

Jet

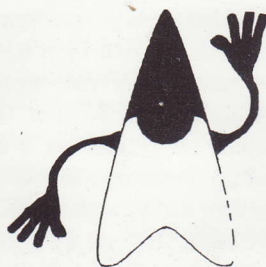
INFO

**МАТЕРИАЛ
НОМЕРА**

Java
как центр архиделага

СОМТЕК'96

Москва, Красная Пресня
22-26 апреля



**СПЕЦИАЛЬНЫЙ
ВЫПУСК**

9
1996

А ТАКЖЕ:

- **ИНТЕРВЬЮ АРТУРА ВАН ХОФФА —
ОДНОГО ИЗ РАЗРАБОТЧИКОВ JAVA**



Александр Таранов,
Владимир Цишевский

Java как центр архипелага

Содержание

1. Введение
2. Интернет, WWW и Интранет
3. Java, Joe, NEO
4. Java — язык и технология
 - 4.1. Язык Java
 - 4.1.1. Объектная модель языка Java
 - 4.1.2. Прimitives типы данных
 - 4.1.3. Пакеты
 - 4.1.4. Управляющие конструкции
 - 4.1.5. Исключительные ситуации
 - 4.1.6. Механизм потоков
 - 4.2. Технология Java
 - 4.2.1. Технологический цикл обработки Java-программ
 - 4.2.2. Java-машина
 - 4.2.3. Java и безопасность
 - 4.2.4. Java WorkShop
 - 4.3. Связь Java с окружением: оконный инструментарий
 - 4.3.1. Общие положения
 - 4.3.2. Из чего строится графический интерфейс (компоненты и контейнеры)
 - 4.3.3. Как организовать интерфейс
 - 4.3.4. События
 - 4.3.5. Методы класса Component, связанные с обработкой событий
5. Joe — технология связывания Java-программ с объектными бизнес-приложениями
6. Заключение

1. Введение

Когда говорят и пишут о Java, самой популярной фразой является "мир сошел с ума". Действительно, и скорость, и

характер распространения (так и хочется вспомнить лексикон недавнего прошлого и сказать о "победном шествии") Java не имеют аналогов. При появлении альфа-версии продукта выстраивается очередь на его лицензирование (рис. 1). Бета-версия стано-



Рис. 1. Они хотят лицензировать технологию Java (снимок сделан на острове Ява).

вится инструментом реализации информационных систем крупных корпораций. Акции компаний, имеющих прочные позиции в Интернет, растут, как на дрожжах. Все бросились реализовывать WWW-навигаторы с поддержкой Java и борются за право передать их в бесплатное использование как можно большему числу клиентов. Идет сражение за долю будущего рынка, контуры которого пока только намечаются, сражение с применением средств, которые с точки зрения "здорового бизнеса" иначе как бредовыми и назвать нельзя.

В чем причина всеобщего помешательства и что это за продукт, околдовавший мир? Попытаемся совсем коротко высказать некоторые соображения по первому вопросу,

после чего перейдем на чисто технические рельсы и сосредоточимся на описании технологии и языка Java и ассоциированных продуктов.

Персональные компьютеры сделали информационные технологии частью массовой культуры. При миллионных тиражах даже единственный "компьютерный хит" способен принести очень большие деньги. Авторы многих подобных хитов, помимо богатства, получают колоссальное влияние на людей, что по существу является источником огромного дополнительного обогащения. И тем не менее, уже довольно длительная история развития персональных компьютеров не знала ничего, подобного феномену Java. Что изменилось в мире в последние годы, почему этот феномен стал возможен?

Изменился Интернет. Он стал доступен миллионам людей, далеких от технических проблем. Число пользователей Интернет по порядку величины уже не отличается от числа пользователей персональных компьютеров и продолжает взрывообразно расти. Одновременно Интернет обеспечил такую скорость распространения новинок информационных технологий, которую не могли и никогда не смогут дать традиционные каналы сбыта. Время спрессовалось. В Интернет, опоздав буквально на день, компьютерная компания, даже крупная, рискует серьезно ослабить свои позиции сразу во всем мире.

2. Интернет, WWW и Интранет

Информационные перегрузки — характерная черта

нашего времени. Созданы мощные механизмы, обеспечивающие производство огромного количества информации. Существенно меньше сделано для облегчения ее получения и усвоения.

Типичной является ситуация, когда инициатива принадлежит поставщику, а не потребителю информации. Поставщик по определенному поводу создает информацию и направляет ее всем, кто, по его мнению, в ней нуждается (рис. 2).

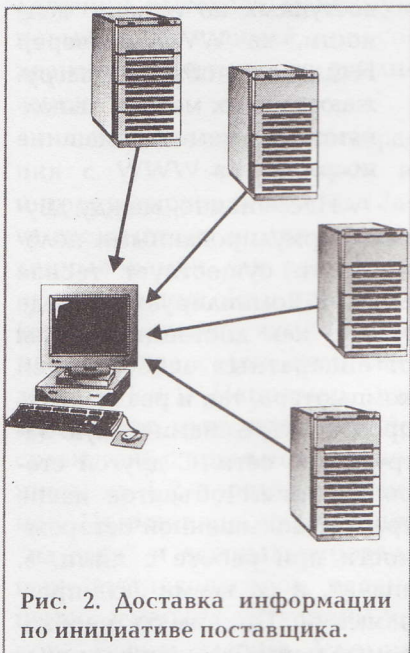


Рис. 2. Доставка информации по инициативе поставщика.

Так работают средства массовой информации, издательства, рекламные агентства. Так работает электронная почта. В большинстве случаев потребителю эта информация, может быть, и нужна, но не в данный момент, не сейчас. Потребитель вынужден архивировать полученную информацию. При этом в лучшем случае велика вероятность, что к моменту, когда информация действительно понадобится, она потеряет актуальность. Обычно же у потребителя просто накапливаются горы мусора, в котором отыскать нечто нужное почти невозможно.

Чтобы информация была актуальной для потребителя, она должна доставляться к не-

му по запросу — в точности тогда, когда в ней возникла необходимость. Кроме того, поставщик должен сохранять возможность управления информацией, он должен не только создавать ее, но и вовремя обновлять и уничтожать.

Централизованные компьютерные системы, доминировавшие еще 10 лет назад, позволяли пользователям сравнительно легко находить информацию в оперативном режиме (рис. 3), однако они затрудняли управление информацией, поскольку ее источники, как правило, разнородны и территориально разнесены. Еще один важный недостаток централизованных систем — их сложность и дороговизна. Подавляющему большинству российских организаций они просто не по карману.



Рис. 3. Взаимодействие с централизованной компьютерной системой.

Сети персональных компьютеров существенно дешевле централизованных систем, они оставляют за поставщиком необходимую свободу управления информацией, однако потребителям приходится искать необходимые данные на множестве машин, среди большого числа приложений с различными интерфейсами (рис. 4). Рядовому пользователю работать в такой разнородной прикладной среде крайне неудобно.

Способ разрешения указанных проблем, к которому

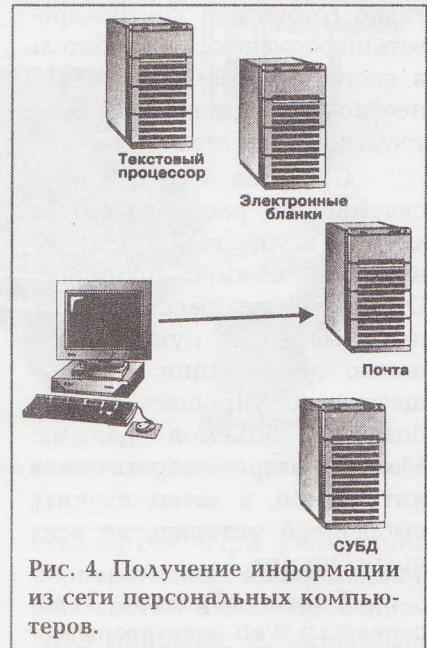


Рис. 4. Получение информации из сети персональных компьютеров.

прибегают ведущие компании, состоит в построении информационной структуры организации по образу и подобию Интернет, с Web-сервисом в качестве концептуальной основы (рис. 5).

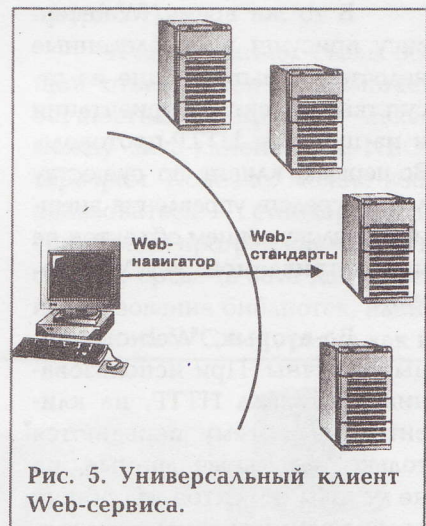


Рис. 5. Универсальный клиент Web-сервиса.

Возможность хранения данных различных типов (текст, графика, аудио, видео) в сочетании с механизмами связывания информации, расположенной в разных узлах компьютерной сети, позволяют рассредоточивать информацию в соответствии с естественным порядком ее создания и потребления, осуществлять единообразный доступ, отправляясь от небольшого числа известных "корней". Тем самым поставщик может эффек-

тивно готовить и контролировать информацию, а потребитель в состоянии без труда найти необходимые данные именно тогда, когда они стали нужны.

Средства Web, помимо связывания распределенных данных, осуществляют еще одну очень важную функцию. Они позволяют рассматривать информацию с нужной степенью детализации, что существенно упрощает анализ больших объемов данных. Можно быстро отобрать самое интересное, а затем изучить выбранный материал во всех подробностях.

Таким образом, Web-серверы и Web-навигаторы могут и должны использоваться не только в "мировом масштабе". Web — это инфраструктурный сервис, необходимый каждой организации со сколько-нибудь заметными информационными потоками.

В то же время, Web-сервису присущи и определенные недостатки, вытекающие из отсутствия объектной ориентации и из природы HTTP-протокола. Во-первых, клиент по существу лишен средств управления внешним представлением объектов на просматриваемой WWW-странице.

Во-вторых, Web-страницы статичны. При использовании протокола HTTP, на клиентскую систему передаются только пассивные данные, но не методы объектов. Из общих соображений очевидна ограниченность подобного подхода. Данный недостаток, разумеется, связан с первым. Объект сам должен знать, как себя показывать — точнее говоря, он должен это выяснить, проанализировав клиентское окружение.

В-третьих, Web-сервис обладает весьма ограниченными интерактивными возможностями, которые сводятся к заполнению пользователем

чисто текстовых форм с последующей отправкой на сервер. Сервер анализирует полученные данные, после чего формирует и возвращает клиенту новую WWW-страницу, которая нередко вновь оказывается формой. Такой стиль общения не всегда устраивает пользователей.

Java-технология позволяет устранить все отмеченные недостатки. Как именно — будет ясно из последующего изложения. В результате Web-сервис, и без того имевший огромную популярность, получил как бы новый импульс. Этот экспресс понесся вперед с удвоенной скоростью, увлекая за собой и Java.

3. Java, Joe, NEO

В узком смысле слова Java — это объектно-ориентированный язык, напоминающий C++, но более простой для освоения и использования. В более широком смысле Java — это целая технология программирования, изначально рассчитанная на интеграцию с Web-сервисом, то есть на использование в сетевой среде. Поскольку Web-навигаторы существуют практически для всех аппаратно-программных платформ, Java-среда должна быть как можно более мобильной, в идеале полностью независимой от платформы.

С целью решения перечисленных проблем были приняты, помимо интеграции с Web-навигатором, два других важнейших постулата:

- Была специфицирована виртуальная Java-машина, на которой должны выполняться (интерпретироваться) Java-программы. Определены ее архитектура, представление элементов данных и система команд. Исходные Java-тексты транслируются в коды этой машины. Тем самым, при появлении новой

аппаратно-программной платформы в портировании будет нуждаться только Java-машина; все программы, написанные на Java, пойдут без изменений.

- Определено, что при редактировании внешних связей Java-программы и при работе Web-навигатора прозрачным для пользователя образом может осуществляться поиск необходимых объектов не только на локальной машине, но и на других компьютерах, доступных по сети (в частности, на WWW-сервере). Найденные объекты загружаются, а их методы выполняются затем на машине пользователя.

Несомненно, между двумя сформулированными положениями существует тесная связь. В компилируемой среде трудно как дистанцироваться от аппаратных особенностей компьютера, так и реализовать прозрачную динамическую загрузку по сети. С другой стороны, прием объектов извне требует повышенной осторожности при работе с ними, а значит, и со всеми Java-программами. Принимать необходимые меры безопасности проще всего в интерпретируемой среде. Вообще, мобильность, динамизм и безопасность — спутники интерпретатора, а не компилятора.

Принятые решения сделали Java-среду идеальным средством разработки клиентских компонентов Web-систем. Особо отметим прозрачную для пользователя динамическую загрузку объектов по сети. Из этого вытекает такое важнейшее достоинство, как нулевая стоимость администрирования клиентских систем, написанных на Java. Достаточно обновить версию объекта на сервере, после чего клиент автоматически получит именно ее, а не старый вариант. Без

этого реальная работа с развитой сетевой инфраструктурой практически невозможна. С другой стороны, при наличии динамической загрузки действительно возможно появление устройств класса Java-терминалов, изначально содержащих только WWW-навигатор, а все остальное (и программы, и данные) получающих по сети.

Здесь уместно отметить замечательную точность в выборе основных посылок проекта Java. Из минимума предположений вытекает максимум новых возможностей при сохранении практичности реализации.

В то же время, интеграция с WWW-навигатором и интерпретируемая природа Java-среды ставят вполне определенные рамки для реального использования Java-программ (хотя, конечно же, язык Java не менее универсален, чем, скажем, C++). Например, известно, что интерпретация, по сравнению с прямым выполнением, на 1-2 порядка медленнее. Применение компиляций "на лету" и специализированных Java-процессоров, несомненно, улучшит ситуацию, но пока использование Java на серверной стороне представляется проблематичным.

Далее, хотя технология Интранет, основанная на использовании Web-сервиса в качестве информационной основы организации, является огромным шагом вперед, существуют и другие сервисы, как унаследованные, так и современные (например, реляционные СУБД), которые обязательно должны входить в состав корпоративной системы. Если вся связь между клиентами и упомянутыми серверами будет осуществляться через сервер WWW, последний станет узким местом, а решения Интранет рискуют лишиться такого важнейшего достоинства, как

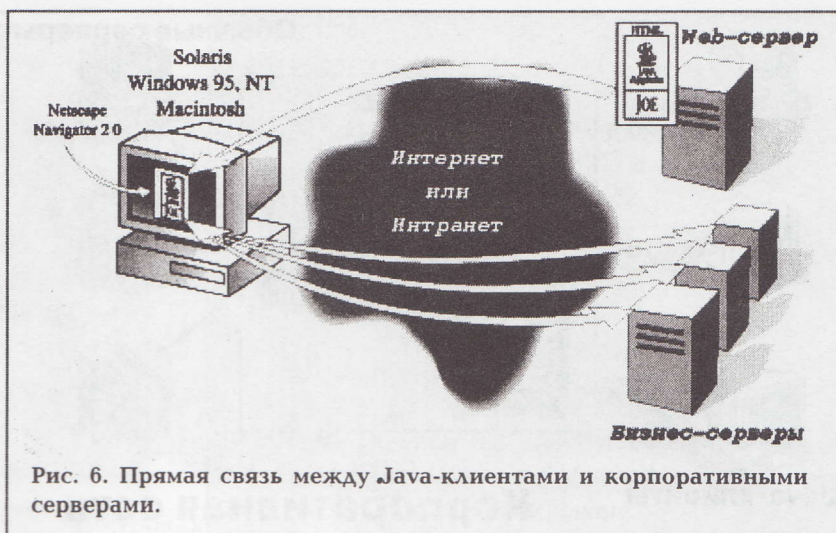


Рис. 6. Прямая связь между Java-клиентами и корпоративными серверами.

масштабируемость. Значит, необходима прямая связь между клиентскими системами, написанными на языке Java, и произвольными сервисами (рис. 6).

Как реализовать такую связь?

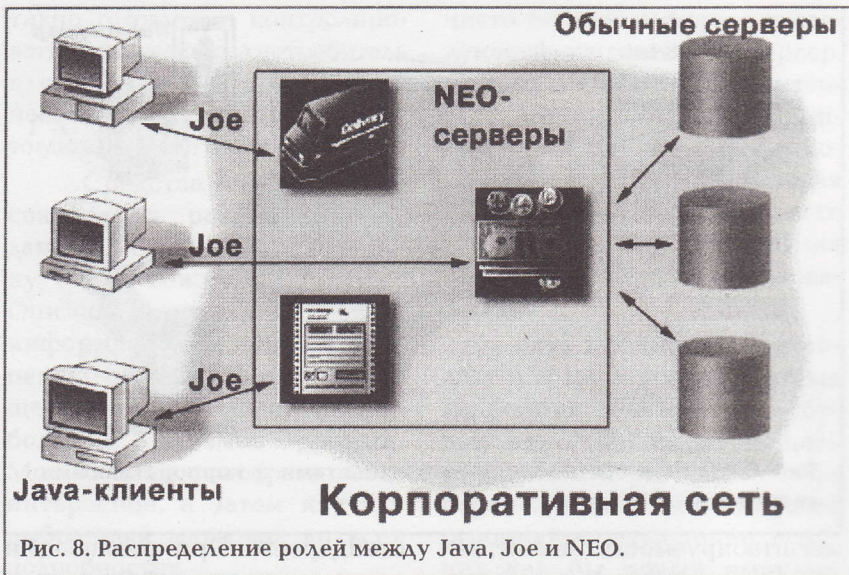
В общем виде ответ очевиден — нужны средства для полноценной интеграции Java в распределенную объектную среду. На серверной стороне компания Sun Microsystems имеет соответствующую технологию — NEO (NEtworked Objects, сетевые объекты). Технология NEO удовлетворяет спецификациям OMG (Object Management Group), являющимся промышленным

стандартом. При реализации корпоративных информационных систем с использованием NEO наиболее естественным представляется использование трехуровневой архитектуры с серверами приложений, построенными на объектных принципах, на втором уровне и с базовыми и унаследованными серверами на третьем уровне (рис. 7).

К сожалению, столь общий ответ никак не помогает осуществлять прямую связь между Java-клиентом и NEO-сервером. Конечно, можно воспользоваться стандартными средствами программирования в сетевой среде (а Java допускает использование библиотек, написанных на C/C++, равно как и



Рис. 7. Трехуровневая архитектура корпоративной информационной системы.



вставку машинных кодов), но если бы это было единственной возможностью, Java рисковала остаться на уровне "оживлялок".

В конце марта компания SunSoft объявила о появлении нового продукта с именем Joe, как раз и предназначенного для существенного облегчения взаимодействия Java-клиентов в информационные системы Интранет, построенные в трехуровневой архитектуре с использованием среды NEO (рис. 8).

Таким образом, сложилась полная и изумительно красивая картина организации современных Интранет-систем.

В данной статье мы уделим основное внимание технологии Java. Далее будет кратко рассмотрены возможности, предоставляемые средой Joe.

4. Java — язык и технология

4.1. Язык Java

При описании языка Java будет предполагаться, что читатель, хотя бы в общих чертах, знаком с языком C++.

4.1.1. Объектная модель языка Java

Когда говорят об объектно-ориентированном языке программирования, предполагают поддержку трех механизмов:

- инкапсуляция
- наследование
- полиморфизм.

Инкапсуляция и наследование в языке Java реализуются с помощью понятия класса.

4.1.1.1. Классы

Понятие класса в языках Java и C++ очень близки. Класс является шаблоном для создания объектов; он может содержать данные и методы. В листинге 1 приведен пример класса, описывающего точки в двумерном пространстве. Здесь и далее номера строк используются для последующих пояснений и не являются частью Java-программ.

В языке Java нельзя отрывать определение метода (функции) от описания заголовка. Синтаксическая конструкция class полностью включает в себя всю информацию о классе. В частности, реализации методов обязаны содержаться внутри этой конструкции.

Для обозначения наследования используется ключевое слово extends (строка 1). Класс Object — это корень дерева наследования. В Java не бывает классов-"сирот": у всех классов, кроме Object, есть предшественник. Подробнее наследование и предопределенные классы будут рассмотрены далее.

Режимы доступа к элементам класса (private, protected, public) те же, что и в C++, за одним важным исключением. Если режим доступа опущен, предполагается, что соответствующий элемент доступен в пределах пакета (см. далее).

В строке 9 приведен пример явного вызова одного конструктора из другого.

Прочие приведенные выше строки не нуждаются в пояснениях кроме одной — отсутствующей. В языке Java не бывает деструкторов. Причина в том, что управление памятью автоматизировано (в фоновом режиме работает сборщик мусора). Для высвобождения прочих ресурсов, ассоциированных с объектом, служит специальный метод finalize. Этот метод вызывается сборщиком мусора в момент утилизации памяти, занимаемой объектом.

Ключевое слово this (см., например, строки 5 и 9) исполь-

```

1 class Point extends Object {
2   private double x;
3   private double y;
4
5   Point (double x, double y) {
6     this.x = x;
7     this.y = y;
8   }
9
10  Point () {
11    this (0.0, 0.0);
12  }
13
14  public void setX (double x) {
15    this.x = x;
16  }
17  }

```

Листинг 1.

```

1 class Point3D extends Point {
2     protected double z;

3     Point3D () {
4         super ();
5         z = 0.0;
6     }

7     Point3D (double x, double y, double z) {
8         super (x, y);
9         this.z = z;
10    }
11 }

```

Листинг 2.

зуется в объекте для ссылки на самого себя. Аналогичную роль по отношению к родительскому классу играет слово `super` (листинг 2). В строках 4 и 8 вызываются конструкторы родительского класса.

Приведенные примеры показывают, что в языке Java, как и в C++, методы могут быть перегруженными, то есть под одним именем могут фигурировать разные методы с разным набором параметров.

Как и в C++, с помощью ключевого слова `static` можно определить данные и методы, которые являются общими для всех объектов класса. (Отметим попутно, что функций, не принадлежащих какому-либо классу, в языке Java не бывает.) Следующий пример (листинг 3) содержит фрагменты стандартного пакета `java.lang`.

Прокомментируем этот пример с точки зрения отличий Java от C++.

Строки с 5 по 21 представляют собой инициализацию статических данных класса, которая осуществляется в момент загрузки класса в Java-машину. По сути этот код играет роль конструктора класса `Character`.

В строках 1 и 3 встречается ключевое слово `final`. В строке 1 это слово обозначает запрет на наследование от класса `Character`. В строке 3 его смысл аналогичен описателю `const` в C++.

Если слово `final` использовано в заголовке метода, то данный метод не может быть переопределен в классах-наследниках.

Как и в C++, в языке Java классы могут быть абстрактными, то есть не до конца конкретизированными. Это означает, что в классе описаны методы, определения которых отсутствуют. Такие методы (как и сам класс) должны снабжаться описателем `abstract`. Абстрактные методы должны конкретизироваться в производных классах.

В языке Java имеется предопределенная иерархия классов, содержащихся в пакете `java.lang`. На рис. 9 эта ие-

```

1 public final
2 class Character extends Object {

3     public static final int MAX_RADIX = 36;

4     static char downCase[];

5     static {
6         char down[] = new char[256];
7         for (int i = 0; i < 256; i++) {
8             down[i] = (char) i;
9         }
10        for (int lower = 'a'; lower <= 'z'; lower++) {
11            int upper = (lower + ('A' - 'a'));
12            down[upper] = (char) lower;
13        }

14        for (int lower = 0xE0; lower <= 0xFE; lower++) {
15            if (lower != 0xF7) {
16                int upper = (lower + ('A' - 'a'));
17                down[upper] = (char) lower;
18            }
19        }
20        downCase = down;
21    }

22    public static boolean isLowerCase(char ch) {
23        return (upCase[ch] != ch);
24    }
25 }

```

Листинг 3.

рархия представлена графически. (Автором рис. 9, 14, 15 является Charles L. Perkins, clp@home.harvardsq.com.)

В этой иерархии несколько особняком стоит класс `Class`. Программист не может создать объект класса `Class` (правда, существуют и другие классы с этим свойством). Ссылки на объект класса `Class` можно получить с помощью метода `getClass`, определенного для объектов класса `Object`.

Объекты класса `Class` используются для получения во время выполнения информации о "классовых" свойствах объекта. К объектам класса `Class`, по-

```

1 public native String getName();
2 public native Class getSuperclass();
3 public static native Class forName(String className) throws ClassNotFoundException;

```

Листинг 4.

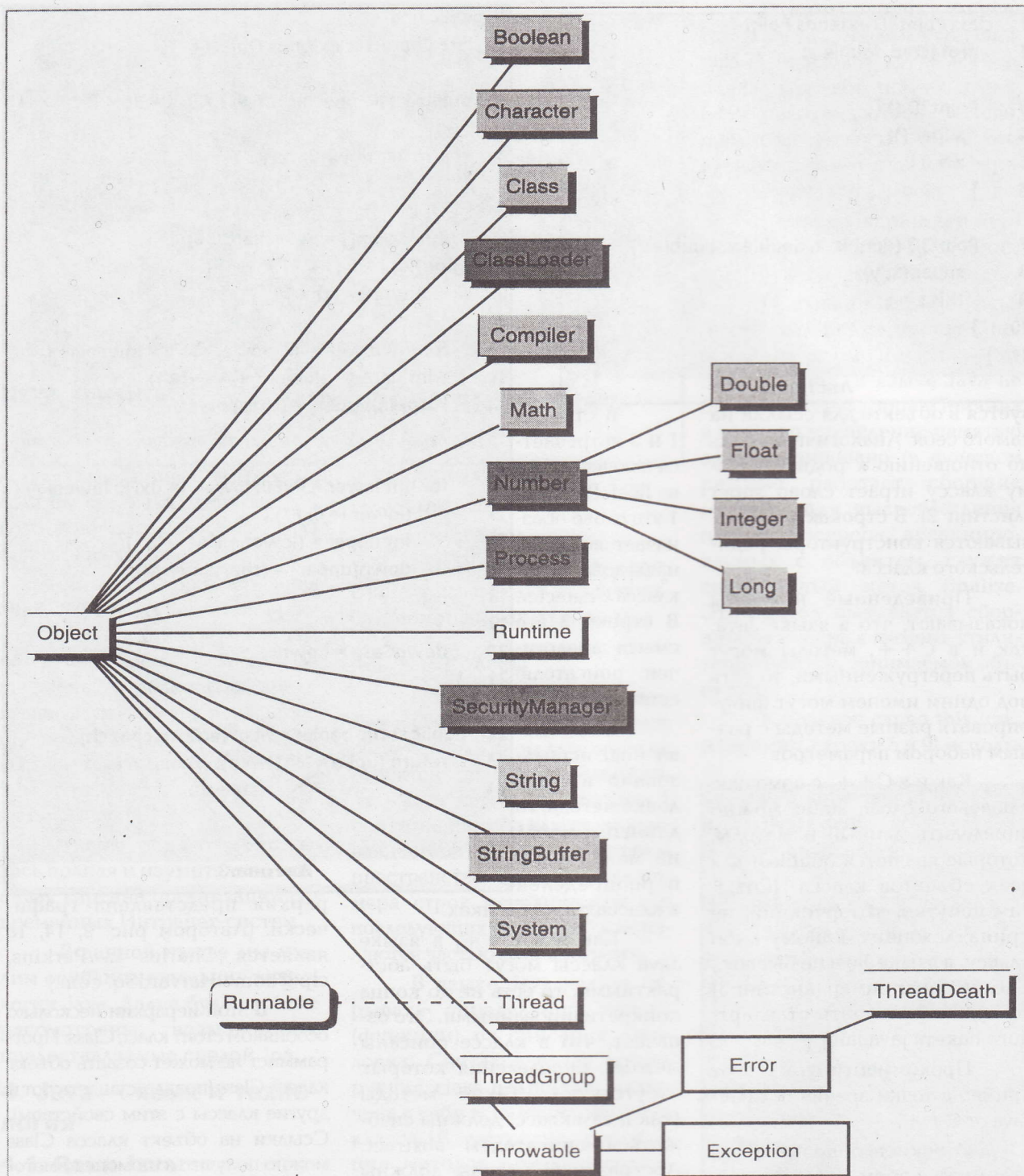


Рис. 9. Иерархия predefined классов языка Java.

мимо прочих, применимы методы, показанные на листинге 4.

Метод `forName` позволяет получить ссылку на класс по его имени. Описатель `native` свидетельствует о том, что метод реализуется средствами, внешними по отношению к Java-системе (например, пишется на языке C).

4.1.1.2. Наследование

Модель наследования в языке Java существенно отличается от модели C++. Во-первых, в Java нет множественного наследования. Во-вторых, в языке предусмотрены средства для запрета дальнейшего наследования (ключевое слово `final` перед определением класса). В-третьих, в языке Java присутст-

вует новое по отношению к C++ понятие интерфейса.

Интерфейс представляет собой набор описаний методов. Пример приведен на листинге 5.

Интерфейс `Verbose` содержит два метода. Первый предназначен для рисования в графическом контексте, второй — для распечатки в выходной поток.


```

public interface Verbose {
    public void drawOn (Graphics g);
    public void printOn (OutputStream os);
}

public class Star extends Polygon implements Verbose {
    public void drawOn (Graphics g) {
        // Конкретная реализация отрисовки
    }
    public void printOn (OutputStream os) {
        // Конкретная реализация печати
    }
}

public class Text extends StringBuffer implements Verbose {
    public void drawOn (Graphics g) {
        // Конкретная реализация отрисовки
    }
    public void printOn (OutputStream os) {
        // Конкретная реализация печати
    }
}

public class Blackboard extends Canvas {
    public void drawVerbose (Verbose d) {
        Graphics g = getGraphics ();
        d.drawOn (g);
    }
}

```

Листинг 5.

Класс `Star` представляет собой подкласс многоугольников (`Polygon`). Помимо прочего, мы хотим рисовать объекты этого класса на доске (`Blackboard`) и выводить их описание в файл.

Для объектов класса `Text` мы хотим иметь возможность начертать текст на доске и выводить его в файл.

Класс `Blackboard` — наследник класса `Canvas`, специально предназначенного для рисования. Как мы видим, этот класс с помощью единственного метода справляется с рисованием объектов; не имеющих общего предка (кроме класса `Object`).

Таким образом, с помощью интерфейсов можно компенсировать отсутствие множественного наследования. В контексте наследования интер-

фейс можно рассматривать как абстрактный класс, не содержащий данных.

4.1.1.3. Жизненный цикл объекта

Объекты создаются с помощью оператора `new`. Инициализация объекта производится с помощью соответствующего конструктора. Эти операции разделить нельзя — за `new` всегда следует конст-

руктор. Пример:

```
Point myPoint = new Point ();
```

Других способов создания объектов (кроме оператора `new`) язык Java не предоставляет.

Объект существует до тех пор, пока на него есть ссылки (то есть пока он прямо или косвенно доступен хотя бы из одной переменной). В языке отсутствуют явные средства удаления объектов. После того, как объект стал недоступен, он оказывается кандидатом для утилизации сборщиком мусора.

Объект может стать недоступным, если хранившей его переменной присвоено новое значение или если эта переменная перестала быть доступной (произошел выход из блока). Пример приведен на листинге 6.

В момент утилизации объекта сборщиком мусора будет вызван метод `finalize`. Из-за того, что сборщик мусора работает в фоновом режиме, вызов `finalize` является асинхронным.

4.1.2. Прimitives типы данных

В языке Java существует набор встроенных типов данных, которые не являются объектами. Их не так много.

Представление чисел в языке Java фиксировано и, тем самым, не зависит от аппаратной платформы.

- целые числа:
 - `byte` — 8 бит,
 - `short` — 16 бит,
 - `int` — 32 бита,

```

Point p = new Point (100.0, 100.0);
...
p = new Point (1.0, 1.0);
// На точку с координатами (100, 100) ссылок больше нет

{
    String s = new String ("Local string");
    System.out.println (s);
}
// На строку "Local string" ссылок больше нет

```

Листинг 6.

- long — 64 бита.

Все числа со знаком, спецификатор unsigned в языке отсутствует.

- числа с плавающей точкой:
 - float — 32 бита,
 - double — 64 бита.

Представление должно соответствовать стандарту IEEE 754.

- char

Значение типа char есть 16-разрядное число без знака (диапазон 0-65535). Кодировка соответствует стандарту Unicode. В результате с самого начала закладывается здоровая основа для решения проблемы локализации Java-программ.

- boolean

Содержит значения true и false, которые не могут быть преобразованы в другой тип.

В языке Java, разумеется, присутствуют массивы. Эти массивы типизированы. Декларация

```
Point myPoints[];
```

описывает переменную myPoints как массив объектов типа Point. Завести массив определенного размера можно с помощью инструкции вида

```
myPoints = new Point[10];
```

Значения элементов массива при этом устанавливаются равными специальной величине null.

Размер массива может быть получен во время выполнения программы:

```
howMany = myPoints.length;
```

Других структурных типов (не являющихся объектами) в языке Java нет, то есть нет структур, объединений и т.п. Нет в Java и указателей.

Отметим, что строки символов являются объектами типа String (текстовые константы) или StringBuffer (изменяемые строки). Пример:

```
String hello = "Hello world!";
```

4.1.3. Пакеты

Классы в языке Java объединяются в пакеты. Все классы, входящие в один пакет, являются дружественными по отношению друг к другу, то есть имеют взаимный доступ к переменным и методам, если противное не оговорено явно посредством спецификаторов private или protected.

Пакеты разграничивают пространство имен. "Просто глобальных" имен в языке Java не бывает.

Пакет оформляется с помощью синтаксической конструкции вида

```
package my_packages.pack1;
```

Инструкция package должна стоять первой в файле с исходным Java-текстом. Она действует до конца файла.

Пакеты могут импортироваться другими пакетами посредством инструкции import. Примеры,

```
import java.util;
import java.util.HashMap;
import java.util.*;
```

Первая инструкция import позволяет обращаться к классам пакета util следующим образом:

```
util.Vector
util.HashMap
```

Вторая инструкция импортирует лишь класс HashMap, позволяя в дальнейшем обращаться к этому классу по короткому имени, без префикса util.

Третья инструкция import позволяет обращаться по коротким именам ко всем классам пакета util.

4.1.4. Управляющие конструкции

Управляющие конструкции языка Java вполне традиционны, за исключением средств выхода из вложенных блоков (в частности, из вложенных циклов). Пример:

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

Для передачи управления можно применять как конструкцию break, так и continue (переход к следующей итерации цикла).

Инструкция goto в языке Java отсутствует.

4.1.5. Исключительные ситуации

Для обработки исключительных ситуаций, возникающих во время выполнения программы, в языке Java используется конструкция try/catch/finally. Блок try содержит инструкции, выполнение которых может привести к возникновению исключительных ситуаций. Следующие за ним один или несколько блоков catch предназначены для обработки исключительных ситуаций. Наконец, блок finally содержит инструкции, которые будут выполнены независимо от возникновения исключительной ситуации в блоке try. При выходе из try-части посредством инструкций передачи управления (break, return и т.п.) блок finally также будет выполнен.

Для передачи информации об исключительной ситуации используются объекты классов — наследников класса Throwable. Например, класс ArrayIndexOutOfBoundsException отвечает за контроль выхода индексов за границы массивов, класс OutOfMemoryException — за реакцию на исчерпание свободной памяти, класс ClassCastException — за ошибки при преобразовании типов, класс InterruptedException — за обработку прерывания текущего потока и т.д. Компонентой всех этих классов является поле типа String, в

которое помещается текст сообщения об ошибке. Метод getMessage возвращает этот текст.

В подобных объектах может содержаться и дополнительная информация. Например, объекты класса InterruptedException содержат поле, в которое заносится число байт, переданных до возникновения исключительной ситуации.

Фрагмент программы, показанный на листинге 7, распечатывает сообщения из массива messages. При этом мы не пытаемся выяснить размер массива, а просто полагаемся на механизм обработки исключительных ситуаций. (Конечно, мы не советуем писать программы в таком стиле).

Исключительные ситуации могут возбуждаться программно при помощи инструкций вида

```
throw new MyException (
    "Something's wrong");
```

Спецификации языка Java подразделяют исключительные ситуации на две категории. К первой категории (класс Error) относятся ситуации, на которые программа не обязана реагировать (это заведомо сделает Java-машина). Ко второй категории (класс Exception) относятся си-

```
try {
    for (int i = 0; i < 100; i++) {
        System.out.println (messages[i]);
    }
}
catch (ArrayOutOfBoundException e) {
    System.out.println ("No more messages");
}
catch (Exception e) {
    System.out.println ("Unexpected exception");
    System.out.println (e.getMessage());
}
finally {
    System.out.println ("Work done");
}
```

Листинг 7.

```
class Foo extends Object {
    ...
    public void readFromFile (String fn) throws InvalidFormatException {
        FileInputStream fis;
        try {
            fis = new FileInputStream (fn);
            // Читаем данные из файла.
            ...
            // Если файл имеет неправильный формат,
            // возбуждаем исключительную ситуацию:
            throw new InvalidFormatException ("Wrong format");
            ...
        }
        catch (FileNotFoundException e) {
            // Предпринимаем соответствующие действия
        }
        finally {
            if (fis != null )
                fis.close(); // всегда закрываем файл, если он был открыт
        }
    }
    ...
}
```

Листинг 8.

туации, которые программа должна обрабатывать обязательно. Если при выполнении метода может возникнуть исключительная ситуация второго типа, он должен либо обрабатывать ее сам с помощью конструкции try/catch/finally, либо в его определении должна фигурировать конструкция throws Exception1, Exception2, ...

В примере, показанном на листинге 8, в методе readFromFile могут возникнуть две исключительные ситуации. Первая связана с тем, что нужный файл недоступен. Эта ситуация обрабатывается внутри метода readFromFile. Вторая исключительная ситуация может возникнуть, если файл имеет неправильный формат. Эта ситуация передается для обработки наверх.

4.1.6. Механизм потоков

Механизм потоков — обязательная черта современных операционных сред. За счет потоков обеспечивается масштабируемость программных систем, оптимальное использование аппаратных ресурсов, высокая скорость отклика на запросы пользователей. Нет ничего удивительного в том, что в языке Java механизм потоков предусмотрен с самого начала и в полном объеме.

В языке Java потоки представлены посредством класса Thread, интерфейса Runnable, спецификатора метода synchronized и методов класса Object wait и notify.

4.1.6.1. Класс Thread и интерфейс Runnable

Поток (thread) представляет собой отдельный поток управления в пределах процесса. Таким образом, у каждого потока есть начало, последовательность

действий, текущее состояние и конец.

Поток запускается с помощью вызова метода `start()` класса `Thread`. Последовательность действий, выполняемых в рамках потока, задается в методе `run()`. Подчеркнем, что метод `run()` используется только для задания последовательности действий; явно вызывать его не только не нужно, но и просто вредно.

Поток заканчивается либо при завершении выполнения метода `run()`, либо с помощью явных вызовов методов класса `Thread` `stop()` или `destroy()`. Возобновить работу завершенного потока невозможно.

Для временной приостановки работы потока с последующим возобновлением служат методы `suspend()`, `sleep()` и `yield()`.

Обычно поток, приостановленный с помощью метода `suspend`, возобновляет работу посредством метода `resume()`.

Вызов метода `sleep()` приводит к приостановке потока на заданное число миллисекунд.

Вызов метода `yield()` означает добровольную уступку процессора другому потоку; первоначальный поток остается готовым к выполнению.

Java-потоки обладают приоритетами. В спецификациях оговаривается, что Java-машина реализует вытесняющую многопоточность. Это означает, что поток с большим приоритетом может прервать выполнение менее приоритетного потока. Однако, спецификации не требуют наличия разделения времени. Это значит, что для передачи управления потоку с тем же приоритетом, вообще говоря, требуются явные действия со стороны первоначального потока — вызов методов `suspend()`, `sleep()` или `yield()`.

Пример на листинге 9 содержит фрагмент одного из многочисленных вариантов решения задачи "производитель/потребитель". Он заимствован из письма, которое написал Mark Tillotson в группу сетевых новостей `comp.lang.java`.

Данный производитель помещает в буфер квадраты целых чисел.

В приведенном простом примере класс `my_producer` является наследником класса `Thread`, что делает его потоком с последовательностью действий, заданной методом `run()`. В реальных программах, как правило, объект должен наследовать у какого-либо предшественника содержательные свойства, а возможность параллельного выполнения ему предоставляется интерфейсом `Runnable`. Этот интерфейс содержит единственный метод — `run()`. Пример показан на листинге 10.

В строке 6 создается новый поток. Аргументом конструктора является объект класса

`SomethingToRun`, а, значит, последовательность выполняемых действий потока будет определяться методом `run()` этого класса. Вызов метода `start()` в строке 7 ставит поток в очередь готовых для выполнения.

4.1.6.2. Средства синхронизации потоков

Как и во всякой многопроцессной или многопоточной среде, в Java существует проблема синхронизации доступа к разделяемым ресурсам. Примером такого ресурса является буфер в задаче "производитель/потребитель".

Для опытных программистов отметим, что модель синхронизации, принятая в языке Java, опирается на концепцию монитора, предложенную в 70-е годы Бринк-Хансеном.

В Java-программах можно выделять критические интервалы, которые обозначаются ключевым словом `synchronized`. Если критическим интервалом является метод, спецификатор `synchronized` поме-

```
class my_producer extends Thread
{
    int items_to_do ;
    my_buffer the_buffer ;

    my_producer (my_buffer buf, int count)
    { super() ;
      the_buffer = buf ;
      items_to_do = count ;
    }

    public void run ()
    {
        while (items_to_do > 0)
        { System.out.println ("producer to_do = " + items_to_do) ;
          Integer item = new Integer (items_to_do*items_to_do) ;
          the_buffer.insert (item) ;
          items_to_do-- ;
        }
        System.out.println ("producer exiting") ;
    }
}
```

Листинг 9.

```

1 class SomethingToRun extends BaseRunner implements Runnable {
2     private Thread aThread;
3     public void run () {
4         // выполняемые действия
5         ...
6     }
7
8     SomethingToRun () {
9         aThread = new Thread (this);
10        aThread.start ();
11    }
12 }

```

Листинг 10.

щается в его (метода) заголовок. Для превращения произвольной инструкции (обычно это блок) в критический интервал служит конструкция

synchronized (выражение) инструкция; где результатом выражения должен быть объект или массив.

Выполнение критического интервала начинается только после получения потоком монопольного доступа к соответствующему объекту или массиву. До наступления этого момента поток блокируется.

Вызов wait() внутри критического интервала приводит к тому, что текущий поток уступает монопольное право на критический интервал и приостанавливается до тех пор, пока из какого-либо другого потока не будет сделан вызов notify() или notifyAll(). Хорошей иллюстрацией использования средств синхронизации потоков является упоминавшаяся выше программа Марка Тиллотсона, которая показана на листинге 11.

Приведенная программа написана в очень хорошем, понятном стиле. Мы прокомментируем лишь один момент. В методах insert() и extract() класса my_buffer вызов wait() содержится внутри бесконечного цикла. Дело в том, что вызов notify() относится к объекту в целом. "Разбуженный" объект должен проанализировать свое состояние и решить, что делать

дальше. Так, если "заснул" метод insert(), то после возобновления работы необходимо проверить, что буфер уже не полон и добавление нового элемента стало возможным. Если это не так, метод insert() заснет вновь.

4.2. Технология Java

4.2.1. Технологический цикл обработки Java-программ

В принципе, технологический цикл подготовки, трансляции, редактирования внешних связей, тестирования, отладки и выполнения Java-программ тот же, что и для других интерпретируемых языков программирования, но с одним существенным отличием — при редактировании внешних связей требуемые компоненты могут доставляться по сети (рис. 10).

Важно отметить, однако, что Java-программы могут представлять как бы в двух ипо-

стасях — как самостоятельное приложение и как апплет, то есть совокупность объектов, выполняющихся в среде WWW-навигатора.

С точки зрения программиста, апплет и приложение отличаются в первую очередь точками входа и жизненным циклом.

Приложение в качестве точки входа имеет метод

```
public static void main (String args[]);
```

Этот метод должен быть определен в том public-классе, который содержится в файле, выполняемом виртуальной Java-машиной. В параметр args передается массив строк — параметров командной строки.

Пример: программа, печатающая свои аргументы

```

public class myTop {
    public static void main (String args[]){
        int argc = args.length;

        for (int i = 0; i < argc; i++)
            System.out.println (args[i]);
    }
}

```

Апплет выполняется в контексте навигатора и его жизненный цикл определяется следующими методами класса Applet:

- public void init () вызывается навигатором при загрузке апплета;
- public void start () вызывается навигатором при показе страницы;

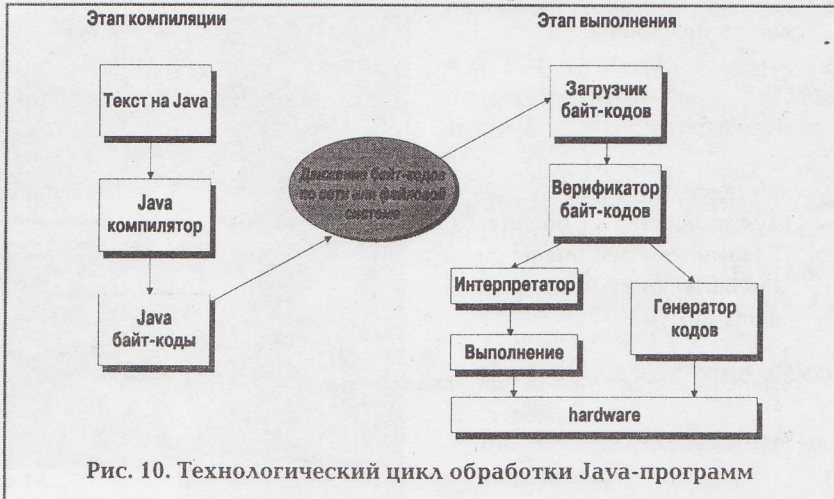


Рис. 10. Технологический цикл обработки Java-программ

```

class my_buffer
{
    Object [] vec = new Object [8];
    int ip = 0;
    int ep = 0;

    synchronized void insert (Object item)
    {
        do
        {
            if (ip-ep < 8)
            { vec [(ip++) & 7] = item;
              if (ip-ep == 1) notify (); // Уведомить, если буфер
                // был пуст
              return;
            }
            try wait (); catch (InterruptedException e);
        } while (true);
    }

    synchronized Object extract ()
    {
        do
        {
            if (ip > ep)
            { Object result = vec [(ep++) & 7];
              if (ip-ep == 7) notify (); // Уведомить, если
                //буфер был полон
              return result;
            }
            try wait (); catch (InterruptedException e);
        } while (true);
    }
}

class my_producer extends Thread
{
    int items_to_do;
    my_buffer the_buffer;

    my_producer (my_buffer buf, int count)
    { super();
      the_buffer = buf;
      items_to_do = count;
    }

    public void run ()
    {
        while (items_to_do > 0)
        { System.out.println ("producer to_do = " + items_to_do);
          Integer item = new Integer (items_to_do*items_to_do);
          the_buffer.insert (item);
          items_to_do--;
        }
        System.out.println ("Производитель заканчивает"
          + " работу");
    }
}

class my_consumer extends Thread
{
    int items_to_do;
    my_buffer the_buffer;

    my_consumer (my_buffer buf, int count)
    { super();
      the_buffer = buf;
      items_to_do = count;
    }

    public void run ()
    {
        while (items_to_do > 0)
        { System.out.println ("consumer to_do = " + items_to_do);
          Object item = the_buffer.extract ();
          System.out.println ("consumer got " + item);
          items_to_do--;
        }
        System.out.println ("Потребитель заканчивает работу");
        synchronized (this){
            notify (); // Посылаем уведомление о завершении
                // работы (см. con.wait() в main())
        }
    }
}

public class threaded3
{
    public static void main (String [] args) throws
        InterruptedException
    {
        my_buffer the_buffer = new my_buffer ();
        my_producer prod = new my_producer (the_buffer, 40);
        my_consumer con = new my_consumer (the_buffer, 40);

        Thread.currentThread().setPriority (5);
        prod.setPriority (4); // Производитель получает более
            // высокий приоритет
        con.setPriority (3); // по сравнению с потребителем

        prod.start();
        con.start();
        synchronized (con)
        {
            con.wait(); // Ждем уведомления от производителя
                //об окончании его работы
        }
        System.out.println ("Производитель и потребитель"
            + " закончили работу");
    }
}
    
```

Листинг 11.

- `public void stop ()` вызывается навигатором, когда тот уходит с Web-страницы;
- `public void destroy ()`; этот метод предназначен для освобождения ресурсов; аналог деструктора, но не вызывается автоматически; всегда вызывает `stop()`; всегда вызывается при выходе из навигатора и при перезагрузке апплета.

Простейший апплет выглядит так (листинг 12).

Метод `paint` (строки 4-6) определяет, как апплет перерисовывает себя в тот момент, когда оконный менеджер посылает WWW-навигатору запрос на перерисовку.

Включение апплета в WWW-страницу производится следующим образом. В языке HTML 2.0 предусмотрены специальные конструкции `<applet>` и `<PARAM>`. Первая из них задает имя загружаемого класса и размеры области в окне навигатора, выделяемой апплету. Конструкция `<PARAM>` служит для передачи информации с WWW-стра-

```

1 import java.awt.Graphics;
2 import java.applet.Applet;

3 class SimpleApplet extends Applet {
4     public void paint (Graphics g) {
5         g.drawString (10, 10, "Hello world!");
6     }
7 }

```

Листинг 12.

```

<applet code=SimpleApplet.class width=200 height=100>
<PARAM NAME=font VALUE="TimesRoman">
<PARAM NAME=size VALUE="12">
<h3>Если вы видите этот текст, то ваш навигатор не поддерживает Java </h3>
</applet>

```

Листинг 13.

```

public void init () {
    String fontname = getParameter ("name");
    String fontSizeString = getParameter ("size");
    int theSize = Int.parseInt (fontSizeString);
    ...
}

```

Листинг 14.

ницы в ту среду, в которой будет выполняться апплет.

Листинг 13 содержит простой пример включения апплета в WWW-страницу.

Поскольку WWW-навигаторы игнорируют неизвестные конструкции, в навигаторе, не поддерживающем Java, будет виден текст

Если вы видите этот текст, то ваш навигатор не поддерживает Java

Опросить значения, передаваемые с помощью конструкции `<PARAM>`, можно следующим образом (листинг 14).

4.2.2. Java-машина

Java-компилятор транслирует исходные тексты Java-программ в коды Java-машины. Вообще говоря, Java-машина является виртуальной в том смысле, что она не существует в виде реальных микросхем и других устройств, а представляет собой программный эмулятор, выполняющийся на какой-либо традиционной аппаратной платформе. Вероятно, уже в ближайшее время следует ожидать появления и все более широкого распространения и прямых аппаратных реализаций Java-машины.

Идея языковых процессоров, разумеется, не нова. Известны попытки внедрить так называемый P-код в

качестве стандарта на результат работы Паскаль-компиляторов; в свое время много писали о языке и машине Форт; была выполнена аппаратная реализация рефал-машины, и список этот можно продолжать и продолжать.

В контексте проекта Java спецификация виртуальной машины является частью комплекса мер, направленных на стандартизацию Java-среды и на обеспечение ее независимости от аппаратно-программной платформы. Кроме того, следует учитывать ту специфическую среду, в которой должны готовиться и работать Java-программы. Если Web-страница содержит Java-апплеты, эти апплеты будут передаваться по сети. Значит, весьма желательно, чтобы Java-код был как можно более компактным; в противном случае время загрузки страницы рискует стать раздражающе большим. Соответственно, архитектура и система команд Java-машины проектировались таким образом, чтобы всячески способствовать компактификации кода. С другой стороны, формат команд Java-машины довольно прост (обычно команды не имеют операндов и занимают один байт), поэтому возможна ее (машины) эффективная эмуляция. По этой причине программы, подготовленные для выполнения на Java-машине, часто называют байт-кодами.

Мы опишем архитектуру Java-машины довольно кратко. Последующее изложение опирается на версию спецификаций 1.0.

4.2.2.1. Типы данных, поддерживаемые Java-машиной

Java-машина поддерживает следующие стандартные типы данных:

- `byte` — однобайтные целые числа в двоичном дополнительном коде;
- `short` — двухбайтные целые числа;
- `int` — четырехбайтные целые числа;

- long — восьмибайтные целые числа;
- float — четырехбайтные вещественные числа в формате IEEE-754;
- double — восьмибайтные вещественные числа;
- char — двухбайтные беззнаковые символы в кодировке Unicode.

Поскольку Java-компилятор в состоянии проверить типы данных во время трансляции, при выполнении нет нужды ассоциировать дополнительную информацию со значениями стандартных типов. Вместо этого генерируются команды, рассчитанные на обработку данных определенных типов. Например, для сложения целых чисел будет сгенерирована команда `iadd`, а для сложения вещественных чисел двойной точности — команда `dadd`.

Значения типа `boolean` представляются однобайтными целыми числами и обрабатываются посредством соответствующих команд.

Имеется еще два стандартных типа данных:

- `object` — четырехбайтная ссылка на объект (массивы трактуются как объекты);
- `returnAddress` — четырехбайтный адрес возврата из метода.

Спецификации Java-машины не описывают внутренней структуры объектов. В реализации Sun Microsystems значение типа `object` указывает на описатель, хранящий две ссылки — на таблицу методов и на данные объекта. Возможны и другие представления.

Java-машина является 32-битной. Более длинные значения (`long`, `double`) представляются как пара четырехбайтных величин. Не оговаривается, в каком порядке располагаются элементы пары; более того, верификатор байт-кодов обязан выявлять и отвергать программы, пытающиеся "вручную" составлять длинные значения.

4.2.2.2. Регистры

В Java-машине должны поддерживаться следующие регистры:

- `pc` — счетчик команд; указывает на код операции для команды, которая будет выполняться следующей.
- `vars` — базовый регистр для доступа к локальным переменным текущего метода.
- `optop` — указатель на вершину стека операндов. Java-машина является стековой, поэтому основная часть команд берет операнды из стека и туда же помещает результат.
- `frame` — указатель на структуру, содержащую окружение времени выполнения.

В свою очередь, окружение времени выполнения используется для реализации трех целей: динамической загрузки, возврата из методов и обработки исключительных ситуаций.

Для обеспечения динамической загрузки, окружение времени выполнения содержит ссылки на таблицу символов текущего метода и текущего класса. Перед началом выполнения метода производится редактирование его внешних связей (настройка ссылок на внешние методы и внешние данные). Подобная поздняя настройка ссылок делает сгенерированный код устойчивым по отношению к изменениям во внешних классах.

Для обеспечения нормального возврата из методов выполняется восстановление регистрового окружения вызывающего метода.

Для обработки исключительных ситуаций Java-машина выполняет проход по стеку вызовов методов и отыскивает самую внутреннюю конструкцию `catch`, обрабатывающую случившееся событие.

В принципе окружение времени выполнения может

содержать дополнительную информацию, необходимую, например, для отладки, но в спецификациях Java-машины это оставлено на усмотрение авторов реализации.

4.2.2.3. Сбор мусора

Для создания объектов во время выполнения выделяется область динамической памяти. Язык Java рассчитан на то, что эту область обслуживает сборщик мусора, поскольку в языке нет средств для освобождения памяти. Как именно работает сборщик мусора, определяется реализацией Java-машины.

4.2.2.4. Система команд Java-машины

Команда Java-машины состоит из однобайтного кода операции, за которым следуют операнды (если таковые имеются). Можно выделить следующие группы команд:

- команды загрузки констант и переменных в стек операндов. Для каждого типа данных имеются свои команды загрузки. Например, команда с кодом операции `dload` и операндом, задающим смещение, загружает в стек из локальной переменной вещественное число двойной точности, а команда `aload` делает то же для ссылки на объект.
- команды запоминания данных из стека в локальных переменных.
- команды управления массивами. Например, команда `newarray` с операндом, задающим тип элементов, извлекает из стека требуемый размер массива, создает его и помещает в стек ссылку на массив. Отметим, что для создания массивов с элементами-объектами служит другая команда, `newobjectarray`. За счет подобной специализации достигается эффективность интерпретации Java-программ.

- команды работы со стеком. К этой группе относятся команды, которые удаляют, дублируют, меняют местами верхние элементы стека операндов, а также выполняют другие, более сложные манипуляции со стеком.
- арифметические команды. Операнды извлекаются из стека; туда же помещается результат.
- логические команды (сдвиг, и, или, исключаящее или).
- команды преобразования к другому типу.
- команды передачи управления. Например, в команде `jsr` (переход на подпрограмму) операндом служит относительный адрес перехода; адрес команды, следующей за `jsr`, помещается на вершину стека операндов. Имеются команды для реализации переключателей.
- команды возврата из функции. Для возврата результатов разных типов используются команды с разными кодами операции. Кроме того, имеется команда `breakpoint`, которая останавливает нормальный ход выполнения и передает управление обработчику этого события.
- команды манипулирования с полями объектов (установить/прочитать обычное/статическое поле).
- команды вызова методов. Их четыре. Команда `invokevirtual` вызывает (виртуальный) метод на основе анализа информации времени выполнения. Команда `invokenonvirtual` осуществляет вызов на основе информации времени компиляции — например, вызов метода родительского класса. Команда `invokestatic` вызывает статический метод класса. Наконец, команда `invokeinterface` вызывает метод, представленный интерфейсом.

Выполнение всех перечисленных команд связано не только с передачей управления, но и с анализом разного рода таблиц.

- команда возбуждения исключительной ситуации — `athrow`.
- прочие объектные операции (создать объект, проверить тип объекта).
- команды синхронизации (войти в критический интервал, выйти из него).

Мы видим, что не существует семантического разрыва между языком Java и Java-машиной. Как уже отмечалось, это важно для компактности скомпилированных Java-программ и для обеспечения высокой скорости трансляции.

4.2.3. Java и безопасность

Концепция загрузки объектов по сети прозрачным для пользователя образом столь же привлекательна, сколь и опасна. Если не предпринимать никаких мер и не накладывать никаких ограничений на возможности Java-апплетов, вход на любую Web-страницу может привести к непредсказуемым последствиям. К счастью, разработчики языка Java с самого начала уделяли самое пристальное внимание вопросам информационной безопасности.

Из языка удалены многие потенциально опасные возможности, такие как оператор `goto` или тип данных "указатель". Интерпретируемый характер выполнения позволяет не допустить выхода за границы массива, обращения по пустой ссылке и т.п. В свое время за подобную осторожность выступал автор языка Паскаль Никлаус Вирт, отмечавший, что при традиционном подходе программист напоминает моряка, который носит спасательный круг только на суше.

Мы, однако, не будем подробно останавливаться на

"обычной" безопасности и уделим основное внимание выполнению потенциально враждебных апплетов. Смежный вопрос — проверка подлинности апплетов, снабженных электронной подписью, видимо, будет решен в последующих версиях Java-систем.

Прежде всего, апплетам, загруженным по сети, запрещены чтение и запись файлов из локальной файловой системы, а также выполнение сетевых соединений со всеми хостами, кроме того, с которого был получен апплет. Кроме того, таким апплетам не разрешается запускать программы на клиентской системе (говоря языком ОС UNIX, для них недоступны системные вызовы `fork` и `exec`), им запрещено загружать новые библиотеки и вызывать программы, внешние по отношению к Java-машине.

На самом деле, перечисленные ограничения не являются частью спецификации Java-системы и могут выполняться с большей или меньшей аккуратностью. Так, в Netscape Navigator 2.0 чтение и запись локальных файлов действительно полностью запрещены. В то же время, среда разработки JDK 1.0 компании Sun Microsystems допускает задание списка каталогов, с которыми апплеты могут работать.

Более точно, вне разрешенного списка каталогов апплет не может:

- проверять существование файлов;
- читать/писать/переименовывать файлы;
- создавать каталоги;
- проверять атрибуты файла — тип, время последней модификации, размер.

Чтобы в JDK сделать каталог доступным для апплета, следует поместить в файл `~/hotjava/properties` строки вида

```
acl.read=/home/welcome
acl.write=/tmp
```

Перед началом работы аплетов они проверяются верификатором байт-кодов. Верификатор убеждается, что загруженный аплет соответствует спецификациям, заданным при компиляции вызывающей программы, что не нарушен формат скомпилированного файла, что нет переполнения или исчерпания стека, нет некорректных преобразований типов, неправильных действий с регистрами и т.п. Все эти проверки верификатор осуществляет на основе анализа потоков данных. Особенно тщательно проверяются конструкции `finally` обработчиков исключительных ситуаций.

Следует отметить, что верный выбор баланса между возможностями загружаемых аплетов и безопасностью клиентской системы является очень тонким вопросом. Ряд компаний, например, Argus System Group, предлагают реализовать на клиентской системе усиленные меры безопасности, чтобы успешно отражать угрозы со стороны враждебных аплетов без ограничения свободы действий для "благонадежных" программ. К сожалению, предлагаемые решения зависят от операционной платформы, что противоречит требованию абсолютной переносимости Java-программ. Можно предположить, что информационная безопасность еще долгое время будет оставаться одним из самых сложных и спорных вопросов, касающихся проекта Java.

4.2.4. Java WorkShop

В конце марта 1996 года компания Sun Microsystems объявила о выпуске версии 1.0 среды разработки Java WorkShop. У этой среды есть два замечательных свойства: она полностью написана на языке Java и имеет интерфейс, выдержанный в Web-стиле. На рис. 11 показан вид экрана при работе в Java WorkShop.

Следует отметить, что гипертекстовый интерфейс является, пожалуй, наиболее естественным для инструментальных сред. Более того, неявно он давно используется, например, при переходе во включаемый файл или в место ошибки. Принципиально важно, что теперь гипертекст стал явной концептуальной основой.

Java WorkShop содержит полный набор инструментов, необходимых для проектирования, разработки, тестирования, отладки и сопровождения программ. В его состав входят:

- Менеджер проектов — инструмент организации информации, составляющей проект, а также средство спецификации окружения для проекта.
- Построитель — инструмент построения результирующих программ проекта. Построитель ведет перекомпиляцией файлов после внесения изменений, выдачей гипертекстового списка сообщений об ошибках и т.п.
- Публикатор — инструмент поддержки коллективной работы над проектами. Позволяет организовать хранилище проектов, предоставлять Web-страницы проектов для использования другими программистами, осуществлять доступ к проектам коллег, задавать права доступа к проектам.
- Просмотрщик аплетов — средство контролируемого выполнения Java-программ.
- Просмотрщик исходных текстов — инструмент изучения программ с учетом их объектной структуры.
- Редактор исходных текстов. Редактор интегрирован с другими компонентами Java WorkShop, а также с популярными системами управления версиями.
- Отладчик. Помимо традиционных возможностей, отладчик Java WorkShop позволяет конт-



Рис. 11. Так выглядит экран при работе в Java WorkShop.

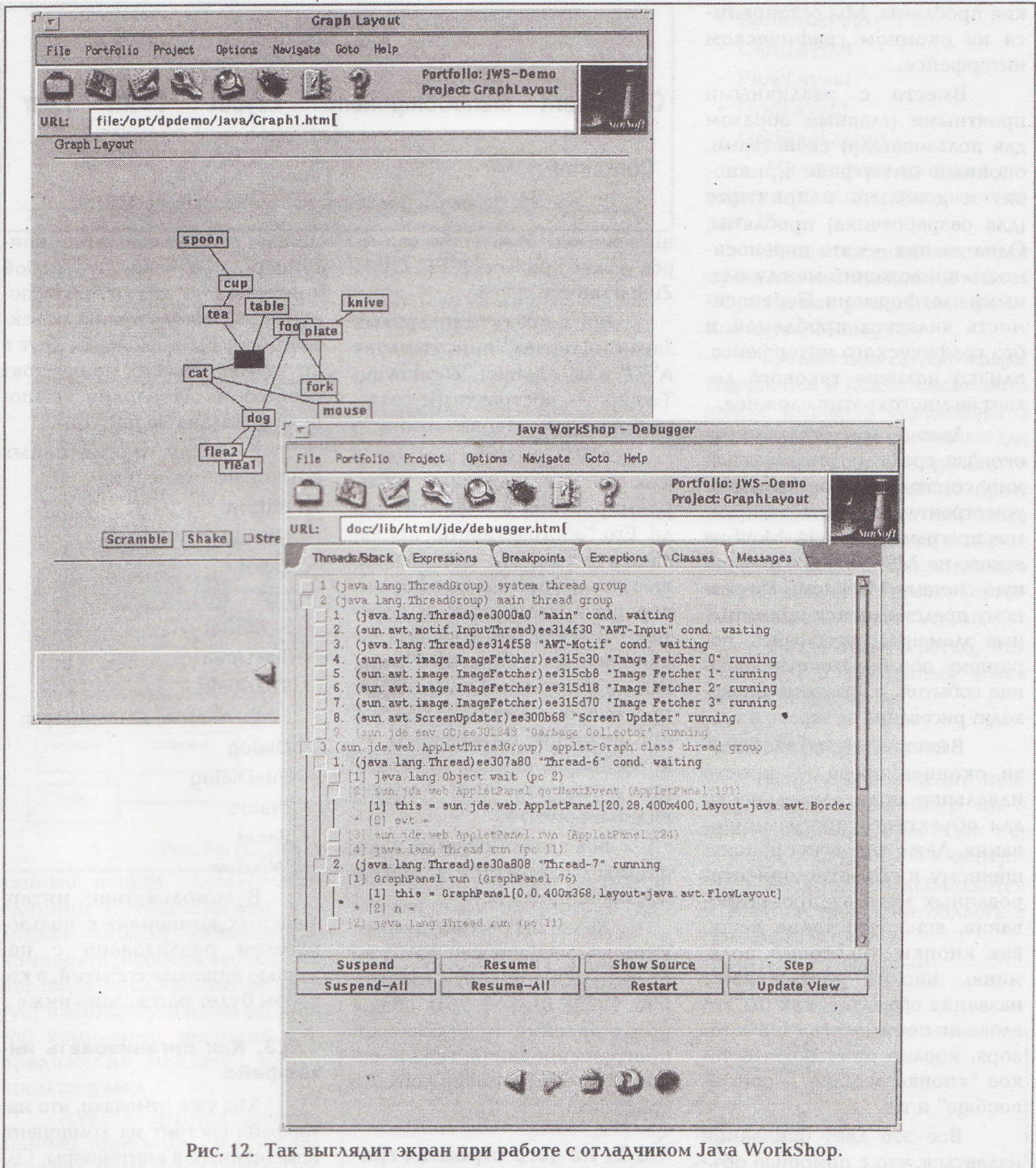


Рис. 12. Так выглядит экран при работе с отладчиком Java WorkShop.

ролировать состояние нескольких потоков выполнения, что необходимо для развитых Java-программ. На рис. 12 представлен образ экрана при работе в отладчике.

- Справочная подсистема.

Java WorkShop может работать как на платформе SPARC/Solaris и Intel/Solaris, так и под Microsoft Windows 95/NT. Каких-то особых требований к аппаратуре

Java WorkShop не предъявляет. Достаточно 45 Мб дискового пространства, 32 (для Solaris) или 16 (для Windows) Мб оперативной памяти.

Замечательно, как в правильно спроектированной системе различные компоненты не просто идеально подходят друг к другу но и усиливают мощь друг друга.

4.3. Связь Java с окружением: оконный инструментари

4.3.1. Общие положения

Одно из важных достоинств Java состоит в том, что это не только язык, но и стандартизованная объектно-ориентированная среда выполнения. Любопытно проследить, как в рамках Java решаются традиционные программистс-

кие проблемы. Мы остановимся на оконном графическом интерфейсе.

Вместе с различными приятными (главным образом для пользователя) свойствами, оконный интерфейс привносит и довольно неприятные (для разработчика) проблемы. Одна из них — это переносимость приложений между разными платформами. Переносимость является проблемой и без графического интерфейса, однако наличие такового делает ее многократно сложнее.

Дело в том, что каждая оконная среда — это сложный мир, со своими законами, набором строительных блоков и приемов программирования. Motif не похож на MS-Windows и оконную систему Macintosh. По-разному представляются примитивные элементы интерфейса, по-разному обрабатываются внешние события, по-разному происходит рисование на экране и т.д.

Вместе с тем, по своей сути оконная среда — просто идеальное поле деятельности для объектного программирования. Даже человеку, неискушенному в объектно-ориентированных методах проектирования, ясно, что такие вещи, как кнопки, текстовые поля, меню, вполне заслуживают названия объектов, как бы это слово ни понималось. Иначе говоря, вполне понятно, что такое "кнопка вообще", "список вообще" и т.д.

Все это дает основания надеяться, что с помощью объектно-ориентированного подхода можно получить по-настоящему высокоуровневую и переносимую оконную среду, основанную на абстрактных типах данных.

Данная особенность оконных сред проявилась, в частности, в появлении довольно большого количества различных классовых библиотек, "обертывающих" оригинальные окон-

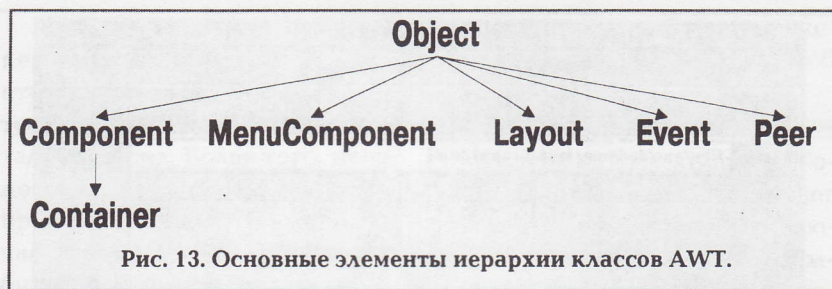


Рис. 13. Основные элементы иерархии классов AWT.

ные системы. В качестве примеров можно привести MFC, OWL, Zink и многие другие.

Вот и среди стандартных Java-библиотек присутствует AWT или Abstract Windowing Toolkit — абстрактный оконный инструментальный.

AWT является системой классов для поддержки программирования в оконной среде. Его "абстрактность" проявляется в том, что все, зависящее от конкретной платформы, хорошо локализовано и спрятано. В AWT реализованы такие простые и понятные вещи, как кнопки, меню, поля ввода; простые и понятные средства организации интерфейса — контейнеры, панели, менеджеры геометрии. Это хорошо видно на рис. 13.

Все зависимости от платформы содержатся в ветви, обозначенной как Peer.

Далее мы рассмотрим некоторые особенности AWT, не претендуя на полноту изложения. Наша цель — дать общее представление о технологии программирования графического оконного интерфейса в среде Java.

4.3.2. Из чего строится графический интерфейс (компоненты и контейнеры)

Если посмотреть на любое оконное приложение, то легко увидеть, что интерфейсная часть состоит из объектов, объединенных в группы. В AWT объекты называются компонентами (на самом деле они все являются наследниками класса Component), а группы объектов реализованы с по-

мощью так называемых контейнеров. Отметим, что любой контейнер — это тоже компонента, поэтому группы объектов могут быть вложены друг в друга. Как обычно, меню стоят особняком. Иерархия компонент показана на рис. 14.

К числу примитивных компонент относятся:

- Button
- Checkbox
- Label
- List
- ScrollBar
- TextArea
- TextField

Основные контейнеры:

- Dialog
- FileDialog
- Frame
- Panel
- Window

Взаимодействие интерфейсных компонент с пользователем реализовано с помощью аппарата событий, о котором будет рассказано ниже.

4.3.3. Как организовать интерфейс

Мы уже отмечали, что интерфейс состоит из компонент, помещенных в контейнеры. Однако, остается открытым вопрос о том, как размещать компоненты друг относительно друга внутри контейнера. Наивный подход (принятый, тем не менее, во многих системах) заключается в задании относительных координат компонент в контейнере. Вариации этого подхода состоят, как правило, в возможности задавать различные единицы длины (пиксели, ты-

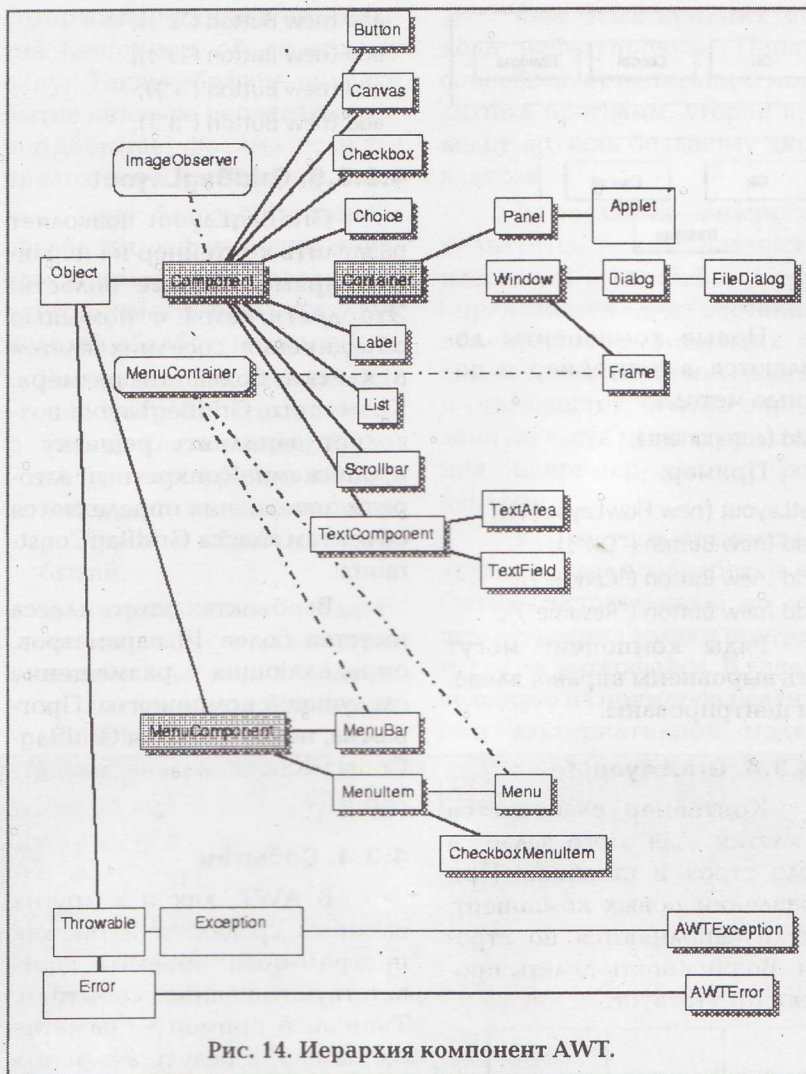


Рис. 14. Иерархия компонент AWT.

сячные дюйма, "диалоговые единицы"). Недостатком подобной модели размещения компонент является то, что при переводе приложения на другую платформу и даже на другой компьютер, внешний вид приложения меняется плохо предсказуемо.

AWT старается решить данную проблему следующим образом. Если нельзя избежать изменений внешнего вида приложений при запуске их на разных платформах, надо постараться хотя бы сделать эти изменения предсказуемыми и в некотором смысле естественными. Для проведения этой программы в жизнь используются объекты под общим названием Layout (расположение). Layout управляет тем, как компоненты будут располагаться

внутри контейнера, следуя определенным принципам или рецептам.

Всего в AWT имеется 5 предопределенных расположений:

- BorderLayout
- CardLayout
- FlowLayout
- GridLayout
- GridBagLayout

Их местоположение в иерархии классов показано на рис. 15.

Расположения являются прямыми наследниками класса Object и все реализуют протокол LayoutManager.

Объект типа LayoutManager содержится в классе Container. В классе Container определены 3 метода add, предназначенные для добавления новых компонент в контейнер. Куда именно будет добавлена компонента с помощью метода add, зависит от конкретного типа LayoutManager данного контейнера. Для задания LayoutManager есть метод setLayout.

Рассмотрим кратко, как работают приведенные выше схемы размещения.

4.3.3.1. BorderLayout

BorderLayout делит прямоугольный контейнер на 5 прямоугольных областей: север, юг, запад, восток и центр:

Компонента добавляется в соответствующую область с помощью метода

```
add (string, component);
```

Пример:

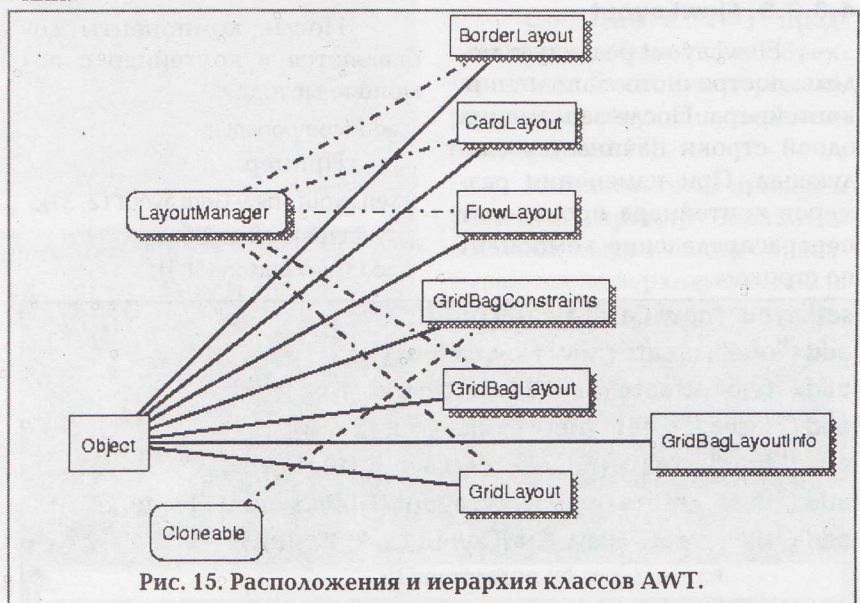
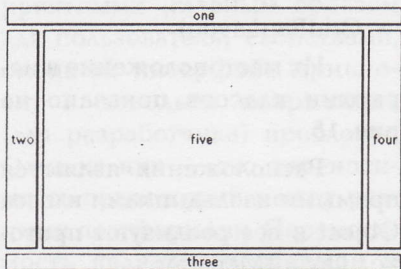


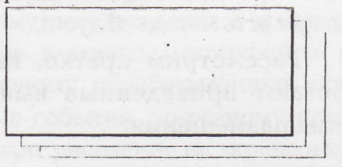
Рис. 15. Расположения и иерархия классов AWT.

```
setLayout (new BorderLayout());
buttonN = new Button ("one");
add ("North", buttonN);
// ... добавление остальных кнопок
```



4.3.3.2. CardLayout

Контейнер состоит из нескольких плоскостей. В каждый момент наверху только одна плоскость. Таким образом, CardLayout представляет собой как бы блокнот со многими страницами. Страницы помечаются с помощью строк.



Метод добавления компонент к контейнеру:

```
add (string, component);
```

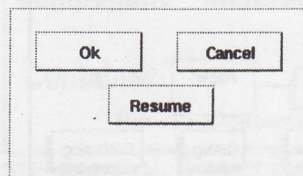
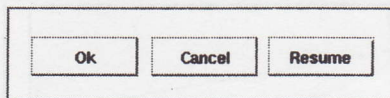
где string — имя "страницы", а component — добавляемая компонента. Как правило, эти компоненты сами будут контейнерами (страница блокнота вряд ли будет состоять из одной большой кнопки). Пример приведен на листинге 15.

4.3.3.3. FlowLayout

FlowLayout реализует модель построчного заполнения контейнера. После заполнения одной строки начинается следующая. При изменении размеров контейнера происходит перераспределение компонент по строкам.

```
setLayout (new CardLayout ());
add ("one", create (new FlowLayout ()));
add ("two", create (new BorderLayout ()));
add ("three", create (new GridLayout (2, 2)));
add ("four", create (new BorderLayout (10, 10)));
add ("five", create (new FlowLayout (FlowLayout.LEFT, 10, 10)));
add ("six", create (new GridLayout (2, 2, 10, 10)));
```

Листинг 15.



Новые компоненты добавляются в контейнер с помощью метода

```
add (component);
```

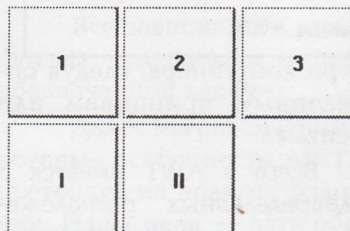
Пример:

```
setLayout (new FlowLayout ());
add (new Button ("OK"));
add (new Button ("Cancel"));
add (new Button ("Resume"));
```

Ряды компонент могут быть выровнены вправо, влево, или центрированы.

4.3.3.4. GridLayout

Контейнер разбивается на клетки. Для этого задается число строк и столбцов. При добавлении новых компонент, клетки заполняются по строкам. Возможность делать пропуски отсутствует.



Новые компоненты добавляются в контейнер с помощью метода

```
add (component);
```

Пример:

```
setLayout (new GridLayout (2, 3));
// 2 строки, 3 столбца
add (new Button ("1"));
```

```
add (new Button ("2"));
add (new Button ("3"));
add (new Button ("I"));
add (new Button ("II"));
```

4.3.3.5. GridBagLayout

GridBagLayout позволяет разделить контейнер на неравные прямоугольные области. Это достигается с помощью объединения соседних клеток в клетки большего размера. Кроме того, GridBagLayout позволяет заполнять решетку с пропусками. Конкретный алгоритм заполнения определяется объектом класса GridBagConstraints.

В объектах этого класса имеется более 10 параметров, определяющих размещение следующей компоненты. Программа, использующая GridBagConstraints, показана на листинге 16.

4.3.4. События

В AWT, как и в других оконных средах, события как программные объекты соответствуют внешним событиям. Типичный пример — нажатие на кнопку, в результате чего в AWT происходит событие ACTION_EVENT.

Таким образом, интерактивная часть программ в AWT соответствует модели программы, управляемой событиями. Другими словами, приложения или апплеты должны отслеживать события и обрабатывать их.

Рассмотрим подробнее, как это происходит в AWT. Каждый оконный интерфейс можно рассматривать как набор (возможно, вложенных друг в друга) компонент. Когда происходит событие, связанное с каким-то элементом интерфейса (например, кнопкой), то вызывается метод handleEvent. По умолчанию этот метод ничего не делает, а просто передает событие вверх, то есть тому объекту, в котором

содержится инициатор события (например, объемлющему окну). Таким образом, если событие никто не перехватывает, оно достигает фрейма (если это приложение) или апплета.

Этот базовый механизм обработки событий при наивном применении навязывает программисту одну из двух моделей обработки событий.

- Во-первых, можно позволить всем событиям всплывать до самого верха и уже на уровне фрейма писать метод, который занимается разбором и обработкой событий.
- Во-вторых, можно создать множество специализированных классов, в каждом из которых переопределен метод `handleEvent`.

Оба этих крайних подхода небезупречны. Первый способствует написанию монолитных программ, второй приводит к очень большому числу классов.

В реальной жизни используется компромиссная идеология — `handleEvent` переопределяется для составных объектов, объединяющих несколько элементов интерфейса и отвечающих за ясно определенную часть работы приложения (например, разного рода диалоги).

Кроме того, возможны и другие модели обработки событий, направленные на лучшее отделение логики программы от ее интерфейса. В качестве одного из примеров реализации альтернативной модели

можно указать пакет "The Command Class for AWT objects", который написал Jan Newmarch, `jan@ise.canberra.edu.au`.

4.3.5. Методы класса Component, связанные с обработкой событий

Ниже перечислены методы класса `Component`, связанные с обработкой событий.

- `postEvent()`. Этот метод, как правило, вызывается автоматически при наступлении соответствующего события.
- Обрабатываются события при помощи метода `handleEvent()`. По умолчанию этот метод, в зависимости от типа события, вызывает один из специализированных обработчиков событий, таких как
 - `action()`
 - `keyUp()`
 - `mouseDown()`
 - `gotFocus()`
 - и т.д.

Метод `handleEvent()` может быть переопределен. В этом случае, как правило, вся обработка соответствующего события будет происходить именно в этом методе, однако, если `handleEvent()` возвращает `false`, то вызывается `handleEvent()` для контейнера, содержащего данную компоненту.

События в AWT представляются с помощью объектов класса `Event`, в котором определены следующие переменные (листинг 17).

На листинге 18 приведен типичный пример обработки событий, когда все они обрабатываются на верхнем уровне (в данном случае на уровне апплета).

В этом примере, если инициатор события имеет тип `Button` с именем "ОК", то выполняются соответствующие действия и возвращается значение `true`, то есть событие дальше не передается.

```
GridBagLayout gridbag = new GridBagLayout ();
GridBagConstraints c = new GridBagConstraints ();
setLayout (gridbag);
```

```
// Устанавливаем характеристики
c.fill = GridBagConstraints.BOTH;
c.weightx = 1.0;
...
Button button1 = new Button ("1");
gridbag.setConstraints (button1, c);
add (button1);
...
// здесь можно изменить некоторые установки
// объекта класса GridBagConstraints,
// например
// c.fill = GridBagConstraints.NONE;
Button buttonX = new Button ("X");
gridbag.setConstraints (buttonX, c);
add (buttonX);
```

Листинг 16.

```
public Object target; // инициатор события
public long when; // время, когда событие произошло
public int id; // тип события (KEY_PRESS, MOUSE_DOWN...)
public int x; // координаты
public int y; // курсора
public int key; // код клавиши
public int modifiers; // код модификатора (control, shift ...)
public Object arg; // вспомогательные данные
public Event evt; // поле для соединения событий в списки
```

Листинг 17.

```
class MyApplet extends Applet {
...
public boolean action (Event evt, Object arg) {
...
if ((ev.target instanceof Button) && arg.equals ("OK")) {
// Выполнить соответствующие действия
...
return true;
} else {
// Другие случаи
...
}
...
return false;
}
...
}
```

Листинг 18.

```
class OKButton extends Button {
...
public boolean action (Event e, Object arg) {
// Выполнить соответствующие действия
...
return true;
}
}
```

Листинг 19.

```
1 import java.awt.Font;
2 import java.awt.Color;

3 public class JAVAhello extends java.applet.Applet {
4   Font f = new Font ("TimesRoman", Font.BOLD, 36);

5   public void init () {
6     resize (150, 25);
7   }

8   public void paint (Graphics g) {
9     g.setFont (f);
10    g.setColor (Color.blue);
11    g.drawString ("Hello, World!", 50, 25);
12  }
13 }
```

Листинг 20.

Другой способ обработать событие заключается в создании специализированной компоненты (см. листинг 19).

Мы видим, насколько глубоко продумана в проекте Java

связь с операционным окружением. Удалось достичь удачного сочетания богатства возможностей с переносимостью.

5. Joe – технология связывания Java-программ с объектными бизнес-приложениями

Выше, в разделе "Java, Joe и NEO", мы писали о том, что новый продукт компании SunSoft – Joe – призван осуществлять связь между клиентскими компонентами, написанными на языке Java, и объектными серверами приложений, созданными в среде NEO. При этом Joe берет на себя все (или почти все) технические проблемы, связанные с работой в распределенной объектной среде, обеспечивая в то же время для Java-программ полноценное взаимодействие с произвольными объектными сервисами.

В предыдущих выпусках Jet Info была опубликована шутивная "Эволюция программиста". Мы добавим к ней еще два раздела. Первый из приводимых ниже примеров (листинг 20) написан на языке Java и может использоваться в составе обычных, локально работающих апплетов.

Теперь модифицируем программу для Joe (листинг 21), чтобы обеспечить возможность работы в среде клиент/сервер.

Поясним смысл новых строк, появившихся во втором варианте программы.

В строке 1 импортируется описание предоставляемых объектных сервисов.

В строке 5 объявляется, что класс JOEhello будет наследником JOEApplet.

В строке 6 декларируется обобщенная (не типизированная) ссылка на NEO-объект. Она будет использована для сохранения результата поиска объекта по имени.

В строке 7 описана типизированная ссылка на NEO-объект, которая будет использоваться для манипулирования удаленным объектом (в частности, для вызова методов) средствами языка Java.


```

1 import sunw.services.*;
2 import java.awt.Graphics;
3 import java.awt.Font;
4 import java.awt.Color;

5 public class JOEhello extends sunw.services.JOEApplet {
6     sunw.corba.ObjectRef obj;
7     Hello.HelloWorldRef hiThere;

8     Font f = new Font("TimesRoman", Font.BOLD, 36);

9     public void init () {
10        super.init ();
11        resize (150, 25);

12        obj = find ("HelloWorldServer");
13        hiThere = Hello.HelloWorldStub.narrow (obj);
14    }

15    public void paint (Graphics g) {
16        g.setFont (f);
17        g.setColor (Color.red);
18        // Получим строку из удаленного объекта, реализованного на
19        // C++, и выведем ее.
20        g.drawString (hiThere.sayHello (), 50, 25);
21    }

```

Листинг 21.

В строке 12 мы получаем от службы имен NEO ссылку на нужный нам серверный объект.

В строке 13 выполняется операция приведения обобщенной ссылки к типизированному виду.

Наконец, в строке 18 выполняется вызов метода удаленного объекта, написанного, вообще говоря, не на языке Java (например, на C++). Впрочем, вид этого оператора не зависит ни от удаленности, ни от языка реализации объекта.

Процесс разработки программ в среде Joe довольно прост. Он состоит из следующих этапов:

- Трансляция интерфейса к NEO-объектам, написанного на языке IDL (Interface Definition Language), в класс на языке Java. Этот этап поддержан вхо-

дящим в состав Joe компилятором IDL — Java. Результат трансляции содержит суррогатные методы, которые в совокупности с брокером объектов обеспечивают прозрачное взаимодействие с удаленными объектами. В свою очередь, брокер объектов, входящий в состав Joe, прозрачным для клиентской стороны образом загружается в WWW-навигатор вместе с Java-апплетами.

- Написание клиентского кода на языке Java с использованием сгенерированного интерфейса и средств Joe для взаимодействия с удаленными объектами.

Дальше Java-программа компилируется и выполняется обычным образом.

Подчеркнем, что описанная объектная среда обеспечивает полноценное, двустороннее взаимодействие между клиен-

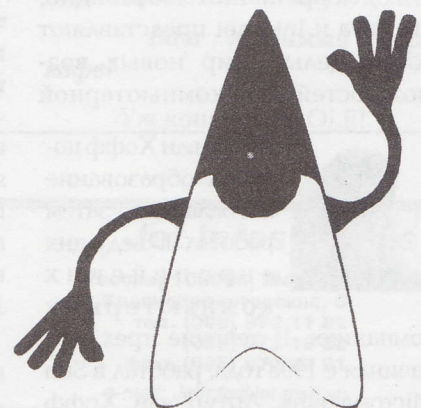
том и сервером. В частности, удаленные серверные объекты могут вызывать методы в объектах-клиентах. Для обеспечения этой возможности в составе Joe имеется транслятор Java-классов в IDL-интерфейсы.

6. Заключение

Инtranет и Java — вот два ключевых слова, символизирующих современный этап развития информационных технологий. Инtranет позволяет пересмотреть подход к пользованию информационными ресурсами, что в огромной степени увеличивает производительность труда отдельных работников и компаний в целом. Кроме того, технология Инtranет позволяет добиться невиданной ранее масштабируемости — от локальной сети до Интернет.

Java снимает ограничения Web-сервиса, делая последний интерактивным и объектно-ориентированным. Java вводит новый — абсолютный — стандарт на переносимость программного обеспечения. Примечательно, что беспрецедентные по своей мощи возможности вводятся при полном сохранении информационной безопасности.

В сочетании с продуктами Sun Microsystems — Joe и NEO — Java обеспечивает распространение концепции Инtranет на произвольные сервисы, что открывает реальную возможность создания корпоративных информационных систем нового поколения.



Артур ван Хофф: Java и Интернет – это целый мир новых возможностей

Интервью одного из разработчиков Java главному редактору бюллетеня Jet Info Express Леониду Черняку

Internet, Web и Java представляют собой сдвиг парадигмы в информационных технологиях. Общественная реакция на этот сдвиг не имеет прецедента в компьютерной индустрии – ведущие компании объявили о поддержке Java, на эту тему в компьютерной и деловой прессе было опубликовано невероятное количество статей, стоимость акций тех компаний, которые связаны с этими технологиями, резко подскочила, технические руководители банков и крупных корпораций поддерживают применение Java в пилотных проектах.

В то же время все существующие сегодня приложения этих технологий ограничены в основном небольшими задачами, такими как маркетинг в Internet, распространение информации, Java-игры и т.д. И даже предполагаемые в будущем приложения не отличаются радикальной новизной.

Что же особенного в этих технологиях, чем можно на самом деле объяснить те реакцию и интерес, которые мы можем наблюдать сегодня?

Современно очевидно, что Java и Internet представляют собой целый мир новых возможностей для компьютерной

индустрии, однако существующие сегодня приложения пока остаются простыми, потому что еще нет соответствующего опыта в решении некоторых внутренних проблем. Тем не менее, рынок очень быстро сдвигается в направлении этого нового мира, ежедневно объявляются новые продукты и технологии. Именно это и свидетельствует о том, что Java и Internet представляют собой огромный сдвиг парадигмы, создающий новые возможности. Это событие сравнимо по масштабу с появлением персональных компьютеров, но сегодня оно захватило всех гораздо быстрее.

В чем же специфика Java и Internet? Я думаю, что они в состоянии обеспечить компьютерной индустрии и пользователям именно то, чего те всегда ожидали: простой доступ к неограниченным объемам информации и приложений.

Как любая технология, Java имеет союзников и оппонентов. Кто, по Вашему мнению, представляет наибольшую опасность для будущего Java (продукт, компания или кто-то лично)? Каковы могут быть ответные конкурирующие начинания оппонентов Java?

Наибольшая опасность состоит в том, что в ближайшие

годы мы должны прийти к новой среде, целиком написанной на Java, но с библиотеками от разных поставщиков. Это значит, что мы получим лишь новый, улучшенный язык реализации и ничего более. Задача же состоит в том, чтобы создать общепринятую среду, не зависящую от платформы.

В то же время, было бы глупо с моей стороны, будучи сторонником Java, делиться с моими оппонентами идеями о том, как строить атаку на Java, не так ли?

Каким будет влияние сдвига компьютерной парадигмы, представленной Internet, Web и Java, на основных действующих лиц текущей парадигмы (IBM, Sun, HP, DEC, Compaq, Oracle и других)?

В современных условиях покупка компьютера приводит к пожизненной необходимости для владельца покупать программное обеспечение у одного поставщика. Это не может длиться бесконечно. Если все компьютеры смогут выполнять любое программное обеспечение на Java, то замок, в виде привязанности к определенной платформе, будет снят и такие монополисты, как Microsoft, будут вынуждены сражаться на равных с более мелкими фирмами. Таким образом будет выровнено игро-



Артур ван Хофф получил образование в Голландии, затем работал в ведущих европейских компьютерных компаниях. В течение трех лет, начиная с 1993 года, работал в Sun Microsystems. Артур ван Хофф

принимал участие в разработке языка Java, в проектировании программного интерфейса для приложений на Java, им написан первый Java-компилятор на языке Java.

В 1996 году он и его коллеги Сэми Шайо и Ким Полезе образовали независимую компанию с

намерением разрабатывать программное обеспечение на Java.

Артур ван Хофф принимает активное участие в популяризации Java, он участвует в наиболее известных конференциях и является автором ряда статей и готовящейся к изданию весной 1996 года книги "Hooked on Java".

вое поле и возникнет среда с большими возможностями для конкуренции, а результатом, в конечном итоге, будут продукты лучшего качества, в большей степени ориентированные на потребителя.

Россия, наряду с Китаем и Индией, — одна из стран, хорошо известных высоким профессиональным уровнем программистов. К сожалению, российские программисты опоздали к первой волне "гаражных компьютеров" в начале 80-х годов. Затем наступила полоса доминирования на рынке больших компаний, попасть на рынок и добиться каких-то успехов стало практически невозможно. Широко распространено мнение о том, что с приходом Java возникнет новая волна "гаражного программирования", и в ней смогут принять участие и российские программисты. Какое, по вашему мнению, направление в программных продуктах и приложениях на Java наиболее перспективно для "гаражных программистов"?

Именно сегодня спрос на Java-программистов чудовищен. Глобальность Internet дает любому программисту возможность претендовать на участие в любом проекте, в любой части света на равных. Что нам реально нужно, так это всемирные организации разработчиков, которые помогали бы отдельным программистам участвовать в разнообразных проектах.

Какие новые технологии принесет с собой Java?

Появление нового языка дает возможность, приобретя новый опыт, спроектировать за-

ново существующие прикладные программные интерфейсы, например, для 3D-программирования. Используя Java, можно построить объектно-ориентированные и многопоточные 3D-библиотеки, которые на других языках реализовать сложно. Существующие библиотеки состоят из сотен вызовов функций и поэтому ими трудно управлять. Если переписать их на Java, то 3D-программирование станет намного доступнее для рядовых программистов.

Сегодня об Internet, Web и Java пишут многие, и компьютерные профессионалы, и журналисты. Вы — один из основных участников проекта Java и обладаете уникальной возможностью увидеть возникающие при этом проблемы. Есть ли нечто существенное, что многие люди упускают или недооценивают в этих технологиях, в их влиянии на компьютерную индустрию и на нашу жизнь вообще?

Пожалуй главное, чего не хватает, так это восторженности по отношению к Java. Появление нового универсального языка для Internet — гигантский прорыв вперед. Открывающиеся возможности можно сравнить только с появлением персональных компьютеров. Именно поэтому мы решили уйти из Sun для того, чтобы создать собственную фирму и писать программное обеспечение на Java.

Netscape и Microsoft борются за лидерство на рынке Web-навигаторов. Netscape владеет 70% этого рынка, а Microsoft — главный игрок на рынке программного обеспечения для персональных компью-

теров. Совершенно очевидно, что в этой битве для третьей компании места не остается. В то же время Sun разрабатывает HotJava — свой собственный Web-навигатор. Может быть, существует что-то такое, чего ни Netscape, ни Microsoft не видят сегодня или в будущем, что обеспечит Sun победу?

Как бывший руководитель проекта HotJava, я чувствую, что у HotJava есть хорошие шансы стать значительной силой в Internet. И Netscape Navigator, и Internet Explorer становятся очень сложными, они более не соответствуют простым запросам средних пользователей. С навигатором HotJava Sun возвращает ситуацию на землю, предлагая продукт, полностью написанный на Java. Именно благодаря простоте и возможности для расширения, основанным на использовании Java, он приобретет значительные преимущества по сравнению с конкурентами.

Что станет следующей ДЕЙТЕЛЬНО ВЕЛИКОЙ ВЕЩЬЮ в информационных технологиях после Internet, Web и Java?

Пока Internet — это устройство для распространения информации. Следующая революция произойдет тогда, когда в Internet войдут технологии для коммуникаций в реальном времени, такие как телефония, видео и виртуальная реальность. Развитие идет очень быстро и ждать придется не так уж долго!

Ваш любимый сорт кофе?

Уж конечно не OLE!

INFO

Информационный бюллетень Jet Info

Индекс по каталогу РОСПЕЧАТИ - 32555

Главный редактор: В.А.Галатенко
Технический редактор: С.И.Демочкин

Полное или частичное воспроизведение материалов, содержащихся в настоящем издании, допускается только с разрешения Jet Infosystems

Jet Infosystems

Россия, 103006, Москва,
ул.Краснопролетарская, 6
тел. (095) 972 11 82
(095) 972 13 32
факс (095) 972 07 91

e-mail: JetInfo@jet.msk.su

Jet INFO

Перед Вами один из номеров бесплатного периодического информационно-технического бюллетеня "Jet Info", издаваемого компанией "Инфосистемы Джет" - ведущим российским системным интегратором в области UNIX-систем.

Если Вы еще не являетесь подписчиком "Jet Info", но хотели бы получать его регулярно, просим Вас отправить в наш адрес заполненный купон этого объявления. Отправьте этот купон и в том случае, если Вы сменили свой почтовый адрес или решили изменить форму доставки Вам бюллетеня.

Информация для подписчиков: не забудьте в течение каждого декабря и июля обязательно отправлять нам этот отрывной купон для перерегистрации.



1. Организация _____

2. Фамилия, имя, отчество и должность лица, в чей адрес будет осуществляться рассылка _____

3. Индекс и почтовый адрес _____

4. Телефон и/или факс _____

5. Адрес электронной почты _____

6. В какой форме Вы хотели бы получать бюллетень

печатный вариант по почте

электронную версию по электронной почте

7. Прежняя форма и адрес подписки (для подписчиков, меняющих адрес и форму подписки, либо тех, кто проходит перерегистрацию) _____

8. Общее число компьютеров в организации _____

9. Ваша организация использует

Вычислительную технику

- Apple
- DEC Alpha
- DEC VAX
- IBM PC
- HP 9000
- Silicon Graphics
- Sun Microsystems
- другое _____

Операционные системы

- MS-DOS (Windows)
- Novell NetWare
- SCO Unix
- Solaris / SunOS
- UnixWare
- VAX VMS
- другое _____

10. Материалы на какую тему заинтересовали бы Вас в первую очередь _____

Мы будем рады, если кто-то из Вас захочет принять участие в подготовке выпусков "Jet Info" в качестве автора. Будем также признательны за любые конструктивные замечания и предложения по всем аспектам подготовки, выпуска и распространения нашего бюллетеня.

Редакция "Jet Info"