Collections and Iterators

Josh Wilcox (jw14g24@soton.ac.uk)

February 5, 2025

Contents

1 ArrayLists

1.1 Problems with Arrays

- Arrays are stored in a contiguous chunk in memory
  - This is useful as it is easy for computers to fetch a specific element in such an array
  - Allows for a constant time access to different positions of the array
  - The problem is: addition and extension of such lists are very difficult

1.2 Solution with ArrayLists

- Changes the size of the array as you had elements
- Has an `add()` method and takes care of its size by itself
- Does allow for the use of indexes

1.3 Syntax

1.3.1 Array vs ArrayList

Array

```
1 Cat[] catArray;
2 catArray = new Cat[10]
3
4 catArray[0] = moggy1;
5 catArray[1] = moggy2;
6
7 callMethodOn(catArray[1]);
8
9 catArray[0] = null
```

ArrayList

```
1 ArrayList catAList;
2 catAList = new ArrayList();
3
4 catAList.add(moggy1);
5 catAList.add(moggy2);
6
7 callMethodOn(catAList.get(1));
8
9 catAList.remove(moggy1);
```

1.4 Advantages of ArrayLists

- Better for more complex tasks
- Have many useful methods
- Both Arrays and `ArrayLists` are objects, but `ArrayLists` feel more object

1.5 Primitives

- ArrayLists can *only* store **Deadlines**

2 Generics

- `ArrayLists` can store objects of any type
- We can mix types of objects in default `ArrayLists`
- However, we can use the **generics** mechanism to force the `ArrayList` to only take one type
  - For example: `ArrayList<Dog> kennel = new ArrayList<Dog>();`
  - Creates a new `ArrayList` that only will take `Dog` objects as elements

3 Iterators

- Iterators help with iteration over a collection
- They track progress through a collection with special functions

3.1 Using Iterators

- An `Iterator` is an object that enables you to traverse through a collection, one element at a time.
- The `Iterator` interface provides several methods to iterate over a collection.

3.2 Special Methods

- `hasNext()`: Returns `true` if the iteration has more elements.
- `next()`: Returns the next element in the iteration.
- `remove()`: Removes from the underlying collection the last element returned by this iterator (optional operation).

3.2.1 Example of Special Methods

```java
import java.util.ArrayList;
import java.util.Iterator;

public class IteratorSpecialMethods {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");

        Iterator<String> iterator = list.iterator();

        while (iterator.hasNext()) {
            String element = iterator.next();
            if (element.equals("Banana")) {
                iterator.remove();
            }
        }

        System.out.println(list); // Output: [Apple, Cherry]
    }
}
```

3.3 Advantages of Using Iterators

- Provides a uniform way to access elements of a collection.
- Hides the underlying implementation of the collection.
- Can be used to remove elements from the collection safely during iteration.