List<E> – Overview

- `List<E>` holds a list of elements of any type

```
List<String> names = new ArrayList<>();
//Create a list of strings
names.add("Peter");
names.add("Maria");
names.add("George");
names.remove("Maria");
for (String name : names)
  System.out.println(name);
//Peter, George
```

List<E> – Overview (2)

```java
List<Integer> nums = new ArrayList<>(
           Arrays.asList(10, 20, 30, 40, 50, 60));
nums.remove(2);
nums.remove(Integer.valueOf(40));
nums.add(100);
nums.add(0, -100);
for (int i = 0; i < nums.size(); i++)
  System.out.print(nums.get(i) + " ");
```

Remove by index

Remove by value

Inserts an element to index

Items count

```
-100 10 20 50 60 100
```

List<E> – Data Structure

- `List<E>` holds a list of elements (like array, but extendable)
- Provides operations to **add** / **insert** / **remove** / **find** elements:
  - `size()` – number of elements in the List<E>
  - `add(element)` – adds an element to the List<E>
  - `add(index, element)` – inserts an element to given position
  - `remove(element)` – removes an element (returns true / false)
  - `remove(index)` – removes element at index
  - `contains(element)` – determines whether an element is in the list
  - `set(index, item)` – replaces the element at the given index

Reading Lists from the Console

- First, read from the console the array **length**:

```
Scanner sc = new Scanner(System.in);
int n = Integer.parseInt(sc.nextLine());
```

- Next, create a list of given size **n** and read its **elements**:

```
List<Integer> list = new ArrayList<>();
for (int i = 0; i < n; i++) {
  int number = Integer.parseInt(sc.nextLine());
  list.add(number);
}
```

Reading List Values from a Single Line

- Lists can be read from a **single line** of **space separated values**:

```
2 8 30 25 40 72 -2 44 56
```

```
String values = sc.nextLine();
List<String> items = Arrays.stream(values.split(" "))
          .collect(Collectors.toList());
List<Integer> nums = new ArrayList<>();
for (int i = 0; i < items.size(); i++)
  nums.add(Integer.parseInt(items.get(i)));
```

Convert a collection into `List`

```
List<Integer> items = Arrays.stream(values.split(" "))
.map(Integer::parseInt).collect(Collectors.toList());
```

Printing Lists On the Console

- Printing a list using a `for`-loop:

```
List<String> list = new ArrayList<>(Arrays.asList(
          "one", "two", "three", "four", "five", "six"));
for (int index = 0; index < list.size(); index++)
  System.out.printf
               ("arr[%d] = %s%n", index, list.get(index));
```

Gets an element at given index

- Printing a list using a `String.join()`:

```
List<String> list = new ArrayList<>(Arrays.asList(
  "one", "two", "three", "four", "five", "six"));
System.out.println(String.join("; ", list));
```

Sorting Lists

- Sorting a list == reorder its elements incrementally: **Sort()**
  - List items should be **comparable**, e.g. numbers, strings, dates, …

```
List<String> names = new ArrayList<>(Arrays.asList(
"Peter", "Michael", "George", "Victor", "John"));
Collections.sort(names);
System.out.println(String.join(", ", names));
// George, John, Michael, Peter, Victor
Collections.sort(names);
Collections.reverse(names);
System.out.println(String.join(", ", names));
// Victor, Peter, Michael, John, George
```

Sort in natural (ascending) order

Reverse the sorted result

Summary

- Lists hold a sequence of elements (variable-length)
- Can **add** / **remove** / **insert** elements at runtime
- Creating (allocating) a list:
  `new ArrayList<E>()`
- Accessing list elements by index
- Printing list elements: `String.join(…)`