NC STATE UNIVERSITY

Java Collections

Zybooks: Chapter 11

NC STATE UNIVERSITY

Common Computing Problem

- Read in the contents of a file and process it to:
  - Count the number of lines in the file
  - Print the contents in reverse
  - Remove lines that start with “Q”
- How do we solve this problem?

NC STATE UNIVERSITY

Possible Solution

- Create a big array to hold the contents

```
public String [] readFile(File filename) {
   String [] contents = new String[1000];
   int idx = 0;
   try {
      Scanner in = new Scanner(filename);
      while (in.hasNextLine()) {
         contents[idx++] = in.nextLine();
      }
   } catch (FileNotFoundException e) { … }
   return contents;
}
```

Create a big array.

Read all the inputs.
(Remember that ++ in this case is a post fix increment. The value will be placed at idx and the very last thing is the increment of idx for the location of the next item.)

NC STATE UNIVERSITY

Possible Solution

- Create a big array to hold the contents

```
public String [] readFile(File filename) {
   String [] contents = new String[1000];
   int idx = 0;
   try {
      Scanner in = new Scanner(filename);
      while (in.hasNextLine()) {
         contents[idx++] = in.nextLine();
      }
   } catch (FileNotFoundException e) { … }
   return contents;
}
```

What if the file contains more than 1000 lines?

What if the file only contains 5 lines?

NC STATE UNIVERSITY

Collections

Collection: an object that holds data – i.e., a data structure

- **State**: the elements (data) in the collection
- **Behaviors**: provides methods to clients to add, remove, set, find elements
- **Specializations**: some collections allow duplicates, have ordering, or other customizations

Java Collections Framework provides ***Lists***, Hash, and Sets. Data may be stored in various ways. We'll focus on `ArrayList`.

NC STATE UNIVERSITY

Lists

- List: an ordered collection of elements
  - Elements are 0-indexed
  - The list has a ***size***, which is the number of elements in the list
  - Elements can be added to an empty list, at the front of the list, the middle of the list, or the end of the list.

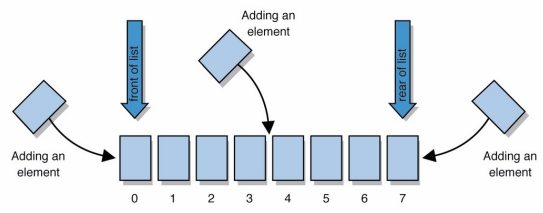


 Image from Building Java Programs

NC STATE UNIVERSITY

Working with a List as an Object

- Start by creating the list. The list is initially empty.

```
[ ]
```

- Add elements to the list (default is to add to the end of the list, but elements may be added to an index between ***0*** and ***size***, inclusive).

```
[apple, banana, pear, peach]
```

The list knows:
- The elements
- Their order and index
- The number of elements in the list (size)

The list "automatically" grows to accommodate new elements.

NC STATE UNIVERSITY

`ArrayList`

- **`ArrayList`** is a list where the elements are stored in an array
- As a client of **`ArrayList`**, we are not concerned about how the elements are stored in the array (for now!)
- Review the **`ArrayList`** API for details about **`ArrayList`** operations

NC STATE UNIVERSITY

`ArrayList` and Type Parameters (Generics)

- When constructing an **`ArrayList`**, you are expected to provide a type parameter (or generic type).
  - Specifies the ***type*** of elements that can be stored in the **`ArrayList`**
  - If no generic type is specified, the **`ArrayList`** stores **`Objects`** (and you have to cast when you get the elements out!)

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

ArrayList<Course> courses = new ArrayList<Course>();
```


NC STATE UNIVERSITY

Arrays → `ArrayList`

Construction

Array

```
String [] words = new String[10];
```

`ArrayList`

```
ArrayList<String> words = new ArrayList<String>();
```

NC STATE UNIVERSITY

Arrays → `ArrayList`

Storing a Value

Array

```
words[3] = "apple";
```

`ArrayList`

```
words.add(3, "apple");
```


NC STATE UNIVERSITY

Arrays → `ArrayList`

Retrieving a Value

Array

```
String word = words[3]; // word = "apple"
```

`ArrayList`

```
String word = words.get(3); // word = "apple"
```

NC STATE UNIVERSITY

Arrays → ArrayList

Working with an element of the array

Array

```
for (int i = 0; i < words.length; i++) {
   if (words[i].length() % 2 == 0) {
      //do something with words[i]
   }
}
```

ArrayList

```
for (int i = 0; i < words.size(); i++) {
   if (words.get(i).length() % 2 == 0) {
      //do something with words.get(i)
   }
}
```


NC STATE UNIVERSITY

Arrays → ArrayList

Finding a value in the list

Array

```
for (int i = 0; i < words.length; i++) {
   if (words[i].equals("apple")) {
      //do something
   }
}
```

ArrayList

```
if (words.contains("apple")) {
   //do something
}
```

NC STATE UNIVERSITY

`ArrayList` – Parameters and Return Types

- **`ArrayList`s** are objects, so they have reference semantics
  - Can be shared with methods as parameters
  - **`ArrayList`s** created in a method can be returned

NC STATE UNIVERSITY

Revisiting File Processing

```
public ArrayList<String>  readFile(File filename) {
   ArrayList<String> contents = new ArrayList<String>();

   try {
      Scanner in = new Scanner(filename);
      while (in.hasNextLine()) {
         contents.add(in.nextLine());
      }
   } catch (FileNotFoundException e) { … }
   return contents;
}
```

NC STATE UNIVERSITY

Exercise

- Write a method that will manipulate an **`ArrayList`** of **`String`**s read in from a file:
  - Reverse the contents of the list
  - Remove lines that end in 's'
  - Transform the lines that end with 's' to all upper case

```
public void transform(ArrayList<String> list) {
   // implement method here
}
```


NC STATE UNIVERSITY

`ArrayList`s of Primitives

- You cannot construct an **`ArrayList`** of a primitive types

```
ArrayList<int> values = new ArrayList<int>();
```



- Instead, use a ***wrapper*** class for the primitive type

```
ArrayList<Integer> values = new ArrayList<Integer>();
```



Primitives Wrapper Classes

- Purpose of the class is to hold a primitive value and treat it as an object.
  - Retrieving an object will automatically cast it to a primitive type

```
ArrayList<Integer> values = new ArrayList<Integer>();
values.add(14);
values.add(37);
int v = values.get(0);
```

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

Exercise

- Given an `ArrayList` of `Integer`s
  - Calculate the average of the values
  - Print the highest and lowest number (max/min loop)
  - Filter out even numbers

```
public void transform(ArrayList<Integer> list) {
    // implement method here
}
```

NC STATE UNIVERSITY

Out-of-Bounds

- **`ArrayList`** bounds: between **`0`** and **`size-1`**, inclusive
  - If out of bounds, **`IndexOutOfBoundsException`** will be thrown.
  - For the add method, bounds is between **`0`** and **`size`** (adding at the end of the list is adding at index **`size`**)

```
ArrayList<Integer> values = new ArrayList<Integer>();
values.add(1);
values.get(1); // exception
values.get(5); // exception
values.get(0); // no exception
```


NC STATE UNIVERSITY

Code Tracing

Given the following code

```
ArrayList<Integer> values = new ArrayList<Integer>();
for (int i = 1; i <= 10; i++) {
   values.add(i * 10);
}  // [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

What would print to the console for the following?

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

NC STATE UNIVERSITY

Code Tracing

Given the following code

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

What would print to the console for the following?

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

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

What happens if you change the continuation test in the loop to `values.size()`?

NC STATE UNIVERSITY

Composition

- Classes can have instances of **`ArrayList`**s as a field
  - Methods can then delegate to the **`ArrayList`** field
  - Don't forget to construct the **`ArrayList`**!

```
public class Student {
   private ArrayList<Course> schedule;

   public Student(…) {
      schedule = new ArrayList<Course>();
      …
   }
}
```

Declaration of the composition relationship!

Construct the ArrayList for use in the class!

NC STATE UNIVERSITY

Testing `ArrayList` Composition

- If a class has an instance of an **`ArrayList`**, you don't need to test the **`ArrayList`** functionality
  - You focus on testing how the class ***uses*** the **`ArrayList`**

Requirement: a student can add a course to their schedule

```
public void addCourseToSchedule(Course c) {
   schedule.add(c);
}
```


NC STATE UNIVERSITY

Testing `ArrayList` Composition

```
@Test
public void testAddCourseToSchedule() {
   Student s = new Student(…);
   Course c = new Course(…);
   s.addCourseToSchedule(c);

   ArrayList<Course> schedule = s.getSchedule();

   // test - check that size is correct
   assertEquals(1, schedule.size());

   // test - check that contents are correct
   assertEquals(c, schedule.get(0));
}
```