CSE 142

Iteration Patterns

Introduction

- Review
  - Collection Classes
  - Iteration and iterators
- Today
  - Iteration patterns and problem solving – how to design a loop
  - Comparing objects, particularly Strings

Running Example for Today

- Collection of weather information for several days
- Each item in the collection contains
  - A description: "clear", "partly cloudy", "snow", etc.
  - High and low temperatures for the day
  - Amount of rainfall that day
- Problems: Examine weather data and
  - Display some or all data
  - Calculate statistics or other information
  - Extract selected data
- Goal: Observe and learn patterns

Weather Data Representation

- Weather data for a single day

```
public class DailyWeather {
    public String description;      // "sunny", "partly cloudy", "rain", etc.
    public double high;             // high temperature for day
    public double low;              // low temperature for day
    public double rain;             // rainfall for the day; 0.0 if none

    /** Construct new DailyWeather object with given initial values */
    public DailyWeather (String description, double high, double low, double rain) { … }
    /** Return string representation of this DailyWeather object */
    public String toString( ) { … }
}
```

- For this example, we're treating DailyWeather as a helper class, meant only to be used to implement collection of weather info, so we'll manipulate the fields directly.
  - (Not good strategy if this class is used more widely)

Collection of Weather Data

```
Class WeatherInfo {
    private ArrayList weather;      // collection of DailyWeather records

    /** Construct empty WeatherInfo object */
    public WeatherInfo( ) {
      this.weather = new ArrayList( );
    }

    /** Add DailyWeather object to this collection */
    public void add(DailyWeather d) {
      this.weather.add(d);
    }
    …
}
```


Processing the Collection

- Sample problems:
  - Print the weather data on System.out
  - Print total rainfall summed over all data in the collection
  - Print # of days with no rainfall
  - Print % of days with high temperature < 75.0
  - Print number of days described as "sunny"
  - Extract a new WeatherInfo collection containing all records in this collection labeled "sunny"
- What do these have in common?
- How do they differ?

Basic Pattern

```
public <type> <name> ( <parameters> ) {
    <initialize>
    Iterator iter = this.weather.iterator( );
    while (iter.hasNext( )) {
      DailyWeather w = (DailyWeather) iter.next( );
      < process w>
    }
    <final processing>
}
```

- **Focus on loop design**
  - **What are <initialize>, <process w>, <final processing> ?**
  - **Invent names (variables) as needed**
  - **Usually best to focus on <process w> at first**

Print All Daily Weather Records

```
public void printRecords ( ) {
                                                    // initialize

    Iterator iter = this.weather.iterator( );
    while (iter.hasNext( )) {
      DailyWeather w = (DailyWeather) iter.next( );
                                                    // process w


    }
                                                    // final processing
}
```


Calculate Total Rainfall

```
public double totalRain ( ) {
                                                    // initialize

    Iterator iter = this.weather.iterator( );
    while (iter.hasNext( )) {
      DailyWeather w = (DailyWeather) iter.next( );
                                                    // process w


    }
                                                    // final processing
}
```


Calculate # of Days with No Rainfall

```
public int numberDry ( ) {
                                                    // initialize

    Iterator iter = this.weather.iterator( );
    while (iter.hasNext( )) {
      DailyWeather w = (DailyWeather) iter.next( );
                                                    // process w


    }
                                                    // final processing
}
```


Calculate % of Days with Temp < t

```
public double percentCold (double t) {
                                                    // initialize

    Iterator iter = this.weather.iterator( );
    while (iter.hasNext( )) {
      DailyWeather w = (DailyWeather) iter.next( );
                                                    // process w


    }
                                                    // final processing
}
```


Calculate # of “sunny” Days

```
public int numberSunny( ) {
                                                    // initialize

    Iterator iter = this.weather.iterator( );
    while (iter.hasNext( )) {
      DailyWeather w = (DailyWeather) iter.next( );
                                                    // process w


    }
                                                    // final processing
}
```

Comparing Strings (1)

- == and != probably don't do what you want for Strings (or other objects)
  - Tests object *identity* (are two things the same String object?).
  - Doesn't test object *equality* (do the two Strings contain the same sequence of characters?).
- Can compare any two objects with method *equals*
  - obj1.equals(obj2) is true if the objects are "equal."
  - The meaning of "equal" depends on definition of equals for the class of the objects.
  - For Strings, obj1.equals(obj2) if they have the same sequence of characters.

Comparing Strings (2)

- Besides equals, class String implements compareTo
  - Returns an int.
- If s1 and s2 are strings,
  - s1.compareTo(s2) == 0 if s1 and s2 are the same.
  - s1.compareTo(s2) < 0 if s1 < s2.
  - s1.compareTo(s2) > 0 if s1 > s2.
- Ordering depends on order in the underlying Unicode character set.
  - Makes sense for English alphabet, but not necessarily other alphabets.

Calculate # of "sunny" Days - Revisited

```
public int numberSunny( ) {
                                                    // initialize

    Iterator iter = this.weather.iterator( );
    while (iter.hasNext( )) {
      DailyWeather w = (DailyWeather) iter.next( );
                                                    // process w


    }
                                                    // final processing

}
```


Make a New List with All "sunny" Days

```
public ArrayList sunnyDays( ) {
                                                    // initialize

    Iterator iter = this.weather.iterator( );
    while (iter.hasNext( )) {
      DailyWeather w = (DailyWeather) iter.next( );
                                                    // process w


    }
                                                    // final processing

}
```


Iteration Summary

- We saw three different kinds of iterations:
  - *Traversal* – Do something with each item (print, modify).
  - *Reduction* – Compute some summary information extracted from the items (averages, totals, counts).
  - *Filtering* – Create a new collection that is a subset of the original collection, based on some filtering criteria (sunny days)