CSC 222: Object-Oriented Programming

Fall 2017

Lists, data storage & access

- ArrayList class
  - methods: add, get, size, remove, contains, set, indexOf, toString
  - generics, for-each loop
- example: Dictionary
- Scanner class, file input, text processing
- example: LetterFreq, autoboxing & unboxing

1

Composite data types

String is a composite data type

- each String object represents a collection of characters in sequence
- can access the individual components & also act upon the collection as a whole

many applications require a more general composite data type, e.g.,

- ✓ a dot race will keep track of a sequence/collection of dots
- ✓ a dictionary will keep track of a sequence/collection of words
- ✓ a payroll system will keep track of a sequence/collection of employee records

Java provides several library classes for storing/accessing collections of arbitrary items

2

ArrayList class

an `ArrayList` is a generic collection of *objects*, accessible via an index

- must specify the type of object to be stored in the list
- create an empty `ArrayList<?>` by calling the constructor (no inputs)

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

- add items to the end of the `ArrayList` using `add`

```
words.add("Billy");            // adds "Billy" to end of list
words.add("Bluejay");          // adds "Bluejay" to end of list
```

- can access items in the `ArrayList` using `get`
  - similar to Strings, indices start at 0

```
String first = words.get(0);     // assigns "Billy"
String second = words.get(1);    // assigns "Bluejay"
```

- can determine the number of items in the `ArrayList` using `size`

```
int count = words.size();      // assigns 2
```

3

Simple example

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

words.add("Nebraska");
words.add("Iowa");
words.add("Kansas");
words.add("Missouri");

for (int i = 0; i < words.size(); i++) {
    String entry = words.get(i);
    System.out.println(entry);
}
```

since an `ArrayList` is a composite object, we can envision its representation as a sequence of indexed memory cells

"Nebraska"	"Iowa"	"Kansas"	"Missouri"
0	1	2	3

exercise:

- given an `ArrayList` of state names, output index where "Hawaii" is stored

4

For-each loop

traversing a list is such a common operation that a variant of for loops was introduced to make it easier/cleaner

```
for (TYPE value : ARRAYLIST) {
    PROCESS value
}
```

```
for (int i = 0; i < words.size(); i++) {
    String entry = words.get(i);
    System.out.println(entry);
}
```

```
for (String entry : words) {
    System.out.println(entry);
}
```

```
int count = 0;
for (int i = 0; i < words.size(); i++) {
    String nextWord = words.get(i);
    if (nextWord.length() > 5) {
        count++;
    }
}
```

```
int count = 0;
for (String nextWord : words) {
    if (nextWord.length() > 5) {
        count++;
    }
}
```

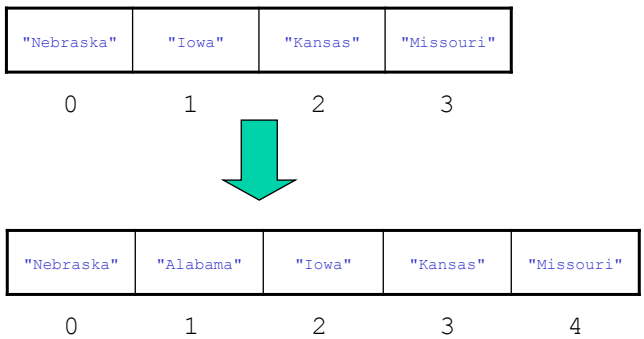
5

Other ArrayList methods: add at index

the general `add` method adds a new item at the end of the `ArrayList`

a 2-parameter version exists for adding at a specific index

```
words.add(1, "Alabama");   // adds "Alabama" at index 1, shifting
                           // all existing items to make room
```



6

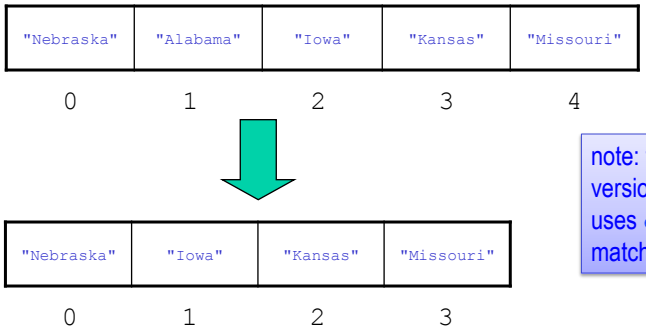
Other ArrayList methods: remove

in addition, you can remove an item using the `remove` method

- either specify the item itself or its index
- all items to the right of the removed item are shifted to the left

```
words.remove("Alabama");
```

```
words.remove(1);
```



note: the item version of remove uses `equals` to match the item

7

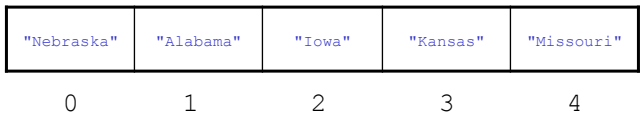
Other ArrayList methods: indexOf & toString

the `indexOf` method will search for and return the index of an item

- if the item occurs more than once, the first (smallest) index is returned
- if the item does not occur in the ArrayList, the method returns -1

```
words.indexOf("Kansas") → 3
words.indexOf("Alaska") → -1
```

similarly, `indexOf` uses `equals`



the `toString` method returns a String representation of the list

- items enclosed in [ ], separated by commas

```
words.toString() → "[Nebraska, Alabama, Iowa, Kansas, Missouri]"
```

- the `toString` method is automatically called when printing an ArrayList

```
System.out.println(words) ≡ System.out.println(words.toString())
```

8

ArrayList<TYPE> methods

`TYPE get(int index)`	returns object at specified index
`TYPE set(int index, TYPE obj)`	sets entry at index to be obj
`boolean add(TYPE obj)`	adds obj to the end of the list
`void add(int index, TYPE obj)`	adds obj at index (shifts to right)
`TYPE remove(int index)`	removes object at index (shifts to left)
`boolean remove(TYPE obj)`	removes specified object (shifts to left) (assumes TYPE has an equals method)
`int size()`	returns number of entries in list
`boolean contains(TYPE obj)`	returns true if obj is in the list (assumes TYPE has an equals method)
`int indexOf(TYPE obj)`	returns index of obj in the list (assumes TYPE has an equals method)
`String toString()`	returns a String representation of the list e.g., "[foo, bar, biz, baz]"

9

Dictionary class

consider designing a simple class to store a list of words

- will store words in an `ArrayList<String>` field
- constructor initializes the field to be an empty list
- `addWord` method adds the word to the list, returns true
- `addWordNoDupes` method adds the word if it is not already stored, returns true if added
- `findWord` method determines if the word is already stored
- `display` method displays each word, one-per-line

```
import java.util.ArrayList;

public class Dictionary {
  private ArrayList<String> words;

  public Dictionary() {
    this.words = new ArrayList<String>();
  }

  public boolean addWord(String newWord) {
    this.words.add(newWord);
    return true;
  }

   public boolean addWordNoDupes(String newWord) {
    if (!this.findWord(newWord)) {
      return this.addWord(newWord);
    }
    return false;
  }

  public boolean findWord(String desiredWord) {
    return this.words.contains(desiredWord);
  }

  public int numWords() {
    return this.words.size();
  }

  public void display() {
    for (String nextWord : words) {
      System.out.println(nextWord);
  }
}
```

10

In-class exercises

download Dictionary.java and try it out

✓ make it so that words are stored in lower-case
- which method(s) need to be updated?

✓ add a method for removing a word

```
/**
 * Removes a word from the Dictionary.
 *   @param desiredWord the word to be removed
 *   @return true if the word was found and removed; otherwise, false
 */
public boolean removeWord(String desiredWord)
```

✓ add a method for finding & returning all partial matches

```
/**
 * Returns an ArrayList of words that contain the specified substring.
 *   @param substr the substring to match
 *   @return an ArrayList containing all words that contain the substring
*/
public ArrayList<String> findMatches(String substr)
```

11

Input files

adding dictionary words one-at-a-time is tedious
- better option would be reading words directly from a file

- `java.io.File` class defines properties & behaviors of text files
- `java.util.Scanner` class provides methods for easily reading from files
  `hasNext` returns true if unread text remains, `next` reads & returns next word

```
import java.io.File;
import java.util.Scanner;

. . .

public Dictionary(String fileName) throws java.io.FileNotFoundException {
  this.words = new ArrayList<String>();

  Scanner infile = new Scanner(new File(fileName));
  while (infile.hasNext()) {
    String nextWord = infile.next();
    this.addWord(nextWord);
  }
  infile.close();
}
```

this addition to the constructor header acknowledges that an error could occur if the input file is not found

opens a text file with the specified name for input

while there are still words to be read from the file, read a word and store it in the Dictionary

12

User input

the Scanner class is useful for reading user input as well

- can create a Scanner object connected to the keyboard (`System.in`)

```
Scanner input = new Scanner(System.in);
```

- can then use Scanner methods to read input entered by the user

`input.next()`	will read the next String (delineated by whitespace)
`input.nextLine()`	will read the next line of text
`input.nextInt()`	will read the next integer
`input.nextDouble()`	will read the next double

```
Scanner input = new Scanner(System.in);

System.out.println("Enter your first name: ");
String firstName= input.next();

System.out.println("Enter your age: ");
int age = input.nextInt();
```

13

Output files

for outputting text to a file, the FileWriter class is easy to use

- `java.io.FileWriter` class provides methods for easily writing to files

```
import java.io.FileWriter;

. . .

public void saveToFile(String filename) throws java.io.IOException {
  FileWriter outfile = new FileWriter(new File(filename));
  for (String nextWord : this.words) {
    outfile.write(nextWord + "\n");
  }
  outfile.close();
}
```

this addition to the method header acknowledges that an error could occur if the file cannot be opened

opens a text file with the specified name for output, writes each word to the file (followed by the end-of-line char)

14