1. Коллекция - это набор однородных элементов.

2. Преимущества - у массива изначально указывается размер и ограничен. У коллекции -нет.

3. String, int, double, массивы и т.д.

5,6,7:

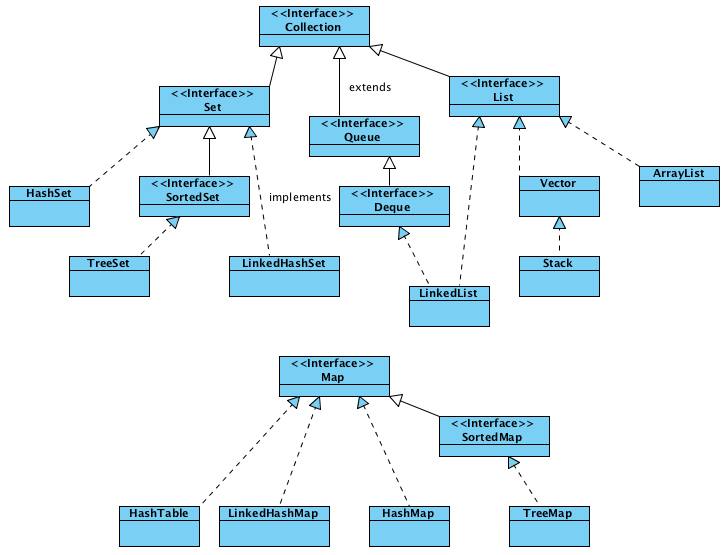
List) - упорядоченная коллекция, есть

Set) - нет дубликатов.

Queue) — очередь. добавлять только в конец, а удалять — только из начала

8. Map - используется для хранения по ключу ли может даже по хэшу.

9.



10. Sorted Set - уникальные элементы, сортированные в порядке возрастания

11. List хранит объекты в порядке вставки, элемент можно получить по индексу.

Set не может хранить одинаковых элементов.

Оба наследованы от Collection

12. ArrayList - внутри реализован по типу массива. Если вставить в середину него элемент, то все элементы после него будут сдвинуты на один шаг вправо.

LinkedList - реализован так, что каждый его элемент ссылается на предыдущий и следующий.

Чтобы вставить элемент в середину LinkedList, достаточно поменя ссылки его будущих соседей. Но чтобы получить элемент по индексу, то нужно перебирать

Список от 0 до Индекса - это так реализовано.

13. Масссив можно использовать вместо ArrayList, если заранее известно сколько элементов будет в массиве и не надо будет его модифицировать.

В этом случае они будут более производительными.

14. Vector считается устаревшим. У вектора методы медленные. Его не рекомендуется использовать.

15. ХэшСет быстрее чем ТриСет, т.к. ХэшСет реализован на хэш-таблицах,

а ТриСет - на основе бинарного дерева.

Трисет гарантирует порядоченность объектов.

16. ХэшМап быстрее чем ТриМап

17. ХэшМап быстрее чем ХэшТэйбл,поскольку у последней некоторые функции синхронизированы.

18. Последнее значение перезапишет предыдущее.

19. ТриМап реализуется интерфейс СортедМап.Он реализован как бинарное дерево,

его элементы хранятся в упорядоченом виде.

Элементы лежат в коллекци по тому порядку, по которому их добавляли.

Сортировать можно с помощью интерфейса Comparator.

20. Итератор - объект, позвоялет перебират коллекции.

21. Основной причиной того, что интерфейс **Map** не наследуется от интерфейса **Collection**, является различие в их основных функциях и спецификах работы.

22. 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.

23. 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.

24. 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 данную функциональность.

25. Есть 4 способа:

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

26. Стек (Stack) - структура данных, где добавление и удаление элементов происходит только с одного конца (вершины). Принцип работы - LIFO (Last-In-First-Out), последний добавленный элемент будет первым удаленным. Пример: стек вызовов функций.

Очередь (Queue) - структура данных, где добавление элементов происходит в одном конце (хвосте), а удаление - в другом (голове). Принцип работы - FIFO (First-In-First-Out), первый добавленный элемент будет первым удаленным. Пример: очередь задач для обработки.

27. Интерфейсы Comparable и Comparator в Java предоставляют различные подходы для сравнения объектов, но оба позволяют установить порядок сортировки. Comparable:

Comparable - это интерфейс, который позволяет классу определить собственный способ сравнения объектов.

Класс, реализующий Comparable, определяет метод compareTo(), который используется для сравнения текущего объекта с другим объектом.

Метод compareTo() возвращает отрицательное значение, ноль или положительное значение, в зависимости от того, является ли текущий объект меньшим, равным или большим, чем другой объект.

Comparable используется для естественной сортировки объектов. Например, строки сортируются в лексикографическом порядке, числа - в порядке возрастания.

Comparator:

Comparator - это интерфейс, который позволяет определить внешний способ сравнения объектов. Класс, реализующий Comparator, определяет метод compare(), который принимает два объекта для сравнения.

Метод compare() возвращает отрицательное значение, ноль или положительное значение, указывая, является ли первый объект меньшим, равным или большим, чем второй объект. Comparator позволяет определить несколько различных способов сортировки одного класса объектов. Он позволяет создавать компараторы для сортировки объектов по различным критериям.

Разница между Comparable и Comparator:

Comparable используется внутри класса для определения способа сравнения и сортировки объектов.

Класс, реализующий Comparable, имеет естественный порядок сортировки.

Comparator предоставляет внешний компаратор, который можно использовать для сравнения и сортировки объектов. Класс, реализующий Comparator, не требует изменений в самом классе объектов.

Оба интерфейса являются полезными для сравнения и сортировки объектов в Java и позволяют управлять порядком элементов в коллекциях и других структурах данных. Выбор между ними зависит от контекста и требований конкретной задачи.