С недавнего времени у меня появилась настойчивая мысль, что профессиональное развитие сильно замедлилось и это хочется как-то исправить. Да, читаю книги, слушаю курсы, но в то же время приходит и понимание того, что возможно пришло время сменить работу, здесь вроде как все изучено, плавно уходим в рутину. Данная мысль сподвигла меня на рассылку своего резюме в несколько компаний — лидеров рынка. После прохождения собеседования в 3 из них, я решил, как водится внести свои 5 копеек в освещение обширной темы собеседования, а именно технических вопросов по Java коллекциям, с которыми приходится сталкиваться. Да, знаю, читатель скажет: «коллекции — избитая тема, сколько можно», но часть из приведенных ниже вопросов, я задавал своим знакомым разработчикам, которые занимают именно позиции разработчиков («крепких середнячков», по меркам недалекой от Москвы глубинки, которые уверенно справляются со своей работой на практике, а вот в теории скажем так есть пробелы, потому, что работа не требует решения каких-то нетривиальных задач, да и потому что не всем это интересно — изучать как внутри работает структура данных), вызывало растерянность. Думаю, что рассмотренный материал будет не очень интересен разработчикам выше уровня Junior (я попрошу их комментировать, дополнять и критиковать изложенный здесь материал), а вот Junior`ы уверен, найдут в этой статье интересное для себя.  
  
Признаюсь честно, сам при прохождении интервью не знал ответы на некоторые из изложенных ниже вопросов, хотя вроде как уже прошел этап джуниорства. Это вдвойне обидно, с учетом того, что позиции в те компании, где симпатию вызывало все, начиная от общения с HR и заканчивая возможной будущей сферой деятельности не удалось получить оффер и как раз там были вопросы по коллекциям, с которыми я не справился (уверен они внесли свою негативную лепту). А вот там, где все прошло вполне неплохо с точки зрения собеседования, предложенная сфера деятельности и общение в целом с будущими коллегами оставили негатив, так что закон «подлости» во всей красе. В итоге, данным топиком я хочу и в своей голове заполнить обнаруженные пробелы+систематизировать на «бумаге» эти знания.  
  
В статье я рассмотрю не только вопросы, вызвавшие у меня сложности на последних собеседованиях, но и вопросы, которые мне задавали за всю мою практику прохождения собеседований. Ну что ж, думаю пора переходить к вопросам:  
  
**1. Чем отличается ArrayList от LinkedList?**  
  
В моем рейтинге это один из двух самых популярных вопросов о коллекции, задают в 90% случаев. Вызвал у меня проблему на моем первом собеседовании на Junior Developer`а. Вкратце ответ на этот вопрос сводится к следующему: ArrayList это список, реализованный на основе массива, а LinkedList — это классический связный список, основанный на объектах с ссылками между ними.   
  
Преимущества ArrayList: в возможности доступа к произвольному элементу по индексу за постоянное время (так как это массив), минимум накладных расходов при хранении такого списка, вставка в конец списка в среднем производится так же за постоянное время. В среднем потому, что массив имеет определенный начальный размер n (в коде это параметр capacity), по умолчанию n = 10, при записи n+1 элемента, будет создан новый массив размером (n \* 3) / 2 + 1, в него будут помещены все элементы из старого массива + новый, добавляемый элемент. В итоге получаем, что при добавлении элемента при необходимости расширения массива, время добавления будет значительно больше, нежели при записи элемента в готовую пустую ячейку. Тем не менее, в среднем время вставки элемента в конец списка является постоянным. Удаление последнего элемента происходит за константное время. Недостатки ArrayList проявляются при вставке/удалении элемента в середине списка — это взывает перезапись всех элементов размещенных «правее» в списке на одну позицию влево, кроме того, при удалении элементов размер массива не уменьшается, до явного вызова метода trimToSize().  
  
LinkedList наоборот, за постоянное время может выполнять вставку/удаление элементов в списке (именно вставку и удаление, поиск позиции вставки и удаления сюда не входит). Доступ к произвольному элементу осуществляется за линейное время (но доступ к первому и последнему элементу списка всегда осуществляется за константное время — ссылки постоянно хранятся на первый и последний, так что добавление элемента в конец списка вовсе не значит, что придется перебирать весь список в поисках последнего элемента). В целом же, LinkedList в абсолютных величинах проигрывает ArrayList и по потребляемой памяти и по скорости выполнения операций. LinkedList предпочтительно применять, когда происходит активная работа (вставка/удаление) с серединой списка или в случаях, когда необходимо гарантированное время добавления элемента в список.  
  
Для углубленного и в то же время экспресс обучения очень рекомендую к прочтению замечательные статьи [tarzan82](https://habrahabr.ru/users/tarzan82/) о [ArrayList](http://habrahabr.ru/post/128269/) и [LinkedList](http://habrahabr.ru/post/127864/). Так же порекомендую статью от [lany](https://habrahabr.ru/users/lany/) о потреблении памяти коллекциями — [очень познавательно](http://habrahabr.ru/post/159557/).  
  
**2. Что вы обычно используете (ArrayList или LinkedList)? Почему?**  
  
Это вопрос является слегка замаскированной версией предыдущего, так как ответ на этот вопрос приведет к постепенному изложению ответа на предыдущей вопрос. В 90% случае ArrayList будет быстрее и экономичнее LinkedList, так что обычно используют ArrayList, но тем не менее всегда есть 10% случаев для LinkedList. Я говорю, что обычно ArrayList использую, ссылаясь на тесты и последний абзац из предыдущего вопроса, но не забываю и про LinkedList (в каких случаях? так же последний абзац предыдущего вопроса помогает).  
  
**3. Что быстрее работает ArrayList или LinkedList?**  
  
Еще одна замаскированная версия первого вопроса. Хитрее приведенных выше вариантов, что постановка вопроса подразумевает односложный ответ с выбором одного из предложенных вариантов, что, по задумке автора вопроса, как я понимаю, должно сразу выявить человека с неглубокими познаниями в collections. Правильным же действием будет встречный вопрос о том, какие действия будут выполняться над структурой. В итоге, диалог плавно переходит к ответу на первый вопрос.  
  
**4. Необходимо добавить 1млн. элемент, какую структуру вы используете?**  
  
Тоже довольно популярная скрытая версия первого вопроса. Так же постановка предполагает выбор одного из предложенных вариантов, хотя на самом деле информации для однозначного выбора нет. Нужно задавать дополнительные вопросы: в какую часть списка происходит добавление элементов? есть ли информация о том, что потом будет происходить с элементами списка? какие то ограничения по памяти или скорости выполнения? В целом, все тот же первый вопрос, но немного с другой стороны: вы через дополнительные вопросы, показываете глубину понимания работы Array и Linked List.  
Однажды я сам «клюнул» на этот крючок, домыслив про себя, что добавить — это «вставить» в конец списка и усиленно продвигал ArrayList, хотя ничего не знал (и не пытался узнать) про дальнейшие действие с этим списком и возможные ограничения.  
  
**5. Как происходит удаление элементов из ArrayList? Как меняется в этом случае размер ArrayList?**  
  
Опять же, ответ на вопрос 1 содержит ответ и на этот вопрос. При удалении произвольного элемента из списка, все элементы находящиеся «правее» смещаются на одну ячейку влево и реальный размер массива (его емкость, capacity) не изменяется никак. Механизм автоматического «расширения» массива существует, а вот автоматического «сжатия» нет, можно только явно выполнить «сжатие» командой trimToSize().  
  
**6. Предложите эффективный алгоритм удаления нескольких рядом стоящих элементов из середины списка, реализуемого ArrayList.**  
  
Неизбитый, по моим меркам вопрос, встречался мне всего однажды, когда я не знал механизма удаления элементов из ArrayList. В итоге вызвал у меня серьезные затруднения. На самом деле все довольно просто и очевидно, когда знаешь как происходит удаление одного элемента. Допустим нужно удалить n элементов с позиции m в списке. Вместо выполнения удаления одного элемента n раз (каждый раз смещая на 1 позицию элементы, стоящие «правее» в списке), нужно выполнить смещение всех элементов, стоящих «правее» n+m позиции на n элементов левее к началу списка. Таким образом, вместо выполнения n итераций перемещения элементов списка, все выполняется за 1 проход.  
  
**7. Как устроена HashMap?**  
  
Это второй из списка самых популярных вопросов по коллекциям. Уж даже не помню был ли случай, когда этот вопрос мне не задавали.  
  
Вкратце, HashMap состоит из «корзин» (bucket`ов). С технической точки зрения «корзины» — это элементы массива, которые хранят ссылки на списки элементов. При добавлении новой пары ключ-значение, вычисляет хеш-код ключа, на основании которого вычисляется номер корзины (номер ячейки массива), в которую попадет новый элемент. Если корзина пустая, то в нее сохраняется ссылка на вновь добавляемый элемент, если же там уже есть элемент, то происходит последовательный переход по ссылкам между элементами в цепочке, в поисках последнего элемента, от которого и ставится ссылка на вновь добавленный элемент. Если в списке был найден элемент с таким же ключом, то он заменяется. Добавление, поиск и удаление элементов выполняется за константное время. Вроде все здорово, с одной оговоркой, хеш-функций должна равномерно распределять элементы по корзинам, в этом случае временная сложность для этих 3 операций будет не ниже lg N, а в среднем случае как раз константное время.  
  
В целом, этого ответа вполне хватит на поставленный вопрос, дальше скорее всего завяжется диалог по HashMap, с углубленным пониманием процессов и тонкостей.  
  
Опять же, рекомендую к прочтению статью [tarzan82](https://habrahabr.ru/users/tarzan82/) по [HashMap](http://habrahabr.ru/post/128017/).  
  
**8. Какое начальное количество корзин в HashMap?**  
  
Довольно неожиданный вопрос, опять же меня он когда-то заставил угадывать число корзин при использовании конструктора по умолчанию.  
  
Ответ здесь — 16. Отвечая, стоит заметить, что можно используя конструкторы с параметрами: через параметр capacity задавать свое начальное количество корзин.  
  
**9. Какая оценка временной сложности выборки элемента из HashMap? Гарантирует ли HashMap указанную сложность выборки элемента?**  
  
Ответ на первую часть вопроса, можно найти в ответе на вопрос 7 — константное время необходимо для выборки элемента. Вот на второй части вопроса, я недавно растерялся. И устройство HashMap знал и про хеш-функцию тоже знал, а вот к такому вопросу не был готов, в уме кинулся вообще в другом направлении и сосредоточился на строении HashMap откинув проблему хеш-кода, который в голове всегда привык считать хеш-кодом с равномерным распределением. На самом деле ответ довольно простой и следует из ответа вопроса 7.   
  
Если вы возьмете хеш-функцию, которая постоянно будет возвращать одно и то же значение, то HashMap превратится в связный список, с отвратной производительностью. Затем даже, если вы будете использовать хеш-функцию с равномерным распределением, в предельном случае гарантироваться будет только временная сложность lg N. Так что, ответ на вторую часть вопроса — нет, не гарантируется.  
  
**10. Роль equals и hashCode в HashMap?**  
  
Ответ на этот вопрос следует из ответа на вопрос 7, хотя явно там и не прописан. hashCode позволяет определить корзину для поиска элемента, а equals используется для сравнения ключей элементов в списке внутри корзины и искомого ключа.  
  
**11. Максимальное число значений hashCode()?**  
  
Здесь все довольно просто, достаточно вспомнить сигнатуру метода: int hashCode(). То есть число значений равно диапазону типа int — 2^32 (точного диапазона никогда не спрашивали, хватало такого ответа).  
  
**12. Как и когда происходит увеличение количества корзин в HashMap?**  
  
Вот это довольно тонкий вопрос. Как показал мой мини-опрос, если суть устройства HashMap себе представляют многие более-менее ясно, то этот вопрос часто ставил собеседника в тупик.  
  
Помимо capacity в HashMap есть еще параметр loadFactor, на основании которого, вычисляется предельное количество занятых корзин (capacity\*loadFactor). По умолчанию loadFactor = 0,75. По достижению предельного значения, число корзин увеличивается в 2 раза. Для всех хранимых элементов вычисляется новое «местоположение» с учетом нового числа корзин.  
  
**13. В каком случае может быть потерян элемент в HashMap?**  
  
Этот интересный вопрос мне прислал [LeoCcoder](https://habrahabr.ru/users/leoccoder/), у меня подобного не спрашивали и честно признаюсь, после прочтения сходу не смог придумать сценарий для потери элемента. Все опять же оказалось довольно просто, хоть и не так явно: допустим в качестве ключа используется не примитив, а объект с несколькими полями. После добавления элемента в HashMap у объекта, который выступает в качестве ключа, изменяют одно поле, которое участвует в вычислении хеш-кода. В результате при попытке найти данный элемент по исходному ключу, будет происходить обращение к правильной корзине, а вот equals (ведь equals и hashCode должны работать с одним и тем же набором полей) уже не найдет указанный ключ в списке элементов. Тем не менее, даже если equals реализован таким образом, что изменение данного поля объекта не влияет на результат, то после увеличения размера корзин и пересчета хеш-кодов элементов, указанный элемент, с измененным значением поля, с большой долей вероятности попадет совсем в другую корзину и тогда он уже совсем потеряется.  
  
**14. Почему нельзя использовать byte[] в качестве ключа в HashMap?**  
  
Еще один вопрос от [LeoCcoder](https://habrahabr.ru/users/leoccoder/). Как обычно, все оказалось довольно просто — хеш-код массива не зависит от хранимых в нем элементов, а присваивается при создании массива (метод вычисления хеш-кода массива не переопределен и вычисляется по стандартному Object.hashCode() на основании адреса массива). Так же у массивов не переопределен equals и выполняет сравнение указателей. Это приводит к тому, что обратиться к сохраненному с ключом-массивом элементу не получится при использовании другого массива такого же размера и с такими же элементами, доступ можно осуществить лишь в одном случае — при использовании той же самой ссылки на массив, что использовалась для сохранения элемента. За ответ на этот вопрос отдельная благодарность уходит пользователю @dark\_dimius.  
  
**15. В чем отличия TreeSet и HashSet?**  
  
Начнем с того, что Set — это множество (так же называют «набором»). Set не допускает хранение двух одинаковых элементов. Формально говоря, термин «множество» и так обозначает совокупность различных элементов, очень важно, что именно различных элементов, так как это главное свойство Set. С учетом такого определения, пояснение про хранение одинаковых элементом не требуется, но в обиходе, понятие «множество» потеряло свой строгий смысл касательно уникальности элементов, входящих в него, поэтому все же уточняйте отдельно данное свойство множества.  
  
TreeSet обеспечивает упорядоченно хранение элементов в виде красно-черного дерева. Сложность выполнения основных операций в TreeSet lg N. HashSet использует для хранения элементов такой же подход, что и HashMap, за тем отличием, что в HashSet в качестве ключа выступает сам элемент, кроме того HashSet (как и HashMap) не поддерживает упорядоченное хранение элементов и обеспечивает временную сложность выполнения операций аналогично HashMap.  
  
**16. Устройство TreeSet?**  
  
Этот вопрос задают вместо вопроса 14 и здесь достаточно краткого ответа, что TreeSet основан на красно-черном дереве. Как правило этого хватает и собеседник сразу переходит к следующему вопросу, у меня ни разу не спрашивали механизм балансировки дерева или другие подробности его реализации.  
  
Для экспресс углубления знаний по красно-черному дереву рекомендую [вот эту статью](http://habrahabr.ru/post/66926/).  
  
**17. Что будет, если добавлять элементы в TreeSet по возрастанию?**  
  
Обычно данный вопрос собеседник предваряет фразой, что в основе TreeSet лежит бинарное дерево и если добавлять элементы по возрастанию, то как они будут распределены по дереву.  
  
Если нет точного представления об устройстве TreeSet, а есть общее понимание о том, что это бинарное дерево (в чем нас дополнительно уверяет собеседник), то данный вопрос может привести к интересному результату: все элементы после доабвления в обычное бинарное дерево будут находится в одной ветви длиной N элементов, что сводит на нет, все преимущества такой структуры, как дерево (фактически получается список). На самом, деле, как выше упоминалось в основе TreeSet лежит красно-черное дерево, которое умеет само себя балансировать. В итоге, TreeSet все равно в каком порядке вы добавляете в него элементы, преимущества этой структуры данных будут сохраняться.  
  
**Заключение**  
  
Надеюсь, рассмотренные вопросы будут полезны хабраюзерам. Прошу так же простить мне возможную некоторую наивность в том, что приведенные выше вопросы требуют такого детального рассмотрения, но в свое время подобная статья мне бы серьезно помогла. Уверен, что в статье присутствуют неточности — прошу в комментарии, кроме того, надеюсь, что более опытные товарищи в комментариях будут активно делится вопросами из своей практики и, если статья будет благосклонно принята хабрасообществом, то вполне возможно продолжение обзора технических вопросов для Java собеседований.  
  
P.S. Немного меркантильного интереса: поиски нового места работы продолжаются и если кто-то из хабраюзеров в процессе поиска Java разработчика в компанию с современным подходом к разработке и интересными задачами или просто может порекомендовать присмотреться к какой-либо подходящей вакансии — я буду благодарен, прошу в личку.