CITS1001 week 4
Grouping objects

Arran Stewart

March 20, 2018
In this lecture, we look at how can group objects together into *collections*.

Main concepts:
- The `ArrayList` collection
- Processing all items in a collection: the for-each loop

Reading: Chapter 4 (section 4.1 to 4.9) of *Objects First with Java - A Practical Introduction using BlueJ*, © David J. Barnes, Michael Kolling
The requirement to group objects

- Nearly every application involves grouping objects into collections in some way:
  - personal organizers
  - library catalogs
  - known prime numbers
  - student record systems
  - etc.
Features of groups

- Typically, for any sort of collection, we’ll want:
  - the ability to retrieve items
  - the ability to add and delete items
A journal of books I have read this year

- A record of books I have read this year
- Book titles may be added – and deleted
- There is no pre-defined limit to the number of books
- The journal will say how many book titles are stored in the collection
- It will list individual book titles
import java.util.ArrayList;

public class BooksReadJournal {
    // Storage for an arbitrary number of book titles
    private ArrayList<String> bookTitles;

    /**
     * Perform any initialization required for the organizer.
     */
    public BooksReadJournal() {
        bookTitles = new ArrayList<String>();
    }

    // ...
}
Using collections

- When declaring a field or variable that holds a collection, we specify:
  - the type of the collection: `ArrayList`
  - the type of the objects it will contain: `<String>`
- So, the whole declaration of the field is:
  - `private ArrayList<String> bookTitles;`
- We say we have an “ArrayList of String” or “ArrayList of Strings”
Generic classes

- `ArrayList<>` is an example of a *generic* or *parameterized* type
- What is it a list *of*? Anything we like – we can have lists of Shapes, Students
- Collections that can store different types of things in this way are known as parameterized or generic types
Generic classes (2)

- ArrayList implements list functionality:
  - we can add and remove items, ask for the size, etc.
- The type parameter says what we want a list of:
  - ArrayList<Person>
  - ArrayList<TicketMachine>
  - etc.
Creating an ArrayList object

- In versions of Java prior to version 7
  - `bookTitles = new ArrayList<String>();`
- Java 7 introduced ‘diamond notation’
  - `bookTitles = new ArrayList<>();`
- The type parameter can be inferred from the variable being assigned to
  - This is just a convenience for programmers – less typing
Exercises

- Write a declaration of a private field named `musicTracks` that can hold an `ArrayList`. The elements of the `ArrayList` are of type `MusicTrack`.
- Write a declaration of a local variable called `cits1001` that can hold an `ArrayList` of `Student`.
- Write assignments to the `musicTracks` and `cits1001` variables to create the appropriate `ArrayList` objects. Write them once using diamond notation and once without diamond notation, specifying the full type.
Object structures with collections

What does the object structure look like of a collection?

```
myMusic: MusicOrganizer
  files
    : ArrayList<String>
      : String
        "MorningBlues.mp3"
      : String
        "DontGo.mp3"
```
What does the object structure look like of a collection?
What about if we add another object to the collection?
Looping – repetition

Object structures with collections (2)
Features of the ArrayList collection

- It increases its capacity as necessary
- It keeps a private count:
  - This can be accessed `size()` accessor
- It keeps the objects in order
- Details of how all this is done are hidden
  - Does that matter?
  - Does not knowing how it works prevent us from using it?
Later, we will see collections with different features.

Sometimes we don’t really care about the *order* a collection of objects is kept in, just whether something is in the collection, or not.

For example:

- I have a collection of items in my shopping cart. Is “five-litre tub of ice-cream” in the collection, or not?
- I have a list of courses I’m enrolled in. Is “Particle Physics for Beginners” in the collection, or not?
public class BooksReadJournal {
    private ArrayList<String> bookTitles;

    public void addBook(String title) {
        bookTitles.add(title); // adding a new title
    }

    public int getNumberOfBooks() {
        return bookTitles.size(); // return the number of titles
    } // (delegation)

    // ...
}
Index numbering

- In an ArrayList, each object has some *position* in the list
- These positions start from 0.
Index numbering (2)

```
myMusic: MusicOrganizer
files

: ArrayList<String>
0 1 2

: String
"MorningBlues.mp3"

: String
"DontGo.mp3"

: String
"MatchBoxBlues.mp3"
```
Retrieving an object from an ArrayList

- We can retrieve individual objects using the `get()` method.
- It takes one parameter – the index of the object to retrieve.
public void printBookTitle(int index) {
    if(index >= 0 && index < bookTitles.size()) { // is index valid?
        String title = bookTitles.get(index); // retrieve object
        System.out.println(title);
    } else {
        // ... not a valid index. Display an error message
    }
}
Removing

- We can also remove items from an ArrayList.
- What happens when we remove an item in the middle of the list?
The general utility of indices

- Using integers to index collections has a general utility:
  - 'next' is index + 1
  - 'previous' is index - 1
  - 'last' is list.size() - 1
  - 'the first three' is the items at indices 0, 1, 2
- We could also think about accessing items in sequence: 0, 1, 2, ...
What about primitive types?

- We’ve seen ArrayLists that hold *objects* of various sorts. What if we want an ArrayList of *ints* or *bools*?
What about primitive types?

- We’ve seen ArrayLists that hold *objects* of various sorts. What if we want an ArrayList of *ints* or *bools*?
- Java’s collection classes can *only* hold objects...
What about primitive types?

- We’ve seen ArrayLists that hold *objects* of various sorts. What if we want an ArrayList of *ints* or *bools*?
- Java’s collection classes can *only* hold objects ... 
- but - for each primitive type, there’s an object equivalent
For `int`, there is an `Integer` class; for `bool`, there is the `Boolean` class; and so on.

If you want to make lists of numbers, you’ll have to use these – e.g.

```java
ArrayList<Integer> myListOfNumbers;
```
Class equivalents to primitive types (2)

- Why not use the class versions all the time, then?
- There is *overhead* in doing so - a plain `int` takes up less room in memory and can be calculated with more quickly
Exercises

- If a collection stores 10 objects, what value would be returned from a call to its `size` method?
- Write a method call using `get` to return the 5th object stored in a collection called `items`.
- What is the index of the last item stored in a collection of 15 objects?
- Write a method call to add the object held in the variable `favouriteTrack` to a collection called `tracks`.
- Write a method call to remove the 3rd object stored in a collection called `dates`.
Review

- Collections allow an arbitrary number of objects to be stored
- Class libraries usually contain tried-and-tested collection classes
- Java’s class libraries are called packages
- We have used the ArrayList class from the java.util package
Review

- Items may be added and removed
- Each item has an index
- Index values may change if items are removed (or further items added)
- The main ArrayList methods are add, get, remove, and size
- ArrayList is a parameterized or generic type
Looping – repetition
Loops

A loop can be used to execute a block of statements repeatedly without having to write them multiple times.
We often want to repeat some actions over and over
  - e.g. “do this action for each student in the university”
  - e.g. “do this action seventeen times”
  - e.g. “do this action until this condition is true”

Java loops provide us with a way to control how many times we repeat these actions

With collections, we often want to repeat things once for every object in a particular collection