Partially Filled Arrays
ArrayLists
Do-It-Yourself ArrayLists
Scope
Static

Lecture 18
Reusing some slides by Margaret Dulat and Joe Luk

Readings
This Week: Ch 8.3-8.8 and into Ch 9.1-9.3
(Ch 9.3-9.8 and Ch 11.1-11.3 in old 2nd ed)
(Reminder: Readings are absolutely vital for learning this stuff!)

Labs and Tutorials
This Week: Lab #8 – A catch-up lab…

Survey #2
■ Dr. Ben Yu’s second survey for you all is up on WebCT now.
■ Completely optional…
■ But, there’s a bit of extra credit for doing all three surveys by their respective deadlines!
■ (I think the deadline for this one is March 22, but check on WebCT to be sure.)

Learning Goals
By the end of class today you will be able to…
■ Read and write programs that use partially-filled arrays (in the typical, idiomatic manner).
■ Read and write programs that use Java’s ArrayList class.
■ Implement your own ArrayList class.
■ Figure out exactly which declarations (variables, methods) are visible from which parts of a program.
■ Explain what “static” means in Java programs and why you might want to use static methods and fields.

Teen Talk Barbie™ Reloaded
■ A few lectures back, we wrote a program that learns some phrases and prints them back at random (inspired by Mattel’s Teen Talk Barbie™ doll).
■ The goal was to have a fun example to introduce arrays.
■ But the programming style wasn’t very good.
■ Better to create a TalkingDoll class…
TalkingDoll Class
- Constructor that specifies name and maximum vocabulary size.
- Getters/Setters for the name.
- A method to add phrases to the doll.
- A method to get a random phrase back.

Partially Filled Arrays
- Arrays have a fixed size, but often we want to hold a variable number of items.
- A Very Common Solution:
  - Make an array bigger than you need.
  - Have an int variable to keep track of what you are actually using.

A Very Common Solution:
- Make an array bigger than you need.
- Have an int variable to keep track of what you are actually using.

**Partially Filled Arrays**
- Arrays have a fixed size, but often we want to hold a variable number of items.
- A Very Common Solution:
  - Make an array bigger than you need.
  - Have an int variable to keep track of what you are actually using.

Teen Talk Barbie™ Reloaded
- A few lectures back, we wrote a program that learns some phrases and prints them back at random (inspired by Mattel’s Teen Talk Barbie™ doll).
  - The goal was to have a fun example to introduce arrays.
  - But the programming style wasn’t very good.
  - Better to create a TalkingDoll class...
**TalkingDoll Class**
- Constructor that specifies name and maximum vocabulary size.
- Getters/Setters for the name.
- A method to add phrases to the doll.
- A method to get a random phrase back.

**Java’s ArrayList Class**
- What if you don’t know (or want to force) a fixed maximum size in advance?
- Java has an ArrayList class for this case:
  - import java.util.ArrayList
  - Declare:
    ArrayList<String> phrases = new ArrayList<String>();

**Java’s ArrayList Class**
- What if you don’t know (or want to force) a fixed maximum size in advance?
- Java has an ArrayList class for this case:
  - import java.util.ArrayList
  - Declare:
    ArrayList<type> phrases = new ArrayList<type>();
- Methods:
  - int size()
  - void add(<type> newValue)
  - <type> get(int index)
  - void set(int index, <type> newValue)
- <type> can be any object type. See Ch 7.2.

**TalkingDoll with ArrayLists**
- Let’s redo the TalkingDoll class using ArrayLists…

**Do-It-Yourself ArrayLists**
- ArrayLists are nothing magical!
  - (OK, the generic <type> stuff is kind of magic.)
- It’s just a class. If we fix the type of the elements (e.g., have an ArrayList of String), you know enough to write your own version.
- But how do you allow arrays to grow?

**Real-Life Analogy: Moving Homes**
- A house (or condo, apartment, etc.) has a fixed size. What happens when your family grows and you need more space?
Real-Life Analogy: Moving Homes
- A house (or condo, apartment, etc.) has a fixed size. What happens when your family grows and you need more space?
- Answer: You buy a bigger place, and then you pack up and move all your stuff to the new place, and get rid of your old home.

Making Your Own ArrayList
- An array has a fixed size. What happens when your list grows and you need more space?
- Answer: You allocate a bigger array, and then you pack up and move all your stuff to the new array, and get rid of your old array.

```plaintext
Making Your Own ArrayList
Answer: You allocate a bigger array, and then you pack up and move all your stuff to the new array, and get rid of your old array.

```

```plaintext
Making Your Own ArrayList
Answer: You allocate a bigger array, and then you pack up and move all your stuff to the new array, and get rid of your old array.

```

```plaintext
Making Your Own ArrayList
Answer: You allocate a bigger array, and then you pack up and move all your stuff to the new array, and get rid of your old array.

```

```plaintext
Making Your Own ArrayList
Answer: You allocate a bigger array, and then you pack up and move all your stuff to the new array, and get rid of your old array.

```
Making Your Own ArrayList
- Answer: You allocate a bigger array, and then you pack up and move all your stuff to the new array, and get rid of your old array.

```
aCount 4
a 3 1 4 1
a.length 4
newA 3 1 4 1
newA.length 8
```

Making Your Own ArrayList
- Answer: You allocate a bigger array, and then you pack up and move all your stuff to the new array, and get rid of your old array.

```
aCount 4
a 3 1 4 1
a.length 4
newA 3 1 4 1
newA.length 8
```

Making Your Own ArrayList
- Answer: You allocate a bigger array, and then you pack up and move all your stuff to the new array, and get rid of your old array.

```
aCount 4
a 3 1 4 1
a.length 4
newA 3 1 4 1
newA.length 8
```

Making Your Own ArrayList
- Answer: You allocate a bigger array, and then you pack up and move all your stuff to the new array, and get rid of your old array.

```
aCount 4
a 3 1 4 1
a.length 4
newA 3 1 4 1
newA.length 8
```

(garbage collection)

MyStringArrayList
- Let's create a MyStringArrayList class.
- Methods:
  int size()
  void add(String newString)
  String get(int index)
Variable Scope
- Scope tells you which declarations you can see from which points in the program.
- The scope of a variable is the places in the program where the variable can be accessed.
- Or you can think of scope as the places where you can see some variable.
- (Think “scope” meaning “extent”, as in “the scope of my legal authority” or “the scope of the inquest”. Not “scope” as in “telescope” or “scoping out the beach”).

Variable Scope
- The world is broken up into packages.
  - `import` lets you see `public` stuff from other packages.
  - Or you can give full name, e.g., `java.util.Scanner`
  - Other than that, can’t see inside other packages
Example:
  ```java
  import java.util.Random;
  Random foo = new Random();
  ```

Variable Scope
- Packages contain classes
  - Within a package, everyone can see all classes and non-private stuff within classes
Example:
  ```java
  TalkingDoll a = new TalkingDoll();
  System.out.println(a.getName());
  System.out.println(a.name);
  ```
  The variable `name` is private in the TalkingDoll class!

Variable Scope
- Classes contain methods and fields
  - All methods can see other methods and fields in same class
Example:
  ```java
  public void setLabScore(int labNum, double score) {
      labScores[labNum-1] = score;
      computeTotalLabScore();
  }
  ```

Variable Scope
- Methods contain statements, executed one-by-one
  - You can see declarations that happened already
  - Parameters are like declarations that happen at start of method.
  - Curly braces define a compound statement, which limits scope
Example:
  ```java
  public static void main(String[] args) { 
      int a = 0;
      System.out.println(args[0]);
      int b = 2*a;
      int a = 3; // Not OK!
      b = c+a; // Not OK! -- can’t see c
      int c;
  }
  ```
Variable Scope

- Compound statement limits scope
  - You can see out, but not in.
  Example:
  ```java
  public static void main(String[] args) {
  int a = 0;
  int b = 0;
  { // start of a block
  int a = 2; // Not OK! a already declared
  int c = 2;
  System.out.println(c); // prints 2
  }
  System.out.println(a); // prints 0
  System.out.println(c); // Not OK!
  int c = 2; // OK
  }
  ```

Variable Scope: Shadowing

- Local variable declarations wins over instance variable declarations.
  Example:
  ```java
  private static int a = 0;
  public static void main(String[] args) {
  int b = 0;
  { // start of a block
  int a = 2; // OK! Shadows other a
  int c = a;
  System.out.println(c); // prints 2
  }
  System.out.println(a); // prints 0
  System.out.println(c); // Not OK!
  int c = 2; // OK
  }
  ```

Variable Scope -- Summary

- The world is broken up into packages.
  - import lets you see public stuff from other packages.
  - Or you can give full name, e.g., java.util.Scanner
- Packages contain classes
  - Within a package, everyone can see all classes and non-private stuff within classes
- Classes contain methods and fields
  - All methods can see other methods and fields in same class
  - Methods contain statements, executed one-by-one
  - You can see declarations that happened already
  - Parameters are like declarations that happen at start of method.
  - Curly braces define a compound statement, which limits scope
  - Compound statement limits scope
    - You can see out, but not in.
    - for loop is special: variables declared in header are visible only inside the loop

Questions

The Meaning of static

- From our very first Java program, we had:
  ```java
  public static void main (String[] args)
  ```
- But what does static mean?
Review: Static Methods
A method that is tagged as static:
- Can be called on the class, rather than on an object of the class (e.g., Math.sin(1.0))
- Does not have an implicit parameter this
- Cannot directly access non-static methods or fields (because there is no this!)

The Meaning of static
- But what does static mean?
- In general, static means something that belongs to the class as a whole, and not to a specific object:
  - Static methods have no implicit parameter this
  - Static fields belong to the whole class, shared by all objects.

The Meaning of static
- But what does static mean?
- In general, static means something that belongs to the class as a whole, and not to a specific object:
  - Static methods have no implicit parameter this
  - Static fields belong to the whole class, shared by all objects.

Java Classes
- When you write a class in Java, you describe what goes into each object of the class:
  ```java
  public class UBCStudent {
      String name;
      int studentID;
      public String getName() { }
      ...
  }
  ``
  That says that each object of type UBCStudent contains a field name, and a field studentID, and the object knows how to do the getName method (applied to itself).

Analogy/Intuition on static
- A class is a pattern/blueprint/factory that shows, by example, how to create specific types of objects:
  - Each object has these instance fields...
  - Each object has these methods, that do the following statements...
  - Sometimes, you want to talk about the pattern/blueprint itself, not the objects. Those are when you want to use static

Real Life Analogy of static Fields
- A factory/blueprint for a car is like the class. The cars are the objects.
  - The blueprint shows an engine, battery, etc.
  - Each car gets its own engine, battery, etc.
    These are instance fields.
  - But to give each car a serial number, you need a field that lives in the factory. That should be a static field.
Example: UBCStudent with Automatically Assigned Student ID Numbers

- Each object should get a unique student ID number assigned to it.
- We’ll keep a variable `nextID` that keeps track of the next available ID number.
- `nextID` is one variable for the whole class, so it should be static!

Recap: Static Fields

A field that is tagged as `static`:

- Is a single field of the class, rather than a separate field of each object of the class
- Is shared by all objects of the class
- Can be accessed without creating an object of the class
- Can be accessed from static methods

Why Use static?

- Static methods are almost never used, with two common exceptions:
  - Numbers aren’t objects in Java, so to have a class of methods that do computation, we’ll often put them in static methods (e.g., the Math class).
  - The main method starts running before there are any objects created, so it has to be static.

Recap: Variable Types

- Static variables
  - declared within class
  - associated with class, not instance, indefinite lifetime
- Instance variables
  - declared within class
  - associated with instance
  - accessible throughout object, lifetime of object
- Local variables
  - declared within method
  - accessible within method after declared, lifetime of method
  - (to be more precise, replace “method” by “block”)
- Parameters
  - declared in parameter list of method
  - accessible throughout method, lifetime of method

Questions
Midterm Info

- Out of 100: raw avg=56, low=1, high=97
  - Will be scaled (info later)
- Long exam, but not unreasonable:
  - Everyone should be able to do it (with time).
  - Talk to friends, learning centre, WebCT, TAs, prof.
  - See Lecture 17 notes for tips from Beth Simon...
- Solutions will be posted.
- Protocol for re-mark requests:
  - read solutions first, carefully
  - no re-mark requests accepted until Thursday
  - re-mark requests must be in writing (paper attached to exam) and submitted to instructor
  - entire exam re-marked

Midterm #1

- Raw Score Stats:
  - Average: 56, Low: 1, High: 97 (Out of 100)
  - I’ve made ArrayList question extra credit (3pts).
- Scaling Formula:
  \[(\text{raw} / 97)^{0.7} \times 100\]

<table>
<thead>
<tr>
<th>Raw Score</th>
<th>Scaled Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>27</td>
</tr>
<tr>
<td>30</td>
<td>44</td>
</tr>
<tr>
<td>45</td>
<td>58</td>
</tr>
<tr>
<td>60</td>
<td>71</td>
</tr>
<tr>
<td>75</td>
<td>84</td>
</tr>
<tr>
<td>90</td>
<td>95</td>
</tr>
</tbody>
</table>