1. Специализация: ООП

Экран	Слова
Титул	Перейдём к интересному: что можно хранить в джаве, как оно там хранится, и как этим манипулировать
На прошлом уроке	На прошлом уроке мы рассмотрели базовый функционал языка, то есть основную встроенную функциональность, такую как математические операторы, условия, циклы, бинарные операторы. Также разобрали способы хранения и представления данных в Java, и в конце поговорили о способах манипуляции данными, то есть о функциях (в терминах языка называющиеся методами)
На этой лекции: классы и объекты; управление памятью; инкапсуляция; наследование; полиморфизм.	После разбора типов данных попробуем с помощью примеров разобраться в таких основополагающих в джава вещах, как классы и объекты, а также с тем, как применять на практике основные принципы ООП: наследование, полиморфизм и инкапсуляцию. Дополнительно поговорим об устройстве памяти в джава.
Класс	Начнём с базового, фундаментального понятия класс.
Чертёж (набросок рисунка) кота	Что такое класс? С точки зрения ООП, класс определяет форму и сущность объекта и является логической конструкцией, на основе которой построен весь язык Java. Наиболее важная особенность класса состоит в том, что он определяет новый тип данных, которым можно воспользоваться для создания объектов этого типа, т.е. класс — это шаблон (чертёж), по которому создаются объекты (экземпляры класса). Для определения формы и сущности класса указываются данные, которые он должен содержать, а также код, воздействующий на эти данные.
Котик и стопки документов	Если мы хотим работать в нашем приложении с документами, то необходимо для начала объяснить приложению, что такое документ, описать его в виде класса (чертежа) Document. Рассказать какие у него должны быть свойства: название, содержание, количество страниц, информация о том, кем он подписан и т.д. В этом же классе мы описываем что можно делать с документами: печатать в консоль, подписывать, изменять содержание, название и т.д. Результатом такого описания и будет класс Document. Однако это по-прежнему всего лишь чертеж.

Экран	Слова
Буквы, написанные по трафарету чтобы было видно сам трафарет. документы, одинаковые по структуре и разные по содержанию.	Если нам нужны конкретные документы, а нам они обязательно нужны, то необходимо создавать объекты: документ №1, документ №2, документ №3. Все эти документы будут иметь одну и ту же структуру (название, содержание, ...), с ними можно выполнять одни и те же действия (печатать, подписать, ...) НО наполнение будет разным (например, в первом документе содержится приказ о назначении работника на должность, во втором, о выдаче премии отделу разработки и т.д.).
03-Структура	Начнём с малого, напишем свой первый класс. Представим, что нам необходимо работать в нашем приложении с котами. Java ничего не знает о том, что такое коты, поэтому нам необходимо создать новый класс (тип данных), и объяснить что же такое кот. Создадим проект, его структура нам не в новизну, и будет иметь вид как на слайде, в мейне пока что простой хелловорлд, а вот с новым файлом кота пойдём разбираться, Создадим его для простоты в том же пакете, что и мейн
лайвкод 03-кот	Начнем описывать класс Cat. как мы все прекрасно помним, имя класса должно совпадать с именем файла, в котором он объявлен, т.е. класс Cat должен находиться в файле Cat.java. Пусть у котов есть три свойства: name (кличка), color (цвет) и age (возраст); и они пока ничего не умеют делать. Класс Cat будет иметь следующий вид. (String name; String color; int age;)
03-экземпляр-кота	Мы рассказали Java что такое коты, теперь если мы хотим создать в нашем приложении конкретного кота, а я напомню, что в 90% случаев мы сильно хотим создавать те или иные экземпляры, следует воспользоваться оператором new Cat(); в основном классе программы. Более подробно разберём, что происходит в этой строке, чуть позже. Пока же нам достаточно знать, что мы создали объект типа Cat (экземпляр класса Cat), и запомнить эту конструкцию. Для того чтобы с ним (экземпляром) работать, можем положить его в переменную, которой дать идентификатор cat1.

Экран	Слова
лайвкод 03-разные-коты	Припоминаем разницу между объявлением, присваиванием и инициализацией, становится понятно, что тут произошло и как ещё можно создавать котов. Припоминаем также, что в переменной не лежит сам объект, а только ссылка на него, подробнее, как и обещал, позже. Объект cat1 создан по чертежу Cat, и значит у него есть поля name, color, age, с которыми можно работать (получать или изменять их значения). Для доступа к полям объекта служит оператор точка, которая связывает имя объекта с именем поля. Например, чтобы присвоить полю color объекта cat1 значение ”Белый нужно выполнить следующий код: cat1.color = ”Белый”; Прошу, не запутайтесь, класс кота мы описали в отдельном файле, а создавать объекты и совершать манипуляции следует в мейне, не может же кот сам себе имя назначить
лайвкод 03-два-кота	Операция-точка служит для доступа к полям и методам объекта по его имени. Рассмотрим пример консольного приложения, работающего с объектами класса Cat. создадим двух котов, один будет белым барсиком 4х лет, второй чёрным мурзиком шести лет, и просто выведем информацию о них в терминал
03-класс-объекты1	Вначале мы создали два объекта типа Cat: cat1 и cat2, соответственно они имеют одинаковый набор полей name, color, age, почему? потому что они принадлежат одному классу, созданы по одному шаблону. Однако каждому из них мы в эти поля записали разные значения. Как видно из результата печати в консоли, изменение значения полей одного объекта, никак не влияет на значения полей другого объекта. Данные объектов cat1 и cat2 изолированы друг от друга. А значит мы делаем вывод о том, поля хранятся в классе, а значения полей хранятся в объектах.
Объекты	Как создавать новые типы данных (классы) мы разобрались, мельком посмотрели и как создаются объекты наших классов
Cat cat1; cat1 = new Cat();	Подробнее разберем как создавать объекты, и что при этом происходит. Создание объекта как любого ссылочного типа данных проходит в два этапа. Мы это помним из рассказа о массивах. Сначала создается переменная, имеющая интересующий нас тип, в нее мы можем записать ссылку на объект. Затем необходимо выделить память под наш объект, создать и положить объект в выделенную часть памяти, и сохранить ссылку на этот объект в памяти в нашу переменную. Для непосредственного создания объекта применяется оператор new, который динамически резервирует память под объект и возвращает ссылку на него, в общих чертах эта ссылка представляет собой адрес объекта в памяти, зарезервированной оператором new.

Экран	Слова
03-разные-коты	В первой строке кода переменная cat1 объявляется как ссылка на объект типа Cat и пока ещё не ссылается на конкретный объект (первоначально значение переменной cat1 равно null). В следующей строке выделяется память для объекта типа Cat, и в переменную cat1 сохраняется ссылка на него. После выполнения второй строки кода переменную cat1 можно использовать так, как если бы она была объектом типа Cat. Обычно новый объект создается в одну строку (Cat cat1 = new Cat()).
Оператор new [квалификаторы] ИмяКласса имяПеременной = new ИмяКласса();	Раз уж начали про объекты - нельзя не сказать про оператор new. Оператор new динамически выделяет память для нового объекта, общая форма применения этого оператора имеет вид как на слайде, но на самом деле справа - не имя класса, как могло показаться на первый взгляд ИмяКласса() в правой части выполняет вызов конструктора данного класса, который подготавливает вновь создаваемый объект к работе.
лайвкод 03-один-кот	Именно от оператора нью будет зависеть, сколько именно объектов будет создано в программе. На первый взгляд может показаться, что переменной cat2 присваивается ссылка на копию объекта cat1, т.е. переменные cat1 и cat2 будут ссылаться на разные объекты в памяти. Но это не так. На самом деле cat1 и cat2 будут ссылаться на один и тот же объект. Присваивание переменной cat1 значения переменной cat2 не привело к выделению области памяти или копированию объекта, лишь к тому, что переменная cat2 содержит ссылку на тот же объект, что и переменная cat1.
03-один-кот-2	Таким образом, любые изменения, внесённые в объекте по ссылке cat2, окажут влияние на объект, на который ссылается переменная cat1, поскольку это один и тот же объект в памяти, как и в примере на слайде, где мы как будто бы указали возраст второго кота 6 лет, а при выводе, возраст 6 лет оказался и у первого кота.
Метод - это функция, принадлежащая классу	Ранее мы уже говорили о том, что в языке Java любая программа состоит из классов и функций, которые могут описываться только внутри них. Именно поэтому все функции в языке Java являются методами. А метод, как мы помним, это - функция, являющаяся частью некоторого класса, которая может выполнять операции над данными этого класса.
03-нестатик	Метод для своей работы может использовать поля объекта и/или класса, в котором определен, напрямую, без необходимости передавать их во входных параметрах. Это похоже на использование глобальных переменных в функциях, но в отличие от глобальных переменных, метод может получать прямой доступ только к членам класса. Самые простые методы работают с данными объектов.

Экран	Слова
Лайвкод 03-метод	Вернёмся к примеру с котиками. Все мы знаем, что котики умеют урчать, мяукать и смешно прыгать. В целях демонстрации мы в описании этих действий просто будем делать разные выводы в консоль, хотя мы и можем научить нашего котика выбирать минимальное значение из массива, но это было бы неожиданно. Итак опишем метод например подать голос и прыгать.
Лайвкод 03-метод-вызов	Обращение к методам выглядит очень похожим на стандартный способом, через точку, как к полям. Теперь когда мы хотим позвать нашего котика, он нам скажет, мяу, я имя котика, а если мы решили что котику надо прыгать, он решит, прилично-ли это - прыгать в его возрасте. Как видно, барсик замечательно прыгает, а мурзик от прыжков воздержался, хотя попрыгать мы попросили их обоих
шрифтом курьер слово static викисловарь - статика этимология	Теперь, когда мы более-менее хорошо знаем, что такое класс и объект, хотелось бы остановиться на специальном модификаторе - static, делающем переменную или метод ”независимыми”от объекта. Если формально, то: Static — модификатор, применяемый к полю, блоку, методу или внутреннему классу, он указывает на привязку субъекта к текущему классу. Для использования таких полей и методов, соответственно, объект создавать не нужно. В Java большинство членов служебных классов являются статическими.
— статические методы; — статические переменные; — статические вложенные классы; — статические блоки.	Мы можем использовать это ключевое слово в четырех контекстах: статические методы; статические переменные; статические вложенные классы; статические блоки. Рассмотрим подробнее только первые два пункта, о третьем поговорим чуть позже, а четвёртый потребует от нас знаний, выходящих не только за этот урок, но и за десяток следующих.
Статические методы - методы класса	Статические методы также называются методами класса, потому что статический метод принадлежит классу, а не его объекту. Что логично, нестатические называются методами объекта. Статические методы обладают интересным свойством - их можно вызывать напрямую через имя класса, не обращаясь к объекту и вообще объект не создавая. Что это и зачем нужно? Например, умение кота мяукать можно вывести в статическое поле, потому что, например, мы весной можем открыть окно, не увидеть ни одного экземпляра котов, но зато услышать их, и точно знать, что мяукают не дома и не машины, а коты

Экран	Слова
03-статические-поля	Аналогично статическим методам, статические поля принадлежат классу и совершенно ничего не знают об объектах. Важной отличительной чертой статических полей является то, что их значения также хранятся в классе, в отличие от обычных полей, чьи значения хранятся в объектах. На слайде довольно понятно это показано. Изображение на слайде именно в этом виде я настоятельно рекомендую если не заучить, то хотя бы хорошо запомнить, оно нам ещё пригодится. Из этой же картинки можно сделать несколько выводов, о которых сейчас поговорим
лайвкод 03-статическое-поле-код	Помимо того, что статические поля - это полезный инструмент создания общих свойств это ещ§ и опасный инструмент создания общих свойств. Так, например, мы знаем, что у котов четыре лапы, а не 6 и не 8. Не создавая никакого барсика будет понятно, что у кота - 4 лапы. Это полезное поведение
лайвкод 03-статическое-поле-ошибка	Посмотрим на опасность. Мы видим, что у каждого кота есть имя, и помним, что коты хранят значение своего имени каждый сам у себя. А знают экземпляры о названии поля потому что знают, какого класса они экземпляры. Но что если мы по невнимательности добавим свойство статичности к имени кота?
03-статическое-поле-признак	Создав тех же самых котов, которых мы создавали весь урок, мы получим двух мурзиков и ни одного барсика. Почему это произошло? По факту переменная у нас одна на всех, и значение тоже одно, а значит каждый раз мы меняем именно его, а все остальные коты ничего не подозревая смотрят на значение общей переменной и бодро его возвращают. Поэтому, чтобы не запутаться, к статическим переменным, как правило, обращаются не по ссылке на объект — cat1.name, а по имени класса — Cat.name.
03-статические-поля	К слову, статические переменные — редкость в Java. Вместо них применяют статические константы. Они определяются ключевыми словами static final и по соглашению о внешнем виде кода пишутся в верхнем регистре.
Введение в управление памятью	Понимая поверхностно, как организовано создание и хранение объектов, можем углубиться в эту тему. Это факультативная часть, выходящая достаточно далеко за рамки джуниор позиции.

Экран	Слова
?	Это глубокое погружение в управление памятью Java позволит расширить ваши знания о том, как работает куча, ссылочные типы и сборка мусора. Поможет лучше понять глубинные процессы и, как следствие, писать более хорошие программы. Для оптимальной работы приложения JVM делит память на область стека (stack) и область кучи (heap). Всякий раз, когда мы объявляем новые переменные, создаем объекты или вызываем новый метод, JVM выделяет память для этих операций в стеке или в куче. Далее будет представлена модель организации памяти в Java. Чуть позже мы её рассмотрим подробнее, а начнем со стека.

Экран	Слова
СТЕК	Стековая память отвечает за хранение ссылок на объекты кучи и за хранение типов значений (также известных в Java как примитивные типы, ниже будут перечислены), которые содержат само значение, а не ссылку на объект из кучи. Данная память в Java работает по схеме LIFO (Последний-зашел-Первый-вышел). Кроме того, переменные в стеке имеют определенную видимость, также называемую областью видимости. Используются только объекты из активной области. Например, предполагая, что у нас нет никаких глобальных переменных (полей) области видимости, а только локальные переменные, если компилятор выполняет тело метода, он может получить доступ только к объектам из стека, которые находятся внутри тела метода. Он не может получить доступ к другим локальным переменным, так как они не выходят в область видимости. Когда метод завершается и возвращается, верхняя часть стека выталкивается, и активная область видимости изменяется. Все потоки, работающие в JVM, имеют свой стек. Стек в свою очередь держит информацию о том, какие методы вызвал поток. Я буду называть это “стеком вызовов”. Стек вызовов возобновляется, как только поток выполняет свой код. Стек потока содержит в себе все локальные переменные, требующиеся для выполнения методов из стека потока. Поток может получить доступ только к своему стеку. Локальные переменные не видны остальным потокам, только потоку, создавшему их. В ситуации, когда два потока выполняют один и тот же код, они оба создают свои локальные переменные. Таким образом, каждый поток имеет свою версию каждой локальной переменной. Все локальные переменные примитивных типов (boolean, byte, short, char, int, long, float, double) полностью хранятся в стеке потоков и не видны другим потокам. Один поток может передать копию примитивной переменной другому потоку, но не может совместно использовать примитивную локальную переменную. Особенности стека: — Он заполняется и освобождается по мере вызова и завершения новых методов; — Переменные в стеке существуют до тех пор, пока выполняется метод в котором они были созданы; — Если память стека будет заполнена, Java бросит исключение java.lang.StackOverFlowError; — Доступ к этой области памяти осуществляется быстрее, чем к куче; — Является потокобезопасным, поскольку для каждого потока создается свой отдельный стек.

Экран	Слова
КУЧА	Куча содержит все объекты, созданные в вашем приложении, независимо от того, какой поток создал объект. К этому относятся и обертки примитивных типов (например, Byte, Integer, Long и так далее). Неважно, был ли объект создан и присвоен локальной переменной или создан как переменная-член другого объекта, он хранится в куче. В случае, когда локальная переменная примитивного типа, она хранится в стеке потока. Локальная переменная также может быть ссылкой на объект. В этом случае ссылка (локальная переменная) хранится в стеке потоков, но сам объект хранится в куче. Объект содержит методы, эти методы содержат локальные переменные. Эти локальные переменные также хранятся в стеке потоков, даже если объект, которому принадлежит метод, хранится в куче. Переменные-члены объекта хранятся в куче вместе с самим объектом. Это верно как в случае, когда переменная-член имеет примитивный тип, так и в том случае, если она является ссылкой на объект. Статические переменные класса также хранятся в куче вместе с определением класса. Эти объекты имеют глобальный доступ и могут быть получены из любого места программы. Эта область памяти разбита на несколько более мелких частей, называемых поколениями: Young Generation — область где размещаются недавно созданные объекты. Когда она заполняется, происходит быстрая сборка мусора; Old (Tenured) Generation — здесь хранятся долгоживущие объекты. Когда объекты из Young Generation достигают определенного порога «возраста», они перемещаются в Old Generation; Permanent Generation — эта область содержит метаинформацию о классах и методах приложения, но начиная с Java 8 данная область памяти была упразднена. В Java 8 PermGen заменён на Metaspace - его динамически изменяемый по размеру аналог. Важно упоминуть, что именно здесь живут статические поля. Особенности кучи: — Когда эта область памяти полностью заполняется, Java бросает java.lang.OutOfMemoryError; — Доступ к ней медленнее, чем к стеку; — Эта память, в отличие от стека, автоматически не освобождается. Для сбора неиспользуемых объектов используется сборщик мусора; — В отличие от стека, куча не является потокобезопасной и ее необходимо контролировать, правильно синхронизируя код.

Экран	Слова
СБОРЩИК МУСОРА	Теперь, узнав основную часть, можно приступить к разбору, но, прежде чем углубиться в детали, давайте сначала упомянем несколько вещей: Этот процесс запускается автоматически Java, и Java решает, запускать или нет этот процесс; На самом деле это дорогостоящий процесс. При запуске сборщика мусора все потоки в вашем приложении приостанавливаются (в зависимости от типа GC, который будет обсуждаться позже); На самом деле это более сложный процесс, чем просто сбор мусора и освобождение памяти. Несмотря на то, что Java решает, когда запускать сборщик мусора, вы можете явно вызвать System.gc() и ожидать, что сборщик мусора будет запускаться при выполнении этой строки кода, верно? Это ошибочное предположение. Вы только как бы просите Java запустить сборщик мусора, но, опять же, Java решать, делать это или нет. В любом случае явно вызывать System.gc() не рекомендуется. Поскольку это довольно сложный процесс и может повлиять на вашу производительность, он реализован разумно. Для этого используется так называемый процесс «Mark and Sweep». Java анализирует переменные из стека и «отмечает» все объекты, которые необходимо поддерживать в рабочем состоянии. Затем все неиспользуемые объекты очищаются. Фактически, чем больше мусора и чем меньше объектов помечены как живые, тем быстрее идет процесс. Чтобы сделать это еще более оптимизированным, память кучи на состоит из нескольких частей, как было показано на картинке с организацией памяти в Java. Давайте пройдёмся по поколениям.