Урок 8.
Коллекции в Java

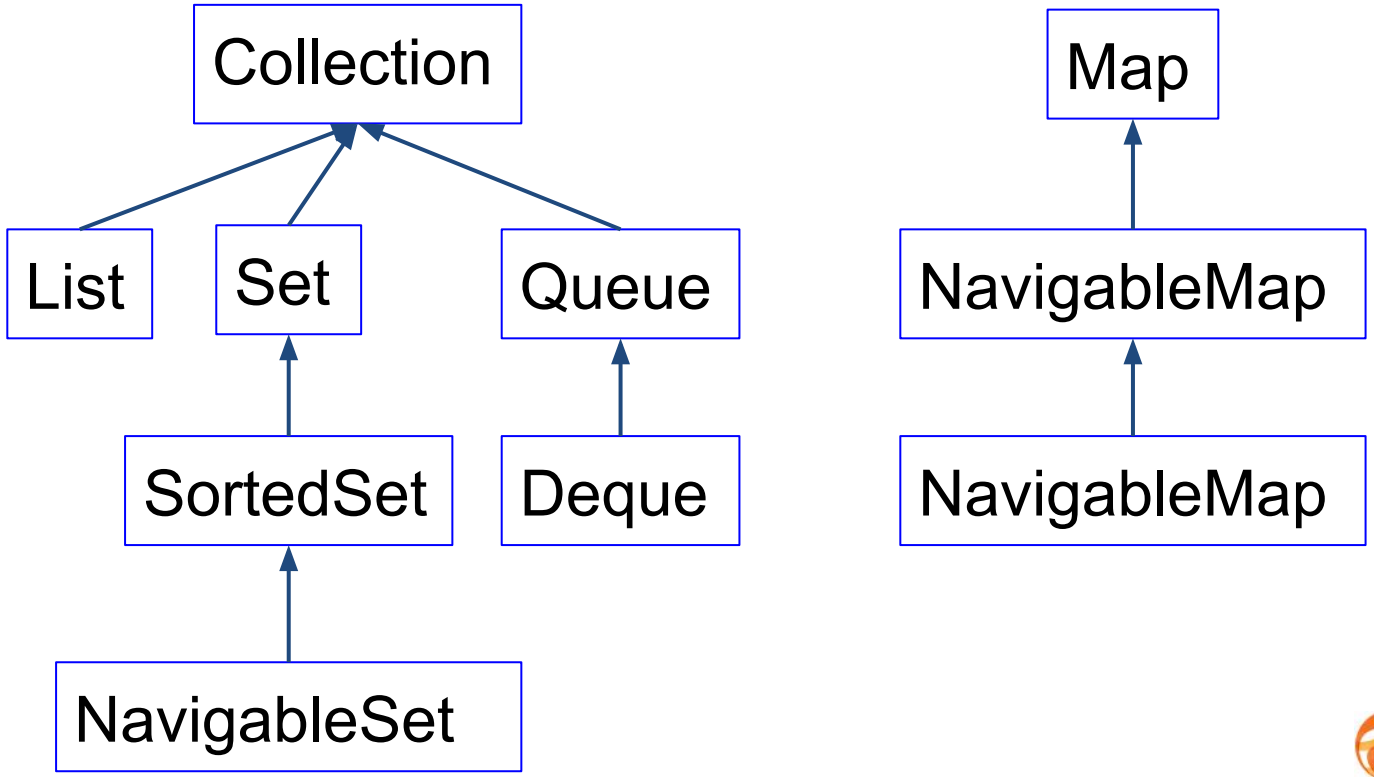
Гарбузов Николай
Donriver Inc.

DonRiver

Agenda

- Виды контейнеров в Java
- Основные реализации контейнеров
- Примеры использования контейнеров
- Проход по элементам коллекции
- Интерфейсы Comparator & Comparable
- Сравнение и сортировка элементов коллекции

Виды контейнеров в Java



Основные реализации контейнеров

Collection<E> - интерфейс коллекции, в котором объявлены основные методы для работы с коллекцией

- boolean add (E e);

- boolean contains(Object o);
- boolean isEmpty();

- Iterator<E> iterator();

- boolean remove(Object o);

- Object[] toArray();
- <T> T[] toArray(T arr[]);

Основные реализации контейнеров

Collection<E> - интерфейс коллекции, в котором объявлены основные методы для работы с коллекцией

- boolean add (E e);
- boolean contains(Object o);
- boolean isEmpty();
- Iterator<E> iterator();
- boolean remove(Object o);
- Object[] toArray();
- <T> T[] toArray(T arr[]);

Нет get метода!

Основные реализации контейнеров

List<E> - упорядоченный динамически расширяемый список элементов типа E

- **ArrayList<E>** - реализация на основе массива
- **LinkedList<E>** - реализация на основе двусвязного списка

Основные реализации контейнеров

List<E> - упорядоченный динамически расширяемый список элементов типа E

- **ArrayList<E>** - реализация на основе массива
- **LinkedList<E>** - реализация на основе двусвязного списка

- **Vector<E>** - реализация на основе массива с синхронизованными операциями
- **Stack<E>** - реализация стека на основе массива с синхронизованными операциями

Основные реализации контейнеров

List<E> - упорядоченный динамически расширяемый список элементов типа E

- **ArrayList<E>** - реализация на основе массива
- **LinkedList<E>** - реализация на основе двусвязного списка

- **Vector<E>** - реализация на основе массива с синхронизованными операциями
- **Stack<E>** - реализация стека на основе массива с синхронизованными операциями

Основные реализации контейнеров

List<E> - упорядоченный динамически расширяемый список элементов типа E

- **ArrayList<E>** - реализация на основе массива
- **LinkedList<E>** - реализация на основе двусвязного списка

- **Vector<E>** - реализация на основе массива с синхронизованными операциями
- **Stack<E>** - реализация стека на основе массива с синхронизованными операциями

Есть метод get(index) !

Основные реализации контейнеров

Queue<E> - очередь элементов типа E

- **ArrayDeque<E>** - реализация Deque (двусторонней очереди) на основе массива
- **LinkedList<E>** - реализация очереди на основе списка
- **PriorityQueue<E>** - реализация очереди с учетом приоритетов элементов

Основные реализации контейнеров

Queue<E> - очередь элементов типа E

- **ArrayDeque<E>** - реализация Deque (двусторонней очереди) на основе массива
- **LinkedList<E>** - реализация очереди на основе списка
- **PriorityQueue<E>** - реализация очереди с учетом приоритетов элементов

Есть методы для работы с головой и хвостом очереди!

Основные реализации контейнеров

Queue<E> - очередь элементов типа E

- **ArrayDeque<E>** - реализация Deque (двусторонней очереди) на основе массива
- **LinkedList<E>** - реализация очереди на основе списка
- **PriorityQueue<E>** - реализация очереди с учетом приоритетов элементов

- **ArrayBlockingQueue<E>** - реализация блокирующей очереди на основе массива
- **LinkedBlockingQueue<E>** - реализация блокирующей очереди на основе массива

Основные реализации контейнеров

Set<E> - неупорядоченное (либо упорядоченное) множество элементов типа E

- **HashSet<E>** - реализация на основе HashMap<E>
- **LinkedHashSet<E>** - реализация на основе LinkedHashMap<E>
- **TreeSet<E>** - реализация на основе TreeMap<E>

Основные реализации контейнеров

Set<E> - неупорядоченное (либо упорядоченное) множество элементов типа E

- **HashSet<E>** - реализация на основе HashMap<E>
- **LinkedHashSet<E>** - реализация на основе LinkedHashMap<E>
- **TreeSet<E>** - реализация на основе TreeMap<E>

Множество не допускает дубликатов элементов,
т.е. все элементы во множестве уникальны!

Основные реализации контейнеров

Коллекции, предоставляющие эффективные произвольный доступ реализуют интерфейс-маркер **RandomAccess**!

Основные реализации контейнеров

Map<K,V> - отображение сохраняющее связи “ключ-значение”

- V get(K key);
- Set<K> keySet();

- V put(K k, V v);
- V replace(K k, V v);

- V remove(Object k);

Основные реализации контейнеров

Map<K,V> - отображение сохраняющее связи “ключ-значение”

- **HashMap<K,V>** - реализация на основе корзин, используют хэш-код
- **LinkedHashMap<K,V>** - похоже на HashMap, но так же реализует связный список элементов
- **TreeMap<K,V>** - реализация NavigableMap<K,V> на основе бинарного дерева

В Map не допускается дубликатов ключей, т.е. все ключи уникальны!

Основные реализации контейнеров

Для использование объекта в качестве ключа, он должен переопределять методы

- boolean equals(Object o);
- int hashCode();

Основные реализации контейнеров

Для использование объекта в качестве ключа, он должен переопределять методы

- boolean equals(Object o);
- int hashCode();

- o1.equals(o2); -> o1.hashCode() == o2.hashCode();

Основные реализации контейнеров

Для использование объекта в качестве ключа, он должен переопределять методы

- boolean equals(Object o);
- int hashCode();

- o1.equals(o2); -> o1.hashCode() == o2.hashCode();
- o1.hashCode() == o2.hashCode(); не обязательно -> o1.equals(o2);

Проход по элементам коллекции

Iterator<E> используется для перебора элементов коллекции Collection<E>

- boolean hasNext();
- E next();

Проход по элементам коллекции

Iterator<E> используется для перебора элементов коллекции Collection<E>

- boolean hasNext();
- E next();

получается с помощью метода интерфейса Iterable<E>

- Iterator<E> iterator();

Проход по элементам коллекции

Iterator<E> используется для перебора элементов коллекции Collection<E>

- boolean hasNext();
- E next();

получается с помощью метода интерфейса Iterable<E>

- Iterator<E> iterator();

- Используется неявно в for-each

Проход по элементам коллекции

Iterator<E> используется для перебора элементов коллекции Collection<E>

- boolean hasNext();
- E next();
- void remove();

получается с помощью метода интерфейса Iterable<E>

- Iterator<E> iterator();
  - Используется неявно в for-each
  - Позволяет удалять элементы!

Проход по элементам коллекции

Spliterator используется для перебора элементов коллекции Collection<E>

Проход по элементам коллекции

Spliterator используется для перебора элементов коллекции Collection<E> и поддерживает параллельную итерацию!

- boolean tryAdvance(Consumer<? super E> cons);

DonRiver

Интерфейсы Comparator & Comparable

Comparator<T> используется для сравнения объектов типа T

- int compare(T o1, T o2);
- boolean equals(T o1, T o2);

Интерфейсы Comparator & Comparable

Comparator<T> используется для сравнения объектов типа T

- int compare(T o1, T o2);
- boolean equals(T o1, T o2);

- много полезных методов (default & static)

DonRiver

Сравнение и сортировка элементов коллекции

Collections предоставляет много методов для работы с коллекциями, в том числе

- int binarySearch(...);
- void sort(...);

- void swap(List<?> lst, int i1, int i2);

- <T> Collection<T> unmodifiableCollection(Collection<? extends T> c)
- <T> Collection<T> synchronizedCollection(Collection<? extends T> c)

Home work

- создание словаря по введенному тексту
- определить “цикл” в LinkedList
- распечатать TreeMap в “красивом виде” с форматированием отступами