ArrayLists

Exercise

- Write a program that reads a file and displays the words of that file as a list.
  - First display all words.
  - Then display them with all plurals (ending in "s") capitalized.
  - Then display them in reverse order.
  - Then display them with all plural words removed.

- Should we solve this problem using an array?
  - Why or why not?

Naive solution

```
String[] allWords = new String[1000];
int wordCount = 0;

Scanner input = new Scanner(new File("data.txt"));
while (input.hasNext()) {
    String word = input.next();
    allWords[wordCount] = word;
    wordCount++;
}
```

- Problem: You don't know how many words the file will have.
  - Hard to create an array of the appropriate size.
  - Later parts of the problem are more difficult to solve.

- Luckily, there are other ways to store data besides in an array.

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*

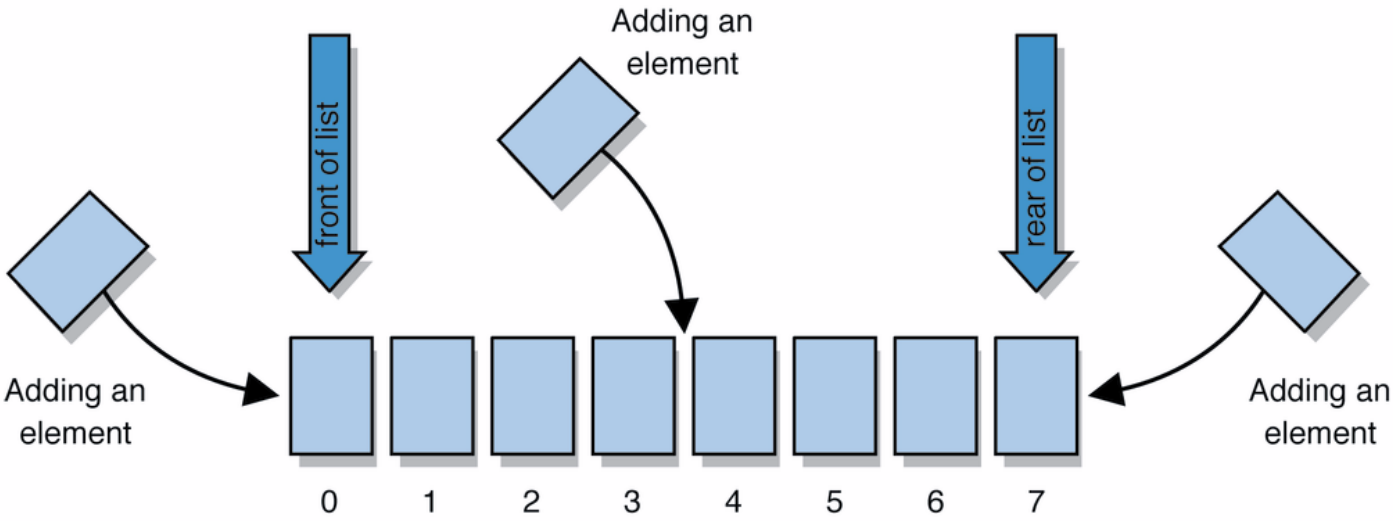
  - examples found in the Java class libraries:
    - `ArrayList`, `LinkedList`, `HashMap`, `TreeSet`, `PriorityQueue`

  - all collections are in the `java.util` package

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

Lists

- **list**: a collection storing an ordered sequence of elements
  - each element is accessible by a 0-based **index**
  - a list has a **size** (number of elements that have been added)
  - elements can be added to the front, back, or elsewhere
  - in Java, a list can be represented as an **`ArrayList`** object



Idea of a list

- Rather than creating an array of boxes, create an object that represents a "list" of items. (initially an empty list.)

```
[]
```

- You can add items to the list.
  - The default behavior is to add to the end of the list.

```
[hello, ABC, goodbye, okay]
```

- The list object keeps track of the element values that have been added to it, their order, indexes, and its total size.
  - Think of an "array list" as an automatically resizing array object.
  - Internally, the list is implemented using an array and a size field.

`ArrayList` methods (10.1)

`add(`**value**`)`	appends value at end of list
`add(`**index**`, `**value**`)`	inserts given value just before the given index, shifting subsequent values to the right
`clear()`	removes all elements of the list
`indexOf(`**value**`)`	returns first index where given value is found in list (-1 if not found)
`get(`**index**`)`	returns the value at given index
`remove(`**index**`)`	removes/returns value at given index, shifting subsequent values to the left
`set(`**index**`, `**value**`)`	replaces value at given index with given value
`size()`	returns the number of elements in list
`toString()`	returns a string representation of the list such as `"[3, 42, -7, 15]"`

`ArrayList` methods 2

`addAll(`**list**`)` `addAll(`**index**`, `**list**`)`	adds all elements from the given list to this list (at the end of the list, or inserts them at the given index)
`contains(`**value**`)`	returns true if given value is found somewhere in this list
`containsAll(`**list**`)`	returns true if this list contains every element from given list
`equals(`**list**`)`	returns true if given other list contains the same elements
`iterator()` `listIterator()`	returns an object used to examine the contents of the list (seen later)
`lastIndexOf(`**value**`)`	returns last index value is found in list (-1 if not found)
`remove(`**value**`)`	finds and removes the given value from this list
`removeAll(`**list**`)`	removes any elements found in the given list from this list
`retainAll(`**list**`)`	removes any elements *not* found in given list from this list
`subList(`**from**`, `**to**`)`	returns the sub-portion of the list between indexes **from** (inclusive) and **to** (exclusive)
`toArray()`	returns the elements in this list as an array

Type Parameters (Generics)

```
ArrayList<Type> name = new ArrayList<Type>();
```

- When constructing an `ArrayList`, you must specify the type of elements it will contain between `<` and `>`.
  - This is called a *type parameter* or a *generic* class.
  - Allows the same `ArrayList` class to store lists of different types.

```
ArrayList<String> names = new ArrayList<String>();
names.add("Marty Stepp");
names.add("Stuart Reges");
```

`ArrayList` vs. array

- construction

```
String[] names = new String[5];
ArrayList<String> list = new ArrayList<String>();
```

- storing a value

```
names[0] = "Jessica";
list.add("Jessica");
```

- retrieving a value

```
String s = names[0];
String s = list.get(0);
```

ArrayList vs. array 2

- doing something to each value that starts with "B"

```
for (int i = 0; i < names.length; i++) {
    if (names[i].startsWith("B")) { ... }
}
```

```
for (int i = 0; i < list.size(); i++) {
    if (list.get(i).startsWith("B")) { ... }
}
```

- seeing whether the value "Benson" is found

```
for (int i = 0; i < names.length; i++) {
    if (names[i].equals("Benson")) { ... }
}
```

```
if (list.contains("Benson")) { ... }
```

Exercise, revisited

- Write a program that reads a file and displays the words of that file as a list.
  - First display all words.
  - Then display them in reverse order.
  - Then display them with all plurals (ending in "s") capitalized.
  - Then display them with all plural words removed.

Exercise solution (partial)

```
ArrayList<String> allWords = new ArrayList<String>();
Scanner input = new Scanner(new File("words.txt"));
while (input.hasNext()) {
    String word = input.next();
    allWords.add(word);
}
System.out.println(allWords);

// remove all plural words
for (int i = 0; i < allWords.size(); i++) {
    String word = allWords.get(i);
    if (word.endsWith("s")) {
        allWords.remove(i);
        i--;
    }
}
```

ArrayList as parameter

```
public static void name(ArrayList<Type> name) {
```

- Example:

```
// Removes all plural words from the given list.
public static void removePlural(ArrayList<String> list) {
    for (int i = 0; i < list.size(); i++) {
        String str = list.get(i);
        if (str.endsWith("s")) {
            list.remove(i);
            i--;
        }
    }
}
```

- You can also return a list:

```
public static ArrayList<Type> methodName(params)
```

`ArrayList` of primitives?

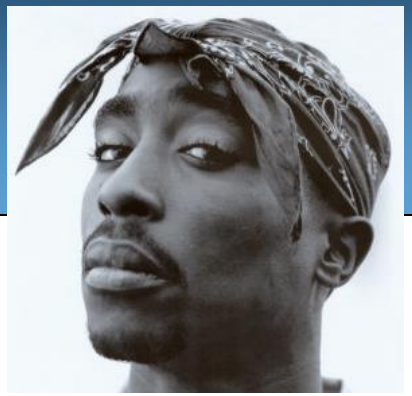
- The type you specify when creating an `ArrayList` must be an object type; it cannot be a primitive type.

```
// illegal -- int cannot be a type parameter
ArrayList<int> list = new ArrayList<int>();
```

- But we can still use `ArrayList` with primitive types by using special classes called *wrapper* classes in their place.

```
// creates a list of ints
ArrayList<Integer> list = new ArrayList<Integer>();
```

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:

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

Exercise

- Write a program that reads a file full of numbers and displays all the numbers as a list, then:
  - Prints the average of the numbers.
  - Prints the highest and lowest number.
  - Filters out all of the even numbers (ones divisible by 2).

Exercise solution (partial)

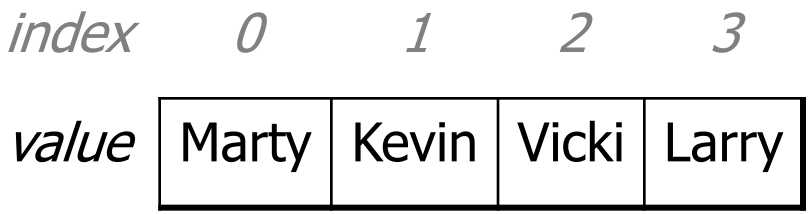
```
ArrayList<Integer> numbers = new ArrayList<Integer>();
Scanner input = new Scanner(new File("numbers.txt"));
while (input.hasNextInt()) {
    int n = input.nextInt();
    numbers.add(n);
}
System.out.println(numbers);
filterEvens(numbers);
System.out.println(numbers);
...

// Removes all elements with even values from the given list.
public static void filterEvens(ArrayList<Integer> list) {
    for (int i = list.size() - 1; i >= 0; i--) {
        int n = list.get(i);
        if (n % 2 == 0) {
            list.remove(i);
        }
    }
}
```

Out-of-bounds

- Legal indexes are between **0** and the **list's size() - 1**.
  - Reading or writing any index outside this range will cause an `IndexOutOfBoundsException`.

```
ArrayList<String> names = new ArrayList<String>();
names.add("Marty");    names.add("Kevin");
names.add("Vicki");    names.add("Larry");
System.out.println(names.get(0));          // okay
System.out.println(names.get(3));          // okay
System.out.println(names.get(-1));         // exception
names.add(9, "Aimee");                     // exception
```



ArrayList "mystery"

```
ArrayList<Integer> list = new ArrayList<Integer>();
for (int i = 1; i <= 10; i++) {
    list.add(10 * i);   // [10, 20, 30, 40, ..., 100]
}
```

- What is the output of the following code?

```
for (int i = 0; i < list.size(); i++) {
    list.remove(i);
}
System.out.println(list);
```

- Answer:

```
[20, 40, 60, 80, 100]
```

ArrayList "mystery" 2

```
ArrayList<Integer> list = new ArrayList<Integer>();
for (int i = 1; i <= 5; i++) {
    list.add(2 * i);   // [2, 4, 6, 8, 10]
}
```

- What is the output of the following code?

```
int size = list.size();
for (int i = 0; i < size; i++) {
    list.add(i, 42);   // add 42 at index i
}
System.out.println(list);
```

- Answer:

```
[42, 42, 42, 42, 42, 2, 4, 6, 8, 10]
```

ArrayList as parameter

```
public static void name(ArrayList<Type> name) {
```

- Example:

```
// Removes all plural words from the given list.
public static void removePlural(ArrayList<String> list) {
    for (int i = 0; i < list.size(); i++) {
        String str = list.get(i);
        if (str.endsWith("s")) {
            list.remove(i);
            i--;
        }
    }
}
```

- You can also return a list:

```
public static ArrayList<Type> methodName(params)
```

Exercise

- Write a method `addStars` that accepts an array list of strings as a parameter and places a `*` after each element.
  - Example: if an array list named `list` initially stores:

    ```
    [the, quick, brown, fox]
    ```

  - Then the call of `addStars(list);` makes it store:

    ```
    [the, *, quick, *, brown, *, fox, *]
    ```

- Write a method `removeStars` that accepts an array list of strings, assuming that every other element is a `*`, and removes the stars (undoing what was done by `addStars` above).

Exercise solution

```
public static void addStars(ArrayList<String> list) {
    for (int i = 0; i < list.size(); i += 2) {
        list.add(i, "*");
    }
}


public static void removeStars(ArrayList<String> list) {
    for (int i = 0; i < list.size(); i++) {
        list.remove(i);
    }
}
```