

РЕКОМЕНДОВАНО  
Sun Microsystems

В. Монахов

# Язык программирования Java и среда NetBeans

3-е издание

Основные концепции и методы объектно-ориентированного программирования

Основные концепции UML-проектирования

Программирование мобильных устройств

Написание серверного программного обеспечения

Численные расчеты с использованием Java

Программы для систем с многоядерными процессорами

Desktop Application — новый вид программ с графическим интерфейсом



bhv

+ dvd

ДИСТРИБУТИВЫ  
JDK 6 и NetBeans 6.9  
для Windows и Linux

**В. Монахов**

**Язык  
программирования  
Java и среда  
NetBeans**

**3-е издание**

Санкт-Петербург

«БХВ-Петербург»

2011

УДК 681.3.068  
ББК 32.973.26-018.1  
М77

**Монахов В. В.**

М77 Язык программирования Java и среда NetBeans. — 3-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2011. — 704 с.: ил. + DVD

ISBN 978-5-9775-0671-7

Книга написана на базе курса лекций, читаемых автором на кафедре вычислительной физики Санкт-Петербургского государственного университета. Изложены основные синтаксические конструкции Java, принципы объектно-ориентированного программирования, особенности проведения численных расчетов. Приводятся сведения о среде NetBeans, предназначенной для профессиональной разработки всех видов программного обеспечения Java (ME, SE и EE). Рассказывается о создании сетевых приложений и приложений для мобильных устройств. Разбираются методики написания многопоточных приложений Java для систем с многоядерными процессорами. Материал сопровождается большим количеством примеров. В третьем издании внесены исправления и на DVD добавлены новые примеры и дистрибутив NetBeans 6.9.1. Также на DVD расположены дистрибутивы JDK 6u22 и NetBeans 6.5 для Windows и Linux.

*Для начинающих Java-программистов и студентов*

УДК 681.3.068  
ББК 32.973.26-018.1

**Группа подготовки издания:**

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Игорь Шишигин</i>
Зав. редакцией	<i>Григорий Добин</i>
Редактор	<i>Леонид Кочин</i>
Компьютерная верстка	<i>Ольги Сергиенко</i>
Корректор	<i>Зинаида Дмитриева</i>
Оформление обложки	<i>Елены Беляевой</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.12.10.  
Формат 70×100<sup>1/16</sup>. Печать офсетная. Усл. печ. л. 56,76.  
Тираж 1500 экз. Заказ №  
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию  
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой  
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов  
в ГУП "Типография "Наука"  
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0671-7

© Монахов В. В., 2011  
© Оформление, издательство "БХВ-Петербург", 2011

# Оглавление

<b>Предисловие к третьему изданию .....</b>	<b>11</b>
<b>Введение.....</b>	<b>13</b>
<b>Глава 1. Общие представления о языке Java.....</b>	<b>17</b>
1.1. Java и другие языки программирования. Системное и прикладное программирование.....	17
1.2. Виртуальная Java-машина, байт-код, JIT-компиляция. Категории программ, написанных на языке Java.....	22
1.3. Алфавит языка Java. Десятичные и шестнадцатеричные цифры и целые числа. Зарезервированные слова.....	28
Алфавит языка Java .....	29
Десятичные и шестнадцатеричные цифры и целые числа .....	29
Зарезервированные слова и литералы языка Java.....	31
1.4. Управляющие последовательности. Символы Unicode. Специальные символы .....	32
Управляющие последовательности.....	32
Управляющие последовательности — символы формирования текста.....	32
Управляющие последовательности — символы Unicode.....	32
Простые специальные символы.....	33
Составные специальные символы .....	34
1.5. Идентификаторы. Переменные и типы. Примитивные и ссылочные типы .....	35
1.6. Процедурное и объектно-ориентированное программирование. Инкапсуляция .....	38
1.7. Работа со ссылочными переменными. Сборка мусора.....	43
1.8. Проекты. Пакеты. Уровни видимости классов. Импорт классов .....	47
1.9. Базовые пакеты и классы Java .....	51
1.10. Технологии Java, .NET, ASP, PHP.....	55
1.11. Среды разработки NetBeans, Eclipse, JDeveloper, JBuilder, IntelliJ IDEA .....	59
NetBeans 6.5.....	61
Eclipse 3.4.1 (Ganymede) .....	63
JDeveloper 11 .....	65
JBuilder 2008 .....	67
IntelliJ IDEA 7.0.4 .....	69
Общий итог сравнения сред разработки.....	70
Краткие итоги по главе.....	70

Типичные ошибки.....	73
Задания.....	74
<b>Глава 2. Среда NetBeans 6.5 .....</b>	<b>75</b>
2.1. Установка JDK 6.....	75
2.2. Установка среды NetBeans 6.5.....	79
2.3. Создание в NetBeans простейшего приложения Java .....	87
2.4. Компиляция файлов проекта и запуск приложения. Jar-файлы.....	92
2.5. Структура проекта NetBeans .....	96
2.6. Создание в NetBeans приложения Java с графическим интерфейсом .....	100
2.7. Редактор экранных форм .....	105
2.8. Свойства компонентов .....	110
2.9. Внешний вид приложения — технология Look and Feel.....	116
2.10. Ведение проектов.....	119
2.11. Редактирование меню экранной формы .....	121
2.12. Создание нового класса.....	129
2.13. Документирование исходного кода в Java.....	130
2.14. Основные компоненты пакетов <i>swing</i> и <i>awt</i> .....	139
2.15. Менеджеры размещения ( <i>Layout</i> ) и якоря ( <i>Anchor</i> ).....	144
2.16. Создание приложения Desktop Application.....	145
Краткие итоги по главе.....	147
Типичные ошибки.....	148
Задания.....	148
<b>Глава 3. Примитивные типы данных и операторы для работы с ними.....</b>	<b>150</b>
3.1. Булевый (логический) тип.....	150
3.2. Целые типы, переменные, константы .....	151
3.3. Основные операторы для работы с целочисленными величинами .....	155
3.4. Вещественные типы и класс <i>Math</i> .....	156
3.5. Правила явного и автоматического преобразования типа при работе с числовыми величинами .....	160
3.6. Оболочечные классы. Упаковка ( <i>boxing</i> ) и распаковка ( <i>unboxing</i> ) .....	162
3.7. Приоритет операторов.....	164
Краткие итоги по главе.....	165
Типичные ошибки.....	167
Задания.....	167
<b>Глава 4. Работа с числами в языке Java.....</b>	<b>169</b>
4.1. Двоичное представление целых чисел.....	169
Позиционные и непозиционные системы счисления .....	169
Двоичное представление положительных целых чисел.....	171
Двоичное представление отрицательных целых чисел. Дополнительный код .....	172
Сложение положительного и отрицательного чисел.....	173
Сложение отрицательных чисел.....	174
Проблемы целочисленной машинной арифметики. Целочисленное переполнение.....	174
Сложение положительных чисел.....	174
Умножение положительных чисел.....	175
Шестнадцатеричное представление целых чисел и перевод из одной системы счисления в другую .....	177
Преобразование чисел из системы с меньшим основанием в систему с большим основанием .....	178

Преобразование чисел из системы с большим основанием в систему с меньшим основанием.....	178
Преобразование чисел в системах счисления с кратными основаниями.....	179
4.2. Побитовые маски и сдвиги.....	180
4.3. Двоичное представление вещественных чисел.....	183
Двоичные дроби.....	183
Мантисса и порядок числа.....	183
Научная нотация записи вещественных чисел.....	185
Стандарт IEEE 754 представления чисел в формате с плавающей точкой.....	186
Краткие итоги по главе.....	190
Типичные ошибки.....	191
Задания.....	191
<b>Глава 5. Управляющие конструкции .....</b>	<b>193</b>
5.1. Составной оператор.....	193
5.2. Условный оператор <i>if</i> .....	194
5.3. Оператор выбора <i>switch</i> .....	198
5.4. Условное выражение <i>... ?... :</i> .....	199
5.5. Операторы инкремента <i>++</i> и декремента <i>--</i> .....	200
5.6. Оператор цикла <i>for</i> .....	200
5.7. Ошибки при использовании вещественного счетчика цикла.....	202
5.8. Эффективная организация циклов при вычислениях в формате с плавающей точкой.....	205
5.9. Особенности целочисленных вычислений — организация циклов, приоритет операторов и арифметическое переполнение.....	210
5.10. Оператор цикла <i>while</i> — цикл с предусловием.....	215
5.11. Оператор цикла <i>do...while</i> — цикл с постусловием.....	217
5.12. Операторы прерывания <i>continue</i> , <i>break</i> , <i>return</i> , <i>System.exit</i> .....	217
Краткие итоги по главе.....	221
Типичные ошибки.....	222
Задания.....	222
<b>Глава 6. Начальные сведения об объектном программировании .....</b>	<b>223</b>
6.1. Наследование и полиморфизм.....	223
6.2. Функции. Модификаторы. Передача примитивных типов в функции.....	231
6.3. Локальные и глобальные переменные. Модификаторы доступа и правила видимости. Ссылка <i>this</i> .....	235
6.4. Передача ссылочных типов в функции. Проблема изменения ссылки внутри подпрограммы.....	238
6.5. Наследование. Суперклассы и подклассы. Переопределение методов.....	245
6.6. Наследование и правила видимости. Резервированное слово <i>super</i> .....	250
6.7. Статическое и динамическое связывание методов. Полиморфизм.....	252
6.8. Базовый класс <i>Object</i> .....	254
6.9. Конструкторы. Резервированные слова <i>super</i> и <i>this</i> . Блоки инициализации.....	256
6.10. Удаление неиспользуемых объектов и метод <i>finalize</i> . Проблема деструкторов для сложно устроенных объектов.....	260
6.11. Перегрузка методов.....	261
6.12. Правила совместимости ссылочных типов как основа использования полиморфного кода. Приведение и проверка типов.....	263

Краткие итоги по главе.....	267
Типичные ошибки.....	268
Задания.....	269

## **Глава 7. UML-диаграммы. Прямое и обратное проектирование.**

<b>Рефакторинг.....</b>	<b>271</b>
7.1. UML-диаграммы.....	271
7.2. Структура окон проектов с UML-диаграммами.....	279
7.3. Панель инструментов проектов с UML-диаграммами.....	282
7.4. Прямое проектирование (Forward engineering) — построение кода классов по UML-диаграммам.....	286
7.5. Обратное проектирование (Reverse engineering) — построение UML-диаграмм по разработанным классам.....	293
7.6. Рефакторинг.....	298
Первый случай — переименование элементов программы.....	301
Второй случай — перемещение класса.....	303
Третий случай — копирование класса для его дальнейшей модификации.....	303
Четвертый случай — безопасное удаление элемента программы.....	304
Пятый случай — изменение списка параметров метода.....	304
Шестой случай — инкапсуляция полей данных.....	305
Краткие итоги по главе.....	306
Типичные ошибки.....	306
Задания.....	307

## **Глава 8. Важнейшие объектные типы..... 308**

8.1. Массивы.....	308
8.2. Коллекции, списки, итераторы.....	314
8.3. Перебор в цикле элементов коллекций. Оператор цикла <i>for-each</i> .....	318
8.4. Работа со строками в Java. Строки как объекты. Классы <i>String</i> , <i>StringBuffer</i> и <i>StringBuilder</i> .....	319
8.5. Типы-перечисления ( <i>enum</i> ).....	333
8.6. Работа с датами и временем. Класс <i>GregorianCalendar</i> .....	335
8.7. Работа с графикой и графическим пользовательским интерфейсом.....	338
Код визуализации и код бизнес-логики приложения.....	338
Графические примитивы.....	339
Пример метода, работающего с графикой.....	341
8.8. Исключительные ситуации.....	345
Обработка исключительных ситуаций.....	345
Иерархия исключительных ситуаций.....	348
Объявление типа исключительной ситуации и оператор <i>throw</i> .....	351
Объявление метода, который может возбуждать исключительную ситуацию. Резервированное слово <i>throws</i> .....	352
8.9. Работа с файлами и папками.....	355
Работа с файлами и папками с помощью объектов типа <i>File</i> .....	356
Выбор файлов и папок с помощью файлового диалога.....	359
Работа с потоками ввода/вывода.....	364
Краткие итоги по главе.....	372
Типичные ошибки.....	373
Задания.....	374

<b>Глава 9. Интерфейсы и композиция .....</b>	<b>375</b>
9.1. Проблемы множественного наследования классов. Интерфейсы .....	375
9.2. Отличия интерфейсов от классов. Проблемы наследования интерфейсов .....	378
9.3. Пример работы с интерфейсами .....	380
9.4. Композиция — еще одна альтернатива множественному наследованию .....	383
Краткие итоги по главе .....	385
Типичные ошибки .....	385
Задания .....	385
<b>Глава 10. Многопоточное программирование и многоядерные системы .....</b>	<b>387</b>
10.1. Потоки выполнения (threads) и синхронизация .....	387
10.2. Преимущества и проблемы при работе с потоками выполнения .....	388
10.3. Синхронизация по ресурсам и событиям .....	389
Синхронизация по ресурсам .....	389
Синхронизация по событиям .....	391
10.4. Класс <i>Thread</i> и интерфейсы <i>Runnable</i> и <i>Callable</i> . Создание и запуск потока выполнения .....	392
10.5. Поля и методы, заданные в классе <i>Thread</i> .....	394
10.6. Работа многопоточных приложений в многопроцессорных и многоядерных системах .....	396
Причины перехода к многопроцессорным и многоядерным системам .....	396
Пример многопоточной программы .....	398
Работа многопоточного приложения на многоядерном компьютере .....	406
10.7. Синхронизация на основе интерфейсов <i>Lock</i> и <i>Condition</i> .....	408
Интерфейс <i>Lock</i> .....	408
Интерфейс <i>Condition</i> .....	412
Краткие итоги по главе .....	414
Типичные ошибки .....	415
Задания .....	416
<b>Глава 11. Введение в сетевое программирование .....</b>	<b>417</b>
11.1. Краткая справка по языку HTML .....	417
Система WWW и язык HTML .....	417
Теги и их атрибуты .....	419
Математические и специальные символы, греческие буквы .....	425
11.2. Апплеты .....	427
Структура апплета .....	427
Примеры апплетов. Аннотация <i>@Override</i> .....	429
Создание проекта с апплетами .....	434
Редактор HTML-документов, стилей CSS и кода JavaScript .....	437
11.3. Сервлеты .....	442
Создание сервлета и конфигурирование сервера приложений .....	444
Локализация сервлетов — работа с языковыми форматами .....	451
11.4. Технология JSP — Java Server Pages .....	456
11.5. Технология JSTL — JSP Standard Tag Library .....	460
Краткие итоги по главе .....	462
Типичные ошибки .....	463
Задания .....	464



<b>Глава 12. Вложенные классы .....</b>	<b>465</b>
12.1. Виды вложенных классов.....	465
12.2. Статические (static) вложенные классы и интерфейсы.....	466
12.3. Внутренние (inner) классы .....	468
12.4. Локальные (local) классы .....	471
12.5. Анонимные (anonymous) классы и обработчики событий .....	471
12.6. Анонимные (anonymous) классы и слушатели событий (listeners) .....	472
Краткие итоги по главе.....	475
Типичные ошибки.....	477
Задания.....	477
<b>Глава 13. Приложение с графическим интерфейсом — <i>DesktopApplication</i> ....</b>	<b>478</b>
13.1. Структура простой заготовки <i>DesktopApplication</i> .....	478
13.2. Исходный код класса <i>DesktopApplication</i> .....	480
13.3. Исходный код класса <i>DesktopApplicationView</i> .....	482
13.4. Конструктор главной формы приложения. Ресурсы приложения и концепция управляющих синглтонов.....	486
13.5. Аннотация <i>@Action</i> и показ справки.....	491
13.6. Концепция управляющего объекта-модели. Модель <i>TaskMonitor</i> , класс <i>Task</i> и обработчик <i>propertyChange</i> .....	493
13.7. Создание задачи типа <i>Task</i> . Назначение иконок и добавление изображений.....	500
13.8. Локализация приложения.....	505
13.9. Измерение времени и досрочное прекращение выполнения задания .....	510
13.10. Работа с межпрограммным буфером обмена — <i>Clipboard</i> .....	511
Копирование строк и изображений из буфера обмена в программу.....	511
Копирование строк из программы в буфер обмена. Класс <i>StringSelection</i> .....	514
Копирование изображений из программы в буфер обмена. Создание класса <i>ImageSelection</i> .....	515
13.11. Многопоточная система вычислений и индикации .....	519
Краткие итоги по главе.....	523
Типичные ошибки.....	524
Задания.....	525
<b>Глава 14. Программирование мобильных телефонов .....</b>	<b>526</b>
14.1. Спецификация Java Micro Edition. Конфигурации и профили. Мидлеты .....	526
14.2. Создание приложений для мобильных устройств.....	527
14.3. Дизайнер пользовательского интерфейса мидлета.....	533
14.4. Исходный код мидлета <i>Hello, World!</i> .....	538
14.5. Заготовка мидлета "Крестики-нолики" .....	547
14.6. Код бизнес-логики мидлета .....	556
14.7. Создание собственного компонента, являющегося наследником <i>TableItem</i> , и мидлета на его основе. Создание библиотеки .....	563
14.8. Запуск мидлета "Крестики-нолики" на реальном телефоне.....	571
14.9. Пример мидлета с переключением экрана и списком выбора.....	573
14.10. Примеры приложений для мобильных телефонов, поставляемые со средой NetBeans.....	574
Краткие итоги по главе.....	577
Типичные ошибки.....	578
Задания.....	578

<b>Глава 15. Компонентное программирование .....</b>	<b>579</b>
15.1. Компонентная архитектура JavaBeans .....	579
15.2. Создание компонента в NetBeans 6.5 .....	580
15.3. Пример создания компонента в NetBeans 6.5 — панель с заголовком .....	584
15.4. Добавление в компонент новых свойств .....	588
15.5. Добавление в компонент новых событий .....	590
Краткие итоги по главе.....	593
Типичные ошибки.....	594
Задания.....	595
<b>Глава 16. Платформо-зависимые методы и модуль C/C++ Pack .....</b>	<b>596</b>
16.1. Использование динамических библиотек. Объявление платформо-зависимых (native) методов.....	596
16.2. Интерфейс JNI (Java Native Interface) взаимодействия Java с C/C++. Соответствие типов Java и C++ .....	597
16.3. Пакет C/C++ Pack. Подключение компиляторов C++, C и библиотек.....	603
16.4. Создание приложений C++ и C.....	612
16.5. Примеры приложений с native-методами .....	616
Первый этап. Создание исходного кода Java для native-методов.....	617
Второй этап. Генерация h-файла заголовка C++, содержащего декларацию методов....	617
Третий этап. Написание на C++ реализации методов, заголовки которых заданы в сгенерированном файле jnipackage_JniMultiply.h .....	619
Четвертый этап. Компиляция кода C++ в динамическую библиотеку (DLL) .....	620
Пятый этап. Запуск приложения вместе с DLL в NetBeans .....	621
Шестой этап. Распространение приложения вместе с DLL .....	621
Краткие итоги по главе.....	625
Типичные ошибки.....	626
Задания.....	626
<b>Глава 17. Отладка, тестирование и профилирование программ .....</b>	<b>627</b>
17.1. Отладка приложений в среде NetBeans.....	627
17.2. Команды режима пошагового выполнения .....	631
17.3. Пример отладки неправильно работающего приложения.....	638
17.4. Принцип "презумпции виновности" и тестирование классов .....	642
17.5. Профилирование приложений .....	656
17.6. Профилирование отдельных методов .....	664
17.7. Настройка профилировщика "на лету". Точки профилирования .....	670
17.8. Нахождение проблем с использованием памяти — телеметрия.....	678
17.9. Профилирование использования памяти классами.....	681
Краткие итоги по главе.....	687
Типичные ошибки.....	688
Задания.....	688
<b>Заключение .....</b>	<b>689</b>
<b>Приложение. Описание прилагаемого DVD.....</b>	<b>691</b>
<b>Литература .....</b>	<b>698</b>
Основная.....	698
Дополнительная .....	698
<b>Предметный указатель .....</b>	<b>699</b>



# Предисловие к третьему изданию

Год, прошедший со времени выхода второго издания, оказался богат на события в мире Java и NetBeans. Во-первых, состоялся выход большого количества обновлений дистрибутивов Java. Во-вторых, появились версии среды NetBeans 6.7, 6.8 и 6.9. В-третьих, вышел дистрибутив Open JDK (см. <http://www.openjdk.org/>), позволивший гарантировать дальнейшее существование платформы Java и основанных на ней технологий независимо от судьбы корпорации Sun Microsystems. И, кроме того, корпорация Oracle, один из мировых лидеров в области индустрии программного обеспечения, приобрела фирму Sun Microsystems.

Преыдущее издание заняло достойное место среди компьютерной литературы по программированию на языке Java для начинающих. Ведь другие учебники на русском языке, в которых изложение материала основано на использовании современных сред разработки, просто-напросто отсутствуют, а учиться современному программированию с помощью морально устаревших технологий — не самый лучший выбор. Третье издание является исправленной версией предыдущего, с коррекцией найденных опечаток и мелких ошибок, а также с новыми дистрибутивами Java и NetBeans.

К книге приложен DVD с материалами, соответствующими ее тексту (примерами, а также дистрибутивами Java JDK 6u11 и NetBeans 6.5), для того чтобы желающие могли проделать все ровно так, как описано в тексте глав. Тем не менее, на DVD имеются и более современные дистрибутивы: текущие на момент верстки третьего издания версии JDK 6u22 и NetBeans 6.9.1 в вариантах для Windows и Linux. Автор рекомендует в любом случае устанавливать более новую версию JDK 6u22. Примеры из книги работают как в NetBeans 6.5, так и в NetBeans 6.9.1, при этом NetBeans 6.5 функционирует быстрее и занимает меньше дискового пространства, зато NetBeans 6.9.1 имеет гораздо более развитую среду редактирования и отладки. При необходимости можно установить на компьютере обе версии среды, т. к. они не мешают друг другу (хотя, конечно, это займет дополнительное дисковое пространство).

Замечания по найденным ошибкам и опечаткам просьба присылать автору по электронной почте на адрес [v.v.monahov@mail.ru](mailto:v.v.monahov@mail.ru). Автор выражает благодарность всем, кто внес вклад в исправление ошибок предыдущих изданий (см. [http://barsic.spbu.ru/www/comlan/java\\_vm.html](http://barsic.spbu.ru/www/comlan/java_vm.html)).

*Доцент кафедры вычислительной физики СПбГУ,  
кандидат физико-математических наук Вадим Валериевич Монахов*



# Введение

Предлагаемый учебный курс можно рассматривать как краткое введение в современное программирование на Java и обзор возможностей NetBeans — одной из самых популярных и быстро развивающихся сред разработки программного обеспечения для языков Java, C, C++, Fortran, PHP, JavaScript, Ruby, Groovy и многих других.

О среде NetBeans в русскоязычной литературе рассказывается только в данной книге автора, а в иностранной — имеется около десятка таких книг, причем лишь в двух-трех из них излагается курс программирования на Java с использованием NetBeans. Поэтому читателю, желающему научиться программированию на языке Java, данное учебное пособие предоставляет уникальную возможность осуществить это с помощью наиболее современных средств, заметно облегчающих работу программиста и делающих ее гораздо более эффективной по сравнению с методиками, описанными в других книгах.

Первое издание книги, вышедшее в марте 2008 года, было полностью распродано к декабрю того же года. Оно оказалось очень популярным как среди начинающих программистов, так и среди профессионалов, переходящих с C или C++ на Java. Во втором издании была изложена работа с NetBeans версии 6.5 (первое издание основывалось на версии 6.0), а также исправлены допущенные ранее ошибки и опечатки. Кроме того, был добавлен раздел, в котором NetBeans сравнивается с другими средами разработки Java-программ, заметно расширено описание работы со строками и дописаны разделы про современные средства Java для работы с датой и временем, а также про применение в NetBeans тестовых пакетов и библиотеки JUnit 4.

В настоящем издании сохранено изложение, основанное на использовании NetBeans 6.5, однако все сказанное практически без изменений относится и к NetBeans 6.9. А различие в версии JDK вообще несущественно, и вместо JDK 6u11 рекомендуется использовать JDK 6u22, т. к. этот комплект содержит обновления системы безопасности и исправления ошибок.

Несмотря на то, что языку Java уже посвящены тысячи книг и учебных курсов, все книги, написанные до выхода JDK 5 (прежнее обозначение JDK 1.5), не соответствуют современным представлениям об этом языке.

Данное учебное пособие содержит информацию об основных синтаксических конструкциях современной версии языка Java (JDK 6), типичных ошибках их использования, рекомендации по написанию на языке Java эффективно работающих программ.

Одной из привлекательных особенностей языка Java с самого начала была бесплатность распространения базовых средств разработки SDK (Software Development Kit) и исполняющей среды Java (виртуальной Java-машины). Однако компилятор, входящий

в состав SDK, работал в режиме командной строки, т. е. отставал идеологически, по крайней мере, лет на 20 от современных профессиональных компиляторов. В 2006 году корпорация Sun Microsystems пошла на беспрецедентный шаг — сделала бесплатными свои профессиональные средства разработки программного обеспечения. В настоящее время все они основаны на бесплатно распространяемой среде разработки NetBeans (платформе NetBeans), подключаемых к ней модулях (packs) и наборе библиотечных компонентов (Beans — зерен). Интересна игра слов: язык Java получил наименование по названию кофе, который любили программисты, так что Java Beans может быть расшифровано как "зерна кофе Java".

Среда NetBeans дала возможность разрабатывать программное обеспечение на языке Java с помощью самых современных методик. Она имеет развитые средства визуального проектирования пользовательского интерфейса, не уступающие Delphi, развитый редактор исходного кода программы с подсветкой ошибок непосредственно во время редактирования, а также множество других возможностей, о которых рассказывается в данной книге.

В NetBeans начиная с версии 6.5 не только существенно усовершенствован редактор исходного кода и отладчик, но и изменен сам подход к развитию среды. Если предыдущие версии ориентировались только на язык Java и в небольшой степени на C/C++, то в NetBeans 6.5 произошел переход к поддержке широкого спектра языков программирования. И если раньше корпорация Sun Microsystems, разработчик языка Java и значительной части среды NetBeans, основное внимание уделяла языку Java, то в последнее время акцент сменился — среда NetBeans и технология Java Scripting (исполнение виртуальной Java-машиной программ, написанных на отличных от Java языках программирования) оказались удобными инструментами для поддержки огромного количества языков программирования и сопутствующих им технологий. При этом связующим звеном с NetBeans и Java-машиной остается язык Java.

В состав дистрибутивов NetBeans 6.5 и 6.9 (имеются на прилагаемом к книге DVD) входят все основные модули, необходимые для программирования на Java. Кроме того, в них входит модуль "C/C++ Pack", позволяющий программировать в среде NetBeans не только на Java, но и на языках C и C++, благодаря подключению к NetBeans внешних компиляторов C/C++. Это дает возможность разрабатывать полный спектр Java-программ в одной и той же среде. Раньше для создания платформо-зависимого (native) кода, используемого native-методами классов Java, требовались сторонние среды разработки программ, написанных на языках C/C++.

Среду разработки NetBeans можно установить с прилагаемого к книге DVD или свободно загрузить с сайта <http://netbeans.org/>. Она распространяется на условиях лицензии, дающей право использовать разработанное с помощью среды NetBeans программное обеспечение не только в некоммерческих, но и в коммерческих целях. Таким образом, теперь как начинающим разработчикам, так и крупным фирмам нет необходимости тратить деньги на покупку профессиональных средств разработки — можно работать в среде NetBeans.

Почему же это не бесплатный сыр? Среда NetBeans разрабатывается сообществом разработки свободного программного обеспечения. Значительный вклад в разработку внесла фирма Sun Microsystems, которая получала деньги не у частных лиц или маленьких компаний, а у крупных корпоративных пользователей за техническую поддержку своих бесплатно распространяемых приложений. И лидерство в данной области косвенно обеспечивало гораздо большие прибыли, чем продажа этих продуктов.

Java очень быстро развивается, особенно в последние годы. Из-за этого многие прежние представления и методы программирования на Java оказываются устаревшими. Среда NetBeans развивается еще быстрее — версия 6.5, появившаяся в 2008 году, заметно превосходит по возможностям версию 6.0, вышедшую в 2007 году. Данная книга — первый шаг к изучению современных технологий на основе Java и NetBeans.

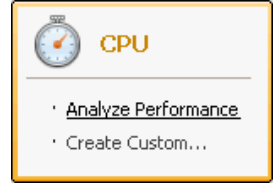
Изложение ведется на основе языка Java для JDK версии 6 и среды NetBeans версии 6.5. Примеры, приведенные в данной книге, проверены в среде NetBeans версий 6.5 и 6.9 с JDK 6u11 и JDK 6u22.

Поскольку технология Java мультиплатформенна, то все изложенное справедливо как для MS Windows, так и для Solaris, Linux, Mac OS и других операционных систем.

Автор выражает благодарность рецензентам фирмы Sun Microsystems Алексею Акоюну и Алексею Уткину за полезные замечания по первоначальному варианту данного учебного пособия, Игорю Богдановичу Керницкому за содействие в подготовке примеров с native-кодом, а также жене Светлане за помощь в оформлении книги.







# ГЛАВА 1

## Общие представления о языке Java

### 1.1. Java и другие языки программирования. Системное и прикладное программирование

Язык программирования Java был создан в рамках проекта корпорации Sun Microsystems по разработке компьютерных программно-аппаратных комплексов нового поколения. Первая версия языка была официально опубликована в 1995 году. С тех пор язык Java стал стандартом "де-факто", вытеснив за десять лет языки C и C++ из многих областей программирования. В 1995 году они были абсолютными лидерами, но к 2006 году число программирующих на Java стало заметно превышать число программистов, использующих C и C++, и сейчас достигло более четырех с половиной миллионов человек. А число устройств, запрограммированных с помощью Java, превысило полтора миллиарда.

Как связаны между собой языки C, C++, JavaScript и Java? Что между ними общего и в чем они отличаются? В каких случаях следует, а в каких не следует их применять? Для того чтобы ответить на эти вопросы, нужно сначала остановиться на особенностях программного обеспечения предыдущих поколений и на современных тенденциях в развитии программного обеспечения.

Первоначально компьютеры программировались в машинных кодах. Затем появились языки ассемблера, которые заменили команды процессоров мнемоническими сокращениями, гораздо более удобными для человека, чем последовательности нулей и единиц. Их принято считать языками программирования *низкого уровня* (т. е. близкими к аппаратному уровню), т. к. они ориентированы на особенности конкретных процессоров. Программы, написанные на языках ассемблера, нельзя было переносить на компьютеры с другим типом процессора из-за несовместимости наборов команд (т. е. несовместимости на уровне исходного кода — source code).

Программы в *машинных кодах* (т. е. в виде последовательности нулей и единиц), соответствующих командам процессора и необходимым для них данным, нет надобности как-то преобразовывать. Их можно просто скопировать в нужное место памяти компьютера и передать управление первой команде программы (задать точку входа в программу).

Программы, написанные на каком-либо языке программирования, сначала нужно перевести из одной формы (текстовой) в другую (двоичную, т. е. в машинные коды). Про-

цесс такого перевода называется *трансляцией* (от англ. *translation* — "перевод", "перемещение"). Не обязательно переводить программу из текстовой формы в двоичные коды, возможен процесс трансляции с одного языка программирования на другой (или из кодов одного типа процессора в коды другого типа).

Имеются два основных вида трансляции — *компиляция* и *интерпретация*.

При *компиляции* первоначальный набор инструкций однократно переводится в исполняемую форму (машинные коды), а в последующем при работе программы используются только эти коды.

При *интерпретации* во время вызова необходимых инструкций каждый раз сначала происходит перевод инструкций из одной формы (текстовой или двоичной) в другую — в исполняемые коды процессора используемого компьютера. И только потом эти коды исполняются. Естественно, что интерпретируемые коды исполняются медленнее, чем скомпилированные, т. к. перевод инструкций из одной формы в другую обычно занимает в несколько раз больше времени, чем выполнение полученных инструкций. Но интерпретация обеспечивает бóльшую гибкость по сравнению с компиляцией, и в ряде случаев без нее не обойтись.

В 1956 году появился язык Fortran — первый язык программирования *высокого уровня* (т. е. не ориентированный на конкретную аппаратную реализацию компьютера). Он обеспечил переносимость программ на уровне исходных кодов, но довольно дорогой ценой. Во-первых, быстродействие программ, написанных на этом языке, было в несколько раз меньше, чем для ассемблерных. Во-вторых, такие программы занимали примерно в два раза больше места в памяти компьютера, чем ассемблерные. И, наконец, пришлось отказаться от поддержки особенностей периферийных устройств — ограничить общение с "внешним миром" простейшими возможностями, которые в программе одинаково реализовывались для ввода/вывода с помощью перфокарточного считывателя, клавиатуры, принтера, текстового дисплея и т. д. Тем не менее, языки программирования высокого уровня постепенно вытеснили ассемблер, поскольку обеспечивали не только переносимость программ, но и гораздо бóльшую их надежность, а также несоизмеримо более высокую скорость разработки сложного программного обеспечения. А Fortran до сих пор остается важнейшим языком программирования для высокопроизводительных численных научных расчетов.

Увеличение аппаратных возможностей компьютеров (рост объема оперативной и дисковой памяти, значительное повышение быстродействия), а также появление разнообразных периферийных устройств привело к необходимости пересмотра того, как должны работать программы. Массовый выпуск компьютеров потребовал унификации доступа из программ к различным устройствам, обеспечивая игнорирование особенностей их аппаратной реализации. Это можно осуществить, если обращение к устройству идет не напрямую, а через прилегающую программу — драйвер устройства (по-английски *driver* означает "водитель").

Появились операционные системы — наборы драйверов и специальных программ, распределяющих ресурсы компьютера между разными программами. Соответственно, программное обеспечение стало разделяться на *системное* и *прикладное*. Системное программное обеспечение — непосредственно обращающееся к аппаратуре, прикладное — решающее какие-либо прикладные задачи и использующее аппаратные возможности компьютера не напрямую, а через вызовы программ операционной системы.

Прикладные программы стали приложениями операционной системы (сокращенно *приложениями* — applications). Этот термин означает, что программа может работать только под управлением операционной системы. Если на том же компьютере установить другой тип операционной системы, то программа-приложение первой операционной системы, возможно, не будет работать.

Требования к прикладным и системным программам принципиально различаются. От системного программного обеспечения требуется максимальное быстродействие и минимальное количество занимаемых ресурсов, а также возможность доступа к любым необходимым аппаратным ресурсам. От прикладного — максимальная функциональность в конкретной предметной области. При этом быстродействие и занимаемые ресурсы не имеют значения до тех пор, пока не влияют на функциональность. Например, нет никакой разницы, реагирует ли программа на нажатие клавиши за одну десятую или за одну миллионную долю секунды. Правда, на первоначальном этапе создания прикладного программного обеспечения даже прикладные по назначению программы являлись системными по реализации, т. к. были вынуждены напрямую обращаться к аппаратуре.

Язык C был создан в 1972 году в одной из исследовательских групп Bell Laboratories при разработке операционной системы UNIX. Сначала была предпринята попытка написать операционную систему на ассемблере, но после появления в группе новых компьютеров возникла необходимость в новом платформи-независимом языке программирования высокого уровня, с помощью которого можно было бы писать операционные системы. Таким образом, язык C предназначался для создания системного программного обеспечения, и таким он остается до сих пор. Его идеология и синтаксические конструкции ориентированы на максимальную близость к аппаратному уровню реализации операций — в той степени, в какой он может быть обеспечен на аппаратно-независимом уровне. При этом главным требованием была максимальная скорость работы и минимальное количество занимаемых ресурсов, а также возможность доступа ко всем аппаратным средствам. Язык C — это язык *процедурного программирования*, т. к. его базовыми конструкциями являются *подпрограммы*. В общем случае подпрограммы принято называть подпрограммами-процедурами (откуда и идет название "процедурное программирование") и подпрограммами-функциями. Но в C имеются только подпрограммы-функции. Обычно их называют просто *функциями*.

Язык C произвел настоящую революцию в разработке программного обеспечения, получил широкое распространение и стал промышленным стандартом. Он до сих пор применяется для написания операционных систем и программирования микроконтроллеров. Но мало кто в полной мере осознает причины его популярности. В чем они заключались? В том, что он смог обеспечить необходимую функциональность программного обеспечения в условиях низкой производительности компьютеров, крайней ограниченности их ресурсов и неразвитости периферийных устройств! При этом повторилась та же история, что и с языком Fortran, но теперь уже для языка системного программирования. Переход на язык программирования высокого уровня, но с минимальными потерями по производительности и ресурсам, дал большие преимущества.

Большое влияние на развитие теории программирования оказал язык Pascal, разработанный в 1974 году швейцарским профессором Никлаусом Виртом. В этом проекте сочетались два оригинальных замысла. Первый — собственно язык Pascal, предназначен

ный для обучения идеям *структурного программирования*. Второй — идея *виртуальной машины*. Никлаус Вирт предложил обеспечивать переносимость программ, написанных на Pascal, за счет компиляции их в набор команд некоей абстрактной Р-машины (Р — сокращение от Pascal), а не в исполняемый код конкретной аппаратной платформы. А на каждой аппаратной платформе должна была работать программа, интерпретирующая эти коды. Говорят, что такая программа эмулирует (т. е. имитирует) систему команд несуществующего процессора. А саму программу называют виртуальной машиной.

В связи с ограниченностью ресурсов компьютеров и отсутствием в Pascal средств системного программирования этот язык не смог составить конкуренцию языку С, т. к. практически все промышленное программирование вплоть до середины последнего десятилетия XX века по реализации было системным. Идеи Р-машины были в дальнейшем использованы и значительно усовершенствованы в Java.

Развитие теории и практики программирования привело к становлению в 1967—1972 годах нового направления — *объектного программирования*, основанного на концепциях работы с *классами* и *объектами*. Оно обеспечило принципиально новые возможности по сравнению с процедурным. Были предприняты попытки расширения различных языков путем введения в них конструкций объектного программирования.

В 1982 году Бьерн Страуструп путем такого расширения языка С создал язык, который он назвал "С с классами". В 1983 году после очередных усовершенствований им был создан первый компилятор языка С++. Два плюса означают: "язык С с очень большим количеством добавлений" и одновременно — "новое, увеличенное на один по сравнению с прежним, значение" (игра понятий: в С существует оператор "+").

Язык С++ является надмножеством над языком С — на нем можно писать программы как на "чистом С", без использования каких-либо конструкций объектного программирования. В связи с этим, а также и с дополнительными преимуществами объектного программирования, язык С++ быстро приобрел популярность и стал промышленным стандартом (сначала "де-факто", а потом и "де-юре"). Так что в настоящее время С++ признан базовым языком системного программирования. Длительное время он использовался и для написания прикладных программ. Но, как мы уже знаем, требования к прикладным программам совпадают с требованиями к системным только в том случае, когда быстродействие компьютера можно рассматривать как низкое, а ресурсы компьютера — малыми. Кроме этого, у языков С и С++ имеются еще два принципиальных недостатка: а) низкая надежность на уровне как исходного, так и исполняемого кода; б) отсутствие переносимости на уровне исполняемого кода. С появлением компьютерных сетей эти недостатки стали очень существенным ограничивающим фактором, поскольку вопросы безопасности при работе в локальных, а особенно в глобальных сетях приобретают первостепенную значимость.

В 1995 году появились сразу два языка программирования, имеющие в настоящее время огромное значение: Java, разработанный в корпорации Sun, и JavaScript, созданный в небольшой фирме Netscape Communication, получившей к тому времени известность благодаря браузеру Netscape Navigator.

Язык Java создавался как универсальный, предназначенный для прикладного программирования в неоднородных компьютерных сетях как со стороны клиентского компью-

тера, так и со стороны сервера. В том числе для использования на *тонких аппаратных клиентах* (устройствах малой вычислительной мощности с крайне ограниченными ресурсами). Скомпилированные программы Java работают только под управлением виртуальной Java-машины, поэтому их называют *приложениями Java*. Синтаксис операторов Java практически полностью совпадает с синтаксисом языка C, но в отличие от C++, Java это не расширение C, а совершенно независимый язык, со своими собственными синтаксическими правилами. Он гораздо сильнее типизирован по сравнению с C и C++, т. е. вносит больше ограничений на действия с переменными и величинами разных типов. Например, в C/C++ нет разницы между целочисленными числовыми, булевыми и символьными величинами, а также адресами в памяти. Так, можно умножить символ на булево значение `true` (истина), из которого вычтено целое число, и разделить результат на адрес! В Java введен вполне разумный запрет на почти все действия такого рода.

JavaScript создавался как узкоспециализированный прикладной язык программирования HTML-страниц, расширяющий возможности HTML, и он в полной мере отвечает этим потребностям до сих пор. Про язык HTML, являющийся основой для создания WWW-документов, более подробно будет рассказано в *главе 11*.

Следует подчеркнуть, что язык JavaScript *не имеет никакого отношения* к Java. Слово "script" означает сценарий. Включение слова "Java" в название JavaScript — рекламный трюк фирмы Netscape Communication. Язык JavaScript также C-образен, но в отличие от Java, интерпретируемый. Основное его назначение — программное управление элементами WWW-документов. Языки HTML и XML позволяют задавать статический, неизменный внешний вид документов, с их помощью невозможно запрограммировать реакцию на действия пользователя. JavaScript позволяет ввести элементы программирования в поведение документа. Программы, написанные на JavaScript, встраиваются в документы в виде исходных кодов (сценариев) и имеют небольшой размер. Для упрощения работы с динамически формируемыми документами JavaScript имеет свободную типизацию: переменные меняют тип по результату присваивания. Поэтому программы JavaScript гораздо менее надежны, чем C/C++, не говоря уже про Java.

Java, JavaScript и C++ относятся к объектно-ориентированным языкам программирования, все они имеют C-образный синтаксис операторов. Но как объектные модели, так и базовые конструкции этих языков (за исключением синтаксиса операторов) принципиально различны. Ни один из них не является версией или упрощением другого — это совсем разные языки, предназначенные для различных целей.

Итак:

- ◆ Java — универсальный язык прикладного программирования;
- ◆ JavaScript — узкоспециализированный язык программирования HTML-документов;
- ◆ C++ — универсальный язык системного программирования.

В 2000 году в корпорации Microsoft была разработана платформа .NET (читается "дот-нет", DotNet — в переводе с английского "точка NET"). Она стала альтернативой платформе Java и во многом повторила ее идеи. Основное различие заключалось в том, что для этой платформы можно было использовать произвольное количество языков программирования, а не один. Причем классы .NET оказываются совместимыми как в

целях наследования, так и по исполняемому коду независимо от языка, на котором они написаны. Важнейшим языком .NET стал Java-образный язык C# (читается "Си шарп"). Фактически, C# унаследовал от Java большинство особенностей — динамическую объектную модель, сборку "мусора", основные синтаксические конструкции, хотя он вполне самостоятельный язык программирования, имеющий много привлекательных черт. В частности, компонентные модели Java и C# принципиально отличаются.

Java стал первым универсальным C-образным языком прикладного программирования, что обеспечило легкость перехода на этот язык большого числа программистов, знакомых с C и C++. А наличие средств строгой проверки типов, ориентация на работу с компьютерными сетями, переносимость на уровне исполняемого кода и поддержка платформи-независимого графического интерфейса, а также запрет прямого обращения к аппаратуре обеспечили выполнение большинства требований, предъявлявшихся к языку прикладного программирования. Чем больше становятся быстродействие и объем памяти компьютеров, тем больше потребность в разделении прикладного и системного программного обеспечения. Соответственно, для прикладных программ исчезает необходимость напрямую обращаться к памяти и другим аппаратным устройствам компьютера. Поэтому среди прикладных программ с каждым годом растет доля программного обеспечения, написанного на Java и языках .NET. Но как по числу программистов, так и по числу устройств, использующих соответствующие платформы, Java в настоящее время лидирует с большим отрывом.

## **1.2. Виртуальная Java-машина, байт-код, JIT-компиляция. Категории программ, написанных на языке Java**

Первоначально слово "программа" означало последовательность инструкций процессора для решения какой-либо задачи. Эти инструкции представляли собой машинные коды, и разницы между исходным и исполняемым кодом программы не было. Разница появилась, когда программы стали писать на языках программирования. При этом программой стали называть как текст, содержащийся в файле с исходным кодом, так и исполняемый файл.

Для устранения неоднозначности термина "программа" исполняемый код программы принято называть приложением (application). Термин "приложение" — сокращение от фразы "приложение операционной системы". Он означает, что исполняемый код программы может работать только под управлением соответствующей операционной системы. Работа под управлением операционной системы позволяет избежать зависимости программы от устройства конкретного варианта аппаратуры на компьютере, где она должна выполняться. Например, как автору программы, так и пользователю совершенно безразлична аппаратная реализация устройства, с которого считывается информация: будет ли это жесткий диск с одной, двумя или шестнадцатью считывающими головками, или это будет CD-привод, DVD-привод, или еще какой-либо другой тип носителя. Но переносимость обычных приложений ограничивается одним типом операционных систем. Например, приложение MS Windows не будет работать под Linux, и наоборот. Существует возможность исключить зависимость приложения от

операционной системы, введя выполнение его команд не средствами операционной системы, а *исполняющей средой* — программой, контролирующей работу приложения и выполняющей его вызовы. Программы, написанные на языке Java, выполняются исполняющей средой — виртуальной Java-машиной, и поэтому обладают переносимостью на любую операционную систему, где имеется соответствующая Java-машина. Благодаря этому они являются не приложениями какой-либо операционной системы, а *приложениями Java*.

Программы, написанные на языке Java, представляют собой наборы классов (о классах будет рассказано несколько позже) и сохраняются в текстовых файлах с расширением `java`. При компиляции текст программы переводится (транслируется) в двоичные файлы с расширением `class`. Такие файлы содержат байт-код, представляющий собой совокупность инструкций для абстрактного Java-процессора в виде байтовых последовательностей команд этого процессора и данных к ним. Для того чтобы выполнить байт-код на каком-либо компьютере, его нужно перевести в инструкции для соответствующего процессора. Именно этим и занимается Java-машина. Первоначально байт-код всегда интерпретировался следующим образом: каждый раз, когда встречалась какая-либо инструкция Java-процессора, она переводилась в последовательность инструкций процессора компьютера, что, естественно, значительно замедляло работу приложений Java.

В настоящее время используется более сложная схема, называемая JIT-компиляцией (Just-In-Time), — это компиляция "по ходу дела", "на лету". Когда какая-либо инструкция (или набор инструкций) Java-процессора выполняется в первый раз, происходит компиляция соответствующего ей байт-кода с сохранением скомпилированного кода в специальном буфере. При последующем вызове той же инструкции вместо ее интерпретации скомпилированный код извлекается из буфера. Поэтому интерпретация происходит только при первом вызове инструкции.

Сложные оптимизирующие JIT-компиляторы действуют еще изощренней. Поскольку обычно компиляция инструкции идет гораздо дольше по сравнению с интерпретацией, время ее выполнения в первый раз при наличии JIT-компиляции может заметно отличаться в худшую сторону по сравнению с "чистой" интерпретацией. Поэтому бывает выгоднее сначала запустить процесс интерпретации, а параллельно ему в фоновом режиме компилировать инструкцию. Только после окончания процесса компиляции при последующих вызовах инструкции будет исполняться ее скомпилированный код (а до этого все ее вызовы будут интерпретироваться). Разработанная Sun виртуальная машина HotSpot осуществляет JIT-компиляцию только тех участков байт-кода, которые критичны ко времени выполнения программы. При этом по ходу работы программы происходит оптимизация скомпилированного кода.

Благодаря компиляции программ Java в платформу-независимый байт-код обеспечивается переносимость этих программ не только на уровне исходного кода, но и на уровне скомпилированных приложений. Конечно, при этом на компьютере, где выполняется приложение, должна быть установлена программа виртуальной Java-машины (Java Virtual Machine — JVM), скомпилированная в коды соответствующего процессора (native code — платформу-зависимый код). На одном и том же компьютере может быть несколько Java-машин разных версий или от разных производителей. Спецификация Java-машины и требования к компилятору языка Java открытые, поэтому различные фирмы (а не только Sun) разрабатывают компиляторы Java и Java-машины.



Обычно приложение операционной системы запускается с помощью средств самой операционной системы. Приложение Java, напротив, исполняется с помощью виртуальной Java-машины, которая сама является приложением операционной системы. Таким образом, сначала стартует Java-машина. Она получает в качестве параметра имя файла с скомпилированным кодом класса. В этом классе ищется и запускается на выполнение подпрограмма с именем `main`.

Приложения Java обладают не только хорошей переносимостью, но и высокой скоростью работы. Однако даже при наличии JIT-компиляции они все-таки выполняются медленнее, чем программы, написанные на C или C++. Это связано с тем, что JIT-компиляция создает не настолько оптимальный код, как многопроходный компилятор C/C++ (он имеет возможность тратить много времени и ресурсов на поиск конструкций программы, которые можно оптимизировать). А JIT-компиляция происходит "на лету", в условиях жесткой ограниченности времени и ресурсов. Для решения этой проблемы были разработаны компиляторы программ Java в код конкретных программно-аппаратных платформ (`native code` — платформо-зависимый код). Например, свободно распространяемый фондом GNU компилятор GCJ (<http://gcc.gnu.org/java/>). Правда, заметные успехи Sun в усовершенствовании Java-машины позволили практически достичь, а в ряде случаев даже обогнать по быстродействию программы, написанные на других языках, и отказаться от статической компиляции в платформо-зависимый код. В частности, приложения Java, активно занимающиеся выделением/высвобождением памяти, работают быстрее своих аналогов, написанных на C/C++, благодаря специальному механизму программных слотов памяти (`slot` — паз, отверстие для вставки чего-либо).

Виртуальная Java-машина не только исполняет байт-код (интерпретирует его, занимается JIT-компиляцией и исполняет JIT-компилированный код), но и выполняет ряд других функций (например, взаимодействует с операционной системой, обеспечивая доступ к файлам или поддержку графики), а также реализует автоматическое высвобождение памяти, занятой ненужными объектами, — так называемую сборку мусора (`garbage collection`).

Программы Java можно разделить на несколько основных категорий.

- ◆ Приложение (`application`) — аналог обычной прикладной программы.
- ◆ Апплет (`applet`) — специализированная программа с ограниченными возможностями, работающая в окне WWW-документа под управлением браузера.
- ◆ Мидлет (`midlet`) — специализированная программа с ограниченными возможностями, работающая на мобильном устройстве. В настоящее время мидлеты — один из наиболее популярных сегментов рынка программного обеспечения, пишущегося на языке Java.
- ◆ Сервлет (`servlet`) — специализированная программа с ограниченными возможностями, работающая в WWW на стороне сервера. Используется преимущественно в рамках технологии JSP (Java Server Pages — серверных страниц Java) для программирования WWW-документов со стороны сервера и обработки отсылаемой на сервер информации.
- ◆ Серверное приложение (`Enterprise application`) — предназначено для многократного выполнения на стороне сервера.

- ◆ Библиотека (Java Class Library — библиотека классов, либо NetBeans Module — модуль платформы NetBeans) — предназначена для многократного использования программами Java.

Между приложениями и апплетами Java существует принципиальное различие: приложение запускается непосредственно с компьютера пользователя и имеет доступ ко всем ресурсам компьютера наравне с любыми другими программами. Апплет же загружается из WWW с постороннего сервера, причем из-за самой идеологии WWW-сайт, с которого загружен апплет, в общем случае не может быть признан надежным. А сам апплет может передавать данные на произвольный сервер в WWW. Поэтому, для того чтобы избежать риска утечки конфиденциальной информации с компьютера пользователя или совершения враждебных действий, у апплетов убраны многие возможности, имеющиеся у приложений.

Сервлеты — это приложения Java, запускаемые на стороне сервера. Они обладают возможностью доступа к файловой системе и другим ресурсам сервера через набор управляющих конструкций, predetermined в рамках технологии JSP и пакета `javax.servlet`. Кроме того, они имеют доступ к содержимому HTTP-запросов со стороны клиентского компьютера, т. е. того компьютера, на котором работает пользователь. HTTP (HyperText Transport Protocol) — это протокол передачи WWW-документов через Интернет.

Технология JSP заключается в наличии дополнительных конструкций (тегов) в HTML-или XML-документах, которые позволяют серверу автоматически генерировать и компилировать сервлеты. В результате удается очень просто и удобно обрабатывать данные или элементы документа, а также внедрять в нужные места документа результаты обработки. Автоматическая генерация сервлетов происходит всего один раз — перед первой обработкой документа с тегами JSP на стороне сервера, поэтому код JSP выполняется достаточно быстро, но конечно требует, чтобы была установлена соответствующая Java-машина (например, входящая в состав сервера GlassFish — программного обеспечения, обеспечивающего поддержку большого количества необходимых серверных возможностей для работы в WWW). Отметим, что программный сервер GlassFish, основанный на кодах Sun Application Server, распространяется бесплатно и входит в состав NetBeans 6.5.

Более подробная информация о сетевом программировании и серверных приложениях Java будет изложена в *главе 11*.

Первоначально корпорация Sun Microsystems позиционировала Java как язык, обеспечивающий развитые графические возможности WWW-документов благодаря включению в них апплетов. Однако в настоящее время основные области применения Java — прикладное программирование на основе приложений, страниц JSP и сервлетов, а также других видов серверных программ. При этом апплеты играют незначительную роль.

Виртуальную Java-машину часто называют исполняющей средой (Java Runtime Environment — JRE).

Существуют два основных способа установки Java-машины на клиентский компьютер:

- ◆ JRE из поставки Java Development Kit (JDK) — комплект разработки программного обеспечения на языке Java. Как синоним может употребляться термин SDK — Software Development Kit (комплект разработки программного обеспечения);
- ◆ специализированный вариант JRE в составе интернет-браузера, называющийся Java plugin.

Комплект последних версий JDK можно свободно загружать с сайта Sun <http://java.sun.com/>.

При использовании апплетов требуется, чтобы в состав браузера входил специализированный комплект JRE. Как правило, он поставляется вместе с браузером и может при необходимости обновляться. Для MS Internet Explorer такой комплект и его обновления могут быть свободно загружены с сайта <http://www.java.com/>.

Имеется возможность установки Java-машины от различных производителей (не обязательно устанавливать комплект JDK от Sun). На одном и том же компьютере можно установить сразу несколько различных Java-машин (в том числе комплекты JDK разных версий). Правда, опыт показывает, что при этом некоторые программы, написанные на Java, теряют работоспособность (частично или полностью).

Комплекты JDK классифицируются исходя из версии Java (языка программирования и, соответственно, Java-машины) и типа создаваемых приложений. Так, ко времени написания данного текста выходили версии JDK 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6 и готовится к выходу 1.7. В каждой версии есть ряд подверсий, не сопровождающихся изменением языка программирования, а связанных в основном с исправлением ошибок или внесением небольших изменений в библиотеки (например, 1.4.1\_01 или 1.5.0\_04).

Версии Java 1.0 и 1.1 принято называть Java 1. Все версии Java, начиная с 1.2, называют Java 2. Однако более надежно классифицировать по номеру JDK, т. к. язык Java для версии JDK 1.5 заметно отличается по возможностям от языка Java для более ранних версий JDK — в него добавлено большое количество новых синтаксических конструкций, а также изменен ряд правил. Поэтому код, правильный в Java для версии JDK 1.5, может оказаться неправильным в Java для версии JDK 1.4 (не говоря уже про Java для версии JDK 1.3 или 1.2). Кроме того, начиная с JDK 1.5 компания Sun перестала использовать в названиях комплектов программного обеспечения термин "Java 2" и происходящие от него сокращения вида j2, а в нумерации комплектов JDK отказалась от префикса "1". Поэтому сейчас принято говорить о версиях JDK 5 или JDK 6, а не о JDK 1.5 или JDK 1.6. Но соответствующие папки при установке среды на компьютере все-таки называются jdk1.5 и jdk1.6.

Комплекты разработки JDK одной версии отличаются по типу создаваемых с их помощью приложений. По применяемым средствам разработки и условиям выполнения приложений Java различают три платформы:

- ◆ Микроиздание — Java Platform, Micro Edition (Java ME) <http://java.sun.com/javame/> для программирования "тонких аппаратных клиентов" (т. е. устройств, обладающих малыми ресурсами — наладонных компьютеров, сотовых телефонов, микроконтроллеров, смарт-карт). Старое название J2ME. Для разработки приложений применяется не JDK, а специальный набор инструментов (Sun Java Wireless Toolkit for CLDC, Sun Java Toolkit for CDC, NetBeans 6.5, инструментарий от других производителей). Приложения Java работают под управлением одной из двух имеющихся для этой платформы вариантов Java-машины: максимально облегченной килобайтной виртуальной машины KVM или несколько более мощной виртуальной машины JVM.
- ◆ Стандартное издание — Java Platform, Standard Edition (Java SE) <http://java.sun.com/javase/> для программирования "толстых клиентов" (т. е. устройств, обладающих достаточно большими ресурсами — обычных компьютеров и серверов).

Старое название J2SE. Основой средств разработки служит JDK. Приложения Java работают под управлением Java-машины JVM (Java Virtual Machine).

- ◆ Корпоративное издание — Java Platform, Enterprise Edition (Java EE) <http://java.sun.com/javaee/> для написания серверного программного обеспечения. Старое название J2EE. Является надстройкой над платформой Java SE, поэтому все приложения Java EE работают с использованием классов Java SE и под управлением JVM. В SDK данной платформы входит мощный программный сервер Sun Java System Application Server (сервер приложений Java), а также большое число других исполняющих сред, основанных на технологиях Java.

При распространении какого-либо продукта, написанного на Java для платформы Java SE, возможна установка только программного обеспечения Java-машины (JRE — Java Runtime Environment). Например, в случае Java 6 update 7 (обновление 7) комплект JDK назывался `jdk1.6.0_07`, и ему будет соответствовать комплект `jre1.6.0_07`. При этом создавалась папка с именем `jre1.6.0_07` с вложенными папками `bin` и `lib`. Начиная с Java 6 update 10, среда `jre` устанавливается в папку `jre6b` без указания номера обновления — эта и более поздние версии допускают обновление JRE без скачивания и переустановки всего дистрибутива.

Фактически, Java 6 update 10 — новый этап в развитии Java, и его отличие от предыдущей версии Java 6 update 7 (обновления 8 и 9 отсутствовали) гораздо больше, чем различия между Java 5 и Java 6. Очередной большой этап — предстоящий переход к Java 7.

Состав `jre6b` достаточно прост. В папке `bin` содержатся файлы и папки, необходимые для работы Java-машины и дополнительных инструментов для работы с ней в специальных режимах. В папке `lib` находятся вспомогательные файлы и библиотеки, в основном связанные с параметрами настроек системы.

Также возможна установка целиком JDK. Например, при установке Java 1.6 update 11 создается папка `jdk1.6.0_11` с вложенными папками:

- ◆ `bin` — содержатся файлы инструментов разработки;
- ◆ `demo` — файлы примеров с исходными кодами;
- ◆ `include` — заголовки файлов C для доступа к ряду библиотек Java и отладчику виртуальной Java-машины на платформо-зависимом уровне — на основе интерфейсов JNI (Java Native Interface) и JVMDI (Java Virtual Machine Debugging Interface) соответственно;
- ◆ `jre` — файлы, необходимые для работы с виртуальной Java-машиной;
- ◆ `lib` — ряд библиотек и сопроводительных файлов для работы инструментов из папки `bin`;
- ◆ `sample` — примеры с исходными кодами.

В папке `jdk1.6.0_11` содержится также архив `src.zip` с исходными кодами стандартных классов Java.

Следует отметить, что комплекты JRE, входящие в состав JDK и поставляемые отдельно от JDK, заметно различаются. Это связано с тем, что в отдельно поставляемом комплекте не нужны средства, предназначенные для отладки приложений и специальных

отладочных режимов работы с Java-машиной (управления загрузкой, профилирования, проверки стека и т. д.).

К сожалению, в JDK отсутствует даже самая простейшая документация с описанием назначения имеющихся в нем инструментов, там даны лишь ссылки на сайт компании Sun, где можно найти эту информацию. В табл. 1.1 перечислено назначение основных инструментов.

**Таблица 1.1.** Средства разработки приложений

Утилита	Назначение
javac	Компилятор в режиме командной строки для программ, написанных на языке Java
java	Утилита для запуска в режиме командной строки откомпилированных программ-приложений
appletviewer	Утилита для запуска на исполнение и отладку апплетов без браузера. При этом не гарантируется работоспособность отлаженного апплета в браузере
jdb	Отладчик программ, написанных на языке Java
javadoc	Генератор документации по классам на основе комментариев, начинающихся с <code>/**</code>
jar	Создание и управление Java-архивами jar
javah	Генератор заголовочных файлов C/C++ для подключения к программам Java внешних библиотек C/C++ на основе интерфейса JNI
javap	Дизассемблер классов
extcheck	Утилита для обнаружения конфликтов между файлами архивов jar
native2ascii	Утилита для конвертации в режиме командной строки параметра, передаваемого в виде текста на национальном алфавите, в последовательность символов Unicode

Кроме того, имеются средства поддержки работы в WWW и корпоративных сетях (интранет) с интерфейсом RMI — интерфейсом удаленного вызова методов (Remote Methods Invocation). Это программы *rmic*, *rmiregistry*, *rmid*. Также присутствуют средства поддержки информационной безопасности *keytool*, *jarsigner*, *policytool* и ряд других категорий утилит.

Подчеркнем, что набор утилит JDK рассчитан на морально устаревший режим командной строки, и что в большинстве случаев гораздо удобнее и правильнее пользоваться современной профессиональной средой разработки NetBeans. Для ее работы из JDK необходим только комплект JRE.

### 1.3. Алфавит языка Java.

#### Десятичные и шестнадцатеричные цифры и целые числа. Резервированные слова

Подробную информацию об алфавите языка Java и основных синтаксических конструкциях можно найти на сайте <http://java.sun.com/docs/books/jls/index.html> (на английском языке).

## Алфавит языка Java

Алфавит языка Java состоит из букв, десятичных цифр, разделителей и специальных символов. Буквами считаются латинские буквы (кодируются в стандарте ASCII), буквы национальных алфавитов (кодируются в стандарте Unicode, кодировка UTF-16), а также соответствующие им символы, кодируемые управляющими последовательностями (о них будет рассказано чуть позже).

Буквы и цифры можно использовать в качестве идентификаторов (т. е. имен) переменных, методов и других элементов языка программирования. Правда, при наличии в идентификаторах букв национальных алфавитов в ряде случаев могут возникнуть проблемы — эти символы будут показываться в виде вопросительных знаков.

Непосредственно как буквы рассматривается только часть символов национальных алфавитов, остальные — специальные символы — используются в качестве операторов и разделителей языка Java и не могут входить в состав идентификаторов.

Латинские буквы ASCII:

- ◆ ABCD...XYZ — заглавные (прописные);
- ◆ abcd...xyz — строчные.

Дополнительные буквенные символы ASCII:

- ◆ \_ — знак подчеркивания;
- ◆ \$ — знак доллара.

Национальные буквы на примере русского алфавита:

- ◆ АБВГ...ЭЮЯ — заглавные (прописные);
- ◆ абвг...эюя — строчные.

Десятичные цифры:

0 1 2 3 4 5 6 7 8 9

Разделители (их девять, задаются как символы ASCII):

( ) { } [ ] ; , .

## Десятичные и шестнадцатеричные цифры и целые числа

Целые числовые константы в исходном коде Java (так называемые литерные константы) могут быть десятичными или шестнадцатеричными. Они записываются либо символами ASCII, либо символами Unicode следующим образом.

Десятичные константы записывают как обычно (например, `-137`).

Константы-значения, записанные в виде последовательности символов (литер), называют *литерными* (иногда — *литералами*). В Java литерные константы бывают не только числовыми, но и булевыми (значения `true` и `false`), символьными (например, `'a'`, `'b'`, `'1'`, `'2'`), строковыми (например, `"Это - строка"`). Также имеется особая литерная константа `null`.