CSE 142 Summer 2001

Classes, Collections &
Introduction to Iteration

Introduction

- Review
  - Song class
  - Compact Disc collection
  - A collection implementation: ArrayList
  - Casting
- Today
  - Implementing a collection class: CompactDisc
  - toString( )
  - Iteration: processing the items in a collection

Problem Setting

- We want to create a class to describe a compact disc
- Class Song
  - Description of a single song
  - Representation: Song title, length (time in seconds)
  - Operations: construct Song; get Song title or length
- Class CompactDisc
  - Description of the tracks (songs) on a single disc
  - Representation:
    Compact disc title, total length
    List of individual Songs on the disc
  - Operations: construct empty CompactDisc; add Song to CompactDisc, get Song given position, get total length, etc.

Class Song (review)

```
/** Representation of a single song on a CD */
public class Song {
  private int seconds;    // song length
  private String title;   // song title
  /** Construct new song ... */
  public Song (String title, int seconds) { ... }
  /** get Song title */
  public String getTitle( ){ return this.title; }
  /** get Song time */
  public int getSeconds( ) { return this.seconds; }
}
```


Client view of class CompactDisc

```
// Create a new CompactDisc and perform operations on it...
CompactDisc cd = new CompactDisc( );
Song tune = new Song("Lovely Melody", 245);
cd.add(tune);
cd.add(new Song("Elevator Music", 640));
Song muzak = cd.get(1);
System.out.println("Total length of " + cd.getTitle( ) + " is " +
                   cd.getSeconds( ) + " seconds.";
```


Implementation of class CompactDisc (1)

- Representation:
  - Title and total seconds are simple (String and int)
  - List of Songs: Use a standard Java collection class: ArrayList

```
public class CompactDisc {
    // instance variables
    private ArrayList songs;          // list of Songs on this CD
    private String title;             // CD title
    private int totalSeconds;         // total length of CD
```

Implementation of class CompactDisc (2)

- Key operation: add a Song to this CompactDisc

```
/** Add song to end of this CompactDisc song list
public void add(Song song) {
    this.songs.add(song);
    this.totalSeconds = this.totalSeconds + song.getSeconds( );
}
```

- Draw the picture

```
CompactDisc cd = new CompactDisc( );
Song aTune = new Song("Lovely Melody", 245);
Song anotherTune = new Song("Elevator Music", 640);
cd.add(aTune);
cd.add(anotherTune);
```


CompactDisc Picture

Implementation of class CompactDisc (3)

- Get a Song from the list

```
/** Return song at given position in the CompactDisc list ... */
public Song get(int pos) {
    return (Song)this.songs.get(pos);
}
```

- Dissect the expression in the return statement: what objects are referred to? What are the types?

Aside: Objects as Strings

- We've already noticed that if we print something like a Color, we get a useful description

```
Color r = Color.red;
Color c = Color.cyan;
System.out.println(r);
System.out.println(c);

Output:
    java.awt.Color[r=255,g=0,b=0]
    java.awt.Color[r=0,g=255,b=255]
```

- How does this happen?
- How can we get our classes to do something similar?

toString( )

- If a class contains a method toString( ), it will be used to generate a String representation of the object when it is used somewhere that a String is expected (like println)

```
class Song {
    ...
    public String toString( ) {
      String description = "Song: title = " + this.title + ", length = " + this.seconds;
      return description;
    }
}
```

- Exact format is up to implementer – do something useful
- Useful for debugging – can print an object to get info about it

Getting information from a Collection

- Let's add a method printTitles( ) to CompactDisc to print the Song titles in the CompactDisc. How?

```
/** Print the titles of the songs in this CD */
public void printTitles( ) {
    Song s0 = this.get(0);
    System.out.println(s0.getTitle( ));
    Song s1 = this.get(1);
    System.out.println(s1.getTitle( ));
    Song s2 = this.get(2);
    System.out.println(s2.getTitle( ));
    ...
}
```

- This doesn't generalize very well. Why not?

Repetition

- What we really want to do is repeat some statements once for each Song in the list

  start at the beginning of the list
  repeat the following:
  if there is another Song in the list,
  print its title and advance to the next Song on the list
  keep going until the last Song's title has been printed

- So we need two things:
  - Some way to repeat statements
  - Some way to access the elements of the Song list one after the other

The while loop pattern

- New Java statement to repeatedly execute other statements:

```
while (<conditional-expression>) {
  <body-statements>
}
```

- Meaning: repeatedly do the following
  - Evaluate <conditional-expression> (a boolean expression)
  - If <conditional-expression> is true, execute <body-statements>

This cycle repeats until <conditional-expression> evaluates to false at some point

Iterators: Iterating over a Collection

- We could figure out how many items are in the collection, then process them in order (get(0), get(1), etc.)
- Better: make the notion of iteration explicit.
- Any collection can be asked to provide an object that allows us to sequentially access the items in the collection.
- The general term for this object is *iterator*
- In Java, there are several different kinds; we'll look at the one that works with ArrayList: *Iterator*

Iterator Operations

- Here are the methods provided by an Iterator:

```
/** return the next Object in the iteration. */
public Object next();

/** return true if the iteration has more elements. */
public boolean hasNext();
```

  - This doesn't seem useful enough to do anything.
  - How do we use this to access every item in an Iteration?

Using an Iterator

- Pattern: *ask the collection for an iteratior and iterate over it*
- Example: Go through the Songs in a CompactDisc and print their titles
  - We get an iterator object from the songs collection (an ArrayList)

```
/** Print the titles of the songs in this CompactDisc */
public void printTitles( ) {
    Iterator it = this.songs.iterator( );
    while (it.hasNext( )) {
        Song currentSong = (Song)it.next( );
        System.out.println(currentSong.getTitle( ));
    }
```


Trace

- Trace that code assuming that songs contains

  {"Lovely Tune", 45}, {"Big Hit", 300}, {"The End", 182}

  - Draw the picture

Exercise 1: Finding the Longest Song

- Suppose we want to print the title of the longest Song?
- How would we do it?

Exercise 2: Print Titles of all Songs > 1 min.

- Your code goes here: