CSE 142 Summer 2001

Iteration Patterns

Introduction

- Review
  - Collection Classes
  - Iteration and iterators
- Today
  - Iteration patterns and problem solving
  - 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

```
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 rainfall) { … }
    /** Return string representation of this DailyWeather object */
    public String ToString( ) { … }
```

- For this example, we’re treating DailyWeather as an auxiliary class, meant only to be used to implement collection of weather info, so we’ll manipulate 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);
    }
    …
}
```


Example Collection

```
WeatherInfo w = new WeatherInfo( );
w.add(new DailyWeather(“Sunny”, 80, 67, 0.0));
w.add(new DailyWeather(“Sunny”, 75, 63, 0.0));
w.add(new DailyWeather(“Cloudy”, 76, 65, 0.05));
w.add(new DailyWeather(“Flood”, 71, 54, 4.6));
```

- Draw the picture

Processing the Collection

- Sample problems
  - Print the weather data on System.out
  - Print # days with high temperature < 75.0
  - Print total rainfall summed over all data in the collection
  - Print % of days with no rainfall
  - Print number of days described as “sunny”
  - Extract a new WeatherInfo collection containing all records in this collection not labeled “sunny”
- What do these have in common?
- How do they differ?

Basic pattern

```
/** Process collection */
public <type> <name> ( <parameters> ) {
    <initialize>
    Iterator it = this.weather.iterator( );
    while (it.hasNext( )) {
      DailyWeather w = (DailyWeather) it.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 it = this.weather.iterator( );
    while (it.hasNext( )) {
      DailyWeather w = (DailyWeather) it.next( );
                                                    // process w


    }
                                                    // terminate
}
```


Calculate Total Rainfall

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

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


    }
                                                    // terminate
}
```


Calculate # of Days with No Rainfall

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

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


    }
                                                    // terminate
}
```


Calculate % of Days with Temp < t

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

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


    }
                                                    // terminate
}
```

Calculate # of "sunny" Days

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

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


    }
                                                    // terminate
}
```


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)
  - Don't test object equality (do the two strings have the same value)
- Can compare any two objects with method equals
  - obj1.equals(obj2) is true if the objects are "equal"
  - Meaning of "equal" depends on definition of equals for the class of the objects
  - For Strings, obj1.equals(obj2) if they contain the same String value
  - We'll see later how to define this for our own classes

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
  - Fast, but not always correct alphabetical order for natural languages (other Java libraries are available for those comparisons)

Calculate # of "sunny" Days - Revisited

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

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


    }
                                                    // terminate

}
```


Calculate # of "cloudy" or "rainy" Days

```
/** ... */
public int cloudsAndRain( ) {
                                                    // initialize

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


    }
                                                    // terminate

}
```


Get a New List with all "sunny" days

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

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


    }
                                                    // terminate

}
```

Iteration Summary

- As you program, start to pick up common patterns
  - Today: iterating through a collection
  - The larger your toolbox, the more proficient you will become
- 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