

CSC 221: Computer Programming I

Spring 2008

Lists, data storage & access

- ArrayList class
 - methods: add, get, size, remove, contains, set, indexOf, toString
- example: Dictionary
- file input, Scanner class
- text processing
- example: Skip3 solitaire

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 to-do list will keep track of a sequence/collection of notes
- ✓ 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 `ArrayList<?>` by calling the `ArrayList<?>` 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:

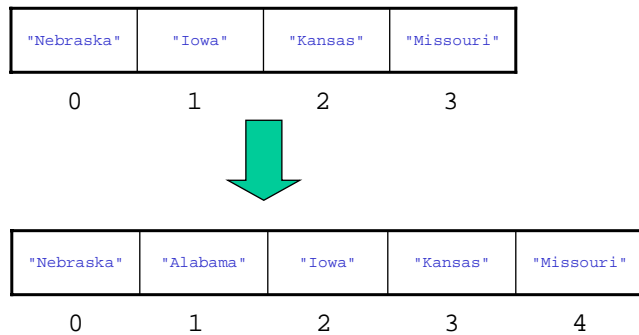
- given an `ArrayList` of state names, output index where "Hawaii" is stored

4

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
```



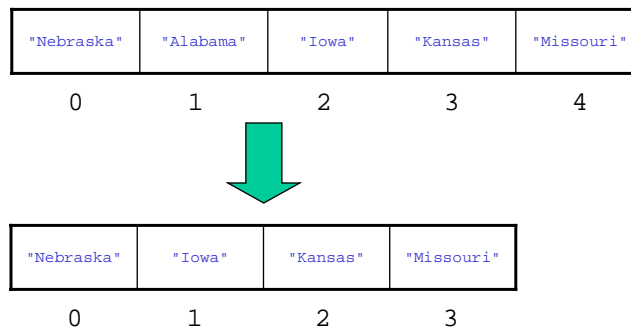
5

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);
```



6

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
```

"Nebraska"	"Alabama"	"Iowa"	"Kansas"	"Missouri"
0	1	2	3	4

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())
```

7

ArrayList<TYPE> methods

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

8

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 if it is not already stored, returns true if added
- `toString` method returns the words in a String (using the `ArrayList` `toString` method)

VERY USEFUL FEATURE IN A CLASS! Automatically called when the object is printed.

```
import java.util.ArrayList;

public class Dictionary {
    private ArrayList<String> words;

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

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

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

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

    public String toString() {
        return this.words.toString();
    }
}
```

any class that uses an `ArrayList` must load the library file that defines it

9

In-class exercises

download [Dictionary.java](#) and try it out

- add words
 - try adding a duplicate word
 - call `toString` to see the String form of the list
- ✓ 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) {
```

10

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

```
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);
    }
}
```

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

11

In-class exercise

use a text editor to create a file of words in the BlueJ project folder

- construct a Dictionary object using the new constructor
- this will automatically load the words from the file
- can then view the words using `toString`

what about capitalization?

what about punctuation?

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

public class Dictionary {
    private ArrayList<String> words;

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

    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);
        }
    }

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

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

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

    public String toString() {
        return this.words.toString();
    }
}
```

12

Processing words

it would be useful to be able to strip a word of punctuation & other chars

- `Character.isLetterOrDigit(ch)` returns true if ch is a letter or digit
- can define a private helper method that takes a word and returns a copy with all non-letters and digits removed
- make it private so that other methods can call it, but it is invisible to the outside

```
private String strip(String word) {
    word = word.toLowerCase();

    String copy = "";
    for (int i = 0; i < word.length(); i++) {
        char ch = word.charAt(i);
        if (Character.isLetterOrDigit(ch)) {
            copy += ch;
        }
    }
    return copy;
}
```

first thing, convert the word to lowercase

create a String that will contain copies of letters from the word

for each char in the word, if it is a letter or digit then add it to the copy string

finally, return the stripped copy

13

In-class exercise

where do we do the strip?

- could just be when reading words from a file
- or, could put the call in `addWord` so that even user-added words are stripped
- should words be stripped in `findWord` and `removeWord`?

what if we wanted to keep track of the total number of words (not just unique words)?

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

public class Dictionary {
    private ArrayList<String> words;

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

    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(this.strip(nextWord));
        }
    }
    . . .

    private String strip(String word) {
        word = word.toLowerCase();

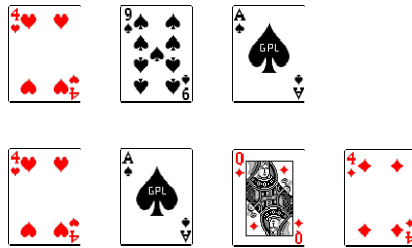
        String copy = "";
        for (int i = 0; i < word.length(); i++) {
            char ch = word.charAt(i);
            if (Character.isLetterOrDigit(ch)) {
                copy += ch;
            }
        }
        return copy;
    }
}
```

14

HW8 application

Skip-3 Solitaire:

- cards are dealt one at a time from a standard deck and placed in a single row
- if the rank or suit of a card matches the rank or suit either 1 or 3 cards to its left, then that card (and any cards beneath it) can be moved on top
- goal is to have the fewest piles when the deck is exhausted



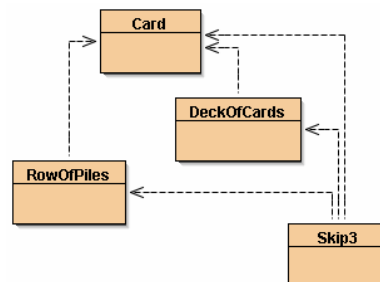
15

Skip-3 design

what are the entities involved in the game that must be modeled?

for each entity, what are its behaviors? what comprises its state?

which entity relies upon another? how do they interact?



16

Card & DeckOfCards

```
public class Card {
    private String cardStr;

    public Card(String cardStr) { ... }

    public char getRank() { ... }
    public char getSuit() { ... }

    public boolean matches(Card other) { ... }
    public boolean equals(Object other) { ... }

    public String toString() { ... }
}
```

card class encapsulates the behavior of a playing card

- cohesive?

DeckOfCards class encapsulates the behavior of a deck

- cohesive?
- coupling with card?

```
public class DeckOfCards {
    private ArrayList<Card> deck;

    public DeckOfCards() { ... }

    public void shuffle() { ... }
    public Card dealCard() { ... }
    public void addCard(Card c) { ... }

    public int cardsRemaining() { ... }
    public String toString() { ... }
}
```

17

Card class

```
public class Card {
    private String cardStr;

    public Card(String cardStr) {
        this.cardStr = cardStr.toUpperCase();
    }

    public char getRank() {
        return this.cardStr.charAt(0);
    }

    public char getSuit() {
        return this.cardStr.charAt(1);
    }

    public String toString() {
        return this.cardStr;
    }

    public boolean matches(Card other) {
        return (this.getRank() == other.getRank() ||
                this.getSuit() == other.getSuit());
    }

    public boolean equals(Object other) {
        return (this.getRank() == ((Card)other).getRank() &&
                this.getSuit() == ((Card)other).getSuit());
    }
}
```

when constructing a Card, specify rank & suit in a String,

e.g.,

```
Card c = new Card("JH");
```

accessor methods allow you to extract the rank and suit

for esoteric reasons, the parameter to the equals method must be of type Object (the generic type that encompasses all object types)

- must then cast the Object into a Card

18

```

import java.util.ArrayList;
import java.util.Collections;

public class DeckOfCards {
    private ArrayList<Card> deck;

    public DeckOfCards() {
        this.deck = new ArrayList<Card>();

        String suits = "SHDC";
        String ranks = "23456789TJQKA";
        for (int s = 0; s < suits.length(); s++) {
            for (int r = 0; r < ranks.length(); r++) {
                Card c = new Card("" + ranks.charAt(r) + suits.charAt(s));
                this.deck.add(c);
            }
        }
    }

    public void addCard(Card c) {
        this.deck.add(0, c);
    }

    public Card dealCard() {
        return this.deck.remove(this.deck.size()-1);
    }

    public void shuffle() {
        Collections.shuffle(this.deck);
    }

    public int cardsRemaining() {
        return this.deck.size();
    }

    public String toString() {
        return this.deck.toString();
    }
}

```

DeckOfCards class

the constructor steps through each suit-rank pair, creates that card, and stores it in the ArrayList

add a card at the front (index 0)

deal from the end (index size()-1)

the Collections class contains a static method for randomly shuffling an ArrayList

toString method uses the ArrayList toString, e.g., [QH, 2D, 8C]

19

Dealing cards: silly examples

```

import java.util.ArrayList;

public class Dealer {
    private DeckOfCards deck;

    public Dealer() {
        this.deck = new DeckOfCards();
        this.deck.shuffle();
    }

    public void dealTwo() {
        Card card1 = this.deck.dealCard();
        Card card2 = this.deck.dealCard();

        System.out.println(card1 + " " + card2);

        if (card1.getRank() == card2.getRank()) {
            System.out.println("IT'S A PAIR");
        }
    }

    public ArrayList<Card> dealHand(int numCards) {
        ArrayList<Card> hand = new ArrayList<Card>();
        for (int i = 0; i < numCards; i++) {
            hand.add(this.deck.dealCard());
        }

        return hand;
    }
}

```

constructor creates a randomly shuffled deck

dealTwo deals two cards from a deck and displays them (also identifies a pair)

dealHand deals a specified number of cards into an ArrayList, returns their String representation

20

HW8: Skip-3 solitaire

you are to define a `RowOfPiles` class for playing the game

- `RowOfPiles()`: constructs an empty row (ArrayList of Cards)
- `void addPile(Card top)`: adds a new pile to the end of the row
- `boolean movePile(Card fromTop, Card toTop)`: moves one pile on top of another, as long as they match and are 1 or 3 spots away
- `int numPiles()`: returns number of piles in the row
- `String toString()`: returns String representation of the row

a simple `skip3` class, which utilizes a `RowOfPiles` to construct an interactive game, is provided for you

- you will make some improvements (error messages, card counts, etc.)

21

```
import java.util.Scanner;

public class Skip3 {
    private DeckOfCards deck;
    private RowOfPiles row;

    public Skip3() {
        this.restart();
    }

    public void restart() {
        this.deck = new DeckOfCards();
        this.deck.shuffle();
        this.row = new RowOfPiles();
    }

    public void playGame() {
        Scanner input = new Scanner(System.in);

        boolean gameOver = false;
        while (!gameOver) {
            System.out.println(this.row);
            System.out.print("Action? ");

            char response = input.next().toLowerCase().charAt(0);
            if (response == 'd') {
                if (this.deck.cardsRemaining() > 0) {
                    this.row.addPile(this.deck.dealCard());
                }
            }
            else if (response == 'm') {
                String from = input.next();
                String to = input.next();
                this.row.movePile(new Card(from), new Card(to));
            }
            else if (response == 'e') {
                gameOver = true;
            }
        }
    }
}
```

Skip3

`skip3` class utilizes a `DeckOfCards` and a `RowOfPiles`

a `Scanner` object is used to read commands from the user

`new Scanner(System.in)` specifies input is to come from the keyboard

the `next()` method reads the next String (delineated by whitespace) entered by the user

22