

13. Коллекции

Информатика, ИТИС, 1 курс 2 семестр

М.М.Абрамский

2017

Что мы умеем после прошлой пары?

- Мы умеем описывать, как себя должен вести набор данных
 - АТД, интерфейс
- Мы знаем, как могут быть устроены коллекции
 - Прямая/ссылочная адресация
- Мы знаем, как делать наборы данных произвольного типа
 - Generics

**НУ ЧТО, ПОШЛИ
РЕАЛИЗОВЫВАТЬ**

Стоп!

- Много людей в мире программирования приходило, приходит и будет приходить к одним и тем же идеям.
- Что влечет за собой договоренность между людьми не придумывать велосипедов и использовать готовые решения.
- Одно из них — коллекции в Java.

Дисклеймер #1

На структурах данных / программировании:

- Студент: «Раз не нужно изобретать велосипед, зачем вы заставляете нас делать все это руками, почему бы не взять готовое?»
- *Что бы ты, дружок, знал как все устроено, понимал причинно-следственные связи и вообще был образованным человеком со светлой головой и прямыми руками!*

Дисклеймер #2

Но не дай Бог ты пойдешь работать и будешь там упражняться в умении делать двухколесные средства передвижения, использующие педальный механизм.

На работе ты с умом используешь готовые инструменты, наилучшим образом подходящие для задачи.

Коллекции

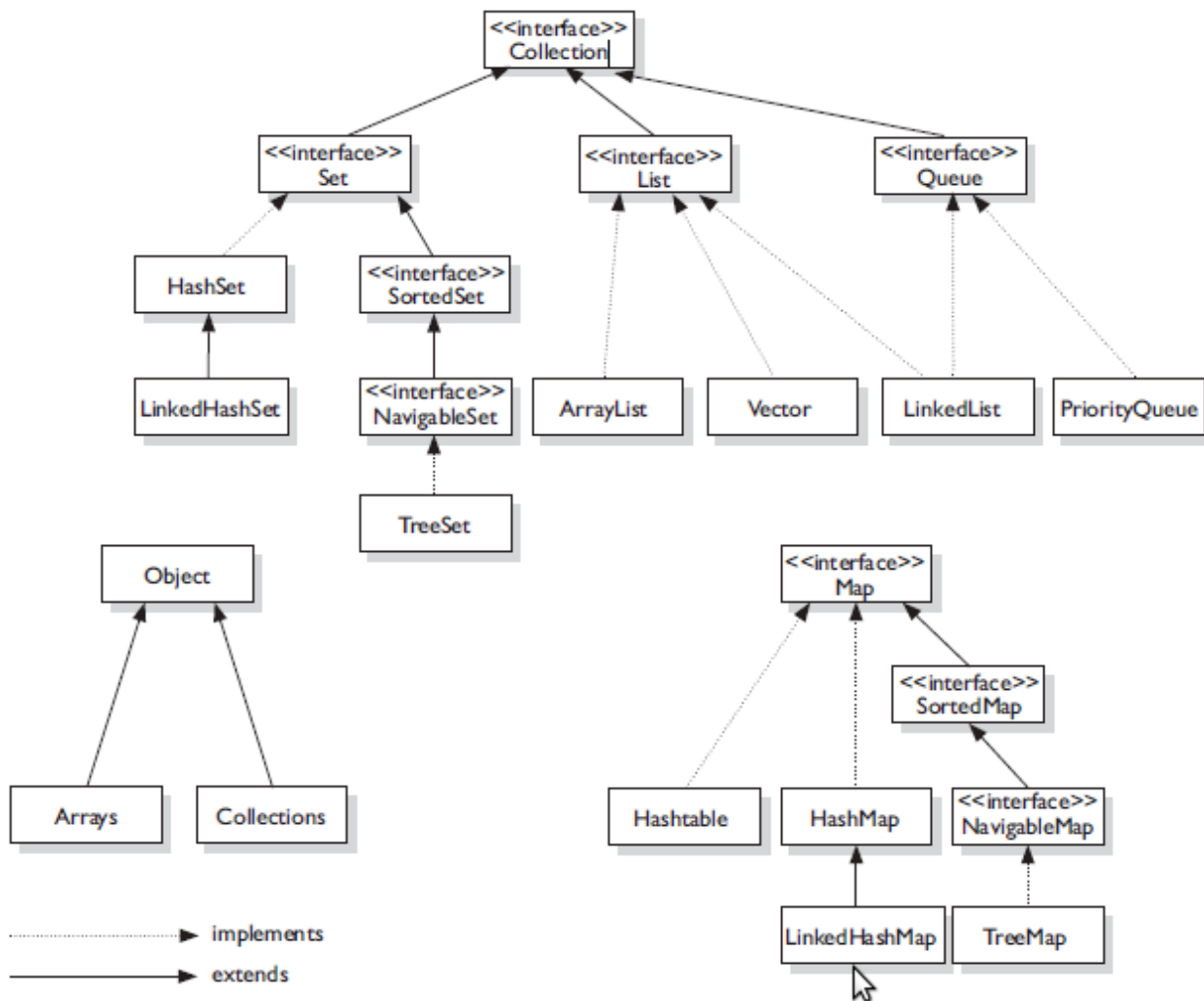
- С 1го дня в Java
- С каждой версией добавлялись новые/модифицировались старые
 - Java 1.2 – Добавились Map
 - Java 1.5 – Добавились generics

По факту

Java Collections – набор интерфейсов и классов-коллекций, а также ряда других полезных служебных классов.

Что надо знать?

- Интерфейс (хотя можно посмотреть в JavaDocs)
- Реализацию (чтобы знать, как работает)



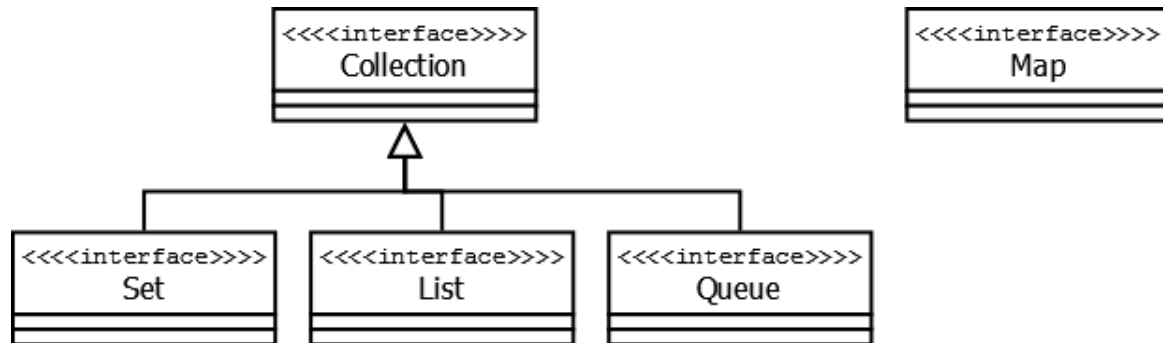
*Отрывок иерархии
(чтобы примерно
осознавался
масштаб)*

Обратная совместимость

- Коллекции – хороший пример следования обратной совместимости.
 - Stack – класс, Queue – интерфейс
 - Stack наследует от Vector, который устарел миллион лет назад.
 - Правда, и Stack потом устарел
 - Стирание типов в Generics

Интерфейсы

- Collection
 - Set (extends Collection)
 - List (extends Collection)
 - Queue (extends Collection)
 - Map (**не** extends Collection)
 - И их потомки



Дисклеймер #3

- Понятно, что интерфейсы-коллекции – это АТД
- **Но не надо, не нужно, не стоит** на экзамене по АиСД / незнакомому человеку на улице говорить «АТД Сортед Сэт» или «АТД Лист», вас могут просто не понять.

Дисклеймер #4

Java 8 – не сегодня!

Следующая лекция вся целиком про Java 8 и то, что она сделала с Java, в том числе и с коллекциями!

Collection

- Самая простой набор данных (*коллекция*)
 - Ни индексов, ни упорядоченности – ничего нет.
- Ключевые методы:
 - `int size()`
 - `boolean add(T t)`
 - `boolean isEmpty()`
 - `boolean contains(T t)`
 - `void remove(T t)`

Set (extends Collection)

- Интерфейс = интерфейс Collection
- Но требуется, чтобы те, кто реализуют Set, добивались уникальности элементов!

Интерфейсы-наследники Set

- SortedSet
 - Подразумевает возможность сравнения двух элементов из Set – comparator()
 - Возвращает наибольший и наименьший элемент - first(), last()
 - Может возвращать, также подмножества элементов, содержащиеся между значениями 2мя значениями
 - subSet(a, b), headSet(a) – меньше a, tailSet(a) – больше a
 - » Как сравнивать объекты, мы еще изучим.
- NavigableSet (extends SortedSet)
 - еще больше методов про упорядоченность – например, возможность вернуть для элемента предыдущий и следующий по значению

List (extends collection)

- Ну это наш список!
 - все, что связано с индексами: `set(i, x)`, `get(i)`, `add(i, x)`, `indexOf(x)`, `lastIndexOf(x)`, `subList(from, to)`, ...

Queue - очередь

- Сколько методов нужно очереди?

Queue - очередь

- А сколько у нее сейчас?

Queue - очередь

- Да, вот так вот, она берет и наследует от Collection – и берет все у нее.
 - Ну, так получилось.
- Методы: offer – добавить элемент – и 4 версии доставания элемента: pull, peek, element, remove

Queue

- «Смысл очереди в информационных системах»

Deque (extends Queue)

- Очередь с двумя концами
 - можно добавлять с обоих концов и удалять с обоих концов.
- Может вести себя и как очередь, и как стек.

Map<K, V>

- Ассоциативный массив, отображение, словарь...
 - Короче, набор пар «Ключ-значение»
 - Телефонный справочник, словарь, результаты экзамена и др.
- КЛЮЧИ УНИКАЛЬНЫ!

Map<K, V>

- Не содержит `add`, но содержит
 - **V put(K, V)**
- Другие методы:
 - **V get(K)**
 - **boolean containsKey(K)**
 - **boolean containsValue(V)**
 - **keySet()** – множество ключей (которые уникальны – значит Set)
 - **entrySet()** – множество пар – объектов с интерфейсом пары элементов `Entry<K,V>`
 - Какой тип у `entrySet()`?
 - `Set<Entry<K, V>>`

по аналогии с Set

- SortedMap
- NavigableMap

Abstract Collections

- Чтобы не все нужно было реализовывать при написании собственных коллекций
- Расписаны методы, которые используют другие методы, суть которых зависит от конкретной реализации класса.
 - contains – понятно как
 - а вот add – непонятно как.
 - паттерн проектирования?
- AbstractCollection, AbstractList, AbstractQueue

Пример метода

contains – не зависит от реализации конкретной коллекции.

```
abstract class MyAbstractCollection<T> implements Collection<T> {  
    abstract boolean add(T e);  
    abstract Iterator<T> iterator();  
    boolean contains(T e) {  
        Iterator<T> i = iterator();  
        while (i.hasNext()) {  
            if (i.next().equals(e))  
                return true;  
        }  
        return false;  
    }  
}
```

Теперь классы

- ArrayList
- LinkedList
- HashSet (HashMap)
- TreeSet (TreeMap)
- ...

ArrayList

- В класс спрятался наш родной массив.
 - Size
 - Capacity
 - loadFactor

LinkedList

- Реализует List
- Как ни странно, реализует Queue и Deque
 - Хороший пример объяснения начинающему программисту, зачем нужно восходящее преобразование:
 - Что лучше?
 - `LinkedList<String> queue = new LinkedList<>();`
 - или
 - `Queue<String> queue = new LinkedList<>();`

HashMap

- Реализация Map
- С хэш-таблицей внутри

Что такое хэш

- Это такая функция:
 - Лавинный эффект
 - Коллизия
 - Сложное вычисление прообраза

Хэширование

- Процесс вычисления хэша
- Есть известные алгоритмы хэширования:
 - md5, sha1, sha256 и др.

Минутка любознательности

- Хэширование – то, благодаря чему админы ваших соцсеток не знают ваши пароли.
 - А вы думали, знают? А вот и нет!
 - Если владелец сайт знает ваши пароли – он [выбрать слово].
 - » Хотя не совсем.

Почему хранить пароли в базе данных плохо?

- Злоумышленники, украв базу, узнают пароли
- Админы могут получать доступ туда, куда не надо.

Что происходит на самом деле?

- Когда вы регистрируетесь, пароль, который вы ввели – хэшируется
 - Получается строка вида d6aabbdd62a11ef721d15
- Когда вы входите на сайт, введенный вами пароль хэшируется
 - Если хэши равны – значит и пароль совпадает с тем, что лежит в базе
 - Вероятность совпадения хэшей мала, вероятность совпадения хэшей у похожих слов – нулевая.

Пример хэша (sha256)

password

5e884898da28047151doe56f8dc6292773603dod6aabbdd62a11ef721d1542d8

Password

e7cf3ef4f17c3999a94f2c6f612e8a888e5b1026878e4e19398b23bd38ec221a

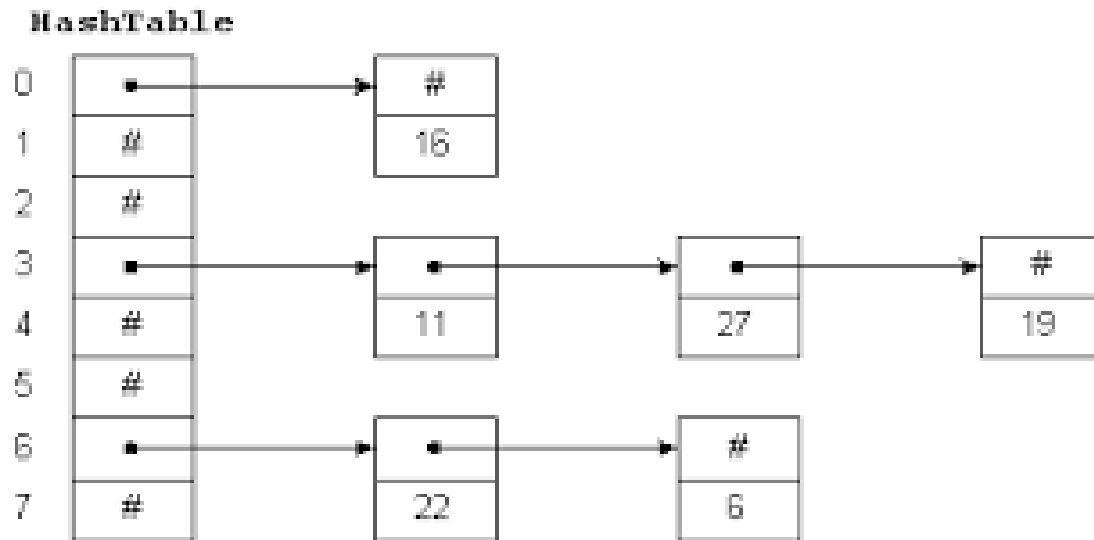
Password1

19513fdc9da4fb72a4a05eb66917548d3c90ff94d5419e1f2363eea89dfec1dd

Вернемся к Java

- Есть такая структура данных – хэш-таблица.
 - Это как массив, только в качестве индекса используется хэш
- Работа:
 - При добавлении вычисляется хэш
 - Прямым доступом проверяют, есть ли уже элементы с этим хэшем (используем простой алгоритм, не крутой)
 - Если нет – вставляем по этому хэшу пустой список, в который добавляют элемент
 - Если есть – вставляем элемент в конец списка.
 - » но у нас – уникальность! поэтому нужно еще проверить каждый элемент

Устройство хэш-таблицы



hashCode

- В Java у Object есть метод hashCode, возвращающий хэш.
 - На него HashMap и опирается – но не сразу
 - Он вычисляет hash(hashcode())

Проблема

- У одинаковых объектов хэшкоды всегда разные.
- Можно переопределить хэш – но это ужасно
- Можно воспользоваться другим Set

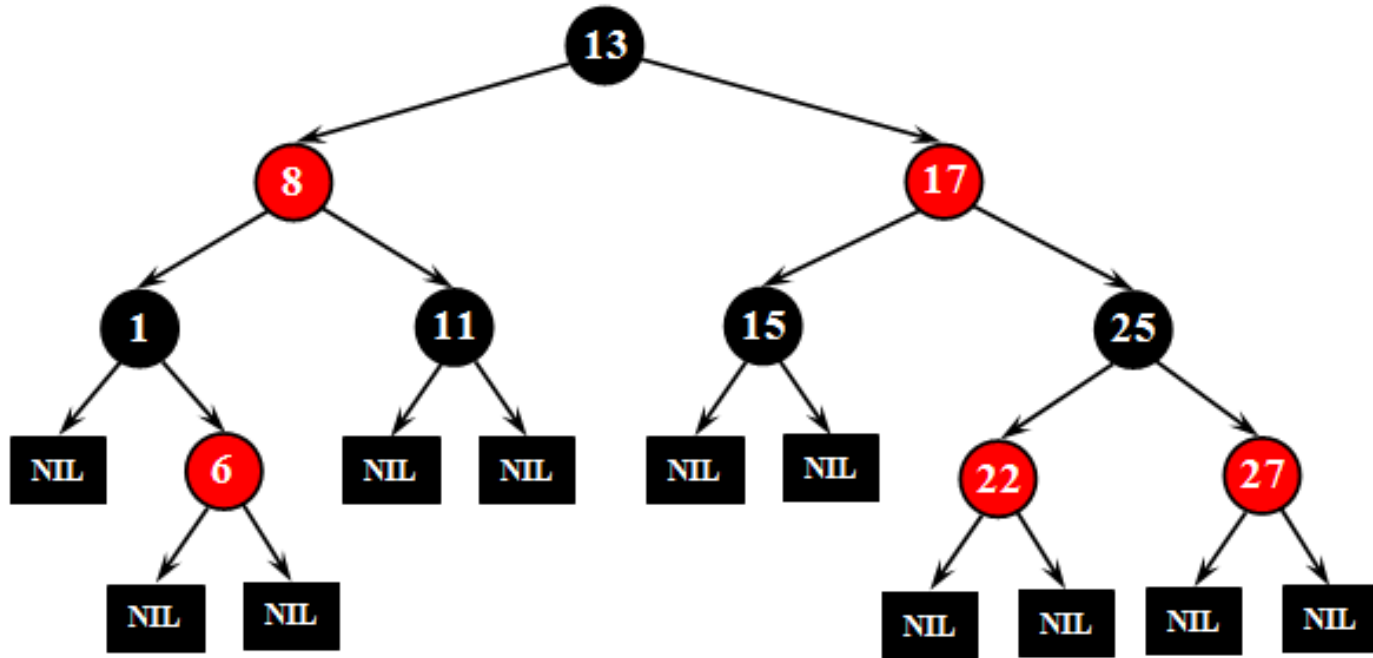
Контракт между equals и hashCode

- Если объекты одинаковы, то и их хэшкоды одинаковы.
- При вызове put у HashMap (add у HashSet) сначала проверяется hashCode ключа (элемента)
 - Если такого hashCode нет – ключ (элемент) добавляется
 - Если есть – включается equals
 - Если equals дает false – добавляется
 - Если true – не происходит замена значения (HashMap) или не добавляется (в случае HashSet)

TreeMap

- В реализации использовано красно-черное дерево.
- Одно из деревьев поиска, изготовленного, чтобы было удобно балансировать.
- При добавлении в TreeSet проверяется просто equals.

Красно-черное дерево



! Подробности в курсе структур данных

Set

- HashSet и TreeSet – это HashMap и TreeMap, где value – объект заглушка

Устаревшие классы

- Vector (implements List)
- Hashtable (implements Map)
- синхронизированы — и это плохо
— и это тема следующих лекций

Учитесь различать интерфейсы и классы

- Сначала смотрим на последнее слово в названии – это суть.
- Затем смотрим на остальные слова (если они есть)
 - Если они про интерфейс – это интерфейс
 - Если они про реализацию – это класс.
 - Если слов других нет – это скорее всего интерфейс (ну кроме Stack)

Потренируемся

- ArrayList
- SortedSet
- TreeSet
- HashMap
- LinkedList
- *Queue*
- *Stack*

Stack

- void push(T x)
- T pop()
- T peek()
- boolean isEmpty()

В java – устарел.

ArrayDeque

- Реализация массивом интерфейса Deque
 - значит и Stack

Используем коллекции

- Как пройти по всей?

Итератор

- Коллекция реализует `Iterable<T>`
- Тогда у нее есть метод `iterator()`, возвращающий объект `Iterator<T>`
 - Который вы хорошо помните по лекции про паттерны

Использование итератора

```
ArrayList<Student> students = new ArrayList<>();  
Iterator<Student> i = students.iterator();  
while (i.hasNext()) {  
    System.out.println(i.next());  
}
```

ListIterator (extends Iterator)

- Специальный итератор для списков
 - Есть дополнительные возможности.
 - ! самостоятельно

Collections

- Аналогичный класс Arrays мы упоминали
- Просто замечательный класс с полезным функционалом для коллекций.

Collections – рассмотрим один метод

- ...
- Sort – быстрая сортировка
 - Попробуем отсортировать.

Student

```
public class Student {  
    private String group;  
    private String lastName;  
    private int averageScore;  
    // ...  
}
```

Сортируем Student

```
public static void main(String[] args) {  
    ArrayList<Student> students = new ArrayList<>();  
    students.add(new Student("11-602", "Gabdreeva", 96));  
    students.add(new Student("11-601", "Mingachev", 94));  
    students.add(new Student("11-604", "Romanov", 94));  
    students.add(new Student("11-605", "Bagautdinov", 94));  
    students.add(new Student("11-603", "Nurgatina", 96));  
    Collections.sort(students);  
    System.out.println(students);  
}
```

Нужно сравнивать

- Умеем ли мы сравнивать объекты класса студент?
 - Да! Нас не интересует абстрактное сравнение. Нужны конкретные критерии: старше/младше, выше в списке по алфавиту/ниже, средний балл выше/ниже.

Comparable<T>

Интерфейс из `java.util`, реализовав который, объекты можно сравнивать друг с другом.

Суть – метод `int compareTo(другой объект)`

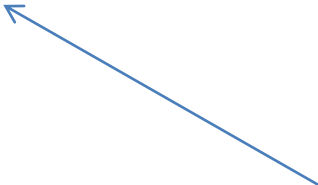
- Возвращается положительное число, если он больше (позже) переданного объекта
- Возвращает отрицательное число, если наоборот
- Возвращается 0, если по критерию равны.

Некоторые классы Java уже реализуют Comparable – например, `String`

Какой самый простой способ упорядочить студентов?

Student implements Comparable<Student>

```
public class Student implements Comparable<Student> {  
  
    @Override  
    public int compareTo(Student s) {  
        return lastName.compareTo(s.getLastName());  
    }  
  
    // ...  
}
```



Используем String
Возвращает расстояние
между строками

Теперь сортировка работает как мы ХОТИМ

```
students.add(new Student("11-602", "Gabdreeva", 96));  
students.add(new Student("11-601", "Mingachev", 94));  
students.add(new Student("11-604", "Romanov", 94));  
students.add(new Student("11-605", "Bagautdinov", 94));  
students.add(new Student("11-603", "Nurgatina", 96));  
Collections.sort(students);  
System.out.println(students);
```

```
[Student@74a14482, Student@1540e19d, Student@677327b6, Student@14ae5a5,  
Student@7f31245a]
```

Ой!

Теперь сортировка работает как мы ХОТИМ

```
students.add(new Student("11-602", "Gabdreeva", 96));  
students.add(new Student("11-601", "Mingachev", 94));  
students.add(new Student("11-604", "Romanov", 94));  
students.add(new Student("11-605", "Bagautdinov", 94));  
students.add(new Student("11-603", "Nurgatina", 96));  
Collections.sort(students);  
System.out.println(students);
```

[**Bagautdinov**{group='11-605', score=94}, **Gabdreeva**{group='11-602', score=96}, **Mingachev**{group='11-601', score=94}, **Nurgatina**{group='11-603', score=96}, **Romanov**{group='11-604', score=94}]

Но вспомните любую сортировку

- Там куча критериев
 - Хотя бы друзей вк – по дате добавления, по популярности, по имени
- Нужен инструмент динамической настройки поиска

Comparator<T>

- Сторонний объект, умеющий сравнивать 2 объекта типа T
- *Он как весы!*



Comparator для Student по среднему баллу

```
public class AverageScoreComparator implements Comparator<Student> {  
    @Override  
    public int compare(Student s1, Student s2) {  
        return (s1.getAverageScore() - s2.getAverageScore());  
    }  
}
```

Использование

```
Collections.sort(students, new AverageScoreComparator());  
System.out.println(students);
```

```
[Mingachev{group='11-601', score=94},  
Romanov{group='11-604', score=94},  
Bagautdinov{group='11-605', score=94},  
Gabdreeva{group='11-602', score=96},  
Nurgatina{group='11-603', score=96}]
```

С анонимным классом

```
ArrayList<Student> students = new ArrayList<>();
students.add(new Student("11-602", "Gabdreeva", 96));
students.add(new Student("11-601", "Mingachev", 94));
students.add(new Student("11-604", "Romanov", 94));
students.add(new Student("11-605", "Bagautdinov", 94));
students.add(new Student("11-603", "Nurgatina", 96));
Collections.sort(students, new Comparator<Student>() {
    @Override
    public int compare(Student o1, Student o2) {
        return o1.getGroup().compareTo(o2.getGroup());
    }
});
System.out.println(students);
```

что выведется?

Уже по группе

```
[Mingachev{group='11-601', score=94},  
Gabdreeva{group='11-602', score=96},  
Nurgatina{group='11-603', score=96},  
Romanov{group='11-604', score=94},  
Bagautdinov{group='11-605', score=94}]
```

Не путайте Comparator и Comparable

- Comparable – свойство самого класса,
- Comparator – сторонний объект

Вот теперь понятно

Наличие этих объектов позволяет реализовывать понятие «следующий» и «предыдущий» элемент по значению для определенных интерфейсов и классов

- SortedSet, PriorityQueue

PriorityQueue

- *Обычная очередь*: первым пришел, первым уйдешь.
- *Очередь с приоритетами*: кто круче, тот первым и уйдет.

Очень известная модель, применяемая в разных областях

- «Например, перейти к процессу, занимающему наибольшее количество ресурсов»

Пример PriorityQueue

```
PriorityQueue<Student> priorityQueue = new PriorityQueue<>(  
    new Comparator<Student>() {  
        public int compare(Student o1, Student o2) {  
            return o1.getGroup().compareTo(o2.getGroup());  
        }  
    }  
);  
priorityQueue.add(new Student("11-602", "Gabdreeva", 96));  
priorityQueue.add(new Student("11-604", "Romanov", 94));  
priorityQueue.add(new Student("11-601", "Mingachev", 94));  
priorityQueue.add(new Student("11-605", "Bagautdinov", 94));  
priorityQueue.add(new Student("11-603", "Nurgatina", 96));  
System.out.println(priorityQueue.poll());
```

КТО ВЫВЕДЕТСЯ?

Ключевые слова

Arrays, Collection, Collections, HashMap, HashSet, List, Map, PriorityQueue, Queue, Set, Коллекции

Почитать

- <https://habrahabr.ru/post/237043/>
- <http://www.quizful.net/post/Java-Collections>
- <https://jsehelper.blogspot.ru/2016/01/java-collections-framework-1.html>
- <http://www.javatpoint.com/collections-in-java>
- https://www.tutorialspoint.com/java/java_collections.htm
- <http://echuprina.blogspot.ru/2012/02/comparable-comparator.html>
- <http://www.quizful.net/interview/java/difference-comparator-from-comparable>