем последовало множество промежуточных модификаций: ruby-1.3.1-990215 — ruby-1.3.4-990625, после чего от указания даты в номере отказались и выпустили 1.3.5 — 1.4 alpha (15 июля 1999), 1.3.6 — 1.4 alpha (28 июля 1999), 1.3.7 — 1.4 beta (6 августа 1999).
1.4.0 (stable)	13 августа 1999	Данная версия появилась через несколько дней после выхода 1.3.7 — 1.4 beta. В дальнейшем выходили новые модификации вплоть до версии 1.4.6, вышедшей 16 августа 2000 года.
1.5.0 (development)	20 ноября 1999	Данная линейка предназначалась исключительно для проверки различных нововведений при разработке. Модификации данной линейки доступны исключительно в репозитории проекта и, соответственно, сборки данной версии на официальный сервер не выкладывались.
1.6.0 (stable)	19 сентября 2000	В дальнейшем выпускались модификации этой версии вплоть до версии 1.6.8 (24 декабря 2002). 21 сентября 2005 года был выпущен патч для версии 1.6.8.
1.7.0 (development)	24 февраля 2001	Данная линейка предназначалась исключительно для проверки различных нововведений при разработке. Модификации данной линейки

		доступны исключительно в репозитории проекта и, соответственно, сборки данной версии на официальный сервер не выкладывались.
1.8.0 (stable)	4 августа 2003	В дальнейшем последовало большое число модификаций, которые выходят до сих пор (1 января 2011 года), например, промежуточная версия 1.8.7-p330 вышла 24 декабря 2010 года.
1.9.0 (development)	25 декабря 2007	Изначально экспериментальная ветка созданная для практической проверки ряда нововведений.
1.9.3 (stable)	31 октября 2011	Последняя ветка на данный момент (10 ноября 2012 года). Соответственно, модификации также выходят до сих пор. Отличия от 1.9.2 — существенны.
2.0.0 (stable)	24 февраля 2013	

Таблица 1

1.2.1 Философия

Мацумото, фанат объектно-ориентированного программирования, мечтал о языке, более мощном, чем Perl, и более объектно-ориентированном, чем Python. Основное назначение Ruby — создание простых и в то же время понятных программ, где важна не скорость работы программы, а малое время разработки, понятность и простота синтаксиса.

Язык следует принципу «наименьшей неожиданности»: программа должна вести себя так, как ожидает программист. Однако в контексте Ruby это означает наименьшее удивление не при знакомстве с языком, а при его основательном изучении. Сам Мацумото утверждает, что целью разработки была минимизация неожиданностей при программировании для него, но после распространения языка он с удивлением узнал, что мышление программистов похоже, и для многих из них принцип «наименьшей неожиданности» совпал с его принципом.

Ruby также унаследовал идеологию языка программирования Perl в части предоставления программисту возможностей достижения одного и того же результата несколькими различными способами. Люди различны, и им для свободы необходима возможность выбирать. «Я предпочитаю обеспечить много путей, если это возможно, но поощрять или вести пользователей, чтобы выбрать лучший путь, если это возможно».

Одной из основных целей разработки было освобождение программистов от рутинной работы, которую вычислитель может выполнять быстрее и качественнее. Особое внимание, в частности, уделялось будничным рутинным занятиям (обработка текстов, администрирование), и для них язык настроен особенно хорошо.

В противовес машинно-ориентированным языкам, работающим быстрее, целью этой разработки был язык, наиболее близкий к человеку. Любая работа с компьютером выполняется людьми и для людей, и необходимо заботиться в первую очередь о затрачиваемых усилиях людей. Язык позволяет максимально быстро и просто для человека выполнить задачу, хотя, возможно, это и потребует дополнительного времени работы компьютера.

Принципы программирования и устройства языка иногда выделяются в термин «Путь Ruby» (англ. Ruby Way). Хэл Фултон выделяет такие принципы, как «просто, но не слишком просто», «принцип наименьшего удивления», вторичность скорости работы программы, динамичность, простые строгие правила, выполнение которых не доходит до педантизма, потребность создавать полезные и красивые программы как причина программирования. В целом они не имеют точной формулировки и иногда этот термин используется для критики.

Реализация

Для Ruby существуют несколько реализаций: официальный интерпретатор, написанный на Си, JRuby — реализация для Java, интерпретатор для платформы .NET IronRuby, Rubinius — написанная в основном на Ruby и базирующаяся на идеях Smalltalk-80 VM, MagLev — другая базирующаяся на Smalltalk разработка от компании Gemstone, Blue Ruby — реализация Ruby для виртуальной машины ABAP, MacRuby — реализация для Mac OS с фокусом на максимальную интеграцию с возможностями операционной системы, mruby — реализация для встраивания в программы.

Официальный интерпретатор портирован под большинство платформ, включая Unix, Microsoft Windows (в том числе Windows CE), DOS, Mac OS X, OS/2, Amiga, BeOS, Syllable, Acorn RISC OS и другие.

Для Windows существует RubyInstaller for Windows и есть возможность запуска под Cygwin для большей совместимости с Unix.

Документация

Система RDoc предназначена для автоматического извлечения документации из исходных кодов и программ на Ruby и её дальнейшей обработки. Является стандартом де-факто для подготовки документации по программному обеспечению, написанному на Ruby.

1.2.2 Преимущества

- Имеет лаконичный и простой синтаксис, частично разработанный под влиянием Ада, Eiffel и Python.
- Позволяет обрабатывать исключения в стиле Java и Python.
- Позволяет переопределять операторы, которые на самом деле являются методами.
- Полностью объектно-ориентированный язык программирования. Все данные в Ruby являются объектами в понимании Smalltalk. Единственное исключение — управляющие конструкции, которые в Ruby в отличие от Smalltalk не являются объектами. Например, число «1» — это экземпляр класса Fixnum. Также поддерживается добавление методов в класс и даже в конкретный экземпляр во время выполнения программы.
- Не поддерживает множественное наследование, но вместо него может использоваться концепция «примесей», основанная в данном языке на механизме модулей.
- Содержит автоматический сборщик мусора. Он работает для всех объектов Ruby, в том числе для внешних библиотек.
- Создавать расширения для Ruby на Си очень просто частично из-за сборщика мусора, частично из-за несложного и удобного API.
- Поддерживает замыкания с полной привязкой к переменным.
- Поддерживает блоки кода (код заключается в { ... } или do ... end). Блоки могут использоваться в методах или преобразовываться в замыкания.
- Целые переменные в Ruby автоматически конвертируются между типами Fixnum (32-разрядные) и Bignum (больше 32 разрядов) в зависимости от их значения, что позволяет производить целочисленные математические расчёты со сколь угодно большой точностью.
- Не требует предварительного объявления переменных, но для интерпретатора желательно, чтобы переменным присваивалось пустое значение nil (тогда интерпретатор знает, что идентификатор обозначает

переменную, а не имя метода). Язык использует простые соглашения для обозначения области видимости. Пример: просто var — локальная переменная, @var — переменная экземпляра (член или поле объекта класса), @@var — переменная класса, $var — глобальная переменная.

- В Ruby непосредственно в языке реализованы многие шаблоны проектирования, так, например, «одиночка» (singleton) может быть (хотя и не обязан) реализован добавлением необходимых методов к одному конкретному объекту (см. ниже).
- Может динамически загружать расширения, если это позволяет операционная система.
- Имеет независимую от ОС поддержку невытесняющей многопоточности.

1.3 Python

Python (англ. python — питон, произносится — па́йтон; в русском языке распространено название пито́н) — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Python поддерживает несколько парадигм программирования, в том числе структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений и удобные высокоуровневые структуры данных. Код в Питоне организовывается в функции и классы, которые могут объединяться в модули (которые в свою очередь могут быть объединены в пакеты).

Эталонной реализацией Python является интерпретатор CPython, поддерживающий большинство активно используемых платформ. Он распространяется под свободной лицензией Python Software Foundation License, позволяющей использовать его без ограничений в любых приложениях, включая проприетарные. Есть реализации интерпретаторов для JVM (с возможностью компиляции), MSIL (с возможностью компиляции),

LLVM и других. Проект PyPy предлагает реализацию Питона на самом Питоне, что уменьшает затраты на изменения языка и постановку экспериментов над новыми возможностями.

Python — активно развивающийся язык программирования, новые версии (с добавлением/изменением языковых свойств) выходят примерно раз в два с половиной года. Вследствие этого и некоторых других причин на Python отсутствуют стандарт ANSI, ISO или другие официальные стандарты, их роль выполняет CPython.

История

Разработка языка Python была начата в конце 1980-х годов сотрудником голландского института CWI Гвидо ван Россумом. Для распределённой ОС Amoeba требовался расширяемый скриптовый язык, и Гвидо начал писать Python на досуге, позаимствовав некоторые наработки для языка ABC (Гвидо участвовал в разработке этого языка, ориентированного на обучение программированию). В феврале 1991 года Гвидо опубликовал исходный текст в ньюсгруппе alt.sources. С самого начала Python проектировался как объектно-ориентированный язык.

Название языка произошло вовсе не от вида пресмыкающихся. Автор назвал язык в честь популярного британского комедийного телешоу 1970-х «Летающий цирк Монти Пайтона». Впрочем, всё равно название языка чаще ассоциируют именно со змеёй, нежели с фильмом — пиктограммы файлов в KDE или в Microsoft Windows и даже эмблема на сайте python.org (до выхода версии 2.5) изображают змеиные головы.

Наличие дружелюбного, отзывчивого сообщества пользователей считается наряду с дизайнерской интуицией Гвидо одним из факторов успеха Python. Развитие языка происходит согласно чётко регламентированному процессу создания, обсуждения, отбора и реализации документов PEP (англ. Python Enhancement Proposal) — предложений по развитию Python[12].

3 декабря 2008 года, после длительного тестирования, вышла первая версия Python 3000 (или Python 3.0, также используется сокращение Py3k). В Python 3000 устранены многие недостатки архитектуры с максимально возможным (но не полным) сохранением совместимости со старыми версиями Python. На сегодня поддерживаются обе ветви развития (Python 3.x и 2.x).

Влияние других языков на Python

Появившись сравнительно поздно, Python создавался под влиянием множества языков программирования:

- ABC — отступы для группировки операторов, высокоуровневые структуры данных (map)[14][15] (Python фактически создавался как попытка исправить ошибки, допущенные при проектировании ABC);
- Modula-3 — пакеты, модули, использование else совместно с try и except, именованные аргументы функций (на это также повлиял Common Lisp);
- C, C++ — некоторые синтаксические конструкции (как пишет сам Гвидо ван Россум — он использовал наиболее непротиворечивые конструкции из C, чтобы не вызвать неприязнь у C-программистов к Python[14]);
- Smalltalk — объектно-ориентированное программирование;
- Lisp — отдельные черты функционального программирования (lambda, map, reduce, filter и другие);
- Fortran — срезы массивов, комплéксная арифметика;
- Miranda — списочные выражения;
- Java — модули logging, unittest, threading (часть возможностей оригинального модуля не реализована), xml.sax стандартной библиотеки, совместное использование finally и except при обработке исключений, использование @ для декораторов;
- Icon — генераторы.

Большая часть других возможностей Python (например, байт-компиляция исходного кода) также была реализована ранее в других языках.

Реализации

CPython является основной, но не единственной реализацией языка программирования Python. Существуют также следующие реализации:

Jython — реализация Python, использующая JVM в качестве среды исполнения. Позволяет прозрачно использовать Java-библиотеки.

PyS60 — реализация языка для смартфонов фирмы Nokia на платформе Series 60.

IronPython — Python для .NET Framework и Mono. Компилирует Python программы в MSIL, таким образом предоставляя полную интеграцию с .NET системой.

Stackless — также написанная на Си реализация Python. Это не полноценная реализация, а патчи к CPython. Предоставляет расширенные возможности многопоточного программирования и значительно большую глубину рекурсии.

Python for .NET — ещё одна реализация Python для .NET. В отличие от IronPython эта реализация не компилирует Python код в MSIL, а только предоставляет интерпретатор, написанный на C#. Позволяет использовать .NET-сборки из Python кода.

PyPy — реализация Python, написанная на Python. Позволяет легко проверять новые возможности. В PyPy кроме стандартного CPython включены возможности Stackless, Psyco, модификация AST «на лету» и многое другое. В проект интегрированы возможности анализа Python кода и трансляция в другие языки и байткоды виртуальных машин (Си, LLVM, Javascript, .NET с версии 0.9.9). Начиная с 0.9.0, возможна полностью автоматическая трансляция интерпретатора на Си, в результате чего достигается скорость, приемлемая для использования (в 2—3 раза медленнее чем CPython при отключённом JIT для версии 0.9.9). JIT находится в активной доработке.

python-safethread— версия CPython без GIL, что позволяет одновременно исполнять Python потоки на всех доступных процессорах. Внесены также некоторые другие изменения.

Unladen Swallow — начатый Google проект по разработке высокоэффективного, максимально совместимого с CPython JIT-компилятора на базе LLVM. Согласно планам по развитию Python, планировалось перенести исходный код Unladen Swallow в CPython в версии 3.3. Но PEP-3146 был отменён в связи с отсутствием интереса к Unladen Swallow со стороны Google, основного спонсора разработки.

tinypy— минималистическая версия Python. Часть возможностей CPython не реализована.

1.3.1 Преимущества

Дизайн языка Python построен вокруг объектно-ориентированной модели программирования. Реализация ООП в Питоне является элегантной, мощной и хорошо продуманной, но вместе с тем достаточно специфической по сравнению с другими объектно-ориентированными языками.

Возможности и особенности:

- Классы являются одновременно объектами со всеми ниже приведёнными возможностями.
- Наследование, в том числе множественное.
- Полиморфизм (все функции виртуальные).
- Инкапсуляция (два уровня — общедоступные и скрытые методы и поля). Особенность — скрытые члены доступны для использования и помечены как скрытые лишь особыми именами.
- Специальные методы, управляющие жизненным циклом объекта: конструкторы, деструкторы, распределители памяти.
- Перегрузка операторов (всех, кроме is, '.', '=' и символьных логических).
- Свойства (имитация поля с помощью функций).
- Управление доступом к полям (эмуляция полей и методов, частичный доступ, и т. п.).
- Методы для управления наиболее распространёнными операциями (истинностное значение, len(), глубокое копирование, сериализация, итерация по объекту, ...)
- Метапрограммирование (управление созданием классов, триггеры на создание классов, и др.)
- Полная интроспекция.

Функциональное программирование

Python поддерживает парадигму функционального программирования, в частности:

- функция является объектом
- функции высших порядков
- рекурсия
- развитая обработка списков (списковые выражения, операции над последовательностями, итераторы)

1.3.2 Недочеты

Низкое быстродействие

Классический Питон, как и многие другие интерпретируемые языки, не применяющие, например, JIT-компиляторы, имеют общий недостаток — сравнительно невысокую скорость выполнения программ. Сохранение байт-кода (файлы .pyc и .pyo) позволяет интерпретатору не тратить лишнее время на перекомпиляцию кода модулей при каждом запуске, в отличие, например, от языка Perl. Кроме того, существует специальная JIT-библиотека psyco, позволяющая ускорить выполнение программ (однако приводящая к увеличению потребления оперативной памяти). Эффективность psyco сильно зависит от архитектуры программы.

Существуют проекты реализаций языка Python, вводящие высокопроизводительные виртуальные машины (ВМ) в качестве компилятора заднего плана. Примерами таких реализаций может служить PyPy, базирующийся на LLVM; более ранней инициативой является проект Parrot. Ожидается, что использование ВМ типа LLVM приведёт к тем же результатам, что и использование аналогичных подходов для реализаций языка Java, где низкая вычислительная производительность в основном преодолена.

Множество программ/библиотек для интеграции с другими языками программирования (см. выше) предоставляют возможность использовать другой язык для написания критических участков.

В самой популярной реализации языка Python интерпретатор довольно велик и более требователен к ресурсам, чем в аналогичных популярных реализациях Tcl, Forth, LISP или Lua, что ограничивает его применение во встроенных системах. Тем не менее, Python нашёл применение в КПК и некоторых моделях мобильных телефонов.

Невозможность модификации встроенных классов

По сравнению с Ruby и некоторыми другими языками, в Python отсутствует возможность модифицировать встроенные классы, такие, как int, str, float, list и другие, что, однако, позволяет Python потреблять меньше оперативной памяти и быстрее работать. Ещё одной причиной введения такого ограничения является необходимость согласования с модулями расширения. Многие модули (в целях оптимизации быстродействия)

преобразуют Python-объекты элементарных типов к соответствующим Си-типам вместо манипуляций с ними посредством Си-API. Также это избавляет от многих потенциальных ошибок при неконтролируемом динамическом переопределении встроенных типов.

Глобальная блокировка интерпретатора (GIL)

GIL (Global Interpreter Lock) — особенность, присущая CPython, Stackless и PyPy, но отсутствующая в Jython и IronPython. При своей работе основной интерпретатор Python постоянно использует большое количество потоко-небезопасных данных. В основном это словари, в которых хранятся атрибуты объектов, и обращения к внешнему коду (написанному на C и т. п.). Во избежание разрушения этих данных при совместной модификации из разных потоков, перед началом исполнения нескольких инструкций (по умолчанию 100) поток интерпретатора захватывает GIL, а по окончании освобождает. Вследствие этого, в каждый момент времени в одном процессе интерпретатора Python может исполняться только один поток Python кода, даже если в компьютере имеется несколько процессоров или процессорных ядер (GIL также освобождается на время выполнения блокирующих операций, таких как ввод-вывод, изменения/проверка состояния синхронизирующих примитивов и других — таким образом, если один поток блокируется, другие могут исполняться). Была предпринята попытка перехода к более гранулированным синхронизациям, однако из-за частых захватов/освобождений блокировок эта реализация оказалась слишком медленной на однопоточных приложениях. В ближайшем будущем переход от GIL к другим техникам не предполагается по причине того, что, по мнению автора языка Python Гвидо Россума, однопоточных программ, ускоряемых GIL, больше, чем многопоточных, которые GIL тормозит. Однако есть python-safethread — CPython без GIL и с некоторыми другими изменениями (по утверждениям его авторов, на однопоточных приложениях скорость соответствует 60-65 % от скорости оригинального CPython).

Эта проблема имеет два основных варианта решения. Первый — отказ от совместного использования изменяемых данных и вызовов внешнего кода. При этом данные дублируются в потоках и необходимость обеспечения их синхронизации (если таковая нужна) лежит на программисте[57]. Этот подход ведёт к увеличению потребления оперативной памяти (однако не настолько сильно, как при использовании процессов в Windows — так как для Unix это не критично, процессы небольшие).

Второй подход — обеспечение более гранулированной синхронизации — для отдельных структур данных. В этом случае падает производительность вследствие увеличения числа освобождений/захватов блокировок.

Если необходимо параллельное исполнение нескольких потоков Python кода, то можно воспользоваться процессами, например, модулем processing, который имитирует семантику стандартного модуля threading, но использует процессы вместо потоков. Начиная с версии 2.6, модуль processing добавлен в стандартную библиотеку и переименован в multiprocessing. Есть множество модулей, упрощающих написание параллельных и/или распределённых приложений на Python, таких как parallelpython, Pypar, pympi и других. GIL освобождается при исполнении кода большинства расширений, например, NumPy/SciPy, позволяя на время расчётов исполняться другому Python потоку. Другим решением может быть использование IronPython или Jython, лишённых этой особенности классического Python.

В версии 3.2 будет внедрён изменённый GIL. Основные нововведения:

Переключение по таймауту, а не по количеству опкодов — в предыдущей версии GIL освобождался через каждые 100 операций. Но операция может выполняться от наносекунд, до долей секунд. В новой версии блокировка будет сниматься каждые 5 миллисекунд;

Уменьшение накладных расходов и увеличение эффективности в случае частых переключений — в некоторых системах (в основном в MacOS X) наблюдаются проблема с эффективностью блокировки, когда GIL уже захвачен: системные вызовы становятся дорогими. В новой версии задержки будут уменьшены;

Уменьшение задержек при переключении потоков благодаря введению механизма принудительного переключения потока и приоритетных запросов (когда поток выполняет приоритетный запрос, то GIL освобождается как можно быстрее).

1.4 Java

Java - объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Приложения Java обычно компилируются в специальный байт-код, поэтому они могут работать на любой виртуальной Java-машине (JVM) вне зависимости от компьютерной архитектуры. Дата официального выпуска — 23 мая 1995 года.

1.4.1 Основные особенности языка

Программы на Java транслируются в байт-код, выполняемый виртуальной машиной Java (JVM) — программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор.

Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии Java является гибкая система безопасности благодаря тому, что исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером) вызывают немедленное прерывание.

- Часто к недостаткам концепции виртуальной машины относят то, что исполнение байт-кода виртуальной машиной может снижать производительность программ и алгоритмов, реализованных на языке Java. В последнее время был внесен ряд усовершенствований, которые несколько увеличили скорость выполнения программ на Java:
- применение технологии трансляции байт-кода в машинный код непосредственно во время работы программы (JIT-технология) с возможностью сохранения версий класса в машинном коде,
- широкое использование платформенно-ориентированного кода (native-код) в стандартных библиотеках,
- аппаратные средства, обеспечивающие ускоренную обработку байт-кода (например, технология Jazelle, поддерживаемая некоторыми процессорами фирмы ARM).

По данным сайта shootout.alioth.debian.org, для семи разных задач время выполнения на Java составляет в среднем в полтора-два раза больше, чем для C/C++, в некоторых случаях Java быстрее, а в отдельных случаях в 7 раз медленнее. С другой стороны, для большинства из них потребление памяти Java-машиной было в 10-30 раз больше, чем программой на C/C++. Также примечательно исследование, проведённое компанией Google, согласно которому отмечается существенно более низкая производительность и бóльшее потребление памяти в тестовых примерах на Java в сравнении с аналогичными программами на C++.

Идеи, заложенные в концепцию и различные реализации среды виртуальной машины Java, вдохновили множество энтузиастов на расширение перечня языков, которые могли бы быть использованы для создания программ, исполняемых на виртуальной машине. Эти идеи нашли также выражение в спецификации общеязыковой инфраструктуры CLI, заложенной в основу платформы .NET компанией Microsoft.

Классификация платформ Java

Внутри Java существуют несколько основных семейств технологий:

- Java SE — Java Standard Edition, основное издание Java, содержит компиляторы, API, Java Runtime Environment; подходит для создания пользовательских приложений, в первую очередь — для настольных систем.
- Java EE — Java Enterprise Edition, представляет собой набор спецификаций для создания программного обеспечения уровня предприятия.
- Java ME — Java Micro Edition, создана для использования в устройствах, ограниченных по вычислительной мощности, например в мобильных телефонах, КПК, встроенных системах;
- JavaFX — технология, являющаяся следующим шагом в эволюции Java как Rich Client Platform; предназначена для создания графических интерфейсов корпоративных приложений и бизнеса.
- Java Card — технология предоставляет безопасную среду для приложений, работающих на смарт-картах и других устройствах с очень ограниченным объёмом памяти и возможностями обработки.

Производительность

Программы, написанные на Java, имеют репутацию более медленных и занимающих больше оперативной памяти, чем написанные на языке Си. Тем не менее, скорость выполнения программ, написанных на языке Java, была существенно улучшена с выпуском в 1997—1998 годах так называемого JIT-компилятора в версии 1.1 в дополнение к другим особенностям языка для поддержки лучшего анализа кода (такие как внутренние классы, класс StringBuffer, упрощенные логические вычисления и т. д.). Кроме того была произведена оптимизация виртуальной машины Java — с 2000 года для этого используется виртуальная машина HotSpot. По состоянию на февраль 2012 года, код Java 7 приблизительно лишь в 1.8 раза медленнее кода, написанного на языке Си.

Некоторые платформы предлагают аппаратную поддержку выполнения для Java. К примеру, микроконтроллеры выполняющие код Java на аппаратном обеспечении вместо программной JVM, а также основанные на ARM процессоры, которые поддерживают выполнение байткода Java через опцию Jazelle.

1.4.2 Основные возможности

- автоматическое управление памятью;
- расширенные возможности обработки исключительных ситуаций;
- богатый набор средств фильтрации ввода/вывода;
- набор стандартных коллекций: массив, список, стек и т. п.;
- наличие простых средств создания сетевых приложений (в том числе с использованием протокола RMI);
- наличие классов, позволяющих выполнять HTTP-запросы и обрабатывать ответы;
- встроенные в язык средства создания многопоточных приложений;
- унифицированный доступ к базам данных:
    - на уровне отдельных SQL-запросов — на основе JDBC, SQLJ;
    - на уровне концепции объектов, обладающих способностью к хранению в базе данных — на основе Java Data Objects (англ.) и Java Persistence API;
- поддержка обобщений (начиная с версии 1.5);
- параллельное выполнение программ.

Преобразования при математических операциях

В языке Java действуют следующие правила:

1. Если один операнд имеет тип double, другой тоже преобразуется к типу double.
2. Иначе, если один операнд имеет тип float, другой тоже преобразуется к типу float.
3. Иначе, если один операнд имеет тип long, другой тоже преобразуется к типу long.
4. Иначе оба операнда преобразуются к типу int.

Данный способ неявного преобразования встроенных типов полностью совпадает с преобразованием типов в C++.

Объектные переменные, объекты, ссылки и указатели

В языке Java имеются только динамически создаваемые объекты. Причем переменные объектного типа и объекты в Java — совершенно разные сущности. Переменные объектного типа являются ссылками, то есть неявными указателями на динамически создаваемые объекты. Это подчёркивается синтаксисом описания переменных. Так, в Java нельзя писать:

При присваиваниях, передаче в подпрограммы и сравнениях объектные переменные ведут себя как указатели, то есть присваиваются, копируются и сравниваются адреса объектов. А при доступе с помощью объектной переменной к полям данных или методам объекта не требуется никаких специальных операций разыменовывания — этот доступ осуществляется так, как если бы объектная переменная была самим объектом.

Объектными являются переменные любого типа, кроме примитивного. Явных указателей в Java нет. В отличие от указателей C, C++ и других языков программирования, ссылки в Java в высокой степени безопасны благодаря жёстким ограничениям на их использование, в частности:

- Нельзя преобразовывать объект типа int или любого другого примитивного типа в указатель или ссылку и наоборот.
- Над ссылками запрещено выполнять операции ++, —, +, − или любые другие арифметические операции.

- Преобразование типов между ссылками жёстко регламентировано. За исключением ссылок на массивы, разрешено преобразовывать ссылки только между наследуемым типом и его наследником, причём преобразование наследуемого типа в наследующий должно быть явно задано и во время выполнения производится проверка его осмысленности. Преобразования ссылок на массивы разрешены лишь тогда, когда разрешены преобразования их базовых типов, а также нет конфликтов размерности.
- В Java нет операций взятия адреса (&) или взятия объекта по адресу (*). Звёздочка в Java означает умножение, и только. Амперсанд (&) означает всего лишь «побитовое и» (двойной амперсанд — «логическое и»). Однако для булевых типов одиночный амперсанд означает «логическое и», отличающееся от двойного тем, что цепь проверок не прекращается при получении в выражении значения false[38] (напр. a == b && foo() == bar() не повлечёт вызовов foo() и bar() в случае, если a != b, тогда как использование & повлечёт в любом случае.

Благодаря таким специально введенным ограничениям в Java невозможно прямое манипулирование памятью на уровне физических адресов (хотя ссылки, не указывающие ни на что, есть: значение такой ссылки обозначается null).

Классы и функции

Java не является процедурным языком: любая функция может существовать только внутри класса. Это подчёркивает терминология языка Java, где нет понятий «функция» или «функция-член» (англ. member function), а только метод. В методы превратились и стандартные функции. Например, в Java нет функции sin(), а есть метод Math.sin() класса Math (содержащего, кроме sin(), методы cos(), exp(), sqrt(), abs() и многие другие). Конструкторы в Java не считаются методами. Деструкторов в Java не существует, а метод finalize() ни в коем случае нельзя считать аналогом деструктора.

Статические методы и поля

В Java (как и в C++) используются статические методы (англ. static method — в теории программирования их также называют методами класса), которые задаются при помощи ключевого слова static. Статические поля

(переменные класса) имеют тот же смысл, что и в C++: каждое такое поле является собственностью класса, поэтому для доступа к статическим полям не требуется создавать экземпляры соответствующего класса.

Например, математические функции, реализованные в классе Math, представляют собой как раз статические методы данного класса.

Завершённость (final)

Ключевое слово final (финальный) означает разные вещи при описании поля, метода или класса. Финальное поле класса инициализируется при описании, или в конструкторе класса (а статическое поле — в статическом блоке инициализации) и дальше его значение не может быть изменено. Значения локальных переменных, а также параметров метода, помеченных ключевым словом final, не могут быть изменены после присвоения. Но при этом, их значения могут использоваться внутри анонимных классов. Если статичное поле класса, или переменная проинициализированны константным выражением, они рассматриваются компилятором, как именованная константа; в таком случае их значение может быть использовано в операторах switch (для констант типа int), а также для условной компиляции (для констант типа boolean) при использовании с оператором if.

Метод класса, отмеченный словом final, не может быть переопределён при наследовании. Финальный класс не может иметь наследников вообще.

Абстрактность

В Java методы, не объявленные явно как static, final или private, являются виртуальными в терминологии C++: при вызове метода, по-разному определённого в базовом и наследующем классах, всегда производится проверка времени выполнения.

Абстрактным методом (модификатор abstract) в Java называется метод, для которого заданы параметры и тип возвращаемого значения, но не тело. Абстрактный метод определяется в классах-наследниках. В C++ то же самое называется чистой виртуальной функцией (pure). Для того чтобы в классе можно было описывать абстрактные методы, сам класс тоже должен быть описан как абстрактный. Объекты абстрактного класса создавать нельзя.

Интерфейсы

Высшей степенью абстрактности в Java является интерфейс (interface). Все методы интерфейса абстрактны: описатель abstract даже не требуется. Интерфейс в Java не считается классом, хотя по сути является полностью абстрактным классом. Класс может наследовать/расширять (extends) другой класс или реализовывать (implements) интерфейс. Кроме того, интерфейс может наследовать/расширять другой интерфейс.

В Java класс не может наследовать более одного класса, зато может реализовывать сколько угодно интерфейсов. Множественное наследование интерфейсов не запрещено, то есть один интерфейс может наследоваться от нескольких.

Интерфейсы можно передавать методам как параметры, но нельзя создавать экземпляры интерфейсов.

Маркерные интерфейсы

В Java есть некоторые интерфейсы, которые не содержат методов для реализации, а специальным образом обрабатываются JVM. Это интерфейсы:

- java.lang.Cloneable
- java.io.Serializable
- java.rmi.Remote

Шаблоны в Java (generics)

Начиная с версии Java 5 в языке появился механизм обобщённого программирования — шаблоны, внешне близкие к шаблонам C++. С помощью специального синтаксиса в описании классов и методов можно указать параметры-типы, которые внутри описания могут использоваться в качестве типов полей, параметров и возвращаемых значений методов.

```
// Объявление обобщённого класса

class GenericClass<E> {

 E getFirst() { ... }

 void add(E obj) { ... }
```

```
}

// Использование обобщённого класса в коде

GenericClass<String> var = new GenericClass<String>();

var.add("qwerty");

String p = var.getFirst();
```

Допускается обобщённое объявление классов, интерфейсов и методов. Кроме того, синтаксис поддерживает ограниченные объявления типов-параметров: указание в объявлении конструкции вида <T extends A & B & C...> требует, чтобы тип-параметр T реализовывал интерфейсы A, B, C и так далее, а конструкция <T super C> требует, чтобы тип-параметр T был типом C или одним из его предков.

В отличие от шаблонов C#, шаблоны Java не поддерживаются средой исполнения — компилятор просто создаёт байт-код, в котором никаких шаблонов уже нет. Реализация шаблонов в Java принципиально отличается от реализации аналогичных механизмов в C++: компилятор не порождает для каждого случая использования шаблона отдельный вариант класса или метода-шаблона, а просто создаёт одну реализацию байт-кода, содержащую необходимые проверки и преобразования типов. Это приводит к ряду ограничений использования шаблонов в программах на Java.

Проверка принадлежности к классу

В Java можно явно проверить, к какому классу принадлежит объект. Выражение foo instanceof Foo истинно, если объект foo принадлежит классу Foo или его наследнику, или реализует интерфейс Foo (или, в общем виде, наследует класс, который реализует интерфейс, который наследует Foo).

Далее, функция getClass(), определённая для всех объектов, выдаёт объект типа Class. Для каждого класса создаётся не более одного описывающего его объекта типа Class, поэтому эти объекты можно сравнивать. Так, например, foo.getClass() == bar.getClass() будет истинно, если объекты foo и bar принадлежат в точности к одному классу (но это не означает что это два одинаковых объекта).

Кроме того, объект типа Class любого типа можно получить так: Integer.class, Object.class.

Однако прямое сравнение классов не всегда является оптимальным средством проверки на принадлежность к классу. Зачастую вместо него используют функцию isAssignableFrom(). Эта функция определена у объекта типа Class и принимает объект типа Class в качестве параметра. Таким образом, вызов Foo.class.isAssignableFrom(Bar.class) вернёт true в случае, если Foo является предком класса Bar. Так как все объекты являются потомками типа Object, вызов Object.class.isAssignableFrom() всегда вернёт true.

В паре с упомянутыми функциями объекта типа Class используются также функции isInstance() (эквивалентно instanceof), а также cast() (преобразует параметр в объект выбранного класса).

Библиотеки классов

- JDK — содержит набор базовых классов для всего ПО на платформах Java SE и Java EE
- CDK — для создания химического ПО
- MARF — модульная библиотека для распознавания аудио
- JDOM — реализация XML DOM, учитывающая особенности платформы Java.

Средства разработки ПО

- JDK — помимо набора библиотек для платформ Java SE и Java EE содержит компилятор командной строки javac и набор утилит, также работающих в режиме командной строки.
- NetBeans IDE — свободная интегрированная среда разработки для всех платформ Java — Java ME, Java SE и Java EE. Пропагандируется Sun Microsystems, разработчиком Java, как базовое средство для разработки ПО на языке Java и других языках (C, C++, PHP, Fortran и др.).
- Eclipse IDE — свободная интегрированная среда разработки для Java SE, Java EE и Java ME[40]. Пропагандируется IBM, одним из важнейших разработчиков корпоративного ПО, как базовое средство для разработки ПО на языке Java и других языках (C, C++, Ruby, Fortran и др.)

- IntelliJ IDEA — среда разработки для платформ Java SE, Java EE и Java ME. Разработчик — компания JetBrains. Распространяется в двух версиях: свободной бесплатной (Community Edition) и коммерческой проприетарной (Ultimate Edition).
- JDeveloper — среда разработки для платформ Java SE, Java EE и Java ME. Разработчик — компания Oracle.
- BlueJ — Среда разработки программного обеспечения на языке Java, созданная в основном для использования в обучении, но также подходящая для разработки небольших программ.

2 АНАЛИЗ ЯЗЫКОВ ПРОГРАМИРОВАНИЯ

2.1 Python

Математический Python

Рассмотриваем различные библиотеки для языка Python, которые могут пригодиться для математических и инженерных задач:

SymPy - это активно развивающаяся библиотека для символьных вычислений в Python (новые версии выходят практически каждый месяц, а иногда и чаще). Очень интересная библиотека, причем благодаря большому количеству примеров в документации освоить ее довольно легко. Правда, это же является и недостатком. Практически вся документация представляет собой набор примеров, описание отдельных классов довольно скудное.

Кроме основных символьных операций вроде упрощения выражений, раскрытия скобок, вычисления пределов и разложения функций в ряд из дробей в пакет Sympy входят следующие модули:

- Модуль для работы с матрицами (модуль линейной алгебры).
- Модуль геометрии, с помощью которого можно символьно вычислять площадь геометрических фигур, находить точки пересечения прямых, отрезков и лучей.
- Модуль статистики, с помощью которого можно получать случайные величины с заданной функцией распределения плотности вероятности.
- Модуль для отображения трехмерных поверхностей, заданных в виде уравнений с символьными переменными

NumPy - это пакет, который просто необходимо ставить в первую очередь, если вы хотите заниматься вычислениями на Python.

Вот неполный список того, что есть внутри **NumPy**.

- Работа с матрицами и векторами
- Быстрое преобразование Фурье (одномерное и двумерное)
- Компиляция модулей на фортране
- Работа с полиномами (вычисление корней полинома, математические операции с полиномами и т.п.)

- Функции для линейной алгебры (вычисление определителя матрицы, вычисление обратных матриц, решение системы линейных уравнений и т.п.)

Matplotlib - это библиотека для построения графиков и визуализации данных. Кроме большого количества типов графиков, которые можно построить с помощью этого пакета, приятной особенностью **Matplotlib** является то, что функции для построения графиков напоминают функции Matlab. Здесь есть проблемы с отображением на графиках русских букв. Зато через Matplotlib можно выводить формулы в виде TeX. Графики, нарисованные с помощью **Matplotlib** можно масштабировать для просмотра интересующей области, причем как программно из скрипта, так и через интерфейс с помощью мыши.

Dislin - это многоязыковая библиотека для построения графиков. Среди поддерживаемых языков кроме Python числятся C, C++, C#, Visual Basic .NET, Java, Perl и Fortran. Работает библиотека очень быстро, а внешний вид настраивается довольно гибко. Правда, внешний вид можно менять только программно, даже трехмерные графики нельзя вращать мышкой, а координаты наблюдателя изменяются программно.

Но сначала хотелось бы сказать несколько слов про установку. После того как вы распакуете дистрибутив и инсталируете его, необходимо добавить еще несколько переменных окружения, чтобы Dislin мог работать из любой папки, а не только из папки Python, расположенной внутри папки установки Dislin. Для краткости будем считать, что Dislin установлен в папку c:\dislin. Вот какие изменения нужно проделать с переменными окружения:

- Создать переменную окружения DISLIN = c:\dislin
- Создать переменную окружения PYTHONPATH = c:\dislin\python
- В переменную окружения PATH добавить путь c:\dislin\win

Dislin может отображать следующие типы графиков:

- Функции одной переменной (в том числе и в полярной системе координат).
- Диаграммы в виде столбиков разной высоты (Bar), в том числе трехмерные.
- Диаграммы в виде секторов круга (Pie), в том числе трехмерные.

- Двумерные диаграммы функций от двух переменных, где значение функции определяет цвет в данной точке.
- Трехмерные графики функций двух переменных, где значение функции определяет положение точки по вертикальной оси (также можно сделать, чтобы значение функции определяло бы и цвет вершин).
- Построение линий уровня.
- Вывод формул в формате TeX.

2.2 Ruby

В языке Ruby осуществлён простой и удобный механизм для расширения языка с помощью библиотек, написанных на Си, позволяющий легко разрабатывать дополнительные библиотеки.

Для унифицированного доступа к базам данных разработана библиотека Ruby DBI (поддерживает SQLite, Oracle, ODBC, MySQL, DB2, MS SQL, InterBase, ADO и др.). Также существуют библиотеки для конкретных баз данных, поддерживающих специфические для них операции. Для реализации ORM существуют несколько библиотек, такие как ActiveRecord, Mongoid, DataMapper или Sequel. Из графических библиотек следует отметить FxRuby — интерфейс к графической библиотеке FOX, графический пакет разработчика wxRuby (интерфейс к кроссплатформенному пакету wxWidgets на C++), QtRuby/Korundum — привязка к Qt и KDE соответственно, графические библиотеки для работы с Tk и Gtk. Также реализована библиотека для работы с OpenGL, позволяющая программировать трёхмерную графику. Win32utils — позволяет обращаться к специфическим возможностям Win32 API. Rmagick — библиотека для работы с изображениями, поддерживающая более 90 форматов (основана на ImageMagick и GraphicsMagick). Библиотека Ruport (Ruby reports) предназначена для лёгкой реализации отчётов и создания диаграмм на основе данных из БД или прямо из текстовых файлов CSV. Причём результаты можно сохранять в форматах PDF, HTML, CSV и TXT. RuTils — обработчик русского текста на Ruby. Позволяет реализовать сумму прописью и выбор числительного. Например, 231.propisju(2) => «двести тридцать одна» или 341.propisju_items(1, «чемодан», «чемодана», «чемоданов») => «триста сорок один чемодан». А также перевод в транслит и работу с датами. Для управления библиотеками и программами Ruby в виде самодостаточных пакетов предназначена система управления пакетами RubyGems (англ. gems,

gem — драгоценный камень). Существует всемирный репозиторий программного обеспечения Ruby RAA (Ruby Application Archive). Репозиторий по состоянию на сентябрь 2007 года насчитывает более полутора тысяч проектов. Большое количество программного обеспечения, написанного на Ruby, пользуются хостингом проекта RubyForge, созданного специально с этой целью.

2.3 MATLAB

Математические функции

В системе MATLAB имеется обширная библиотека математических функций. Каждой функции соответствует определенное имя. Функция ставит в соответствие значениям своих аргументов значение результата. Аргументы функции всегда указываются в круглых скобках после имени функции и, если их больше одного, разделяются запятыми. В качестве аргументов могут использоваться другие функции и любые выражения языка MATLAB (при условии соответствия типов аргументов).

Элементарная математическая функция - это, как правило, функция от одной переменной, и в этом случае устанавливается соответствие между массивами значений аргумента и результата. Аргумент указывается в круглых скобках после имени функции. Имя переменной, которой присваивается значение функции, располагается слева от знака равенства. Если имя присваиваемой переменной не указано, значение функции присваивается служебной переменной ans. Тип результата вычисления математической функции всегда совпадает с типом ее аргумента. Например, если аргументом функции является вектор-столбец, то значением этой функции также будет вектор-столбец. Рассмотрим встроенные математические функции системы MATLAB, которые применяются к числам, скалярным переменным и к массивам (поэлементно).

Базовые функции

- ABS - абсолютное значение
- ANGLE - аргумент комплексного числа
- REAL, IMAG - действительная и мнимая части комплексного числа
- CONJ - операция комплексного сопряжения

- SIGN - вычисление знака числа
- CEIL, FIX, FLOOR, ROUND - функции округления
- REM - функция остатка
- GCD - наибольший общий делитель
- LCM - наименьшее общее кратное
- RAT, RATS - представление результата в виде рационального числа или цепной дроби

Трансцендентные функции

- SQRT - квадратный корень
- EXP - экспоненциальная функция
- LOG - функция натурального логарифма
- POW2 - экспонента по основанию 2
- NEXTPOW2 - ближайшая степень по основанию 2
- LOG2 - фунции логарифма
- LOG10 - функции логарифма

Тригонометрические функции

- SIN, SINH - функции синуса
- ASIN, ASINH - функции обратного синуса
- CSC, CSCH - функции косеканса
- ACSC, ACSCH - функции обратного косеканса
- COS, COSH - функции косинуса
- ACOS, ACOSH - функции обратного косинуса
- SEC, SECH - функции секанса
- ASEC, ASECH - функции обратного секанса
- TAN, TANH - функции тангенса
- ATAN, ATAN2, ATANH - функции обратного тангенса
- COT, COTH - функции котангенса
- ACOT, ACOTH - функции обратного котангенса

Преобразования системы координат

- CART2POL - преобразование декартовой системы координат в полярную и цилиндрическую
- CART2SPH - преобразование декартовой системы координат в сферическую

- POL2CART - преобразование полярной и цилиндрической систем координат в декартову
- SPH2CART - преобразование сферической системы координат в декартову

Специальные функции

- BESSEL - функции Бесселя
- BETA, BETACORE, BETAINC, BETALN - бета-функции
- ELLIPJ - эллиптические функции Якоби
- ELLIPKE - полные эллиптические интегралы
- ERF, ERFCORE, ERFC, ERFCX, ERFINV - функции ошибок
- GAMMA, GAMMAINC, GAMMALN - гамма-функции

2.4 JAVA

Основы проекта Java

Целью проекта было создание небольшой, надежной, переносимой и распределенной системы реального времени. Исходно в качестве языка планировалось использование языка программирования C++, но постепенно связанные с ним трудности привели к необходимости создания нового языка. Система должна была вобрать в себя лучшие черты из множества современных систем программирования -- Eiffel, Smalltalk, Objective C, Cedar/Mesa. Стремительный рост сетевых технологий привел к необходимости нового взгляда на процесс создания и распределения приложений. Современные приложения должны быть безопасны, высокопроизводительны, работать в распределенной среде на множестве машин различной архитектуры.
Требования к переносимости заставили отказаться от традиционного способа создания и доставки бинарных файлов, содержащих машинные коды и, следовательно, привязанных к определенной платформе. Сегодня, чтобы выжить в этих джунглях из архитектур процессоров, операционных систем и графических оболочек, приложение должно быть нейтрально к архитектуре и динамически настраиваемо.
Созданная система разработки Java удовлетворяет всем этим требованиям, а следовательно:

- проста, поэтому может быть использована широким кругом разработчиков
- объектно ориентирована, что соответствует современному взгляду на программирование
- поддерживает многопотоковость, что позволяет выполнять несколько задач одновременно
- интерпретируема, что обеспечивает переносимость и улучшает динамические свойства
- Чтобы не быть голословным, рассмотрим каждое из этих свойств по отдельности.

Свойства Java, краткий обзор.

Java проста, объектно ориентированна и знакома
Система Java создана на основе *простого* языка программирования, техника использования которого близка к общепринятой и обучение которому не требует значительных усилий.
Java как язык программирования является объектно ориентированной с момента основания. Кроме того программист с самого начала обеспечивается набором *стандартных* библиотек, обеспечивающих функциональность от стандартного ввода/вывода и сетевых протоколов до графических пользовательских интерфейсов. Эти библиотеки легко могут быть расширены.
Несмотря на то, что язык C++ был отвергнут, синтаксис языка Java максимально приближен к синтаксису C++. Это делает язык знакомым широкому кругу программистов. В то же время из языка были удалены многие свойства, которые делают C++ излишне сложным для пользования, не являясь абсолютно необходимыми. В результате язык Java получился более простым и органичным, чем C++.
Надежность и безопасность
Java существенно облегчает создание надежного программного обеспечения. Кроме исчерпывающей проверки на этапе компиляции, система предусматривается анализ на этапе выполнения. Сам язык спроектирован так, чтобы вырабатывать у программиста привычку писать "правильно". Модель работы с памятью, в которой исключено использование указателей, делает невозможными целый класс ошибок, характерных для C и C++. В силу того, что Java предназначена для работы в распределенной среде, безопасность становится чрезвычайно важной проблемой. Требования безопасности определяют многие черты как языка, так и реализации всей системы.

Независимость от архитектуры и переносимость. Компилятор Java производит байт-коды, т.е. модули приложения имеют архитектурно-независимый формат, который может быть проинтерпретирован на множестве разнообразных платформ. Это уже не исходные тексты, но еще не платформно-зависимые машинные коды. Следующий шаг -- "замораживание" стандарта на формат основных встроенных типов данных. Программа, созданная на одной платформе, работает на всех остальных.
Этот стандарт фиксирован в документе, описывающем Java Virtual Machine. Стандарт может быть реализован на любой аппаратно-программной платформе, поддерживающей многопотоковость.
Производительность
Схема работы системы и набор байт-кодов виртуальной машины Java таковы, что позволяют достичь высокой производительности на этапе выполнения программы:

- анализ кодов на соблюдение правил безопасности производится один раз до запуска кодов на выполнение, в момент выполнения таких проверок уже не нужно, и коды выполняются максимально эффективно
- работа с базовыми типами максимально эффективна, для операций с ними зарезервированы специальные байт-коды
- методы в классах не обязательно связываются динамически
- автоматический сборщик мусора работает отдельным фоновым потоком, не замедляя основную работу программы, но в то же время обеспечивая своевременный возврат свободной памяти в систему
- стандарт предусматривает возможность написания критических по производительности участков программы в машинных кодах

Интерпретируемый, многопотоковый и динамический Интерпретируемая природа языка позволяет сделать фазу линкования простой, инкрементальной и, следовательно, быстрой. Это резко сокращает цикл разработки и тестирования программных фрагментов. Многопотоковость позволяет выполнять в рамках одного приложения несколько задач одновременно. Это становится особенно актуально в современных распределенных приложениях, когда процессы сетевого обмена могут идти одновременно и асинхронно. При этом программа продолжает реагировать на ввод информации пользователем без неприятных задержек. Многопотоковость поддерживается на уровне языка -- часть примитивов синхронизации встроена в систему реального времени, а библиотека содержит базовый класс Thread. К тому же системные библиотеки написаны

thread-safe, т.е. все они могут быть использованы в многопотоковых приложениях.
Система обеспечивает динамическую сборку программы. Классы подгружаются по мере необходимости, причем загружены они могут быть с любой точки сети, что позволяет сделать внесение изменений в приложения прозрачным для пользователя. Пользователь может быть уверен, что всегда работает со свежей версией приложения.

Базовая система Java

Опыт показывает, что отсутствие стандартных базовых библиотек для языка C++ чрезвычайно затрудняет работу с ним. В силу того, что любое нетривиальное приложение требует наличия некоторого набора базовых классов, разработчикам приходится пользоваться различными несовместимыми между собой библиотеками или писать свой собственный вариант такого набора. Все это затрудняет как разработку, так и дальнейшую поддержку приложений, затрудняет стыковку приложений, написанных разными людьми.
Полная система Java включает в себя готовый набор библиотек, который можно разбить на следующие пакеты:

- java.lang -- базовый набор типов, отраженных в самом языке. Этот пакет обязательно входит в состав любого приложения. Содержит описания классов Object и Class, а также поддержку многопотоковости, исключительных ситуаций, оболочку для базовых типов, а также некоторые фундаментальные классы.
- java.io -- потоки и файлы произвольного доступа. Аналог библиотеки стандартного ввода-вывода системы UNIX. Поддержка сетевого доступа (sockets, telnet, URL) содержится в пакете java.net.
- java.util -- классы-контейнеры (Dictionary, HashTable, Stack) и некоторые другие утилиты. Кодирование и декодирование. Классы Date и Time.
- java.awt -- Abstract Windowing Toolkit, архитектурно-независимый оконный интерфейс, позволяющий запускать интерактивные оконные Java-приложения на любой платформе. Содержит базовые компоненты интерфейса, такие как события, цвета, фонты, а также основные оконные элементы -- кнопки, scrollbars и т.д..

Результат -- новый подход к распределенным вычислениям

Каждая из перечисленных характеристик по отдельности может быть найдена в уже существующих программных пакетах. Новым является соединение их в стройную непротиворечивую систему, которая должна стать всеобщим стандартом.

Основные свойства языка программирования Java

Встроенные (примитивные) типы данных
В языке Java, так же как и в C++, существует набор встроенных типов данных, которые (так же как и в C++) не являются объектами. Набор их также сходен с набором базовых типов C++ за некоторыми исключениями.

Numeric
Характерным отличием от C++ является то, что бинарное представление чисел отныне фиксировано:

целые числа: 8-бит byte, 16-бит short, 32-бит int, 64-бит long. Все числа со знаком, ключ unsigned из языка удален.

числа с плавающей точкой. 32-бит float, 64-бит double. Представление должно соответствовать стандарту IEEE 754.

Character
Отличаются от C++ как синтаксисом, так и представлением. Тип character есть 16-разрядное число без знака (диапазон 0-65,535). Кодировка соответствует стандарту Unicode. В силу того, что эта кодировка в идеале должна охватывать все существующие в мире языки, это представление должно облегчить локализацию приложений.

Boolean
Этот тип данных не выделен в C++, однако неявно присутствует практически во всех программах. В Java тип называется boolean, может принимать значения true и false и не может (в отличие от C++) быть преобразован в другой тип.
Операторы
Добавлен новый оператор >>> логического сдвига вправо (т.к. нет беззнаковых целых чисел). Встроенная операция слияния строк (оператор +).
Массивы
В отличие от C++ массивы в Java являются полноценными объектами с определенным runtime представлением. Декларация:

Point myPoints[];

резервирует ссылку на массив, а не место под реальный объект. Сам массив может быть затем создан выполнением

myPoints = new Point[10];

а его элементы заполнены операцией типа:

myPoints[2] = new Point();

Размер массива может быть получен во время выполнения программы:

howMany = myPoints.length;

Значение индекса проверяется при каждом обращении, при ошибке возбуждается исключительная ситуация.
Указатели полностью исключены из языка вместе с целой категорией трудноуловимых ошибок "замедленного действия". К тому же наличие указателей противоречит требованиям безопасности и усложняет реализацию сборщика мусора.
Strings
Строки в Java являются полноценными объектами. Они делятся на текстовые константы (Strings) и модифицируемые строки (StringBuffer). Компилятор позволяет явно определять текстовые литералы в программе подобно тому, как это делается в C++

String hello = "Hello world!";

Ссылка hello инициируется объектом класса String на основе представления "Hello world!" в кодировке Unicode.
Оператор "+" может быть применен к строкам, например

System.out.println("There are" + num + "characters in the file.");

Multi-Level Break
В Java отсутствует выражение goto. Анализ C/C++ текстов показал, что подавляющее число случаев использования этого оператора связано с необходимостью выхода из вложенного цикла. Для отработки таких

ситуаций в Java перед началом блока может ставиться метка, а инструкции break и continue также могут сопровождаться меткой, на которую должен быть осуществлен переход. Например:

```
test:
    for(int i = 0; i < 10; i++)
      for(int j = 0; j < 10; j++)
       if( i > 3)
         break test;
```

Управление памятью, сборка мусора

Необходимость явно управлять памятью в C/C++ программах всегда была большой занозой для программистов. Мало того, что сами программы изобиловали вызовами функции free или операторами delete, непосредственно к логике программы отношения не имеющими. Ошибки, связанные с неосвобождением памяти или наоборот, с удалением уже однажды удаленных объектов, относятся к категории ошибок наиболее трудных для обнаружения и исправления.

Java полностью снимает эту заботу с программиста. Автоматический сборщик мусора обязан быть встроен в run-time системы. Память объектов, на которые больше нет ссылок, в конце концов возвращается в систему. Опыт показывает, что несмотря на относительную сложность сборщиков мусора, производительность системы в целом может оказаться не меньше, а часто и больше, чем при явном освобождении памяти программой.

Сборка мусора в фоновом режиме

Одно из преимуществ того, что Java-приложения многопотоковые, заключается в том, что сборка мусора может производиться в фоновом потоке. Этот поток имеет меньший приоритет выполнения, чем остальные, поэтому система всегда готова ответить на действия пользователя, отсутствуют "периоды молчания", в которые производится только сборка мусора. С другой стороны, паузы в операциях пользователя сборщик мусора может использовать для своей работы, обеспечивая наличие свободной памяти в моменты, когда это необходимо.

Встроенная синхронизация потоков

Java поддерживает многопотоковость не только на уровне библиотек, но и на

уровне самого языка, что значительно облегчает построение приложений, надежно работающих в многопотоковом режиме.

Язык Java объектно-ориентирован

Система Java создавалась объектно ориентированной с самого начала. Объектно-ориентированная парадигма наиболее удобна при создании программного обеспечения типа клиент-сервер, а также для организации распределенных вычислений.
Одна из черт, присущих объектам, заключается в том, что объекты обычно переживают процедуру, их создающую. Они затем могут перемещаться по сети, храниться в базах данных и т.д.
Идейными наследниками Java являются такие языки, как C++, Eiffel, Smalltalk и Objective C. За исключением примитивных типов данных, практически все в языке является объектом.

Основные требования к объектно-ориентированной системе

- инкапсуляция -- сокрытие реализации за абстрактным интерфейсом
- полиморфизм -- одно и то же сообщение, посланное различным объектам, приводит к выполнению разных операций
- наследование -- новые классы могут наследовать данные и функциональность уже существующих классов
- динамическое связывание -- новые классы могут появляться в системе откуда угодно, в том числе и из сети. Необходимо иметь возможность динамически включать их в систему

Интеллектуальность

Система Java предназначена для создания программного обеспечения, которое должно быть интеллектуальным, предельно надежным и безопасным по множеству параметров. Особое внимание уделяется как ранней диагностике возможных проблем, так и поздней, во время выполнения кодов.
Жесткая проверка на этапе компиляции и во время выполнения
Компиляция с языка Java предусматривает жесткую проверку исходных текстов, множество ошибок может быть выявлено уже на этом этапе.
Одним из преимуществ языка C++ как строго типизованного языка является возможность раннего выявления некоторых категорий ошибок. Однако во многом этот язык наследует свойства C, позволяя нарушать требования строгого объявления функций и методов. Язык Java требует явного

объявления прототипов и не поддерживает характерных для C неявных преобразований.
Значительное число проверок, производимых компилятором, повторяются виртуальной машиной непосредственно перед выполнением приложения. Линкер получает всю информацию о прототипах методов и на основе ее производит такую же проверку, как и компилятор, позволяя избежать расхождений в версиях между отдельными модулями.
Наиболее существенное отличие языка Java от C или C++ заключается в том, что архитектура Java не позволяет случайно или намеренно повредить память программы. Вместо арифметики указателей Java использует полноценные объекты для массивов и строк, что позволяет контролировать индексы доступа к ним во время выполнения. Кроме того, невозможны превращения между целыми числами и указателями.
Естественно, что все это не может полностью гарантировать программиста от любых ошибок, однако, Java устраняет целый класс их, существенно облегчая задачу разработчика.

Интерпретируемый и динамический

Для разработчика, использующего в своей работе обычные компилируемые языки, цикл разработки обычно выглядит следующим образом: редактировать текст -- скомпилировать его -- собрать программу линкером -- загрузить -- довести ее до "падения" -- рассмотреть обломки -- начать все с начала.
Кроме того, приходится постоянно следить за тем, какие из исходных текстов подлежат перекомпиляции. Для этого обычно используются дополнительные инструменты (например, популярная утилита make), часто не связанные с конкретным языком программирования и использующие крайне консервативный подход -- перекомпилировано должно быть все, что теоретически могло быть затронуто изменением. По мере того, как исходные тексты приложения разрастаются до сотен тысяч строк, взаимозависимости связывают части проекта крепче и крепче, скорость разработки приближается к нулю.
Система Java в силу своей интерпретируемой и динамической природы значительно более подходит для целей быстрой разработки надежных программ.
Как уже было отмечено выше, на выходе компилятора Java мы получаем байт-коды для Виртуальной Машины Java. Полная спецификация виртуальной машины открыта и общедоступна. Она может быть реализована практически на любой из современных программно-аппаратных платформ.

После этого программы на языке Java могут быть собраны из любых мест в сети и работать на этой платформе так же, как и на любой другой. Процесс сборки программы (linking) существенно ускорен по сравнению с обычными компилируемыми системами. Он представляет собой подгрузку необходимых классов и производится инкрементально, т.е. недостающие части подгружаются по мере надобности, что также приводит к сокращению времени цикла разработки.

Динамическая загрузка и связывание

То, что Java является интерпретатором, позволяет расширять систему динамически. Отдельные классы загружаются лишь по мере необходимости и могут быть собраны из различных мест в сети. Перед запуском на выполнение коды проходят жесткую проверку.
В настоящее время объектно-ориентированный подход стал общепринятым. В качестве языка программирования при этом обычно выбирают C++. Однако этот язык обладает определенным недостатком, который известен под названием "проблемы хрупкости базового класса" (fragile superclass problem).
"Проблема хрупкости базового класса" в C++
Эта проблема возникает как побочный эффект реализации модели C++. Каждый раз, когда Вы добавляете новый метод или переменную в класс, все остальные модули приложения, использующие этот класс, требуют перекомпиляции. В противном случае программа успешно собирается, а при запуске так же успешно разваливается. Даже при использовании специальных утилит типа make неточное отслеживание взаимозависимостей между классами является неиссякающим источником ошибок. Эта проблема "хрупкости базового класса" также часто именуется как проблема "постоянной перекомпиляции". Избежать ее можно путем разнообразных уловок, обычно связанных с отказом от прямого использования объектно-ориентированных свойств языка.
Решение "проблемы хрупкости базового класса"
В системе программирования Java эта проблема решается в несколько этапов. Во-первых, компилятор не разрешает ссылок вплоть до численных значений. Напротив, символьная информация передается вместе с байт-кодами для проверки и интерпретации. Окончательное связывание имен производится интерпретатором в момент загрузки класса. После этого ссылки уже "прописаны", как непосредственные указатели, и интерпретатор может работать с нормальной скоростью.
Во вторых то, как объект должен выглядеть в памяти машины, определяется

не компилятором, а самим интерпретатором. Добавление в класс новых переменных или методов не требует изменений в остальных кодах.

Понятие интерфейса в языке Java

Под интерфейсом (interface) в Java понимается спецификация дополнительного набора методов, которые "обязан знать" объект. Идея заимствована из языка Objective C, где такая спецификация называется протоколом (protocol). Интерфейс не включает в себя модифицируемых переменных или выполняемых кодов. Класс может реализовать любое количество интерфейсов, без всех трудностей организации иерархии классов при множественном наследовании в C++.

Представление в исполняемом модуле

Классы в Java реально представлены в работающей системе. Существует выделенный класс по имени Class, экземпляры которого создаются виртуальной машиной и содержат информацию о всех классах в системе. Для любого объекта возможно найти соответствующий ему объект, представляющий его класс. Класс может сообщить свое имя и ссылку на своего непосредственного предшественника в иерархии. Возможен также поиск классов по имени.

Итоги

- Интерпретируемая и динамическая природа языка Java предоставляет разработчику определенные преимущества:
- интерпретирующее окружение позволяет быстрое создание прототипов без обычного цикла перекомпиляции и сборки
- среда динамически расширяема, т.к. классы подгружаются на лету по мере необходимости
- характерная для C++ проблема "хрупкого базового класса" решена в силу того, что расположение элементов объекта в памяти определяется не на этапе компиляции, а на этапе выполнения

Безопасность в Java

По мере стремительного роста использования глобальных сетей в спектре услуг, простирающемся от электронного распространения программного обеспечения и объектов multimedia до электронных платежей, безопасность становится ключевой проблемой. Мы коснемся того, как компилятор Java и run-time предотвращают создание и проникновение "диверсионных" кодов.

Компилятор и run-time включают в себя несколько уровней обороны против

потенциально опасных программ. В общем случае система исходит из предположения, что доверять нельзя никому. Следующие несколько секций касаются проблемы более детально.

Резервирование и распределение памяти

Во-первых, решение о распределении памяти принимает не компилятор, а run-time система. Оно может зависеть от особенностей архитектуры конкретной системы.
Во-вторых, язык не поддерживает указателей. Символические ссылки на объекты разрешаются интерпретатором на этапе выполнения. Выделение памяти и работа со ссылками находятся полностью под управлением системы и не доступны непосредственно из программы.
Отложенное до последнего момента размещение структур в памяти не позволяет определить реальное положение полей класса по его описанию.

Процесс проверки байт-кодов

Несмотря на то, что компилятор гарантирует, что коды не нарушают требований безопасности, если они были получены из других точек сети возникает следующая проблема: коды могут быть созданы не компилятором Java, а другими средствами. Или они могут быть намеренно модифицированы после создания. Поэтому run-time система подвергает полученные коды тщательной проверке.
Проверка включает в себя несколько этапов, начиная с контроля целостности формата полученного файла до анализа каждого фрагмента кодов на предмет выполнения следующих правил:

нет незаконных манипуляций с указателями

нет попыток нарушения прав доступа

объекты используются в строгом соответствии с их типами, например, объекты класса InputStream используются только как InputStream и никак иначе.

Верификатор байт-кодов
Верификатор байт-кодов (bytecode verifier) сканирует байт-коды, извлекает информацию о типах объектов в каждой точке выполнения фрагмента кода.
На приведенном рисунке изображен путь от исходных текстов на языке Java через компилятор Java, верификатор кодов до интерпретатора. Важно

отметить, что загрузчик и верификатор байт-кодов не делают никаких предположений относительно происхождения кодов -- получены они с локальной файловой системы или с другого континента. Верификатор гарантирует, что любой код, прошедший проверку, может быть использован интерпретатором без риска повредить его (интерпретатор), а именно:

- не может произойти переполнение или "исчерпание" стека
- параметры для инструкций байт-машины имеют нужный тип
- доступ к полям и методам объектов не нарушает объявленных в классе правил (public, private, protected)

Правила безопасности при загрузке

В ходе выполнения программы может потребоваться загрузка дополнительных классов. После того как, полученный код прошел проверку на валидность байт-кодов, он поступает в загрузчик кодов. Для загрузчика все пространство имен загружаемых классов может быть подразделено на отдельные области (name spaces). Причем классы, полученные локально (заслуживающие безусловного доверия), и классы, присланные по сети из остального мира (и потенциально враждебные), находятся в разных имен. пространствах
При разрешении ссылки на какой-либо класс он ищется прежде всего в локальном пространстве. Это не позволяет "внешним" кодам подменить один из базовых классов в системе.

Безопасность в сетевом пакете

Сетевой пакет Java включает в себя поддержку различных сетевых протоколов (FTP, HTTP, Telnet и т.д.). Это -- передовая линия защиты от по сети. вторжения
Осторожность при установке прав сетевого доступа в локальную систему может быть доведена до параноидальной. Вы можете

- запретить сетевой доступ вообще
- разрешить доступ только к тому узлу, с которого был получен код
- разрешить доступ только к узлам за пределами firewall, если код также получен с той стороны firewall
- разрешить любой сетевой доступ

Многопотоковость в Java

Современного пользователя компьютера все чаще раздражает ситуация, когда программа способна выполнять в один момент времени лишь одну задачу. Реальный мир наполнен событиями, происходящими одновременно и независимо. Пользователь требует от компьютера адекватной реакции.
К сожалению, написание программ, отвечающих этим требованиям, значительно сложнее, чем написание программ, выполняющихся последовательно. Они могут быть созданы с использованием C или C++, однако делать это сложнее, т.к. отсутствует поддержка в самом языке, а также большинство существующих на сегодняшний день внешних библиотек часто не могут быть использованы в таких программах в силу того, что они не удовлетворяют так называемому thread-safe условию. Термин thread-safe означает, что каждая функция данной библиотеки может быть использована одновременно несколькими потоками.
Основная проблема при прямом управлении потоками состоит в том, что Вы никогда не можете быть полностью уверены, что поставили все нужные замки (locks) и вовремя их освободили. При преждевременном завершении процедуры или при возникновении исключительной ситуации замок может остаться неснятым, что обычно приводит к блокировке программы (deadlock).
Встроенная многопотоковость -- существенная черта архитектуры Java. Стандартная библиотека включает в себя класс Thread, с методами, позволяющими стартовать новый поток, завершить его работу и проверить текущее состояние потока.
Интеграция примитивов синхронизации непосредственно в язык упрощает работу с ними.
Потоки в Java вытесняющие (pre-emptive), а также могут выполняться в режиме разделения времени (time-sliced), но только на платформах, которые поддерживают это. В системах, в которых такая поддержка отсутствует, после того, как поток был запущен, он может быть прерван только другим потоком с более высоким приоритетом. Если ваше приложение требует больших периодов вычислений, рекомендуется явно отдавать управление другим потокам (вызовом Thread.yield()).

Интегрированная синхронизация потоков

Система Java содержит поддержку многопотоковости как на уровне синтаксиса языка, так и на уровне библиотек и системных вызовов. На уровне самого языка методы, объявленные с признаком synchronized, гарантировано не будут выполняться одновременно для данного объекта.

Методы запускаются под управлением монитора (monitor). Каждый класс и объект имеют свой собственный монитор. Если объект находится в состоянии выполнения одного из синхронизованных методов, попытка вызвать этот метод или любой другой синхронизованный метод для этого же объекта из другого потока приостанавливается до того момента, когда выполнение метода каким-то образом завершится (обычным образом или в результате возбуждения исключительной ситуации).

Математические функции в Java

Math class methods

Method	Description	Example

`abs( x )`	absolute value of x	`abs( . ) is 23.7`
		`abs( . ) is 0.0`
		`abs( - . ) is 23.7`

`ceil( x )`	rounds x to the smallest integer not less than x	`ceil( . ) is 10.0` `ceil( . ) is -9.0`
`cos( x )`	trigonometric cosine of x (x in radians)	`cos( . ) is 1.0`
`exp( x )`	exponential method e^x	`exp( . ) is 2.71828` `exp( . ) is 7.38906`
`floor( x )`	rounds x to the largest integer not greater than x	`floor( . ) is 9.0` `floor( - . ) is -10.0`
`log( x )`	natural logarithm of x (base e)	`log( . ) is 1.0` `log( . * . ) is 2.0`
`max( x, y )`	larger value of x and y	`max( . , . ) is 12.7` `max( - . , - . ) is -2.3`
`min( x, y )`	smaller value of x and y	`min( . , . ) is 2.3` `min( - . , - . ) is -12.7`
`pow( x, y )`	x raised to the power y (i.e., x^y)	`pow( . , . ) is 128.0` `pow( . , . ) is 3.0`
`sin( x )`	trigonometric sine of x (x in radians)	`sin( . ) is 0.0`

Math class methods

Method	Description	Example
`sqrt( x )`	square root of x	`sqrt( . . )` is `30.0`
`tan( x )`	trigonometric tangent of x (x in radians)	`tan( . )` is `0.0`

Таблица 2

Математические примеры в Java

1. Нахождение максимума

```
1  // Fig. 6.3: MaximumFinder.java
2  // Determine the maximum of three double values.
3  import java.util.Scanner;
4
5  public class MaximumFinder
6  {
7     // obtain three floating-point values and locate the
maximum value
8     public void determineMaximum()
9     {
10       // create Scanner for input from command window
11       Scanner input = new Scanner( System.in );
12
13       // obtain user input
14       System.out.print(
15          "Enter three floating-point values separated by
spaces: " );
16       double number1 = input.nextDouble(); // read first
double
17       double number2 = input.nextDouble(); // read second
double
18       double number3 = input.nextDouble(); // read third
double
19
20       // determine the maximum value
21       double result = maximum( number1, number2, number3 );
22
23       // display maximum value
```

```
24          System.out.println( "Maximum is: " + result );
25       } // end method determineMaximum
26
27       // returns the maximum of its three double parameters
28       public double maximum( double x, double y, double z )
29       {
30          double maximumValue = x; // assume x is the largest to
start
31
32          // determine whether y is greater than maximumValue
33          if ( y > maximumValue )
34             maximumValue = y;
35
36          // determine whether z is greater than maximumValue
37          if ( z > maximumValue )
38             maximumValue = z;
39
40          return maximumValue;
41       } // end method maximum
42    } // end class MaximumFinder
```

```
1  // Fig. 6.4: MaximumFinderTest.java
2  // Application to test class MaximumFinder.
3
4  public class MaximumFinderTest
5  {
6     // application starting point
7     public static void main( String args[] )
8     {
9        MaximumFinder maximumFinder = new MaximumFinder();
10       maximumFinder.determineMaximum();
11    } // end main
12 } // end class MaximumFinderTest
```

```
Enter three floating-point values separated by spaces: 9.35 2.74
5.1
Maximum is: 9.35
```

```
Enter three floating-point values separated by spaces: 5.8 12.45
8.32
Maximum is: 12.45
```

```
Enter three floating-point values separated by spaces: 6.46 4.12
10.54
Maximum is: 10.54
```

2. GradeBook

```
1  // Fig. 7.11: GradeBook.java
2  // GradeBook using an array to store test grades.
3
4  public class GradeBook
5  {
6     private String courseName; // name of course this
   gradebook represents
7     private int grades[]; // array of student grades
8
9     // two-argument constructor initializes courseName and
   grades array
10    public GradeBook( String name, int gradesArray[] )
11    {
12       courseName = name; // initialize courseName
13       grades = gradesArray; // store grades
14    } // end two-argument GradeBook constructor
15
16    // method to set the course name
17    public void setCourseName( String name )
18    {
19       courseName = name; // store the course name
20    } // end method setCourseName
21
22    // method to retrieve the course name
23    public String getCourseName()
24    {
25       return courseName;
26    } // end method getCourseName
27
28    // display a welcome message to the GradeBook user
29    public void displayMessage()
30    {
31       // getCourseName gets the name of the course
32       System.out.printf( "Welcome to the grade book
   for\n%s\n",
33          getCourseName() );
34    } // end method displayMessage
```

```
35
36       // perform various operations on the data
37       public void processGrades()
38       {
39          // output grades array
40          outputGrades();
41
42          // call method getAverage to calculate the average
grade
43          System.out.printf( "\nClass average is %.2f\n",
getAverage() );
44
45          // call methods getMinimum and getMaximum
46          System.out.printf( "Lowest grade is %d\nHighest grade
is %d\n\n",
47             getMinimum(), getMaximum() );
48
49          // call outputBarChart to print grade distribution
chart
50          outputBarChart();
51       } // end method processGrades
52
53       // find minimum grade
54       public int getMinimum()
55       {
56          int lowGrade = grades[ 0 ]; // assume grades[ 0 ] is
smallest
57
58          // loop through grades array
59          for ( int grade : grades )
60          {
61             // if grade lower than lowGrade, assign it to
lowGrade
62             if ( grade < lowGrade )
63                lowGrade = grade; // new lowest grade
64          } // end for
65
66          return lowGrade; // return lowest grade
67       } // end method getMinimum
68
69       // find maximum grade
70       public int getMaximum()
71       {
72          int highGrade = grades[ 0 ]; // assume grades[ 0 ] is
largest
73
74          // loop through grades array
```

```
75          for ( int grade : grades )
76          {
77             // if grade greater than highGrade, assign it to
highGrade
78             if ( grade > highGrade )
79                highGrade = grade; // new highest grade
80          } // end for
81
82          return highGrade; // return highest grade
83       } // end method getMaximum
84
85       // determine average grade for test
86       public double getAverage()
87       {
88          int total = 0; // initialize total
89
90          // sum grades for one student
91          for ( int grade : grades )
92             total += grade;
93
94          // return average of grades
95          return (double) total / grades.length;
96       } // end method getAverage
97
98       // output bar chart displaying grade distribution
99       public void outputBarChart()
100      {
101         System.out.println( "Grade distribution:" );
102
103         // stores frequency of grades in each range of 10
grades
104         int frequency[] = new int[ 11 ];
105
106         // for each grade, increment the appropriate
frequency
107         for ( int grade : grades )
108            ++frequency[ grade / 10 ];
109
110         // for each grade frequency, print bar in chart
111         for ( int count = 0; count < frequency.length;
count++ )
112         {
113            // output bar label ( "00-09: ", ..., "90-99: ",
"100: " )
114            if ( count == 10 )
115               System.out.printf( "%5d: ", 100 );
116            else
```

```
117               System.out.printf( "
118                  count * , count *   + );
119
120               // print bar of asterisks
121               for ( int stars = 0; stars < frequency[ count ];
stars++ )
122                  System.out.print( "*" );
123
124               System.out.println(); // start a new line of
output
125            } // end inner for
126         } // end method outputBarchart
127
128         // output the contents of the grades array
129         public void outputGrades()
130         {
131            System.out.println( "The grades are:\n" );
132
133            // output each student's grade
134            for ( int student = 0; student < grades.length;
student++ )
135               System.out.printf( "Student %2d: %3d\n",
136                  student + 1, grades[ student ] );
137         } // end method outputGrades
138      } // end class GradeBook
```

```
1  // Fig. 7.12: GradeBookTest.java
2  // Creates GradeBook object using an array of grades.
3
4  public class GradeBookTest
5  {
6     // main method begins program execution
7     public static void main( String args[] )
8     {
9        // array of student grades
10       int gradesArray[] = { 87, 68, 94, 100, 83, 78, 85, 91,
76, 87 };
11
12       GradeBook myGradeBook = new GradeBook(
13          "CS101 Introduction to Java Programming",
gradesArray );
14       myGradeBook.displayMessage();
15       myGradeBook.processGrades();
16    } // end main
17 } // end class GradeBookTest
```

```
Welcome to the grade book for
CS101 Introduction to Java Programming!

The grades are:

Student  1:  87
Student  2:  68
Student  3:  94
Student  4: 100
Student  5:  83
Student  6:  78
Student  7:  85
Student  8:  91
Student  9:  76
Student 10:  87

Class average is 84.90
Lowest grade is 68
Highest grade is 100

Grade distribution:
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
  100: *
```

3. Математические функции

```
1
2
3  import static java.lang.Math.*;
4
5  public class StaticImportTest
6  {
7     public static void main( String args[] )
```

```
   {
      System.out.printf( "sqrt( 900.0 ) = %.1f\n", sqrt(
900.0 ) );
      System.out.printf( "ceil( -9.8 ) = %.1f\n", ceil( -9.8
) );
      System.out.printf( "log( E ) = %.1f\n", log( E ) );
      System.out.printf( "cos( 0.0 ) = %.1f\n", cos( 0.0 )
);
   } // end main
} // end class StaticImportTest
```

```
sqrt( 900.0 ) = 30.0
ceil( -9.8 ) = -9.0
log( E ) = 1.0
cos( 0.0 ) = 1.0
```

4. Сложение

```
// Fig. 11.3: Addition.java
// Addition program that uses JOptionPane for input and
output.
import javax.swing.JOptionPane; // program uses JOptionPane

public class Addition
{
   public static void main( String args[] )
   {
      // obtain user input from JOptionPane input dialogs
      String firstNumber =
         JOptionPane.showInputDialog( "Enter first integer"
);
      String secondNumber =
         JOptionPane.showInputDialog( "Enter second
integer" );

      // convert String inputs to int values for use in a
calculation
      int number1 = Integer.parseInt( firstNumber );
      int number2 = Integer.parseInt( secondNumber );

      int sum = number1 + number2; // add numbers

      // display result in a JOptionPane message dialog
```

```
22          JOptionPane.showMessageDialog( null, "The sum is " +
sum,
23             "Sum of Two Integers", JOptionPane.PLAIN_MESSAGE );
24       }
25    }
```

5. Факториал

```
 1  // Fig. 15.4: FactorialCalculator.java
 2  // Recursive factorial method.
 3
 4  public class FactorialCalculator
 5  {
 6     // recursive method factorial
 7     public long factorial( long number )
 8     {
 9        if ( number <= 1 ) // test for base case
10           return 1; // base cases: 0! = 1 and 1! = 1
11        else // recursion step
12           return number * factorial( number - 1 );
13     } // end method factorial
14
15     // output factorials for values 0-10
16     public void displayFactorials()
17     {
18        // calculate the factorials of 0 through 10
19        for ( int counter = 0; counter <= 10; counter++ )
20           System.out.printf( "%2d! = %d\n", counter,
factorial( counter ) );
21     } // end method displayFactorials
22  } // end class FactorialCalculator
```

```
 1  // Fig. 15.5: FactorialTest.java
 2  // Testing recursive factorial method.
 3
 4  public class FactorialTest
 5  {
 6     // calculate factorials of 0-10
 7     public static void main( String args[] )
 8     {
 9        FactorialCalculator factorialCalculator = new
FactorialCalculator();
10        factorialCalculator.displayFactorials();
11     } // end main
```

```
12  }
```

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
```

6. Фибоначчи

```
1  // Fig. [illegible]: [illegible]
2  // Recursive [illegible].
3
4  public class FibonacciCalculator
5  {
6     // recursive declaration of method fibonacci
7     public long fibonacci( long number )
8     {
9        if ( ( number == 0 ) || ( number == 1 ) ) // base
cases
10          return number;
11       else // recursion step
12          return fibonacci( number - 1 ) + fibonacci( number
- 2 );
13    } // end method fibonacci
14
15    public void displayFibonacci()
16    {
17       for ( int counter = [illegible]; counter <= [illegible]; counter++ )
18          System.out.printf( "[illegible]: %d\n",
counter,
19             fibonacci( counter ) );
20    } [illegible]
21 } [illegible]
```

```
1  [illegible]
2  [illegible]
3
```

```
 4  public class FibonacciTest
 5  {
 6     public static void main( String args[] )
 7     {
 8        FibonacciCalculator fibonacciCalculator = new
FibonacciCalculator();
 9        fibonacciCalculator.displayFibonacci();
10     } // end main
11  } // end class FibonacciTest
```

```
Fibonacci of 0 is: 0
Fibonacci of 1 is: 1
Fibonacci of 2 is: 1
Fibonacci of 3 is: 2
Fibonacci of 4 is: 3
Fibonacci of 5 is: 5
Fibonacci of 6 is: 8
Fibonacci of 7 is: 13
Fibonacci of 8 is: 21
Fibonacci of 9 is: 34
Fibonacci of 10 is: 55
```

7. Перестановки

```
 1  // Fig. 15.12: Permutation.java
 2  // Recursive method to find all permutations of a String.
 3
 4  public class Permutation
 5  {
 6     // method to initiate recursive permutation of a String
 7     private void permuteString(
 8        String beginningString, String endingString )
 9     {
10        // base case: if string to permute is length less than
 or equal to 1
11        // , then display this string concatenated with
 beginningString
12        if ( endingString.length() <= 1 )
13           System.out.println( beginningString + endingString
);
14        else // recursive step: permute remainingString
15        {
16           // for each character in endingString
17           for ( int i = 0; i < endingString.length(); i++ )
18           {
19              try
```

```
20             {
21
22
23                String newString = endingString.substring( 0,
i ) +
24                   endingString.substring( i + 1 );
25
26
27
28
29                permuteString( beginningString +
30                   endingString.charAt( i ), newString );
31             }
32             catch ( StringIndexOutOfBoundsException
exception )
33             {
34                exception.printStackTrace();
35             }
36          }
37       }
38    }
39 }
```

```
1
2
3  import java.util.Scanner;
4
5  public class PermutationTest
6  {
7     public static void main( String args[] )
8     {
9        Scanner scanner = new Scanner( System.in );
10       Permutation permutationObject = new Permutation();
11
12       System.out.print( "
13       String input = scanner.nextLine();
14
15       permutationObject.permuteString( "", input );
16    }
17 }
18
```

math
maht
mtah
mtha
mhat
mhta
amth
amht
atmh
athm
ahmt
ahtm
tmah
tmha
tamh
tahm
thma
tham
hmat
hmta
hamt
hatm
htma
htam

3 РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ

В целом работа была проведена успешно. Основные задачи, поставленные перед автором выполненной работы можно считать достигнутыми. Произведен анализ четырех языков программирования, и выявлен среди них наиболее подходящий для обучения программированию студентов “Математики”. Впоследствии будет проработан четкий план, по которому и будет производиться обучение, а точнее язык программирования Java в математической форме. То есть основная часть объясняемого материала будет задачи из математики.

ЗАКЛЮЧЕНИЕ

В заключении я бы хотел отметить, что поставленная задача выполнена. А именно были проанализированы множества языков программирования, и выбрано четыре из них, которые более менее соответствуют курсу программирования для математиков. С точки зрения логики и восприятия таких сложных задач, я считаю, что легче будет изучать какой-то из этих четырех языков программирования, нежели Java.

ЛИТЕРАТУРА

1. Д. Флэнаган, Ю. Мацумото Язык программирования Ruby = The Ruby Programming Language / пер. с англ. Н. Вильчинский. — 1-е изд. — СПб.: Питер, 2011. — 496 с. — (Бестселлеры O'Reilly). — ISBN 978-5-459-00562-2
2. Х. Фултон. Программирование на языке Ruby = The Ruby Way / пер. с англ. А. Слинкин. — 2-е изд. — М.: ДМК Пресс, 2007. — 688 с. — (Языки программирования). — ISBN 5-94074-357-9
3. М. Фитцджеральд. Изучаем Ruby = Learning Ruby / пер. с англ. Н. Гаврилова. — 1-е изд. — СПб.: БХВ-Петербург, 2008. — 336 с. — ISBN 978-5-9775-0225-2
4. Д. Х. Спольски. Лучшие примеры разработки ПО = The Best Software Writing / пер. с англ. Е. Матвеев. — СПб.: Питер, 2007. — 208 с. — (Библиотека программиста). — ISBN 5-469-01291-3
5. Ю. Мацумото Ruby — Руководство пользователя = Ruby User's Guide / пер. на русский Александр Мячков на OpenNET — 2005—2008.
6. Марк Лутц. Изучаем Python, 4-е издание. — Перевод с английского. — СПб.: Символ-Плюс, 2010. — 1280 с — ISBN 978-5-93286-159-2
7. Дэвид М. Бизли. Python. Подробный справочник, 4-е издание. — Перевод с английского. — СПб.: Символ-Плюс, 2010. — 864 с — ISBN 978-5-93286-157-8
8. Марк Саммерфилд. Программирование на Python 3. Подробное руководство. — Перевод с английского. — СПб.: Символ-Плюс, 2009. — 608 с — ISBN 978-5-93286-161-5
9. Ноа Гифт, Джереми М. Джонс. Python в системном администрировании UNIX и Linux. — Перевод с английского. — СПб.: Символ-Плюс, 2009. — 512 с — ISBN 978-5-93286-149-3
10. Дьяконов В. П. Вейвлеты. От теории к практике. Полное руководство пользователя. Изд-е 2-е переработанное и дополненное. — Москва.: «СОЛОН-Пресс», 2004. — С. 400. — ISBN 5-98003-171-5
11. Чарльз Генри Эдвардс , Дэвид Э. Пенни. Дифференциальные уравнения и проблема собственных значений: моделирование и вычисление с помощью Mathematica, Maple и MATLAB = Differential Equations and Boundary Value Problems: Computing and Modeling. — 3-е изд. — М.: «Вильямс», 2007. — ISBN 978-5-8459-1166-7
12. Алексеев Е.Р., Чеснокова О.В. MATLAB 7. Самоучитель. — Пресс, 2005. — С. 464.