В. Монахов

Язык программирования Java и среда NetBeans

2-е издание

Санкт-Петербург
«БХВ-Петербург»
2009

УДК 681.3.068
ББК 32.973.26-018.1
М77

Монахов В. В.

М77 Язык программирования Java и среда NetBeans. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2009. — 720 с.: ил. + CD-ROM

ISBN 978-5-9775-0424-9

Книга написана на базе курса лекций, читаемых автором на кафедре вычислительной физики Санкт-Петербургского государственного университета. Второе издание ориентировано на использование среды NetBeans 6.5 и дополнено новыми материалами. Изложены основные синтаксические конструкции Java, принципы объектно-ориентированного программирования, особенности проведения численных расчетов. Приводятся сведения о разработке основных категорий 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 с подробным анализом их исходных кодов. На компакт-диске находятся дистрибутивы JDK 6u11 и NetBeans 6.5 для Windows и Linux, а также исходные тексты примеров.

Для начинающих Java-программистов и студентов

УДК 681.3.068
ББК 32.973.26-018.1

Группа подготовки издания:

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

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 27.02.09.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 58,05.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.003650.04.08 от 14.04.2008 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 978-5-9775-0424-9

© Монахов В. В., 2009
© Оформление, издательство "БХВ-Петербург", 2009

Оглавление

Введение

Предлагаемый учебный курс может рассматриваться 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 сделать 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. Возникла необходимость во втором издании. В нем изложена работа с NetBeans версии 6.5 (первое издание основывалось на версии 6.0), а также исправлены ошибки и опечатки, допущенные в первом издании. Кроме того, 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.

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 профессио-

нальные 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 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 дистрибутива NetBeans 6.5 (имеется на прилагаемом к книге компакт-диске) входят все основные модули, необходимые для программирования на Java. Кроме того, в него 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++.

Среда разработки NetBeans может быть установлена с прилагаемого к книге компакт-диска или свободно загружена с сайта Sun. 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? Фирма Sun получает 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 обеспечивает гораздо бóльшие прибыли, чем продажа этих продуктов.

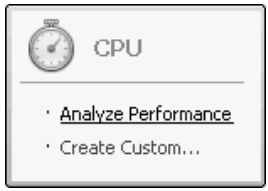
Java очень быстро развивается, особенно в последние годы. Из-за этого многие прежние представления и методы программирования на Java оказываются устаревшими. Среда

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 версии 6.5 с JDK 6.

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 Акопяну 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 составило 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). Процесс

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 прилагающуюся 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 платформо-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, предназначенный

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 некой абстрактной P-машины (P — 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 C, т. к. практически все промышленное программирование вплоть до середины последнего десятилетия XX века по реализации было системным. Идеи P-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 Бьерном Страуструпом путем такого расширения языка C был создан язык, который он назвал "C с классами". В 1983 году после очередных усовершенствований им был создан первый компилятор языка C++. Два плюса означают: "язык C с очень большим количеством добавлений", и одновременно — "новое, увеличенное на один по сравнению с прежним, значение" (игра понятий: в C существует оператор "++").

Язык C++ является надмножеством над языком C — на нем можно писать программы как на "чистом C", без использования каких-либо конструкций объектного программирования. В связи с этим, а также и с дополнительными преимуществами объектного программирования, язык C++ быстро приобрел популярность и стал промышленным стандартом (сначала "де-факто", а потом и "де-юре"). Так что в настоящее время C++ 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 C и C++ 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 компьюте-

ра, так и со стороны сервера. В том числе — для использования на *тонких аппаратных клиентах* (устройствах малой вычислительной мощности с крайне ограниченными ресурсами). При этом скомпилированные программы Java работают только под управлением виртуальной Java-машины, поэтому они называются *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 "дотнет", 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 платформо-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. Су-

ществует 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 платформо-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 — платформо-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 компилированным 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, предопределенных 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) — комплекта 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 CVM.
- Стандартное издание — Java Platform, Standard Edition (Java SE) **http://java.sun.com/javase/** для программирования "толстых клиентов" (т. е. устройств, об-

ладающих 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 jre6 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 jre6 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 комплек-

те 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 /**
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. Зарезервированные слова

Подробную информацию об алфавите языка его 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 "буквы" 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`.

Шестнадцатеричная литерная константа целого типа начинается с символов `0x` или `0X` (цифра `0`, после которой следует латинская буква `X`), затем идет само число в шестнадцатеричной нотации. Например:

`0x10` соответствует $10_{16} = 16$;

`0x2F` соответствует $2F_{16} = 47$ и т. д.

О шестнадцатеричной нотации будет рассказано чуть позже.

Ранее иногда применялись восьмеричные числа, в языках C/C++, а также и в Java их записывали в виде числа, начинающегося с цифры 0 (т. е. `010` означало $10_8 = 8$).

В настоящее время в программировании восьмеричные числа практически не применяют, а неадекватное использование ведущего нуля может приводить к логическим ошибкам в программе.

Целая литерная константа в обычной записи имеет тип `int`. Если после константы добавить букву `L` (или `l`, что хуже видно в тексте, хотя в среде разработки выделяется цветом), то она будет иметь тип `long`, обладающий более широким диапазоном значений, чем тип `int`.

Поясним теперь, что такое шестнадцатеричная нотация записи чисел и зачем она нужна.

Информация представляется в компьютере в двоичном виде — как последовательность битов. Бит — это минимальная порция информации, он может быть представлен в виде ячейки, в которой хранится ноль или единица. Но бит — слишком мелкая единица, поэтому в компьютерах информация хранится, кодируется и передается байтами — порциями по 8 битов.

В данной книге под "ячейкой памяти" будет пониматься непрерывная область памяти (с последовательно идущими адресами), выделенная программой для хранения данных. На рисунках мы будем изображать ячейку прямоугольником, внутри которого находятся хранящиеся в ней данные. Если у ячейки имеется имя, оно будет выведено рядом с этим прямоугольником.

Мы привыкли работать с числами, записанными в так называемой десятичной системе счисления. В этой системе имеются 10 цифр (от 0 до 9), а в числе имеются *десятичные разряды*. Каждый разряд слева имеет вес 10 по сравнению с предыдущим, т. е. для получения значения числа, соответствующего цифре в каком-то разряде, стоящую в нем цифру надо умножать на 10 в соответствующей степени (т. е. $52 = 5 * 10 + 2$; $137 = 1 * 10^2 + 3 * 10^1 + 7$, и т. п.).

В программировании десятичной системой счисления пользоваться не всегда удобно, т. к. в компьютерах информация организована в виде битов, байтов и более крупных порций. Человеку неудобно оперировать данными в виде длинных последовательностей нулей и единиц. В настоящее время в программировании стандартной является шестнадцатеричная система записи чисел. Например, с ее помощью естественным образом кодируется цвет, устанавливаются значения отдельных битов числа, осуществляется шифрование и дешифрование информации, и т. д. В этой системе счисления все очень похоже на десятичную, но только не 10, а 16 цифр, и вес разряда не 10, а 16. Первые десять цифр записывают обычными десятичными цифрами, а недостающие цифры (бóльшие 9) — заглавными латинскими буквами: `A`, `B`, `C`, `D`, `E`, `F`:

```
0 1 2 3 4 5 6 7 8 9 A B C D E F
```

То есть: `A = 10`, `B = 11`, `C = 12`, `D = 13`, `E = 14`, `F = 15`.

Заметим, что в шестнадцатеричной системе счисления числа от 0 до 9 записываются одинаково, а превышающие 9 отличаются. Для чисел от 10 до 15 в шестнадцатеричной системе счисления используют буквы от `A` до `F`, после чего задействуют следующий шестнадцатеричный разряд. Десятичное число 16 в шестнадцатеричной системе счисления записывается как 10. Для того чтобы не путать числа, записанные в разных системах счисления, около них справа пишут индекс с указанием *основания* системы счисления. Для десятичной системы счисления это 10, для шестнадцатеричной — 16. Для десятичной системы основание обычно не указывают (если это не приводит к путанице).

Точно так же в технической литературе часто не указывают основание для чисел, записанных в шестнадцатеричной системе счисления, если в записи числа встречаются не только обычные цифры от 0 до 9, но и буквенные цифры от `A` до `F`. Обычно употребляют заглавные буквы, но допустимы и строчные.

Примеры

$0x10 = 10_{16} = 16;$
$0x100 = 100_{16} = 16 \cdot 16 = 256;$
$0x1000 = 1000_{16} = (16)^3 = 4096;$
$0x20 = 20_{16} = 2 \cdot 16 = 32;$
$0x21 = 21_{16} = 2 \cdot 16 + 1 = 33;$
$0xF = F_{16} = 15;$
$0x1F = 1F_{16} = 1 \cdot 16 + 15 = 31;$
$0x2F = 2F_{16} = 2 \cdot 16 + 15 = 47;$
$0xFF = FF_{16} = 15 \cdot 16 + 15 = 255;$

Более подробно представление чисел в компьютере будет рассмотрено в *главе 4*.

Зарезервированные слова и литералы языка Java

В табл. 1.2 приведены ключевые слова (keywords), зарезервированные для синтаксических конструкций языка. Их назначение нельзя переопределять внутри программы, они не могут быть идентификаторами (именами переменных, подпрограмм и т. п.).

Таблица 1.2. *Зарезервированные слова языка Java*

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
enum	extends	final	finally	float
for	goto	if	implements	import
instanceof	int	interface	long	native
new	package	private	protected	public
return	short	static	strictfp	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

Кроме ключевых слов имеются предопределенные литералы `false`, `null` и `true`. Фактическое отличие предопределенных литералов от ключевых слов заключается только в том, что они представляют собой значения, а не элементы синтаксических конструкций. Их также нельзя переопределять внутри программы — использовать в качестве идентификаторов или изменять значение.

1.4. Управляющие последовательности. Символы Unicode. Специальные символы

Управляющие последовательности

Иногда в тексте программы в строковых константах требуются символы, которые обычным образом в текст ввести нельзя (например, кавычки, а также различные специальные символы). В этом случае прибегают к управляющей последовательности — символу обратной косой черты, после которой следует один управляющий символ.

Управляющие последовательности — символы формирования текста

В табл. 1.3 приведены управляющие последовательности, применяющиеся в языке Java для форматирования текста. Кроме очевидных символов форматирования текста, к ним относятся также кавычки (поскольку одни кавычки могут оказаться внутри других, что требует специальных соглашений в языке программирования).

Таблица 1.3. *Управляющие последовательности*

Символ	Что означает
`\b`	Возврат на один символ назад
`\f`	Перевод на новую страницу
`\n`	Перевод на новую строку
`\r`	Возврат к началу строки
`\t`	Горизонтальная табуляция
`\'`	Кавычка
`\"`	Двойные кавычки
`\\`	Обратная косая черта
`\u`	Начало кодировки символа Unicode

Управляющие последовательности Java, C и C++ в основном совпадают. В Java отсутствуют только последовательности, использование которых неактуально (`\a`, `\v`) или неудобно (`\?`).

Управляющие последовательности — символы Unicode

Управляющая последовательность может содержать несколько символов. Например, символы национальных алфавитов могут кодироваться последовательностью `\u`, после