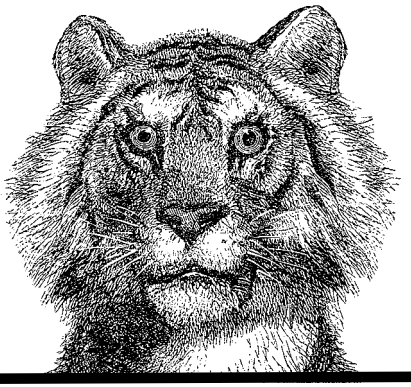


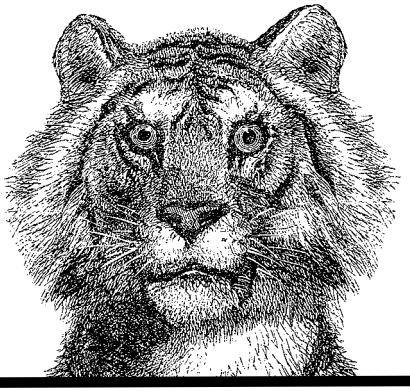
JAVA™ IN A NUTSHELL

A Desktop Quick Reference

Fourth Edition

David Flanagan

O'REILLY®



JAVA™

СПРАВОЧНИК

Четвертое издание

Дэвид Флэнаган

Санкт-Петербург — Москва
2004

Дэвид Флэнаган

Java. Справочник
4-е издание

Перевод К. Волкова и В. Шальнева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*И. Васильев*
Редактор	*Ю. Кунивер*
Корректор	*С. Беляева*
Верстка	*Ю. Кунивер*

Флэнаган Д.
Java. Справочник, 4-е издание – Пер. с англ. – СПб: Символ-Плюс, 2004. – 1040 с., ил.
ISBN 5-93286-067-7

Этот бестселлер представляет собой краткий справочник, необходимый каждому Java-программисту. Книга содержит ускоренный вводный курс в язык Java и обзор ключевых API, благодаря чему опытные программисты смогут сразу перейти к написанию Java-кода. Четвертое издание «Java. Справочник» посвящено Java 1.4 и включает краткое описание синтаксиса Java, изложение объектно-ориентированных возможностей Java и обзор основных API Java, в котором объясняется, как выполнять такие стандартные задачи, как работа со строками, ввод/вывод, обработка XML, SSL и поддержка потоков при помощи классов и интерфейсов, составляющих платформу Java 2.

Книга также содержит заслуживающий доверия справочник O'Reilly по всем классам, входящим в базовые Java-пакеты, такие как *java.lang*, *java.io*, *java.beans*, *java.math*, *java.net*, *java.text* и *java.util*. Справочник охватывает множество новых классов Java 1.4, включая NIO (новый интерфейс ввода/вывода), протоколирование и средства работы с XML.

При изучении Java неоценимую помощь окажет вам и книга «Java в примерах. Справочник» (Символ-Плюс, 2003), которая отлично дополняет данное издание.

ISBN 5-93286-067-7
ISBN 0-596-00283-1 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 29.12.2003. Формат 70x100 1/16. Печать офсетная.
Объем 65 печ. л. Тираж 2000 экз. Заказ N
Отпечатано с диапозитивов в Академической типографии «Наука» РАН
199034, Санкт-Петербург, 9 линия, 12.

Эта книга посвящается всем,
кто учит жить мирно и противостоит насилию.

Оглавление

Предисловие 11

Часть I. Введение в Java 21

Глава 1. Введение 23

Что такое Java? 23
Основные преимущества Java 26
Пример программы 29

Глава 2. Синтаксис Java: от подножия к вершине 38

Символьный набор Unicode 39
Комментарии 39
Идентификаторы и зарезервированные слова 40
Примитивные типы данных 41
Выражения и операторы 48
Операторы-инструкции 62
Методы 81
Классы и объекты 83
Массивы 86
Ссылочные типы 91
Пакеты и пространство имен Java 98
Структура файла Java 99
Определение и выполнение Java-программ 100
Различия между языками C и Java 101

Глава 3. Объектно-ориентированное программирование в Java 104

Члены класса 104
Создание и инициализация объектов 110
Подготовка к уничтожению и уничтожение объектов 114
Подклассы и наследование 117
Сокрытие данных и инкапсуляция 126
Абстрактные классы и методы 131
Интерфейсы 133
Внутренние классы: обзор 137
Статические классы-члены 139
Классы-члены 140

Локальные классы 144
Анонимные классы 147
Как работают внутренние классы 150
Модификаторы: сводка 152
Особенности C++, отсутствующие в Java 153

Глава 4. Java-платформа 155

Обзор Java-платформы 155
Строки и символы 158
Числа и математика 163
Дата и время 166
Массивы 167
Коллекции 168
Типы, отражение и динамическая загрузка 170
Потоки 172
Файлы и каталоги 177
Потоки ввода/вывода 179
Работа в сети 184
Свойства и предпочтения 189
Протоколирование 190
Новый API ввода/вывода 191
XML 203
Процессы 208
Безопасность 209
Шифрование 211

Глава 5. Безопасность в Java 214

Угрозы безопасности 214
Защита виртуальной машины Java и верификация файлов классов 215
Аутентификация и криптография 215
Контроль доступа 216
Безопасность для всех 219
Классы прав доступа 221

Глава 6. Компоненты JavaBeans 222

Основы компонентов Java 223
Соглашения JavaBeans 225
Контексты и сервисы компонентов Java 232

Глава 7. Соглашения по программированию и документированию в Java 233

Соглашения по именованию и применению прописных букв 233
Соглашения по переносимости и правила чистого языка Java (Pure Java) 234
Документация в комментариях 237

Глава 8. Средства разработки Java 245

appletviewer 245
extcheck 249
Jar 249
jarsigner 252
java 254
javac 261
javadoc 264
javah 271
javap 273
jdb 274
keytool 279
native2ascii 283
policytool 283
serialver 285

Часть II. Справочник по API 287

Как использовать этот справочник 289

Как найти статью в справочнике 289
Как читать статью в справочнике 290

Глава 9. java.beans и java.beans.beancontext 297

Глава 10. java.io 343

Глава 11. java.lang, java.lang.ref и java.lang.reflect 401

Глава 12. java.math 478

Глава 13. java.net 482

Глава 14. java.nio и подпакеты 518

Глава 15. java.security и подпакеты 577

Глава 16. java.text 663

Глава 17. java.util и подпакеты 691

Глава 18. javax.crypto и подпакеты 805

Глава 19. javax.net и javax.net.ssl 832

Глава 20. javax.security.auth и подпакеты 854

Глава 21. javax.xml.parsers, java.xml.transform и подпакеты 879

Глава 22. org.ietf.jgss 902

Глава 23. org.w3c.dom 909

Глава 24. org.xml.sax, org.xml.sax.ext и org.xml.sax.helpers 925

Глава 25. Указатель классов, методов и полей 951

Алфавитный указатель 999

Предисловие

Данная книга представляет собой настольный справочник для программистов, пишущих на Java™. Она предназначена для того, чтобы лежать рядом с вашей клавиатурой и быть под рукой, когда вы программируете. Первая часть книги – краткое, «сухое» введение в язык программирования Java и в базовый набор функций *прикладного программного интерфейса* (API) платформы Java. Вторая часть – это справочный раздел, который дает лаконичное описание деталей большинства классов и интерфейсов базовых API Java. Книга охватывает версии Java 1.0, 1.1, 1.2, 1.3 и 1.4.

Изменения в четвертом издании

Так как платформа Java снова значительно расширилась с выпуском Java 1.4, соответственно увеличился и объем данной книги. Далее приводятся некоторые из наиболее важных особенностей, появившихся в Java 1.4 (а также в этой книге):

Оператор assert (Assert statement)

Язык Java был расширен для поддержки утверждений при помощи оператора `assert`. Этот новый оператор описан в главе 2.

Постоянство JavaBeans (JavaBeans persistence)

Компоненты JavaBeans и связанные с ними объекты могут быть сериализованы в XML-документы. Более подробную информацию можно получить в разделе `java.beans.XMLEncoder` главы 9.

Новый прикладной программный интерфейс ввода/вывода (I/O API)

Для поддержки высокопроизводительного и неблокирующего файлового и сетевого ввода/вывода в версию Java 1.4 включен новый прикладной программный интерфейс. См. описание пакета `java.nio` и его подпакетов в главе 14. В главе 4 приведено несколько примеров использования этого важного интерфейса.

Прикладной программный интерфейс маршрутов сертификации (Certification path API)

Пакет `java.security.cert` был расширен новыми классами и интерфейсами для создания цепочек сертификатов или «маршрутов сертификации», обычно используемых в сетевой аутентификации.

Прикладной программный интерфейс протоколирования (Logging API)

Новый пакет `java.util.logging` определяет мощный и гибкий каркас протоколирования для приложений Java.

Прикладной программный интерфейс для работы с настройками пользователя (Preferences API)

Пакет `java.util.prefs` определяет прикладной программный интерфейс, позволяющий приложениям хранить и запрашивать значения настроек пользователей и общесистемных опций конфигурации.

Регулярные выражения (Pattern matching with regular expressions)

Новый пакет `java.util.regex` позволяет обрабатывать текстовые шаблоны с помощью регулярных выражений в стиле Perl.

Защищенные сетевые сокеты (Secure network sockets)

Расширение прикладного программного интерфейса протокола защищенных сокетов (Java Secure Sockets Extension (JSSE) API), определенное в новых пакетах `javax.net` и `javax.net.ssl`, обеспечивает защиту при работе в сети посредством протоколов SSL и TLS.

Сетевая аутентификация и авторизация (Network authentication and authorization)

Служба аутентификации и авторизации Java (Java Authentication and Authorization Service (JAAS)) определяется в пакете `javax.security.auth` и его подпакетах. JAAS позволяет приложениям Java безопасно устанавливать личность пользователя и выполнять код в соответствии с политикой безопасности на базе прав данного пользователя.

Синтаксический анализ и преобразование XML (XML parsing and transformations)

Прикладной программный интерфейс Java для обработки данных XML (Java API for XML Processing (JAXP)) определяется в пакетах `javax.xml.parsers`, `javax.xml.transform` и их подпакетах. JAXP предоставляет средства для синтаксического анализа XML-документов с использованием стандартных прикладных программных интерфейсов SAX и DOM, а также преобразование содержимого этих документов с использованием XSLT. Подобно JAXP, стандартные прикладные программные интерфейсы DOM и SAX являются частью платформы Java 1.4. Вы сможете найти их в пакете `org.w3c.dom`, описанном в главе 23, а также в пакете `org.xml.sax` и подпакетах, представленных в главе 24.

В главе 4 вы найдете примеры использования большинства прикладных программных интерфейсов.

Помимо добавления нового материала, было внесено несколько изменений и в структуру книги. В предыдущих изданиях на каждый пакет выделялась отдельная глава справочного раздела. Сорок шесть отдельных пакетов, освещенных в данном издании, привели бы к чрезмерному количеству глав. Поэтому связанные пакеты (с общим префиксом) сгруппированы в одну главу, благодаря чему количество глав справочного раздела сокращено до 15. Однако из-за того что краткий справочник организован в строгом алфавитном порядке, разбиение на главы выполнено не по тематическому признаку. Интересующий вас раздел можно найти путем пролистывания раздела, как в случае словаря или телефонного справочника.

Кроме того, по причине большого количества новых пакетов пришлось убрать рисунки, представляющие иерархию пакетов, которые располагались в начале каждой главы в прошлых изданиях. Данные рисунки легли непомерным грузом на плечи группы технических иллюстраторов компании O'Reilly & Associates, Inc., поскольку их нужно было тщательно вырисовывать вручную. Более того, рисунки оказались не

так полезны, как это представлялось вначале. Для данного издания было решено, что рисунки не стоят затрачиваемых на них средств. Если вы все же принадлежите к числу тех немногочисленных читателей, которым пригодились эти диаграммы, приношу свои извинения за их удаление.

Есть две новые особенности в справочном разделе, способные компенсировать потерю иерархических диаграмм. Во-первых, раздел ссылок для каждого пакета включает в себя перечень всех интерфейсов и классов пакета. Данные в этом списке будут группироваться по категориям (например, интерфейсы, классы и исключения) и по иерархическому представлению. Такой перечень, не будучи графическим, предоставляет такое же количество информации, что и прежние иерархические диаграммы. Во-вторых, для каждого класса и интерфейса иерархический подраздел был переведен из неудобного текстового формата в более совершенный графический.

Структура книги

В первых восьми главах данной книги описаны язык Java, платформа Java и средства разработки Java, которые входят в набор инструментальных средств разработки Java, поставляемый компанией Sun (Java SDK). Первые четыре главы являются базовыми, а последующие четыре будут интересны не всем Java-программистам.

Глава 1 «Введение»

Эта глава содержит обзор языка Java и Java-платформы, с объяснением важных возможностей и преимуществ Java. В конце главы новоиспеченный Java-программист знакомится с примером Java-программы и проходит ее шаг за шагом.

Глава 2 «Синтаксис Java: от подножия к вершине»

В этой главе подробно описывается язык программирования Java. Глава большая и обстоятельная. Опытные Java-программисты могут использовать ее в качестве справочника по языку. Прочитав данную главу, программисты со значительным опытом работы с языком C или C++ ознакомятся с синтаксисом Java. Однако глава не подразумевает обязательное наличие большого опыта программирования и не требует знания C или C++. Данное издание разработано таким образом, чтобы даже начинающие программисты со скромным опытом смогли разобраться с программированием на Java, тщательно изучив данную главу.

Глава 3 «Объектно-ориентированное программирование в Java»

В этой главе описаны пути использования базового синтаксиса Java, рассматриваемого в главе 2, при написании объектно-ориентированных программ на языке Java. Данная глава не предполагает наличия какого-либо предварительного опыта в ООП. Она может служить учебным пособием для начинающих программистов и справочником для опытных Java-разработчиков.

Глава 4 «Java-платформа»

В этой главе дается краткий обзор основных прикладных программных интерфейсов Java, рассматриваемых в данной книге. Глава содержит множество небольших примеров решения типовых задач с использованием классов и интерфейсов, входящих в платформу Java. Программисты, не знакомые с Java, особенно те, кому легче учиться на примерах, по достоинству оценят эту главу.

Глава 5 «Безопасность в Java»

В данной главе описана архитектура безопасности Java, которая позволяет запускать ненадежный код в безопасной среде, из которой он не сможет намеренно

причинить вред системе. Для каждого Java-программиста важно иметь хотя бы общее представление о механизмах безопасности Java.

Глава 6 «Компоненты JavaBeans»

В данной главе представлена структура компонентов Java (JavaBeans™) и объяснено, что должен знать программист для создания и применения многократно используемых встраиваемых классов Java, известных как компоненты (beans).

Глава 7 «Соглашения по программированию и документированию в Java»

В данной главе изложены важные общепринятые соглашения по написанию Java-программ. Здесь же описано, как сделать Java-код самодокументируемым путем включения в него специальных комментариев.

Глава 8 «Средства разработки Java»

В набор средств разработки Java (Java SDK) входит большое количество полезных инструментальных средств разработки. В частности, можно выделить интерпретатор и компилятор Java. В данной главе описаны эти средства.

Первые восемь глав помогут вам изучить язык Java и дадут возможность освоить и начать работать с прикладными программными интерфейсами Java. Однако большая часть книги (главы 9–24) является справочником по прикладным программным интерфейсам – детальным и одновременно лаконичным описанием, удобным в применении. Важно внимательно изучить первую главу в начале справочного раздела, где объясняется, как наиболее продуктивно использовать справочный материал. Также обратите внимание, что за главами справочного раздела следует заключительная глава под названием «Алфавитный указатель классов, методов и полей». По этому указателю можно отыскать имя класса и найти пакет, в котором он определен, или отыскать имя метода или поля и найти соответствующий пакет.

Книги по теме

O'Reilly издает целую серию книг по Java-программированию, включая несколько книг, дополняющих данное издание. Вот эти книги:

«Java Enterprise in a Nutshell»

Данная книга является кратким учебным пособием и справочником по таким прикладным программным интерфейсам Java для приложений масштаба предприятия, как JDBC, RMI, JNDI и CORBA.

«Java Foundation Classes in a Nutshell»

Эта книга является кратким учебным пособием и справочником по графике, графическому интерфейсу пользователя и по соответствующим прикладным программным интерфейсам платформы Java. Сюда входят апплеты, AWT, Java2D и Swing.

«Java Examples in a Nutshell»[1]

Книга содержит сотни законченных рабочих примеров, иллюстрирующих решения большинства распространенных задач, использующих базовый прикладной программный интерфейс Java, дополнительные программные интерфесы и программные интерфейсы для приложений уровня предприятия. Подобно главе 4 данного издания, книга «Java Examples in a Nutshell» значительно расширена, а все фрагменты кода доведены до уровня работающих примеров. Эта книга особенно по-

[1] Д. Флэнаган «Java в примерах. Справочник». – Пер. с англ. – СПб: Символ-Плюс, 2003.

лезна тем читателям, которые легче обучаются, экспериментируя с уже существующим кодом.

«J2ME in a Nutshell»

Данная книга является кратким учебным пособием и справочником по графике, работе в сети и прикладным программным интерфейсам баз данных платформы Java 2 Micro Edition (J2ME).

Полный перечень книг о Java от O'Reilly можно найти на сайте *http://java.oreilly.com/*. Ниже представлен список книг об основных прикладных программных интерфейсах Java (как, например, данная книга):

«Learning Java», Пэт Нимейер (Pat Niemeyer) и Джонатан Кнадсен (Jonathan Knudsen)

Исчерпывающее введение в Java с акцентом на создание клиентских приложений.

«Java Threads», Скотт Оукс (Scott Oaks) и Генри Уонг (Henry Wong)

Хотя Java и облегчает многопоточное программирование, оно все еще требует определенной сноровки. В данной книге изложена вся необходимая информация.

«Java I/O», Эллиот Расти Гарольд (Elliotte Rusty Harold)

Архитектура ввода/вывода в Java, основанная на концепции потоков, прекрасна сама по себе. В данной книге она рассматривается с той тщательностью, которую заслуживает.

«Java Network Programming», Эллиот Расти Гарольд (Elliotte Rusty Harold)

В книге детально рассматриваются сетевые прикладные программные интерфейсы Java.

«Java Security», Скотт Оукс (Scott Oaks)

В данной книге детально рассматриваются механизмы управления доступом, а также описываются механизмы проверки достоверности цифровых подписей и дайджестов сообщений.

«Java Cryptography», Джонатан Кнадсен (Jonathan Knudsen)

Всестороннее рассмотрение криптографических расширений Java, пакетов `javax.crypto.*` и всего того, что необходимо знать о криптографии в Java.

«Developing Java Beans», Роберт Ингландер (Robert Englander)

Полное руководство по написанию компонентов, работающих с прикладным программным интерфейсом компонентов Java (JavaBeans API).

Программные ресурсы Java в Интернете

Эта книга является пособием для быстрого доступа к часто используемой информации. Она не содержит, да и не может содержать всей информации о Java. В дополнение к уже перечисленным выше книгам далее представлены еще несколько ценных (и бесплатных) электронных источников информации по Java-программированию.

Главный сайт компании Sun, посвященный Java: *http://java.sun.com/*. Веб-сайт для Java-разработчиков: *http://developer.java.sun.com/*. Большая часть информации, представленной на этом сайте, защищена паролем. Чтобы получить к ней доступ, необходимо зарегистрироваться (бесплатно).

Компания Sun распространяет в электронном виде документацию для всех классов и методов Java (HTML-формат *javadoc*). Хотя искать необходимые данные в этой документации довольно неудобно, а информация местами неточная либо устаревшая, все же она станет отличным отправным пунктом в поиске более подробных сведений о

каком-либо отдельном пакете, классе, методе или поле Java. Если вы еще не обзавелись файлами *javadoc*, то последнюю доступную версию можно найти по ссылке на сайте *http://java.sun.com/docs/*. Кроме того, компания Sun распространяет в Интернете свои высококачественные учебные пособия по Java (Java Tutorial). Вы сможете просмотреть и загрузить их с сайта *http://java.sun.com/docs/books/tutorial/*.

Если вы хотите обсудить Java (на английском) в сети Usenet, обратите внимание на сетевую конференцию *comp.lang.java.programmer* и связанные с ней конференции *comp.lang.java.**. По адресу *http://www.afu.com/javafaq.htm* вы сможете найти исчерпывающие ответы на часто задаваемые вопросы (FAQ). Этот документ составляется Питером ван дер Линденом.

И наконец, не следует забывать о веб-сайте O'Reilly, посвященном Java, – *http://java.oreilly.com*. Здесь публикуются новости и комментарии по Java. В состав сети O'Reilly (*www.oreillynet.com*) входит сайт *onjava.com*, посвященный Enterprise Java.

Примеры в Интернете

Примеры из рассматриваемой нами книги доступны в сети Интернет. Их можно загрузить с домашней страницы книги, расположенной по адресу *http://www.oreilly.com/catalog/javanut4/*. Также на этом сайте публикуются важные замечания по книге и список найденных опечаток.

Типографские соглашения

В данной книге приняты следующие правила оформления:

Курсив

Используется для обозначения важных моментов в тексте, выделения терминов при их первом применении, а также для выделения команд, адресов электронной почты, веб- и FTP-сайтов, имен каталогов, файлов и названий сетевых конференций.

`Моноширинный шрифт`

Применяется для имен классов, констант, методов, пакетов и ключевых слов Java, а также для элементов XML, тегов и команд SQL.

`Моноширинный курсив`

Используется для имен аргументов функций, переменных, а также для выделения комментариев и обозначения элемента, подлежащего замене на фактическое значение в вашей программе.

`Моноширинный полужирный шрифт`

Используется в примерах программ и их фрагментах для выделения ключевых слов Java, имен классов, методов, полей, свойств и конструкторов.

Вопросы и комментарии

Свои комментарии и вопросы по данной книге направляйте издателю по адресу:

O'Reilly & Associates, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

(800) 998-9938 (для Соединенных Штатов и Канады)

(707) 829-0515 (международный либо местный)
(707) 829-1014 (факс)

Существует веб-страница, посвященная данной книге, которая содержит список опечаток, примеры и другую дополнительную информацию. Страница расположена по адресу:

http://www.oreilly.com/catalog/javanut4/

Задать технические вопросы либо прислать комментарии по этой книге можно по адресу:

bookquestions@oreilly.com

Более подробно о книгах, конференциях, центрах ресурсов и Сети O'Reilly можно узнать, посетив веб-сайт O'Reilly:

http://www.oreilly.com/

Как формируется справочник

Для самых любопытных читателей в этом разделе кратко объясняется процесс подготовки справочного материала для книги «Java. Справочник».

Моя система подготовки материала для справочного раздела эволюционировала по мере развития Java. Настоящая система является частью большой системы просмотра коммерческой документации, которую я сейчас разрабатываю (для получения более подробной информации посетите сайт *http://www.davidflanagan.com/Jude/*). Программа работает в два этапа: на первом этапе собирается и систематизируется информация по прикладным программным интерфейсам, а на втором эта информация выводится в виде глав справочника.

Первый этап начинается с прочтения файлов для всех классов и интерфейсов, подлежащих документированию. Практически вся информация по прикладным программным интерфейсам в справочнике берется из файлов классов. Главное исключение составляют имена аргументов методов, которые не сохраняются в файлах классов. Имена аргументов добываются путем синтаксического анализа исходного файла Java для каждого класса и интерфейса. Если исходные файлы недоступны, я получаю имена аргументов методов путем синтаксического анализа документации по прикладному программному интерфейсу, сгенерированной утилитой *javadoc*. Для получения информации по прикладному программному интерфейсу из исходных файлов и файлов *javadoc* я использую парсеры (parsers). Они созданы при помощи генератора парсеров Antlr, разработанного Терренсом Парром из Института Магеланг (Magelang Institute). (Более подробную информацию об этом мощном инструментальном программном средстве можно найти по адресу *http://www.antlr.org/*.)

Когда после прочтения файлов классов, исходных файлов и файлов *javadoc* получена информация по API, программа тратит некоторое время на создание и сортировку перекрестных ссылок. Затем вся информация по API сохраняется в одном большом файле данных.

На втором этапе информация считывается из файла данных и выводится в виде глав справочника посредством специализированного XML-документа. После того как информация выведена в формате XML, я передаю ее производственной команде компании O'Reilly. Там ее обрабатывают и преобразовывают в исходный код для troff. Исходный troff-код обрабатывается с помощью программы *groff* фонда GNU (*ftp://*

ftp.gnu.org/gnu/groff/) и специализированного набора troff-макросов. В результате получается текст на языке PostScript, который выводится на принтер.

Благодарности

Много людей помогали мне в создании этой книги, и я всем им признателен. Я в долгу перед множеством читателей первых трех изданий, приславших свои комментарии, предложения, сообщения об ошибках и похвальные отзывы. Множество песчинок их общего вклада рассеяны по всей книге. Я хочу извиниться перед теми читателями, чьи ценные предложения не вошли в данное издание.

Редактором первых трех изданий книги была Паула Фергюсон (Paula Ferguson), мой друг и коллега. Книга стала более солидной, понятной и полезной благодаря тому, Паула внимательно прочла текст и дала ценные советы. Редакторские обязанности увели Паулу от книг по Java к книгам по веб-программированию, и четвертое издание книги вышло под редакцией Боба Экстэйна (Bob Eckstein), внимательного редактора с потрясающим чувством юмора.

Рецензию на новый материал для этого издания давала группа специалистов компании Sun. Очень часто случалось так, что они рецензировали именно те части прикладного программного интерфейса, создателями которых они являлись. Мне очень повезло, что я смог обратиться за рецензией «прямо к первоисточнику». Я очень признателен этим разработчикам. Несмотря на свой загруженный график они нашли время прочесть и прокомментировать мои наброски. Я перечислю рецензентов в алфавитном порядке:

- Джош Блок (Josh Bloch), автор потрясающей книги «*Effective Java Programming Language Guide*», рецензировал новый материал по утверждениям и прикладному программному интерфейсу для работы с настройками пользователя.
- Грэхэм Гамильтон (Graham Hamilton) рецензировал материал по прикладному программному интерфейсу протоколирования.
- Джонатан Кнадсен (Jonathan Knudsen) рецензировал материалы по JSSE и маршрутам сертификации (он также является автором книг O'Reilly).
- Чарли Лай (Charlie Lai) рецензировал материалы по JAAS.
- Рэм Марти (Ram Marti) рецензировал материалы по JGSS.
- Филипп Милн (Philip Milne), прежде работавший в Sun, а теперь являющийся сотрудником компании Dresdner Kleinwort Wasserstein, рецензировал материалы по механизму постоянства Java-компонентов.
- Марк Рэйнхольд (Mark Reinhold) рецензировал материалы по `java.nio`. Марк заслуживает особой благодарности за рецензирование второго, третьего и четвертого издания книги.
- Андреас Стербенц (Andreas Sterbenz) и Брэд Вэтмор (Brad Wetmore) рецензировали материалы по JSSE.

В дополнение к уже перечисленным рецензентам из Sun следует упомянуть Рона Хитченса (Ron Hitchens), рецензировавшего мой новый материал по вводу/выводу, и моего редактора, Боба Экстайна, который проделал двойную работу в качестве технического рецензента по XML. Я искренне признателен каждому из перечисленных здесь людей за их кропотливо проделанную работу. Конечно, любые ошибки в книге допущены только мной.

Свою неоспоримую лепту в создание третьего издания внесли рецензенты, хорошо разбирающиеся в Java-платформе. Джошуа Блок (Joshua Bloch), один из ведущих авторов структуры Java-коллекций, рецензировал описания классов и интерфейсов коллекций. Джошуа также помог исследовать классы Java 1.3 `Timer` и `TimerTask`. Марк Рэйнхольд, создатель пакета `java.lang.ref`, объяснил его структуру и отрецензировал созданную мной документацию, посвященную этому пакету. Скотт Оукс рецензировал мои описания классов и интерфейсов защиты и криптографии Java. Джошуа, Марк и Скотт – специалисты компании Sun Microsystems, и я очень признателен им за уделенное мне внимание. Кроме них документацию по пакету `javax.crypto` и его подпакетам рецензировал Йон Ивз. Йон работал над технологическим применением криптографических расширений Java (об этом можно прочесть по адресу *http://www.aha.net.au/*). Его предложения оказались очень полезны. В настоящее время Йон работает в компании Fluent Technologies (*http://www.fluent.com.au/*) консультантом по Java и электронной коммерции. И наконец, в первой главе использованы комментарии от рецензентов, не знакомых с платформой Java. Кристина Берн (Christina Byrne) дала рецензию с точки зрения неопытного программиста, а Джудита Берн (Judita Byrne) из Virginia Power высказала свое мнение как квалифицированный COBOL-программист.

Для второго издания Джон Зуковски (John Zukowski) отрецензировал мой справочник по Java 1.1 AWT, а Джорж Риз (George Reese) – большую часть остального нового материала. Второе издание получило свое благословение от «команды-мечты» технических рецензентов компании Sun. Джон Роуз (John Rose), автор спецификации внутренних классов Java, рецензировал главу, посвященную внутренним классам. Марк Рэйнхольд, автор новых классов символьных потоков в `java.io`, рецензировал мою документацию по этим классам. Накул Сарайа (Nakul Saraiya), дизайнер нового прикладного программного интерфейса Java Reflection, рецензировал документацию по пакету `java.lang.reflect`. Я глубоко признателен этим разработчикам и архитекторам; благодаря их усилиям книга обрела достоверный и исчерпывающий материал.

Майк Лукидес (Mike Loukides) осуществлял общее руководство выпуском первого издания книги. Эрик Рэймонд (Eric Raymond) и Трой Даунинг (Troy Downing) рецензировали первое издание – они помогли мне увидеть ошибки и упущения и дали дельные советы по улучшению книги.

Как обычно, производственная команда O'Reilly с успехом справилась с задачей создания книги из предоставленных мною электронных файлов. Выражаю всем сотрудникам рабочей группы свою признательность.

Как всегда, признаюсь в любви к Кристи.

Дэвид Флэнаган (David Flanagan)
http://www.davidflanagan.com/
январь 2002

Глава 10

java.io

Пакет java.io

java 1.0

Пакет java.io является достаточно большим, однако основная часть его классов подпадает под иерархию с четкой структурой.

Большая часть пакета состоит из байтовых потоков (подклассов InputStream или OutputStream) и потоков символов (подклассов Reader или Writer). Каждый из этих подтипов потока имеет конкретное предназначение, поэтому несмотря на свой размер java.io является понятным и удобным пакетом. В Java 1.4 пакет java.io дополнен новым API ввода/вывода, который определяется в пакете java.nio и его подпакетах. Пакет java.nio является абсолютно новым, хотя в какой-то мере он совместим с классами из пакета java.io. Пакет java.nio был разработан для обеспечения высокоскоростного ввода/вывода, в особенности для использования на серверах. Он имеет низкоуровневый программный интерфейс приложения по сравнению с данным пакетом. Средства ввода/вывода java.io по-прежнему достаточно адекватны для большей части ввода/вывода, которая требуется обычным клиентским приложениям.

Перед тем как приступить к рассмотрению классов потоков, которые составляют основную часть этого пакета, рассмотрим те важные классы, которые непосредственно не работают с потоками. File представляет файл или имя каталога независимо от системы. Он предоставляет методы для чтения содержимого каталогов, доступа к атрибутам файлов, переименования и удаления файлов. FilenameFilter – это интерфейс, который определяет метод, принимающий или отклоняющий конкретные имена файлов. Он используется File для того, чтобы указать те типы файлов, которые следует включать в перечни файлов каталога. RandomAccessFile позволяет вам читать или записывать что-либо в произвольные участки файла. Тем не менее зачастую предпочтительнее использовать последовательный доступ к файлу. Кроме того, вам следует использовать один из классов потоков.

InputStream и OutputStream являются абстрактными классами, которые определяют методы для чтения и записи байтов. Их подклассы позволяют считывать и записывать байты в разнообразные приемники и источники. FileInputStream и FileOutputStream считывают данные из файлов и записывают их в файлы. ByteArrayInputStream и ByteArrayOutputStream считывают данные из массива байтов в памяти и записывают их в массив байтов в памяти. PipedInputStream считывает байты с PipedOutputStream, а PipedOutputStream записывает байты в PipedInputStream. Эти классы работают

вместе для того, чтобы реализовать *канал* (pipe), связывающий разные потоки выполнения (threads). FilterInputStream и FilterOutputStream являются особенными; они фильтруют входящие и исходящие байты. Когда вы создаете FilterInputStream, то вы указываете InputStream, подлежащий фильтрации.

Когда вы вызываете метод read(), принадлежащий FilterInputStream, он вызывает метод read() своего InputStream, обрабатывает байты, которые он считывает, и возвращает отфильтрованные байты. Когда вы аналогичным образом создаете FilterOutputStream, то вы указываете OutputStream, подлежащий фильтрации. Вызов метода write(), принадлежащего FilterOutputStream, приводит к тому, что он каким-либо образом обрабатывает ваши байты, а затем передает эти отфильтрованные байты методу write() своего OutputStream.

FilterInputStream и FilterOutputStream сами не выполняют какой-либо фильтрации; это делают их подклассы.

BufferedInputStream и BufferedOutputStream являются отфильтрованными потоками, которые обеспечивают буферизацию ввода и вывода и могут увеличить эффективность операций ввода/вывода. DataInputStream считывает необработанные байты из потока и интерпретирует их в различные двоичные форматы. У него есть различные методы для чтения примитивных типов данных Java в их стандартных двоичных форматах. DataOutputStream позволяет записывать примитивные типы данных Java в двоичном формате.

Описанные потоки байтов дополнены аналогичным набором потоков ввода/вывода символов. Reader является родительским классом всех потоков ввода символов. Writer является родительским классом всех потоков вывода символов. Большая часть потоков Reader и Writer имеют явные аналоги типа байт-потока. Поток BufferedReader широко используется; он обеспечивает буферизацию, улучшающую эффективность, и предоставляет метод readLine(), позволяющий читать строку текста за раз. PrintWriter является другим распространенным потоком; его методы выводят текстовое представление любого примитивного Java-типа или любого объекта (через метод объекта toString()). Классы ObjectInputStream и ObjectOutputStream – особенные. Эти классы байт-потоков применяются для сериализации и десериализации внутреннего состояния объектов с целью хранения или межпроцессного взаимодействия.

Интерфейсы

```
public interface DataInput;
public interface DataOutput;
public interface Externalizable extends Serializable;
public interface FileFilter;
public interface FilenameFilter;
public interface ObjectInput extends DataInput;
public interface ObjectInputValidation;
public interface ObjectOutput extends DataOutput;
public interface ObjectStreamConstants;
public interface Serializable;
```

Классы

```
public class File implements Comparable, Serializable;
public final class FileDescriptor;
public final class FilePermission extends java.security.Permission implements Serializable;
public abstract class InputStream;
   └ public class ByteArrayInputStream extends InputStream;
   └ public class FileInputStream extends InputStream;
```

```
   └ public class FilterInputStream extends InputStream;
       └ public class BufferedInputStream extends FilterInputStream;
       └ public class DataInputStream extends FilterInputStream implements DataInput;
       └ public class LineNumberInputStream extends FilterInputStream;
       └ public class PushbackInputStream extends FilterInputStream;
   └ public class ObjectInputStream extends InputStream
     implements ObjectInput, ObjectStreamConstants;
    └ public class PipedInputStream extends InputStream;
   └ public class SequenceInputStream extends InputStream;
   └ public class StringBufferInputStream extends InputStream;
public abstract static class ObjectInputStream.GetField;
public abstract static class ObjectOutputStream.PutField;
public class ObjectStreamClass implements Serializable;
public class ObjectStreamField implements Comparable;
public abstract class OutputStream;
    └ public class ByteArrayOutputStream extends OutputStream;
   └ public class FileOutputStream extends OutputStream;
   └ public class FilterOutputStream extends OutputStream;
       └ public class BufferedOutputStream extends FilterOutputStream;
      └ public class DataOutputStream extends FilterOutputStream implements DataOutput;
      └ public class PrintStream extends FilterOutputStream;
   └ public class ObjectOutputStream extends OutputStream
                                       implements ObjectOutput, ObjectStreamConstants;
   └ public class PipedOutputStream extends OutputStream;
public class RandomAccessFile implements DataInput, DataOutput;
public abstract class Reader;
   └ public class BufferedReader extends Reader;
       └ public class LineNumberReader extends BufferedReader;
   └ public class CharArrayReader extends Reader;
    └ public abstract class FilterReader extends Reader;
       └ public class PushbackReader extends FilterReader;
   └ public class InputStreamReader extends Reader;
       └ public class FileReader extends InputStreamReader;
   └ public class PipedReader extends Reader;
    └ public class StringReader extends Reader;
public final class SerializablePermission extends java.security.BasicPermission;
public class StreamTokenizer;
public abstract class Writer;
   └ public class BufferedWriter extends Writer;
   └ public class CharArrayWriter extends Writer;
   └ public abstract class FilterWriter extends Writer;
   └ public class OutputStreamWriter extends Writer;
      └ public class FileWriter extends OutputStreamWriter;
   └ public class PipedWriter extends Writer;
   └ public class PrintWriter extends Writer;
   └ public class StringWriter extends Writer;
```

Исключения

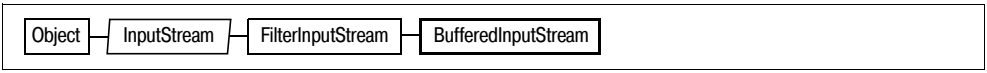
```
public class IOException extends Exception;
   └ public class CharConversionException extends IOException;
   └ public class EOFException extends IOException;
   └ public class FileNotFoundException extends IOException;
   └ public class InterruptedIOException extends IOException;
   └ public abstract class ObjectStreamException extends IOException;
     └ public class InvalidClassException extends ObjectStreamException;
     └ public class InvalidObjectException extends ObjectStreamException;
```

```
    └ public class NotActiveException extends ObjectStreamException;
    └ public class NotSerializableException extends ObjectStreamException;
    └ public class OptionalDataException extends ObjectStreamException;
    └ public class StreamCorruptedException extends ObjectStreamException;
    └ public class WriteAbortedException extends ObjectStreamException;
public class SyncFailedException extends IOException;
public class UnsupportedEncodingException extends IOException;
public class UTFDataFormatException extends IOException;
```

BufferedInputStream Java 1.0

java.io

Этот класс является потомком класса `FilterInputStream`, который обеспечивает буферизацию входных данных; эффективность повышается благодаря считыванию большого количества данных и их хранению во внутреннем буфере. Когда запрашиваются данные, они извлекаются из буфера. Таким образом, большинство запросов на считывание данных на самом деле не должны обращаться за данными к диску, сети или другому медленному источнику. Создайте `BufferedInputStream`, указав в запросе к конструктору на конкретный `InputStream`, подлежащий буферизации. См. также `BufferedReader`.



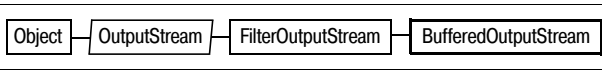

```
public class BufferedInputStream extends FilterInputStream {
// Открытые конструкторы
   public BufferedInputStream(java.io.InputStream in);
   public BufferedInputStream(java.io.InputStream in, int size);
// Открытые методы, замещающие методы класса FilterInputStream
   public int available() throws IOException;                          // синхронизирован
1.2 public void close() throws IOException;
   public void mark(int readlimit);                                    // синхронизирован
   public boolean markSupported();                                     // константа
   public int read() throws IOException;                               // синхронизирован
   public int read(byte[] b, int off, int len) throws IOException;     // синхронизирован
   public void reset() throws IOException;                             // синхронизирован
   public long skip(long n) throws IOException;                        // синхронизирован
// Защищенные поля экземпляра
   protected byte[] buf;
   protected int count;
   protected int marklimit;
   protected int markpos;
   protected int pos;
}
```

BufferedOutputStream Java 1.0

java.io

Этот класс является потомком класса `FilterOutputStream`, который обеспечивает буферизацию выходных данных; эффективность вывода увеличивается за счет хранения значений, подлежащих записи, в буфере. Фактическая запись этих значений происходит только в том случае, если переполняется буфер или вызывается метод `flush()`.

Создайте BufferedOutputStream, указав в запросе к конструктору на конкретный OutputStream, подлежащий буферизации. См. также BufferedWriter.



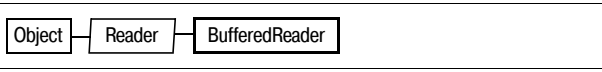

```
public class BufferedOutputStream extends FilterOutputStream {
// Открытые конструкторы
   public BufferedOutputStream(java.io.OutputStream out);
   public BufferedOutputStream(java.io.OutputStream out, int size);
// Открытые методы, замещающие методы класса FilterOutputStream
   public void flush() throws IOException;                                  // синхронизирован
   public void write(int b) throws IOException;                             // синхронизирован
   public void write(byte[] b, int off, int len) throws IOException;        // синхронизирован
// Защищенные поля экземпляра
   protected byte[] buf;
   protected int count;
}
```

BufferedReader

Java 1.1

java.io

Этот класс буферизует поток входных символов и таким образом улучшает эффективность их ввода. Вы создаете BufferedReader путем указания на какой-либо другой поток ввода символов, подлежащих буферизации. На этом этапе вы также можете задать размер буфера, хотя размер по умолчанию обычно вполне подходит. Как правило, такая буферизация применяется с FileReader или InputStreamReader. BufferedReader определяет стандартный набор методов Reader и предоставляет метод readLine(), который считывает строку текста (за исключением символа-разделителя строк) и возвращает ее как String. BufferedReader — это аналог BufferedInputStream, представленный как поток символов. Он также является заменой устаревшему методу readLine(), который представлен в DataInputStream и не способен корректно переводить байты в символы.




```
public class BufferedReader extends Reader {
// Открытые конструкторы
   public BufferedReader(Reader in);
   public BufferedReader(Reader in, int sz);
// Открытые методы экземпляра
   public String readLine() throws IOException;
// Открытые методы, замещающие методы класса Reader
   public void close() throws IOException;
   public void mark(int readAheadLimit) throws IOException;
   public boolean markSupported();                                          // константа
   public int read() throws IOException;
   public int read(char[] cbuf, int off, int len) throws IOException;
   public boolean ready() throws IOException;
   public void reset() throws IOException;
   public long skip(long n) throws IOException;
}
```

Подклассы: LineNumberReader

BufferedWriter Java 1.1

java.io

Этот класс буферизует поток вывода символов, улучшая эффективность вывода путем объединения небольших запросов на запись в один более крупный запрос. Вы создаете BufferedWriter путем указания на какой-либо другой поток вывода символов, которому он посылает свой буферизованный и объединенный вывод. (На этом этапе вы можете задать размер буфера, хотя размер по умолчанию обычно вполне приемлем.) Как правило, такая буферизация применяется с FileWriter или OutputStreamWriter. BufferedWriter определяет стандартные методы write(), flush() и close(), определяемые всеми потоками вывода, но он добавляет к потоку метод newLine(), который выводит разделитель строк, зависящий от конкретной платформы (обычно символ новой строки, символ возврата каретки или оба эти символа). BufferedWriter – это аналог BufferedOutputStream, представленный как поток символов.


```
public class BufferedWriter extends Writer {
// Открытые конструкторы
   public BufferedWriter(Writer out);
   public BufferedWriter(Writer out, int sz);
// Открытые методы экземпляра
   public void newLine() throws IOException;
// Открытые методы, замещающие методы класса Writer
   public void close() throws IOException;
   public void flush() throws IOException;
   public void write(int c) throws IOException;
   public void write(char[] cbuf, int off, int len) throws IOException;
   public void write(String s, int off, int len) throws IOException;
}
```

ByteArrayInputStream Java 1.0

java.io

Этот класс является подклассом InputStream, в котором входные данные берутся из указанного массива значений байтов. Это полезно в том случае, если нужно прочесть данные, находящиеся в памяти, как будто они поступают из файла, канала или сокета. Обратите внимание, что указанный массив байтов не копируется, когда создается ByteArrayInputStream. См. также CharArrayReader.


```
public class ByteArrayInputStream extends java.io.InputStream {
// Открытые конструкторы
   public ByteArrayInputStream(byte[] buf);
   public ByteArrayInputStream(byte[] buf, int offset, int length);
// Открытые методы, замещающие методы класса InputStream
   public int available();                                    // синхронизирован
1.2 public void close() throws IOException;                   // пустой
1.1 public void mark(int readAheadLimit);
```

```
1.1 public boolean markSupported();                                    // константа
   public int read();                                                  // синхронизирован
   public int read(byte[] b, int off, int len);                        // синхронизирован
   public void reset();                                                // синхронизирован
   public long skip(long n);                                           // синхронизирован
// Защищенные поля экземпляра
   protected byte[] buf;
   protected int count;
1.1 protected int mark;
   protected int pos;
}
```

ByteArrayOutputStream

java 1.0

java.io

Этот класс является подклассом `OutputStream`, в котором выходные данные хранятся во внутреннем массиве байтов. Внутренний массив увеличивается по мере необходимости и может быть извлечен при помощи методов `toByteArray()` или `toString()`. Метод `reset()` удаляет все данные, которые в настоящий момент хранятся во внутреннем массиве, и начинает записывать данные с самого начала. См. также `CharArrayWriter`.


```
public class ByteArrayOutputStream extends java.io.OutputStream {
// Открытые конструкторы
   public ByteArrayOutputStream();
   public ByteArrayOutputStream(int size);
// Открытые методы экземпляра
   public void reset();                                                // синхронизирован
   public int size();
   public byte[] toByteArray();                                        // синхронизирован
1.1 public String toString(String enc) throws UnsupportedEncodingException;
   public void writeTo(java.io.OutputStream out) throws IOException;   // синхронизирован
// Открытые методы, замещающие методы класса OutputStream
1.2 public void close() throws IOException;                            // пустой
   public void write(int b);                                           // синхронизирован
   public void write(byte[] b, int off, int len);                      // синхронизирован
// ПОткрытые методы, замещающие методы класса Object
   public String toString();
// Защищенные поля экземпляра
   protected byte[] buf;
   protected int count;
// Устаревшие открытые методы
# public String toString(int hibyte);
}
```

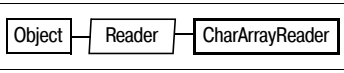
CharArrayReader

Java 1.1

java.io

Этот класс является потоком ввода символов, который использует массив символов в качестве источника тех символов, которые он возвращает. Вы создаете `CharArrayReader` путем указания массива символов (или части массива), из которого он должен

осуществлять чтение. CharArrayReader определяет обычные методы Reader и поддерживает методы mark() и reset(). Обратите внимание на то, что массив символов, передаваемый конструктору CharArrayReader(), не копируется. Это означает, что те изменения, которые вы внесете в элементы массива после создания входящего потока, повлияют на значения, считываемые из массива. CharArrayReader — это аналог ByteArrayInputStream, представленный как массив символов; он подобен StringReader.



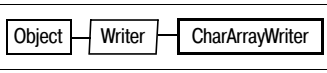

```
public class CharArrayReader extends Reader {
// Открытые конструкторы
   public CharArrayReader(char[] buf);
   public CharArrayReader(char[] buf, int offset, int length);
// Открытые методы, замещающие методы класса Reader
   public void close();
   public void mark(int readAheadLimit) throws IOException;
   public boolean markSupported();  // константа
   public int read() throws IOException;
   public int read(char[] b, int off, int len) throws IOException;
   public boolean ready() throws IOException;
   public void reset() throws IOException;
   public long skip(long n) throws IOException;
// Защищенные поля экземпляра
   protected char[] buf;
   protected int count;
   protected int markedPos;
   protected int pos;
}
```

CharArrayWriter

Java 1.1

java.io

Этот класс является потоком вывода символов, который использует внутренний массив символов в качестве приемника возвращаемых символов. Когда вы создаете CharArrayWriter, то вы можете задать начальный размер массива символов, однако не указывайте массив символов; этот массив контролируется самим CharArrayWriter и увеличивается по мере необходимости, чтобы вместить все символы. Методы toString() и toCharArray() возвращают копию всех символов, записанных в поток, в виде строки и массива символов соответственно. CharArrayWriter определяет стандартные методы write(), flush() и close(), которые определяются всеми подклассами Writer. Он также определяет ряд других полезных методов. size() возвращает количество символов, которые были записаны в поток. reset() приводит поток к его изначальному состоянию (с пустым массивом символов); это более эффективно, чем создание нового CharArrayWriter. И наконец, writeTo() записывает содержимое внутреннего массива символов в какой-либо другой поток символов. CharArrayWriter — это аналог ByteArrayOutputStream, представленный как поток символов; он подобен StringWriter.



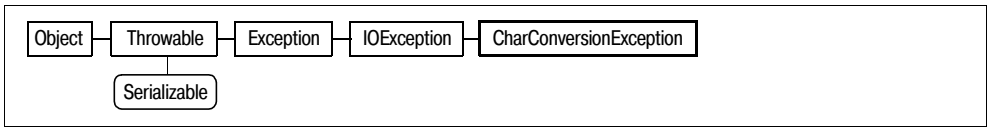
```
public class CharArrayWriter extends Writer {
// Открытые конструкторы
   public CharArrayWriter();
   public CharArrayWriter(int initialSize);
// Открытые методы экземпляра
   public void reset();
   public int size();
   public char[] toCharArray();
   public void writeTo(Writer out) throws IOException;
// Открытые методы, замещающие методы класса Writer
   public void close();  // пустой
   public void flush();  // пустой
   public void write(int c);
   public void write(char[] c, int off, int len);
   public void write(String str, int off, int len);
// Открытые методы, замещающие методы класса Object
   public String toString();
// Защищенные поля экземпляра
   protected char[] buf;
   protected int count;
}
```

CharConversionException

Java 1.1

java.io ***сериализуемое, проверяемое***

CharConversionException обозначает ошибку, которая произошла во время перевода байтов в символы или наоборот.




```
public class CharConversionException extends IOException {
// Открытые конструкторы
   public CharConversionException();
   public CharConversionException(String s);
}
```

DataInput

Java 1.0

java.io

Этот интерфейс определяет методы, необходимые потокам, которые могут считывать примитивные типы данных Java в машинно-независимом двоичном формате. Его реализуют классы DataInputStream и RandomAccessFile. Для получения более подробной информации о методах обратитесь к описанию DataInputStream.

```
public interface DataInput {
// Открытые методы экземпляра
   public abstract boolean readBoolean() throws IOException;
   public abstract byte readByte() throws IOException;
   public abstract char readChar() throws IOException;
   public abstract double readDouble() throws IOException;
```

```
    public abstract float readFloat() throws IOException;
    public abstract void readFully(byte[] b) throws IOException;
    public abstract void readFully(byte[] b, int off, int len) throws IOException;
    public abstract int readInt() throws IOException;
    public abstract String readLine() throws IOException;
    public abstract long readLong() throws IOException;
    public abstract short readShort() throws IOException;
    public abstract int readUnsignedByte() throws IOException;
    public abstract int readUnsignedShort() throws IOException;
    public abstract String readUTF() throws IOException;
    public abstract int skipBytes(int n) throws IOException;
}
```

Реализации: `java.io.DataInputStream`, `ObjectInput`, `RandomAccessFile`, `javax.imageio.stream.ImageInputStream`

Передается методам: `java.io.DataInputStream.readUTF()`, `java.rmi.server.UID.read()`

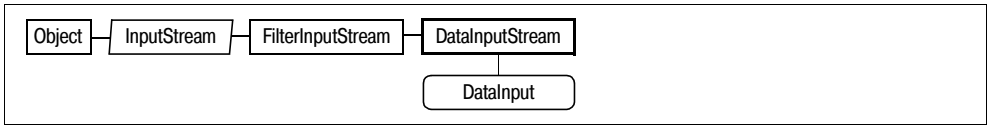
DataInputStream

Java 1.0

java.io

Этот класс является видом `FilterInputStream`. Он позволяет читать бинарные представления примитивных типов данных Java вне зависимости от платформы. Создайте `DataInputStream`, указав при вызове конструктора на конкретный `InputStream`, подлежащий фильтрации. `DataInputStream` считывает только примитивные Java-типы; используйте `ObjectInputStream` для считывания значений объектов.

Большинство методов считывают и возвращают из потока одиночную переменную примитивного Java-типа (в двоичном формате). Методы `readUnsignedByte()` и `readUnsignedShort()` считывают беззнаковые значения и возвращают их как значения `int`, поскольку беззнаковые типы `byte` и `short` не поддерживаются в Java. `read()` считывает данные в массив байтов, блокируя выполнение программы до тех пор, пока не будут доступны какие-нибудь данные. В отличие от него, `readFully()` считывает данные в массив байтов, но блокирует выполнение до получения всех данных, которые были запрошены. `skipBytes()` осуществляет блокировку до тех пор, пока не будет считано и пропущено указанное количество байтов. `readLine()` считывает символы из потока до тех пор, пока он не обнаружит символ новой строки, возврат каретки или пару, состоящую из символа новой строки и возврата каретки. Возвращенная строка не завершается символами новой строки или возврата каретки. Этот метод признан устаревшим Java 1.1; см. `BufferedReader` на предмет альтернативного решения. `readUTF()` считывает строку текста в Unicode, которая закодирована с помощью видоизмененной версии преобразовательного формата UTF-8. UTF-8 – это кодировка, совместимая с Американским стандартным кодом для обмена информацией (ASCII). Она часто применяется для передачи и хранения текста в Unicode. Этот класс использует модифицированную кодировку UTF-8, которая не содержит вложенных нуль-символов.




```
public class DataInputStream extends FilterInputStream implements DataInput {
// Открытые конструкторы
```

```
   public DataInputStream(java.io.InputStream in);
// Открытые методы класса
   public static final String readUTF(DataInput in) throws IOException;
// Методы, реализующие DataInput
   public final boolean readBoolean() throws IOException;
   public final byte readByte() throws IOException;
   public final char readChar() throws IOException;
   public final double readDouble() throws IOException;
   public final float readFloat() throws IOException;
   public final void readFully(byte[] b) throws IOException;
   public final void readFully(byte[] b, int off, int len) throws IOException;
   public final int readInt() throws IOException;
   public final long readLong() throws IOException;
   public final short readShort() throws IOException;
   public final int readUnsignedByte() throws IOException;
   public final int readUnsignedShort() throws IOException;
   public final String readUTF() throws IOException;
   public final int skipBytes(int n) throws IOException;
// Открытые методы, замещающие методы класса FilterInputStream
   public final int read(byte[] b) throws IOException;
   public final int read(byte[] b, int off, int len) throws IOException;
// Устаревшие открытые методы
# public final String readLine() throws IOException;                    // Реализует:DataInput
}
```

Передается методам: javax.swing.text.html.parser.DTD.read()

DataOutput Java 1.0

java.io

Этот интерфейс определяет методы, требующиеся для потоков, которые могут записывать примитивные типы данных Java в машинно-независимом двоичном формате. Он реализуется классами DataOutputStream и RandomAccessFile. Более детальная информация о методах представлена в описании DataOutputStream.

```
public interface DataOutput {
// Открытые методы экземпляра
   public abstract void write(byte[] b) throws IOException;
   public abstract void write(int b) throws IOException;
   public abstract void write(byte[] b, int off, int len) throws IOException;
   public abstract void writeBoolean(boolean v) throws IOException;
   public abstract void writeByte(int v) throws IOException;
   public abstract void writeBytes(String s) throws IOException;
   public abstract void writeChar(int v) throws IOException;
   public abstract void writeChars(String s) throws IOException;
   public abstract void writeDouble(double v) throws IOException;
   public abstract void writeFloat(float v) throws IOException;
   public abstract void writeInt(int v) throws IOException;
   public abstract void writeLong(long v) throws IOException;
   public abstract void writeShort(int v) throws IOException;
   public abstract void writeUTF(String str) throws IOException;
}
```

Реализации: java.io.DataOutputStream, ObjectOutput, RandomAccessFile, javax.imageio.stream.ImageOutputStream

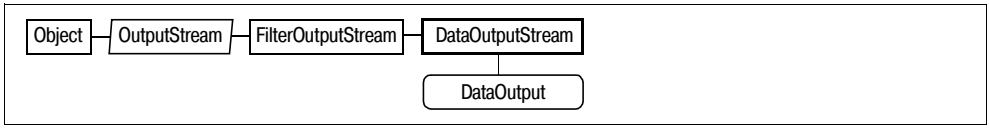
Передается методам: java.rmi.server.UID.write()

DataOutputStream Java 1.0

java.io

Этот класс является подклассом FilterOutputStream. Он позволяет записывать примитивные типы данных Java в машинно-независимом двоичном формате. Создайте DataOutputStream, указав в запросе к конструктору на конкретный OutputStream, подлежащий фильтрации. DataOutputStream имеет методы, которые выводят только примитивные типы; для вывода значений объектов применяйте ObjectOutputStream.

Многие из методов этого класса записывают одиночный примитивный Java-тип в выходной поток (в двоичном формате). write() записывает один байт, массив или подмассив байтов. flush() выводит любые буферизованные данные. size() возвращает количество байтов, которые были записаны на текущий момент. writeUTF() выводит Java-строку в символах Unicode, используя видоизмененную версию преобразовательного формата UTF-8. UTF-8 – это кодировка, совместимая с Американским стандартным кодом для обмена информацией (ASCII). Она часто применяется для передачи и хранения текста в Unicode. За исключением метода writeUTF(), этот класс применяется для двоичного вывода данных. Текстовый вывод должен производиться при помощи PrintWriter (или PrintStream в Java 1.0).



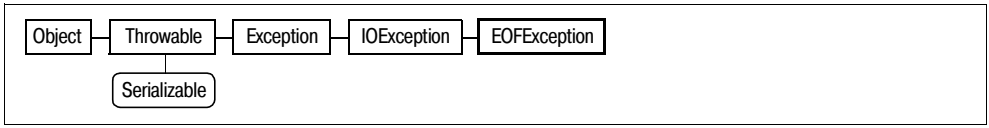

```
public class DataOutputStream extends FilterOutputStream implements DataOutput {
// Открытые конструкторы
   public DataOutputStream(java.io.OutputStream out);
// Открытые методы экземпляра
   public final int size();
// Методы, реализующие DataOutput
   public void write(int b) throws IOException;                              // синхронизирован
   public void write(byte[] b, int off, int len) throws IOException;         // синхронизирован
   public final void writeBoolean(boolean v) throws IOException;
   public final void writeByte(int v) throws IOException;
   public final void writeBytes(String s) throws IOException;
   public final void writeChar(int v) throws IOException;
   public final void writeChars(String s) throws IOException;
   public final void writeDouble(double v) throws IOException;
   public final void writeFloat(float v) throws IOException;
   public final void writeInt(int v) throws IOException;
   public final void writeLong(long v) throws IOException;
   public final void writeShort(int v) throws IOException;
   public final void writeUTF(String str) throws IOException;
// Открытые методы, замещающие методы класса FilterOutputStream
   public void flush() throws IOException;
// Защищенные поля экземпляра
   protected int written;
}
```

EOFException

Java 1.0

java.io *сериализуемое, проверяемое*

Исключение `EOFException` – это `IOException`, которое обозначает конец файла.




```
public class EOFException extends IOException {
// Открытые конструкторы
   public EOFException();
   public EOFException(String s);
}
```

Externalizable

Java 1.1

java.io *сериализуемый*

Этот интерфейс определяет методы, которые должны быть реализованы объектом, желающим иметь полный контроль над процессом своей сериализации. Методы `writeExternal()` и `readExternal()` должны быть реализованы для записи и считывания данных объекта в каком-либо произвольном формате с использованием методов интерфейсов `DataOutput` и `DataInput`. Объекты `Externalizable` должны сериализовать свои собственные поля; кроме того, они отвечают за сериализацию полей родительских классов. Большая часть объектов не нуждается в определении специализированного формата вывода и может использовать интерфейс `Serializable` вместо `Externalizable` для сериализации.


```
public interface Externalizable extends Serializable {
// Открытые методы экземпляра
   public abstract void readExternal(ObjectInput in) throws IOException, ClassNotFoundException;
   public abstract void writeExternal(ObjectOutput out) throws IOException;
}
```

Реализации: `java.awt.datatransfer.DataFlavor`, `java.rmi.server.RemoteRef`

File

Java 1.0

java.io *сериализуемый, сравнимый*

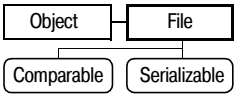
Этот класс поддерживает определение имен файлов и каталогов, которое не зависит от платформы. Кроме того, он предоставляет методы для перечисления файлов в каталоге; для проверки существования, читаемости, записываемости, типа, размера и времени изменения файлов и каталогов; для создания новых каталогов; для переименования файлов и каталогов; для создания и удаления временных файлов и файлов блокировки. Константы, определяемые этим классом, являются символами разделения пути и каталога, которые зависят от платформы. Они доступны как `String` и `char`.

getName() возвращает имя File без имен каталогов. getPath() возвращает полное имя файла, включая имя каталога. getParent() и getParentFile() возвращают каталог, который содержит File; единственное различие между этими методами заключается в том, что один из них возвращает String, а другой – File. isAbsolute() проверяет, задан ли File абсолютным именем. Если нет, то getAbsolutePath() возвращает абсолютное имя файла, созданное путем добавления относительного имени файла к текущему каталогу. getAbsoluteFile() возвращает эквивалентный абсолютный объект File. getCanonicalPath() и getCanonicalFile() похожи: они возвращают абсолютное имя файла или объект File, переведенный в его каноническую форму, зависимую от системы. Такая возможность полезна при сравнении двух объектов File, когда нужно понять, ссылаются ли они на один и тот же файл или каталог. В Java 1.4 и последующих версиях метод toURI() возвращает объект java.net.URI, который применяет схему file: для присваивания имени этому файлу. Преобразование из file в URI можно изменить на прямо противоположное, передав объект file: URI конструктору File().

exists(), canWrite(), canRead(), isFile(), isDirectory() и isHidden() выполняют очевидные тесты с конкретным File. length() возвращает длину файла. lastModified() возвращает время изменения файла, что следует применять только для сравнения с другими временны*ми* характеристиками файла, а не интерпретировать как какой-либо конкретный формат времени). setLastModified() позволяет задавать время модификации; setReadOnly() устанавливает какому-либо файлу или каталогу атрибут «только для чтения».

list() возвращает имена всех тех элементов в каталоге, которые не отвергнуты необязательным FilenameFilter. listFiles() возвращает массив объектов File, которые представляют все элементы в каталоге, не отвергнутые необязательным FilenameFilter или FileFilter. listRoots() возвращает массив объектов File, которые представляют все корневые каталоги в системе. Например, в системах Unix обычно существует только один корень, /. В системах Windows присутствуют разные корни для каждого имени дисковода, например *c:*\, *d:*\ и *e:*\.

mkdir() создает каталог, а mkdirs() создает все каталоги, описанные в объекте File. renameTo() переименовывает файл или каталог; delete() удаляет файл или каталог. До версии Java 1.2 класс File не предоставлял ни одного способа создания файла; эту задачу можно выполнить с помощью FileOutputStream. В Java 1.2 были добавлены два специальных метода для создания файла. Статический метод createTempFile() возвращает объект File, который ссылается на вновь созданный пустой файл с уникальным именем. Имя начинается с заданной приставки, которая должна быть длиной как минимум в три символа и заканчиваться заданным суффиксом. Одна из версий этого метода создает файл в указанном каталоге, а другая создает его во временном каталоге системы. Приложения могут использовать временные файлы в любых целях без опасения перезаписи файлов, принадлежащих другим приложениям. Другой метод создания файла в Java 1.2 – это createNewFile(). Этот метод экземпляра пытается создать новый, пустой файл с именем, указанным объектом File. Если это удается, он возвращает true. Если файл уже существует, то он возвращает false. Метод createNewFile() выполняется атомарно, поэтому он полезен для блокировки файла и других схем взаимного исключения. При работе с createTempFile() или createNewFile() рассмотрите возможность использования deleteOnExit(), чтобы запрашивать удаление файлов, когда виртуальная машина Java корректно завершает работу.




```
public class File implements Comparable, Serializable {
// Открытые конструкторы
   public File(String pathname);
1.4 public File(java.net.URI uri);
   public File(String parent, String child);
   public File(File parent, String child);
// Открытые константы
   public static final String pathSeparator;
   public static final char pathSeparatorChar;
   public static final String separator;
   public static final char separatorChar;
// Открытые методы класса
1.2 public static File createTempFile(String prefix, String suffix) throws IOException;
1.2 public static File createTempFile(String prefix, String suffix, File directory)
       throws IOException;
1.2 public static File[] listRoots();
// Методы доступа к свойствам (по имени свойства)
   public boolean isAbsolute();
1.2 public File getAbsoluteFile();
   public String getAbsolutePath();
1.2 public File getCanonicalFile() throws IOException;
1.1 public String getCanonicalPath() throws IOException;
   public boolean isDirectory();
   public boolean isFile();
1.2 public boolean isHidden();
   public String getName();
   public String getParent();
1.2 public File getParentFile();
   public String getPath();
// Открытые методы экземпляра
   public boolean canRead();
   public boolean canWrite();
1.2 public int compareTo(File pathname);
1.2 public boolean createNewFile() throws IOException;
   public boolean delete();
1.2 public void deleteOnExit();
   public boolean exists();
   public long lastModified();
   public long length();
   public String[] list();
   public String[] list(FilenameFilter filter);
1.2 public File[] listFiles();
1.2 public File[] listFiles(FilenameFilter filter);
1.2 public File[] listFiles(java.io.FileFilter filter);
   public boolean mkdir();
   public boolean mkdirs();
   public boolean renameTo(File dest);
1.2 public boolean setLastModified(long time);
1.2 public boolean setReadOnly();
1.4 public java.net.URI toURI();
1.2 public java.net.URL toURL() throws java.net.MalformedURLException;
```

```
// Методы, реализующие Comparable
1.2 public int compareTo(Object o);
// Открытые методы, замещающие методы класса Object
   public boolean equals(Object obj);
   public int hashCode();
   public String toString();
}
```

Передается методам: Методов слишком много, чтобы их перечислить.

Возвращается методами: Методов слишком много, чтобы их перечислить.

FileDescriptor Java 1.0

java.io

Этот класс является платформо-независимым представлением низкоуровневого дескриптора открытого файла или сокета. Статические переменные `in`, `out` и `err` являются объектами `FileDescriptor`, которые представляют соответственно стандартные входные/выходные потоки и поток сообщений об ошибках. Для создания `FileDescriptor` не существует открытых конструкторов. Вы можете получить его при помощи метода `getFD()` классов `FileInputStream`, `FileOutputStream` или `RandomAccessFile`.

```
public final class FileDescriptor {
// Открытые конструкторы
   public FileDescriptor();
// Открытые константы
   public static final FileDescriptor err;
   public static final FileDescriptor in;
   public static final FileDescriptor out;
// Открытые методы экземпляра
1.1 public void sync() throws SyncFailedException;  // зависит от платформы
   public boolean valid();
}
```

Передается методам: `FileInputStream.FileInputStream()`, `FileOutputStream.FileOutputStream()`, `FileReader.FileReader()`, `FileWriter.FileWriter()`, `SecurityManager.{checkRead(), checkWrite()}`

Возвращается методами: `FileInputStream.getFD()`, `FileOutputStream.getFD()`, `RandomAccessFile.getFD()`, `java.net.DatagramSocketImpl.getFileDescriptor()`, `java.net.SocketImpl.getFileDescriptor()`

Экземпляры: `FileDescriptor.{err, in, out}`, `java.net.DatagramSocketImpl.fd`, `java.net.SocketImpl.fd`

FileFilter Java 1.2

java.io

Этот интерфейс определяет метод `accept()`, который фильтрует список файлов. Вы можете прочитать содержимое каталога, вызвав метод `listFiles()` объекта `File`, который представляет необходимый каталог. Если вам необходим отфильтрованный список, например список файлов, не содержащий подкаталогов, или список файлов, чьи имена заканчиваются на `.class`, то вы можете передать объект `FileFilter` для `listFiles()`. Для каждого элемента каталога объект `File` передается методу `accept()`. Если `accept()` возвращает `true`, то `File` включается в возвращаемое значение `listFiles()`. Если `accept()` возвращает `false`, то этот элемент не включается в список. `FileFilter`

введен в Java 1.2. Используйте FilenameFilter, если требуется совместимость с предыдущими версиями Java или если вы предпочитаете фильтровать имена файлов (то есть объекты String), а не объекты File.

```
public interface FileFilter {
// Открытые методы экземпляра
   public abstract boolean accept(File pathname);
}
```

Передается методам: File.listFiles()

FileInputStream

Java 1.0

java.io

Этот класс является подклассом InputStream, который считывает байты из файла, указанного по имени или посредством объекта File или FileDescriptor. read() считывает байт или массив байтов из файла. Он возвращает –1, когда достигнут конец файла. При чтении двоичных данных этот класс обычно используется совместно с BufferedInputStream и DataInputStream. Для прочтения текста его обычно применяют с InputStreamReader и BufferedReader. Вызовите close() для закрытия файла, когда ввод уже не нужен.

В Java 1.4 и последующих версиях применяйте getChannel() для получения объекта FileChannel, чтобы читать данные из базового файла с использованием нового API ввода/вывода java.nio и его подпакетов.


```
public class FileInputStream extends java.io.InputStream {
// Открытые конструкторы
   public FileInputStream(String name) throws FileNotFoundException;
   public FileInputStream(File file) throws FileNotFoundException;
   public FileInputStream(FileDescriptor fdObj);
// Открытые методы экземпляра
1.4 public java.nio.channels.FileChannel getChannel();
   public final FileDescriptor getFD() throws IOException;
// Открытые методы, замещающие методы класса InputStream
   public int available() throws IOException;                          // зависит от платформы
   public void close() throws IOException;
   public int read() throws IOException;                               // зависит от платформы
   public int read(byte[] b) throws IOException;
   public int read(byte[] b, int off, int len) throws IOException;
   public long skip(long n) throws IOException;                        // зависит от платформы
// Замещенные защищенные методы класса Object
   protected void finalize() throws IOException;
}
```

FilenameFilter

Java 1.0

java.io

Этот интерфейс определяет метод accept(), который должен быть реализован любым объектом, фильтрующим имена файлов, то есть выбирающим подмножество имен

файлов из списка имен файлов. В Java не реализованы стандартные классы FilenameFilter, но объекты, которые реализуют этот интерфейс, используются объектом java.awt.FileDialog и методом File.list(). Типичный объект FilenameFilter будет осуществлять проверку того, что указанный File представляет файл (а не каталог), является читаемым (и по возможности перезаписываемым), а его имя заканчивается на требуемое расширение.

```
public interface FilenameFilter {
// Открытые методы экземпляра
   public abstract boolean accept(File dir, String name);
}
```

Передается методам: java.awt.FileDialog.setFilenameFilter(), java.awt.peer.FileDialogPeer.setFilenameFilter(), File.{list(), listFiles()}

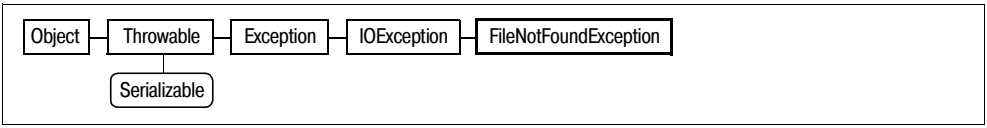
Возвращается методами: java.awt.FileDialog.getFilenameFilter()

FileNotFoundException

Java 1.0

java.io ***сериализуемое, проверяемое***

FileNotFoundException — это исключение IOException, которое свидетельствует о том, что указанный файл не найден.




```
public class FileNotFoundException extends IOException {
// Открытые конструкторы
   public FileNotFoundException();
   public FileNotFoundException(String s);
}
```

Генерируется методами: FileInputStream.FileInputStream(), FileOutputStream.FileOutputStream(), FileReader.FileReader(), RandomAccessFile.RandomAccessFile(), javax.imageio.stream.FileImageInputStream.FileImageInputStream(), javax.imageio.stream.FileImageOutputStream.FileImageOutputStream()

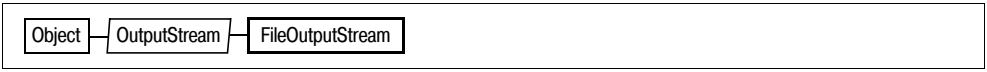
FileOutputStream

Java 1.0

java.io

Этот класс является подклассом OutputStream, который записывает данные в файл, указанный по имени или посредством объекта File или FileDescriptor. Если указанный файл уже существует, то FileOutputStream может быть сконфигурирован либо для перезаписи, либо для дополнения существующего файла. write() записывает байт или массив байтов в файл. Для записи двоичных данных этот класс применяется совместно с BufferedOutputStream и DataOutputStream. Для записи текста он обычно используется с PrintWriter, BufferedWriter и OutputStreamWriter либо применяется подходящий класс FileWriter. Используйте close() для закрытия FileOutputStream, если в него больше не будут записываться выходные данные.

В Java 1.4 и последующих версиях используйте `getChannel()` для получения объекта `FileChannel`, чтобы записать данные в базовый файл с помощью нового API ввода/вывода `java.nio` и его подпакетов.




```
public class FileOutputStream extends java.io.OutputStream {
// Открытые конструкторы
   public FileOutputStream(FileDescriptor fdObj);
   public FileOutputStream(File file) throws FileNotFoundException;
   public FileOutputStream(String name) throws FileNotFoundException;
1.1 public FileOutputStream(String name, boolean append) throws FileNotFoundException;
1.4 public FileOutputStream(File file, boolean append) throws FileNotFoundException;
// Открытые методы экземпляра
1.4 public java.nio.channels.FileChannel getChannel();
   public final FileDescriptor getFD() throws IOException;
// Открытые методы, замещающие методы класса OutputStream
   public void close() throws IOException;
   public void write(int b) throws IOException;                          // зависит от платформы
   public void write(byte[] b) throws IOException;
   public void write(byte[] b, int off, int len) throws IOException;
// Замещенные защищенные методы класса Object
   protected void finalize() throws IOException;
}
```

FilePermission

Java 1.2

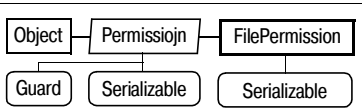
java.io ***сериализуемый***

Этот класс – `java.security.Permission`, который управляет доступом к локальной файловой системе. `FilePermission` имеет имя, или целевой объект (указывает на то, к какому файлу или файлам он относится), и список действий, которые могут быть выполнены с этим файлом или файлами (действия разделены запятыми). Поддерживаются следующие действия: чтение, запись, удаление и выполнение. Права на чтение и запись требуются любым методам, которые читают или записывают файл. Право на удаление необходимо `File.delete()`, а право на выполнение – `Runtime.exec()`.

Имя `FilePermission` может быть простым именем файла или каталога. Кроме того, `FilePermission` поддерживает использование определенных подстановочных знаков для указания права, которое применимо более чем к одному файлу. Если имя `FilePermission` является именем каталога, за которым следуют символы «/*» («*» в случае с платформами Windows), то оно указывает на все файлы в названном каталоге. Если имя является именем каталога, за которым следуют символы «/-» (либо «\-» для платформы Windows), то оно указывает на все файлы в каталоге и, рекурсивно, на все файлы во всех подкаталогах. Одна звездочка (*) указывает на все файлы в текущем каталоге, а одно тире (–) указывает на все файлы в текущем каталоге и его подкаталогах. Наконец, особое имя <<ALL FILES>> обозначает все файлы в файловой системе.

Приложениям не нужно использовать этот класс напрямую. Тем не менее он может понадобиться программистам, пишущим код системного уровня, и системным администраторам, отвечающим за конфигурацию системы безопасности. Будьте очень осмотрительны при предоставлении любых типов `FilePermission`. Ограничение досту-

па к файлам (в особенности доступа для записи) является одним из краеугольных камней механизма защиты применительно к ненадежному коду.



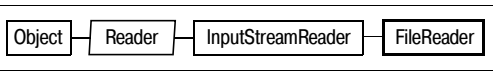

```
public final class FilePermission extends java.security.Permission implements Serializable {
// Открытые конструкторы
   public FilePermission(String path, String actions);
// Открытые методы, замещающие методы класса Permission
   public boolean equals(Object obj);
   public String getActions();
   public int hashCode();
   public boolean implies(java.security.Permission p);
   public java.security.PermissionCollection newPermissionCollection();
}
```

FileReader

Java 1.1

java.io

`FileReader` является удобным подклассом `InputStreamReader`, который полезен, когда вам необходимо прочесть из файла текст, а не двоичные данные. Вы создаете `FileReader`, указав файл для чтения в любой из трех возможных форм. Внутри конструктора `FileReader` создает `FileInputStream` для чтения байтов из указанного файла. Он применяет функциональность родительского класса `InputStreamReader` для перевода этих байтов из символов локальной кодировки в символы Unicode, используемые в Java. `FileReader` является тривиальным подклассом `InputStreamReader`, поэтому он не определяет какие-либо методы `read()` или другие собственные методы. Напротив, он наследует все методы из родительского класса. Если вам необходимо прочесть символы Unicode из файла, который использует какую-либо другую кодировку вместо кодировки по умолчанию, соответствующей данному региону (locale), то вы должны создать ваш собственный `InputStreamReader` для перевода байтов в символы.




```
public class FileReader extends InputStreamReader {
// Открытые конструкторы
   public FileReader(FileDescriptor fd);
   public FileReader(File file) throws FileNotFoundException;
   public FileReader(String fileName) throws FileNotFoundException;
}
```

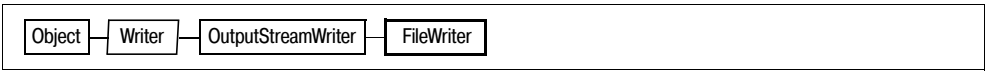
FileWriter

Java 1.1

java.io

`FileWriter` является удобным подклассом `OutputStreamWriter`, который полезен, когда вам необходимо записать в файл текст, а не двоичные данные. Вы создаете `FileWriter` путем указания файла, в который будет производиться запись. Вы можете опреде-

лить необходимость добавлять данные к концу существующего файла вместо того, чтобы перезаписывать этот файл. Класс FileWriter создает внутренний FileOutputStream для записи байтов в указанный файл. Он использует функциональность родительского класса OutputStreamWriter для перевода символов Unicode, записанных в поток, в байты с помощью кодировки по умолчанию, соответствующей данному региону. Если вам необходима кодировка, отличающаяся от кодировки по умолчанию, то вы не можете использовать FileWriter; в таком случае нужно создать собственный OutputStreamWriter и FileOutputStream. FileWriter является тривиальным подклассом OutputStreamWriter, поэтому он не определяет какие-либо собственные методы, а просто наследует их от родительского класса.




```
public class FileWriter extends OutputStreamWriter {
// Открытые конструкторы
   public FileWriter(File file) throws IOException;
   public FileWriter(FileDescriptor fd);
   public FileWriter(String fileName) throws IOException;
1.4 public FileWriter(File file, boolean append) throws IOException;
   public FileWriter(String fileName, boolean append) throws IOException;
}
```

FilterInputStream

Java 1.0

java.io

Этот класс предоставляет определения методов, которые требуются для фильтрации данных, полученных из InputStream, указанного при создании FilterInputStream. От него должен быть создан подкласс, реализующий какой-либо алгоритм фильтрации, поскольку непосредственное создание экземпляра данного класса невозможно. См. подклассы BufferedInputStream, DataInputStream и PushbackInputStream.


```
public class FilterInputStream extends java.io.InputStream {
// Защищенные конструкторы
   protected FilterInputStream(java.io.InputStream in);
// Открытые методы, замещающие методы класса InputStream
   public int available() throws IOException;
   public void close() throws IOException;
   public void mark(int readlimit);                                    // синхронизирован
   public boolean markSupported();
   public int read() throws IOException;
   public int read(byte[] b) throws IOException;
   public int read(byte[] b, int off, int len) throws IOException;
   public void reset() throws IOException;                             // синхронизирован
   public long skip(long n) throws IOException;
// Защищенные поля экземпляра
   protected java.io.InputStream in;
}
```

Подклассы: BufferedInputStream, java.io.DataInputStream, LineNumberInputStream, PushbackInputStream, java.security.DigestInputStream, java.util.zip.CheckedInputStream, java.util.zip.InflaterInputStream, javax.crypto.CipherInputStream, javax.swing.ProgressMonitorInputStream

FilterOutputStream

Java 1.0

java.io

Этот класс обеспечивает определения методов, необходимых для фильтрации данных, которые нужно записать в OutputStream, указываемый при создании FilterOutputStream. От него должен быть создан подкласс, реализующий какой-либо алгоритм фильтрации, поскольку непосредственное создание экземпляра данного класса невозможно. См. подклассы BufferedOutputStream, DataOutputStream.


```
public class FilterOutputStream extends java.io.OutputStream {
// Открытые конструкторы
   public FilterOutputStream(java.io.OutputStream out);
// Открытые методы, замещающие методы класса OutputStream
   public void close() throws IOException;
   public void flush() throws IOException;
   public void write(int b) throws IOException;
   public void write(byte[] b) throws IOException;
   public void write(byte[] b, int off, int len) throws IOException;
// Защищенные поля экземпляра
   protected java.io.OutputStream out;
}
```

Подклассы: BufferedOutputStream, java.io.DataOutputStream, PrintStream, java.security.DigestOutputStream, java.util.zip.CheckedOutputStream, java.util.zip.DeflaterOutputStream, javax.crypto.CipherOutputStream

FilterReader

Java 1.1

java.io

Этот абстрактный класс выступает в качестве родительского класса для входных потоков символов, которые считывают данные из какого-либо другого входного потока символов. С его помощью символы фильтруются каким-либо образом, а затем отфильтрованные данные возвращаются при вызове метода read(). FilterReader объявляется абстрактным, поэтому создать его экземпляр невозможно. Но ни один из его методов сам по себе не является абстрактным: они вызывают запрошенную операцию над входным потоком, передаваемым конструктору FilterReader(). Если создать экземпляр FilterReader, то можно обнаружить, что он является нулевым фильтром – он просто считывает символы из указанного входного потока и возвращает их без какой-либо фильтрации.

FilterReader реализует нулевой фильтр, поэтому он является идеальным родительским классом для классов, которые желают реализовать простые фильтры, но не хотят замещать все методы Reader. Для создания собственного отфильтрованного входного потока символов следует создать подкласс FilterReader и переопределить оба его метода read(), чтобы выполнять требуемую операцию фильтрации. Заметьте, что вы можете реализо-

вать один из методов read() посредством другого, то есть реализовать фильтрацию только один раз. Вспомните о том, что другие методы read(), определенные в классе Reader, реализуются посредством этих методов, поэтому у вас нет необходимости их замещать. В некоторых случаях понадобится заменить другие методы FilterReader и обеспечить методы или конструкторы, которые специфичны для вашего подкласса. FilterReader является аналогом, состоящим из потока символов, для FilterInputStream.


```
public abstract class FilterReader extends Reader {
// Защищенные конструкторы
   protected FilterReader(Reader in);
// Открытые методы, замещающие методы класса Reader
   public void close() throws IOException;
   public void mark(int readAheadLimit) throws IOException;
   public boolean markSupported();
   public int read() throws IOException;
   public int read(char[] cbuf, int off, int len) throws IOException;
   public boolean ready() throws IOException;
   public void reset() throws IOException;
   public long skip(long n) throws IOException;
// Защищенные поля экземпляра
   protected Reader in;
}
```

Подклассы: PushbackReader

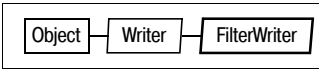
FilterWriter

Java 1.1

java.io

Этот абстрактный класс выступает в качестве родительского класса для выходных потоков символов, которые фильтруют записанные в них данные прежде, чем записать их в какой-либо другой выходной поток символов. FilterWriter объявляется абстрактным, чтобы нельзя было создать его экземпляр. Но ни один из его методов сам по себе не является абстрактным: они просто вызывают соответствующий метод в выходном потоке, который был передан конструктору FilterWriter. Если создать экземпляр объекта FilterWriter, то можно обнаружить, что он выступает в качестве нулевого фильтра, то есть просто передает записанные в него символы без какой-либо фильтрации.

FilterWriter реализует нулевой фильтр, поэтому он является идеальным родительским классом для классов, которые желают реализовать простые фильтры без необходимости замещать все методы Writer. Для создания собственного отфильтрованного выходного потока символов следует создать подкласс FilterWriter и заменить все его методы write(), чтобы выполнять требуемую операцию фильтрации. Заметьте, что вы можете реализовать два метода write() посредством третьего, то есть реализовать алгоритм фильтрации только один раз. В некоторых случаях необходимо заменить другие методы класса Writer и добавить дополнительные методы или конструкторы, которые специфичны для вашего подкласса. FilterWriter является аналогом, состоящим из потока символов, для FilterOutputStream.




```
public abstract class FilterWriter extends Writer {
// Защищенные конструкторы
   protected FilterWriter(Writer out);
// Открытые методы, замещающие методы класса Writer
   public void close() throws IOException;
   public void flush() throws IOException;
   public void write(int c) throws IOException;
   public void write(char[] cbuf, int off, int len) throws IOException;
   public void write(String str, int off, int len) throws IOException;
// Защищенные поля экземпляра
   protected Writer out;
}
```

InputStream

Java 1.0

java.io

Этот абстрактный класс является родительским классом всех входных потоков. Он определяет основные методы для чтения символов, предоставляемые всеми классами входящих потоков. `read()` считывает один байт или массив (или подмассив) байтов. Он возвращает считанный байт, количество считанных байтов или –1, если был достигнут конец файла. `skip()` пропускает указанное количество байтов ввода. `available()` возвращает количество байтов, которые могут быть прочитаны без блокировки. `close()` закрывает входной поток и освобождает все системные ресурсы, которые с ним ассоциированы. Поток не должен быть использован после вызова `close()`.

Если `markSupported()` возвращает `true` для данного `InputStream`, то такой поток поддерживает методы `mark()` и `reset()`. `mark()` запоминает текущую позицию во входном потоке, чтобы `reset()` мог вернуться к такой позиции (если только количество байтов, прочитанных между вызовами `mark()` и `reset()`, не превысит заданного предела). См. также `Reader`.

```
public abstract class InputStream {
// Открытые конструкторы
   public InputStream();
// Открытые методы экземпляра
   public int available() throws IOException;                          // константа
   public void close() throws IOException;                             // пустой
   public void mark(int readlimit);                                    // синхронизирован; пустой
   public boolean markSupported();                                     // константа
   public abstract int read() throws IOException;
   public int read(byte[] b) throws IOException;
   public int read(byte[] b, int off, int len) throws IOException;
   public void reset() throws IOException;                             // синхронизирован
   public long skip(long n) throws IOException;
}
```

Подклассы: `ByteArrayInputStream`, `FileInputStream`, `FilterInputStream`, `ObjectInputStream`, `PipedInputStream`, `SequenceInputStream`, `StringBufferInputStream`, `javax.sound.sampled.AudioInputStream`, `org.omg.CORBA.portable.InputStream`

Передается методам: Методов слишком много, чтобы их перечислить.

Возвращается методами: Методов слишком много, чтобы их перечислить.

Экземпляры: `FilterInputStream.in`, `System.in`

InputStreamReader Java 1.1

java.io

Этот класс является входным потоком символов, который использует входной поток байтов в качестве источника данных. Он считывает байты из указанного InputStream и переводит их в символы Unicode в соответствии с конкретной кодировкой символов, зависящей от платформы и региона. Эта важная функциональная особенность, связанная с локализацией, реализована в Java 1.1 и последующих версиях. InputStreamReader поддерживает стандартные методы класса Reader. Он также имеет метод getEncoding(), который возвращает имя кодировки, используемой для перевода байтов в символы.

При создании InputStreamReader вы указываете InputStream, из которого InputStreamReader должен читать байты. Кроме того, вы можете задать название кодировки символов, которая используется этими байтами. Если вы не указываете имя кодировки, то InputStreamReader использует кодировку по умолчанию, соответствующую предопределенному региону. В Java 1.4 и последующих версиях этот класс использует средства преобразования набора символов из пакета java.nio.charset и позволяет явно указать Charset или CharsetDecoder, который следует использовать. В версиях, предшествующих 1.4, этот класс позволяет указывать имя кодировки набора символов.

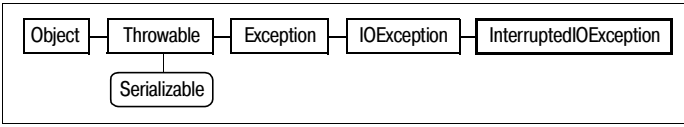

```
public class InputStreamReader extends Reader {
// Открытые конструкторы
   public InputStreamReader(java.io.InputStream in);
   public InputStreamReader(java.io.InputStream in, String charsetName)
        throws UnsupportedEncodingException;
1.4 public InputStreamReader(java.io.InputStream in, java.nio.charset.Charset cs);
1.4 public InputStreamReader(java.io.InputStream in, java.nio.charset.CharsetDecoder dec);
// Открытые методы экземпляра
   public String getEncoding();
// Открытые методы, замещающие методы класса Reader
   public void close() throws IOException;
   public int read() throws IOException;
   public int read(char[] cbuf, int offset, int length) throws IOException;
   public boolean ready() throws IOException;
}
```

Подклассы: FileReader

InterruptedIOException Java 1.0

java.io ***сериализуемое, проверяемое***

InterruptedIOEException — это исключение IOException, свидетельствующее о том, что операция ввода или вывода была прервана. Поле bytesTransferred содержит количество байтов, прочитанных или записанных до прерывания операции.




```
public class InterruptedIOException extends IOException {
// Открытые конструкторы
   public InterruptedIOException();
   public InterruptedIOException(String s);
// Открытые поля экземпляра
   public int bytesTransferred;
}
```

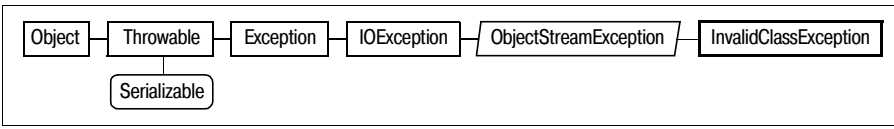
Подклассы: java.net.SocketTimeoutException

InvalidClassException

Java 1.1

java.io ***сериализуемое, проверяемое***

InvalidClassException свидетельствует о том, что механизм сериализации столкнулся с одной из нескольких возможных проблем, связанных с классом объекта, который сериализуется или десериализуется. Поле classname должно содержать имя рассматриваемого класса, а метод getMessage() замещается, чтобы возвращать имя класса вместе с сообщением.



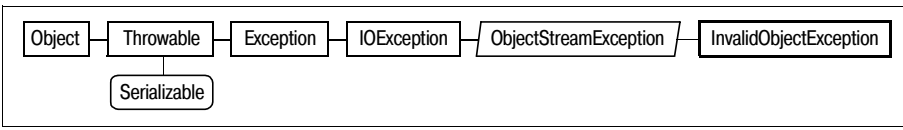

```
public class InvalidClassException extends ObjectStreamException {
// Открытые конструкторы
   public InvalidClassException(String reason);
   public InvalidClassException(String cname, String reason);
// Открытые методы, замещающие методы класса Throwable
   public String getMessage();
// Открытые поля экземпляра
   public String classname;
}
```

InvalidObjectException

Java 1.1

java.io ***сериализуемое, проверяемое***

Это исключение должно генерироваться методом validateObject() объекта, который реализует интерфейс ObjectInputValidation, если десериализуемый объект по какой-либо причине не проходит тест на подтверждение входных данных.




```
public class InvalidObjectException extends ObjectStreamException {
```

```
// Открытые конструкторы
   public InvalidObjectException(String reason);
}
```

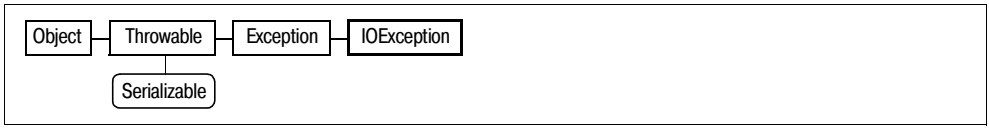
Генерируется методами: java.awt.font.TextAttribute.readResolve(), ObjectInputStream.registerValidation(), ObjectInputValidation.validateObject(), java.text.AttributedCharacterIterator.Attribute.readResolve(), java.text.DateFormat.Field.readResolve(), java.text.MessageFormat.Field.readResolve(), java.text.NumberFormat.Field.readResolve()

IOException

Java 1.0

java.io ***сериализуемое, проверяемое***

IOException свидетельствует о том, что во время ввода или вывода есть исключительное условие. В этом классе представлены несколько более специализированных подклассов. См. EOFException, FileNotFoundException, InterruptedIOException и UTFDataFormatException.




```
public class IOException extends Exception {
// Открытые конструкторы
   public IOException();
   public IOException(String s);
}
```

Подклассы: Классов слишком много, чтобы их перечислить.

Передается методам: java.awt.print.PrinterIOException.PrinterIOException()

Возвращается методами: java.awt.print.PrinterIOException.getIOException()

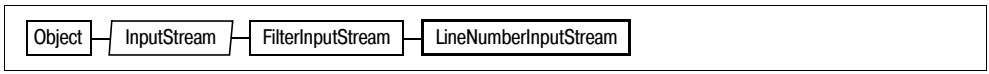
Генерируется методами: Методов слишком много, чтобы их перечислить.

LineNumberInputStream

Java 1.0, устарел в Java 1.1

java.io

Этот класс является FilterInputStream, который отслеживает количество прочитанных строк данных. getLineNumber() возвращает номер текущей строки; setLineNumber() устанавливает номер текущей строки. Последующим строкам присваиваются номера, начиная с этого номера. В Java 1.1 этот класс был признан устаревшим, поскольку он некорректно переводит байты в символы. Вместо него применяйте LineNumberReader.




```
public class LineNumberInputStream extends FilterInputStream {
// Открытые конструкторы
   public LineNumberInputStream(java.io.InputStream in);
// Открытые методы экземпляра
   public int getLineNumber();
   public void setLineNumber(int lineNumber);
```

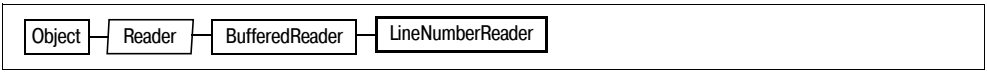
```
// Открытые методы, замещающие методы класса FilterInputStream
   public int available() throws IOException;
   public void mark(int readlimit);
   public int read() throws IOException;
   public int read(byte[] b, int off, int len) throws IOException;
   public void reset() throws IOException;
   public long skip(long n) throws IOException;
}
```

LineNumberReader

Java 1.1

java.io

Этот класс является входным потоком символов. Он отслеживает количество строк текста, прочитанных из потока. Поддерживаются обычные методы класса `Reader` и метод `readLine()`, введенный родительским классом. В дополнение к этим методам можно вызывать `getLineNumber()`, чтобы запросить количество строк, установленное на текущий момент. Кроме того, можно вызвать `setLineNumber()` для установки номера текущей строки. Последующие строки нумеруются последовательно, начиная с указанной отправной точки. Этот класс является аналогом, выраженным потоком символов, класса `LineNumberInputStream`, который был признан устаревшим в Java 1.1.



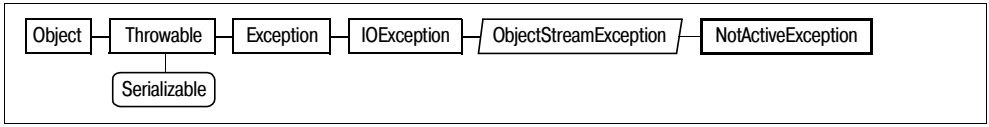

```
public class LineNumberReader extends BufferedReader {
// Открытые конструкторы
   public LineNumberReader(Reader in);
   public LineNumberReader(Reader in, int sz);
// Открытые методы экземпляра
   public int getLineNumber();
   public void setLineNumber(int lineNumber);
// Открытые методы, замещающие методы класса BufferedReader
   public void mark(int readAheadLimit) throws IOException;
   public int read() throws IOException;
   public int read(char[] cbuf, int off, int len) throws IOException;
   public String readLine() throws IOException;
   public void reset() throws IOException;
   public long skip(long n) throws IOException;
}
```

NotActiveException

Java 1.1

java.io ***сериализуемое, проверяемое***

Это исключение генерируется в нескольких случаях. Оно означает, что вызванный метод был вызван в неподходящее время или в неправильном контексте. Как правило, `ObjectOutputStream` или `ObjectInputStream` в настоящий момент неактивен, поэтому запрошенная операция не может быть выполнена.




```
public class NotActiveException extends ObjectStreamException {
// Открытые конструкторы
   public NotActiveException();
   public NotActiveException(String reason);
}
```

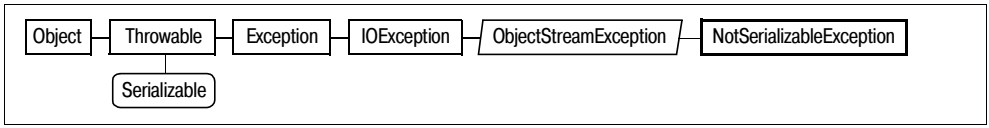
Генерируется методами: `ObjectInputStream.registerValidation()`

NotSerializableException

Java 1.1

java.io ***сериализуемое, проверяемое***

Это исключение свидетельствует о том, что объект не может быть сериализован. Оно генерируется в том случае, если происходит попытка сериализации экземпляра класса, который не реализует интерфейс `Serializable`. Имейте в виду, что оно выдается, если предпринимается попытка сериализовать объект `Serializable`, который ссылается на (или содержит) объект, не являющийся `Serializable`. Подкласс класса, являющегося `Serializable`, может предотвратить свою сериализацию путем генерации этого исключения из своих методов `writeObject()` и/или `readObject()`.




```
public class NotSerializableException extends ObjectStreamException {
// Открытые конструкторы
   public NotSerializableException();
   public NotSerializableException(String classname);
}
```

ObjectInput

Java 1.1

java.io

Этот интерфейс расширяет интерфейс `DataInput` и добавляет методы для десериализации объектов и чтения байтов и массивов байтов.


```
public interface ObjectInput extends DataInput {
// Открытые методы экземпляра
   public abstract int available() throws IOException;
   public abstract void close() throws IOException;
   public abstract int read() throws IOException;
   public abstract int read(byte[] b) throws IOException;
   public abstract int read(byte[] b, int off, int len) throws IOException;
   public abstract Object readObject() throws ClassNotFoundException,IOException;
```

```
   public abstract long skip(long n) throws IOException;
}
```

Реализации: ObjectInputStream

Передается методам: java.awt.datatransfer.DataFlavor.readExternal(), Externalizable.readExternal(), java.rmi.server.ObjID.read()

Возвращается методами: java.rmi.server.RemoteCall.getInputStream()

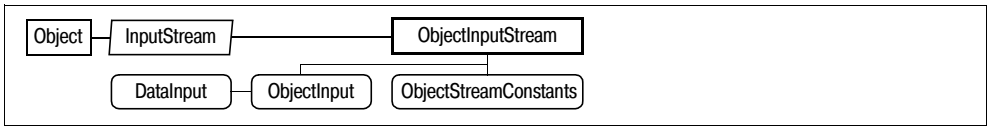
ObjectInputStream Java 1.1

java.io

ObjectInputStream десериализует объекты, массивы и другие значения из потока, который был ранее создан при помощи ObjectOutputStream. Метод readObject() десериализует объекты и массивы, которые затем должны быть приведены к подходящему типу; другие методы считывают значения примитивных данных из потока. Имейте в виду, что сериализовать или десериализовать можно только объекты, которые реализуют интерфейс Serializable или Externalizable.

Класс может реализовывать собственный закрытый метод readObject(ObjectInputStream), чтобы настроить способ своей десериализации. Если вы определяете такой метод, то в вашем распоряжении появляется несколько методов класса ObjectInputStream, которые вы можете использовать, чтобы сделать десериализацию объекта более легкой. defaultReadObject() является самым простым. Он считывает содержимое объекта точно так же, как это сделал бы ObjectInputStream. Если вы записали дополнительные данные до или после содержимого объекта по умолчанию, то вам следует прочитать эти данные до или после вызова defaultReadObject(). При работе с различными версиями или реализациями класса вам придется десериализовать набор полей, которые не совпадают с полями вашего класса. В этом случае создайте в своем классе статическое поле с именем serialPersistentFields. Его значением будет массив объектов ObjectStreamField, которые описывают поля, подлежащие десериализации. В этом случае ваш метод readObject() сможет вызвать readFields() для прочтения указанных полей из потока и возвращения их в объекте ObjectInputStream.GetField. Более подробная информация представлена в описании ObjectStreamField и ObjectInputStream.GetField. Наконец, вы можете вызвать registerValidation() из вашего метода readObject(). Этот метод регистрирует объект ObjectInputValidation (как правило, это объект, который десериализуется). Таким образом происходит уведомление о завершении десериализации целого дерева объектов, которое будет получено непосредственно перед завершением вызова метода readObject() класса ObjectInputStream.

Оставшиеся методы представляют собой разнообразные методы работы с потоками и несколько защищенных методов, предназначенных для использования подклассами, которые хотят настроить десериализацию ObjectInputStream.




```
public class ObjectInputStream extends java.io.InputStream
                         implements ObjectInput, ObjectStreamConstants {
// Открытые конструкторы
   public ObjectInputStream(java.io.InputStream in) throws IOException;
```

```
// Защищенные конструкторы
1.2 protected ObjectInputStream() throws IOException, SecurityException;
// Внутренние классы
1.2 public abstract static class GetField;
// Открытые методы экземпляра
   public void defaultReadObject() throws IOException,ClassNotFoundException;
1.2 public ObjectInputStream.GetField readFields() throws IOException,ClassNotFoundException;
1.4 public Object readUnshared() throws IOException,ClassNotFoundException;
   public void registerValidation(ObjectInputValidation obj, int prio)
        throws NotActiveException, InvalidObjectException;
// Методы, реализующие DataInput
   public boolean readBoolean() throws IOException;
   public byte readByte() throws IOException;
   public char readChar() throws IOException;
   public double readDouble() throws IOException;
   public float readFloat() throws IOException;
   public void readFully(byte[] buf) throws IOException;
   public void readFully(byte[] buf, int off, int len) throws IOException;
   public int readInt() throws IOException;
   public long readLong() throws IOException;
   public short readShort() throws IOException;
   public int readUnsignedByte() throws IOException;
   public int readUnsignedShort() throws IOException;
   public String readUTF() throws IOException;
   public int skipBytes(int len) throws IOException;
// Методы, реализующие ObjectInput
   public int available() throws IOException;
   public void close() throws IOException;
   public int read() throws IOException;
   public int read(byte[] buf, int off, int len) throws IOException;
   public final Object readObject() throws IOException, ClassNotFoundException;
// Защищенные методы экземпляра
   protected boolean enableResolveObject(boolean enable) throws SecurityException;
1.3 protected ObjectStreamClass readClassDescriptor() throws IOException, ClassNotFoundException;
1.2 protected Object readObjectOverride() throws IOException,
      ClassNotFoundException;  // константа
   protected void readStreamHeader() throws IOException,StreamCorruptedException;
   protected Class resolveClass(ObjectStreamClass desc) throws IOException,
      ClassNotFoundException;
   protected Object resolveObject(Object obj) throws IOException;
1.3 protected Class resolveProxyClass(String[] interfaces) throws IOException,
      ClassNotFoundException;
// Устаревшие открытые методы
# public String readLine() throws IOException;  // Реализует:DataInput
}
```

Передается методам:

java.beans.beancontext.BeanContextServicesSupport.bcsPreDeserializationHook(), java.beans.beancontext.BeanContextSupport.{bcsPreDeserializationHook(), deserialize(), readChildren()}, javax.rmi.CORBA.StubDelegate.readObject(), javax.swing.text.StyleContext.{readAttributes(), readAttributeSet()}

ObjectInputStream.GetField Java 1.2

java.io

Этот класс содержит значения именованных полей, прочитанных ObjectInputStream. Он дает программисту возможность жестко контролировать процесс десериализации и обычно используется при реализации объекта с набором полей, которые не соответствуют набору полей (и формату потока сериализации) оригинальной реализации объекта. Этот класс позволяет реализации класса изменяться без нарушения совместимости сериализации.

Для того чтобы использовать класс GetField, ваш класс должен реализовывать закрытый метод readObject(), который отвечает за специализированную десериализацию. Как правило, при использовании класса GetField указывается массив объектов ObjectStreamField как значение закрытого статического поля с именем serialPersistentFields. Этот массив указывает имена и типы всех полей, которые должны быть обнаружены при прочтении из потока сериализации. Если поле serialPersistentFields отсутствует, то массив объектов ObjectStreamField создается из фактических полей класса, за исключением полей static и transient.

В пределах метода readObject() вашего класса вызовите метод readFields(), принадлежащий ObjectInputStream(). Этот метод считывает значения всех полей из потока и сохраняет их в возвращаемом объекте ObjectInputStream.GetField. Объект GetField, по существу, является проекцией имен полей на их значения. Вы можете извлечь значения любых нужных вам полей, чтобы восстановить должное состояние объекта, который подвергается десериализации. Различные методы get() возвращают значения именованных полей указанных типов. Каждый метод принимает значение по умолчанию в качестве аргумента на тот случай, если в потоке сериализации не присутствовало какого-либо значения указанного поля. Например, это может произойти при десериализации объекта, записанного более ранней версией класса. Используйте метод defaulted() для определения того, содержит ли объект GetField значение для именованного поля. Если этот метод возвращает true, то именованное поле не имело значения в потоке, поэтому метод get() объекта GetField должен вернуть указанное значение по умолчанию. Метод getObjectStreamClass() объекта GetField возвращает объект ObjectStreamClass для объекта, который проходит десериализацию. Этот ObjectStreamClass может получить массив объектов ObjectStreamField для класса. См. также ObjectOutputStream.PutField.

```
public abstract static class ObjectInputStream.GetField {
// Открытые конструкторы
   public GetField();
// Открытые методы экземпляра
   public abstract boolean defaulted(String name) throws IOException;
   public abstract boolean get(String name, boolean val) throws IOException;
   public abstract byte get(String name, byte val) throws IOException;
   public abstract char get(String name, char val) throws IOException;
   public abstract short get(String name, short val) throws IOException;
   public abstract int get(String name, int val) throws IOException;
   public abstract long get(String name, long val) throws IOException;
   public abstract float get(String name, float val) throws IOException;
   public abstract double get(String name, double val) throws IOException;
   public abstract Object get(String name, Object val) throws IOException;
   public abstract ObjectStreamClass getObjectStreamClass();
}
```

Возвращается методами: ObjectInputStream.readFields()

ObjectInputValidation Java 1.1

java.io

Класс реализует этот интерфейс и определяет метод `validateObject()`, чтобы подтвердить правильность самого себя, когда он и все объекты, от которых он зависит, будут полностью десериализованы из `ObjectInputStream`. Однако метод `validateObject()` вызывается, если объект передается методу `ObjectInputStream.registerValidation()`; такая операция выполняется из метода `readObject()` объекта. Обратите внимание: если объект десериализуется как часть более крупного графа объекта, то его метод `validateObject()` не вызывается до тех пор, пока не будет прочитан весь граф; первоначальный вызов метода `ObjectInputStream.readObject()` будет готов выполнить возврат. `validateObject()` должен генерировать `InvalidObjectException`, если объект не проходит подтверждение. Это останавливает сериализацию объекта, а первоначальный вызов, направленный к `ObjectInputStream.readObject()`, завершается исключением `InvalidObjectException`.

```
public interface ObjectInputValidation {
// Открытые методы экземпляра
   public abstract void validateObject() throws InvalidObjectException;
}
```

Передается методам: `ObjectInputStream.registerValidation()`

ObjectOutput Java 1.1

java.io

Этот интерфейс расширяет интерфейс `DataOutput` и добавляет методы для сериализации объектов и записи байтов и массивов байтов.


```
public interface ObjectOutput extends DataOutput {
// Открытые методы экземпляра
   public abstract void close() throws IOException;
   public abstract void flush() throws IOException;
   public abstract void write(byte[] b) throws IOException;
   public abstract void write(int b) throws IOException;
   public abstract void write(byte[] b, int off, int len) throws IOException;
   public abstract void writeObject(Object obj) throws IOException;
}
```

Реализации: `ObjectOutputStream`

Передается методам: `java.awt.datatransfer.DataFlavor.writeExternal()`, `Externalizable.writeExternal()`, `ObjectOutputStream.PutField.write()`, `java.rmi.server.ObjID.write()`, `java.rmi.server.RemoteRef.getRefClass()`

Возвращается методами: `java.rmi.server.RemoteCall.{getOutputStream(), getResultStream()}`

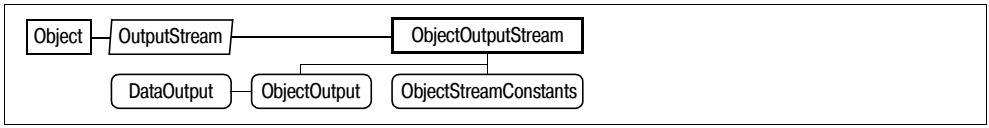
ObjectOutputStream Java 1.1

java.io

ObjectOutputStream сериализует объекты, массивы и другие значения в поток. Метод writeObject() сериализует объект или массив, а другие методы записывают в поток значения примитивных данных. Имейте в виду, что могут быть сериализованы только объекты, которые реализуют интерфейс Serializable или Externalizable.

Класс, который желает настроить сериализацию своих экземпляров, должен объявить закрытый метод writeObject(ObjectOutputStream). Этот метод вызывается при сериализации объекта и может использовать несколько дополнительных методов ObjectOutputStream. defaultWriteObject() выполняет ту же сериализацию, которая бы произошла, если бы не существовал метод writeObject(). Объект может вызвать этот метод, чтобы сериализовать самого себя, а затем вызвать другие методы ObjectOutputStream для записи дополнительных данных в поток сериализации. Конечно, класс должен определить подходящий метод readObject() для считывания дополнительных данных. При работе с различными версиями или реализациями класса вам, возможно, придется сериализовать набор полей, которые неточно совпадают с полями вашего класса. В этом случае придайте своему классу статическое поле с именем serialPersistentFields, чьим значением будет массив объектов ObjectStreamField, которые описывают поля, подлежащие сериализации. В вашем методе writeObject() вызовите putFields() для получения объекта ObjectOutputStream.PutField. Сохраните имена полей и значения в этом объекте, а затем вызовите writeFields() для их записи в поток сериализации. Более подробная информация представлена в описании ObjectStreamField и ObjectOutputStream.PutField.

Оставшиеся методы ObjectOutputStream являются разнообразными методами по работе с потоками и защищенными методами, которые предназначены для использования подклассами, желающими настроить свою сериализацию.




```
public class ObjectOutputStream extends java.io.OutputStream
       implements ObjectOutput, ObjectStreamConstants {
// Открытые конструкторы
   public ObjectOutputStream(java.io.OutputStream out) throws IOException;
// Защищенные конструкторы
1.2 protected ObjectOutputStream() throws IOException, SecurityException;
// Внутренние классы
1.2 public abstract static class PutField;
// Открытые методы экземпляра
   public void defaultWriteObject() throws IOException;
1.2 public ObjectOutputStream.PutField putFields() throws IOException;
   public void reset() throws IOException;
1.2 public void useProtocolVersion(int version) throws IOException;
1.2 public void writeFields() throws IOException;
1.4 public void writeUnshared(Object obj) throws IOException;
// Методы, реализующие DataOutput
   public void writeBoolean(boolean val) throws IOException;
   public void writeByte(int val) throws IOException;
   public void writeBytes(String str) throws IOException;
```

```
    public void writeChar(int val) throws IOException;
    public void writeChars(String str) throws IOException;
    public void writeDouble(double val) throws IOException;
    public void writeFloat(float val) throws IOException;
    public void writeInt(int val) throws IOException;
    public void writeLong(long val) throws IOException;
    public void writeShort(int val) throws IOException;
    public void writeUTF(String str) throws IOException;
// Методы, реализующие ObjectOutput
    public void close() throws IOException;
    public void flush() throws IOException;
    public void write(int val) throws IOException;
    public void write(byte[] buf) throws IOException;
    public void write(byte[] buf, int off, int len) throws IOException;
    public final void writeObject(Object obj) throws IOException;
// Защищенные методы экземпляра
    protected void annotateClass(Class cl) throws IOException;                    // пустой
1.3 protected void annotateProxyClass(Class cl) throws IOException;               // пустой
    protected void drain() throws IOException;
    protected boolean enableReplaceObject(boolean enable) throws SecurityException;
    protected Object replaceObject(Object obj) throws IOException;
1.3 protected void writeClassDescriptor(ObjectStreamClass desc) throws IOException;
1.2 protected void writeObjectOverride(Object obj) throws IOException;            // пустой
    protected void writeStreamHeader() throws IOException;
}
```

Передается методам: java.awt.AWTEventMulticaster.{save(), saveInternal()}, java.beans.beancontext.BeanContextServicesSupport.bcsPreSerializationHook(), java.beans.beancontext.BeanContextSupport.{bcsPreSerializationHook(), serialize(), writeChildren()}, javax.rmi.CORBA.StubDelegate.writeObject(), javax.swing.text.StyleContext.{writeAttributes(), writeAttributeSet()}

ObjectOutputStream.PutField Java 1.2

java.io

Этот класс содержит значения именованных полей и обеспечивает возможность их записи в ObjectOutputStream во время процесса сериализации объекта. Это дает программисту возможность четко контролировать процесс сериализации. Обычно bjectOutputStream.PutField используется, если набор полей, определенных классом, не соответствует набору полей (и формату потока сериализации), который определен оригинальной реализацией класса. Другими словами, ObjectOutputStream.PutField позволяет реализации класса изменяться без нарушения совместимости сериализации.

Для использования класса PutField обычно определяют закрытое статическое поле serialPersistentFields, которое ссылается на массив объектов ObjectStreamField. Этот массив определяет набор полей, записываемых в ObjectOutputStream, и таким образом определяет формат сериализации. Если вы не объявите поле serialPersistentFields, то набором полей будут все поля класса, за исключением полей static и transient.

В дополнение к полю serialPersistentFields ваш класс должен определить закрытый метод writeObject(), который отвечает за специализированную сериализацию вашего класса. В этом методе вызовите метод putFields(), принадлежащий ObjectOutputStream, для получения объекта ObjectOutputStream.PutField. Как только вы получите этот объект, используйте различные методы put() для указания имен и значений поля, которые следует записать. Набор именованных полей должен совпадать с полями,

которые указаны в serialPersistentFields. Вы можете указывать поля в любом порядке; класс PutField отвечает за то, чтобы они были записаны в правильном порядке. Как только вы указали значения всех полей, вызовите метод write() вашего объекта PutField, чтобы записать значения полей в поток сериализации.

Изменение специализированного процесса сериализации на противоположный представлено в описании ObjectInputStream.GetField.

```
public abstract static class ObjectOutputStream.PutField {
// Открытые конструкторы
   public PutField();
// Открытые методы экземпляра
   public abstract void put(String name, long val);
   public abstract void put(String name, int val);
   public abstract void put(String name, float val);
   public abstract void put(String name, Object val);
   public abstract void put(String name, double val);
   public abstract void put(String name, byte val);
   public abstract void put(String name, boolean val);
   public abstract void put(String name, short val);
   public abstract void put(String name, char val);
// Устаревшие открытые методы
# public abstract void write(ObjectOutput out) throws IOException;
}
```

Возвращается методами: ObjectOutputStream.putFields()

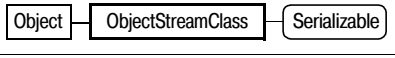
ObjectStreamClass

Java 1.1

java.io ***сериализуемый***

Этот класс представляет собой класс, который проходит сериализацию. Объект ObjectStreamClass содержит имя класса, уникальный идентификатор версии, имя и тип полей, которые составляют формат сериализации для класса. getSerialVersionUID() возвращает уникальный идентификатор версии класса. Он возвращает значение закрытого поля serialVersionUID класса или вычисленное значение, основанное на открытом API класса. В Java 1.2 и последующих версиях getFields() возвращает массив объектов ObjectStreamField, которые представляют имена и типы полей класса, подлежащие сериализации. getField() возвращает один объект ObjectStreamField, который представляет одно именованное поле. По умолчанию эти методы используют все поля класса за исключением тех, которые являются static или transient. И все же этот набор полей по умолчанию может быть изменен путем объявления в классе закрытого поля serialPersistentFields. Значением этого поля должен быть желаемый массив объектов ObjectStreamField.

Класс ObjectStreamClass не имеет конструктора; вам следует использовать статический метод lookup() для получения объекта ObjectStreamClass, предназначенного для данного объекта Class. Метод экземпляра forClass() выполняет противоположную операцию; он возвращает объект Class, который соответствует данному ObjectStreamClass. У большинства приложений никогда не возникает необходимости использовать этот класс.



```
public class ObjectStreamClass implements Serializable {
// Конструкторов нет
// Открытые константы
1.2 public static final ObjectStreamField[] NO_FIELDS;
// Открытые методы класса
   public static ObjectStreamClass lookup(Class cl);
// Открытые методы экземпляра
   public Class forClass();
1.2 public ObjectStreamField getField(String name);
1.2 public ObjectStreamField[] getFields();
   public String getName();
   public long getSerialVersionUID();
// Открытые методы, замещающие методы класса Object
   public String toString();
}
```

Передается методам:
`ObjectInputStream.resolveClass()`, `ObjectOutputStream.writeClassDescriptor()`

Возвращается методами: `ObjectInputStream.readClassDescriptor()`, `ObjectInputStream.GetField.getObjectStreamClass()`, `ObjectStreamClass.lookup()`

ObjectStreamConstants

Java 1.2

java.io

Этот интерфейс определяет различные константы, которые используются механизмом сериализации объектов Java. Двумя важными константами являются `PROTOCOL_VERSION_1` и `PROTOCOL_VERSION_2`, которые указывают, какую версию протокола сериализации следует использовать. В Java 1.2 вы можете передать любое из этих значений методу `useProtocolVersion()`, принадлежащему классу `ObjectOutputStream`. По умолчанию при сериализации объектов Java 1.2 использует версию 2 протокола, а Java 1.1 – версию 1. Подобно Java 1.1.7, Java 1.2 может десериализовать объекты, написанные с использованием любой версии протокола. Если вы хотите сериализовать объект таким образом, чтобы он мог быть прочитан версиями Java, предшествующими 1.1.7, используйте `PROTOCOL_VERSION_1`.

Другие константы, определяемые этим интерфейсом, являются низкоуровневыми значениями, которые используются протоколом сериализации. Вам нет необходимости использовать их, если только вы сами не собираетесь заново реализовать механизм сериализации.

```
public interface ObjectStreamConstants {
// Открытые константы
   public static final int baseWireHandle;                                    // =8257536
   public static final int PROTOCOL_VERSION_1;                                // =1
   public static final int PROTOCOL_VERSION_2;                                // =2
   public static final byte SC_BLOCK_DATA;                                    // =8
   public static final byte SC_EXTERNALIZABLE;                                // =4
   public static final byte SC_SERIALIZABLE;                                  // =2
   public static final byte SC_WRITE_METHOD;                                  // =1
   public static final short STREAM_MAGIC;                                    // =-21267
   public static final short STREAM_VERSION;                                  // =5
   public static final SerializablePermission SUBCLASS_IMPLEMENTATION_PERMISSION;
   public static final SerializablePermission SUBSTITUTION_PERMISSION;
   public static final byte TC_ARRAY;                                         // =117
   public static final byte TC_BASE;                                         // =112
```

```
   public static final byte TC_BLOCKDATA;                    // =119
   public static final byte TC_BLOCKDATALONG;                // =122
   public static final byte TC_CLASS;                        // =118
   public static final byte TC_CLASSDESC;                    // =114
   public static final byte TC_ENDBLOCKDATA;                 // =120
   public static final byte TC_EXCEPTION;                    // =123
1.3 public static final byte TC_LONGSTRING;                  // =124
   public static final byte TC_MAX;                          // =125
   public static final byte TC_NULL;                         // =112
   public static final byte TC_OBJECT;                       // =115
1.3 public static final byte TC_PROXYCLASSDESC;              // =125
   public static final byte TC_REFERENCE;                    // =113
   public static final byte TC_RESET;                        // =121
   public static final byte TC_STRING;                       // =116
}
```

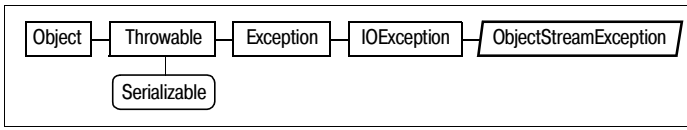
Реализации: ObjectInputStream, ObjectOutputStream

ObjectStreamException

Java 1.1

java.io ***сериализуемое, проверяемое***

Этот класс является родительским классом более специфичных типов исключений, которые могут возникнуть в процессе сериализации и десериализации объектов в классах ObjectOutputStream и ObjectInputStream.




```
public abstract class ObjectStreamException extends IOException {
// Защищенные конструкторы
   protected ObjectStreamException();
   protected ObjectStreamException(String classname);
}
```

Подклассы: InvalidClassException, InvalidObjectException, NotActiveException, NotSerializableException, OptionalDataException, StreamCorruptedException, WriteAbortedException

Генерируется методами: java.awt.AWTKeyStroke.readResolve(), java.awt.color.ICC_Profile.readResolve(), java.security.cert.Certificate.writeReplace(), java.security.cert.Certificate.CertificateRep.readResolve(), java.security.cert.CertPath.writeReplace(), java.security.cert.CertPath.CertPathRep.readResolve(), javax.print.attribute.EnumSyntax.readResolve()

ObjectStreamField

Java 1.2

java.io ***сравнимый***

Этот класс представляет именованное поле указанного типа (то есть указанного Class). Когда класс сериализует себя путем записи набора полей, которые отличаются от полей, используемых им в своей собственной реализации, то он определяет набор полей, которые будут записываться в массив объектов ObjectStreamField. Этот массив должен быть значением закрытого статического поля с именем serialPersis-

tentFields. Обычно методы этого класса используются механизмом сериализации. См. также ObjectOutputStream.PutField и ObjectInputStream.GetField.


```
public class ObjectStreamField implements Comparable {
// Открытые конструкторы
   public ObjectStreamField(String name, Class type);
1.4 public ObjectStreamField(String name, Class type, boolean unshared);
// Методы доступа к свойствам (по имени свойства)
   public String getName();
   public int getOffset();
   public boolean isPrimitive();
   public Class getType();
   public char getTypeCode();
   public String getTypeString();
1.4 public boolean isUnshared();
// Методы, реализующие Comparable
   public int compareTo(Object obj);
// Открытые методы, замещающие методы класса Object
   public String toString();
// Защищенные методы экземпляра
   protected void setOffset(int offset);
}
```

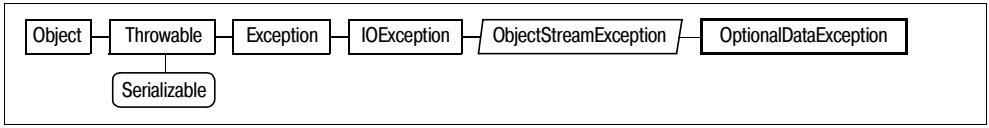
Возвращается методами: ObjectStreamClass.{getField(), getFields()}

Экземпляры: ObjectStreamClass.NO_FIELDS

OptionalDataException Java 1.1

java.io ***сериализуемое, проверяемое***

Генерируется методом readObject(), принадлежащим ObjectInputStream, когда он сталкивается с данными примитивного типа в том месте, где ожидает данные объекта. Несмотря на название исключения, эти данные не являются необязательными, а десериализация объекта прекращается.




```
public class OptionalDataException extends ObjectStreamException {
// Конструктор отсутствует
// Открытые поля экземпляра
   public boolean eof;
   public int length;
}
```

OutputStream Java 1.0

java.io

Этот абстрактный класс является родительским классом всех выходных потоков. Он определяет основные методы вывода, предоставляемые всеми классами выходных потоков. `write()` записывает один байт или массив (или подмассив) байтов. `flush()` выводит запись любого буферизованного вывода. `close()` закрывает поток и освобождает все системные ресурсы, которые с ним связаны. Поток не должен быть использован после вызова `close()`. См. также `Writer`.

```
public abstract class OutputStream {
// Открытые конструкторы
   public OutputStream();
// Открытые методы экземпляра
   public void close() throws IOException;                                    // пустой
   public void flush() throws IOException;                                    // пустой
   public abstract void write(int b) throws IOException;
   public void write(byte[] b) throws IOException;
   public void write(byte[] b, int off, int len) throws IOException;
}
```

Подклассы: `ByteArrayOutputStream`, `FileOutputStream`, `FilterOutputStream`, `ObjectOutputStream`, `PipedOutputStream`, `org.omg.CORBA.portable.OutputStream`

***Передается методам*:** Методов слишком много, чтобы их перечислить.

Возвращается методами: `Process.getOutputStream()`, `Runtime.getLocalizedOutputStream()`, `java.net.Socket.getOutputStream()`, `java.net.SocketImpl.getOutputStream()`, `java.net.URLConnection.getOutputStream()`, `java.nio.channels.Channels.newOutputStream()`, `java.rmi.server.LogStream.getOutputStream()`, `java.sql.Blob.setBinaryStream()`, `java.sql.Clob.setAsciiStream()`, `javax.print.StreamPrintService.getOutputStream()`, `javax.xml.transform.stream.StreamResult.getOutputStream()`

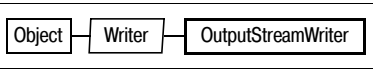
Экземпляры: `FilterOutputStream.out`

OutputStreamWriter Java 1.1

java.io

Этот класс является выходным потоком символов, который использует выходной поток байтов в качестве пункта назначения для своих данных. Когда символы записываются в `OutputStreamWriter`, он переводит их в байты в соответствии с конкретной кодировкой символов, специфичной для данного региона и/или платформы, а затем записывает эти байты в указанный `OutputStream`. Это важная функциональная особенность, связанная с локализацией, представлена в Java 1.1 и последующих версиях. `OutputStreamWriter` поддерживает обычные методы `Writer`. Он также имеет метод `getEncoding()`, который возвращает имя кодировки, используемой для перевода символов в байты.

При создании `OutputStreamWriter` вы указываете `OutputStream`, в который он записывает байты. Кроме того, вы можете указать название кодировки символов, которая должна использоваться для перевода символов в байты. Если вы не указываете имя кодировки, то `OutputStreamWriter` использует кодировку по умолчанию, существующую для предопределенного региона. В Java 1.4 и последующих версиях этот класс использует средства перевода набора символов из пакета `java.nio.charset` и позволяет явно указать необходимый `Charset` или `CharsetEncoder`. В версиях, предшествующих 1.4, класс позволяет указывать имя кодировки набора символов.



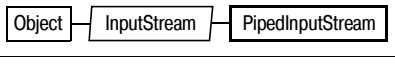

```
public class OutputStreamWriter extends Writer {
// Открытые конструкторы
   public OutputStreamWriter(java.io.OutputStream out);
   public OutputStreamWriter(java.io.OutputStream out, String charsetName)
        throws UnsupportedEncodingException;
1.4 public OutputStreamWriter(java.io.OutputStream out, java.nio.charset.CharsetEncoder enc);
1.4 public OutputStreamWriter(java.io.OutputStream out, java.nio.charset.Charset cs);
// Открытые методы экземпляра
   public String getEncoding();
// Открытые методы, замещающие методы класса Writer
   public void close() throws IOException;
   public void flush() throws IOException;
   public void write(int c) throws IOException;
   public void write(char[] cbuf, int off, int len) throws IOException;
   public void write(String str, int off, int len) throws IOException;
}
```

Подклассы: FileWriter

PipedInputStream Java 1.0

java.io

Этот класс – InputStream, который реализует одну половину канала и полезен при передаче данных между потоками выполнения (threads). PipedInputStream должен быть присоединен к объекту PipedOutputStream, который можно указать при создании PipedInputStream или при помощи метода connect(). Данные, прочитанные из объекта PipedInputStream, получены из PipedOutputStream, к которому он присоединен. В описании InputStream рассказано о низкоуровневых методах, предназначенных для прочтения данных из PipedInputStream. FilterInputStream может обеспечить интерфейс более высокого уровня для прочтения данных из PipedInputStream.




```
public class PipedInputStream extends java.io.InputStream {
// Открытые конструкторы
   public PipedInputStream();
   public PipedInputStream(PipedOutputStream src) throws IOException;
// Защищенные константы
1.1 protected static final int PIPE_SIZE;                                    // =1024
// Открытые методы экземпляра
   public void connect(PipedOutputStream src) throws IOException;
// Открытые методы, замещающие методы класса InputStream
   public int available() throws IOException;                              // синхронизирован
   public void close() throws IOException;
   public int read() throws IOException;                                   // синхронизирован
   public int read(byte[] b, int off, int len) throws IOException;         // синхронизирован
// Защищенные методы экземпляра
1.1 protected void receive(int b) throws IOException;                      // синхронизирован
// Защищенные поля экземпляра
1.1 protected byte[] buffer;
```

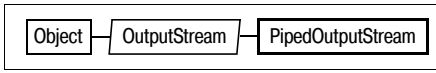
```
1.1 protected int in;
1.1 protected int out;
}
```

Передается методам: PipedOutputStream.{connect(), PipedOutputStream()}

PipedOutputStream Java 1.0

java.io

Этот класс – OutputStream, который реализует одну половину канала и полезен при взаимодействии потоков. PipedOutputStream должен быть присоединен к PipedInputStream, который можно указать при создании PipedOutputStream или при помощи метода connect(). Данные, записанные в PipedOutputStream, доступны для чтения в PipedInputStream. В описании OutputStream рассказано о низкоуровневых методах, предназначенных для записи данных в PipedOutputStream. FilterOutputStream может обеспечить интерфейс более высокого уровня для записи данных в PipedOutputStream.



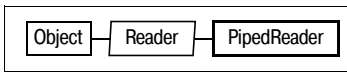

```
public class PipedOutputStream extends java.io.OutputStream {
// Открытые конструкторы
   public PipedOutputStream();
   public PipedOutputStream(PipedInputStream snk) throws IOException;
// Открытые методы экземпляра
   public void connect(PipedInputStream snk) throws IOException;          // синхронизирован
// Открытые методы, замещающие методы класса OutputStream
   public void close() throws IOException;
   public void flush() throws IOException;                                // синхронизирован
   public void write(int b) throws IOException;
   public void write(byte[] b, int off, int len) throws IOException;
}
```

Передается методам: PipedInputStream.{connect(), PipedInputStream()}

PipedReader Java 1.1

java.io

PipedReader – это выходной поток символов, считывающий символы из выходного потока символов PipedWriter, к которому он присоединен. PipedReader реализует одну половину канала. Он полезен при передаче данных между двумя потоками приложения. PipedReader не может быть использован до тех пор, пока он не будет присоединен к объекту PipedWriter, который может быть передан конструктору PipedReader() или методу connect(). PipedReader наследует большую часть методов родительского класса. См. также Reader. PipedReader является символьным аналогом класса PipedInputStream.




```
public class PipedReader extends Reader {
// Открытые конструкторы
   public PipedReader();
```

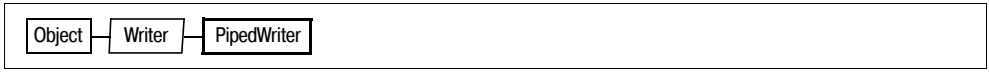
```
   public PipedReader(PipedWriter src) throws IOException;
// Открытые методы экземпляра
   public void connect(PipedWriter src) throws IOException;
// Открытые методы, замещающие методы класса Reader
   public void close() throws IOException;
1.2 public int read() throws IOException;                                    // синхронизирован
   public int read(char[] cbuf, int off, int len) throws IOException;        // синхронизирован
1.2 public boolean ready() throws IOException;                               // синхронизирован
}
```

Передается методам: PipedWriter.{connect(), PipedWriter()}

PipedWriter Java 1.1

java.io

PipedWriter — это выходной поток символов, который записывает символы во входящий поток символов PipedReader, к которому он присоединен. PipedWriter реализует одну половину канала и полезен при передаче данных между двумя потоками приложения. PipedWriter не может быть использован до тех пор, пока он не будет присоединен к объекту PipedReader, который может быть передан конструктору PipedWriter() или методу connect(). PipedWriter наследует большую часть методов родительского класса. См. также Writer. PipedWriter является символьным аналогом класса PipedOutputStream.




```
public class PipedWriter extends Writer {
// Открытые конструкторы
   public PipedWriter();
   public PipedWriter(PipedReader snk) throws IOException;
// Открытые методы экземпляра
   public void connect(PipedReader snk) throws IOException;  // синхронизирован
// Открытые методы, замещающие методы класса Writer
   public void close() throws IOException;
   public void flush() throws IOException;                   // синхронизирован
1.2 public void write(int c) throws IOException;
   public void write(char[] cbuf, int off, int len) throws IOException;
}
```

Передается методам: PipedReader.{connect(), PipedReader()}

PrintStream Java 1.0

java.io

Этот класс является FilterOutputStream, который реализует ряд методов, предназначенных для отображения текстового представления примитивных типов данных Java. Методы print() выводят стандартное текстовое представление каждого типа данных. Методы println() делают то же самое, но добавляют в конец символ новой строки. Каждый метод переводит примитивный тип Java в String и выводит итоговую строку. Когда Object передается print() или println(), он преобразуется в String путем вызова его метода toString(). PrintStream является типом OutputStream, который делает вывод текста наиболее легким. По этой причине он наиболее широко применяется

по сравнению с другими выходными потоками. Переменной System.out является объект класса PrintStream.

Обратите внимание, что в Java 1.0 этот класс обрабатывает символы Unicode неправильно; он не учитывает старшие 8 бит всех 16-битных символов. Таким образом, он работает только с символами Latin-1 (ISO8859-1). Хотя в Java 1.1 эта проблема была решена, PrintStream был вытеснен PrintWriter. Конструкторы этого класса признаны устаревшими, однако сам класс все еще используется стандартными выходными потоками System.out и System.err.

PrintStream и его замена, PrintWriter, выводят текстовые представления типов данных Java. Для вывода двоичных представлений данных используйте DataOutputStream.


```
public class PrintStream extends FilterOutputStream {
// Открытые конструкторы
   public PrintStream(java.io.OutputStream out);
   public PrintStream(java.io.OutputStream out, boolean autoFlush);
1.4 public PrintStream(java.io.OutputStream out, boolean autoFlush, String encoding)
        throws UnsupportedEncodingException;
// Замещенные методы экземпляра
   public boolean checkError();
   public void print(float f);
   public void print(long l);
   public void print(int i);
   public void print(double d);
   public void print(char[] s);
   public void print(String s);
   public void print(Object obj);
   public void print(char c);
   public void print(boolean b);
   public void println();
   public void println(float x);
   public void println(long x);
   public void println(int x);
   public void println(String x);
   public void println(Object x);
   public void println(double x);
   public void println(char[] x);
   public void println(char x);
   public void println(boolean x);
// Открытые методы, замещающие методы класса FilterOutputStream
   public void close();
   public void flush();
   public void write(int b);
   public void write(byte[] buf, int off, int len);
// Защищенные методы экземпляра
1.1 protected void setError();
}
```

Подклассы: java.rmi.server.LogStream

Передается методам: Методов слишком много, чтобы их перечислить.

Возвращается методами: java.rmi.server.LogStream.getDefaultStream(), java.rmi.server.RemoteServer.getLog(), java.sql.DriverManager.getLogStream(), javax.swing.DebugGraphics.logStream()

Экземпляры: System.{err, out}

PrintWriter Java 1.1

java.io

Этот класс является выходным потоком символов, который реализует ряд методов print() и println(), выводящих текстовые представления примитивных значений и объектов.

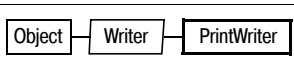
Когда вы создаете объект PrintWriter, вы указываете выходной поток символов или байтов, в которые он должен записывать свои символы. Кроме того, вы можете указать, будет ли автоматически сбрасываться поток PrintWriter при вызове println().

Если в качестве пункта назначения вы укажете выходной поток байтов, конструктор PrintWriter() автоматически создаст необходимый объект OutputStreamWriter для перевода символов в байты с использованием кодировки по умолчанию.

PrintWriter реализует обычные методы write(), flush() и close(), которые определяются всеми подклассами Writer. Чаще применяются методы print() и println(). Это методы более высокого уровня, каждый из которых переводит свой аргумент в строку перед тем, как его выводить. Кроме того, println() может ограничивать строку и сбрасывать буфер после распечатки своего аргумента.

Методы PrintWriter никогда не генерируют исключений. Вместо этого при возникновении ошибки они устанавливают внутренний флаг. Этот флаг можно проверить, вызвав checkError(). checkError() сначала сбрасывает внутренний поток, а затем возвращает true, если произошло какое-либо исключение во время записи значений в этот поток. Как только произошла ошибка в объекте PrintWriter, все последующие запросы к checkError() будут возвращать true; нет никакой возможности сбросить флаг ошибки.

PrintWriter является символьным аналогом класса PrintStream, который он замещает. Обычно вы можете тривиально заменить любые объекты PrintStream в программе на объекты PrintWriter. Это особенно важно для многоязыковых программ. Единственный приемлемый вариант использования класса PrintStream – это применение стандартных выходных потоков System.out и System.err. См. PrintStream на предмет более детальной информации.




```
public class PrintWriter extends Writer {
// Открытые конструкторы
   public PrintWriter(java.io.OutputStream out);
   public PrintWriter(Writer out);
   public PrintWriter(java.io.OutputStream out, boolean autoFlush);
   public PrintWriter(Writer out, boolean autoFlush);
// Открытые методы экземпляра
   public boolean checkError();
   public void print(int i);
   public void print(long l);
   public void print(char c);
   public void print(boolean b);
   public void print(float f);
   public void print(double d);
   public void print(char[] s);
   public void print(Object obj);
   public void print(String s);
```

```
   public void println();
   public void println(int x);
   public void println(long x);
   public void println(boolean x);
   public void println(char x);
   public void println(String x);
   public void println(Object x);
   public void println(char[] x);
   public void println(float x);
   public void println(double x);
// Открытые методы, замещающие методы класса Writer
   public void close();
   public void flush();
   public void write(char[] buf);
   public void write(String s);
   public void write(int c);
   public void write(String s, int off, int len);
   public void write(char[] buf, int off, int len);
// Защищенные методы экземпляра
   protected void setError();
// Защищенные поля экземпляра
1.2 protected Writer out;
}
```

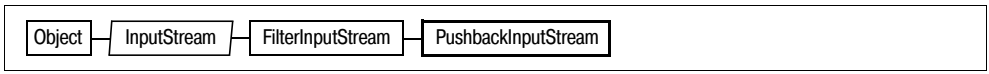
Передается методам: java.awt.Component.list(), java.awt.Container.list(), Throwable.printStackTrace(), java.security.cert.CertPathBuilderException.printStackTrace(), java.security.cert.CertPathValidatorException.printStackTrace(), java.security.cert.CertStoreException.printStackTrace(), java.sql.DriverManager.setLogWriter(), java.util.Properties.list(), javax.naming.NamingException.printStackTrace(), javax.sql.ConnectionPoolDataSource.setLogWriter(), javax.sql.DataSource.setLogWriter(), javax.sql.XADataSource.setLogWriter(), javax.xml.transform.TransformerException.printStackTrace()

Возвращается методами: java.sql.DriverManager.getLogWriter(), javax.sql.ConnectionPoolDataSource.getLogWriter(), javax.sql.DataSource.getLogWriter(), javax.sql.XADataSource.getLogWriter()

PushbackInputStream Java 1.0

java.io

Этот класс – FilterInputStream, который реализует однобайтный буфер возврата в поток или (начиная с Java 1.1) буфер возврата в поток указанной длины. Методы unread() возвращают байты обратно в поток; эти байты являются первыми байтами, которые прочитываются последующим вызовом метода read(). Этот класс может быть полезен при написании парсеров (программ синтаксического анализа, parsers). См. также PushbackReader.



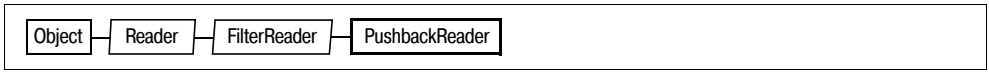

```
public class PushbackInputStream extends FilterInputStream {
// Открытые конструкторы
   public PushbackInputStream(java.io.InputStream in);
1.1 public PushbackInputStream(java.io.InputStream in, int size);
```

```
// Открытые методы экземпляра
   public void unread(int b) throws IOException;
1.1 public void unread(byte[] b) throws IOException;
1.1 public void unread(byte[] b, int off, int len) throws IOException;
// Открытые методы, замещающие методы класса FilterInputStream
   public int available() throws IOException;
1.2 public void close() throws IOException;                          // синхронизирован
   public boolean markSupported();                                   // константа
   public int read() throws IOException;
   public int read(byte[] b, int off, int len) throws IOException;
1.2 public long skip(long n) throws IOException;
// Защищенные поля экземпляра
1.1 protected byte[] buf;
1.1 protected int pos;
}
```

PushbackReader Java 1.1

java.io

Этот класс является входным потоком символов, который использует другой входной поток в качестве источника входных данных и добавляет возможность возвращать символы обратно в поток. Эта функциональная особенность часто оказывается полезной при написании парсеров. При создании потока PushbackReader вы указываете поток, из которого будет производиться считывание. Кроме того, можно указать размер буфера возврата в поток, то есть количество символов, которое может быть возвращено обратно в поток (unread). Если вы не укажете размер этого буфера, то размер по умолчанию будет равен одному символу. PushbackReader наследует или замещает все стандартные методы Reader и добавляет три метода unread(), которые возвращают обратно в поток один символ, массив символов или часть массива символов. Этот класс является символьным аналогом класса PushbackInputStream.




```
public class PushbackReader extends FilterReader {
// Открытые конструкторы
   public PushbackReader(Reader in);
   public PushbackReader(Reader in, int size);
// Открытые методы экземпляра
   public void unread(int c) throws IOException;
   public void unread(char[] cbuf) throws IOException;
   public void unread(char[] cbuf, int off, int len) throws IOException;
// Открытые методы, замещающие методы класса FilterReader
   public void close() throws IOException;
1.2 public void mark(int readAheadLimit) throws IOException;
   public boolean markSupported();                                   // константа
   public int read() throws IOException;
   public int read(char[] cbuf, int off, int len) throws IOException;
   public boolean ready() throws IOException;
1.2 public void reset() throws IOException;
}
```

RandomAccessFile Java 1.0

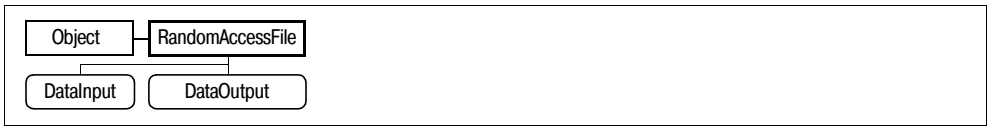
java.io

Этот класс позволяет записывать и считывать произвольные байты, текст и примитивные типы данных Java из любого или в любое место файла. Так как этот класс предоставляет произвольный, а не последовательный доступ к файлам, то он не является ни подклассом `InputStream`, ни подклассом `OutputStream`; он предоставляет абсолютно независимый способ чтения и записи данных из файлов и в файлы. `RandomAccessFile` реализует те же интерфейсы, что и `DataInputStream` и `DataOutputStream`. Таким образом, он определяет те же методы для чтения и записи данных, что и вышеупомянутые классы.

Метод `seek()` обеспечивает произвольный доступ к файлу; он используется для выбора того места в файле, в которое будут записаны данные или из которого они будут прочитаны. Различные методы записи и чтения обновляют позицию в файле, чтобы последовательность операций записи или чтения могла быть выполнена на непрерывной части файла без необходимости вызова метода `seek()` перед каждой операцией записи или чтения.

Для файла, который предназначен только для чтения, аргумент `mode` к методам конструктора должен быть «r», а для файла, который предназначен для записи (и, возможно, также для чтения) – «rw». Для аргумента `mode` в Java 1.4 и последующих версиях также разрешены два других значения. Режим «rwd» открывает файл для чтения и записи. Если файл находится в локальной файловой системе, то каждое обновление содержимого файла должно быть синхронно записано в базовый файл. Режим «rws» похож на предыдущий, но требует синхронных обновлений как содержимого файла, так и его «метаданных» (например, времени доступа к файлу). Использование режима «rws» может потребовать того, чтобы метаданные файла модифицировались при каждом чтении файла.

В Java 1.4 и последующих версиях используйте метод `getChannel()` для получения объекта `FileChannel`, который можно задействовать для доступа к файлу с применением нового API ввода/вывода `java.nio` и его подпакетов. Если `RandomAccessFile` был открыт в режиме «r», то `FileChannel` разрешит только чтение. В остальных случаях он разрешит как запись, так и чтение.




```
public class RandomAccessFile implements DataInput,
      DataOutput {
// Открытые конструкторы
   public RandomAccessFile(File file, String mode) throws FileNotFoundException;
   public RandomAccessFile(String name, String mode) throws FileNotFoundException;
// Открытые методы экземпляра
   public void close() throws IOException;                          // зависит от платформы
1.4 public final java.nio.channels.FileChannel getChannel();
   public final FileDescriptor getFD() throws IOException;
   public long getFilePointer() throws IOException;                 // зависит от платформы
   public long length() throws IOException;                         // зависит от платформы
   public int read() throws IOException;                            // зависит от платформы
   public int read(byte[] b) throws IOException;
```

```
   public int read(byte[] b, int off, int len) throws IOException;
   public void seek(long pos) throws IOException;                    // зависит от платформы
1.2 public void setLength(long newLength) throws IOException;        // зависит от платформы
// Методы, реализующие DataInput
   public final boolean readBoolean() throws IOException;
   public final byte readByte() throws IOException;
   public final char readChar() throws IOException;
   public final double readDouble() throws IOException;
   public final float readFloat() throws IOException;
   public final void readFully(byte[] b) throws IOException;
   public final void readFully(byte[] b, int off, int len) throws IOException;
   public final int readInt() throws IOException;
   public final String readLine() throws IOException;
   public final long readLong() throws IOException;
   public final short readShort() throws IOException;
   public final int readUnsignedByte() throws IOException;
   public final int readUnsignedShort() throws IOException;
   public final String readUTF() throws IOException;
   public int skipBytes(int n) throws IOException;
// Методы, реализующие DataOutput
   public void write(int b) throws IOException;                      // зависит от платформы
   public void write(byte[] b) throws IOException;
   public void write(byte[] b, int off, int len) throws IOException;
   public final void writeBoolean(boolean v) throws IOException;
   public final void writeByte(int v) throws IOException;
   public final void writeBytes(String s) throws IOException;
   public final void writeChar(int v) throws IOException;
   public final void writeChars(String s) throws IOException;
   public final void writeDouble(double v) throws IOException;
   public final void writeFloat(float v) throws IOException;
   public final void writeInt(int v) throws IOException;
   public final void writeLong(long v) throws IOException;
   public final void writeShort(int v) throws IOException;
   public final void writeUTF(String str) throws IOException;
}
```

Передается методам: `javax.imageio.stream.FileImageInputStream.FileImageInputStream()`, `javax.imageio.stream.FileImageOutputStream.FileImageOutputStream()`

Reader

Java 1.1

java.io

Этот абстрактный класс является родительским классом для всех входных потоков символов. Он представляет собой аналог `InputStream`, который является родительским классом всех входных потоков байтов. `Reader` определяет базовые методы, предоставляемые всеми входными потоками символов. `read()` возвращает один символ или массив (или подмассив) символов, при необходимости проводя блокировку; он возвращает –1, если был достигнут конец потока. `ready()` возвращает `true`, если есть символы, доступные для чтения. Если `ready()` возвращает `true`, то последующий вызов метода`read()` не вызовет блокировки. `close()` закрывает входной поток символов. `skip()` пропускает указанное количество символов во входном потоке. Если `markSupported()` возвращает `true`, то `mark()` отмечает позицию в потоке и при необходимости создает буфер упреждающей выборки указанного размера. Последующие запросы к `reset()` восстанавливают поток в отмеченную позицию при условии, если они имеют

место в пределах указанного лимита упреждающей выборки. Отметим, что не все типы потоков поддерживают функциональность вида «отметить-и-сбросить». Для создания подкласса Reader вам нужно лишь реализовать версию метода read() с тремя аргументами и метод close(). Тем не менее многие подклассы реализуют дополнительные методы.

```
public abstract class Reader {
// Защищенные конструкторы
   protected Reader();
   protected Reader(Object lock);
// Открытые методы экземпляра
   public abstract void close() throws IOException;
   public void mark(int readAheadLimit) throws IOException;
   public boolean markSupported();                                    // константа
   public int read() throws IOException;
   public int read(char[] cbuf) throws IOException;
   public abstract int read(char[] cbuf, int off, int len) throws IOException;
   public boolean ready() throws IOException;                         // константа
   public void reset() throws IOException;
   public long skip(long n) throws IOException;
// Защищенные поля экземпляра
   protected Object lock;
}
```

Подклассы: BufferedReader, CharArrayReader, FilterReader, InputStreamReader, PipedReader, StringReader

Передается методам: Методов слишком много, чтобы их перечислить.

Возвращается методами: java.awt.datatransfer.DataFlavor.getReaderForText(), java.nio.channels.Channels.newReader(), java.sql.Clob.getCharacterStream(), java.sql.ResultSet.getCharacterStream(), java.sql.SQLInput.readCharacterStream(), javax.print.Doc.getReaderForText(), javax.print.SimpleDoc.getReaderForText(), javax.xml.transform.stream.StreamSource.getReader(), org.xml.sax.InputSource.getCharacterStream()

Экземпляры: FilterReader.in

SequenceInputStream Java 1.0

java.io

Этот класс обеспечивает способ эффективного сцепления данных из двух или более входных потоков. Он обеспечивает интерфейс InputStream для ряда объектов InputStream. Данные считываются из потоков в том порядке, в каком указаны потоки. Когда достигнут конец одного потока, данные автоматически считываются из следующего потока. Например, этот класс может пригодиться при реализации возможности вставки (include) файлов в каком-либо парсере.


```
public class SequenceInputStream extends java.io.InputStream {
// Открытые конструкторы
   public SequenceInputStream(java.util.Enumeration e);
```

```
   public SequenceInputStream(java.io.InputStream s1, java.io.InputStream s2);
// Открытые методы, замещающие методы класса InputStream
1.1 public int available() throws IOException;
   public void close() throws IOException;
   public int read() throws IOException;
   public int read(byte[] b, int off, int len) throws IOException;
}
```

Serializable Java 1.1

java.io ***сериализуемый***

Интерфейс Serializable не определяет никаких методов или констант. Класс должен реализовывать этот интерфейс, чтобы обозначить, что он позволяет сериализовать или же десериализовать себя при помощи ObjectOutputStream.writeObject() и ObjectInputStream.readObject().

Объекты, требующие специальной обработки во время сериализации или десериализации, могут реализовывать один или оба следующих метода. Обратите внимание, что эти методы не являются частью интерфейса Serializable:

```
private void writeObject(java.io.ObjectOutputStream out) throws IOException;
private void readObject(java.io.ObjectInputStream in) throws
                                                    ception,ClassNotFoundException;
```

Как правило, метод writeObject() выполняет всю необходимую очистку или подготовку к сериализации, вызывает метод defaultWriteObject(), принадлежащий ObjectOutputStream, для сериализации нерезидентных полей класса и при необходимости записывает какие-либо дополнительные данные. Аналогично, метод readObject() вызывает метод defaultReadObject(), принадлежащий ObjectInputStream, считывает какие-либо дополнительные данные, записанные соответствующим методом writeObject(), и выполняет любую дополнительную инициализацию, требуемую объектом. Метод readObject() может регистрировать объект ObjectInputValidation в целях подтверждения целостности объекта в тот момент, когда он будет полностью десериализован.

```
public interface Serializable {
}
```

Реализации: Классов слишком много, чтобы их перечислить.

Передается методам: Методов слишком много, чтобы их перечислить.

Возвращается методами: Методов слишком много, чтобы их перечислить.

Экземпляры: org.omg.CORBA.ValueBaseHolder.value

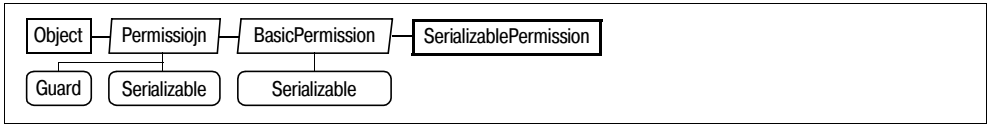
SerializablePermission Java 1.2

java.io ***сериализуемый***

Этот класс — java.security.Permission. Он управляет использованием определенных особенностей сериализации. Объекты SerializablePermission имеют имя, или целевой объект, но у них нет списка операций. Имя «enableSubclassImplementation» представляет разрешение на сериализацию или десериализацию объектов с использованием подклассов ObjectOutputStream и ObjectInputStream. Эта функция защищена правом доступа, поскольку злонамеренный программный код может определить подклассы потока объектов, которые неправильно сериализуют и десериализуют объекты.

Еще одно имя, поддерживаемое SerializablePermission, – «enableSubstitution», которое представляет для одного объекта право замены на другой во время сериализации или десериализации. Методам ObjectOutputStream.enableReplaceObject() и ObjectInputStream.enableResolveObject() необходимо право доступа такого типа.

У приложений никогда не возникает необходимости использовать этот класс. Его могут применять программисты, пишущие код системного уровня, а системные администраторы, отвечающие за конфигурацию политики безопасности, должны быть с ним знакомы.




```
public final class SerializablePermission extends java.security.BasicPermission {
// Открытые конструкторы
   public SerializablePermission(String name);
   public SerializablePermission(String name, String actions);
}
```

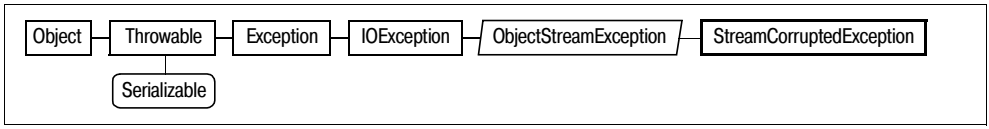
Экземпляры: ObjectStreamConstants.{SUBCLASS_IMPLEMENTATION_PERMISSION, SUBSTITUTION_PERMISSION}

StreamCorruptedException

Java 1.1

java.io ***сериализуемое, проверяемое***

Это исключение свидетельствует, что поток данных, который считывается ObjectInputStream, искажен и не содержит достоверных данных по сериализованному объекту.




```
public class StreamCorruptedException extends ObjectStreamException {
// Открытые конструкторы
   public StreamCorruptedException();
   public StreamCorruptedException(String reason);
}
```

Генерируется методами: ObjectInputStream.readStreamHeader(), java.rmi.server.RemoteCall.getResultStream()

StreamTokenizer

Java 1.0

java.io

Этот класс выполняет лексический анализ указанного входного потока и разбивает ввод на токены (tokens). Он может быть чрезвычайно полезен при написании простых парсеров. nextToken() возвращает следующий токен в потоке; это будет одна из констант, определенных классом (которые представляют конец файла, конец строки, число с плавающей запятой и слово), или же значение символа. pushBack() возвращает токен обратно в поток, чтобы он был возвращен при следующем вызове метода

nextToken(). Открытые переменные sval и nval содержат строковые и числовые значения того токена, который был прочитан самым последним. Они применимы в том случае, если возвращенный токен является TT_WORD или TT_NUMBER. lineno() возвращает номер текущей строки.

Оставшиеся методы позволяют указать, как будут распознаваться токены. wordChars() задает диапазон символов, которые должны трактоваться как часть слова. whitespaceChars() задает диапазон символов, которые служат для разграничения токенов. ordinaryChars() и ordinaryChar() указывают символы, которые никогда не являются частью токенов и должны возвращаться в том состоянии, в котором они находятся. resetSyntax() делает все символы обычными. eolIsSignificant() указывает, должен ли игнорироваться символ конца строки. Если эти символы не игнорируются, то для концов строк возвращается константа TT_EOL; в других случаях они обрабатываются как неотображаемые символы. commentChar() определяет символ, который начинает комментарий, длящийся до конца строки. Никакие символы в комментариях не возвращаются. slashStarComments() и slashSlashComments() определяют, будет ли StreamTokenizer распознавать комментарии в стиле C и C++. Если да, то никакая часть комментария не возвращается как токен. quoteChar() указывает символ, используемый для разграничения строк. Когда обрабатывается строковый токен, символ кавычек возвращается как значение токена, а тело строки сохраняется в переменной sval. lowerCaseMode() указывает на необходимость перевода токенов TT_WORD в символы, которые представлены только нижним регистром, перед тем как они будут сохранены в sval. parseNumbers() указывает, что StreamTokenizer должен распознавать и возвращать токены удвоенной точности, содержащие числа с плавающей точкой.

```
public class StreamTokenizer {
// Открытые конструкторы
# public StreamTokenizer(java.io.InputStream is);
1.1 public StreamTokenizer(Reader r);
// Открытые константы
   public static final int TT_EOF;                                    // =-1
   public static final int TT_EOL;                                    // =10
   public static final int TT_NUMBER;                                 // =-2
   public static final int TT_WORD;                                   // =-3
// Открытые методы экземпляра
   public void commentChar(int ch);
   public void eolIsSignificant(boolean flag);
   public int lineno();
   public void lowerCaseMode(boolean fl);
   public int nextToken() throws IOException;
   public void ordinaryChar(int ch);
   public void ordinaryChars(int low, int hi);
   public void parseNumbers();
   public void pushBack();
   public void quoteChar(int ch);
   public void resetSyntax();
   public void slashSlashComments(boolean flag);
   public void slashStarComments(boolean flag);
   public void whitespaceChars(int low, int hi);
   public void wordChars(int low, int hi);
// Открытые методы, замещающие методы класса Object
   public String toString();
// Открытые поля экземпляра
   public double nval;
   public String sval;
   public int ttype;
}
```

StringBufferInputStream

Java 1.0; устарел в java 1.1

java.io

Этот класс является подклассом InputStream, в котором входными байтами являются символы указанного объекта String. Этот класс некорректно переводит символы StringBuffer в байты, поэтому в Java 1.1 он признан устаревшим. Для перевода символов в байты служит StringReader, а для чтения байтов из массива – ByteArrayInputStream.

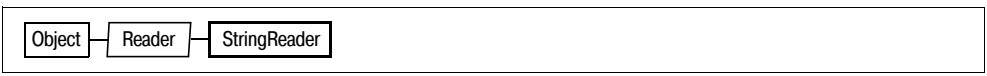

```
public class StringBufferInputStream extends java.io.InputStream {
// Открытые конструкторы
   public StringBufferInputStream(String s);
// Открытые методы, замещающие методы класса InputStream
   public int available();                                          // синхронизирован
   public int read();                                               // синхронизирован
   public int read(byte[] b, int off, int len);                     // синхронизирован
   public void reset();                                             // синхронизирован
   public long skip(long n);                                        // синхронизирован
// Защищенные поля экземпляра
   protected String buffer;
   protected int count;
   protected int pos;
}
```

StringReader

Java 1.1

java.io

Этот класс является входным потоком символов, который использует объект String в качестве источника возвращаемых символов. Когда вы создаете StringReader, укажите String, из которого будет производиться считывание. StringReader определяет обычные методы Reader и поддерживает mark() и reset(). Если reset() вызывается до вызова mark(), то поток устанавливается в начало указанной строки. StringReader является символьным аналогом класса StringBufferInputStream, который признан устаревшим в Java 1.1. StringReader также подобен CharArrayReader.




```
public class StringReader extends Reader {
// Открытые конструкторы
   public StringReader(String s);
// Открытые методы, замещающие методы класса Reader
   public void close();
   public void mark(int readAheadLimit) throws IOException;
   public boolean markSupported();                                  // константа
   public int read() throws IOException;
   public int read(char[] cbuf, int off, int len) throws IOException;
   public boolean ready() throws IOException;
   public void reset() throws IOException;
   public long skip(long ns) throws IOException;
}
```

StringWriter Java 1.1

java.io

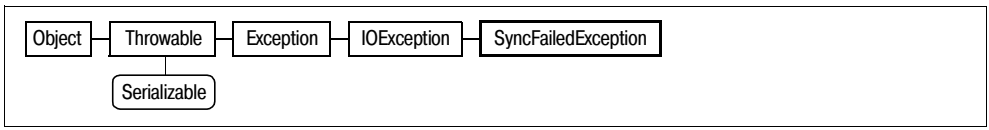
Этот класс является выходным потоком символов, который использует внутренний объект StringBuffer в качестве пункта назначения для символов, записанных в поток. При создании StringWriter вы можете указать внутренний размер StringBuffer. Однако вы не указываете сам StringBuffer; управление этим объектом происходит внутри StringWriter, а по мере необходимости StringBuffer увеличивается, чтобы вмещать записываемые в него символы. StringWriter определяет стандартные методы write(), flush() и close(), которые определяются всеми подклассами Writer, а также два метода, необходимые для получения символов, записанных во внутренний буфер потока. toString() возвращает содержимое внутреннего буфера в виде объекта String, а getBuffer() возвращает сам буфер. Отметим, что getBuffer() возвращает ссылку на реальный внутренний буфер, а не на его копию, а любые изменения, которые вы внесете в буфер, будут отражены в последующих запросах к toString(). StringWriter довольно похож на CharArrayWriter, но не имеет байтового аналога.


```
public class StringWriter extends Writer {
// Открытые конструкторы
   public StringWriter();
   public StringWriter(int initialSize);
// Открытые методы экземпляра
   public StringBuffer getBuffer();
// Открытые методы, замещающие методы класса Writer
   public void close() throws IOException;                    // пустой
   public void flush();                                       // пустой
   public void write(int c);
   public void write(String str);
   public void write(String str, int off, int len);
   public void write(char[] cbuf, int off, int len);
// Открытые методы, замещающие методы класса Object
   public String toString();
}
```

SyncFailedException Java 1.1

java.io ***сериализуемое, проверяемое***

Это исключение свидетельствует о том, что запрос к FileDescriptor.sync() не был завершен успешно.




```
public class SyncFailedException extends IOException {
// Открытые конструкторы
   public SyncFailedException(String desc);
}
```

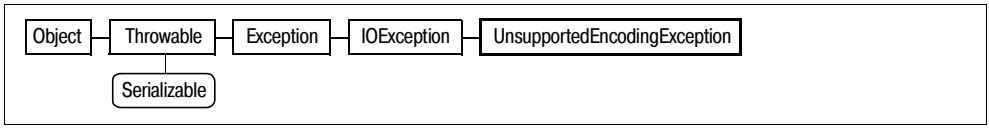
Генерируется методами: `FileDescriptor.sync()`

UnsupportedEncodingException

Java 1.1

java.io ***сериализуемое, проверяемое***

Это исключение свидетельствует о том, что запрошенная кодировка символов не поддерживается текущей виртуальной машиной Java.




```
public class UnsupportedEncodingException extends IOException {
// Открытые конструкторы
   public UnsupportedEncodingException();
   public UnsupportedEncodingException(String s);
}
```

Генерируется методами:

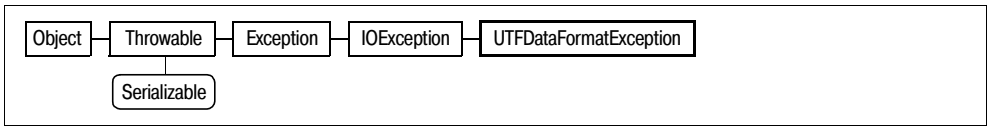
`ByteArrayOutputStream.toString()`, `InputStreamReader.InputStreamReader()`, `OutputStreamWriter.OutputStreamWriter()`, `PrintStream.PrintStream()`, `String.{getBytes(), String()}`, `java.net.URLDecoder.decode()`, `java.net.URLEncoder.encode()`, `java.util.logging.Handler.setEncoding()`, `java.util.logging.StreamHandler.setEncoding()`

UTFDataFormatException

Java 1.0

java.io ***сериализуемое, проверяемое***

Это исключение – `IOException`. Оно свидетельствует о том, что искаженная строка UTF-8 была обнаружена классом, который реализует интерфейс `DataInput`. UTF-8 является преобразовательным форматом, совместимым с ASCII. Он используется для хранения и передачи Unicode-текста.




```
public class UTFDataFormatException extends IOException {
// Открытые конструкторы
   public UTFDataFormatException();
   public UTFDataFormatException(String s);
}
```

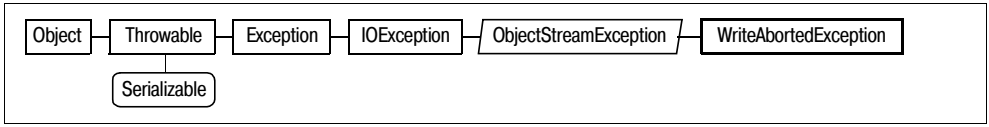
WriteAbortedException

Java 1.1

java.io ***сериализуемое, проверяемое***

Это исключение генерируется при прочтении потока данных, который является неполным, поскольку во время его записи было сгенерировано исключение. Поле `detail` может содержать исключение, которое прервало создание выходного потока. В Java 1.4

и последующих версиях это исключение может быть также получено при помощи стандартного метода Throwable getCause(). Метод getMessage() был замещен, чтобы включать сообщение этого исключения detail, если таковое имеется.




```
public class WriteAbortedException extends ObjectStreamException {
// Открытые конструкторы
   public WriteAbortedException(String s, Exception ex);
// Открытые методы, замещающие методы класса Throwable
1.4 public Throwable getCause();
   public String getMessage();
// Открытые поля экземпляра
   public Exception detail;
}
```

Writer Java 1.1

java.io

Данный абстрактный класс является родительским классом всех выходных потоков символов. Это аналог OutputStream, который является родительским классом всех выходных потоков байтов. Writer определяет базовые методы write(), flush() и close(), предоставляемые всеми выходными потоками символов. Пять версий метода write() записывают один символ, массив, подмассив символов, строку или подстроку в пункт назначения потока. Наиболее общая версия этого метода записывает указанную часть массива символов. Это абстрактная версия, которая должна быть реализована всеми подклассами. По умолчанию другие методы write() реализуются сообразно с этой абстрактной версией. Метод flush() является еще одним абстрактным методом, который должны реализовывать все подклассы. Любой вывод, буферизированный потоком, записывается в его пункт назначения. Если такой пункт назначения сам является выходным потоком байтов или символов, то он также должен вызывать метод flush(), принадлежащий потоку пункта назначения. Метод close() тоже является абстрактным. Подкласс должен реализовать этот метод, чтобы он сбрасывал и затем закрывал текущий поток, а также закрывал любой поток пункта назначения, к которому он присоединен. Как только поток закрыт, любые последующий запросы к write() или flush() должны генерировать IOException.

```
public abstract class Writer {
// Защищенные конструкторы
   protected Writer();
   protected Writer(Object lock);
// Открытые методы экземпляра
   public abstract void close() throws IOException;
   public abstract void flush() throws IOException;
   public void write(String str) throws IOException;
   public void write(char[] cbuf) throws IOException;
   public void write(int c) throws IOException;
   public void write(String str, int off, int len) throws IOException;
   public abstract void write(char[] cbuf, int off, int len) throws IOException;
```

```
// Защищенные поля экземпляра
   protected Object lock;
}
```

Подклассы: BufferedWriter, CharArrayWriter, FilterWriter, OutputStreamWriter, PipedWriter, PrintWriter, StringWriter

Передается методам: Методов слишком много, чтобы их перечислить.

Возвращается методами: java.nio.channels.Channels.newWriter(), java.sql.Clob.setCharacterStream(), javax.swing.text.AbstractWriter.getWriter(), javax.xml.transform.stream.StreamResult.getWriter()

Экземпляры: FilterWriter.out, PrintWriter.out