CSE 142 Summer 2001

Designing Classes &
Introduction to Collections

Introduction

- Quick review
  - Objects = collection of data and methods
  - Methods as operational abstractions
  - Constructors and initialization
- Today
  - Designing classes
  - Relationship between classes
  - Introduction to collections
  - Class Object and casting

Analysis

- What objects do these names refer to?
  - Itchy, Tom, Mickey
  - Ronald, George, Bill, George Jr.
  - Elvis, John, Janis, Jimi
- Particular instances (concrete objects) vs. abstract concepts
- Particulars (objects) vs. universals (classes)

An Example Domain

- A music collection catalog. Start with the compact disc, what are its properties?

Music Collection: Relationships

- Show the relationships between the following things:
  - MusicCollection, CompactDisc, Song

Representing a Song

- What are the instance variables? Methods?

```
public class Song {
  private int seconds;    // song length
  private String title;    // song title
  /** Construct new song … */
  public Song (String title, int length) { … }
  /** get Song title */
  public String getTitle( ){ return this.title; }
  /** get Song time */
  public int getTime( ) { return this.seconds; }
}
```

Using Songs

- Let's create a new Song:

```
Song aSong = new Song("Imagine", 175);
```

- Getting information about the song:

```
System.out.println("Time of " + aSong.getTitle( ) + " is " + aSong.getTime( ) );
```

- How do we combine several of these to represent information about a CD?

Collections in the Real World

- Think about:
  - dictionary
  - class list
  - deck of cards
  - library
- These things are all collections.
- Ordered collections vs. Unordered collections
- How can we represent the songs on a CD?

An Ordered Collection: ArrayList

- ArrayList is a Java class that specializes in representing an ordered collection of things. Here's part of its interface:

```
public class ArrayList {

  // return the size of the collection
  public int size( );

  // return the object at the given index (numbered from 0)
  public Object get(int index);

  // Add the given object to the end of the collection
  public void add(Object o);

  // Remove the object at the given position from the collection.
  public Object remove(int index);
}
```

- New: class Object – means any kind of object at all

Using ArrayLists (1)

- Adding things:

```
ArrayList names = new ArrayList ( );
names.add("Billy");
names.add("Susan");
names.add("Frodo");
```

- Getting the size:

```
int numberOfNames = names.size( );
```

- Removing things:

```
names.remove("Billy");
```


Using ArrayLists (2)

- Accessing items. ArrayLists provide *indexed* access. We can ask for the n-th item of the list.

```
ArrayList names = new ArrayList ( );
names.add("Billy");
names.add("Susan");

String aName = names.get(0);
```

  - What's wrong with this? (Hint, look at the signature for the get( ) method.)

The class Object

- The return type of the method get() is Object.
- Think of Object as Java's way of saying "any type".
- All classes in Java (including the ones we write) have an "is-a" relationship to Object. In other words:
  - every String is an Object
  - every Rectangle is an Object
  - every Vector is an Object
- The reverse is not, in general, true!

Making False Claims

- We can say…

```
Object someObject = new Song(. . .);
```

  - … because every Song is an Object.
- Going back to our example:

```
ArrayList names = new ArrayList ( );
names.add("Billy");
names.add("Susan");

String aName = names.get(0);
```

  - We are claiming that an Object is a String, which is not in general true!

Making Promises: Casting

- It looks like we're stuck. We can add things to the collection, but we can't really get them back out!
- The solution is to make a promise.
  - We know that we've only placed String objects into the collection, so we'll promise the compiler that the thing coming back out of the collection is actually a String:

```
String aName = (String)names.get(0);
```

  - This promise is called a *cast*.

Casting

- In general a cast looks like this:

```
(<class-name>)<expression>
```

  - For example:

```
String aName = (String)names.get(0);
```

- Casting does ***not*** change the type of the object: It is a promise that the object really is of another type.
- We can abuse casting, but will be caught at runtime:

```
Vector things = new Vector( );
things.add(new Rectangle( ));
Song aSong = (Song)things.get(0);       // Run time error!!
```


Building a CompactDisc class

- Let's build a compact disc class now.
- Let's say a compact disc object should understand the following messages:
  - get a song (by index)
  - add a song
  - get the title of the CD
  - get the total running time of the CD

A CompactDisc class in Java

- Try it yourself. What are the instance variables? Methods?