
НА ЗАМЕТКУ

В таких врезках приведены полезные советы, ценные указания или замечания общего характера.

ВНИМАНИЕ!

Так оформлены места, на которые вы должны обратить особое внимание.

Об авторах

Роберт Джеймс Лигуори (Robert James Liguori) — руководитель компании Gliesian LLC, дипломированный специалист Oracle (Oracle Certified Expert). Он поддерживает работоспособность нескольких приложений управления воздушным движением, созданных на основе Java.

Патриция Лигуори (Patricia Liguori) работает в компании The MITRE Corporation в должности многопланового инженера по информационным системам. Начиная с 1994 года она занимается разработкой систем управления воздушным движением в реальном времени и информационных систем, связанных с авиацией.

Благодарности

Авторы хотят выразить особую благодарность редактору Меган Бланшет (Meghan Blanchette). Постоянный контроль и сотрудничество с ее стороны имели огромное значение для публикации этой книги.

Особое признание мы хотели бы выразить техническому редактору первого издания этой книги Майклу Лукидису (Michael Loukides) и нашему техническому рецензенту Райану Супраку (Ryan Suprak), а также многим членам коллектива издательства O'Reilly, нашей семье и друзьям.

Содержание

Введение	15
Структура книги	16
Соглашения, используемые в книге	16
Об авторах	17
Благодарности	17
Ждем ваших отзывов!	18
<hr/>	
Часть I. Язык	19
<hr/>	
Глава 1. Соглашение об именах	21
Имена классов	21
Имена интерфейсов	21
Имена методов	21
Имена переменных экземпляра и статических переменных	22
Имена параметров и локальных переменных	22
Имена параметров обобщенного типа	23
Имена констант	23
Имена перечислений	23
Имена пакетов	24
Имена аннотаций	24
Аббревиатуры	24
Глава 2. Лексические элементы	25
Символы Unicode и ASCII	25
Отображаемые ASCII-символы	26
Неотображаемые ASCII-символы	26
Комментарии	27
Ключевые слова	28
Идентификаторы	29
Разделители	30
Операторы	30
Литералы	32
Булевы литералы	32
Символьные литералы	32

Целочисленные литералы	32
Литералы с плавающей точкой	34
Строковые литералы	34
Нулевые литералы	35
Управляющие последовательности	35
Символы валют в Unicode	36
Глава 3. Основные типы	39
Простые типы	39
Литералы простого типа	40
Сущности с плавающей точкой	42
Операции с особыми сущностями	44
Числовое продвижение простых типов	44
Унарное числовое продвижение	45
Бинарное числовое продвижение	45
Особые случаи для условных операторов	46
Интерфейсные классы	46
Автоупаковка и распаковка	47
Автоупаковка	47
Распаковка	48
Глава 4. Ссылочные типы	51
Сравнение ссылочных и простых типов	51
Стандартные значения	52
Переменные экземпляра и локальные переменные	52
Массивы	53
Преобразование ссылочных типов	55
Расширяющие преобразования	55
Сужающие преобразования	55
Преобразования между простыми и ссылочными типами	56
Передача ссылочных типов в методы	56
Сравнение переменных ссылочного типа	57
Использование операторов равенства	57
Использование метода <code>equals()</code>	58
Сравнение строк	59
Сравнение перечислений	60
Копирование переменных ссылочного типа	60
Копирование ссылки на объект	60
Клонирование объектов	61

Поверхностное и глубокое клонирование	61
Распределение памяти и сборка мусора	62
Глава 5. Объектно-ориентированное программирование	63
Классы и объекты	63
Синтаксис класса	64
Создание экземпляра класса (объекта)	64
Данные-члены и методы	65
Доступ к данным-членам и методам объектов	65
Перегрузка	65
Переопределение	66
Конструкторы	67
Суперклассы и подклассы	68
Ключевое слово <code>this</code>	69
Списки с переменным числом параметров	70
Абстрактные классы и методы	72
Абстрактные классы	72
Абстрактные методы	72
Статические данные-члены, методы, константы и инициализаторы	73
Статические данные-члены	73
Статические методы	74
Статические константы	74
Статические инициализаторы	74
Интерфейсы	75
Перечисления	76
Типы аннотаций	76
Встроенные аннотации	77
Аннотации, определяемые разработчиком	77
Функциональные интерфейсы	79
Глава 6. Операторы и блоки	81
Операторы выражений	81
Пустой оператор	82
Блоки	82
Условные операторы	82
Оператор <code>if</code>	83
Оператор <code>if else</code>	83
Оператор <code>if else if</code>	83
Оператор <code>switch</code>	84

Итерационные операторы	85
Оператор цикла <code>for</code>	85
Операторы расширенного цикла <code>for</code>	85
Оператор цикла <code>while</code>	86
Оператор цикла <code>do-while</code>	86
Передача управления	87
Оператор <code>break</code>	87
Оператор <code>continue</code>	88
Оператор <code>return</code>	88
Оператор <code>synchronized</code>	89
Оператор <code>assert</code>	89
Операторы обработки исключений	90
Глава 7. Обработка исключений	91
Иерархия исключений	91
Проверяемые и непроверяемые исключения и ошибки	91
Проверяемые исключения	92
Непроверяемые исключения	92
Ошибки	93
Стандартные проверяемые и непроверяемые исключения и ошибки	93
Популярные проверяемые исключения	93
Популярные непроверяемые исключения	94
Популярные ошибки	95
Ключевые слова, используемые для обработки исключений	96
Ключевое слово <code>throw</code>	97
Ключевые слова <code>try/catch/finally</code>	97
Оператор <code>try-catch</code>	97
Оператор <code>try-finally</code>	99
Оператор <code>try-catch-finally</code>	100
Оператор <code>try</code> с ресурсами	100
Конструкция для перехвата нескольких исключений	101
Процесс обработки исключений	101
Определение собственного класса исключений	102
Вывод информации об исключениях	103
Метод <code>getMessage()</code>	103
Метод <code>toString()</code>	103
Метод <code>printStackTrace()</code>	104

Глава 8. Модификаторы Java	105
Модификаторы доступа	106
Остальные модификаторы доступа	106

Часть II. Платформа 109

Глава 9. Платформа Java, стандартный выпуск	111
Общие библиотеки Java SE API	112
Библиотеки поддержки языка и утилит	112
Базовые библиотеки	114
Библиотеки интеграции	116
Различные библиотеки пользовательского интерфейса	117
Библиотеки пользовательского интерфейса: JavaFX	118
Библиотеки пользовательского интерфейса: API AWT (устарело)	120
Библиотеки пользовательского интерфейса: Swing API (устарело)	121
Протокол RMI и библиотеки CORBA	123
Библиотеки обеспечения безопасности	125
Библиотеки XML	126
Глава 10. Основы разработки	129
Среда исполнения Java-программ (JRE)	129
Комплект разработчика программ на языке Java (JDK)	129
Структура программы на языке Java	130
Утилиты командной строки	132
Компилятор Java	133
Интерпретатор Java	134
Упаковщик Java-программ	137
Запуск JAR-файлов на выполнение	138
Документатор Java	139
Опция <code>classpath</code>	140
Глава 11. Управление памятью	143
Сборщики мусора	143
Последовательный сборщик мусора	144
Параллельный сборщик мусора	144
Параллельный сборщик мусора с уплотнением	144
Сборщик мусора с одновременной маркировкой и очисткой	145
Сборщик мусора Garbage-First (G1)	145

Средства управления памятью	145
Опции командной строки	147
Изменение размера кучи JVM	150
Метапространство	150
Взаимодействие с GC	151
Сборка мусора в явном виде	151
Финализация	151
Глава 12. Основы ввода-вывода	153
Стандартные потоки <code>in</code> , <code>out</code> и <code>err</code>	153
Иерархия основных классов ввода-вывода	154
Чтение и запись файлов	154
Чтение символьных данных из файла	155
Чтение двоичных данных из файла	156
Запись символьных данных в файл	156
Запись двоичных данных в файл	157
Чтение и запись сокетов	157
Чтение символьных данных из сокета	158
Чтение двоичных данных из сокета	158
Запись символьных данных в сокет	158
Запись двоичных данных в сокет	159
Сериализация	159
Сериализация	160
Десериализация	160
Сжатие и распаковка файлов	160
Работа с ZIP-архивами	161
Архивы в формате GZIP	161
Работа с файлами и каталогами	162
Часто используемые методы класса <code>File</code>	162
Доступ к существующим файлам	162
Позиционирование данных в файле	163
Глава 13. Новое API ввода-вывода NIO 2.0	165
Интерфейс <code>Path</code>	165
Класс <code>Files</code>	166
Дополнительные возможности	167
Глава 14. Параллелизм	169
Создание потоков	169

Расширение класса <code>Thread</code>	169
Реализация интерфейса <code>Runnable</code>	170
Состояния потока	170
Приоритеты потоков	171
Типичные методы	171
Синхронизация	173
Классы для поддержки параллелизма	174
Исполнители	174
Классы коллекций для параллельного выполнения	176
Синхронизаторы	176
Средства хронометража	177
Глава 15. Коллекции Java	179
Интерфейс <code>Collection</code>	179
Реализации	180
Методы инфраструктуры коллекций	180
Алгоритмы класса <code>Collections</code>	181
Эффективность алгоритмов	182
Функциональный интерфейс <code>Comparator</code>	183
Глава 16. Обобщения	187
Обобщенные классы и интерфейсы	187
Конструкторы с обобщениями	188
Принцип подстановки	189
Параметры типа, символы подстановки и ограничения	190
Принцип “взять и положить”	190
Обобщенная специализация	191
Обобщенные методы в первичных типах	192
Глава 17. Языки сценариев Java	195
Языки сценариев	195
Реализации интерпретаторов сценарных языков	195
Встраивание сценариев в Java-программы	195
Вызов методов сценарных языков	196
Доступ к ресурсам Java из сценариев	197
Установка сценарных языков и интерпретаторов	198
Установка сценарного языка	198
Установка интерпретатора сценарного языка	199
Проверка правильности установки интерпретатора	199

Глава 18. Дата и время	203
Устаревшая функциональная совместимость	204
Региональные календари	204
Календарь ISO	205
Основные классы API	205
Машинный интерфейс	207
Длительности и периоды	208
Соответствие типов JDBC и XSD	209
Форматирование	209
Глава 19. Лямбда-выражения	211
Основы лямбда-выражений	211
Синтаксис и примеры	212
Ссылки на методы и конструкторы	213
Функциональные интерфейсы специального назначения	214
Функциональные интерфейсы общего назначения	215
Дополнительная информация по лямбда-выражениям	216
Руководства	217
Общедоступные ресурсы	217
<hr/>	
Часть III. Приложения	219
Приложение А. Текущие интерфейсы API	221
Приложение Б. Средства сторонних разработчиков	223
Средства разработки, конфигурирования и тестирования	223
Библиотеки	226
Интегрированные среды разработки	228
Платформы веб-приложений	229
Интерпретируемые языки (совместимые с JSR-223)	231
Приложение В. Основы UML	233
Диаграммы классов	233
Наименование	234
Атрибуты	234
Операции	234
Видимость	235
Диаграммы объектов	235

Представление в виде графических символов	236
Классы, абстрактные классы и интерфейсы	236
Примечания	236
Пакеты	237
Соединения	237
Индикаторы кратности	237
Имена ролей	238
Отношения между классами	238
Ассоциация	238
Прямая ассоциация	239
Композиция	239
Агрегация	240
Временная ассоциация	240
Генерализация	240
Реализация	240
Диаграммы последовательности	240
Участник (1)	241
Найденное сообщение (2)	241
Синхронное сообщение (3)	241
Ответный вызов (4)	241
Асинхронное сообщение (5)	242
Сообщение, адресованное самому себе (6)	242
Линия жизни (7)	242
Полоса активности (8)	242
Предметный указатель	243

*Эта книга посвящается нашей прелестной, потрясающей
дочери — Эшли.*

Введение

Книги серии *Карманный справочник* призваны быть вашим спутником в офисе, в учебном классе и даже в пути. Эта книга может служить удобным справочником по стандартным возможностям языка программирования Java и его платформы.

Здесь собрана информация, которая понадобится вам при разработке или отладке программ на Java, в том числе полезные примеры программирования, таблицы, рисунки и листинги программ.

В книге содержится также вспомогательная информация по таким темам, как Java Scripting API, средства разработки сторонних фирм и основы унифицированного языка моделирования (Unified Modeling Language — UML).

Материал, представленный в книге, также поможет в подготовке к сдаче экзамена на получение квалификации Oracle Certified Associate Java SE 7 Programmer I. Если у вас есть желание получить такой сертификат, то, возможно, вам следует также приобрести книгу *OCA Java SE 7 Programmer I Study Guide (Exam IZO-803)* Эдварда Файнегана (Edward Finegan) и Роберта Лайгори (Robert Liguori), вышедшую в издательстве McGraw-Hill Osborne Media, 2012.

В настоящей книге представлены сведения о возможностях языка Java (до Java SE 8 включительно), в том числе новое API, предназначенное для работы с датой и временем, а также отдельная глава, посвященная лямбда-выражениям. Здесь описаны также новинки Java SE 7, дана базовая информация о NIO 2.0, описаны сборщик мусора G1 и небольшие улучшения языка Java JSR-334 (Project Coin). Улучшения Project Coin включают усовершенствованные литералы (например, возможность использования символа подчеркивания), новый ромбовидный оператор, связанный с обобщениями, и расширения, предназначенные для обработки исключений, такие, как, например, новые операторы мультиперехвата (multi-catch) и оператор try с ресурсами (try-with-resources).

Структура книги

Книга состоит из трех частей: “Язык”, “Платформа” и “Приложения”. В главах 1–8 подробно рассматривается язык программирования Java с точки зрения спецификации языка Java (Java Language Specification — JLS). В главах 9–19 подробно описаны компоненты платформы Java и связанные с этим темы. В приложениях рассматриваются средства разработки сторонних производителей и основы унифицированного языка моделирования (Unified Modeling Language — UML).

Соглашения, используемые в книге

Для выделения различных элементов используются следующие соглашения об обозначениях.

Текст курсивом

Обозначает новые термины и понятия, а также важные моменты, на которых акцентируется внимание.

Моноширинный шрифт

Используется для листингов программ, а также в абзацах для ссылки на такие элементы программ, как названия переменных или функций, типы, базы данных, переменные среды, операторы и ключевые слова. Кроме того, таким шрифтом выделяются URL и адреса электронной почты.

Полужирный моноширинный шрифт

Используется для выделения команд или текстовых литералов, которые должны быть введены с клавиатуры непосредственно пользователем.

Моноширинный курсив

Обозначает текст, вместо которого следует подставить значение, указанное пользователем, или значение, определяемое по контексту.

Операторы и блоки

Оператор представляет собой отдельно взятую команду, которая осуществляет то или иное действие интерпретатора Java при выполнении программы.

```
GigSim simulator = new GigSim(  
    "Давай, сыграем на гитаре!");
```

К числу операторов Java относятся: выражение, пустой оператор, блок, условный оператор, итерация, передача управления, обработка исключения, переменная, метка, утверждение и синхронизированные операторы.

Зарезервированными словами, которые используются в операторах Java, являются: `if`, `else`, `switch`, `case`, `while`, `do`, `for`, `break`, `continue`, `return`, `synchronized`, `throw`, `try`, `catch`, `finally` и `assert`.

Операторы выражений

Оператор выражения — это оператор, который изменяет состояние программы; любое выражение в Java заканчивается точкой с запятой. К числу операторов выражения относятся: присвоения, префиксные и постфиксные приращения (инкременты), префиксные и постфиксные вычитания (декременты), создание объекта и вызовы методов. Ниже приведены примеры операторов выражений.

```
isWithinOperatingHours = true;  
++fret; patron++; --glassOfWater; pick--;  
Guitarist guitarist = new Guitarist();  
guitarist.placeCapo(guitar, capo, fret);
```

Пустой оператор

Пустой оператор не выполняет какого-либо действия и записывается в виде символа точки с запятой (;) или как пустой блок {}.

Блоки

Группа операторов называется блоком или блоком операторов. Блок операторов заключается в фигурные скобки. Переменные и классы, объявленные в блоке, называются локальными переменными и локальными классами соответственно. Областью действия локальных переменных и локальных классов является блок, в котором они объявлены.

В блоках операторы интерпретируются друг за другом, в той последовательности, в которой они записаны, или в порядке управления выполнением программы. Ниже приведен пример блока.

```
static {
    GigSimProperties.setFirstFestivalActive(true);
    System.out.println("Первый фестиваль открыт!");
    gigsimLogger.info("Начался первый фестиваль.");
}
```

Условные операторы

Операторы `if`, `if else` и `if else if` являются операторами управления выполнением программы (операторами принятия решений). Они используются для выполнения операторов в зависимости от наступления тех или иных условий. Выражение для любого из этих операторов должно иметь тип `Boolean` или `boolean`. При использовании типа `Boolean` будет выполнена распаковка, т.е. автоматическое преобразование `Boolean` в `boolean`.

Оператор `if`

Оператор `if` состоит из выражения и оператора или блока операторов, которые выполняются, если значение соответствующего выражения равно `true`.

```
Guitar guitar = new Guitar();
guitar.addProblemItem("Треснувший гриф");
if (guitar.isBroken()) {
    Luthier luthier = new Luthier();
    luthier.repairGuitar(guitar);
}
```

Оператор `if else`

Если оператор `if` используется в сочетании с `else`, то первый блок операторов выполняется, если значение соответствующего выражения равно `true`; в противном случае выполняется блок кода в `else`.

```
CoffeeShop coffeeshop = new CoffeeShop();
if (coffeeshop.getPatronCount() > 5) {
    System.out.println("Отобразить приветствие.");
} else {
    System.out.println("Автомат неисправен!");
}
```

Оператор `if else if`

Оператор `if else if` обычно используется, когда нужно сделать выбор между несколькими блоками кода. Когда используемый критерий выбора не позволяет выполнить ни один из этих блоков, выполняется код в последнем блоке `else`.

```
ArrayList<Song> playList = new ArrayList<>();
Song song1 = new Song("Mister Sandman");
Song song2 = new Song("Amazing Grace");
playList.add(song1);
playList.add(song2);
...
int numOfSongs = playList.size();
```

```

if (numOfSongs <= 24) {
    System.out.println("Мало песен.");
} else if ((numOfSongs > 24) & (numOfSongs < 50)){
    System.out.println("Хватит на один вечер");
} else if ((numOfSongs >= 50) & (numOfSongs < 100)) {
    System.out.println("Хватит на два вечера.");
} else {
    System.out.println("Хватит на неделю.");
}

```

Оператор switch

Оператор `switch` является оператором управления выполнением программы. В зависимости от значения его выражения управление передается одному из следующих за ним операторов `case`. Оператор `switch` работает с типами `char`, `byte`, `short`, `int`, а также с интерфейсными типами `Character`, `Byte`, `Short` и `Integer`; с типами перечислений и типом `String`. Поддержка объектов `String` была обеспечена только в Java SE 7. Оператор `break` используется для завершения работы оператора `switch`. Если оператор `case` не содержит оператора `break`, то будет выполняться следующая за ним строка кода (обычно следующий оператор `case`).

Это продолжается до тех пор, пока либо не будет достигнут оператор `break`, либо конец оператора `switch`. Допускается использование одной метки `default`; как правило, она используется в конце оператора `switch` с целью улучшения читабельности кода.

```

String style;
String guitarist = "Эрик Клэптон";
...
switch (guitarist) {
    case "Чет Аткинс":
        style = "Фингерстайл";
        break;

    case "Томми Эммануэль":
        style = "Сложная импровизация";
        break;
}

```

```
default:
    style = "Неизвестен";
    break;
}
```

Итерационные операторы

Операторы простого и расширенного цикла `for`, а также операторы `while` и `do-while` являются итерационными. Они используются для циклического выполнения определенных фрагментов кода.

Оператор цикла `for`

Оператор цикла `for` состоит из трех частей: инициализации, условного выражения и обновления. Как показано ниже, перед использованием этого оператора должна быть инициализирована переменная цикла (т.е. `i`). Условное выражение (т.е. `i < bArray.length()`) всегда вычисляется до начала выполнения цикла. Итерация (т.е. выполнение цикла) начинается лишь в случае, если значение условного выражения истинно, а переменная цикла обновляется (т.е. `i++`) после каждой очередной итерации.

```
Banjo [] bArray = new Banjo[2];

bArray[0] = new Banjo();
bArray[0].setManufacturer("Windsor");

bArray[1] = new Banjo();
bArray[1].setManufacturer("Gibson");

for (int i=0; i < bArray.length; i++){
    System.out.println(bArray[i].getManufacturer());
}
```

Операторы расширенного цикла `for`

Операторы расширенного цикла `for`, т.е. циклы типа “`for in`” или “`for each`”, используются для итерационной обработки того

или иного объекта или массива, допускающего перебор своих элементов. Цикл выполняется однократно для каждого элемента массива или коллекции и не использует счетчик, поскольку количество итераций известно заранее.

```
ElectricGuitar eGuitar1 = new ElectricGuitar();
eGuitar1.setName("Blackie");

ElectricGuitar eGuitar2 = new ElectricGuitar();
eGuitar2.setName("Lucille");

ArrayList <ElectricGuitar> eList = new ArrayList<>();
eList.add(eGuitar1);
eList.add(eGuitar2);

for (ElectricGuitar e : eList) {
    System.out.println("Name:" + e.getName());
}
```

Оператор цикла while

В операторе цикла while сначала вычисляется выражение, и цикл выполняется лишь в том случае, если значение выражения истинно. Выражение может быть типа boolean или Boolean.

```
int bandMembers = 5;
while (bandMembers > 3) {
    CoffeeShop c = new CoffeeShop();
    c.performGig(bandMembers);
    Random generator = new Random();
    bandMembers = generator.nextInt(7) + 1; // 1-7
}
```

Оператор цикла do-while

В операторе do-while цикл всегда выполняется по меньшей мере один раз и будет выполняться до тех пор, пока значение выражения истинно. Само выражение может быть типа boolean или Boolean.

```
int bandMembers = 1;
do {
```

```
CoffeeShop c = new CoffeeShop();
c.performGig(bandMembers);
Random generator = new Random();
bandMembers = generator.nextInt(7) + 1; // 1-7
} while (bandMembers > 3);
```

Передача управления

Операторы передачи управления используются для изменения потока команд в программе. К числу таких операторов относятся: `break`, `continue` и `return`.

Оператор `break`

Оператор `break` без метки используется для выхода из текущей ветки оператора `switch` или немедленного выхода из цикла. Этот оператор может прекращать работу простого и расширенного цикла `for`, а также циклов типа `while` и `do-while`.

```
Song song = new Song("Pink Panther");
Guitar guitar = new Guitar();
int measure = 1;
int lastMeasure = 10;

while (measure <= lastMeasure) {
    if (guitar.checkForBrokenStrings()) {
        break;
    }
    song.playMeasure(measure);
    measure++;
}
```

Оператор `break` с меткой приводит к завершению цикла с указанной меткой и передаче управления следующему за этим циклом оператору. Метки обычно используются с циклами `for` и `while`, когда есть вложенные циклы и требуется точно определить, из какого именно цикла нужно выйти. Чтобы пометить какой-либо цикл или оператор, поместите оператор метки непосредственно перед помечаемым циклом или оператором, как показано в приведенном ниже примере.

```

...
playMeasures:
while (isWithinOperatingHours()) {
    while (measure <= lastMeasure) {
        if (guitar.checkForBrokenStrings()) {
            break playMeasures;
        }
        song.playMeasure(measure);
        measure++;
    }
}
// Выход осуществляется в эту точку

```

Оператор `continue`

Выполнение оператора `continue` без метки приводит к прекращению выполнения текущей итерации простого или расширенного цикла `for`, а также цикла `while` или `do-while` и началу следующей итерации соответствующего цикла. При этом будет выполнена проверка условий цикла. Оператор `continue` с меткой вызывает следующую итерацию указанного цикла:

```

for (int i=0; i<25; i++) {
    if (playList.get(i).isPlayed()) {
        continue;
    } else {
        song.playAllMeasures();
    }
}

```

Оператор `return`

Оператор `return` используется для выхода из метода и возвращения того или иного значения, если в описании метода указано, что данный метод должен вернуть значение определенного типа.

```

private int numberOfFrets = 18; // Стандартное значение
...
public int getNumberOfFrets() {
    return numberOfFrets;
}

```

Оператор `return` использовать необязательно, если он является последним оператором в методе и этот метод ничего не возвращает.

Оператор `synchronized`

Ключевое слово `synchronized` в Java может использоваться для предоставления доступа к определенным участкам кода (например, к определенным методам) только одному потоку команд. Оператор `synchronized` позволяет управлять доступом к ресурсам, которые совместно используются в нескольких потоках. Подробнее об этом рассказывается в главе 14.

Оператор `assert`

Утверждения (assertions) представляют собой булевы выражения, используемые в режиме отладки для проверки, ведет ли себя код именно так, как ожидалось (т.е. при запуске приложения с использованием параметра командной строки `-enableassertions` или `-ea` интерпретатора Java). Утверждения записываются следующим образом:

```
assert булево_выражение;
```

Утверждения облегчают выявление ошибок, в том числе обнаружение неожиданных значений. Они предназначены для подтверждения предположений, которые всегда должны быть истинны. При выполнении в режиме отладки, если значение утверждения ложно, генерируется исключение `java.lang.AssertionError` и осуществляется выход из программы; в противном случае ничего не происходит. Утверждения нужно активизировать в явном виде. Аргументы командной строки, используемые для активизации утверждений, описаны в главе 10.

```
// Значение переменной 'strings' должно  
// равняться 4, 5, 6, 7, 8 или 12
```

```
assert (strings == 12 ||
        (strings >= 4 & strings <= 8));
```

В утверждении можно также указать необязательный код ошибки. Несмотря на такое название (“код ошибки”), в действительности он представляет собой обычный текст или значение, используемые исключительно для информационных целей.

```
assert булево_выражение : код_ошибки;
```

Когда утверждение, в котором содержится код ошибки, истинно, значение кода ошибки преобразуется в строку и отображается для пользователя непосредственно перед завершением работы. Ниже приведен пример утверждения, в котором используется код ошибки.

```
// Показать недопустимое количество струн
// для музыкального инструмента'
assert (strings == 12 ||
        (strings >= 4 && strings <= 8))
        : "Недопустимое число струн: " + strings;
```

Операторы обработки исключений

Операторы обработки исключений используются для указания кода, который должен быть выполнен при возникновении необычных обстоятельств. Для обработки исключений используются ключевые слова `throw` и `try/catch/finally`. Подробнее об обработке исключений можно прочитать в главе 7.