
CSE 142

Iteration Patterns

5/7/2004

(c) 2001-3, University of Washington

N-1

Outline for Today

- **Quick Review**
 - Collection classes
 - Iteration and iterators
- **Today**
 - Iteration patterns and problem solving – how to design a loop
 - Comparing objects, particularly Strings
 - Kinds of iterations: traversals, reductions, and filtering

5/7/2004

(c) 2001-3, University of Washington

N-2

Running Example

- Collection of information about MP3 audio files
- Each item in the collection describes one song
 - Title
 - Artist
 - Length in seconds
- **Problems: Examine collection and**
 - Display some or all data
 - Calculate statistics or other information
 - Extract selected data
- **Goal: Observe and learn patterns**

5/7/2004

(c) 2001-3, University of Washington

N-3

MP3 Data Representation

- **Description of a single MP3 song file**

```
public class MP3 {
    private String title;           // song title
    private String artist;         // performing artist
    private int length;           // song length in seconds

    /** Construct new MP3 object with given initial values */
    public MP3 (String name, String artist, int length) { ... }
    /** Getter methods to access song data */
    public String getTitle() { return title; }
    public String getArtist() { return artist; }
    public int getLength() { return length; }
    /** Return string representation of this MP3 object */
    public String toString() { ... }
}
```

5/7/2004

(c) 2001-3, University of Washington

N-4

Collection of MP3 Data

```
public class MP3Library {
    private ArrayList songs; // collection of MP3 objects

    /** Construct empty MP3Library object */
    public MP3Library() {
        songs = new ArrayList();
    }

    /** Add an MP3 object to this collection */
    public void add(MP3 song) {
        songs.add(song);
    }
    ...
}
```

5/7/2004

(c) 2001-3, University of Washington

N-5

Processing the Collection

- **Sample problems:**
 - Print the song collection on System.out
 - Compute length of all songs if played one after the other
 - Compute number of songs with length > 3 minutes
 - Print all songs with artist of "Beatles"
 - Return true or false depending on whether the collection contains a song with the title "Yesterday"
 - Extract a new MP3Library collection containing all MP3s in the collection by the Rolling Stones
- What do these tasks have in common?
- How do they differ?

5/7/2004

(c) 2001-3, University of Washington

N-6

Basic Pattern

```
public <type> <name> ( <parameters> ) {
    <initialize>
    Iterator iter = songs.iterator();
    while (iter.hasNext()) {
        MP3 song = (MP3) iter.next();
        < process song>
    }
    <final processing>
}
```



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

5/7/2004

(c) 2001-3, University of Washington

N-7

Print All Songs

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

    Iterator iter = songs.iterator();
    while (iter.hasNext()) {
        MP3 song = (MP3) iter.next();
        // process

    }
    // final processing
}
```

5/7/2004

(c) 2001-3, University of Washington

N-8

Calculate Total Length of Songs

```
public int totalLength() {  
    // initialize  
  
    Iterator iter = songs.iterator();  
    while (iter.hasNext()) {  
        MP3 song = (MP3) iter.next();  
        // process  
    }  
    // final processing  
}
```

5/7/2004

(c) 2001-3, University of Washington

N-9

Calculate % of Songs With Length < t

```
public double percentShort(double t) {  
    // initialize  
  
    Iterator iter = songs.iterator();  
    while (iter.hasNext()) {  
        MP3 song = (MP3) iter.next();  
        // process  
    }  
    // final processing  
}
```

5/7/2004

(c) 2001-3, University of Washington

N-10

Calculate # of Songs by an Artist

```
public int numberSongsBy(String artist) {  
    // initialize  
  
    Iterator iter = songs.iterator();  
    while (iter.hasNext()) {  
        MP3 song = (MP3) iter.next();  
        // process  
    }  
    // final processing  
}
```

5/7/2004

(c) 2001-3, University of Washington

N-11

Comparing Strings – equals

- **== and != probably don't do what you want for Strings (or other objects)**

- **Tests object identity**

"are these two things the same String object?"

- **Doesn't test object values**

"do these two Strings contain the same sequence of characters?"

- **Draw pictures to see if you understand**

```
String s1 = "CSE142";
```

```
String s2 = "CSE142";
```

```
String s3 = s1;
```

//True or false:

```
s1 == s2
```

```
s1 == s3
```

```
s2 == s3
```

```
s1.equals(s2)
```

```
s1.equals(s3)
```

```
s2.equals(s2)
```

5/7/2004

(c) 2001-3, University of Washington

N-12

More About *equals*

- Can compare any two objects with method *equals*
 - All objects have a least a default *equals* method
- The meaning of “equals” depends on definition of equality for the class of the objects
 - For Strings, `obj1.equals(obj2)` if they have the same sequence of characters (case-sensitive)
 - In general: `obj1.equals(obj2)` is supposed to be true if the object values are “equal”
 - This is a programming convention
 - Unenforceable except by practice and social contract

5/7/2004

(c) 2001-3, University of Washington

N-13

Comparing Strings – *compareTo*

- 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* comes before *s2*
 - `s1.compareTo(s2) > 0` if *s1* comes after *s2*
- “Before” and “after” depends on order in the underlying Unicode character set
 - An international standard
 - Used in Java but defined externally

5/7/2004

(c) 2001-3, University of Washington

N-14

Calculate # of Songs by Artist Revisited

```
public int numberSongsBy(String artist) {  
    // initialize  
  
    Iterator iter = songs.iterator();  
    while (iter.hasNext()) {  
        MP3 song = (MP3) iter.next();  
        // process  
    }  
    // final processing  
}
```

5/7/2004

(c) 2001-3, University of Washington

N-15

Search For Song by Title

```
public boolean containsTitle(String title) {  
    // initialize  
  
    Iterator iter = songs.iterator();  
    while (iter.hasNext()) {  
        MP3 song = (MP3) iter.next();  
        // process  
    }  
    // final processing  
}
```

5/7/2004

(c) 2001-3, University of Washington

N-16

Make New List of Songs by an Artist

```
public MP3Library songsBy(String artist) {  
    // initialize  
    Iterator iter = songs.iterator();  
    while (iter.hasNext()) {  
        MP3 song = (MP3) iter.next();  
        // process  
    }  
    // final processing  
}
```

5/7/2004

(c) 2001-3, University of Washington

N-17

Iteration Summary

- We saw three different patterns 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 (artist name, song title, length)
- All of these have the same basic pattern for iterating through the original collection
 - The initialization, processing, and final processing steps depend on the specific task

5/7/2004

(c) 2001-3, University of Washington

N-18