Objectives

- Start with declarations of arrays
- Provide an initialization list, as well as look into implicit array sizing
- Talk about array data and methods
- Discuss arrays of user-defined objects
- Introduce another set of classes into the mix: JRadioButton, ButtonGroup, and JFrame
Starting with Arrays

- An array is a collection of components (called array elements), all of the same data type, given a single name, and stored in adjacent memory locations.
- The individual components are accessed by using the array name together with an integral valued index in square brackets.
- The index indicates the position of the component within the collection.
Single Line Array Declarations in Java

- **Format**
  ```java
  DataType[ ] Array_Name = new DataType [LengthExpression];
  OR
  DataType Array_Name [ ] = new DataType [LengthExpression];
  ```

- **Note** that in array declaration, the length expression in the square bracket can be any integer type expression except the long data type.

- The array length expression must evaluate to a specific positive value at run time.
Examples of Array Declarations in Java

- The following array declarations below are legal

  ```java
  int[] my_int = new int[100];
  int[] your_int = new int[‘A’];
  
  Here, the array length will be 65 since compiler will take the ASCII value of the character ‘A’, which is 65
  
  int[] their_int = new int[“Hello Dad”.charAt(0) * 2];
  
  Here, the method charAt will return a character ‘H’, whose ASCII value is 72, therefore the their_int array would have a length of 144
  ```
The following are the requirements for the LengthExpression for an array:

- Array length expression must evaluate to a concrete positive integer at run time.
- It cannot be a long data type.
- The array length expression should not evaluate to a negative value.
- A Zero length array will have no elements in it.
- The parameters in the array length expressions can be named or literal constants, initialized variables, or values returned from method calls.
Initializing Arrays and Implicit Array Sizing

Java allows one to define an array and provide the values for its elements in one declaration line, using the following syntax:

```
DataType [ ] Arr = { value1, value2, value3, value4};
```

For an int type array the above syntax could boil down to the following statement type:

```
int [ ] arr = { 5, -8, 11, 22, 89};
```

NOTE: No length is specified when followed by initialization list
 Initializing Arrays and Implicit Array Sizing

- This declaration...
  ```java
  int [ ] arr = { 5, -8, 11, 22, 89};
  ```

- ...is equivalent to the following declaration
  ```java
  int [ ] arr = new int [5];
  arr [0] = 5;
  arr [1] = -8;
  arr [2] = 11;
  arr [3] = 22;
  arr [4] = 89;
  ```
Data and Method Members for Java Array Reference Types

- The Java array reference type, in addition to methods from the Iterator interface, by default carries a public length member with it, which stores the length of the array.

- The statement:
  ```java
  System.out.println(arr.length);
  ```
  - Would output the value 5

- The for loop can be written in the form:
  ```java
  for (Object type var : Iterator Object)
  {
    ...
  }
  ```
  - Where the first parameter is a variable of the type of Object, and the second parameter is an iterator Object that is a collection of the same type of Objects.
  - This version of the for loop is sometimes referred to as the foreach loop.
The following is sample program that illustrates the properties of an array and example of a special form of the for loop in Java that can be used with objects that implement the Iterator interface

public class Primes
{
    // Stores some prime numbers in arrays and prints them.
    public static void main (String[] args)
    {
        int[] primeNums = {2, 3, 5, 7, 11, 13, 17, 19};
        int[] clone_arr = (int[]) primeNums.clone();
Java Array Example (2)

// Clone the array primeNums and returns a reference to it of type
// java.lang.Object, then casts the reference java.lang.Object
// to int[].

System.out.println("Array length: " + primeNums.length);
System.out.println("The first array of prime numbers are:");
for (int prime : primeNums) {
    System.out.print(prime + " ");
}
System.out.println(" ");
System.out.println("The clone array of prime numbers are:");
for (int clone : clone_arr) {
    System.out.print(clone + " ");
}
}
This produces the following output:

run:

Array length: 8

The first array of prime numbers are:
2 3 5 7 11 13 17 19

The clone array of prime numbers are:
2 3 5 7 11 13 17 19
Array of User-Defined Objects

- Referring back to the ChemicalElement objects in the previous lecture, if we want to establish an array of 10 ChemicalElement objects, we would code:

  `ChemicalElement[] eList = new ChemicalElement[10];`

- The reference eList will point to a list of 10 references for objects of ChemicalElement type, but no objects have been instantiated.
The JRadioButton and ButtonGroup are GUI components which are placed into a JPanel so that they can be used to prompt the user to make a choice between mutually exclusive options (when one is chosen any previous choice is removed).

To determine which button in the group is chosen, a listener object must be assigned to the radio buttons.
When a radio button is selected it creates a runtime event called an `ActionEvent`, which generates an `ActionEvent` object carrying the identity of the component that produced the event.

The program can create a listener object of type `ActionListener`, which will receive the `ActionEvent` objects and test them for their source in order to determine what actions must be taken.
The following class creates a JRadioButton group of three radio buttons, places it into a JPanel, and sets up a listener to process the ChemicalElement objects in the program.

```java
class EOptionsPanel extends JPanel {
    private JLabel prompt;
    private JRadioButton one, two, three;
    // ...
}
```

**EOptionsPanel.java**

Author: Edwin Ambrosio

Demosntrates the use of radio buttons to capture input options. and listener to call methods to process the options.

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

```java
public class EOptionsPanel extends JPanel {
    // ...
}
```
JRadioButton and ButtonGroup Example (2)

```java
// -------------------------------------------------------------------
// Sets up a panel with a label and a set of radio buttons
// that present options to the user.
// -------------------------------------------------------------------
public EOptionsPanel()
{
    prompt = new JLabel("Choose your input option?");
    prompt.setFont(new Font("Helvetica", Font.BOLD, 24));

    one = new JRadioButton("input element");
    one.setBackground(Color.green);
    two = new JRadioButton("list all elements");
    two.setBackground(Color.green);
    three = new JRadioButton("show element count");
    three.setBackground(Color.green);
}```
JRadioButton and ButtonGroup Example (3)

```java
ButtonGroup group = new ButtonGroup();
group.add(one);
group.add(two);
group.add(three);

EOptionListener listener = new EOptionListener();
one.addActionListener(listener);
two.addActionListener(listener);
three.addActionListener(listener);
```
JRadioButton and ButtonGroup Example (4)

// add the components to the JPanel
add (prompt);
add (one);
add (two);
add (three);

setBackground (Color.green);
setPreferredSize (new Dimension(400, 100));
}
JRadioButton and ButtonGroup Example

private class EOptionListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        Object source = event.getSource();
        if (source == one) {
            MyElements2.inputElements();
        }
    }
}
else
    if (source == two)
        MyElements2.listElements();
    else
    {
        String message;
        message = "The number of elements processed was " + ChemicalElement.getNumOfElements();

        JOptionPane.showMessageDialog(null, message);
    }
}
}
Closer Look Into the JRadioButton and ButtonGroup Example

- The code follows the steps
  - Create the radio buttons
  - Add the radio buttons to the button group (this is what makes them mutually exclusive)
  - Add the listener to the radio buttons
  - Add the radio buttons to the Jpanel
  - Declare the private “inner” class for the listener object
    - It calls the helper methods of the Main class when it detects which one of the radio buttons has been selected
**JFrame Example**

- The following Main Class creates a JFrame places the JPanel into it and displays it
- It allows the user to process an array of ChemicalElement objects in an event-driven application
- Notice how the main method has very little logic because the processing is done in helper functions called by the listener
- The main ends when the JFrame is closed by the user
- Otherwise, whatever the user chooses from the radio button group drives the program
import javax.swing.JOptionPane;
import javax.swing.JFrame;
public class MyElements2
{
    public static JFrame frame;
    public static ChemicalElement[] eList;
public static void main (String[] args) {
    ChemicalElement[] eList = new ChemicalElement[10];
    String message;

    JFrame frame = new JFrame("Chemical Elements Options");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    EOptionsPanel panel = new EOptionsPanel();
    frame.getContentPane().add(panel);

    frame.pack();
    frame.setVisible(true);
}
public static void listElements()
{
    String message, ntabs;
    int num;
    System.out.println("\nNumber"+"\t"+"Name"+"\t"+"Symbol"+"\t"+"AtomicNumber"+"\n");
    for ( num=0; num < ChemicalElement.getNumOfElements();num++ )
    {
        if (eList[num].getName().length() < 8)
            ntabs="\t"
        else
            ntabs="\t"
        message = num + "\t"+eList[num].getName()+ntabs+eList[num].getSymbol()+
            "\t"+eList[num].getNumber();
        System.out.println(message);
    }
}
public static void inputElements()
{
    String name, symbol, numStr, message;
    int number, again, index;
    frame.setVisible(false);
    do
    {
        name = JOptionPane.showInputDialog("Enter the Element name: ");
        symbol = JOptionPane.showInputDialog("Enter the Element symbol: ");
        numStr = JOptionPane.showInputDialog("Enter the Element number: ");
        number = Integer.parseInt(numStr);
    }
index = ChemicalElement.getNumOfElements();
eList[index] = new ChemicalElement(name,symbol,number);

message = eList[index].toString();
JOptionPane.showMessageDialog (null, message);
again = JOptionPane.showConfirmDialog (null, "Input Another?");
}
while (again == JOptionPane.YES_OPTION);
frame.setVisible(true);
}
public  String toString()
{
    String message;
    message = "The element name is "+nameOfElement+"\n" +
    "Its atomic symbol is "+chemicalSymbol+"\n" +
    "Its atomic number is "+atomicNumber;
    return message;
}
Summary

- Declaration of arrays format
  
  ```
  DataType[ ] Array_Name = new DataType [LengthExpression];
  OR
  DataType Array_Name [ ] = new DataType [LengthExpression];
  ```

- Format for array initialization, as well as implicit array sizing
  
  ```
  DataType [ ] Arr = { value1, value2, value3, value4};
  ```
Talked about array data and methods

Discussed arrays of user-defined objects

```java
 ChemicalElement[] eList = new ChemicalElement[10];
```

Introduced the following classes:

- JRadioButton
- ButtonGroup
- JFrame