Вопросы

1. Дайте определение понятию “коллекция”.  
2. Назовите преимущества использования коллекций.  
3. Какие данные могут хранить коллекции?  
4. Какова иерархия коллекций?  
5. Что вы знаете о коллекциях типа List?  
6. Что вы знаете о коллекциях типа Set?  
7. Что вы знаете о коллекциях типа Queue?  
8. Что вы знаете о коллекциях типа Map, в чем их принципиальное отличие?  
9. Назовите основные реализации List, Set, Map.  
10. Какие реализации SortedSet вы знаете и в чем их особенность?  
11. В чем отличия/сходства List и Set?  
12. Что разного/общего у классов ArrayList и LinkedList, когда лучше использовать ArrayList, а когда LinkedList?  
13. В каких случаях разумно использовать массив, а не ArrayList?  
14. Чем отличается ArrayList от Vector?  
15. Что вы знаете о реализации классов HashSet и TreeSet?  
16. Чем отличаются HashMap и TreeMap? Как они устроены и работают? Что со временем доступа к объектам, какие зависимости?  
17. Что такое Hashtable, чем она отличается от HashMap? На сегодняшний день она deprecated, как все-таки использовать нужную функциональность?  
18. Что будет, если в Map положить два значения с одинаковым ключом?  
19. Как задается порядок следования объектов в коллекции, как отсортировать коллекцию?  
20. Дайте определение понятию “итератор”.  
21. Какую функциональность представляет класс Collections?  
22. Как получить не модифицируемую коллекцию?  
23. Какие коллекции синхронизированы?  
24. Как получить синхронизированную коллекцию из не синхронизированной?  
25. Как получить коллекцию только для чтения?  
26. Почему Map не наследуется от Collection?  
27. В чем разница между Iterator и Enumeration?  
28. Как реализован цикл foreach?  
29. Почему нет метода iterator.add() чтобы добавить элементы в коллекцию?  
30. Почему в классе iterator нет метода для получения следующего элемента без передвижения курсора?  
31. В чем разница между Iterator и ListIterator?  
32. Какие есть способы перебора всех элементов List?  
33. В чем разница между fail-safe и fail-fast свойствами?  
34. Что делать, чтобы не возникло исключение ConcurrentModificationException?  
35. Что такое стек и очередь, расскажите в чем их отличия?  
36. В чем разница между интерфейсами Comparable и Comparator?  
37. Почему коллекции не наследуют интерфейсы Cloneable и Serializable?

Ответы

Тема коллекций невероятно обширная и для того, чтобы ответить на каждый вопрос глубоко нужна отдельная статья почти под каждый вопрос. При проработке этого раздела рекомендую прочитать дополнительный материал, указанный в ответах.

**1. Дайте определение понятию “коллекция”.**

Коллекциями/контейнерами в Java принято называть классы, основная цель которых – хранить набор других элементов.

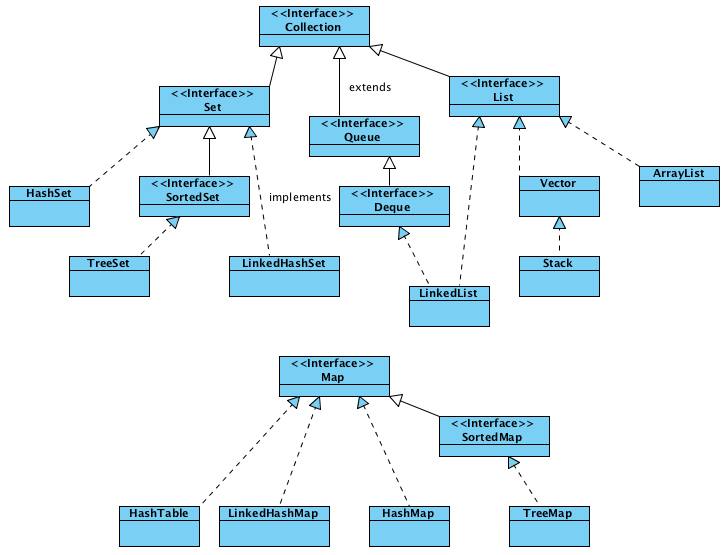
**2. Назовите преимущества использования коллекций.**

Массивы обладают значительными недостатками. Одним из них является конечный размер массива, как следствие, необходимость следить за размером массива. Другим — индексная адресация, что не всегда удобно, т.к. ограничивает возможности добавления и удаления объектов. Чтобы избавиться от этих недостатков уже несколько десятилетий программисты используют **рекурсивные типы данных**, такие как **списки** и **деревья**. Стандартный набор коллекций Java служит для избавления программиста от необходимости самостоятельно реализовывать эти типы данных и снабжает его дополнительными возможностями.

**3. Какие данные могут хранить коллекции?**

Коллекции могут хранить любые ссылочные типы данных.

**4. Какова иерархия коллекций?**

[](http://javastudy.ru/wp-content/uploads/2016/01/CollectionsHierarchy.png)

Здесь следует обратить внимание, что interface Map не входит в иерархию interface Collection.

С Java 1.6 классы TreeSet и TreeMap имплементируют интерфейсы NavigableSet и NavigableMap, которые расширяют интерфейсы SortedSet и SortedMap соответственно (SortedSet и SortedMap расширяют Set и Map).

Подробная статья про коллекции с описанием основных методов: http://www.quizful.net/post/Java-Collections

**5. Что вы знаете о коллекциях типа List?**

List — это упорядоченный список. Объекты хранятся в порядке их добавления в список. Доступ к элементам списка осуществляется по индексу.

**6. Что вы знаете о коллекциях типа Set?**

Set — множество **неповторяющихся** объектов. В коллекции этого типа разрешено наличие только одной ссылки типа null.

**7. Что вы знаете о коллекциях типа Queue?**

Queue — коллекция, предназначенная для хранения элементов в порядке, нужном для их обработки. В дополнение к базовым операциям интерфейса Collection, очередь предоставляет дополнительные операции вставки, получения и контроля.

Очереди обычно, но не обязательно, упорядочивают элементы в FIFO (first-in-first-out, «первым вошел — первым вышел») порядке.

Метод offer() вставляет элемент в очередь, если это не удалось — возвращает false. Этот метод отличается от метода add() интерфейса Collection тем, что метод add() может неудачно добавить элемент только с использованием unchecked исключения.

Методы remove() и poll() удаляют верхушку очереди и возвращают ее. Какой элемент будет удален (первый или последний) зависит от реализации очереди. Методы remove() и poll() отличаются лишь поведением, когда очередь пустая: метод remove() генерирует исключение, а метод poll() возвращает null.

Методы element() и peek() возвращают (но не удаляют) верхушку очереди.

**java.util.Queue<E>** реализует FIFO–буфер. Позволяет добавлять и получать объекты. При этом объекты могут быть получены в том порядке, в котором они были добавлены.

**Реализации**: java.util.ArrayDeque<E>, java.util.LinkedList<E>.

**java.util.Deque<E>** наследует java.util.Queue<E>. Двунаправленная очередь. Позволяет добавлять и удалять объекты с двух концов. Так же может быть использован в качестве стека.

**Реализации**: java.util.ArrayDeque<E>, java.util.LinkedList<E>.

Подробнее http://www.seostella.com/ru/article/2012/08/09/kollekcii-collections-v-java-queue.html

**8. Что вы знаете о коллекциях типа Map, в чем их принципиальное отличие?**

Интерфейс java.util.Map<K,V> используется для отображения каждого элемента из одного множества объектов (ключей) на другое (значений). При этом, каждому элементу из множества ключей ставится в соответствие ровно 1 элемент из множества значений. В то же время одному элементу из множества значений может соответствовать 1, 2 и более элементов из множества ключей. Интерфейс java.util.Map<K,V> описывает функциональность ассоциативных массивов.

**Реализации**: java.util.HashMap<K,V>, java.util.LinkedHashMap<K,V>, java.util.TreeMap<K,V>, java.util.WeakHashMap<K,V>.

java.util.SortedMap<K,V> наследует java.util.Map<K,V>. Реализации этого интерфейса обеспечивают хранение элементов множества ключей в порядке возрастания (см. java.util.SortedSet). **Реализации**: java.util.TreeMap<K,V>.

http://developer.alexanderklimov.ru/android/java/map.php

**9. Назовите основные реализации List, Set, Map.**

|  |  |  |
| --- | --- | --- |
| **Интерфейс** | **Класс/Реализация** | **Описание** |
| List | ArrayList | Список |
| LinkedList | Список |
| Vector | Вектор |
| Stack | Стек |
| Set | HashSet | Множество |
| TreeSet | Множество |
| SortedSet | Отсортированное множество |
| Map | HashMap | Карта/Словарь |
| TreeMap | Карта/Словарь |
| SortedMap | Отсортированный словарь |
| Hashtable | Хеш-таблица |

**10. Какие реализации SortedSet вы знаете и в чем их особенность?**

java.util.SortedSet<E> наследует java.util.Set<E>. Реализации этого интерфейса, помимо того что следят за уникальностью хранимых объектов, поддерживают их в порядке возрастания. Отношение порядка между объектами может быть определено, как с помощью метода compareTo интерфейса java.lang.Comparable<T>, так и при помощи специального класса-компаратора, наследующего интерфейс java.util.Comparator<T>.

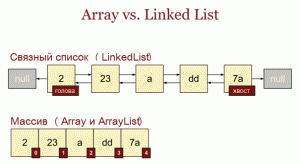
Реализации: java.util.TreeSet<E> — коллекция, которая хранит свои элементы в виде упорядоченного по значениям дерева. TreeSet инкапсулирует в себе TreeMap, который в свою очередь использует сбалансированное бинарное красно-черное дерево для хранения элементов. TreeSet хорош тем, что для операций add, remove и contains потребуется гарантированное время log(n).

**11. В чем отличия/сходства List и Set?**

Оба унаследованы от Collection, а значит имеют одинаковый набор и сигнатуры методов. List хранит объекты в порядке вставки, элемент можно получить по индексу. Set не может хранить одинаковых элементов.

**12. Что разного/общего у классов ArrayList и LinkedList, когда лучше использовать ArrayList, а когда LinkedList?**

**ArrayList** реализован внутри в виде обычного массива. Поэтому при вставке элемента в середину, приходится сначала сдвигать на один все элементы после него, а уже затем в освободившееся место вставлять новый элемент. Зато в нем быстро реализованы взятие и изменение элемента – операции get, set, так как в них мы просто обращаемся к соответствующему элементу массива.

[](http://javastudy.ru/wp-content/uploads/2016/01/ArrayListAndLinkedList.gif)

**LinkedList** реализован внутри по-другому. Он реализован в виде связного списка: набора отдельных элементов, каждый из которых хранит ссылку на следующий и предыдущий элементы. Чтобы вставить элемент в середину такого списка, достаточно поменять ссылки его будущих соседей. А вот чтобы получить элемент с номером 130, нужно пройтись последовательно по всем объектам от 0 до 130.Другими словами операции set и get тут реализованы очень медленно. Посмотри на таблицу:

|  |  |  |  |
| --- | --- | --- | --- |
| **Описание** | **Операция** | **ArrayList** | **LinkedList** |
| Взятие элемента | get | Быстро | Медленно |
| Присваивание элемента | set | Быстро | Медленно |
| Добавление элемента | add | Быстро | Быстро |
| Вставка элемента | add(i, value) | Медленно | Быстро |
| Удаление элемента | remove | Медленно | Быстро |

Если необходимо вставлять (или удалять) в середину коллекции много элементов, то лучше использовать LinkedList. Во всех остальных случаях – ArrayList.

LinkedList требует больше памяти для хранения такого же количества элементов, потому что кроме самого элемента хранятся еще указатели на следующий и предыдущий элементы списка, тогда как в ArrayList элементы просто идут по порядку

Из лекции javarush.ru

Структуры данных в картинках. LinkedList: http://habrahabr.ru/post/127864/

**13. В каких случаях разумно использовать массив, а не ArrayList?**

Если коротко, то Oracle пишет — используйте ArrayList вместо массивов. Если ответить на этот вопрос нужно по-другому, то можно сказать следующее: массивы могут быть быстрее и кушать меньше памяти. Списки теряют в производительности из-за возможности автоматического увеличения размера и сопутствующих проверок. Плюс к этому, что размер списка увеличивается не на 1, а на большее кол-во элементов (+15)\*. Так же доступ к [10] в массиве может быть быстрее чем вызов get(10) у списка.

\*Читатель прислал комментарий «У ArrayList увеличение происходит в 1.5 раза. int newCapacity = oldCapacity + (oldCapacity >> 1);».

Структуры данных в картинках. ArrayList: http://habrahabr.ru/post/128269/

Еще о ArrayList на сайте http://developer.alexanderklimov.ru/android/java/arraylist.php

**14. Чем отличается ArrayList от Vector?**

Vector deprecated. У Vector некоторые методы синхронизированы и поэтому они медленные. В любом случае Vector не рекомендуется использовать вообще.

**15. Что вы знаете о реализации классов HashSet и TreeSet?**

Название Hash… происходит от понятия хэш-функция. Хэш-функция — это функция, сужающая множество значений объекта до некоторого подмножества целых чисел. Класс Object имеет метод hashCode(), который используется классом HashSet для эффективного размещения объектов, заносимых в коллекцию. В классах объектов, заносимых в HashSet, этот метод должен быть переопределен (override).

HashSet реализован на основе хеш-таблицы, а TreeSet — на основе бинарного дерева.

Подробнее о Set, HashSet, LinkedHashSet, TreeSet: http://developer.alexanderklimov.ru/android/java/set.php

HashSet гораздо быстрее чем TreeSet (константное время против логарифмического для большинства операций, таких как add, remove, contains), но TreeSet гарантирует упорядоченность объектов. Оба не синхронизированы.

HashSet

* предоставляет константное время для add(), remove(), contains() и size()
* порядок элементов в контейнере может меняться
* производительность итерации по контейнеру зависит от емкости и «коэффициента загрузки» (рекомендуется оставлять load factor значением по умолчанию равным 0.75, что является хорошим компромиссом между временем доступа и объемом хранимых данных)

TreeSet

* время для базовых операций add(), remove(), contains() — log(n)
* гарантирует порядок элементов
* не предоставляет каких-либо параметров для настройки производительности
* предоставляет дополнительные методы для упорядоченного списка: first(), last(), headSet(), tailSet() и т.д.

Годный ответ на StackOverflow http://stackoverflow.com/questions/1463284/hashset-vs-treeset

**16. Чем отличаются HashMap и TreeMap? Как они устроены и работают? Что со временем доступа к объектам, какие зависимости?**

В целом ответ про HashSet и TreeSet подходит и к этому вопросу.

HashMap работает строго быстрее TreeMap.

TreeMap реализован на красно-черном дереве, время добавления/поиска/удаления элемента — O(log N), где N — число элементов в TreeMap на данный момент.

У HashMap время доступа к отдельному элементу — O(1) при условии, что хэш-функция (Object.hashCode()) определена нормально (что является правдой в случае Integer).

Общая рекомендация — если не нужна упорядоченность, использовать HashMap. Исключение — ситуация с вещественными числами, которые в качестве ключей почти всегда очень плохи. Для них нужно использовать TreeMap, предварительно поставив ему компаратор, который сравнивает вещественные числа так, как это нужно в данной задаче. Например, для обычных геометрических задач два вещественных числа могут считаться равными, если отличаются не более, чем на 1e-9.

Структуры данных в картинках. HashMap: http://habrahabr.ru/post/128017/

**17. Что такое Hashtable, чем она отличается от HashMap? На сегодняшний день она deprecated, как все-таки использовать нужную функциональность?**

Некоторые методы HashTable синхронизированы, поэтому она медленнее HashMap.

* HashTable синхронизирована, а HashMap нет.
* HashTable не позволяет иметь null ключи или значения. HashMap позволяет иметь один null ключ и сколько угодно null значений.
* У HashMap есть подкласс LinkedHashMap, который добавляет возможности по итерации. Если вам нужна эта функциональность, то можно легко переключаться между классами.

Общее замечание — не рекомендуется использовать HashTable даже в многопоточных приложениях. Для этого есть ConcurrentHashMap.

http://stackoverflow.com/questions/40471/differences-between-hashmap-and-hashtable

**18. Что будет, если в Map положить два значения с одинаковым ключом?**

Последнее значение перезапишет предыдущее.

**19. Как задается порядок следования объектов в коллекции, как отсортировать коллекцию?**

Класс ТгееМар полностью реализует интерфейс SortedMap. Он реализован как бинарное дерево поиска, значит его элементы хранятся в упорядоченном виде. Это  значительно ускоряет поиск нужного элемента. Порядок задается либо естественным следованием элементов, либо объектом, реализующим интерфейс сравнения Comparator.

В этом классе четыре конструктора:

ТгееМар() — создает пустой объект с естественным  порядком элементов;  
TreeМар(Comparator с) — создает пустой объект, в котором порядок задается объектом сравнения с;  
ТгееМар(Map f) — создает объект, содержащий все элементы отображения f, с естественным порядком его элементов;  
ТгееМар(SortedMap sf) — создает объект, содержащий все элементы отображения sf, в том же порядке.

Интерфейс Comparator описывает два метода сравнения:

int compare(Object obj1, object obj2) — возвращает отрицательное число, если obj1 в каком-то смысле меньше obj2; нуль, если они считаются равными; положительное число, если obj1 больше obj2. Для читателей, знакомых с теорией множеств, скажем, что этот метод сравнения обладает свойствами тождества, антисимметричности и транзитивности;

boolean equals(Object obj) — сравнивает данный объект с объектом obj, возвращая true, если объекты совпадают в каком-либо смысле, заданном этим методом.

Для каждой коллекции можно реализовать эти два метода, задав конкретный способ сравнения элементов, и определить объект класса SortedMap вторым конструктором. Элементы коллекции будут автоматически отсортированы в заданном порядке.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | public static void main(String[] args) {          Comparator comparator = new Comparator<String>() {              @Override              public int compare(String obj1, String obj2) {                  if (obj1 == null) {                      return -1;                  }                  if (obj2 == null) {                      return 1;                  }                  if (obj1.equals(obj2)) {                      return 0;                  }                  return obj1.compareTo(obj2);              }          };          TreeMap<String, String> treeMap1 = new TreeMap<>(comparator);            //or          TreeMap<Integer, String> treeMap = new TreeMap<>(new Comparator<Integer>() {              @Override              public int compare(Integer o1, Integer o2) {                  return Integer.compare(o1,o2);              }          });      } |

**20. Дайте определение понятию “итератор”.**

Итератор — объект, позволяющий перебирать элементы коллекции. Например foreach реализован с использованием итератора. Одним из ключевых методов интерфейса Collection является метод Iterator<E> iterator(). Он возвращает итератор — то есть объект, реализующий интерфейс Iterator.

Интерфейс Iterator имеет следующее определение:



|  |  |
| --- | --- |
| 1  2  3  4  5 | public interface Iterator <E>{      E next();      boolean hasNext();      void remove();  } |

**21. Какую функциональность представляет класс Collections**

Некоторые из методов

|  |  |
| --- | --- |
| Сигнатура метода | Описание |
| Collections.sort(List myList) | Сортирует список в естественном порядке. |
| Collections.sort(List, Comparator c) | Сортировка с использованием компаратора. |
| Collections.shuffle(List myList) | Перемешивает коллекцию в случайном порядке. |
| Collections.reverse(List myList) | Переворачивает коллекцию в обратном порядке. |
| Collections.binarySearch(List mlist, T key) | поиск в коллекции по ключу с использованием бинарного поиска. |
| Collections.copy(List dest, List src) | Копирует коллекцию источник src в dest. |
| Collections.frequency(Collection c, Object o) | Возвращает число вхождений объекта в коллекции. |
| Collections.synchronizedCollection(Collection c) | Возвращает синхронизированную (потокобезопасную) коллекцию. |

**22. Как получить не модифицируемую коллекцию?**

Коллекцию, доступную только для чтения можно получить с помощью методов:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | public static <T> SortedSet<T> unmodifiableSortedSet(SortedSet<T> s) {          return new UnmodifiableSortedSet<>(s);      }    public static <T> List<T> unmodifiableList(List<? extends T> list) {      return (list instanceof RandomAccess ?              new UnmodifiableRandomAccessList<>(list) :              new UnmodifiableList<>(list));  }  и т.д. для каждого типа (Map, SortedMap и т.п.) |

**23. Какие коллекции синхронизированы?**

Для этого используется пакет Concurrent. А так @Deprecated HashTable, Vector.

**24. Как получить синхронизированную коллекцию из не синхронизированной?**

Используйте следующие методы:

* Collections.synchronizedList(list);
* Collections.synchronizedSet(set);
* Collections.synchronizedMap(map);

Все они принимают коллекцию в качестве параметра, и возвращают потокобезопасную коллекцию с теми же элементами внутри.



|  |  |
| --- | --- |
| 1  2  3  4  5 | public static <T> Set<T> synchronizedSet(Set<T> s) {          return new SynchronizedSet<>(s);      }    и т.д. для каждого типа коллекции |

**25. Как получить коллекцию только для чтения?**

Используйте следующие методы:

* Collections.unmodifiableList(list);
* Collections.unmodifiableSet(set);
* Collections.unmodifiableMap(map);

Все они принимают коллекцию в качестве параметра, и возвращают коллекцию только для чтения с теми же элементами внутри.

**26. Почему Map не наследуется от Collection?**

Они не совместимы, т.к. созданы для различных структур данных. Map использует пару ключ-значение.

**27. В чем разница между Iterator и Enumeration?**

Enumeration в два раза быстрее Iterator и использует меньше памяти. Iterator потокобезопасен, т.к. не позволяет другим потокам модифицировать коллекцию при переборе. Enumeration можно использовать только для read-only коллекций. Так же у него отсутствует метод remove();

Enumeration: hasMoreElement(), nextElement()

Iterator: hasNext(), next(), remove()

**28. Как реализован цикл foreach?**

Реализован на основе Iterator.



|  |  |
| --- | --- |
| 1 | for(тип итер\_пер : коллекция) блок\_операторов |

**29. Почему нет метода iterator.add() чтобы добавить элементы в коллекцию?**

Единственная задача итератора это перебор коллекции. Каждая коллекция имеет метод add() которым вы можете воспользоваться. Нет смысла добавлять этот метод в итератор, потому что коллекции могут быть упорядоченными и неупорядоченными, и метод add() при этом должен быть устроен по разному.

**30. Почему в классе iterator нет метода для получения следующего элемента без передвижения курсора?**

Итератор похож на указатель своими основными операциями: он указывает на отдельный элемент коллекции объектов (предоставляет *доступ к элементу*) и содержит функции для перехода к другому элементу списка (следующему или предыдущему). Контейнер, который реализует поддержку итераторов, должен предоставлять первый элемент списка, а также возможность проверить, перебраны ли все элементы контейнера (является ли итератор конечным). Таким образом без курсора просто нельзя будет реализовать безошибочное передвижение по коллекции.

**31. В чем разница между Iterator и ListIterator?**

Есть три различия:

1. Iterator может использоваться для перебора элементов Set, List и Map. В отличие от него, ListIterator может быть использован только для перебора элементов коллекции List
2. Iterator позволяет перебирать элементы только в одном направлении, при помощи метода next(). Тогда как ListIterator позволяет перебирать список в обоих направлениях, при помощи методов next() и previous()
3. При помощи ListIterator вы можете модифицировать список, добавляя/удаляя элементы с помощью методов add() и remove(). Iterator не поддерживает данного функционала

**32. Какие есть способы перебора всех элементов List?**

Есть 4 способа:

* Цикл с итератором
* Цикл for
* Расширенный цикл for
* Цикл while

**33. В чем разница между fail-safe и fail-fast свойствами?**

В противоположность fail-fast, итераторы fail-safe не вызывают никаких исключений при изменении структуры, потому что они работают с клоном коллекции вместо оригинала.  
Итератор коллекции CopyOnWriteArrayList и итератор представления keySet коллекции ConcurrentHashMap являются примерами итераторов fail-safe.

**34. Что делать, чтобы не возникло исключение ConcurrentModificationException?**

Первым делом, можно подобрать другой итератор, работающий по принципу fail-safe. К примеру, если вы используете List, то можете взять ListIterator. Если же вам нужна устаревшая коллекция — то используйте перечислители.  
В том случае, когда вышеизложенное вам не подходит, у вас есть три варианта:  
При использовании JDK 1.5 или выше, вам подойдут классы ConcurrentHashMap и CopyOnWriteArrayList. Это самый лучший вариант  
Вы можете преобразовать список в массив и перебирать массив  
Вы можете блокировать изменения списка на время перебора с помощью блока synchronized  
Обратите внимание, что последние два варианта негативно скажутся на производительности.

**35. Что такое стек и очередь, расскажите в чем их отличия?**

Коллекции, созданные для того чтобы хранить элементы для дальнейшей обработки. Кроме базовых операций интерфейса Collection, очереди поддерживают дополнительные операции добавления, удаления и проверки состояния элемента.  
Обычно, но не обязательно очереди работают по принципу FIFO — первым пришел, первым ушел.  
Стэк — почти как очередь, но работает по принципу LIFO — последним пришел, первым ушел.  
Независимо от порядка добавления/удаления, голова очереди это элемент, который будет удален при вызове методов remove() или poll(). Также обратите внимание на то, что Stack и Vector оба потокобезопасны.

Использование: используйте очередь если вы хотите обрабатывать поток элементов в том же порядке в котором они поступают. Хорошо для списка заданий и обработки запросов.  
Используйте стэк если вы хотите класть и удалять элементы только с вершины стэка, что полезно в рекурсивных алгоритмах.

**36. В чем разница между интерфейсами Comparable и Comparator?**

В Java все коллекции, поддерживающие автоматическую сортировку, используют методы сравнения для того чтобы правильно рассортировать элементы. В качестве примера таких классов мы можем указать TreeSet, TreeMap и т.д.  
Для того чтобы рассортировать элементы, класс должен реализовать интерфейсы Comparator или Comparable. Именно поэтому классы-обертки как Integer, Double и String реализуют интерфейс Comparable.  
Интерфейс Comparable помогает сохранять естественную сортировку, тогда как Comparator позволяет сортировать элементы по разным особым шаблонам. Экземпляр компаратора обычно передается конструктору коллекции, если коллекция это поддерживает. Следует отметить, что интерфейс Comparable может быть реализован именно элементами коллекции или ключами Map, а Comparatorреализуется отдельным объектом (это удобно, так как можно заготовить несколько реализаций для разных правил сортировок, не меняя при этом код элементов коллекции/ключей Map).

**37. Почему коллекции не наследуют интерфейсы Cloneable и Serializable?**

Ну, простейший ответ — «потому что не надо». Функционал предоставляемый интерфейсами Cloneable и Serializable просто не нужен для коллекций.

Еще одна причина — далеко не всегда нужен подкласс Cloneable потому что каждая операция клонирования потребляет очень много памяти, и неопытные программисты могут расходовать ее сами не понимая последствий.

И последняя причина — клонирование и сериализация являются очень узкоспецифичными операциями, и реализовывать их нужно только когда это необходимо. Многие классы коллекции реализуют данные интерфейсы, но совершенно незачем закладывать их для всех коллекций вообще. Если вам нужно клонирование и сериализация — просто воспользуйтесь теми классами где она есть, если нет — остальными классами.

http://goo.gl/Eq2IOA — Часто задаваемые на собеседованиях вопросы по классам коллекциям в Java (Часть 2) javarush.ru

http://www.parshinpn.pro/content/voprosy-i-otvety-na-sobesedovanii-po-teme-java-collection-framework-chast-1 Вопросы и ответы на собеседовании по теме Java Collection Framework. Часть 1.

http://www.parshinpn.pro/content/voprosy-i-otvety-na-sobesedovanii-po-teme-java-collection-framework-chast-2 Вопросы и ответы на собеседовании по теме Java Collection Framework. Часть 2.

http://www.parshinpn.pro/content/voprosy-i-otvety-na-sobesedovanii-po-teme-java-collection-framework-chast-3 Вопросы и ответы на собеседовании по теме Java Collection Framework. Часть 3.