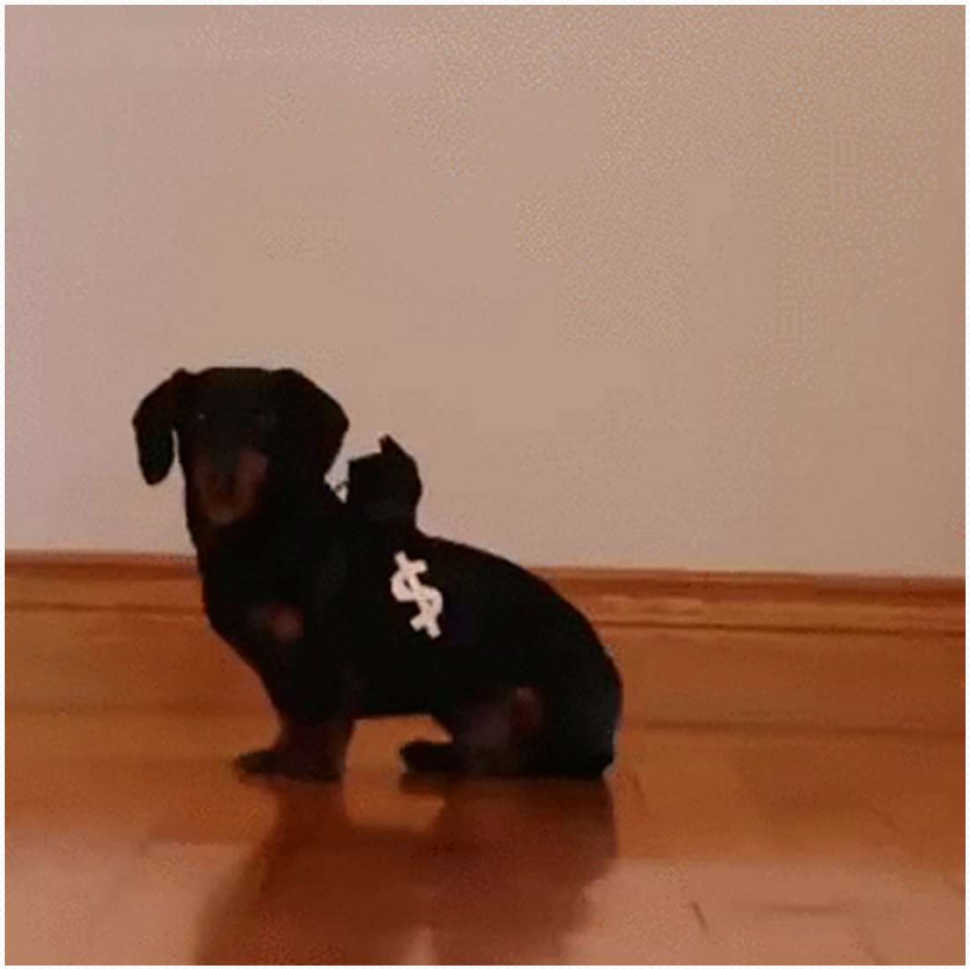
Building Java Programs

Chapter 10 & 11

Lists and Sets

reading: 10.1, 11.2



Week 2: 9/30-10/4

- Monday
  - Client of Collections: Lists and Sets
- Tuesday
  - Style
- Wednesday
  - Stacks and Queues
- Thursday
  - Stacks and Queues
- Friday
  - Classes, Objects, and References

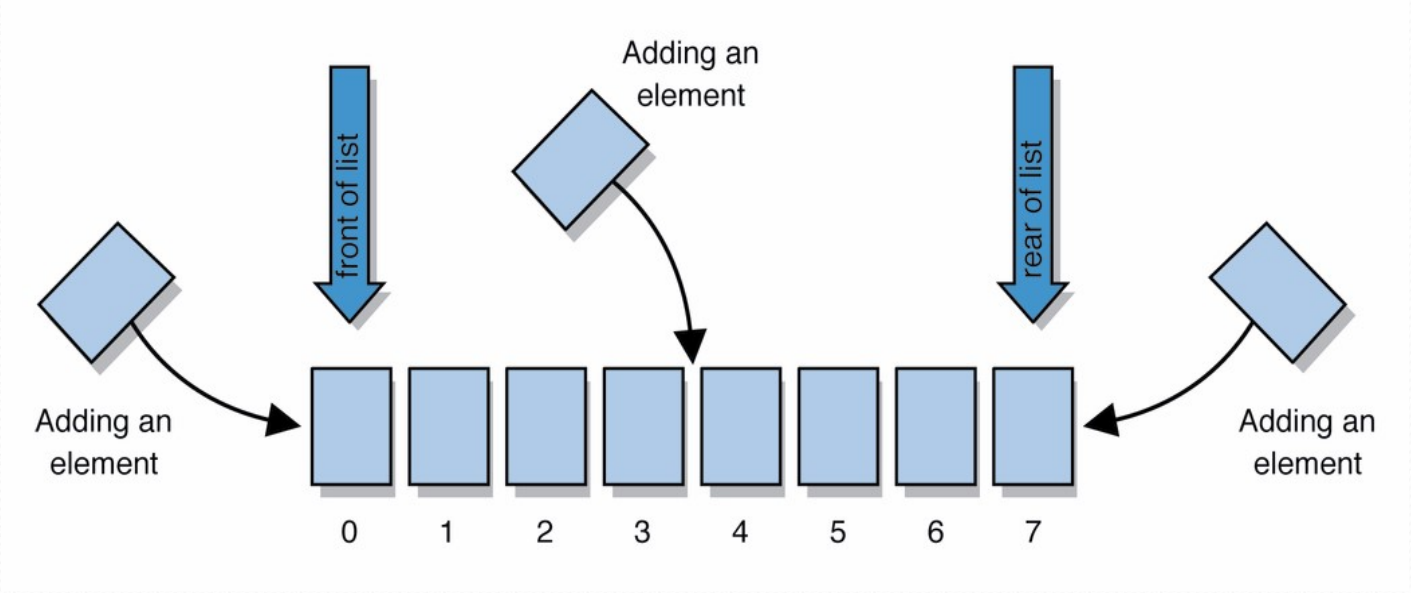
Collections

- **collection**: an object that stores data; a.k.a. "data structure"
  - the objects stored are called **elements**
  - some collections maintain an ordering; some allow duplicates
  - typical operations: *add*, *remove*, *clear*, *contains* (search), *size*
  - examples found in the Java class libraries: (covered in this course!)
    - `ArrayList`, `LinkedList`, `HashMap`, `TreeSet`, `PriorityQueue`
  - all collections are in the `java.util` package

```
import java.util.*;
```

Lists

- **list**: a collection of elements with 0-based **indexes**
  - elements can be added to the front, back, or elsewhere
  - a list has a **size** (number of elements that have been added)



List methods

```
List<String> list = new ArrayList<String>();  // empty
List<Integer> list2 = new LinkedList<Integer>();
list.add("hello");
list.add("goodbye");
System.out.println(list); // ["hello", "goodbye"]
```

`add(`**value**`)`	adds the given value to the list
`add(`**index, value**`)`	Adds the given value at the given index to the list
`contains(`**value**`)`	returns `true` if the given value is found in this list
`indexOf(`**value**`)`	returns the index of the given value in the list (-1 if not found)
`remove(`**value**`)`	removes the given value from the list
`size()`	returns the number of elements in list
`isEmpty()`	returns `true` if the list's size is 0
`toString()`	returns a string such as `"[3, 42, -7, 15]"`

Wrapper classes

Primitive Type	Wrapper Type
int	Integer
double	Double
char	Character
boolean	Boolean

- A wrapper is an object whose sole purpose is to hold a primitive value.
- Once you construct the list, use it with primitives as normal:

```
List<Double> grades = new ArrayList<Double>();
grades.add(3.2);
grades.add(2.7);
...
double myGrade = grades.get(0);
```

Exercise

- Write a program that counts the number of unique words in a large text file (say, *Moby Dick* or the King James Bible).
  - Store the words in a collection and report the # of unique words.
  - Once you've created this collection, allow the user to search it to see whether various words appear in the text file.
- What collection is appropriate for this problem?

The "for each" loop (7.1)

```
for (type name : collection) {
    statements;
}
```

- Provides a clean syntax for looping over the elements of a `List`, `Set`, array, or other collection

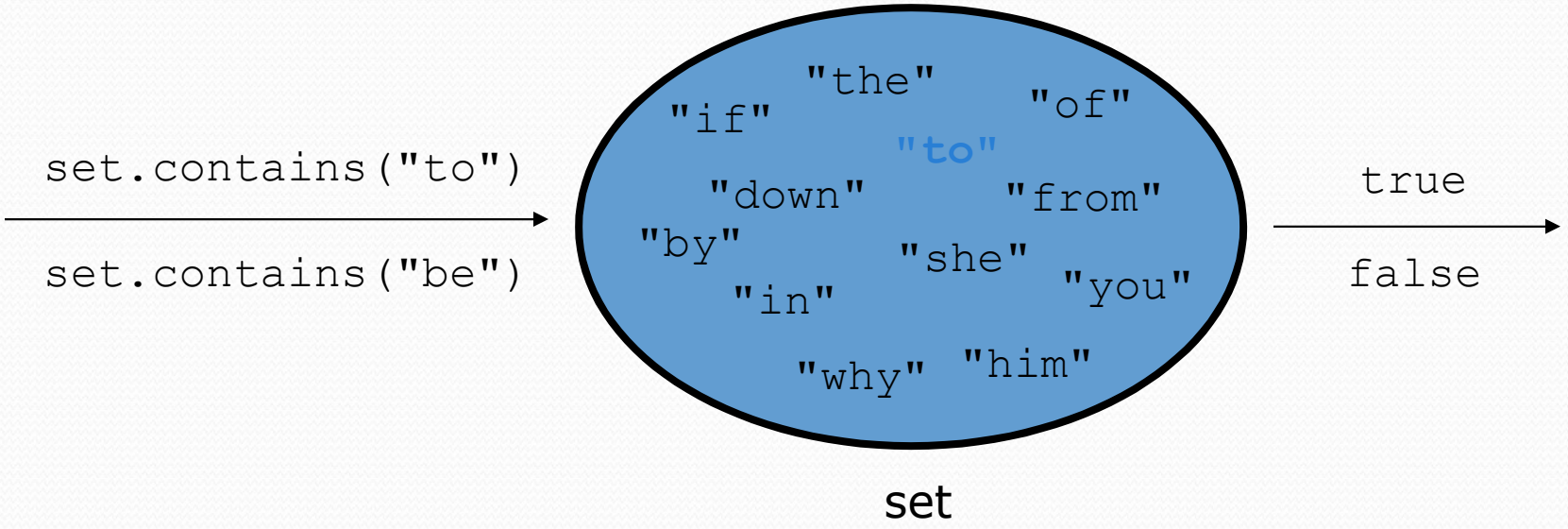
```
List<Double> grades = new ArrayList<Double>();
...

for (double grade : grades) {
    System.out.println("Student's grade: " + grade);
}
```

- More readable and can be more efficient

Sets (11.2)

- **set**: A collection of unique values (no duplicates allowed) that can perform the following operations efficiently:
  - add, remove, search (contains)
  - We don't think of a set as having indexes; we just add things to the set in general and don't worry about order



`Set` implementation

- in Java, sets are represented by `Set` type in `java.util`
- `Set` is implemented by `HashSet` and `TreeSet` classes
  - `TreeSet`: implemented using a "binary search tree";
    pretty fast: **O(log N)** for all operations
    elements are stored in sorted order
  - `HashSet`: implemented using a "hash table" array;
    very fast: **O(1)** for all operations
    elements are stored in unpredictable order

Note: This O(something) notation won't be covered until next week. It's okay not to know what it means yet.

`Set` methods

```
Set<String> set = new TreeSet<String>();      // empty
Set<Integer> set2 = new HashSet<Integer>();
set.add("hello");
set.add("goodbye");
set.add("hello");
System.out.println(set); // ["goodbye", "hello"]
```

`add(`**value**`)`	adds the given value to the set
`contains(`**value**`)`	returns `true` if the given value is found in this set
`remove(`**value**`)`	removes the given value from the set
`clear()`	removes all elements of the set
`size()`	returns the number of elements in list
`isEmpty()`	returns `true` if the set's size is 0
`toString()`	returns a string such as `"[3, 42, -7, 15]"`