

Коллекции

Когда массивы не годятся

Содержание

1. Минусы массивов
2. List

1. Минусы массивов

В предыдущей лекции мы рассматривали проблему с неким объемом задач которые должны исполнять разные работники. Если вы не читали предыдущую лекцию то настоятельно рекомендую, иначе вы не поймете о чем речь в этой лекции. Мы хранили все задачи в одном массиве и просто перезаписывали их новыми статусами. Минус решения был в том, что каждый раз задача из общего массива попадала в руки дизайнера, который после уже смотрел на статус и отдавал программисту, а тот в свою очередь тестировщику. И мы решили поменять подход. Сделать как в реальности – 3 столбца для задач, на изменения которых подпишутся наши работники.

Но это не самая тривиальная проблема которую нужно решить. Давайте двигаться постепенно. В этой лекции мы попробуем обойтись массивом и поймем в чем его минусы.

Итак, у нас есть фабрика задач. Предположим в ней 100 задач и у нас есть массив на 100 элементов. Мы хотим создать еще 2 хранилища. В начале пути мы уже забираем у памяти 300 ячеек. И 200 из них инициализируем null. После этого нам нужно при завершении задачи менять хранилище задачи. Значит из одного массива мы забираем задачу и кладем в другой массив. И здесь самый большой минус массива. Смотрите. В начале во втором массиве задач для статуса IN_PROGRESS все ячейки массива заполнены нулями. Далее мы добавили в первую ячейку элемент и далее нам нужно добавить во вторую. Но как нам узнать об этом? Нужно пройти по массиву и найти первый элемент который ноль и его перезаписать. Далее когда мы убираем из этого массива элемент и кладем в третий массив порядок изменится. Да, можно хранить какой-нибудь числовой индикатор заполненности и менять его каждый раз но это еще одна переменная которая ненапрямую связана с массивом. Да и ситуации когда во втором или третьем массиве будет по 100 элементов не наступит. Ведь наши работники не будут ждать пока в столбец с их задачами навалится 100 штук.

И еще одна проблема в том, что когда первый массив опустошится, то в нем будет 100 ссылок на ноль. Но он все равно по прежнему будет занимать место 100 элементов. Т.е. сейчас мы можем описать все минусы массивов

- нужно знать сразу размер массива и его менять нельзя, если вам нужен массив другого размера то придется создавать новый и туда копировать данные
- даже если вам не нужны все ячейки массива их все равно придется держать в памяти

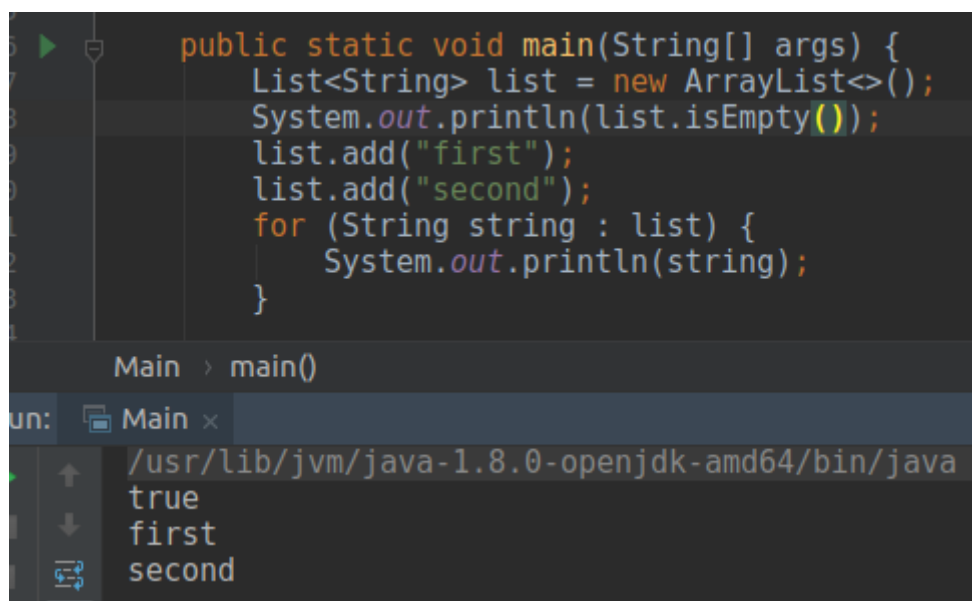
- сложно находить пустое место и туда записывать что-то
- так же сложно находить последний непустой элемент чтобы его убирать

Теперь встает закономерный вопрос, хорошо, если нам под эту конкретную задачу не подходит массив, то во-первых где он вообще используется (ответ - там где нужны его простота и неизменяемость, расскажу когда пройдем стримы) и во-вторых, что тогда использовать.

На второй вопрос мы сможем ответить прямо сейчас. Коллекции

2. List

В языке джава есть такой пакет `java.util` и в нем много всякого полезного. Одно из них это коллекции. Считайте что это те же массивы, но на стероидах. Давайте детальнее посмотрим на это.



```
public static void main(String[] args) {
    List<String> list = new ArrayList<>();
    System.out.println(list.isEmpty());
    list.add("first");
    list.add("second");
    for (String string : list) {
        System.out.println(string);
    }
}
```

Main > main()

un: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java

true

first

second

Помните дженерики? Параметризованный интерфейс `List<T>` в который мы передаем тип аргумента `String` и вуаля – у нас список из строк. Мы инициализируем его через реализацию `ArrayList` и как видите нам не нужно передавать объем списка. Это можно сделать просто передав число в конструктор, но это лишь будет оптимизировать наш код. Ведь есть некое количество памяти по умолчанию которое список забирает при инициализации. И если вы проинициализируете его числом 3 например, то вы поможете и себе тоже, ведь будете знать начальный объем списка (длина списка).

Хорошо, мы создали список строк и теперь он пустой. Т.е. он реально пустой. Если хотите это проверить можно вызвать метод для этого как со строкой.

Ладно, раз он пустой, давайте добавим туда элементы. Добавляем 2 строки и теперь хотим проверить что это произошло. Выводим их в цикле в консоль. Все хорошо.

И мы сказали, что еще одним минусом массива является то, что после удаления элемента в нем будет храниться нул. Давайте посмотрим на наш список и удалим объект который лежит первым. Для этого в списках есть метод удаления самого элемента.

```
public static void main(String[] args) {
    List<String> list = new ArrayList<>();
    System.out.println(list.isEmpty());
    list.add("first");
    list.add("second");
    for (String string : list) {
        System.out.println(string);
    }

    list.remove(0: "first");
    for (int i = 0; i < list.size(); i++) {
        System.out.println("i " + i + " item " + list.get(i));
    }
}
```

Main > main()

in: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...

true

first

second

i 0 item second

Как видите мы удалили строку которая совпадает с первой и мы можем так же как и массивом выводить в консоль не только сами значения, но и индексы элементов.

Теперь же давайте добавим новую строку не в конец списка, а например в начало. Для этого есть перегруженный метод `add(int index, T item)` где мы по индексу кладем новый элемент. И так же есть метод который удалит нам элемент по позиции. Предположим мы не знаем длину списка и нам нужно удалить последний элемент. Для этого мы находим длину списка и отнимаем единицу. И все прекрасно работает.

Кстати, вы заметили что для получения размера массива мы вызывали его публик свойство `length` а в списке мы вызываем функцию. Это опять же потому что в массиве длина постоянна, а в списке нет. Ладно, а какие еще методы доступны списку?

Например метод `contains`, с помощью которого можно сразу узнать есть ли в списке объект. И не нужно проходить циклом и искать его самому.

```
public static void main(String[] args) {
    List<String> list = new ArrayList<>();
    list.add("first");
    list.add("second");
    list.add("third");
    System.out.println(list.contains("second"));
}
```

```
public static void main(String[] args) {
    List<String> list = new ArrayList<>();
    System.out.println(list.isEmpty());
    list.add("first");
    list.add("second");
    for (String string : list) {
        System.out.println(string);
    }

    list.remove(0: "first");
    for (int i = 0; i < list.size(); i++) {
        System.out.println("i " + i + " item " + list.get(i));
    }

    list.add(i: 0, e: "third");
    list.add("fourth");
    System.out.println("added 2 items");
    list.remove(i: list.size() - 1);
    for (int i = 0; i < list.size(); i++) {
        System.out.println("i " + i + " item " + list.get(i));
    }
}
```

Main

n: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...

true

first

second

i 0 item second

added 2 items

i 0 item third

i 1 item second

Process finished with exit code 0

Так же вы можете в любой момент времени преобразовать ваш список к массиву, если вам нужно.

```
String[] array = new String[list.size()];
list.toArray(array);
```

Еще одна полезная функция – добавить в список все элементы другого списка. Этот метод полезен если вам нужны элементы другого списка в конце вашего текущего списка. Если же в начале, то можно использовать конструктор.

Так же вы можете найти индекс элемента, давайте на это посмотрим внимательно. Ведь у нас в списке может быть несколько одинаковых элементов, так?

```
public static void main(String[] args) {
    List<String> list = new ArrayList<>();
    list.add("first");
    list.add("second");
    list.add("third");
    List<String> big = new ArrayList<>();
    big.add("big");
    big.addAll(list);
    for (String string : big) {
        System.out.print(string + " ");
    }
}
```

Main > main()

un: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...

big first second third

Process finished with exit code 0

```
public static void main(String[] args) {
    List<String> list = new ArrayList<>();
    list.add("first");
    list.add("second");
    list.add("third");
    List<String> big = new ArrayList<>(list);
    big.add("big");
    for (String string : big) {
        System.out.print(string + " ");
    }
}
```

Main > main()

un: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...

first second third big

Process finished with exit code 0

Первый метод `indexOf` находит индекс первого попавшегося элемента, а второй метод `lastIndexOf` находит индекс последнего элемента который равен переданному аргументу. И перед тем как мы пойдем дальше, давайте еще рассмотрим метод очищения списка

```
public static void main(String[] args) {
    List<String> list = new ArrayList<>();
    list.add("first");
    list.add("second");
    list.add("third");
    list.add("first");
    list.add("so");
    System.out.println(list.indexOf("first"));
    System.out.println(list.lastIndexOf("first"));
}
```

Main > main()

Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...

0

3

Process finished with exit code 0

Это наверное самое часто используемое в списках. Зачастую мы отображаем на экране список одних элементов и потом нам нужно заменить его другими элементами. Для этого хорошо подходит метод очищения. Он опустошает ваш список и помогает вам не инициализировать повторно. Т.е. ваш список может быть объявлен final. И кстати говоря о null. Список может хранить null если вы положите его в него.

```
public static void main(String[] args) {
    List<String> list = new ArrayList<>();
    list.add("first");
    list.add("second");
    list.add("third");
    list.add("first");
    list.add("so");
    list.clear();
    list.add("new item");
    for (String string : list) {
        System.out.println(string);
    }
}
```

Main > main()

un: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java .

new item

Process finished with exit code 0

```
public static void main(String[] args) {
    List<String> list = new ArrayList<>();
    list.add("first");
    list.add(null);
    for (String string : list) {
        System.out.println(string);
    }
}
```

Main

n: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java .

first

null

Process finished with exit code 0

И еще один плюс списка – можно заменить элемент другим по индексу

```
public static void main(String[] args) {
    List<String> list = new ArrayList<>();
    list.add("first");
    list.add(null);
    list.set(0, "second");
    for (String string : list) {
        System.out.println(string);
    }
}
```

Main > main()

n: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java

second

null

Process finished with exit code 0

И конечно же, там где есть `set` там и должен быть `get(int index)` – получение элемента по индексу. Но здесь будьте осторожны – вы можете указать индекс выходящий за длину списка. Так что проверьте длину списка перед тем как вызвать этот метод.

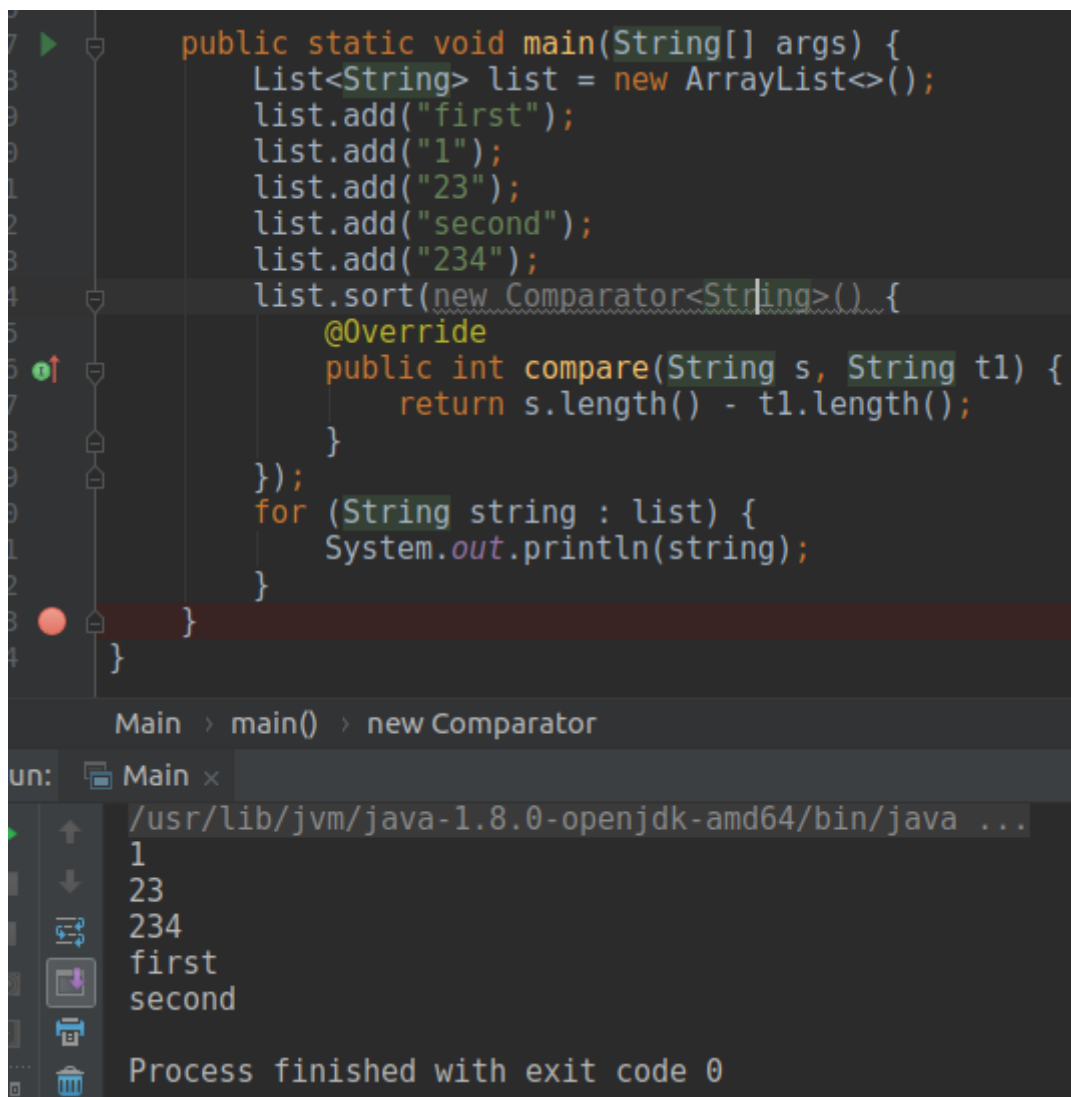
```
if (list.size() - 1 >= 4) {
    System.out.println(list.get(4));
}
```

Иначе вы получите `IndexOutOfBoundsException`

Итак, у списка есть определенные плюсы

- Можно создавать пустой список и заполнять его после
- Добавлять новые элементы в начало или в середину или в конец списка
- Добавлять сразу несколько элементов из другого списка
- Преобразовать к массиву
- Находить элемент и индекс
- Проверять наличие элемента в списке
- Удалять элемент по содержанию или по индексу
- Очищать список и заполнить его заново
- Заменять элемент по индексу как в случае с массивом

И это не все, список можно сортировать!



```
7 public static void main(String[] args) {
8     List<String> list = new ArrayList<>();
9     list.add("first");
10    list.add("1");
11    list.add("23");
12    list.add("second");
13    list.add("234");
14    list.sort(new Comparator<String>() {
15        @Override
16        public int compare(String s, String t1) {
17            return s.length() - t1.length();
18        }
19    });
20    for (String string : list) {
21        System.out.println(string);
22    }
23 }
24 }
```

Main > main() > new Comparator

un: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...

1

23

234

first

second

Process finished with exit code 0

Метод сортировки принимает интерфейс компаратора, который сравнивает 2 строки. В нашем случае мы сортируем по длине строки, вы можете написать свою логику. Кстати, вы знаете что делать с серым кодом – Alt+Enter.

Ладно, мы нашли минусы списка

- Может содержать дубликаты
- Может содержать null
- Прочие мелкие трудности связанные с индексом