Lecture 4

Strings, Arrays, Swing & File IO
notes about slides

• I will NOT be including code samples in slides very often from now on
• I will NOT be including screen shots from the netbeans IDE very often from now on
• to understand these slides you should look at the code files on the class website
• you should load and run the code files to understand the lecture slides
• you should write your own code to understand these topics
String Class

• import java.lang.*; //automatically imported
• Stores strings of text in an object
• Methods:
  - String(char[]);
  - String(StringBuffer myStringBuffer);
  - toString();
  - compareTo(String); compareToIgnoreCase();
  - getBytes(); //returns an array of bytes
  - valueOf(); //converts primitive types to a string
String Class (2)

- Java overloads the ‘+’ operator for strings - concatenation operator
- Strings are **immutable**. After you create them you cannot change them. You need to allocate a new string to hold any changes.
- String class is **final**
- Java shares a single string in memory with multiple references if they are equivalent strings equality and object identity will both give the same results
  - if (string1 == string 2)
  - if (string1.compareTo(string2)) //these have same result
String Class Demos

- Lecture4.* package
- StringDemo
  - Shows auto allocation by Java of strings.
  - Shows overloading of the ‘+’ for strings.
  - Shows using an explicit new String().
- StringDemo2
  - Shows that strings are immutable.
  - Shows that references can point to the same string object
  - Show that because Java stores only one copy of a given string even if multiple copies are created the == and equals() work the same for strings (not the way it works for most other classes)
- StringDemo3
  - Shows concatenation of strings ends up allocating a new string and if two references originally pointed to the same string after concatenation they now point to different objects. A possible bug source.
StringBuffer

• Holds a string but is mutable (changeable)

• methods
  - StringBuffer(); //creates a 16 char buffer
  - StringBuffer(String); //loads the buffer with a copy of the characters in String
  - append(String); //adds the chars to the end
  - length(); //returns length in chars of buffer
  - delete(int start, int end); //remove this strip of characters from the StringBuffer (maybe all)
  - toString(); //returns a new string object
StringBufferDemo class

- StringBufferDemo
  - This shows that string buffers are mutable. It shows that they are not automatically allocated when doing `append()` - unlike `concatenate` in `String`
  - Shows that a `StringBuffer` will automatically grow if `append()` adds more characters that the buffer is sized to.
Arrays

- In Java they are objects!
- **syntax** `Class[] nameOfArrayObject;`
  - `String[] myArrayOfCountries;`
  - `Object[] myArrayOfObjects;`
  - `int[] myArrayOfInts;`

- Doesn’t allocate objects only the reference to an array of references to objects of that type. You need to allocate the array object and then the objects reference by the array elements.
- Indexed starting at 0
- Can get the length `length;` variable
Arrays (2)

• Doesn’t allocate objects only the array of references to objects of that type.
  - `Object[] myArrayOfObjects = Object [10];`
  - `int[] myArrayOfInts = new int [100];`

• You must then allocate the objects referenced by the elements in the array
  - `myArrayOfObjects[8] = new String(“Buon Giorno”);`
Array in memory

the reference to the array object
the array object
the ints referenced by the array object
Array Demos

- **ArrayDemo**
  - Shows allocation of an array and then allocation of the objects referenced by the elements of the array and shows arrays have bounds and not autoreallocated

- **ArrayDemo2**
  - Shows use of base class and derived class arrays
  - Shows use of Integer wrapper class for int primitive
  - Shows getting the class of an object in n array
  - Shows that arrays can contain a mixture of object types in the same array
Wrapper Classes

• all primitive types have associated wrapper classes
  – Integer and int
  – Long and long
  – Double and double
• Wrap a single instance of a primitive
• Allow call by reference with primatives
• Class methods
Wrapper class methods

byte byteValue()  
Returns the value of this Integer as a byte.

int compareTo(Integer another)  
Compares two Integer objects numerically.

int compareTo(Object o)  
Compares this Integer object to another object.

static Integer decode(String nm)  
Decodes a String into an Integer.

double doubleValue()  
Returns the value of this Integer as a double.

boolean equals(Object obj)  
Compares this object to the specified object.

float floatValue()  
Returns the value of this Integer as a float.

int intValue()  
Returns the value of this Integer as an int.

long longValue()  
Returns the value of this Integer as a long.

static int parseInt(String s)  
Parses the string argument as a signed decimal integer.

short shortValue()  
Returns the value of this Integer as a short.

String toString()  
Returns a String object representing this Integer's value.

static String toString(int i)  
Returns a String object representing the specified integer.

static Integer valueOf(String s)  
Returns an Integer object holding the value of the String.
Use the online (or download) API javadoc
http://java.sun.com/j2se/1.3/docs/api/
File class

- package java.io;
- Abstract platform independent representation of a file, directory, path, ...
  - File(String pathAndFileName);
  - createNewFile();
  - delete();
  - exists(); //tests if a file exists
  - isDirectory(); isFile();
  - getPath(); length();

- Everything but reading and writing!
FileReader extends InputStream Class

• Allow reading of streams of characters from a file. (not bytes, chars).
  • FileReader(File file);
    – Creates a new FileReader, given the File to read from.
  • FileReader(FileDescriptor fd);
    – Creates a new FileReader, given the FileDescriptor to read from.
  • FileReader(String fileName);
    – Creates a new FileReader, given the name of the file to read from.
  • int read();
    – Read a single character.
  • int read(char[] cbuf, int offset, int length);
    – Read characters into a portion of an array.
  • boolean ready();
    – Tell whether this stream is ready to be read.
FileWriter extends OutputStream Class

• Allow writing of streams of characters to a file. (not bytes, chars).

  • `FileWriter(File file)`
    – Constructs a FileWriter object given a File object.

  • `FileWriter(String fileName)`
    – Constructs a FileWriter object given a file name.

  • `FileWriter(String fileName, boolean append)`
    – Constructs a FileWriter object given a file name with a boolean indicating whether or not to append the data written.

  • `public void write(char[] cbuf, int off, int len)`
    – a portion of an array of characters.

  • `public void write(String str, int off, int len)`
  • `public void flush()`
    – Flush the stream to disk.
Streams

• Files are one type of stream device
• Also:
  – Pipe, Buffer, String
  – Classes:
    • BufferedReader
    • LineNumberReader
    • CharArrayReader
    • InputStreamReader
    • FilterReader
    • PushbackReader
    • PipedReader
    • StreamReader
    • Writer
What is Swing?

• A set of classes (part of JFC) that support platform independent GUI (Graphical User Interface)

• Successor to the original Java GUI classes (AWT) which didn’t work very well (they had platform dependencies that really made it a difficult API to use)

• AWT wasn’t very “sexy”
Swing

• Visible “widgets” - windows, buttons, combo boxes, trees, tables, checkboxes, text fields, menus, …

• Containers of components – applets, dialogs, windows and frames

• Supporting classes and utility methods
Some important Swing visible component classes

- JApplet **
- JButton
- JCheckBox
- JColorChooser
- JComboBox
- JDialog **
- JFileChooser
- JFormattedTextField
- JFrame **
- JLabel
- JList
- JMenu
- JMenuBar
- JMenuItem
- JPanel
- JPasswordField
- JPopupMenu
- JProgressBar
- JRadioButton
- JScrollPane
- JScrollBar
- JSlider
- JSpinner
- JTable
- JTextArea
- JTextField
- JToggleButton
- JToolBar
- JTree
- JWindow **

** means a top level containers
Using netbeans to create a JDialog
adding fields
my empty CustomerInfoDialog:JDialog
code created
executing the class displays:
To kill a zombie or running process in netbeans right click and choose: ”terminate”
Editing a dialog

• 1st select a layout manager for the dialog
select component

edit properties
changing the layout manager
what layout manager should I use?

• Start with the absolute and then experiment when you feel comfortable (or hire a graphic artist and let them worry about it ;-).
Adding other components to the view - JTextFields
execute the class
Adding a combo box
edit the model property for the combo box

type in state abbreviations separated by commas
preferred size property

hor, vert
“Yes” button event handler

```java
private void YesButtonActionPerformed(ActionEvent evt)
{
    // Add your handling code here:
    saveCustomerInfoToFile("/CustomerInfoDialog.txt");
}
```
I added this code to `saveCustomerInfoToFile`

```java
boolean saveCustomerInfoToFile(String fileNameToSaveTo)
{
    StringBuffer myCustInfoSB = new StringBuffer();
    try{
        myCustInfoSB.append(custNameText.getText());
        myCustInfoSB.append("|");
        myCustInfoSB.append(custAddressText.getText());
        myCustInfoSB.append("|");
        myCustInfoSB.append(custCityText.getText());
        myCustInfoSB.append("|");
        myCustInfoSB.append(custStateCombo.getSelectedItem());
        myCustInfoSB.append("|");
        myCustInfoSB.append(custZipcodeText.getText());
        myCustInfoSB.append("|");
        return FileIODemo.WriteStringToFile(fileNameToSaveTo,myCustInfoSB.toString() );
    }catch(Exception e){
        return false; //just catch the exceptions an fail
    }
}
```
Run the class

- Creates a file called CustomerInfoDialog.txt
copy the CustomerInfoDialog

• I copied CustomerInfoDialog in the tree view of netbeans then pasted it into Lecture4
• Then changed the name
• Then added a Load button
I clicked the button and then added the method call

```java
private void LoadButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // Add your handling code here:
    getCustomerInfoFromFile("/CustomerInfoDialog.txt");
}
```
change the main to instantiate the correct class

```java
public static void main(String args[]) {
    new CustomerInfoDialog_2(new javax.swing.JFrame(), true).show();
}
```
code I added to the
getCustomerInfoFromFile()

boolean getCustomerInfoFromFile(String customerFileName)
{
    StringBuffer myCustInfoSB = new StringBuffer();

    if (FileIODemo.ReadStringFromFile(customerFileName,myCustInfoSB))
    {
        //this uses a quick and dirty parser to get values from the string returned by reading the
        //file
        //it is highly dependant on the code that wrote the file and very non-robust
        StringTokenizer myStringTokenizer = new StringTokenizer(myCustInfoSB.toString(),"|");

        custNameText.setText( myStringTokenizer.nextToken());
        custAddressText.setText( myStringTokenizer.nextToken());
        custCityText.setText( myStringTokenizer.nextToken());
        custStateCombo.setSelectedItem( myStringTokenizer.nextToken());
        custZipcodeText.setText( myStringTokenizer.nextToken());
        return true;
    }
    return false;
}
StringTokenizer class

- Parses a string into substrings
- You can specify a number of delimiters
  - StringTokenizer(String str)
  - StringTokenizer(String str, String delim)
  - int countTokens()
    - Calculates the number of times that this tokenizer's nextToken method can be called before it generates an exception.
  - boolean hasMoreElements()
    - Returns the same value as the hasMoreTokens method.
  - boolean hasMoreTokens()
    - Tests if there are more tokens available from this tokenizer's string.
  - Object nextElement()
    - Returns the same value as the nextToken method, except that its declared return value is Object rather than String.
  - String nextToken()
    - Returns the next token from this string tokenizer.
  - String nextToken(String delim)
    - Returns the next token in this string tokenizer's string.
executing the new version
JFrameButtonListenerDemo class
JFrameButtonListenerDemo class

• Shows how to add an event listener to a component (the JTextField).
• Then respond to the events in some way.
• Shows how to handle exiting from a swing top level component
  – myJFrameButtonListenerDemo.addWindowListener()
  – This should the use of an adapter class
SourceTextKeyListener class

public class SourceTextKeyListener implements KeyListener {

    public void keyPressed(KeyEvent e) {
        //Invoked when a key has been pressed.
        System.out.println("KeyPressed : " + e);
    }

    public void keyReleased(KeyEvent e) {
        System.out.println("KeyReleases : " + e);
        // Invoked when a key has been released.
    }

    public void keyTyped(KeyEvent e) {
        System.out.println("KeyTyped : " + e);
        //Invoked when a key has been typed.
        eventsTextArea.setText(sourceText.getText()); //this should get placed in the key pressed event listener
    }
}

Copyright Poelman & Associates, Inc. 2002 - All Rights Reserved
Registering a listener on a component in the JFrame

- In the constructor for the JFrame we add the listener

```java
public JFrameButtonListenerDemo()
{
    initComponents();
    //register my listeners here
    sourceText.addKeyListener(mySourceTextKeyListener);
}
```
MVC – Model View Controller

• Swing components are designed as MVC components
  – Model = data or object that is the to be visually represented
  – View = one or more visual representations of that data/object
  – Controller = code to manage input to the model
MVC

- © Sun 2002
MVC in Swing Components

- The Swing component class is the view and controller
- A separate class is the model
- Most components come with a default model
- You can set the model to your own model for a control
- Several controls could share a model!
JFrame

JTree

JTree

JTree

JButton

JTable

JPasswordField

JSlider

JFileChooser

JProgressBar

Copyright Poelman & Associates, Inc. 2002 - All Rights Reserved
private javax.swing.JPasswordField jPasswordField1;
private javax.swing.JTree jTree1;
private javax.swing.JSlider jSlider1;
private javax.swing.JProgressBar jProgressBar1;
private javax.swing.JTable jTable1;
private javax.swing.JButton jButton2;
private javax.swing.JButton jButton1;
private javax.swing.JFileChooser jFileChooser1;
private javax.swing.JLabel jLabel1;
Swing based MenuLookDemo

![Swing based MenuLookDemo](image)
SliderDemo
Summary

- String & StringBuffer
- File IO – File, FileReader, FileWriter
- Array – real objects
  - hold references to objects (or primitives)
- Swing
  - GUI classes
  - Containers and components
  - MVC
Homework #1

Inventory Browser

<table>
<thead>
<tr>
<th>Book Title</th>
<th>ISBN</th>
<th>Author</th>
<th>Publisher</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning Java</td>
<td>0-596-00285-8</td>
<td>Niemeyer &amp; Knudsen</td>
<td>O'Reilly</td>
<td>$44.95</td>
</tr>
<tr>
<td>Java in a Nutshell</td>
<td>0-596-00286-9</td>
<td>unknown</td>
<td>O'Reilly</td>
<td>$39.95</td>
</tr>
<tr>
<td>Java in a Nutshell J2EE</td>
<td>0-599-00286-8</td>
<td>unknown</td>
<td>O'Reilly</td>
<td>$59.95</td>
</tr>
<tr>
<td>Taming Java Threads</td>
<td>0-396-00286-9</td>
<td>Holub</td>
<td>APress</td>
<td>$34.95</td>
</tr>
<tr>
<td>Hitchhiker's Guide To The Galaxy</td>
<td>0345391802</td>
<td>Adams</td>
<td>Ballantine Books</td>
<td></td>
</tr>
<tr>
<td>Restaurant at the end of the universe</td>
<td>0345391810</td>
<td>Adams</td>
<td>Ballantine Books</td>
<td></td>
</tr>
<tr>
<td>So long and thanks for all the fish</td>
<td>0671746530</td>
<td>Adams</td>
<td>Pocket Books</td>
<td></td>
</tr>
</tbody>
</table>

So long and thanks for all the fish

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Add To Shopping Cart</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Book Title</th>
<th>ISBN</th>
<th>Quantity</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hitchhiker's Guide To The Galaxy</td>
<td>0345391802</td>
<td>1</td>
<td>$7.95</td>
</tr>
<tr>
<td>Restaurant at the end of the universe</td>
<td>0345391810</td>
<td>5</td>
<td>$7.95</td>
</tr>
<tr>
<td>Taming Java Threads</td>
<td>0-396-00286-9</td>
<td>1</td>
<td>$34.95</td>
</tr>
</tbody>
</table>

Copyright Poelman & Associates, Inc. 2002 - All Rights Reserved
Classes

• Customer
• Item
  - InventoryItem
  - POItem
• PO
• Inventory
• CustomerInfoDialog
• CustomerOrderDialog
• FileIOUtil
<table>
<thead>
<tr>
<th>Book Title</th>
<th>ISBN</th>
<th>Author</th>
<th>Publisher</th>
<th>Price</th>
<th>Quantity</th>
<th>ISBN Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning Java</td>
<td>0-596-00285-8</td>
<td>Niemeyer &amp; Knudsen</td>
<td>O'Reilly</td>
<td>44.95</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Java in a Nutshell</td>
<td>0-596-00286-8</td>
<td>unknown</td>
<td>O'Reilly</td>
<td>39.95</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Java in a Nutshell J2EE</td>
<td>0-599-00286-8</td>
<td>unknown</td>
<td>O'Reilly</td>
<td>59.95</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Taming Java Threads</td>
<td>0-396-00286-8</td>
<td>Holub</td>
<td>APress</td>
<td>34.95</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>Hitch Hiker's Guide To The Galaxy</td>
<td>0345391802</td>
<td>Adams</td>
<td>Ballantine Books</td>
<td>7.95</td>
<td>42</td>
<td>42</td>
</tr>
<tr>
<td>Restaurant at the end of the universe</td>
<td>0345391810</td>
<td>Adams</td>
<td>Ballantine Books</td>
<td>7.95</td>
<td>42</td>
<td>42</td>
</tr>
<tr>
<td>So long and thanks for all the fish</td>
<td>0671745530</td>
<td>Adams</td>
<td>Pocket Books</td>
<td>7.95</td>
<td>42</td>
<td>42</td>
</tr>
</tbody>
</table>
submitting it

- zip up all the files - *.java, *.txt
- or jar them
- send to YOUR TA before start of lecture October 14th.
See the site for detailed description
Resources

• Tutorials

• Design patterns