More on Arrays and Array-Like Objects

CS 3: Computer Programming in Java
Objectives

- Introduce ArrayLists
- Get deeper with two-dimensional arrays
- Cover more GUI concepts
The Java class `ArrayList` (`java.util.ArrayList`) is a fast and easy to use object class representing a one-dimensional array.

- The `ArrayList` stores an array of references to `Object`s, therefore, primitives must be placed in a `Wrapper` class object before they can be stored in an `ArrayList` object.
- The `ArrayList`, unlike the array, will allow a collection of different type objects to be stored in one list.
- When this is done, it is important for the user to know exactly what type of object is to be stored or retrieved from the `ArrayList`.
ArrayList Methods

- `add(Object o)`: puts reference to object into ArrayList
- `get(int index)`: retrieves object reference from ArrayList index position
- `size()`: returns ArrayList size
- `remove(int index)`: removes the element at the specified position in this list; shifts any subsequent elements to the left and returns the element that was removed from the list
- `indexOf(Object o)`: finds the index in this list of the first occurrence of the specified element
- `clear()`: removes all of the elements
The following example illustrates how an ArrayList is declared and used. It does not require the use of brackets [ ] like an array. Its members are added, accessed, and retrieved with the methods of the class.

```java
import java.util.ArrayList;

public class Beatles2
{
    // Stores and modifies a list of band members.
    public static void main (String[] args)
    {
        ArrayList band = new ArrayList();
        ChemicalElement element = new ChemicalElement("Hydrogen","H",1);
    }
}
```
ArrayList Example (2)

```java
band.add("Paul");
band.add("Pete");
band.add("John");
band.add("George");
band.add(element);

System.out.println(band);

int location = band.indexOf("Pete");
band.remove(location);

location = band.indexOf(element);
System.out.println(location);
```
System.out.println (band);
System.out.println ("At index 1: " + band.get(1));

ChemicalElement element2 = (ChemicalElement)band.get(3);
System.out.println (element2);

band.add (2, "Ringo");

System.out.println (band);
System.out.println ("Size of the band: " + band.size());
}
run:

[Paul, Pete, John, George, The element name is Hydrogen
Its atomic symbol is H
Its atomic number is 1]
3

[Paul, John, George, The element name is Hydrogen
Its atomic symbol is H
Its atomic number is 1]
At index 1: John
ArrayList Example Output (2)

The element name is Hydrogen
Its atomic symbol is H
Its atomic number is 1
[Paul, John, Ringo, George, The element name is Hydrogen
Its atomic symbol is H
Its atomic number is 1]
Size of the band: 5
Restricting ArrayList

- If you want to restrict an ArrayList to a specific type of object, you must put the object type inside <> brackets.
- The following would declare an ArrayList restricted to String objects:

  ```java
  ArrayList<String> names = new ArrayList<String>();
  ```
Two-Dimensional Arrays

- Java supports multi-dimensional arrays: 1-dimensional, 2-dimensional, 3-dimensional, . . . , up to as many as memory space will allow.
- Two-dimensional arrays are used whenever the model data is best represented with rows and columns.
- When we have an array to define, it is very useful to define constants for the number of rows and columns:

  ```java
  final int ROWS = 2;
  final int COLS = 3;
  int[][][] board = new int[ROWS][COLS];
  ```
Two-Dimensional Arrays (2)

- We can initialize the array as it is declared, placing the values for each row inside its own set of braces:

```java
int[][] scores = {
    {3, 4, 5, 2, 1, 4, 3, 2, 4, 4}, // representing the ratings of 10
    {2, 4, 3, 4, 3, 2, 1, 2, 2}, // people (cols) who rated 4
    {3, 5, 4, 5, 3, 2, 5, 5, 5}, // different sodas (rows)
    {1, 1, 1, 3, 1, 2, 1, 3, 2, 4}
};
```

- The above declares an array of 4 rows with 10 columns per row
- The variable `scores` is a reference to a list of 4 references to arrays of 10 ints
- The variable `scores[0]` is a reference to the array of 10 ints in row 0
Two-Dimensional Arrays Example

- Two-dimensional arrays are almost always processed with nested for loops
- The following example computes averages for each person (row) and each soda (col):

```java
public class SodaSurvey {
    // -----------------------------------------------------------------
    // Determines and prints the average of each row (soda) and each
    // column (respondent) of the survey scores.
    // -----------------------------------------------------------------
    public static void main (String[] args)
    {
        int[][] scores = { {3, 4, 5, 2, 1, 4, 3, 2, 4, 4},
                          {2, 4, 3, 4, 3, 3, 2, 1, 2, 2},
                          {3, 5, 4, 5, 5, 3, 2, 5, 5, 5},
                          {1, 1, 1, 3, 1, 2, 1, 3, 2, 4} };
    }
```
Two-Dimensional Arrays Example (2)

```java
final int SODAS = scores.length;
final int PEOPLE = scores[0].length;

int[] sodaSum = new int[SODAS];
int[] personSum = new int[PEOPLE];

for (int soda=0; soda < SODAS; soda++)
    for (int person=0; person < PEOPLE; person++)
    {
        sodaSum[soda] += scores[soda][person];
        personSum[person] += scores[soda][person];
    }
```
Two-Dimensional Arrays Example (3)

DecimalFormat fmt = new DecimalFormat("0.#");
System.out.println("Averages:
");

for (int soda=0; soda < SODAS; soda++)
    System.out.println("Soda #" + (soda+1) + ": " +
        fmt.format((float)sodaSum[soda]/PEOPLE));

System.out.println();
for (int person =0; person < PEOPLE; person++)
    System.out.println("Person #" + (person+1) + ": " +
        fmt.format((float)personSum[person]/SODAS));
}
Two-Dimensional Arrays Example Output

run:
Averages:

Soda #1: 3.2
Soda #2: 2.6
Soda #3: 4.2
Soda #4: 1.9

Person #1: 2.2
Person #2: 3.5
Person #3: 3.2
Person #4: 3.5
Person #5: 2.5
Person #6: 3
Person #7: 2
Person #8: 2.8
Person #9: 3.2
Person #10: 3.8
More GUI Concepts

- Layout manager
- Paint component
- Mouse events
- Key events
A layout manager is an object that implements the LayoutManager interface and determines the size and position of the components within a container.

As a rule, the only containers whose layout managers you need to worry about are JPanels and content panes.

Each JPanel object is initialized to use a FlowLayout, unless you specify differently when creating the JPanel.

Content panes use BorderLayout by default.

If you do not like the default layout manager that a panel or content pane uses, you are free to change it to a different one.
Layout Manager Types

- FlowLayout
- GridLayout
- BorderLayout
- BoxLayout
FlowLayout

- The FlowLayout class puts components in a row, sized at their preferred size.
- If the horizontal space in the container is too small to put all the components in one row, the FlowLayout class uses multiple rows.
- If the container is wider than necessary for a row of components, the row is, by default, centered horizontally within the container.
- We would use the following statement in the constructor of the JPanel:
  
  ```java
  setLayout (new FlowLayout());
  ```
GridLayout

- A GridLayout object places components in a grid(row,col) of cells
- The grid is filled row by row as the components are added to the panel
- Each component takes all the available space within its cell, and each cell is exactly the same size
- If the window is resized, the GridLayout object changes the cell size so that the cells are as large as possible, given the space available to the container
- We would use:
  ```java
gLayout (new GridLayout (2, 3));
```
A BorderLayout object places components in the four major compass directions (North, South, East, and West) and in the Center of the Panel.

The Center receives the most space available after the 4 compass directions have been filled.

We would use:

```java
setLayout (new BorderLayout());
add (b1, BorderLayout.CENTER);
add (b2, BorderLayout.NORTH);
add (b3, BorderLayout.SOUTH);
add (b4, BorderLayout.EAST);
add (b5, BorderLayout.WEST);
```
BoxLayout

- BoxLayout either stacks its components on top of each other or places them in a row – your choice
- You might think of it as a version of FlowLayout, but with greater functionality
- Each component controlled by a box layout butts up against its neighboring components
- If you want to have space between components, you can create invisible components with the help of the Box class
- The Box class provides convenience methods to help you create common kinds of filler
BoxLayout (2)

- **RIGID AREA**: Use this when you want a fixed-size space between two components
- **GLUE**: Use this to specify where excess space in a layout should go
  - Think of it as a kind of elastic glue — stretchy and expandable, yet taking up no space unless you pull apart the components that it is sticking to
  - For example, by putting horizontal glue between two components in a left-to-right box, you make any extra space go between those components, instead of to the right of all the components
- **We would use:**
  ```java
  setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
  add (b1);
  add (Box.createRigidArea (new Dimension (0, 10)));
  add (b2);
  add (Box.createVerticalGlue());
  add (b3);
  add (b4);
  ```
The 'paintComponent' method handles the user’s needs for placing graphics content into a panel.

When you extend a JComponent or JPanel to draw graphics, override the 'paintComponent()' method.

This method is called because the user did something with the user interface that required redrawing, or your code has explicitly requested that it be redrawn.
Paint Component Calls

- AUTOMATIC CALL: when a window becomes visible (uncovered or unminimized) or is resized, the "system" automatically calls the `paintComponent()` method for all areas of the screen that have to be redrawn.

- LISTENER CALL: when a listener (mouse, button, keyboard, ...) of yours is called, the listener code often makes changes that should be displayed in your graphics area.
  - The listener calls the method indirectly using the method `repaint()`.
  - Never call `paintComponent()` method directly.
  - The listener code should set instance variables that `paintComponent()` uses in drawing the panel.
  - After changing the values, the next time `paintComponent()` is called, these new values will be used.
  - But, if you don’t want to wait for a call to `paintComponent()`, call `repaint()`.
Mouse Events

- Mouse events notify when the user uses the mouse (or similar input device) to interact with a component.
- Mouse events occur when the cursor enters or exits a component's onscreen area and when the user presses or releases one of the mouse buttons.
NOTE: A mouseClick is when the mouse is pressed and released without any significant movement of the mouse; thus a mouseClick event also generates both mousePressed and mouseReleased events.
Key Events

- Key events indicate when the user is typing at the keyboard.
- Specifically, key events are fired by the component with the keyboard when the user presses or releases keyboard keys.
- Two kinds of events
  - Key-typed event
  - Key-pressed or key-released event
Key Events (2)

- In general, you react to only key-typed events unless you need to know when the user presses keys that do not correspond to characters.
- For example, to know when the user types a Unicode character — whether by pressing one key such as 'a' or by pressing several keys in sequence — you handle key-typed events.
- On the other hand, to know when the user presses the F1 key, or whether the user pressed the '3' key on the number pad, you handle key-pressed events.
Summary

- Looked at the ArrayList class and its functions, including:
  - add
  - get
  - size
  - remove
  - indexOf
  - clear
Dove into two-dimensional arrays

- Best to use when representing a table with rows and columns
  - Define constants in advance for those rows and columns
- Can initialize the array upon declaration
  - Can do so by placing values within its own braces
Summary (3)

- Covered more GUI concepts, including
  - Layout manager
    - FlowLayout
    - GridLayout
    - BorderLayout
    - BoxLayout
  - Paint component
  - Mouse events
  - Key events