Collections

Common Methods

What is a collection?

- these *interfaces* are commonly referred to as **collections**
  - `List`, `Set`, `Queue` (`Deque`), `Map`
- in the end we want to work with *classes*:
  - interfaces `List`, `Set` and `Queue` implement `Collection`
  - class `ArrayList` implements `List`
  - classes `HashSet` and `TreeSet` implement `Set`
  - interface `Deque` implements `Queue`
  - class `LinkedList` implements `Queue` and `List`
  - interface `Map` **doesn't** implement `Collection`
  - classes `HashMap` and `ThreeMap` implement `Map` interface

OCA (1Z0-808)

```
// diamond operator (<>) is used to imply the type of the element in collection
List<String> names = new ArrayList<String>();

// you can omit the type on the right hand side
List<String> names = new ArrayList<>();
```

OK

on the left side you can use the name of the class (`ArrayList`) or the name of the interface which the class implements (here: `List` or `Collection`)

```
// but not on the left-hand side!
List<> names = new ArrayList<String>();
```

does not compile

```
// if you use var* you have to specify the type on right-hand side:
var names = new ArrayList<String>();
```

OK

* only for Java 17 [OCP exam]

```
// add() method: adds an element in the Collection, returns true or false


Collection<String> names = new ArrayList<>();
System.out.println(names.add("John"));
  => true
System.out.println(names.add("John"));
  => true


Collection<String> names = new HashSet<>();
System.out.println(names.add("John"));
  => true
System.out.println(names.add("John"));
  => false
```

Set doesn't allow duplicates [OCP only]

```
// remove() method: removes an element in the Collection, returns true or false
Collection<String> names = new ArrayList<>();
names.add("John");
names.add("George");
names.add("John");
System.out.println(names);
  => [John, George, John]
System.out.println(names.remove("John"));
  => true
System.out.println(names);
  => [George, John]
System.out.println(names.remove("Luke"));
  => false
```

only the first match is removed

there is no element "Luke" in the collection

```
// isEmpty() method
Collection<String> names = new ArrayList<>();
System.out.println(names.isEmpty());
  => true

// size() method
Collection<String> names = new ArrayList<>();
names.add("John");
names.add("George");
names.add("John");
System.out.println(names.size());
  => 3
```

```
// clear() method


Collection<String> names = new ArrayList<>();

names.add("John");

names.add("George");

names.add("John");

names.clear();

System.out.println(names.size());

  => 0
```

```
// contains() method


Collection<String> names = new ArrayList<>();
names.add("John");
names.add("George");
names.add("John");
System.out.println(names.contains("George"));
  => true
System.out.println(names.contains("Luke"));
  => false
```

```
// removeIf() method


Collection<String> names = new ArrayList<>();
names.add("John");
names.add("George");
names.add("Luke");
names.removeIf(s -> s.length() > 4);
System.out.println(names);
  => [John, Luke]
```

takes `Predicate` as an argument, implemented by a lambda expression

```
// forEach() method
Collection<String> names = new ArrayList<>();
names.add("John");
names.add("George");
names.add("Luke");
names.forEach(name -> System.out.print(name + ", "));
  => John, George, Luke,


// equals() method
  // comparing the type and contents of the Collection
  // implementations vary (ArrayList checks order, HashSet does not, etc.)
```

takes Consumer as an argument, implemented by a lambda expression