Задачи на тему «Коллекции»

# Простые задачи

Реализовать консольное приложение, которое будет предоставлять UI для выполнения операций над коллекциями.

Стартовый консольный UI:

**Выберите тип коллекции:**

1. **List**
2. **Set**
3. **Deque**
4. **Map**

При выборе конкретного пункта меню проваливаться в следующее меню, содержание которого будет зависеть от выбранного пункта.

**List:**

**Выберите действие над коллекцией**

1. **showItems** – вывести список элементов коллекции
2. **isEmpty** – пуст ли список
3. **contains** – содержит ли коллекция элемент (ожидается считывание элемента из консоли)
4. **add** – добавление элемента в конец списка (ожидается считывание элемента из консоли)
5. **addByIndex** – добавление элемента на определенную позицию (ожидается считывание двух параметров из консоли: элемент и индекс)
6. **remove** – удаление элемента (ожидается считывание элемента из консоли)
7. **removeByIndex** – удаление элемента, находящегося на определенной позиции (ожидается считывание позиции элемента из консоли)
8. **goToMainMenu** – возврат в меню выбора коллекции

После ввода соответствующей команды в случае необходимости считывания доп. параметров необходимо сообщить об этом пользователю, после чего считать веденные им данные, вывести результат операции, после чего снова вывести текущее меню работы над коллекцией.

**Deque:**

**Выберите действие над коллекцией**

1. **showItems** – вывести список элементов коллекции
2. **isEmpty** – пуст ли список
3. **contains** – содержит ли коллекция элемент
4. **addFirst** – добавить элемент в начало коллекции
5. **addLast** – добавить элемент в конец коллекции
6. **removeFirst** – удалить элемент из начала коллекции (возвращается удаленный элемент, в случае отсутствия элементов в коллекции бросается исключение NoSuchElementException)
7. **removeLast** – удалить элемент из конца коллекции (возвращается удаленный элемент, в случае отсутствия элементов в коллекции бросается исключение NoSuchElementException)
8. **getFirst** – получить первый элемент коллекции, при этом не удаляя его из самой коллекции (в случае отсутствия элементов в коллекции бросается NoSuchElementException)
9. **getLast** – получить последний элемент коллекции, при этом не удаляя его из самой коллекции (в случае отсутствия элементов в коллекции бросается NoSuchElementException)
10. **pollFirst** – получить первый элемент коллекции, удалив его при этом из самой коллекции (в случае отсутствия элементов в коллекции возвращается null)
11. **pollLast** – получить последний элемент коллекции, удалив его при этом из самой коллекции (в случае отсутствия элементов в коллекции возвращается null)
12. **goToMainMenu** – возврат в меню выбора коллекции

**Set:**

**Выберите действие над коллекцией**

1. **showItems** – вывести список элементов коллекции
2. **isEmpty** – пуст ли список
3. **contains** – содержит ли коллекция элемент (ожидается считывание элемента из консоли)
4. **add** – добавление элемента в конец списка (ожидается считывание элемента из консоли)
5. **remove** – удаление элемента (ожидается считывание элемента из консоли)
6. **goToMainMenu** – возврат в меню выбора коллекции

**Map:**

**Выберите действие над коллекцией**

1. **showItems** – вывести список элементов коллекции
2. **isEmpty** – пуст ли список
3. **get** – получение элемента коллекции, либо null в случае отсутствия (ожидается считывание элемента из консоли)
4. **put** – получение элемента в коллекцию (ожидается считывание элемента из консоли)
5. **remove** – удаление элемента из коллекции (ожидается считывание элемента из консоли)
6. **goToMainMenu** – возврат в меню выбора коллекции

# Продвинутые задачи

1. Реализовать свой список:
   1. Написать свой интерфейс List, содержащий методы:
      * boolean isEmpty() – пуст ли список;
      * int size() – количество элементов в списке;
      * void add(Object item) – вставка элемента item в конец списка;
      * void add(int index, Object item) — вставка элемента item на позицию index, при этом список раздвигается (все элементы, начиная с позиции index, увеличивают свой индекс на 1, т.е. сдвигаются на один элемент вправо);
      * Object get(int index) — возвращает объект, находящийся на позиции index;
      * int indexOf(Object obj) — возвращает индекс первого появления элемента obj в списке;
      * int lastIndexOf(Object obj) — возвращает индекс последнего появления элемента obj в списке;
      * void set(int index, Object item) — заменяет элемент, находящийся на позиции index объектом item;
      * void remove(int index) — удаляет элемент, находящийся на позиции index;
      * void remove(Object item) — удаляет элемент iteam из коллекции;
      * List subList(int from, int to) — возвращает новый список, представляющий собой часть данного (начиная с позиции from до позиции to-1 включительно).
   2. Создать класс ArrayList, который будет реализовывать наш интерфейс List, инкапсулируя в себе обычный массив, на основе следующего принципа:

При удалении произвольного элемента из списка, все элементы находящиеся «правее» смещаются на одну ячейку влево, при этом реальный размер массива (его емкость, capacity) не изменяется. Если при добавлении элемента, оказывается, что массив полностью заполнен, будет создан новый массив размером в полтора раза больше предыдущего, в него будут помещены все элементы из старого массива + новый, добавляемый элемент.

*Подсказка*:

В качестве инкапсулируемого массива следует использовать обычный массив из объектов Object[], который при создании коллекции содержит, например, 10 null объектов, т.е. заранее резервирует место для хранения будущих объектов. Иными словами, размер массива != количество реальных объектов, поэтому вам придется завести отдельное поля для хранения количества элементов в массиве.

При достижении правой границы инкапсулируемого массива его необходимо заменить более емким, например, посредством увеличения его размера в полтора раза. Для увеличения размера инкапсулируемого в коллекции массива можно воспользоваться следующей формулой: (n \* 3) / 2 + 1

Стоит заметить, что данную формулу можно улучшить за счет бинарных операций.

* 1. Создать интерфейс Deque
     + void addFirst(Object item) – добавить элемент в начало коллекции
     + void addLast(Object item) – добавить элемент в конец коллекции
     + Object removeFirst() – удалить элемент из начала коллекции (возвращается удаленный элемент, в случае отсутствия элементов в коллекции бросается исключение NoSuchElementException)
     + Object removeLast() – удалить элемент из конца коллекции (возвращается удаленный элемент, в случае отсутствия элементов в коллекции бросается исключение NoSuchElementException)
     + Object getFirst() – получить первый элемент коллекции, при этом не удаляя его из самой коллекции (в случае отсутствия элементов в коллекции бросается NoSuchElementException)
     + Object getLast() – получить последний элемент коллекции, при этом не удаляя его из самой коллекции (в случае отсутствия элементов в коллекции бросается NoSuchElementException)
     + Object pollFirst() - получить первый элемент коллекции, удалив его при этом из самой коллекции (в случае отсутствия элементов в коллекции возвращается null)
     + Object pollLast() - получить последний элемент коллекции, удалив его при этом из самой коллекции (в случае отсутствия элементов в коллекции возвращается null)
  2. Создать класс LinkedList, реализующий наши интерфейсы List и Deque посредством классического двусвязного списка, основанного на объектах с ссылками между ними.

*Подсказка:* Пример класса, который можно использовать в качестве узла в двусвязном списке:

class Node {

Object item;

Node next;

Node prev;

}

* 1. Реализовать для наших классов интерфейс Iterable