**Class ArrayList**

[Java Syntax](https://javarush.ru/quests/QUEST_JAVA_SYNTAX)

[Уровень 7](https://javarush.ru/quests/lectures?quest=QUEST_JAVA_SYNTAX&level=7), Лекция 5

— А вот и я.

— Привет, Элли!

— Сегодня новая интересная тема! Сейчас я расскажу тебе о новом интересном классе — **ArrayList**.

— О, новый класс? Круто! И что он умеет делать?

— Начну с небольшой предыстории. Программистам очень не нравилось одно свойство массива – его размер нельзя изменять. Что делать, если нужно сохранить в массиве ещё три элемента, а свободное место только одно?

— Единственным решением проблемы нехватки места в массиве было создание массива очень большого размера, чтобы все элементы туда точно поместились. Но это часто приводило к нерациональному расходу памяти. Если чаще всего в массиве хранилось три элемента, но был хотя бы мизерный шанс, что там их будет 100, приходилось создавать массив на 100 элементов.

— И что же придумали программисты?

— Они написали класс **ArrayList** (списочный массив), который выполнял ту же работу, что и Array (массив), но мог изменять свой размер.

— Интересный ход. И как же они это сделали?

— Внутри каждого объекта типа **ArrayList** хранится обычный массив элементов. Когда ты считываешь элементы из ArrayList, он считывает их из своего внутреннего массива. Когда записываешь – записывает их во внутренний массив. Вот сравни:

|  |  |
| --- | --- |
| **Array** | **ArrayList** |
| Создание контейнера элементов | |
| String[] list = new String[10]; | ArrayList<String> list = new ArrayList<String>(); |
| Получение количества элементов | |
| int n = list**.length;** | int n = list**.size**(); |
| Взятие элемента из массива/коллекции | |
| String s = list[3]; | String s = list**.get**(3); |
| Запись элемента в массив | |
| list[3] = s; | list**.set**(3, s); |

— И в чем же преимущество ArrayList? Как по мне, так код стал длиннее.

— Во-первых, ***ArrayList*** поддерживает несколько дополнительных действий, которые очень часто приходится делать программистам во время работы, и которых нет у массива. Например – вставка и удаление элементов из середины массива, и чтобы не оставалось дырок.

— Во-вторых, изменение размера: когда нужно записать во внутренний массив ещё один элемент, а свободного места там нет, то внутри ***ArrayList*** делается вот что:

а) создаётся ещё один массив, в полтора раза больше размера внутреннего массива, плюс один элемент.

б) все элементы из старого массива копируются в новый массив.

в) новый массив сохраняется во внутренней переменной объекта ArrayList, старый массив объявляется мусором (мы просто перестаём хранить на него ссылку).

|  |  |
| --- | --- |
| **Array** | **ArrayList** |
| Добавление элемента в конец массива | |
| Невозможно выполнить данное действие | list**.add**(s); |
| Вставка элемента в середину массива | |
| Невозможно выполнить данное действие | list**.add**(15, s); |
| Вставка элемента в начало массива | |
| Невозможно выполнить данное действие | list**.add**(0, s); |
| Удаление элемента из массива | |
| Можно стереть элемент с помощью list[3] = null. Но тогда останется «дыра» в массиве. | list**.remove**(3); |

2

Задача

Java Syntax,  7 уровень,  5 лекция

Набираем код Ӏ Java Syntax: 7 уровень, 5 лекция

Java Syntax: 7 уровень, 5 лекция. Иногда думать не надо, строчить надо! Как ни парадоксально звучит, порой пальцы «запоминают» лучше, чем сознание. Вот почему во время обучения в секретном центре JavaRush вы иногда встречаете задания на набор кода. Набирая код, вы привыкаете к синтаксису и зарабатываете немного материи. А ещё — боретесь с ленью.

— А как работать с этим ArrayList’ом?

— А практически так же, как и с массивом. Вот смотри. Давай сравним работу с ArrayList с работой с массивом. Для примера решим такую задачу «ввести 10 строк с клавиатуры и вывести их на экран в обратном порядке» .

— Смотри:

Используем Array

public static void main(String[] args)

{

Reader r = new InputStreamReader(System.in);

BufferedReader reader = new BufferedReader(r);

// ввод строк с клавиатуры

String[] list = new String[10];

for (int i = 0; i < list.length; i++)

{

String s = reader**.readLine**();

list[i] = s;

}

// вывод содержимого массива

// на экран в обратном порядке

for (int i = 0; i < list.length; i++)

{

int j = list.length - i - 1;

System.out.println( list[j] );

}

}

Используем ArrayList

public static void main(String[] args)

{

Reader r = new InputStreamReader(System.in);

BufferedReader reader = new BufferedReader(r);

// ввод строк с клавиатуры

ArrayList<String> list = new ArrayList<String>();

for (int i = 0; i < 10; i++)

{

String s = reader**.readLine**();

list.add(s);

}

// вывод содержимого коллекции

// на экран в обратном порядке

for (int i = 0; i < list.size(); i++)

{

int j = list.size() - i - 1;

System.out.println( list.get(j) );

}

}

— Я раскрасила одним цветом эквивалентные действия в обеих колонках.

— С одной стороны все по-другому, с другой – все осталось тем же самым.

— Ага. Только мы теперь не используем квадратные скобочки при работе с **Class ArrayList**. Вместо них мы используем методы **get**, **set** и **add**.

— Да, я уже заметил. Но все равно очень похоже.