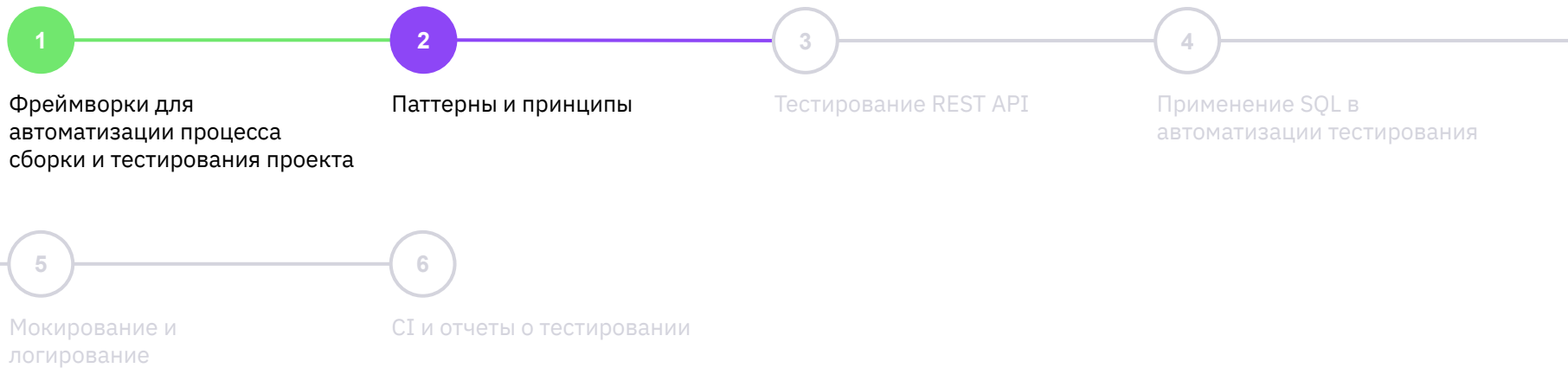


Java Development Kit

Урок 4
Коллекции








План курса





Чему посвящен урок

-  Массивы
-  Интерфейс List
-  Интерфейс Set
-  Интерфейс Queue
-  Интерфейс Map





big O





О большое (Big O)

— это оценка верхней асимптотической (оценка при наихудшем сценарии) границы времени работы алгоритма в зависимости от количества элементов n , которые необходимо обработать





Сортировка выбором



В не отсортированном подмассиве ищется локальный максимум (минимум)



Найденный максимум (минимум) меняется местами с последним (первым) элементом в подмассиве

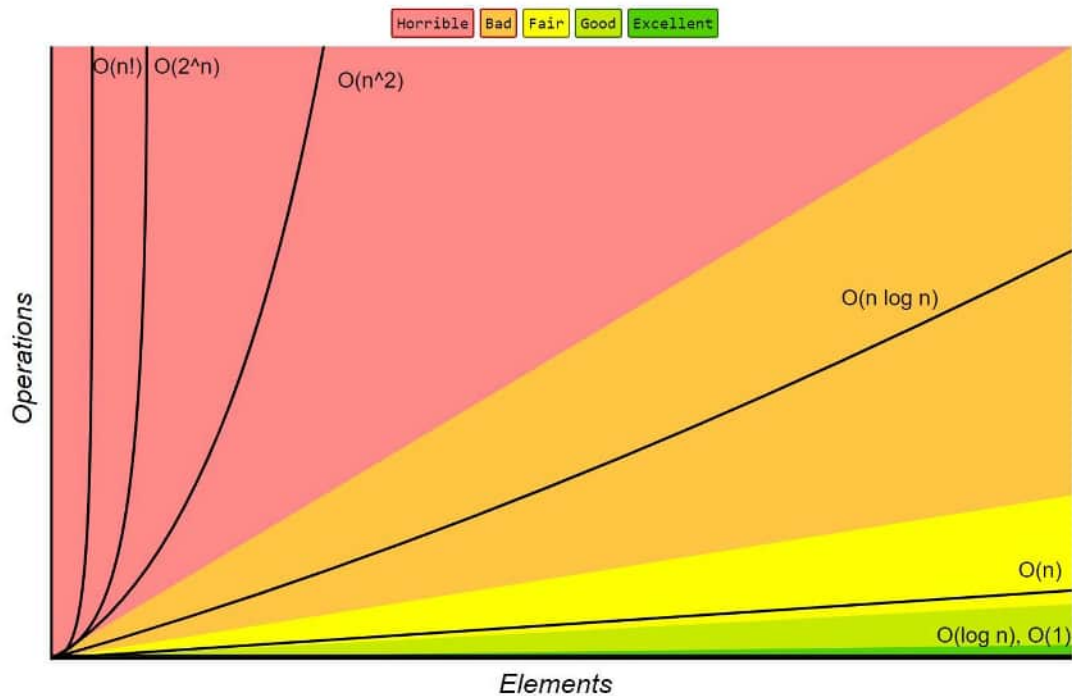


Если в массиве остались неотсортированные подмассивы — повторяем пункт 1.



Примеры O big

Big-O Complexity Chart





Примеры O big



$O(1)$ или O константа — получение и вставка элемента в i позицию массива



$O(\log n)$ — двоичное (бинарное дерево) поиск в вставка элемента



$O(n)$ — прохождение по элементам массива, списка (цикл for each



$O(n \log n)$ — алгоритм быстрой сортировки



$O(n^2)$ — сортировка пузырьком, сортировка вставкой (двойной цикл)



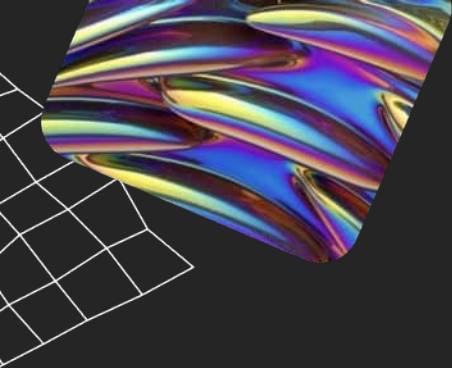
$O(2^n)$ — рекурсивный расчет чисел Фибоначчи



$O(n!)$ — задача коммивояжера



Если вы сравниваете между собой два
алгоритма с одинаковыми O большое,
то это ничего не говорит о том какой
из алгоритмов эффективнее





Коллекции





Массив в Java (Java Array)

— это структура данных, которая хранит набор пронумерованных значений одного типа (элементы массива)



O big и массивы



Получение элемента массива $O(1)$



Поиск элемента в массиве $O(n)$



Поиск и замена элемента в массиве $O(n)$



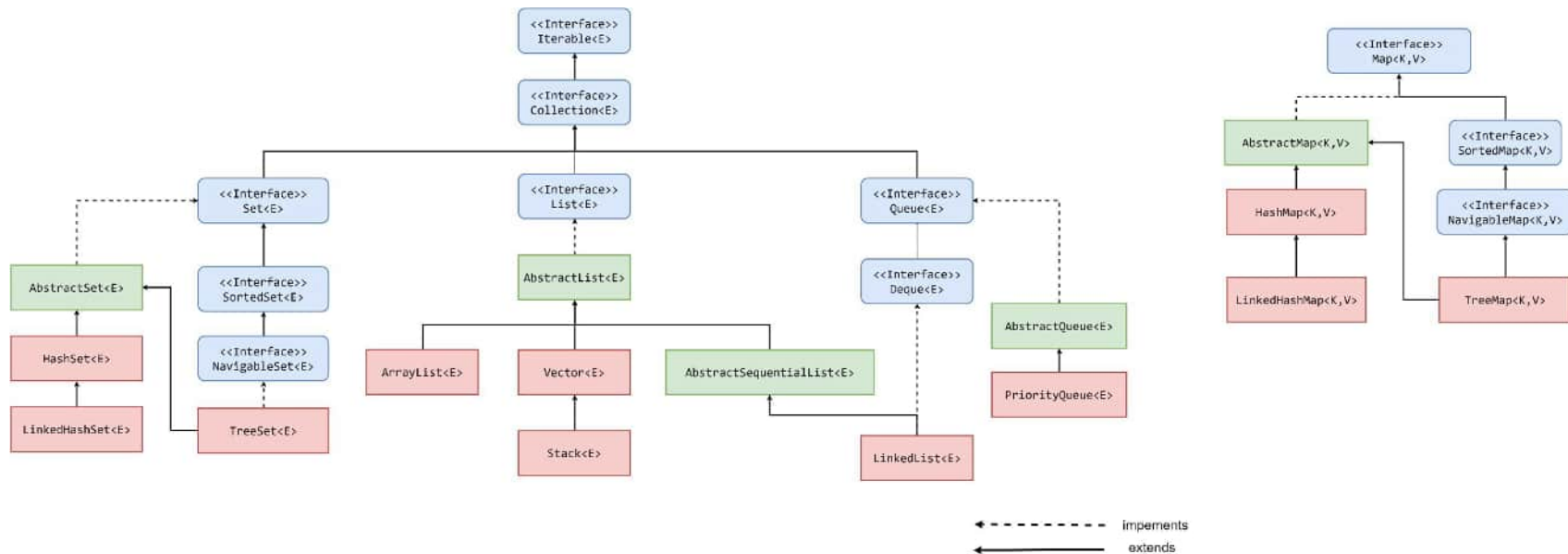
Ограничения

- 💡 Фиксированный размер.
- 💡 Требуется заранее знать количество элементов.
- 💡 Нерациональное использование памяти.
- 💡 “Примитивный интерфейс”





Иерархия коллекций





Обход элементов коллекции



Iterator



For each



Stream API



Обход элементов коллекции



Iterator



For each



Stream API

```
1 public static void main(String[] args) {  
2     Collection<String> colors = List.of("Белый", "Черный",  
    "Красный", "Зеленый", "Синий");  
3  
4     Iterator<String> iterator = colors.iterator();  
5     while (iterator.hasNext()) {  
6         System.out.println(iterator.next());  
7     }  
8 }
```




Обход элементов коллекции



Iterator



For each



Stream API

```
1 public static void main(String[] args) {  
2     Collection<String> colors = List.of("Белый", "Черный",  
    "Красный", "Зеленый", "Синий");  
3     for (String str: colors) {  
4         System.out.println(str);  
5     }  
6 }
```



List



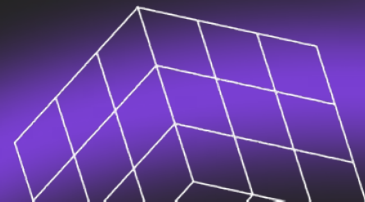


Интерфейс List используется для хранения упорядоченной последовательности объектов — он содержит основанные на индексах методы для вставки, обновления, удаления и поиска элементов. Он также может иметь повторяющиеся элементы и null в качестве значения



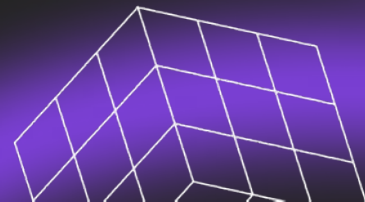


Для List есть метод `listIterator()` который возвращает объект `ListIterator` предназначенный для обхода коллекций реализующий интерфейс `List`





ArrayList является обычным массивом, обернутым
в интерфейс List (динамический массив)





Добавление нового элемента



Создается новый массив размер которого больше текущего в 1.5 раза



Элементы старого массива копируются в новый с сохранением позиции



Добавляется новый элемент к массиву в конец.



Создание коллекции

```
1 public static void main(String[] args) {  
2     //Пустой конструктор с начальной емкостью внутреннего массива  
   = 10  
3     ArrayList<String> list = new ArrayList<>();  
4  
5     //Конструктор принимает другую коллекцию, создавая новый  
   массив с элементами переданной коллекции  
6     ArrayList<String> listFromCollection = new ArrayList<>(list);  
7  
8     //В качестве параметра конструктора выступает значения  
   начального размера внутреннего массива.  
9     ArrayList<String> listWithCapacity = new ArrayList<>(1000);  
10 }
```



O big и ArrayList



Доступ к элементу — $O(1)$



Добавление элемента — $O(1)$



Вставка элемента в середину по индексу — $O(n)$



Поиск индекса элемента — $O(n)$



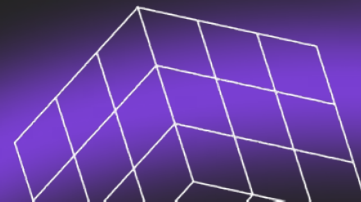
Проверка наличия элемента в коллекции — $O(n)$



Удаление элемента — $O(n)$



Обобщенный класс `LinkedList<E>` представляет
структуру данных в виде связанного списка —
т.е. каждый его элемент хранит ссылку
на предыдущий и последующий





Создание коллекции

```
1 public static void main(String[] args) {  
2     //Пустой конструктор  
3     LinkedList<String> linkedList = new LinkedList<>();  
4  
5     //создает список, в который добавляет все элементы другой  
    коллекции Collection  
6     LinkedList<String> linkedListFromCollection = new LinkedList<>  
    (linkedList);  
7  
8 }
```



O big и LinkedList



Добавление элемента — $O(1)$



Вставка элемента в середину по индексу — $O(n)$



Поиск — $O(n)$



Удаление элемента — $O(n)$



Проверка наличия элемента в коллекции — $O(n)$



Stack и Vector



Stack — реализует принцип LIFO (последний вошел — первый вышел)



Vector — методы синхронизированы



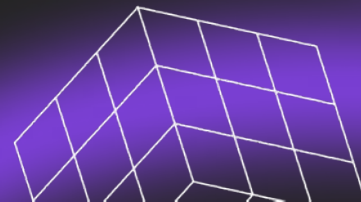


Map





Интерфейс Map<K, V> представляет отображение (словарь),
где каждый элемент представляет пару "ключ-значение".
Map гарантирует, что все ключи уникальны





HashMap

— это структура из пар «ключ-значение» — или динамический массив ключей. Каждый элемент массива — это bucket (корзина), в каждом из которых хранятся связанные списки со значением Map.Entry (если корзина пустая то связанный список будет состоять из одного элемента, а ссылаться этот элемент будет на null)



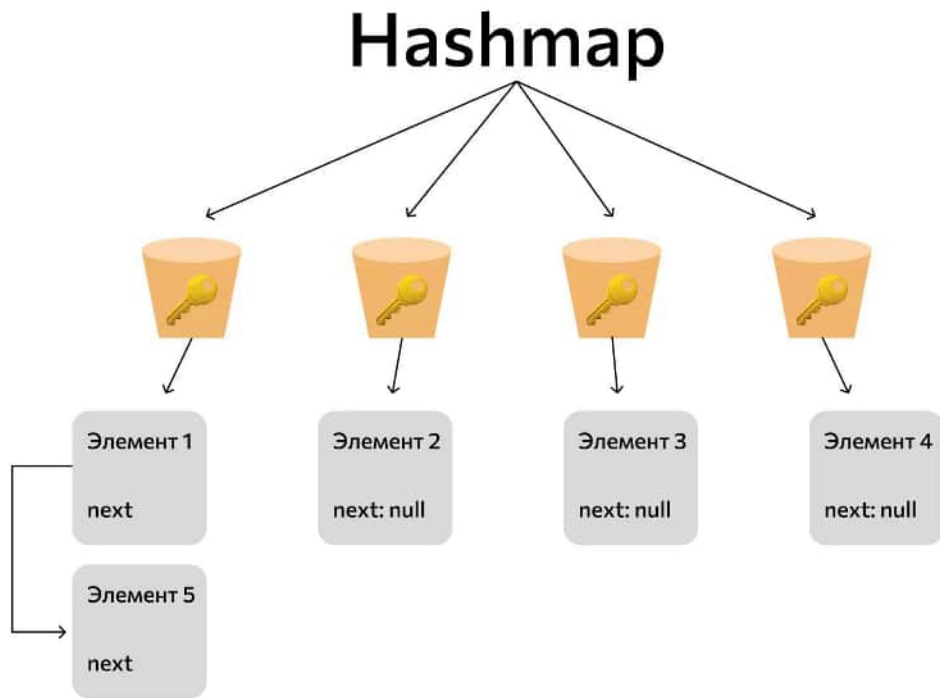


Создание коллекции

```
1 public static void main(String[] args) {  
2  
3     HashMap<String, Integer> hashMap = new HashMap<>();  
4     hashMap.put("One", 1);  
5     hashMap.put("Two", 2);  
6     hashMap.put("Three", 3);  
7  
8     for(Map.Entry<String, Integer> entry: hashMap.entrySet()) {  
9         System.out.println(entry.getKey()+" = " +  
10        entry.getValue());  
11    }  
12 }
```




HashMap





Добавление нового значения в HashMap

- 💡 Происходит вычисление `K.hashCode()`
- 💡 Происходит вычисление корзины куда стоит положить данную пару
- 💡 Далее мы обращаемся к связанному списку который храниться в корзине и сравниваем значение `hashCode`;
- 💡 Если `hashCode` не совпали мы добавляем новую пару в конец;
- 💡 Если `hashCode` совпали, то мы сравниваем `K` по `equals`;
- 💡 Если объекты `K` равны то просто отбрасываем новое значение;
- 💡 Если они отличаются то добавляем в конец списка.

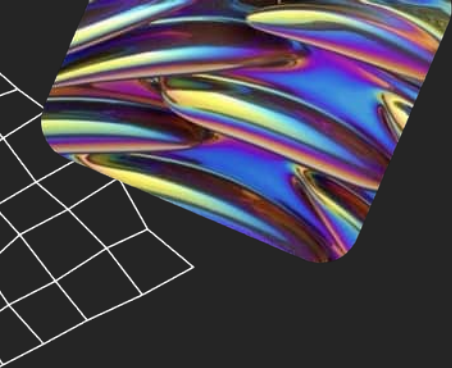


Поиск, удаление и добавления элемента
в среднем при работе с HashMap — $O(1)$





В HashMap для ключей всегда
используйте immutable объекты





В отличие от коллекции HashMap в TreeMap все объекты автоматически сортируются по возрастанию их ключей (натуральный порядок или comparator)





Set





Коллекция реализующая интерфейс Set гарантирует
уникальность элементов. Всегда переопределяйте
методы `hashCode()` и `equals()`





HashSet это частный случай **HashMap** — вместо значения `value` используется константа





Создание коллекции

```
1 public static void main(String[] args) {  
2     //Начальная емкость по умолчанию – 16, коэффициент загрузки –  
   0,75  
3     HashSet defaultConstructor = new HashSet();  
4  
5     //Конструктор с заданной начальной емкостью. Коэффициент  
   загрузки – 0,75  
6     HashSet constructorWithCapacity = new HashSet(32);  
7  
8     //Конструктор с заданными начальной емкостью и коэффициентом  
   загрузки  
9     HashSet constructorWithCapacityFactor = new HashSet(32, 0.6f);  
10  
11     //Конструктор, добавляющий элементы из другой коллекции  
12     HashSet fromCollection = new HashSet(defaultConstructor);  
13 }
```



TreeSet

```
1 public static void main(String[] args) {
2     //Конструктор по умолчанию
3     TreeSet defaultConstructor = new TreeSet();
4
5     //Конструктор, добавляющий элементы из другой коллекции
6     TreeSet fromCollection = new TreeSet(defaultConstructor);
7
8     //Конструктор создает пустое дерево, где все добавляемые
    элементы впоследствии будут отсортированы компаратором
9     TreeSet withComparator = new TreeSet(new Comparator() {
10         @Override
11         public int compare(Object o1, Object o2) {
12             return 0;
13         }
14     });
15 }
```



Queue





Очереди представляют структуру данных,
работающую по принципу **FIFO (first in — first out)**
— первый вошел, первый вышел.





Интерфейс Deque расширяет интерфейс Queue и определяет поведение двунаправленной очереди, которая работает как обычная однонаправленная очередь, либо как стек, действующий по принципу LIFO (последний вошел — первый вышел)





Первый элемент PriorityQueue или элемент из начала очереди

— это наименьший элемент, определенный с использованием естественного порядка или компаратора. Если имеется несколько элементов с одинаковым весом (Comparator при сравнении возвращает 0), очередь может произвольно возвращать любого из них





PriorityQueue

```
1 public static void main(String[] args) {
2     PriorityQueue<Integer> integerQueue = new PriorityQueue<>();
3     PriorityQueue<Integer> integerQueueWithComparator =
4         new PriorityQueue<>((Integer c1, Integer c2) → c2-
5         c1); //обратный порядок
6
7     integerQueueWithComparator.add(3);
8     integerQueue.add(3);
9
10    integerQueueWithComparator.add(2);
11    integerQueue.add(2);
12
13    integerQueueWithComparator.add(1);
14    integerQueue.add(1);
15
16    System.out.println(integerQueue.poll());
17    System.out.println(integerQueue.poll());
18    System.out.println(integerQueue.poll());
19    System.out.println(integerQueueWithComparator.poll());
20    System.out.println(integerQueueWithComparator.poll());
21    System.out.println(integerQueueWithComparator.poll());
22 }
```

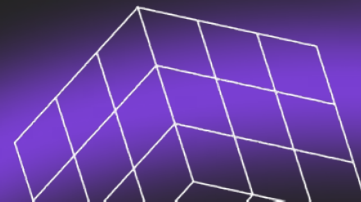


Collections





Класс `java.util.Collections` это утилитарный класс (состоит исключительно из статических методов), предоставляющий дополнительные методы для работы с `Collection` и `Map`





Подведем итоги



- List — для организации хранения “обычных” списков;
- Set — для хранения уникальных объектов;
- Queue — для реализации задачи FIFO и LIFO;
- Map — для реализации ассоциативного массива.



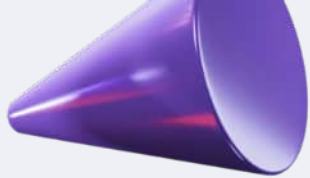


Чек-лист подготовки к семинару

Перед семинаром рекомендуем вам:

-  Изучить API интерфейсы List, Set, Queue, Map
-  Реализовать пример кода для создания коллекции (ArrayList, HashSet, HashMap), добавление элементов, удаление элементов, получения элемента.





Спасибо за внимание

