1. Специализация: тонкости работы

Экран	Слова
Титул	Здравствуйте, отвлечёмся от фундаментальных механик
Отбивка	и немного поговорим о практической составляющей программирования
На прошлом уроке	На прошлой лекции, разобрались с понятиями внутренних и вложенных классов; процессами создания, использования и расширения перечислений. Посмотрели на исключения с точки зрения ООП, обработали немного, разделили понятия штатных и нештатных ситуаций. Детально разобрали понятие исключений, их, скажем так, философию и тесную связь с многопоточностью в джава.
На этом уроке	В качестве ставшего традиционным общеобразовательного отступления поговорим о файловых системах и представлении данных в запоминающих устройствах; Посмотрим на популярные пакеты ввода-вывода java.io, java.nio, без особенного погружения в их недра, потому что там начинаются всякие сложности. Подробно разберём один из самых популярных ссылочных типов данных String и разные механики вокруг него, такие как стрингбилдер и стринг-пул. Получается, На этом уроке мы начнём с файловой системы, как с ней взаимодействовать в целом, с файлами и каталогами. Узнаем как управлять содержимым файлов, затем перейдём к разбору класса String, а так же частично разберёмся с сериализаторами и десериализаторами в Java.
отбивка файловая система	Настало время разобраться в тонкастях работы языка в части общения программы со внешним миром, а делать это не разобравшись в тонкостях работы файловой системы не совсем эффективно. Далее последует материал, напрямую не относящийся к Java. А если оно напрямую не относится к языку, зачем мы вообще об этом говорим? знать о некоторых различиях нужно, чтобы не удивляться тому, что в одной ОС программа работает, а в другой вылетает с каким-нибудь грустным исключением. Поэтому, если будет слишком уж туго заходить не напрягайтесь, сегодня наша задача заложить такой фундамент, чтобы когда в работе у вас возникнет проблема, вы где-то в подсознании припомнили, что кто-то вам что-то такое рассказывал, пойдёте в поисковик и уже целенаправленно отыщете ответ на свой вопрос.

Экран	Слова
Что такое файловая система? ФС - один из ключевых компонентов всех операционных систем. В ее обязанности входит структуризация, чтение, хранение, запись файлов. Выделяют Windows-подход и Linux-подход	Перед тем, как приступить к изучению классов, которые работают с файловой системой, предлагаю разобраться вообще с понятием файловой системы, что она из себя представляет и как с ней взаимодействовать. файловая система (File System) - FS, ФС – один из ключевых компонентов всех операционных систем. В ее обязанности входит структуризация, чтение, хранение, запись файловой документации. Она напрямую влияет на физическое и логическое строение данных, особенности их формирования, управления, допустимый объем файла, количество символов в его названии и прочие. В качестве примеров и в какой-то степени антагонистов идеологий работы с файлами выступают две операционные системы Linux и Windows.
Linux на этапе установки предоставляет выбор из огромного числа ФС. Также возможно самостоятельно выбрать разделы жёсткого диска и подобрать ФС строго под свои нужды.	В ядре ОС Линукс предусмотрен огромный набор заблаговременно установленных файловых систем. Их задача – помогать пользователю в решении той или иной поставленной задачи. Для определенного раздела можно выбирать свою систему, ориентируясь на предстоящие потребности: обеспечение быстродействия, гарантированное восстановление информации, повышенная производительность. Речь идет как о стандартных, так и о специализированные либо же виртуальных файловых системах. В ОС Линукс еще на этапе установки пользователю предоставляется на выбор большое количество ФС, вмонтированных в ее ядро. Пользователь самостоятельно выбирает вариант, который будет соответствовать его запросам и проблемам, требующим решения в рабочем процессе. Обратите внимание: подобное актуально только для операционной системы Linux и Windows NT (с файловой системой сравнительно новой технологии NTFS). Для обычного Windows не предусмотрено возможности выбора вида файловых систем. Отличными здесь будут и строение каталога, и иерархия самих ФС.

Экран	Слова
Разделы GPT и MBR	ОС Линукс также предоставляет возможность разбивки жесткого диска вашего компьютера на отдельные разделы. Пользователи могут определить их границы по так называемым таблицам разделов – GPT, MBR. Основная загрузочная запись (MBR) и таблица разделов GUID (GPT) — это два стиля формата разделов, которые позволяют вашему компьютеру загружать операционную систему с жесткого диска, а также индексировать и упорядочивать данные. Для большинства людей предпочтительным стилем разделов должен быть GPT — более новый из двух. Однако не всегда всё бывает просто. Основная мысль, которую следует запомнить, что у старого загрузчика есть довольно много ограничений, которых лишён новый, поэтому по возможности следует его обновить.
Основная загрузочная запись (MBR) 05-МБР	Основная загрузочная запись (MBR) — это устаревшая форма разделения загрузочного сектора. Это первый сектор диска, который содержит информацию о том, как разбит диск. Он также содержит загрузчик, который сообщает вашей машине, как загрузить ОС. Main Boot Record состоит из трех частей: Основной загрузчик; Таблица разделов диска; Конечная подпись.
05- МБР-состав	Итак, основной загрузчик. MBR резервирует первые байты дискового пространства для основного загрузчика. Windows размещает здесь очень упрощенный загрузчик, в то время как другие ОС могут размещать более сложные многоступенчатые загрузчики. Таблица разделов диска находится в 0м цилиндре, 0й головке и 1м секторе жесткого диска. Она хранит информацию о том, как разбит диск. MBR выделяет 16 байт данных для каждой записи раздела и может выделить всего 64 байта. Таким образом, Main Boot Record может адресовать не более 4 основных разделов или 3 основных раздела и 1 расширенный раздел. Расширенный раздел используется для создания нескольких логических разделов. Это полезно, когда пользователю нужно создать более четырёх. Однако, операционная система может быть установлена только в основных разделах, а не в логических. Конечная подпись - это 2-байтовая подпись, которая отмечает конец MBR. Всегда устанавливается в шестнадцатеричное значение 0x55AA. Вообще, программисты любят всякие 16-теричные подписи вроде 0xDEADBEEF, 0xA11F0DAD, 0xDEADFA11, одну из таких подписей мы видели на самой первой лекции, помните 0xcafebabe? они называются hexspeak.

Экран	Слова
05-МБР-особенности	Main boot record обладает некоторыми особенностями, такими как: Возможность инициализировать загрузчик в устаревшем режиме BIOS; Может адресовать до 2 ТБ дискового пространства; Может иметь 4 основных раздела или 3 основных раздела и 1 дополнительный раздел; Может использоваться для загрузки Windows 7 и более ранних версий Windows. Из плюсов можно отметить Совместимость со всеми версиями Windows, включая Windows 7 и более ранние версии; Требуется для обеспечения совместимости со старым 32-разрядным оборудованием; Использует 32-битные значения, поэтому имеет меньшие накладные расходы, чем GPT. К минусам можно отнести Максимальную емкость раздела всего в 2 ТБ; Ограничение 4мя основными разделами или 3 основными разделами и 1 расширенным разделом; Не устойчивость к повреждению; Отсутствие встроенного исправления ошибок для защиты данных, поскольку использует BIOS.
Вопросы для самопроверки	MBR - это:(1) main boot record; master BIOS recovery; minimizing binary risks
GUID Partition Table (GPT) United Extensible Firmware Interface (UEFI)	Таблица разделов GUID (GPT) — это стиль формата раздела, который был представлен в рамках инициативы United Extensible Firmware Interface (UEFI). GPT был разработан для архитектурного решения некоторых ограничений MBR. Стиль GPT новее, гибче и надежнее, чем MBR. GPT использует логическую адресацию блоков для указания блоков данных. Первый блок помечен как LBA0, затем LBA1, LBA2, и так далее GPT хранит защитную запись MBR в LBA0, свой основной заголовок в логическом блоке и записи разделов в блоках со второго по тридцать третий.

Экран	Слова
Структура GPT: (постепенное появление) — Защитная MBR; — Основной тег GPT; — Записи разделов; — Дополнительный GPT. (и эти разом) — Максимальная емкость раздела 9.4ZB; — 128 первичных разделов; — Устойчив к повреждению первичного раздела, так как имеет вторичный GPT; — Возможность использовать функции UEFI.	GPT состоит из: Защитной основной загрузочной записи. Защитная MBR — это пространство, зарезервированное в таблице разделов для устаревших целей. Оно находится в нулевом логическом блоке адресации. Система, которая не распознает новую таблицу разделов, скорее всего, захочет перезаписать диски с такой таблице, наличие же защитного раздела MBR обеспечивает обратную совместимость с системами, которые не распознают GPT. Защитная MBR охватывает либо весь диск, либо 2 ТБ, в зависимости от того, что меньше. Далее основной тег таблицы разделов, охватывающий с первого по 33й логические блоки адресации. LBA1 состоит из основного заголовка GUID Partition Table, который содержит указатель на таблицу разделов. Он также определяет объем свободного места на диске. Разделительные блоки - это используемые блоки диска, отформатированные в разделе в стиле GUID Partition Table, где хранятся фактические данные. На диске с 512-байтовыми секторами первый используемый блок это 34й логический блок. Схема таблицы разделов требует, чтобы копия основного GPT хранилась в последних секторах диска. Это обеспечивает избыточность схемы GPT, которую можно использовать в качестве резервной на случай повреждения или сбоя основного GPT. Из явных плюсов, Максимальная емкость раздела 9.4ZB (Зеттабайт); Максимум 128 первичных разделов; Устойчив к повреждению первичного GPT, поскольку он также имеет вторичный GPT; Возможность использовать функции UEFI, такие как безопасная загрузка, быстрый запуск и т. п.

Экран	Слова
MBR vs. GPT (постепенное появление) 1. Требования к прошивке: BIOS vs UEFI 2. Поддержка Windows: x32 vs x64 3. Максимальная ёмкость раздела: (232-1) x 512 байт = 2,19ТБ vs (264-1) x 512 байт = 9,44 ZB 4. Количество разделов: 4 (или 3+1) vs 128 5. Скорость загрузки: BIOS vs UEFI 6. Безопасность данных	Резюмируя вышесказанное можно сказать, что различия между MBR и GPT заключаются в следующем: Прошивка — это программное обеспечение, обеспечивающее низкоуровневое управление аппаратным устройством и встроенное в само устройство. Базовая система ввода-вывода (BIOS) и унифицированный расширенный интерфейс встроенного ПО (UEFI) — это две встроенные программы, которые сегодня широко распространены в компьютерах. Для работы MBR требуется устаревшая прошивка BIOS, в то время как GPT, это часть спецификации UEFI. Если ваш диск разбит на разделы MBR, Windows предоставляет инструмент «diskpart» для преобразования его в GPT без потери данных. Windows 7 и более ранние версии Windows, работающие на 32-разрядных компьютерах, совместимы только с дисками с разделами MBR. Windows 8 и более поздние версии могут использовать диски с разделами GPT и MBR. 64-разрядные версии более ранних версий Windows могут читать и записывать с дисков с разделами GPT, но не могут загружаться с них. Максимальный размер диска, который может адресовать раздел MBR, ограничен 2 ТБ. Это связано с тем, что MBR хранит адреса и размеры блоков в таблице разделов с использованием 32-битных данных. С другой стороны, таблица разделов GPT может использовать 64-битные адреса. Таким образом, теоретический максимальный размер диска с разделами GPT составляет более 9ZB. Следует также отметить, что файловые системы Windows в настоящее время ограничены 256ТБ каждая. MBR выделяет 16 байт данных для каждой записи раздела и может выделить всего 64 байта. Таким образом, MBR может адресовать не более 4 основных разделов или 3 основных раздела и 1 расширенный раздел. Вы можете иметь неограниченное количество логических разделов внутри расширенного, однако вы можете установить ОС только в основной. Раздел GPT, с другой стороны, теоретически может иметь неограниченное количество первичных разделов. Несмотря на то, что его реализация в Windows ограничена только 128, каждый из них однако может быть основным. Хотя ни разделы MBR, ни разделы GPT не предназначены для работы быстрее друг друга, между ними может быть некоторая разница в скорости загрузки. Это связано с тем, что MBR использует устаревший BIOS, а GPT использует UEFI. MBR — это простая схема таблицы разделов, которая объединяет загрузочные данные и разделы. Таким образом, разделы MBR имеют более высокую вероятность потери данных в случае повреждения раздела. GPT разделяет таблицу разделов и блоки данных, что обеспечивает более надежную конфигурацию. Также, GPT имеет функцию безопасной загрузки, которая чуть лучше предотвращает захват процесса загрузки вредоносными программами

Экран	Слова
Вопросы для самопроверки	Что такое GPT?(2) General partition trace; GUID partition table; Greater pass timing
отбивка файловые системы	теперь мы знаем, что такое загрузчики, а значит понимаем, что устройство компьютера это не только монитор мышка и клавиатура. Настало время взглянуть на особенности организации файловой системы в операционной системе.
ФС - это архитектура хранения информации, размещенной на жестком диске и в оперативной памяти	файловая система всякой ОС, включая Linux – это некая архитектура хранения информации, размещенной на жестком диске и в оперативной памяти. С ее помощью пользователь получает доступ к структуре ядра системы. Он же отвечает за размещение файлов во всех разделах, поддерживая актуальную для него структуру, формирует правила для ее генерации, управляет блоками, исходя из особенностей определенного типа файловой системы.
отбивка Как обстоят дела в Linux	Начинаем рассмотрение всяких непонятных явлений и технологий, традиционно с тех, что попроще.

Экран	Слова
На что влияет файловая система? 1. скорость обработки данных; 2. сбережение файлов; 3. оперативность записи; 4. допустимый размер блока; 5. возможность хранения информации в оперативной памяти; 6. способы корректировки пользователями ядра 7. и пр.	ОС Линукс предоставляет возможность устанавливать в каждый отдельный блок свою файловую систему, которая и будет обеспечивать порядок поступающих и хранящихся данных, поможет с их организацией. Каждая ФС работает на наборе правил. Исходя из этого и определяется в каком месте и каким образом будет выполняться хранение информации. Эти правила лежат в основе иерархии системы, то есть всего корневого каталога. От того, насколько правильно администратор компьютера выберет тип файловой системы для каждого раздела, зависит ряд параметров, таких как: оперативность записи; скорость обработки данных; сбережение файлов; допустимый размер блока; возможность хранения информации в оперативной памяти; и другие

Экран	Слова
ФС. Категории: журналируемые, не журналируемые. — Ext (extended) FS; — Ext2, 3, 4; — JFS, ReiserFS, XFS, Btrfs, F2FS; — EncFS, Aufs, NFS; — Tmpfs, Procfs, Sysfs;	Все файловые системы Linux, которые применяются сегодня можно разделить на 2 отдельные категории: Журналируемые. Сохраняющие историю манипуляций пользователя и позволяющие её посмотреть, выполнить диагностику системы в отдельном специальном файле. Отличаются повышенной стойкостью к сбоям в функционировании, сохранностью целостности данных. и Не журналируемые. Здесь не предусмотрено сбережение логов, нет гарантий сохранности информации. Но зато в работе такие файловые системы более быстрые. При установке на компьютер операционной системы, пользователь сможет остановить выбор на одной из множества файловых систем представленных на слайде.
Файлы в Linux (последовательное появление) — Regular File — Device File — Soft Link — Directories — Block devices and sockets	В файловой системе, что логично, хранятся файлы. Файлы в линукс - это особенная отдельная категория данных. Если коротко, с точки зрения ОС - всё файл. И все файлы, следовательно, делятся на категории. Обычные. Предназначены для хранения двоичной и символьной информации. Это жесткая ссылка, ведущая на фактическую информацию, размещенную в каталоге. Если этой ссылке присвоить уникальное имя, получим Named Pipe, то есть именованный канал. файлы для устройств, туннелей. Речь идет о физических устройствах, представленными в Линукс файлами. Могут классифицировать специальные символы и блоки. Обеспечивают мгновенный доступ к дисководам, принтерам, модемам, воспринимая их как простой файл с данными. мягкая (символьная) ссылка. Отвечает за мгновенный доступ к файлам, размещенным на любых носителях информации. В процессе копирования, перемещения и других действий пользователя, работающего по ссылке, будет выполняться операция над документом, на который ссылаются. Каталоги. Обеспечивают быстрый и удобный доступ к каталогам. Представляет собой файл с директориями и указателями на них. Это некого рода картотека: в папках размещаются документы, а в директориях – дополнительные каталоги. Блочные и символьные устройства. Выделяют интерфейс, необходимый для взаимодействия приложений с аппаратной составляющей. Каналы и сокеты. Отвечают за взаимодействие внутренних процессов в операционной системе.

Экран	Слова
отбивка Как обстоят дела в Windows	рассмотрим всю линейку файловых систем для Windows, чтобы понять, какую роль они играют в работе системы и как они развивались в процессе становления этой операционной системы
Файловая система FAT(16) File Allocation Table (с англ. таблица размещения файлов) MS-DOS 3.0, Windows 3.x, Windows 95, Windows 98, Windows NT/2000	была разработана достаточно давно и предназначалась для работы с небольшими дисковыми и файловыми объемами, простой структурой каталогов. Это таблица размещается в начале тома, причем хранятся две ее копии (в целях обеспечения большей устойчивости). Данная таблица используется операционной системой для поиска файла и определения его физического расположения на жестком диске. В случае повреждения и таблицы и ее копии чтение файлов операционной системой становится невозможно. Она просто не может определить, где какой файл, где он начинается и где заканчивается. В таких случаях говорят, что файловая система «упала». система изначально разрабатывалась компанией Microsoft для дискет. Только потом её стали применять для жестких дисков. Сначала это была FAT12 (для дискет и жестких дисков до 16 МБ), а потом она переросла в FAT16, которая была введена в эксплуатацию с операционной системой MS-DOS 3.0. Далее она поддерживается в Windows 3.x, Windows 95, Windows 98, Windows NT/2000 и остальных.
Файловая система FAT32 — возможность перемещения корневого каталога — возможность хранения резервных копий	Начиная с Windows 95, компания Microsoft начинает активно использовать в своих операционных системах FAT32 - тридцатидвухразрядную версию FAT. Технический прогресс не стоит на месте и возможностей FAT стало явно недостаточно. FAT32 стала обеспечивать более оптимальный доступ к дискам, более высокую скорость выполнения операций ввода/вывода, а также поддержку больших файловых объемов (объем диска до 2 Тбайт). Из-за использования более мелких кластеров, выгода по сравнению с FAT16 составляет порядка 10%. Кроме того, необходимо отметить, что FAT32 обеспечивает более высокую надежность работы и более высокую скорость запуска программ. Обусловлено это двумя существенными нововведениями: Возможностью перемещения корневого каталога и резервной копии FAT (если основная копия получила повреждения); Возможностью хранения резервной копии системных данных.

Экран	Слова
Файловая система NTFS — от англ. New Technology File System, файловая система новой технологии — права доступа — Шифрование данных — Дисковые квоты — Хранение разреженных файлов — Журналирование	Ни одна из версий FAT не обеспечивает хоть сколько-нибудь приемлемого уровня безопасности. Это, а также необходимость в добавочных файловых механизмах (сжатия, шифрования) привело к необходимости создания принципиально новой файловой системы. И ею стала файловая система NTFS. В ней для файлов и папок могут быть назначены права доступа (на чтение, на запись и т.д.). Благодаря этому существенно повысилась безопасность данных и устойчивость работы системы. Кроме того, NTFS обеспечивает лучшую производительность и возможность работы с большими объемами данных. Начиная с Windows 2000, используется версия NTFS 5.0, которая, помимо стандартных, позволяет реализовывать Шифрование данных, то есть данные могут быть прочитаны только на компьютере, на котором произошла шифровка. Стало возможно назначать пользователям определенный (ограниченный) размер на диске. Встречаются файлы, в которых содержится большое количество последовательных пустых байтов. Файловая система NTFS позволяет оптимизировать их хранение. Позволяет регистрировать все операции доступа к файлам и томам. При этом, необходимо иметь в виду, что если для файла под NTFS были установлены определенные права доступа, а потом вы его скопировали на раздел FAT, то все его права доступа и другие уникальные атрибуты, присущие NTFS, будут утеряны. Так что будьте бдительны.

Экран	Слова
05-fat	Преимущества NTFS касаются практически всего: производительности, надежности и эффективности работы с данными (файлами) на диске. Так, одной из основных целей создания NTFS было обеспечение скоростного выполнения операций над файлами (копирование, чтение, удаление, запись), а также предоставление дополнительных возможностей: сжатие данных, восстановление поврежденных файлов системы на больших дисках и т.д. Другой основной целью создания NTFS была реализация повышенных требований безопасности, так как файловые системы FAT, FAT32 в этом отношении вообще никуда не годились. В NTFS можно разрешить или запретить доступ к какому-либо файлу или папке (разграничить права доступа). Файловая система FAT для современных жестких дисков просто не подходит (ввиду ее ограниченных возможностей). Что касается FAT32, то ее еще можно использовать, но уже с натяжкой. Если купить жесткий диск на 1000 ГБ, то вам придется разбивать его как минимум на несколько разделов. А если вы собираетесь заниматься видеомонтажом, то вам будет очень мешать ограничение в 4 Гб как максимально возможный размер файла. Всех перечисленных недостатков лишена файловая система NTFS. Так что, даже не вдаваясь в детали и специальные возможности файловой системы NTFS, можно сделать выбор в ее пользу. Последние версии ОС Windows также начали поддерживать некоторые другие ФС, по крайней мере на уровне использования внешних запоминающих устройств.
Вопросы для самопроверки	Открытый вопрос: Что такое файловая система? (способ разбиения диска на сегменты, чтобы записывать данные). Возможно ли использовать разные файловые системы в рамках одной ОС? (да, главное, чтобы диск был корректно разделён)
отбивка Файловая система и представление данных	До появления Java 7 все операции проводились с помощью класса File. Немного разберём его, а потом посмотрим, на что его заменили.

Экран	Слова
лайвкод 05-Класс-File	В Java есть специальный класс (File), с помощью которого можно управлять файлами на диске компьютера. Для того чтобы управлять содержимым файлов, есть другие классы: FileInputStream, FileOutputStream и другие, которые мы рассмотрим позже. Под словами управлять файлами я подразумеваю, что их можно создавать, удалять, переименовывать, узнавать их свойства и еще много чего. Практически во все классы, которые работают с содержимым файла (читают, пишут, изменяют), можно передавать объект класса File. File file = new File(”file.txt”); Здесь мы видим привычное создание объекта, причём в параметре конструктора задано только имя и так называемое расширение файла, а значит файл будет открыт там, где исполняется программа.
05-директории	представим, что нужно вывести на экран список всех файлов, которые находятся в определенной директории. File folder = new File(”.”); for (File file : folder.listFiles()) System.out.println(file.getName()); Такой код выведет на экран всё содержимое текущей директории, о чём говорит точка в строке с именем файла. Если просто указать имя файла, то будет использован файл в той же папке, что исполняемая программа, а если указана точка, то это означает использование непосредственно данной папки. метод listFiles() возвращает список файлов из указанной папки. А метод getName() выдаёт имя файла с расширением.
05-манипуляции-x-2	Вкратце поговорим о методах, которые доступны для объекта класса файл при работе с файлами и директориями, кроме рассмотренных на предыдущем слайде. В первую очередь, можно проверить, является ли объект файлом или директорией. Далее возможно узнать размер файла в байтах и его абсолютный путь (в примерах ранее мы пользовались относительным), порядковый номер жёсткого диска, на котором расположен файл. Манипулировать файлом мы тоже можем, например, удалить его (метод, кстати, возвращает истину или ложь, как результат, то есть удалось ли действие), а потом проверить, существует ли такой файл. Ну и так, например, можно узнать, сколько ещё свободного места доступно на диске, в директории которого мы находимся. Результаты работы методов мы видим на слайде. Могло показаться, что класс предоставляет достаточное количество функционала для работы с файловой системой, но это не совсем так, сейчас узнаем, почему. Также стоит сказать о том, что фактически у класса File почти все методы дублированы: одна версия возвращает String, вторая File

Экран	Слова
отбивка Классы Paths, Path, Files, FileSystem в Java 7 и позднее	В Java 7 создатели языка решили изменить работу с файлами и каталогами
05-файловая-система	Это произошло из-за того, что у класса File был ряд недостатков. Например, в нем не было метода copy(), который позволил бы скопировать файл из одного места в другое (казалось бы, довольно часто необходимая функция). Кроме того, в классе File было достаточно много методов, которые возвращали boolean-значения. При ошибке такой метод возвращает false, а не выбрасывает исключение, что делает диагностику ошибок и установление их причин достаточно непростым делом. Вместо единого класса File появились целых 3 класса: Paths, Path и Files. Если совсем-совсем точно, то Path — это интерфейс, а не класс, а интерфейсы мы рассмотрим буквально на следующей лекции. Также появился класс FileSystem, который предоставляет интерфейс к файловой системе. Файловая система работает как фабрика для создания различных объектов (Path, PathMatcher, Files). Этот объект помогает получить доступ к файлам и другим объектам в файловой системе. Например, можем получить все корневые каталоги и на этом пока остановимся.
URI - Uniform Resource Identifier (унифицированный идентификатор ресурса) URL - Uniform Resource Locator (унифицированный определитель местонахождения ресурса) URN - Unifrorm Resource Name (унифицированное имя ресурса) 05-путь	Разберемся с оставшимися классами уже более подробно, чем они друг от друга отличаются и зачем нужен каждый из них. Начнем с как всегда от простого к сложному. Paths — это совсем простой класс с единственным статическим методом get(). Его создали исключительно для того, чтобы из переданной строки или URI получить объект типа Path. Другой функциональности у него нет. Дополнительно раз и навсегда разделим понятия URI, URL, URN, ай - это идентификатор, Л - это локатор, то есть местонахождение, Н - это имя ресурса.

Экран	Слова
05-нет-файла	говоря о файлах очень сильно хочется сделать небольшое отступление и обратить внимание на FileNotFoundException. Это исключение возникает в случаях, которые нужно обработать на этапе компиляции: Файл по указанному пути не существует или существует, но почему-то недоступен (например, запрошена запись в файл со свойством только для чтения или разрешения файловой системы не позволяют получить доступ к файлу). Как видно на структуре, прекрасно видно всю цепочку наследования. Непосредственно FileNotFoundException наследуется от IOException, который сигнализирует о том, что произошло какое-то исключение ввода-вывода.
05-действия-с-путями	Вернёмся к коду, и, раз уж мы получили объект типа Path, разберёмся, что это за Path такой и зачем он нужен. По большому счету — это переработанный аналог класса File. Работать с ним значительно проще, чем с File. Во-первых, из него убрали многие статические методы, во-вторых, в Path были упорядочены возвращаемые значения методов. В классе File методы возвращали то String, то boolean, то File — разобраться было непросто. Например, метод getParent(), возвращал родительский путь для текущего файла в виде строки. Но при этом есть метод getParentFile(), который возвращал то же самое, но в виде объекта File. Это явно избыточно. Поэтому в Path метод getParent() и другие методы работы с файлами возвращают просто объект Path. Никакой кучи вариантов — все легко и просто. С помощью этого класса можем как получить родителя, так и абсолютный корень пути, проверить начинается ли наш путь с чего-то и не заканчивается ли чем-то. Обратите внимание на то, как работает метод endsWith(). Он проверяет, заканчивается ли текущий путь на переданный путь. Именно на путь, а не на набор символов. В методы startsWith и endsWith() нужно передавать именно путь, а не просто набор символов: в противном случае результатом всегда будет false, даже если текущий путь действительно заканчивается такой последовательностью символов

Экран	Слова
05-очистка-путей	Кроме того, в Path есть группа методов, которая упрощает работу с абсолютными, то есть полными и относительными путями. Проверка на абсолютность в скриптовом ноутбуке не может вернуть истину, а в реальном приложении вполне. Метод normalize() - «нормализует» текущий путь, удаляя из него ненужные элементы. Вы, могли заметить, что в популярных операционных системах при обозначении путей часто используются символы “.” (для обозначения текущей директории”) и “..” (для родительской директории). Так вот, если в нашей программе появился путь, использующий “.” или “..”, метод normalize() позволит удалить их и получить путь, в котором они не будут содержаться. У Path довольно много методов, их вы можете рассмотреть подробнее уже самостоятельно.
05-манипуляции-файл	а пока что перейдём к рассмотрению класса Files. Files — это утилитный класс, куда были вынесены статические методы из класса File. Он сосредоточен на управлении файлами и директориями. Используя статические методы Files, мы можем создавать, удалять, копировать и перемещать файлы и директории. Для этих операций используются соответствующие методы createFile() для файлов, для директорий — createDirectory(), copy(), move() и delete(). для начала, методом createFile() в директории pics создали файл.txt для экспериментов, далее по немного другому пути создаём директорию методом createDirectory(). После этого перемещаем файл методом move() из директории pics в эту новую папку, (здесь можно заметить параметр REPLACE_EXISTING, - то есть нужно выполнить замену существующего файла, если он существует), затем копирование файла из тестовой директории обратно в директорию Java методом copy() (который тоже принимает этот параметр, если нужно), а в конце — удаляется файл и тестовая директория методом delete().

Экран	Слова
05-работа-с-содержимым	класс Files позволяет не только управлять самими файлами, но и работать с его содержимым. Для записи данных в файл у него есть метод write(), а для чтения — целых 3: read(), readAllBytes() и readAllLines() Мы подробно остановимся на методе записи и методе чтения всех строк. Почему именно на нём? Потому что у него есть очень интересный тип возвращаемого значения — List. То есть он возвращает нам список строк файла. Конечно, это делает работу с содержимым очень удобной, ведь весь файл, строку за строкой, можно, например, вывести в консоль в обычном цикле for. Кстати именно так представляет данные в файле текстовый редактор vim. Создадим файлик cat.txt и добавим в него какое-нибудь многострочное описание, а затем прочитаем его из файла обратно. При этом, обратите внимание, что если файл уже существует, создать его не получится, будет выброшено исключение типа FileAlreadyExists, поэтому будет логично после всех наших манипуляций файл удалить.
Вопросы для самопроверки	Какое исключение выбрасывает программа, если не может открыть файл? (filenotfoundexception). Ссылка на местонахождение - это: URI, URL, URN.
отбивка java.io	Полностью отказаться от использования пакета ИО вряд ли получится, поэтому начнём с короткого обзора именно его, ну и ещё потому что он появился раньше.
05-ввод-вывод-без-потоков	Подавляющее большинство программ обменивается данными со внешним миром. Это, безусловно, делают любые сетевые приложения – они передают и получают информацию от других компьютеров и специальных устройств, подключенных к сети. Оказывается, можно точно таким же образом представлять обмен данными между устройствами внутри одной машины. Так, например, программа может считывать данные с клавиатуры и записывать их в файл, или же, наоборот - считывать данные из файла и выводить их на экран. Таким образом, устройства, откуда может производиться считывание информации, могут быть самыми разнообразными – файл, клавиатура, входящее сетевое соединение и т.д. То же касается и устройств вывода – это может быть файл, экран монитора, принтер, исходящее сетевое соединение и т.п. В конечном счете, все данные в компьютерной системе в процессе обработки передаются от устройств ввода к устройствам вывода.

Экран	Слова
05-абстрактный-ввод-вывод	Реализация системы ввода/вывода осложняется не только широким спектром источников и получателей данных, но еще и различными форматами передачи информации. Ею можно обмениваться в двоичном представлении, символьном или текстовом, с применением некоторой кодировки (кодировок только для русского языка их более 4 типов), или передавать числа в различных представлениях. Доступ к данным может потребоваться как последовательный, так и произвольный. Зачастую для повышения производительности применяется буферизация. В Java для описания работы по вводу/-выводу используется специальное понятие потока данных (stream). Поток данных это абстракция, физически, конечно никакие потоки в компьютере не текут. Поток связан с некоторым источником, или приемником, данных, способным получать или предоставлять информацию. Соответственно, потоки делятся на входящие – читающие данные и выходящие – передающие (записывающие) данные. Введение концепции stream позволяет абстрагировать основную логику программы, обменивающейся информацией с любыми устройствами одинаковым образом, от низкоуровневых операций с такими устройствами ввода/вывода, для программы нет разницы, передавать данные в файл или в сеть, принимать с клавиатуры или со специализированного устройства. В Java потоки естественным образом представляются объектами. Описывающие их классы как раз и составляют основную часть пакета java.io. Они довольно разнообразны и отвечают за различную функциональность. Все классы разделены на две части – одни осуществляют ввод данных, другие – вывод.

Экран	Слова
Классы InputStream и OutputStream InputStream – это базовый абстрактный класс для потоков ввода, т.е. чтения. 05-1-input	Как и говорилось, все типы поделены на две группы. Практически все классы пакета ИО осуществляющие ввод-вывод, так или иначе наследуются от InputStream для входных данных, и для выходных – от OutputStream. Input Stream описывает базовые методы для работы со входящими байтовыми потоками данных. Класс довольно подробно задокументирован, при желании можно прочитать информацию перед методом и понять, как именно он работает. Простейшая операция представлена методом read() (без аргументов). Он является абстрактным и, соответственно, должен быть определен в классах-наследниках. Согласно документации, этот метод предназначен для считывания ровно одного байта из потока, однако возвращает при этом значение типа int. В том случае, если считывание произошло успешно, возвращаемое значение лежит в диапазоне от 0 до 255 и представляет собой полученный байт (значение int содержит 4 байта и получается простым дополнением нулями в двоичном представлении). Обратите внимание, что полученный таким образом байт не обладает знаком и не находится в диапазоне от -128 до +127, как примитивный тип byte в Java. Если достигнут конец потока, то есть в нем больше нет информации для чтения, то возвращаемое значение равно -1. Если же считать из потока данные не удается из-за каких-то ошибок, или сбоев, будет брошено исключение java.io.IOException. Этот класс наследуется от Exception, если вспомнить ранее нами разобранную струтуру наследования, т.е. его всегда необходимо обрабатывать явно. Дело в том, что каналы передачи информации, будь то Internet или, например, жесткий диск, могут давать сбои независимо от того, насколько хорошо написана программа. А это означает, что нужно быть готовым к ним, чтобы пользователь не потерял нужные данные. Когда работа с входным потоком данных окончена, его следует закрыть. Для этого вызывается метод close(). Этим вызовом будут освобождены все системные ресурсы, связанные с потоком.
Классы InputStream и OutputStream OutputStream – это базовый абстрактный класс для потоков вывода, т.е. записи.	

Экран	Слова
05-1-output	В классе OutputStream аналогичным образом определяются три метода write() – один принимающий в качестве параметра int, второй – byte[] и третий – byte[], и два int-числа. Все эти методы ничего не возвращают. Для записи в поток сразу некоторого количества байт методу write() передается массив байт. Или, если мы хотим записать только часть массива, то передаем массив byte[] и два числа – отступ и количество байт для записи. Понятно, что если указать неверные параметры – например, отрицательный отступ, отрицательное количество байт для записи, либо если сумма отступ плюс длина будет больше длины массива, – во всех этих случаях кидается исключение IndexOutOfBoundsException. Реализация потока вывода может быть такой, что данные записываются не сразу, а хранятся некоторое время в памяти. Чтобы убедиться, что данные записаны в поток, а не хранятся в буфере, вызывается метод flush(), определенный в OutputStream. Когда работа с потоком закончена, его следует закрыть. Для этого вызывается метод close(). Закрытый поток не может выполнять операции вывода и не может быть открыт заново. В классе OutputStream реализация метода close() не производит никаких действий. Итак, классы InputStream и OutputStream определяют необходимые методы для работы с байтовыми потоками данных. Эти классы являются абстрактными. Их задача – определить общий интерфейс для классов, которые получают данные из различных источников. Такими источниками могут быть, например, массив байт, файл, строка и т.д.

Экран	Слова
Классы ByteArrayInputStream и ByteArrayOutputStream 03-байт-массивы	Самый естественный и простой источник, откуда можно считывать байты, – это, конечно, массив байт. Класс ByteArrayInputStream представляет поток, считывающий данные из массива байт. Этот класс имеет конструктор, которому в качестве параметра передается массив byte[]. Соответственно, при вызове методов read() возвращаемые данные будут браться именно из этого массива. Аналогично, для записи байт в массив применяется класс ByteArrayOutputStream. Этот класс использует внутри себя объект byte[], куда записывает данные, передаваемые при вызове методов write(). Чтобы получить записанные в массив данные, вызывается метод toByteArray(). В этом примере вначале будет создан массив, который состоит из трёх элементов: 1, -1 и 0. Затем, при вызове метода read() данные считывались из массива, переданного в конструктор ByteArrayInputStream. Обратите внимание, в данном примере второе считанное значение равно 255, а не -1, как можно было бы ожидать. Чтобы понять, почему это произошло, нужно вспомнить, что метод read считывает byte, но возвращает значение int, полученное добавлением необходимого числа нулей (в двоичном представлении). Про двоичные представления мы довольно подробно говорили на самой первой лекции. Использовать эти классы может быть очень удобно, когда нужно проверить, что именно записывается в выходной поток. Например, при отладке и тестировании сложных процессов записи и чтения из потоков. Эти классы хороши тем, что позволяют сразу просмотреть результат и не нужно создавать ни файл, ни сетевое соединение, ни что-либо еще.

Экран	Слова
Классы FileInputStream и FileOutputStream 03-файл-потоки	Класс FileInputStream используется для чтения данных из файла. Конструктор такого класса в качестве параметра принимает название файла, из которого будет производиться считывание. При указании строки имени файла нужно учитывать, что она будет напрямую передана операционной системе, поэтому формат имени файла и пути к нему может различаться на разных платформах. Если при вызове этого конструктора передать строку, указывающую на несуществующий файл или каталог, то будет брошено java.io.FileNotFoundException. Если же объект успешно создан, то при вызове его методов read() возвращаемые значения будут считываться из указанного файла. Для записи байт в файл используется класс FileOutputStream. При создании объектов этого класса, то есть при вызовах его конструкторов, кроме имени файла, также можно указать, будут ли данные дописываться в конец файла, либо файл будет полностью перезаписан. Здесь есть очень важный момент, если не указан флаг добавления, то всегда сразу после создания FileOutputStream файл будет создан. При вызовах методов write() передаваемые значения будут записываться в этот файл. По окончании работы необходимо вызвать метод close(), чтобы сообщить системе, что работа по записи файла закончена. При работе с FileInputStream метод available() практически наверняка вернет длину файла, то есть число байт, сколько вообще из него можно считать. Но не стоит закладываться на это при написании программ, которые должны устойчиво работать на различных платформах,– метод available() возвращает число байт, которое может быть на данный момент считано без блокирования. Тот факт, что, скорее всего, это число и будет длиной файла, является всего лишь частным случаем работы на некоторых платформах. В приведенном примере для наглядности закрытие потоков производилось сразу же после окончания их использования в основном блоке. Однако лучше, как мы помним, использовать трай-с-ресурсами, чтобы гарантировать, что поток будет закрыт и будут высвобождены все занимаемые им ресурсы.

Экран	Слова
Другие потоковые классы (последовательное появление) — PipedInputStream и PipedOutputStream — StringBufferInputStream (deprecated) — SequenceInputStream — FilterInputStream и FilterOutputStream и их наследники	Классы PipedInputStream и PipedOutputStream характеризуются тем, что их объекты всегда используются в паре – к одному объекту PipedInputStream привязывается (подключается) один объект PipedOutputStream. Они могут быть полезны, если в программе необходимо организовать обмен данными между модулями (например, между потоками выполнения). Более явно выгода от использования PipedI/OStream в основном проявляется при разработке многопоточных приложений. Иногда бывает удобно работать с текстовой строкой String как с потоком байт. Для этого можно воспользоваться классом StringBufferInputStream. При создании объекта этого класса необходимо передать конструктору объект String. Данные, возвращаемые методом read(), будут считываться именно из этой строки. При этом символы будут преобразовываться в байты с потерей точности – старший байт отбрасывается (напомню, что char и как символ и как тип данных состоит из двух байт). Класс SequenceInputStream объединяет поток данных из других двух и более входных потоков. Данные будут вычитываться последовательно – сначала все данные из первого потока в списке, затем из второго, и так далее. Конец потока SequenceInputStream будет достигнут только тогда, когда будет достигнут конец потока, последнего в списке. Задачи, возникающие при вводе/выводе весьма разнообразны - это может быть считывание байтов из файлов, объектов из файлов, объектов из массивов, буферизованное считывание строк из массивов и т.д. В такой ситуации решение с использованием простого наследования приводит к возникновению слишком большого числа подклассов. Более эффективно применение надстроек (в ООП этот шаблон называется адаптер) Надстройки – наложение дополнительных объектов для получения новых свойств и функций. Таким образом, необходимо создать несколько дополнительных объектов – адаптеров к классам ввода/вывода. В java.io их еще называют фильтрами. Отсюда и название фильтрующих потоков

Экран	Слова
Другие потоковые классы (продолжение)(последовательное появление) — LineNumberInputStream, LineNumberReader, — PushBackInputStream, — PrintStream, PrintWriter	Класс LineNumberInputStream во время чтения данных производит подсчет, сколько строк было считано из потока. Номер строки, на которой в данный момент происходит чтение, можно узнать путем вызова метода getLineNumber(). Также можно и перейти к определенной строке вызовом метода setLineNumber(int lineNumber). Этот класс практически разу объявили устаревшим и вместо него используется LineNumberReader с аналогичным функционалом. Этот фильтр позволяет вернуть во входной поток считанные из него данные. Такое действие производится вызовом метода unread(). Понятно, что обеспечивается подобная функциональность за счет наличия в классе специального буфера – массива байт, который хранит считанную информацию. PrintStream используется для конвертации и записи строк в байтовый поток. В нем определен метод print(), принимающий в качестве аргумента различные примитивные типы Java, а также тип Object. При вызове передаваемые данные будут сначала преобразованы в строку, после чего записаны в поток. Если возникает исключение, оно обрабатывается внутри метода print и дальше не бросается (узнать, произошла ли ошибка, можно с помощью метода checkError()). Данный класс также считается устаревшим, и вместо него рекомендуется использовать PrintWriter, однако старый класс продолжает активно использоваться, поскольку статические поля out и err класса System имеют именно это тип.
BufferedInputStream и BufferedOutputStream 05-сравнение-1	Отдельно стоит сказать о BufferedInputStream и BufferedOutputStream. На практике при считывании с внешних устройств ввод данных почти всегда необходимо буферизировать. BufferedInputStream содержит массив байт, который служит буфером для считываемых данных. То есть когда байты из потока считываются либо пропускаются (методом skip()), сначала заполняется буферный массив, причем, из потока загружается сразу много байт, чтобы не требовалось обращаться к нему при каждой операции read или skip. BufferedOutputStream предоставляет возможность производить многократную запись небольших блоков данных без обращения к устройству вывода при записи каждого из них. Сначала данные записываются во внутренний буфер. Непосредственное обращение к устройству вывода и, соответственно, запись в него, произойдет, когда буфер заполнится. Инициировать передачу содержимого буфера на устройство вывода можно и явным образом, вызвав метод flush(). Для наглядности заполним небольшой файл данными, буквально 10 миллионов символов, не так много.

Экран	Слова
05-сравнение-2	На следующем шаге просто прочитаем эти символы из файла простым потоком ввода из файла, видим, что это заняло сколько-то времени
05-сравнение-3	и видим, что буферизованный поток справляется ровно с той-же задачей на пару порядков быстрее. Выгода использования налицо.
Усложнение данных DataInputStream и DataOutputStream 05-данные-1	До сих пор речь шла только о считывании и записи в поток данных в виде byte. Для работы с другими типами данных Java определены интерфейсы DataInput и DataOutput и их реализации – классы-фильтры DataInputStream и DataOutputStream. Интерфейсы DataInput и DataOutput определяют, а классы DataInputStream и DataOutputStream, соответственно, реализуют методы считывания и записи значений всех примитивных типов. При этом происходит конвертация этих данных в набор byte и обратно. Чтение необходимо организовать так, чтобы данные запрашивались в виде тех же типов, в той же последовательности, как и производилась запись. Если записать, например, int и long, а потом считывать их как short, чтение будет выполнено корректно, без исключительных ситуаций, но числа будут получены совсем другие. Собственно, ничто не мешает нам взглянуть на это на примере.
05-данные-2	Далее прочитаем данные так, как мы записали, видим, что все значения прочитаны корректно
05-данные-3	А затем занимаемся нашим любимым делом - всё ломаем и видим, что всё послушно сломалось. А зачем мы это ломаем? правильно, чтобы сломать сейчас и не ломать потом на работе.
Ещё сложнее ObjectInputStream и ObjectOutputStream	Для объектов процесс преобразования в последовательность байт и обратно организован несколько сложнее – объекты имеют различную структуру, хранят ссылки на другие объекты и т.д. Поэтому такая процедура получила специальное название - сериализация (serialization), обратное действие, – то есть воссоздание объекта из последовательности байт – десериализация. Подробнее изучать мы их будем чуть позже, когда разберёмся с интерфейсами и всякими внешними библиотеками, но не упомянуть их в нашем сегодняшнем контексте было нельзя.

Экран	Слова
Классы Reader и Writer 05-байты-символы Классы-мосты InputStreamReader и OutputStreamWriter	Рассмотренные классы – наследники InputStream и OutputStream – работают с байтовыми данными. Если с их помощью записывать или считывать текст, то сначала необходимо сопоставить каждому символу его числовой код. Такое соответствие называется кодировкой. Так вот, джава предоставляет классы, практически полностью дублирующие байтовые потоки по функциональности, но называющиеся читателями и писателями, соответственно. Различия между байтовыми и символьными классами весьма незначительны. FileInputStream - FileReader, FileOutputStream - FileWriter, BufferedInputStream - BufferedReader, PrintStream - PrintWriter и так далее. Классы-мосты InputStreamReader и OutputStreamWriter при преобразовании символов также используют некоторую кодировку. Ее можно задать, передав в конструктор в качестве аргумента ее название. Если оно не будет соответствовать никакой из известных кодировок, будет брошено исключение.
Вопросы для самопроверки	Возможно ли чтение совершенно случайного байта данных из объекта BufferedInputStream? (да, потому что мы читаем из буфера). Возможно ли чтение совершенно случайного байта данных из потока, к которому подключен объект BufferedInputStream? (нет, потому что потоки всегда только однонаправленные, мы не можем знать, есть ли вообще там данные, которые мы собрались читать)
отбивка java.nio и nio2	Основное отличие между двумя подходами к организации ввода/-вывода в том, что Java IO является потокоориентированным, а Java NIO – буфер-ориентированным
io vs nio 05-потоки-и-буферы	Потокоориентированный ввод/вывод подразумевает чтение или запись из потока и в поток одного или нескольких байт в единицу времени поочередно. Данная информация нигде не кэшируются. Таким образом, невозможно произвольно двигаться по потоку данных вперед или назад. Если вы хотите произвести подобные манипуляции, вам придется сначала кэшировать данные в буфере. Подход, на котором основан Java NIO немного отличается. Данные считываются в буфер для последующей обработки. Вы можете двигаться по буферу вперед и назад. Это дает немного больше гибкости при обработке данных. В то же время, вам необходимо проверять содержит ли буфер необходимый для корректной обработки объем данных. Также необходимо следить, чтобы при чтении данных в буфер вы не уничтожили ещё не обработанные данные, находящиеся в буфере.

Экран	Слова
05-ожидание-чтения	Потоки ввода/вывода (streams) в Java IO являются блокирующими. Это значит, что когда в потоке выполнения вызывается read() или write() метод любого класса из пакета java.io.*, происходит блокировка до тех пор, пока данные не будут считаны или записаны. Поток выполнения в данный момент не может делать ничего другого. Неблокирующий режим Java NIO позволяет запрашивать считанные данные из канала (channel) и получать только то, что доступно на данный момент, или вообще ничего, если доступных данных пока нет. Вместо того, чтобы оставаться заблокированным пока данные не станут доступными для считывания, поток выполнения может заняться чем-то другим.
05-каналы	Каналы – это логические порталы, через которые осуществляется ввод/вывод данных, а буферы являются источниками или приёмниками этих переданных данных. При организации вывода, данные, которые вы хотите отправить, помещаются в буфер, а он передается в канал. При вводе, данные из канала помещаются в предоставленный вами буфер. Также, в Java NIO появилась возможность создать поток, который будет знать, какой канал готов для записи и чтения данных и может обрабатывать этот конкретный канал. Данные считываются в буфер для последующей обработки. Разработчик может двигаться по буферу вперед и назад, что дает нам немного больше гибкости при обработке данных. В то же время нужно проверять, содержит ли буфер необходимый для корректной обработки объем данных. Также не забывать следить, чтобы при чтении данных в буфер не уничтожились еще не обработанные данные, находящиеся там.
05-канал-буфер	Взглянем на пример с использованием буфера и канала. Подготовив всё необходимое мы прочитали данные в буфер, сохранив число прочитанных байт, а затем посимвольно их вывели в консоль. Для чтения данных из файла используется файловый канал. Объект файлового канала может быть создан только вызовом метода getChannel() для файлового объекта, поскольку нельзя напрямую создать объект файлового канала. Кроме FileChannel есть и другие реализации каналов, которые затрагивать пока не будем. При этом, FileChannel нельзя переключить в неблокирующий режим.
отбивка String	Класс String отвечает за создание строк, состоящих из символов.

Экран	Слова
private final char value[]; private final byte[] value; — immutable — final	если быть точнее, заглянув в реализацию и посмотрев способ их хранения, то строки (до Java 9) представляют собой массив символов, а начиная с Java 9 строки хранятся как массив байт. Как я говорил ранее, в первую очередь нас будет интересовать джава8. В отличие от других языков программирования, где символьные строки представлены последовательностью символов, в Java они являются объектами класса String. В результате создания объекта типа String получается неизменяемая символьная строка, т.е. невозможно изменить символы имеющейся строки. При любом изменении строки создается новый объект типа String, содержащий все изменения. А значит у String есть две фундаментальные особенности: это immutable (неизменный) класс; это final класс (у класса String не может быть наследников).
Экземпляр класса String 05-конструкторы-строки ASCII - American standard code for information interchange	Экземпляр класса String можно создать множеством способов. Несмотря на кажущуюся простоту и если можно так сказать, общеупотребимость, класс строки - это довольно сложная структура данных и гигантский набор методов. Одних только конструкторов, вон, больше 6 штук. В примере выше представлена лишь часть возможных вариантов. Самым простым способом является создание строки по аналогии с s1. Обычно строки требуется создавать с начальными значениями. Для этого предоставлены разнообразные конструкторы. При использовании варианта s2 в памяти создается новый экземпляр строки с начальным значением “Home”. Если в программе есть массив символьных строк, то из них можно собрать строку с помощью конструктора, которому в качестве аргумента передается ссылка на массив char[]. Если нужно создать объект типа String, содержащий ту же последовательность символов, что и другой объект типа String, можно использовать вариант s4. Варианты s5 и s6 позволяют «построить» строку из байтового массива без указания кодировки или с ней. Если кодировка не указана, будет использована ASCII.
05-конкатенация	С помощью операции конкатенации, которая записывается, как знак +, можно соединять две символьные строки, порождая в итоге совершенно новый объект типа String. Операции такого сцепления символьных строк можно объединять в цепочку. Ниже приведено несколько примеров. Как видно из примера, строки можно сцеплять с другими типами данных, например, целыми числами(int). Так происходит потому, что значение типа int автоматически преобразуется в своё строковое представление в объекте типа String. После этого символьные строки сцепляются, как и прежде.

Экран	Слова
05-проблема-конкатенации	Сцепляя разные типы данных с символьными строками, следует быть внимательным, иначе можно получить неожиданные результаты. Такой результат объясняется так называемой ассоциативностью оператора сложения. Оператор сложения работает слева направо, а значит сначала будет выполнять то, что слева, а потом то, что справа. Расширение типа до максимального происходит автоматически, так если складывать целое и дробное, в результате будет дробное, если складывать любое число и строку получится строка. А однажды получив строку всё остальное будет тоже становиться строкой и сложение превратится в конкатенацию.
Часто используемые методы строки — int length() — char charAt(int pos) — char[] toCharArray() — boolean equals(Object obj) — boolean equalsIgnoreCase(Object obj) — String concat(String obj) — String toLowerCase(), String toUpperCase()	На слайде представлено несколько часто используемых методов для работы со строками. Это не полный список. Здесь получение длины строки, Извлечение из строки символа, находящегося на позиции pos, индексация символов начинается с нуля, Преобразование строки в массив символов, Посимвольное сравнение строк, Сравнение строк без учета регистра, Объединение двух строк в одну. Этот метод создает новый строковый объект, содержащий вызываемую строку, в конце которой добавляется содержимое параметра строка. Метод выполняет то же действие, что и операция +. Преобразование всех символов строки в нижний регистр (toLowerCase), в верхний регистр (toUpperCase), все их перечислять, конечно же, смысла не имеет, исходный код класса открыт и я рекомендую пройти туда.

Экран	Слова
!!!строго лайвкод!!! StringBuffer, StringBuilder 05-билдер	Поскольку строка это достаточно неповоротливо, придумали прекрасные классы, которые позволяют ускорить работу с ними. Сразу пойдём от практики и объявим переменную String s = “Example”; сделаем цикл, скажем, до 30000, и будем делать одно и тоже 30 тыщ раз. s = s + i; В результате получим строку Example012345... просто какая-то бесполезная очень длинная строчка. Много арифметических действий, много созданий новых строк, а старые будут попадать в маленький ад для Java строк. Шучу, попадут они под сборку мусора. Быстро сделаем более точный замер времени. long timeStart = System.nanoTime(); for (int i = 0; i < 30000; ++i) s = s + i; double deltaTime = (System.nanoTime() - timeStart) * 0.000000001; System.out.println(”Delta time: ”+ deltaTime); Жмём Run Ждём и видим, что наша супер сложная программа выполнялась 0,8. секунд. Давайте сделаем побольше, скажем, сто тысяч итераций, а лучше миллион, просто хочется показать контраст. Пока ждём расскажу немного о самих стрингбилдере и стрингбуфере и разнице между ними. Если у вас в программе намечается какая-то большущая работа со стрингами в каких то гигантских циклах, вспомните про StringBuilder. И помните, что Если у вас что-то “тормозит”, то с 90% вероятностью уже придумали способ это ускорить, и велосипеды изобретать не нужно. Если немного приоткрыть завесу, то все строковые классы работают с массивами чаров, только стринг делает это примитивно, выделяя каждый раз новый кусок памяти под массив с длиной равной длине строки, а билдер сразу выделяет большой кусок памяти под массив, добавляет к нему элементы по индексу, а если массив заканчивается, то тогда делает массив ещё больше, раза в два, копирует в него старый, и заполняет уже его. Схожим образом работает ArrayList, соответственно СБУФЕР работает точно также но его можно использовать в многопоточных программах, но со связными списками и многопоточностью Вы будете работать на следующем этапе обучения. СБУФЕР потокобезопасен. потокобезопасным называют класс, с которым можно работать из разных потоков в рамках одной программы, и быть уверенным, что ничего не сломается. Там ОЧЕНЬ интересно и прикольно, происходит рассинхронизация программы на несколько потоков и начинается самое крутое.

Экран	Слова
	ну вот, у нас получилось достаточно много времени, по ощущениям примерно столько времени и прошло. Ну а теперь к оптимизации. Разработчики Java знали, что перед программистами будут стоять задачки и посерьёзнее, чем обработка 100000 символьных строк, например, при разборе текстовых файлов, и поиске информации в электронных книгах. И придумали StringBuilder и StringBuffer. Создают они изменяемые строки и динамические ссылки на них. Их разница в том, что StringBuilder не потокобезопасный, и работает чуть быстрее, а StringBuffer - используется в многопоточных средах, но в одном потоке работает чуть медленнее. Мы в нашем примере будем использовать StringBuilder. Повторим тоже самое, но только с использованием StringBuilder StringBuilder sb = new StringBuilder(”Example”); long timeStart = System.nanoTime(); for (int i = 0; i < 100_000; ++i) sb = sb.append(i); double deltaTime = (System.nanoTime() - timeStart) * 0.000000001; System.out.println(”Delta time (sec): ”+ deltaTime); В конструктор передали начальное значение нашей строки. То есть теперь это всё ещё стринг, но представленный другим классом, не являющимся тем удобным для большинства нужд String, то есть мы до одной из последних редакции джавы не могли просто взять и например вывести в sout его. или, к примеру, не можем неявно его создать, как мы создаём обычные String, но в нём есть много других весьма полезных методов, которых нет в классе String. Например, метод append, который делает, по сути, тоже самое, что и s=s+” ”; то есть добавляет что то в конец существующей строки. Ну и чтобы удостовериться, что никакого обмана там нет, можем сравнить полученные первым и вторым способом строки. Хотел сказать Ждём ВАУ эффекта, но ждать не придётся, вот он. Разница очевидна. и в результате сравнения строк, можете проверить сами, у нас вернулся true значит строки получились одинаковые. Вот и всё, что вам пока что нужно знать о строках. Надеюсь вы за это время поняли, что String не так то прост. Так вот тоже самое характерно и для всех остальных языков программирования, вообще для всех. Поэтому в чистом Си нет класса String, там есть только массив из чар фиксированной длины, и кому охота работать со стрингами пишет их самостоятельно ручками, думая о выделении памяти и так далее. А для более высокоуровневых языков - каждый разработчик языка пляшет как хочет, поэтому и получается местами сложно, часто не очевидно, и у всех по-разному.
Вопросы для самопроверки	String - это: -объект; -примитив; -класс. Строки в языке Java это: -классы; -массивы; -объекты.

Экран	Слова
отбивка String pool	Строки в бассейне? Бассейн из строк? Бассейн для строк? Ну да, что-то вроде этого.
05-интернирование-строк	Экземпляр класса String хранится в памяти, именуемой куча (heap), но есть некоторые нюансы. Если строка, созданная при помощи конструктора хранится непосредственно в куче, то строка, созданная как строковый литерал, уже хранится в специальном месте кучи — в так называемом пуле строк (string pool). В нем сохраняются исключительно уникальные значения строковых литералов, а не все строки подряд. Процесс помещения строк в пул называется интернирование (от англ. interning, внедрение, интернирование). Когда мы объявляем переменную типа String и присваиваем ей строковый литерал, то JVM обращается в пул строк и ищет там такое же значение. Если пул содержит необходимое значение, то компилятор просто возвращает ссылку на соответствующий адрес строки без выделения дополнительной памяти. Если значение не найдено, то новая строка будет интернирована, а ссылка на нее возвращена и присвоена переменной.
05-пул-и-конкатенация	В строке «Best» + «Cat» создаются два строковых объекта со значениями «Best» и «Cat», которые помещаются в пул. «Склеенные» строки образуют еще одну строку со значением «BestCat», ссылка на которую берется из пула строк (а не создается заново), т.к. она была интернирована в него ранее. Значения всех строковых литералов из данного примера известно на этапе компиляции. А вот если предварительно поместить один из фрагментов строки в переменную, мы можем запутать стрингпул и заставить его думать, что в результате получится совсем новая строка.

Экран	Слова
05-пул-и-конструктор	Когда мы создаем экземпляр класса String с помощью оператора new, компилятор размещает строки в куче. При этом каждая строка, созданная таким способом, помещается в кучу (и имеет свою ссылку), даже если такое же значение уже есть в куче или в пуле строк. Таким образом, создав четыре одинаковых строки, в памяти зафиксируются только три объекта. Согласитесь, что это нерационально. В Java существует возможность вручную выполнить интернирование строки в пул путем вызова метода intern() у объекта типа String. Принимая во внимание всё вышесказанное, вы можете спросить: «Почему бы все строки сразу после их создания не добавлять в пул строк? Ведь это приведет к экономии памяти». Да, среди достаточно большого количества программистов такое заблуждение присутствует. Именно заблуждение, поскольку не все учитывают дополнительные затраты виртуальной машины на процесс интернирования, а также падение производительности, связанное с аппаратными ограничениями памяти, ведь невозможно читать ячейку памяти одновременно бесконечным числом процессов, электроника ещё до этого не доросла. Можно сказать, что интернирование (в виде применения метода intern()) рекомендуется вообще не использовать. Вместо интернирования необходимо использовать дедупликацию, но подробный разговор о ней мы отложим на некоторое время, поскольку мы пока что не изучили много всяких вспомогательных шестерёнок. Если коротко, во время сборки мусора гарбич коллектор проверяет живые (имеющие рабочие ссылки) объекты в куче на возможность провести их дедупликацию. Ссылки на подходящие объекты вставляются в очередь для последующей обработки. Далее происходит попытка дедупликации каждого объекта String из очереди, а затем удаление из нее ссылок на объекты, на которые они ссылаются. Также для отслеживания всех уникальных массивов байт, используемых объектами String, используется хеш-таблица. При дедупликации в этой хеш-таблице выполняется поиск идентичных массивов байт.
На этом уроке мы	Поговорили о файловых системах и представлении данных в запоминающих устройствах; Поверхностно посмотрели на популярные пакеты ввода-вывода. Подробно разобрали один из самых популярных ссылочных типов данных String и разные механики вокруг него, такие как стрингбилдер и стрингпул.

Экран	Слова
практическое задание	В качестве практического задания предлагаю — создать пару-тройку текстовых файлов. Для упрощения (не разбираться с кодировками) внутри файлов следует писать текст только латинскими буквами. — написать метод, осуществляющий конкатенацию переданных ей в качестве параметров файлов; — написать метод поиска слова внутри файла.
Учитесь так, словно вы постоянно ощущаете нехватку своих знаний, и так, словно вы постоянно боитесь растерять свои знания. Конфуций	Надеюсь, на этой лекции вы не уснули от объёма теоретической информации, потому что я наоборот постарался дать как можно больше практических особенностей, которые кажутся естественными и о них особенно не задумываешься. Всего доброго и до новых встреч.