Collections

List Interface

What is a List?

- an ordered collection which can contain duplicate entries
- items can be reached and inserted using the index (`int`)
- unlike array, list can change in size after being declared
- there are two classes which implement `List` interface:
  - `ArrayList` and `LinkedList`
- `ArrayList` is better when you read more than you write
- `LinkedList` implements both `List` and `Deque`

* for OCA exam you only need to know `ArrayList`

Creating a List using factory methods

1. `Arrays.asList(varargs)`
   - fixed size list backed by an array
2. `List.of(varargs)`
   - returns immutable list
3. `List.copyOf(collection)`
   - immutable list with copy of original values

- when you create a List in this way, its sized is **fixed** (no adding or removing)

```
String[] names = new String[] {"John", "George", "Luke"}
```
(an array)

```
List<String> namesAsList = Arrays.asList(names);
List<String> namesOf = List.of(names);
List<String> namesCopyOf = List.copyOf(namesAsList);

names[1] = "Ben";
```
names = [John, Ben, Luke]

```
System.out.println(namesAsList);
  => [John, Ben, Luke]
```
because the list is "backed" by the array

```
System.out.println(namesOf);
  => [John, George, Luke]
```
no change in the list

```
System.out.println(namesCopyOf);
  => [John, George, Luke]
```
no change in the list

```
// backing up works both ways
String[] names = new String[] {"John", "George", "Luke"}
List<String> namesAsList = Arrays.asList(names);

namesAsList.set(2, "Paul");

System.out.println(namesAsList);
  => [John, George, Paul]
System.out.println(Arrays.toString(names));
  => [John, George, Paul]

// when you have a list backed by the array (Arrays.asList)
// then change in one results with change in another
```

```
// list created by a factory method has a fixed size!
String[] names = new String[] {"John", "George", "Luke"}
List<String> namesAsList = Arrays.asList(names);
List<String> namesOf = List.of(names);
List<String> namesCopyOf = List.copyOf(namesAsList);

namesOf.add("Mike");
  => DOES NOT COMPILE
namesAsList.remove("John");
  => DOES NOT COMPILE

// the list created by List.copyOf is immutable
namesCopyOf.set(0, "Paul");
  => DOES NOT COMPILE
```

```
// creating a List with a constructor


List<String> myList1 = new ArrayList<>();
  => creates new empty List myList1
List<String> myList2 = new ArrayList<>(myList1);
  => makes a copy of myList1 and stores it in myList2



ArrayList<String> arrayList1 = new ArrayList<>();
  => creates new empty ArrayList myList1
ArrayList<String> arrayList2 = new ArrayList<String>(arrayList1);
  => makes a copy of arrayList1 and stores it in arrayList2
ArrayList<String> arrayList3 = new ArrayList<String>(5);
  => you have reserved 5 slots, but you can always add more if you want
```

List Methods

```
add(E element)
add(int index, E element)
get(int index)
remove(int index)
remove(E element)
replaceAll(UnaryOperator<E> op)
set(int index, E element)
sort(Comparator<? super E> c)
```

```
List<String> names = new ArrayList<>();
names.add("John");
names.add("George");
names.add("Paul");
names.add("Ringo");
System.out.println(names);
names.add(1, "Luke");
System.out.println(names);
System.out.println(names.get(0));
names.set(3, "Ben");
System.out.println(names);
names.remove(1);
System.out.println(names);
names.remove("Ben");
System.out.println(names);
names.replaceAll(s -> s + " A.");
System.out.println(names);
```

```
[John, George, Paul, Ringo]
[John, Luke, George, Paul, Ringo]
John
[John, Luke, George, Ben, Ringo]
[John, George, Ben, Ringo]
[John, George, Ringo]
[John A., George A., Ringo A.]
```

```
// two ways of using remove() method
List<Integer> nums = new ArrayList<Integer>();
nums.add(2);
nums.add(-11);
nums.add(7);
  => nums = [2, -11, 7]
nums.remove(2);
  => nums = [2, -11]

// if we want to remove element 2:
nums.remove(Integer.valueOf(2));
  => now the argument is an Object, and remove(E element) will be used
  => nums = [-11]
```

What element will be removed, 2 or 7?

=> since 2 is primitive, `remove(int index)` will be used

=> number 7 will be removed

```
// converting List to Array using toArray() method
List<Integer> myList = new ArrayList<>();
myList.add(3);
myList.add(5);
myList.add(7);

Object[] objArray = myList.toArray();
  => array of Objects in the list

Integer[] intArray = myList.toArray(new Integer[0]);
  => array of Integers
  => initial size is 0, but Java will automatically adjust sizes to fit
```