

Java Advanced

Collections Framework

Содержание

1. Коллекции
2. Множества
3. Списки
4. Очереди
5. Отображения
6. Упорядоченные коллекции
7. Алгоритмы
8. Устаревшие коллекции
9. Заключение

Collections Framework

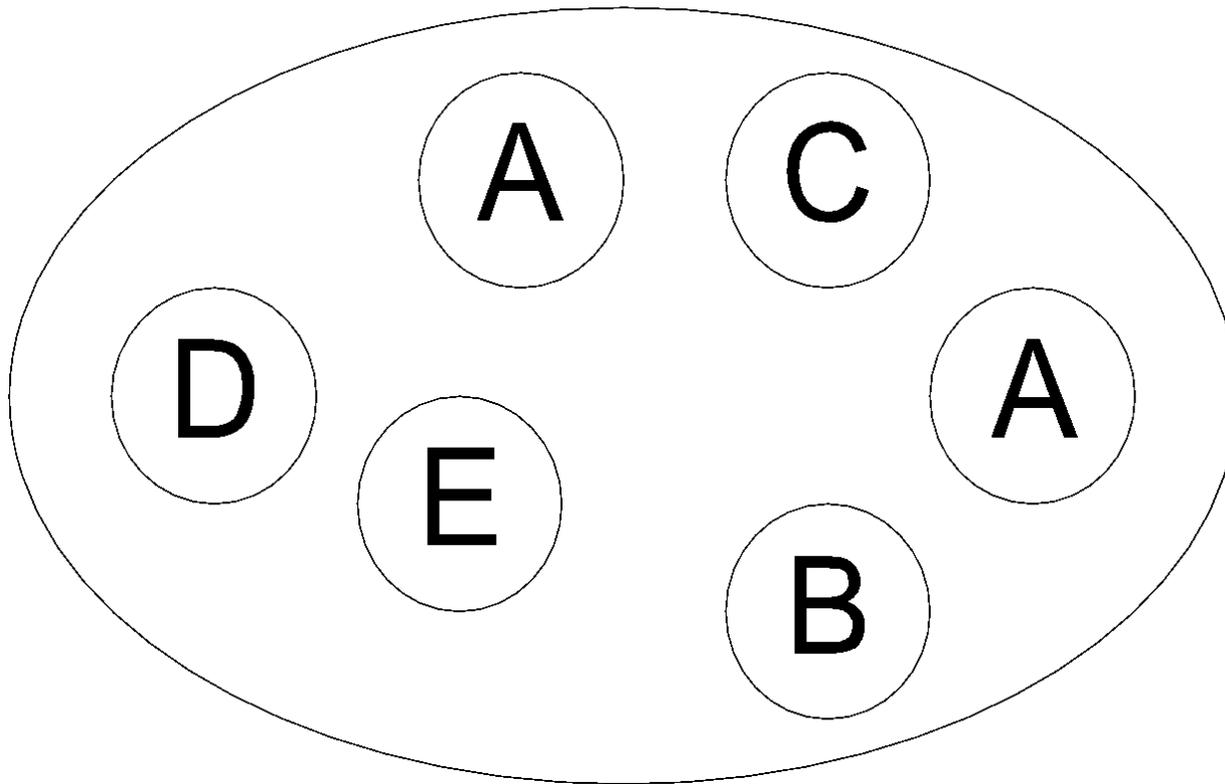
- Набор стандартных контейнеров (коллекций) и правил их использования
 - Интерфейсы
 - Релизации
 - Алгоритмы
- Пакет `java.util`

Часть 1

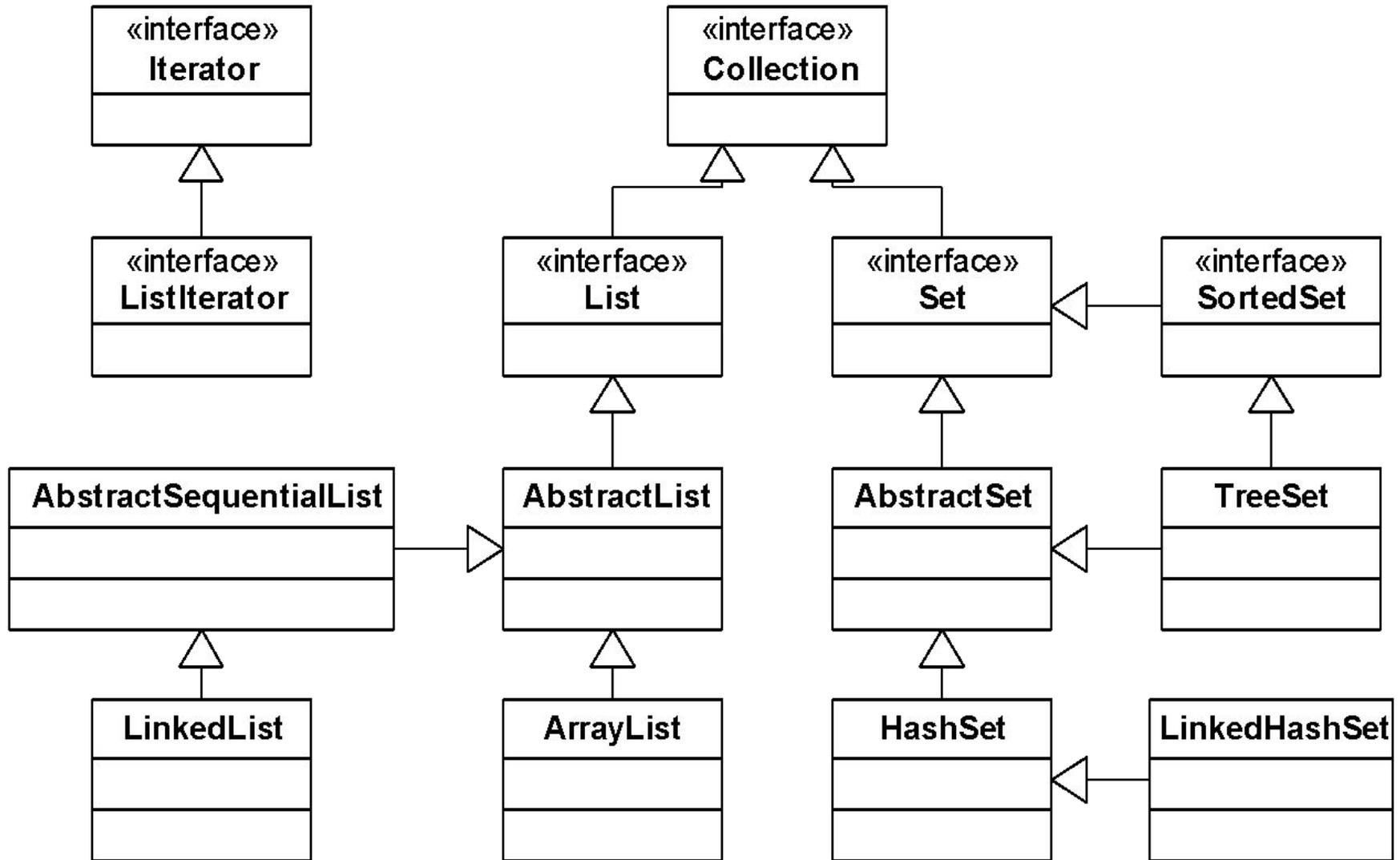
Коллекции

Коллекции

- Коллекция — неупорядоченный набор элементов
- Интерфейс `Collection`



Структура Collections Framework (1)



Немодифицирующие операции

- Определение размера
 - `size()` — количество элементов
 - `isEmpty()` — проверка на пустоту
- Проверки на вхождение
 - `contains(Object o)` — одного элемента
 - `containsAll(Collection c)` — всех элементов коллекции `c`

Модифицирующие операции

- Добавление элементов
 - `add(Object e)` — одного элемента
 - `addAll(Collection c)` — элементов коллекции
- Удаление элементов
 - `remove(Object e)` — одного элемента
 - `removeAll(Collection c)` — элементов коллекции
 - `retainAll(Collection c)` — удаление элементов не из коллекции
 - `clear()` — удаление всех элементов
- Исключения
 - `UnsupportedOperationException`

Пример. ?

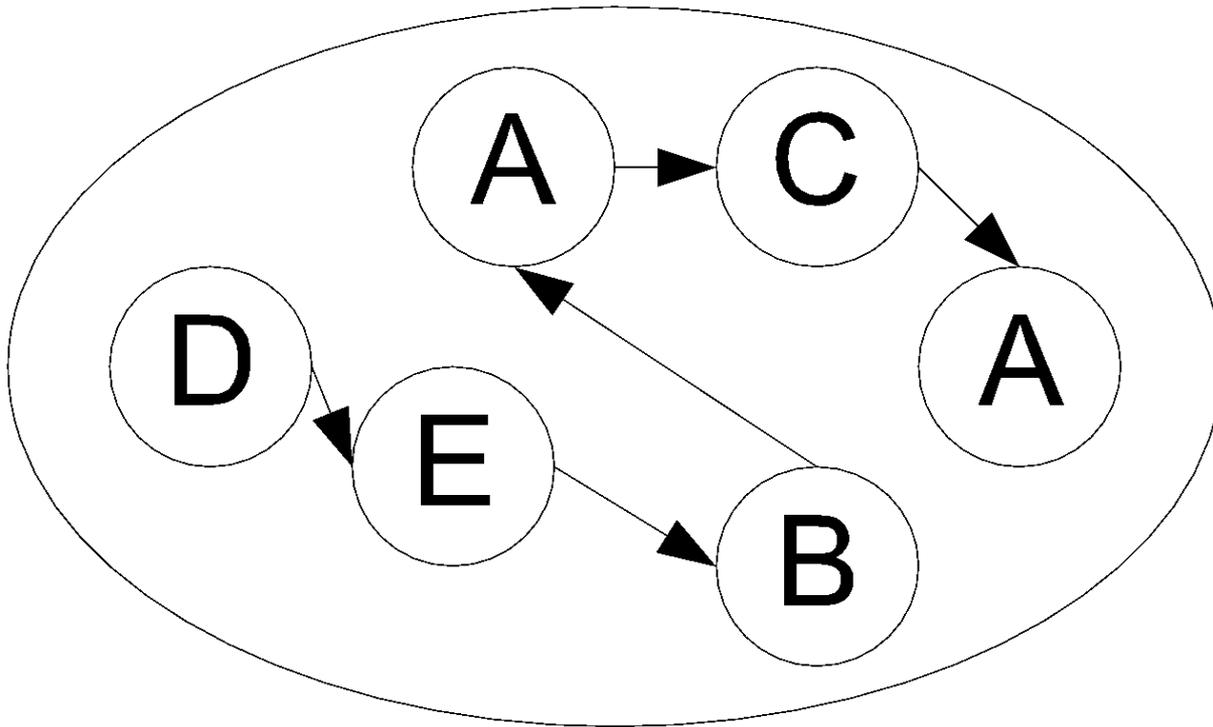
```
public int read(String file) throws IOException {
    Scanner scanner = new Scanner(
        new File(file), "Cp1251");

    int read = 0;
    while (scanner.hasNext()) {
        read++;
        c.add(scanner.next());
    }

    return read;
}
```

Итераторы

- Итератор — обход коллекции
- Интерфейс `Iterator`
- Метод `Iterator Collection.iterator()`



Методы итераторов

- `hasNext()` — определение наличия следующего элемента
- `next()` — взятие следующего элемента
- `remove()` — удаление элемента

- Исключения
 - `NoSuchElementException` — бросается при достижении конца коллекции
 - `ConcurrentModificationException` — бросается при изменении коллекции

Применение итераторов

- ?

```
for(Iterator i = c.iterator(); i.hasNext(); ) {  
    E element = (E) i.next();  
    ...  
}
```

- ?

```
for(Iterator i = c.iterator(); i.hasNext(); ) {  
    if (!p(i.next())) i.remove();  
}
```

Пример. ?

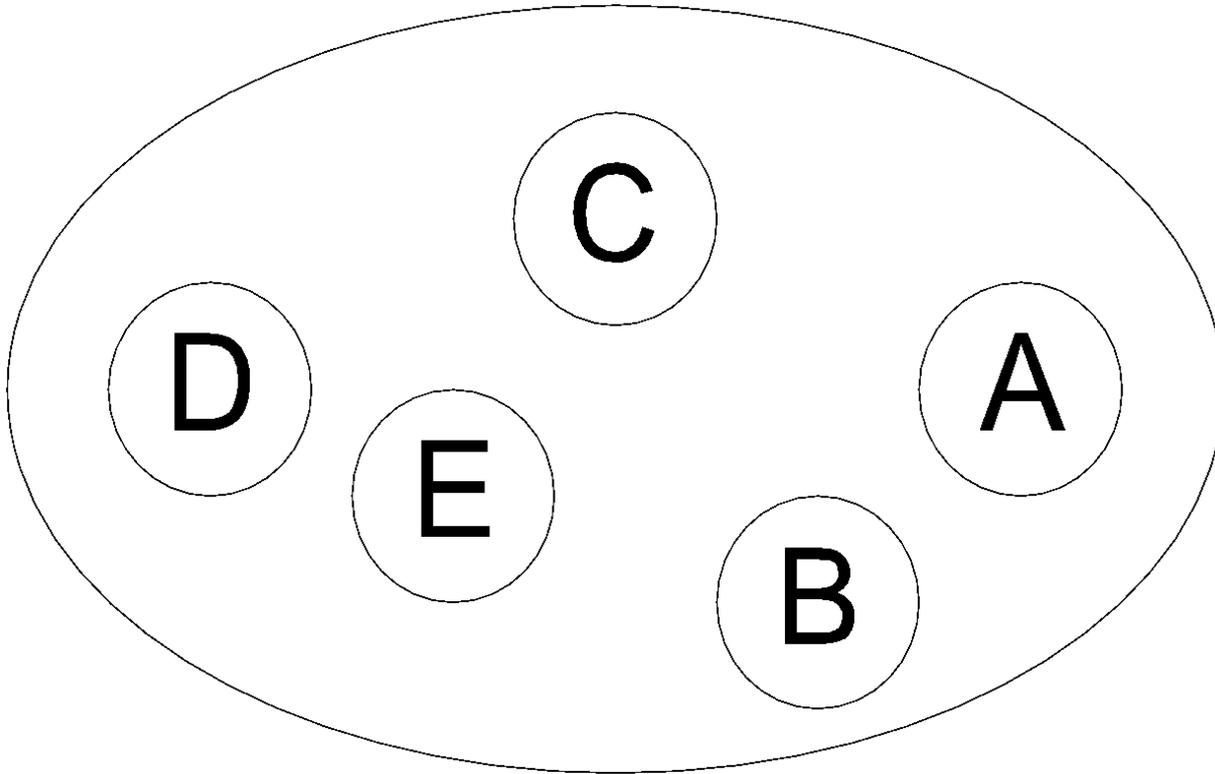
```
public void dump() {  
    for (Iterator i = c.iterator(); i.hasNext(); ) {  
        String word = (String) i.next();  
        System.out.print(word + ", ");  
    }  
    System.out.println();  
}
```

Часть 2

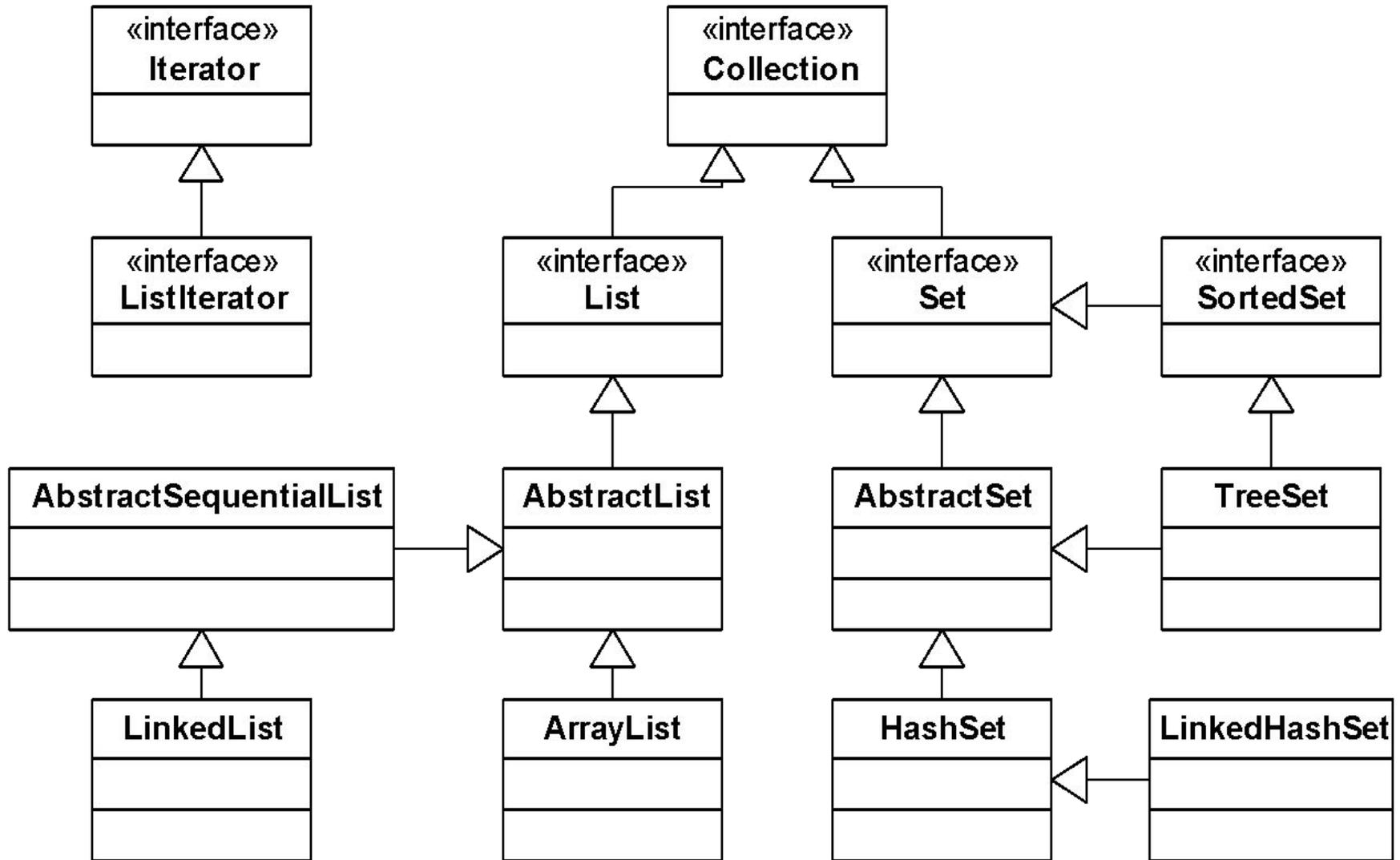
Множества

Множества

- Множество — коллекция без повторяющихся элементов
- Интерфейс `Set`



Структура Collections Framework (1)



Сравнение элементов

- Метод `Object.equals(Object object)`
- Рефлексивность
`o1.equals(o1)`
- Симметричность
`o1.equals(o2) == o2.equals(o1)`
- Транзитивность
`o1.equals(o2) && o2.equals(o3) => o1.equals(o3)`
- Устойчивость
`o1.equals(o2)` не изменяется, если `o1` и `o2` не изменяются
- Обработка `null`
`o1.equals(null) == false`

Операции над множествами

- `addAll(Collection c)` – объединение
МНОЖЕСТВ
- `retainAll(Collection c)` – пересечение
МНОЖЕСТВ
- `containsAll(Collection c)` – проверка
ВХОЖДЕНИЯ
- `removeAll(Collection c)` – разность
МНОЖЕСТВ

Классы HashSet и LinkedHashSet

- HashSet — множество на основе хэша
- LinkedHashSet — множество на основе хэша с сохранение порядка обхода

Вычисление хэшей

- Метод `Object.hashCode()`
- Устойчивость
`hashCode()` не изменяется, если объект не изменяется
- Согласованность с `equals`
`o1.equals(o2) => o1.hashCode() == o2.hashCode()`

Конструкторы HashSet

- `HashSet()` — пустое множество
- `HashSet(Collection c)` — элементы коллекции
- `HashSet(int initialCapacity)` — начальная вместимость
- `HashSet(int initialCapacity, double loadFactor)` — начальная вместимость и степень заполнения

Пример. ?

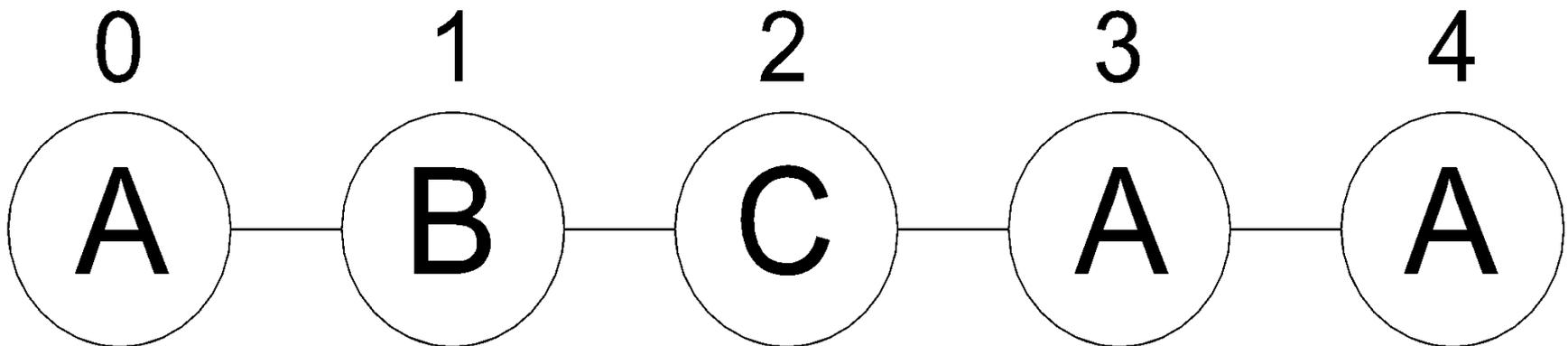
```
CollectionExample c =  
    new CollectionExample(new HashSet());  
  
int words = c.read(args[0]);  
System.out.println("? total: " + words);  
System.out.println("? words: " +  
    c.getCollection().size());  
  
c.dump();
```

Часть 3

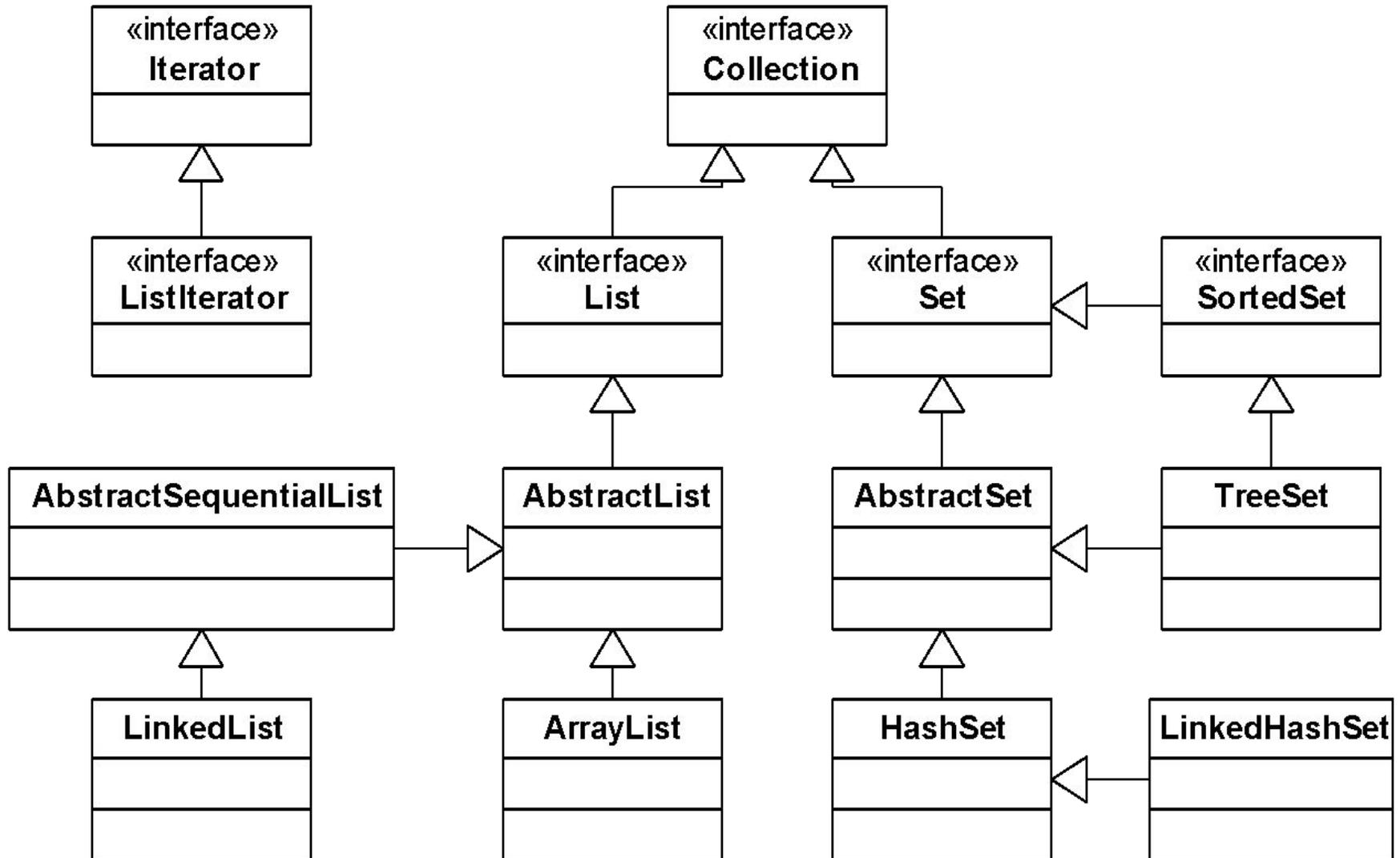
Списки

Списки

- Список — коллекция с индексированными элементами
- Интерфейс `List`



Структура Collections Framework (1)

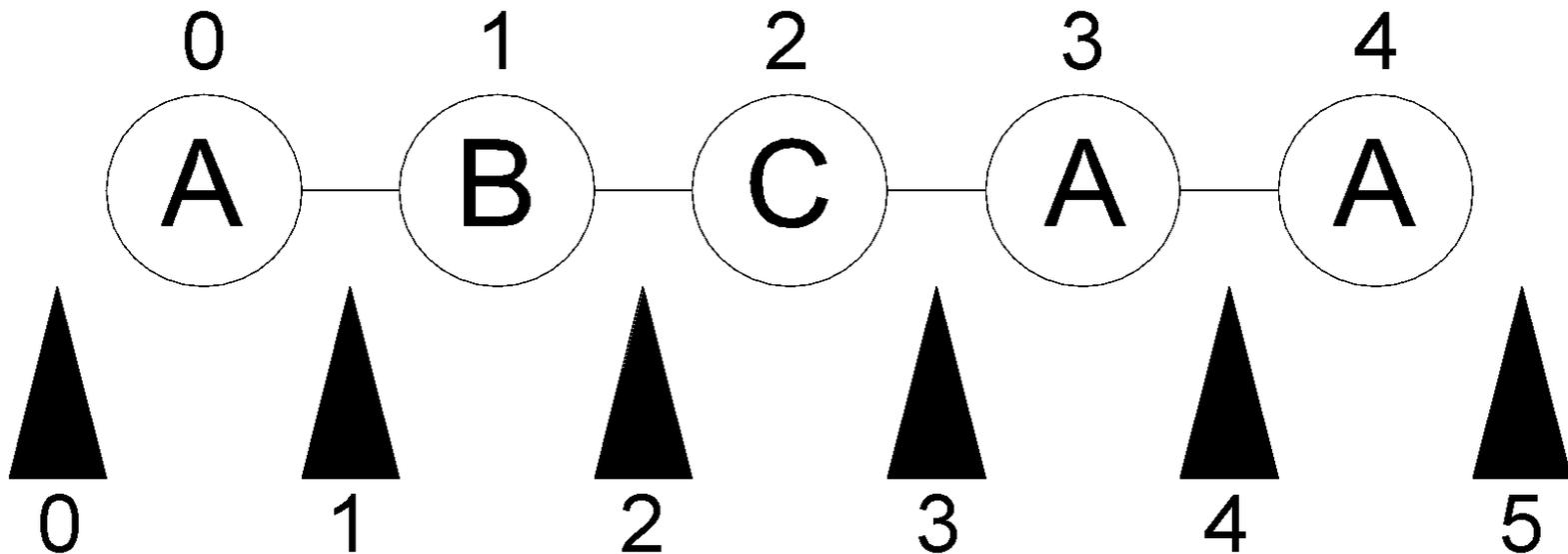


Операции со списками

- Доступ по индексу
 - `get(int i)` — чтение
 - `set(int I, Object e)` — запись
 - `add(int i, Object e)` — добавление
 - `remove(int i)` — удаление
- Поиск элементов
 - `indexOf(Object e)` — поиск с начала
 - `lastIndexOf(Object e)` — поиск с конца
- Взятие вида
 - `List subList(int from, int to)`

Итератор по списку

- Интерфейс `ListIterator` extends `Iterator`
- Метод `listIterator()`
- Предыдущий / Следующий элементы



Операции итератора по списку

- Передвижение
 - `hasNext()` / `hasPrevious()` — проверка
 - `next()` / `previous()` — взятие элемента
 - `nextIndex()` / `previousIndex()` — определение индекса
- Изменение
 - `remove()` — удаление элемента
 - `set(Object e)` — изменение элемента
 - `add(Object e)` — добавление элемента

Класс ArrayList

- `ArrayList` — список на базе массива
- Плюсы
 - Быстрый доступ по индексу
 - Быстрая вставка и удаление элементов с конца
- Минусы
 - Медленная вставка и удаление элементов

Вместимость ArrayList

- Вместимость — реальное количество элементов
- Дополнительные методы
 - `ensureCapacity(int c)` — определение вместимости
 - `trimToSize()` — “подгонка” вместимости

Конструкторы ArrayList

- `ArrayList()` — пустой список
- `ArrayList(Collection c)` — копия коллекции
- `ArrayList(int initialCapacity)` — пустой список заданной вместимости

Применения ArrayList

- “Бесконечный” массив
- Стек

Пример. ?

```
List list = new ArrayList();
```

```
...
```

```
for (int i = list.size() - 1; i >= 0; i--) {  
    System.out.println(list.get(i));  
}
```

Класс LinkedList

- `LinkedList` — двусвязный список
- Плюсы
 - Быстрое добавление и удаление элементов
- Минусы
 - Медленный доступ по индексу

Возможности LinkedList

- Конструкторы
 - `LinkedList()` — пустой список
 - `LinkedList(Collection c)` — копия коллекции
- Методы
 - `addFirst(Object o)` — добавить в начало списка
 - `addLast(Object o)` — добавить в конец списка
 - `removeFirst()` — удалить первый элемент
 - `removeLast()` — удалить последний элемент

Применения LinkedList

- Стек
- Очередь
- Дек

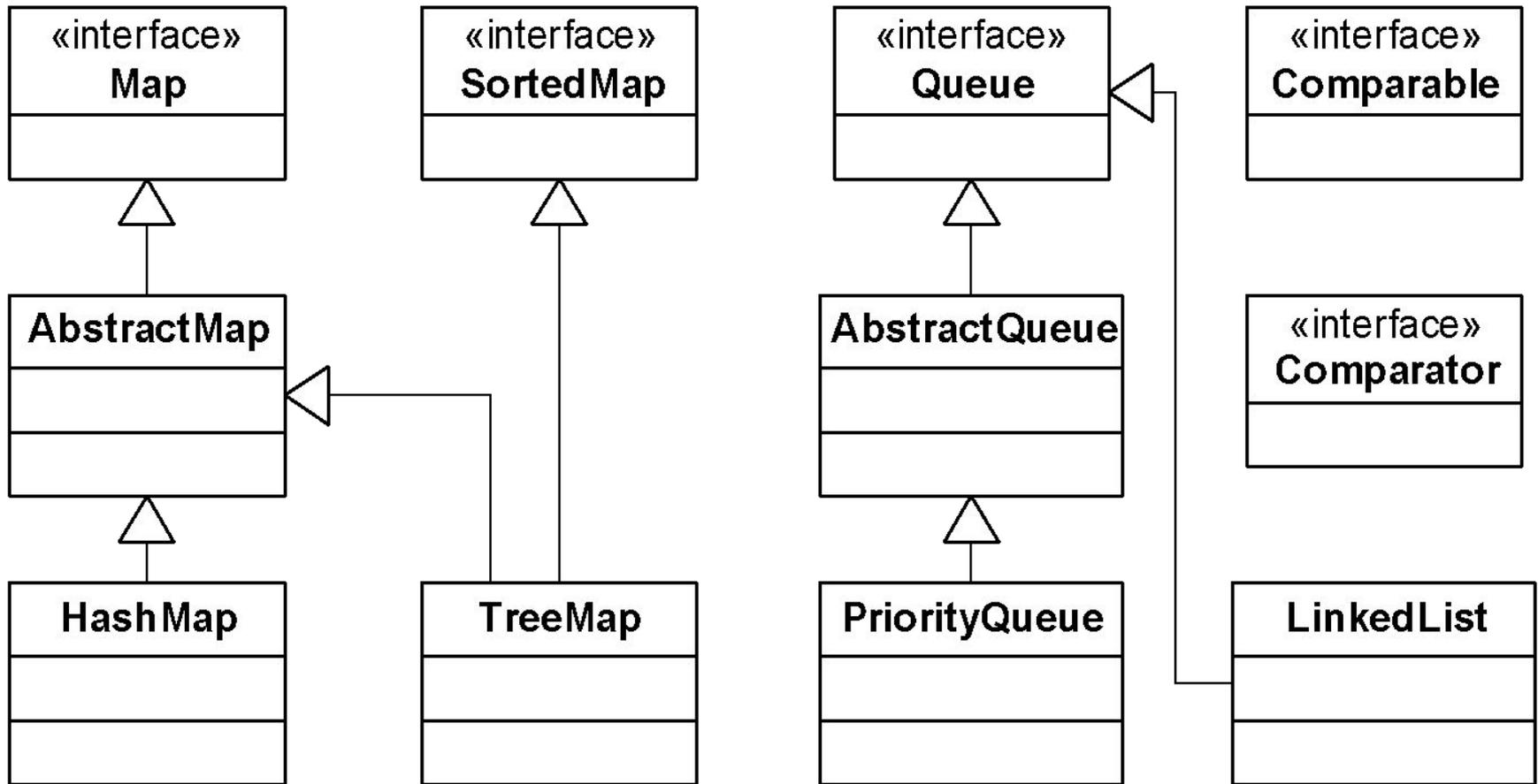
Пример. ?

```
List list = new LinkedList();  
...  
for (ListIterator li = list.listIterator(list.size());  
    li.hasPrevious(); )  
{  
    System.out.println(li.previous());  
}
```

Часть 4

Очереди

Структура Collections Framework (2)



Очередь

- Очередь – хранилище элементов для обработки
- Интерфейс `Queue`
- Свойства очередей
 - Порядок выдачи элементов определяется конкретной реализацией
 - Очереди не могут хранить `null`
 - У очереди может быть ограничен размер

Методы очередей

- Обычные методы
 - `add(Object o)` – добавить элемент
 - Бросает `IllegalStateException`
 - `Object element()` – вершина очереди
 - Бросает `NoSuchElementException`
 - `Object remove()` – удалить элемент из вершины
 - Бросает `NoSuchElementException`
- Методы, не бросающие исключений
 - `offer(Object o)` – добавить элемент
 - `Object peek()` – вершина очереди
 - `Object poll()` – удалить элемент из вершины

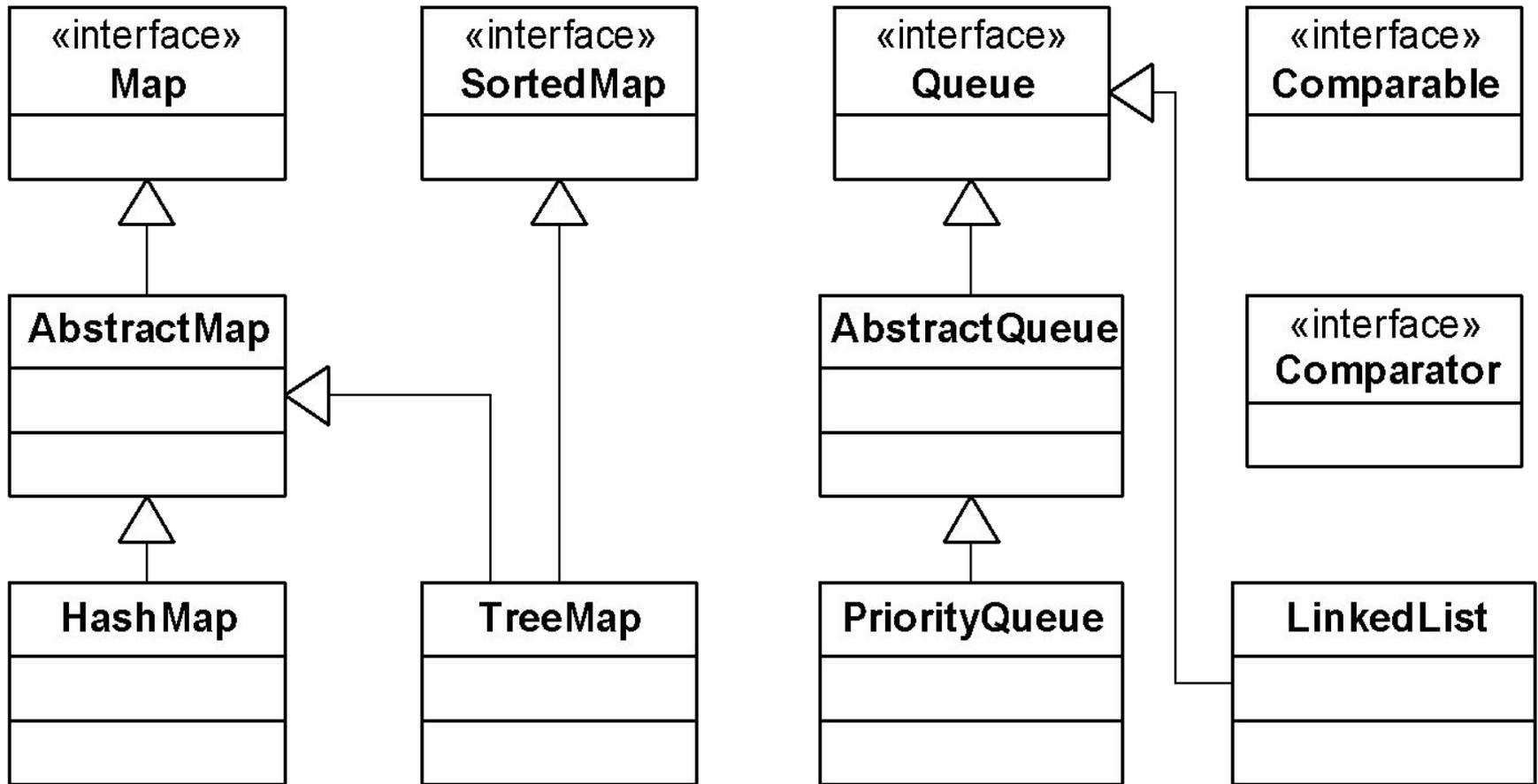
Класс LinkedList

- Очередь на двусвязном списке

Часть 5

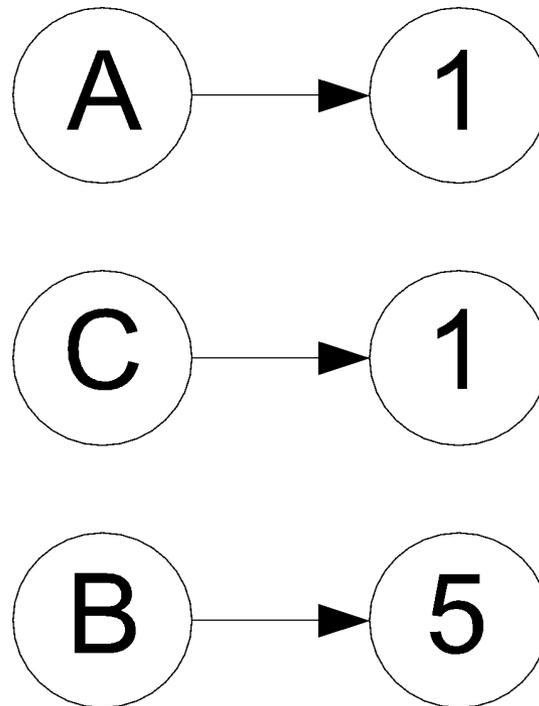
Отображения

Структура Collections Framework (2)



Отображение

- Отображение — множество пар ключ-значение при уникальности ключа
- Интерфейс `Map`



Методы отображений (1)

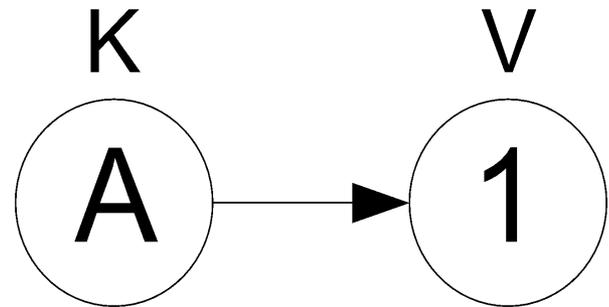
- Доступ
 - `get(Object k)` — получение значение
 - `put(Object k, Object v)` — запись
 - `remove(Object k)` — удаление
- Проверки
 - `containsKey(Object k)` — наличие ключа
 - `containsValue(Object v)` — наличие значения
- Определения размера
 - `size()` — размер отображения
 - `isEmpty()` — проверка на пустоту

Методы отображений (2)

- Взятие видов
 - `entrySet()` — множество пар
 - `values()` — коллекция значений
 - `keySet()` — множество ключей
- Массовые операции
 - `putAll(Map map)` — добавление всех пар

Пары

- Пара — ключ + значение
- Интерфейс `Map.Entry`
- Методы
 - `Object getKey()`
 - `Object getValue()`
 - `setValue(Object v)`



Классы HashMap и LinkedHashMap

- `HashMap` — отображение на основе хэшей
- `LinkedHashMap` — отображение на основе хэшей с сохранением порядка обхода

Конструкторы HashMap

- `HashMap()` — пустое отображение
- `HashMap(Map m)` — копия отображения
- `HashMap(int initialCapacity)` — начальная ВМЕСТИМОСТЬ
- `HashMap (int initialCapacity, int loadFactor)` — начальная ВМЕСТИМОСТЬ и степень заполнения

Пример. ?

```
while (scanner.hasNext()) {  
    String word = scanner.next();  
    Integer count = (Integer) map.get(word);  
    int value = (count == null)  
        ? 0  
        : count.intValue();  
    map.put(word, new Integer(value + 1));  
}
```

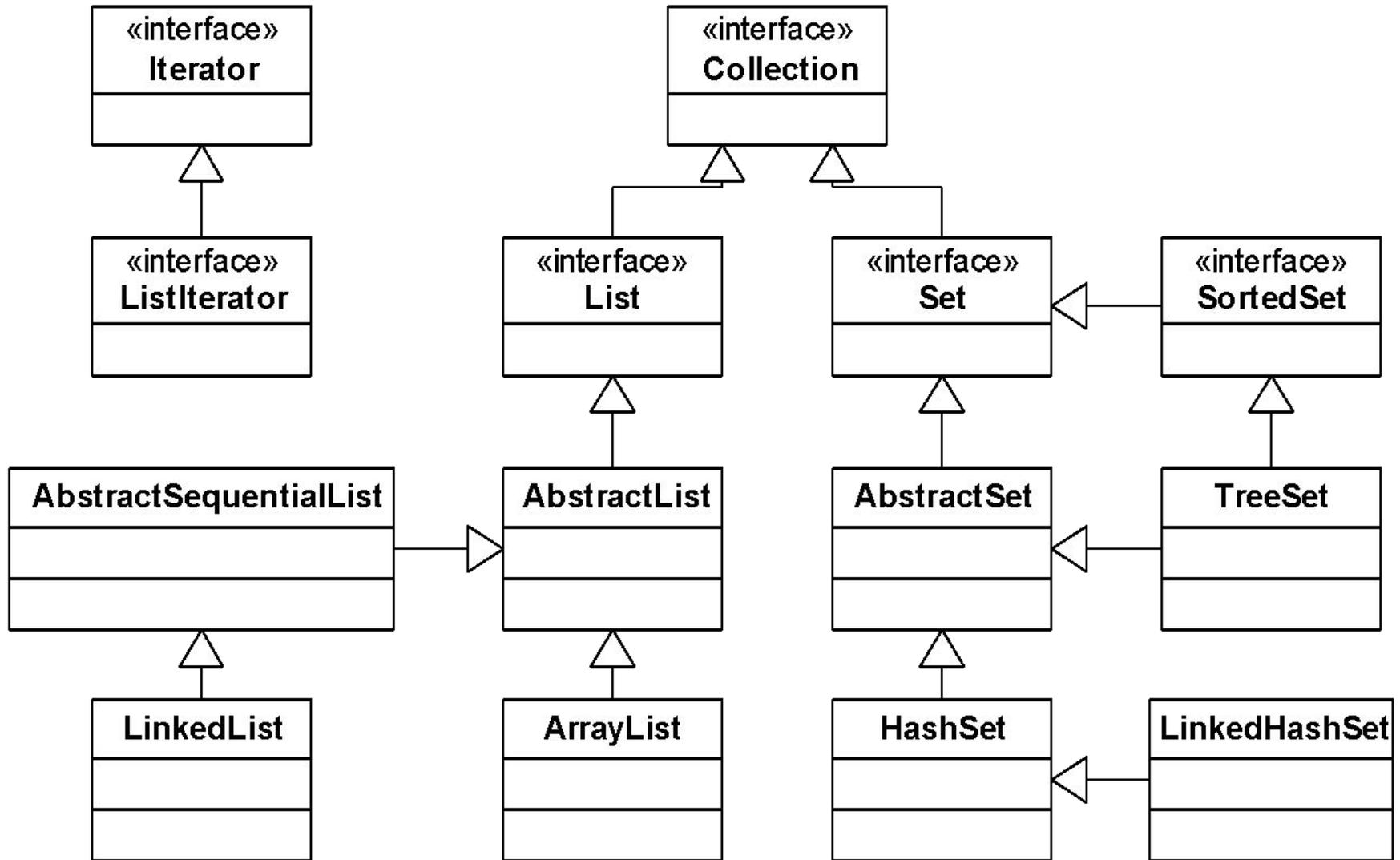
Пример. ?

```
for (  
    Iterator i = map.entrySet().iterator();  
    i.hasNext();  
) {  
    Map.Entry entry = (Map.Entry) i.next();  
    System.out.println(  
        entry.getKey() + " " + entry.getValue());  
}
```

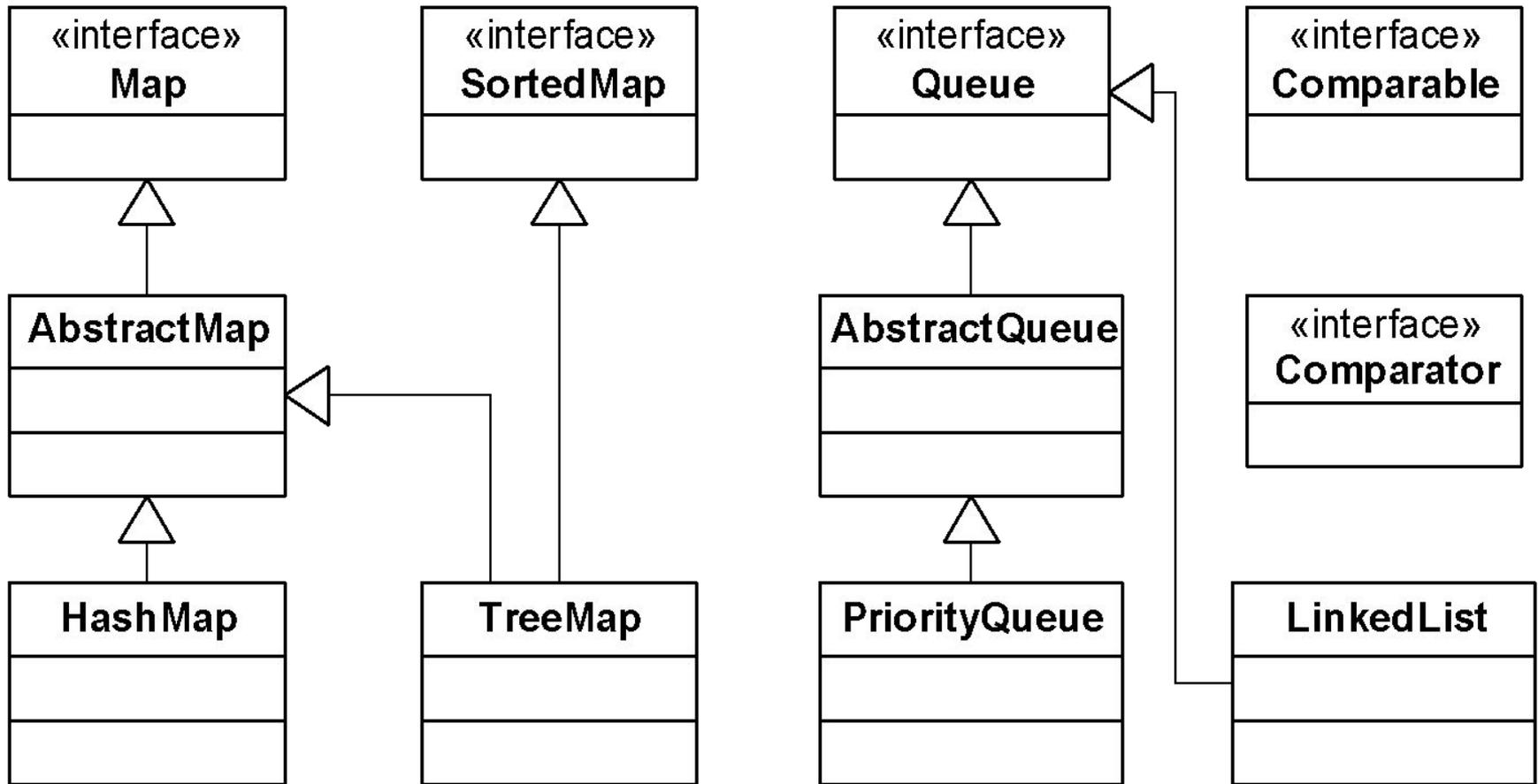
Часть 6

Упорядоченные коллекции

Структура Collections Framework (1)



Структура Collections Framework (2)



Сравнение элементов

- Интерфейс `Comparable`
 - `int compareTo(Object o)` — естественный порядок
- Интерфейс `Comparator`
 - `int compare(Object o1, Object o2)` — сравнение элементов

< = >
- 0 +

Сравнение элементов (контракт)

- Транзитивность
- Антисимметричность
 - $\text{sgn}(o1.\text{compareTo}(o2)) == -\text{sgn}(o2.\text{compareTo}(o1))$
- Согласованность с равенством
 - $o1.\text{compareTo}(o2) == 0 \Rightarrow$
 $\text{sgn}(o1.\text{compareTo}(o3)) == \text{sgn}(o2.\text{compareTo}(o3))$
- Согласованность с `equals()`
 - $o1.\text{equals}(o2) == (o1.\text{compareTo}(o2) == 0)$

Упорядоченные множества

- Интерфейс `SortedSet`
 - `first()` – минимальный элемент
 - `last()` – максимальный элемент
 - `headSet(Object o)` – подмножество элементов меньших `o`
 - `tailSet(Object o)` – подмножество элементов больших либо равных `o`
 - `subSet(Object o1, Object o2)` – подмножество элементов меньших `o2` и больше либо равных `o1`
- Класс `TreeSet`

Упорядоченные отображения

- Интерфейс `SortedMap`
 - `firstKey()` – минимальный ключ
 - `lastKey()` – максимальный ключ
 - `headMap(Object o)` – отображение ключей меньших `o`
 - `tailMap(Object o)` – отображение ключей больших либо равных `o`
 - `subMap(Object o1, Object o2)` – отображение ключей меньших `o2` и больше либо равных `o1`
- Класс `TreeMap`

Класс PriorityQueue

- Очередь с приоритетами
- Реализована на основе двоичной кучи

Пример. Применение TreeSet

- Естественный порядок

```
CollectionExample c = new CollectionExample(  
    new TreeSet());  
c.read(args[0]);  
c.dump();
```

- Порядок без учета регистра

```
CollectionExample c = new CollectionExample(new  
    TreeSet(String.CASE_INSENSITIVE_ORDER));  
int words = c.read(args[0]);  
c.dump();
```

Часть 7

Алгоритмы

Класс Collections

- Алгоритмы для работы с коллекциями
 - Простые операции
 - Перемешивание
 - Сортировка
 - Двоичный поиск
 - Поиск минимума и максимума
- Специальные коллекции
- Оболочки коллекций

Простые операции

- Заполнение списка указанным значением
 - `fill(List l, Object v)`
- Переворачивание списка
 - `reverse(List l)`
- Копирование из списка в список
 - `copy(List l1, List l2)`

Перемешивание

- Генерирует случайную перестановку
- Методы
 - `shuffle(List l)`
 - `shuffle(List l, Random r)`

Сортировки

- Устойчивая сортировка
- Алгоритм – Merge Sort
- Методы
 - `sort(List l)` – сортировка списка (естественный порядок)
 - `sort(List l, Comparator c)` – сортировка списка (указанный порядок)

Двоичный поиск

- Осуществляет двоичный поиск в списке
- Методы
 - `binarySearch(List l, Object o)` – ищет `o` в списке
 - `binarySearch(List l, Object o, Comparator c)` – ищет `o` в списке

Поиск минимума и максимума

- Поиск минимума
 - `min(Collection c)` – минимальный элемент (естественный порядок)
 - `min(Collection c, Comparator cmp)` – минимальный элемент (указанный порядок)
- Поиск максимума
 - `max(Collection c)` – максимальный элемент (естественный порядок)
 - `max(Collection c, Comparator cmp)` – максимальный элемент (указанный порядок)

Пример. Алгоритмы на списках

```
List list = new ArrayList();  
CollectionExample c = new CollectionExample(list);  
c.read(args[0]);
```

```
Collections.reverse(list);  
Collections.shuffle(list);  
Collections.sort(list);  
Collections.sort(list,  
    String.CASE_INSENSITIVE_ORDER);  
Collections.fill(list, "temp");  
System.out.println(Collections.min(list));  
System.out.println(Collections.min(list,  
    String.CASE_INSENSITIVE_ORDER));
```

Оболочки коллекций

- Неизменяемые виды на коллекции
 - `unmodifiableSet(Set s)` – неизменяемое МНОЖЕСТВО
 - `unmodifiableSortedSet(SortedSet s)` – неизменяемое упорядоченное множество
 - `unmodifiableList(List l)` – неизменяемый список
 - `unmodifiableMap(Map m)` – неизменяемое отображение
 - `unmodifiableSortedMap(SortedMap m)` – неизменяемое упорядоченное отображени

Класс Arrays

- Операции с массивами
 - Сортировка
 - Двоичный поиск
 - Поиск минимума и максимума
 - Заполнение
 - Перемешивание
- Вид массива как списка
 - `List asList()`

Часть 8

Устаревшие коллекции

Устаревшие коллекции

- Устаревшие коллекции являются синхронизированными
- **Vector** (**ArrayList**)
 - **Stack** (**ArrayList**)
- **Dictionary** (**Map**)
 - **Hashtable** (**HashMap**)
- **Enumeration** (**Iterator**)

Часть 9

Заключение

- Collections Framework // <http://java.sun.com/j2se/1.5.0/docs/guide/collections/index.html>
- Collections Tutorial // <http://java.sun.com/docs/books/tutorial/collections/index.html>
- Introduction to the Collections Framework // <http://java.sun.com/developer/onlineTraining/collections/>

Вопросы