Collections, More About Swing
Collections

• A Java collection is a class that is used to organize a set of related objects.
• A collection implements one or more interfaces, including:
  – List
  – Set
  – Map
• Java collections are found in java.util:
  – ArrayList
  – HashSet
  – HashMap

Oracle:
Interfaces based on Map “are not true collections. However, these interfaces contain collection-view operations, which enable them to be manipulated as collections.”
Collections Are Generic

- Collections cannot hold primitive types.
- Any collection can hold almost any type of object, but any one collection must limit itself to a single type.
- The type of a collection must be declared using the diamond (<>) operator.

```java
ArrayList<Integer> aList = new ArrayList<>();
Creates an array list that can hold Integer objects.

HashMap<String, Person> hMap = new HashMap<>();
Creates a HashMap in which Person objects are indexed by String objects.
```
Collections Retrospective

• In the beginning, collections were declared like this:
ArrayList list = new ArrayList();
This is an example of weak typing.

• Next Java introduced the ability to specify the type of a collection:
ArrayList<Turtle> list = new ArrayList<Turtle>();
This is an example of strong typing.

• Currently declarations can be declared using simplified syntax:
ArrayList<Turtle> list = new ArrayList<>();
ArrayLists

- Array lists are ordered collections. Elements of the collection can be added and accessed by index, somewhat like an array:

```java
public class ArrayListDemo1
{
    public static void main(String[] args)
    {
        ArrayList<Person> personList = new ArrayList<>();
        personList.add( new Person( "Mickey", "Mantle" ) );
        personList.add( new Person( "Sandy", "Koufax" ) );
        personList.add( new Person( "Willy", "Mays" ) );
        personList.add( 1, new Person( "Roger", "Maris" ) );
        for ( int inx = 0 ; inx < personList.size() ; ++inx )
            System.out.println( personList.get( inx ) );
    }
}
```

continued on next slide
Arrays

continued

• Output:

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mantle, Mickey</td>
</tr>
<tr>
<td>Maris, Roger</td>
</tr>
<tr>
<td>Koufax, Sandy</td>
</tr>
<tr>
<td>Mays, Willy</td>
</tr>
</tbody>
</table>

See the documentation for class ArrayList and interface List
For-Each Loops

For-each loops are convenient for processing arrays and collections:

```java
for ( type identifier : collection )
    statement
```

```java
public class ArrayListDemo2 {
    public static void main(String[] args) {
        ArrayList<Person> personList = new ArrayList<>();
        personList.add( new Person( "Mickey", "Mantle" ) );
        personList.add( new Person( "Sandy", "Koufax" ) );
        personList.add( new Person( "Willy", "Mays" ) );
        personList.add( 1, new Person( "Roger", "Maris" ) );
        for ( Person person : personList )
            System.out.println( person );
    }
}
```
The USState Class

public class USState
{
    private final String name_;  
    //    private String capitol_;  
    //    private String flower_;  
    //    motto_, etc.;  
    public USState( String name )
    {
        name_ = name;
    }
    private String getName()
    {
        return name_;  
    }
    public String toString()
    {
        return name_;  
    }
    //    public String getCapitol()
    //    {
    //        return capitol_;  
    //    }
    //    etc.
}

The USState class is intended to hold information about states in the US.
The USState Class

```java
public class USState {
    private final String name_; // private String capital_; // private String flower_; // motto_, etc.;
    public USState( String name )
    {
        name_ = name;
    }
    public String getName() {
        return name_;
    }
    public String toString() {
        return name_; // public String getCapital() // { //     return capital_; // } // etc.
    }
}
```

The USState class is intended to hold information about states in the US.
Exercises

1. Write and compile USState class.
2. Declare an ArrayList to hold USState objects. Populate it with three or four objects; use a for-each loop to print out the contents of the list.
3. Write an application that repeatedly asks the operator to enter the name of a state. For each name create a USState object and store the object in an array list. When the operator cancels the operation print out the contents of the ArrayList.
4. Recall that List is an interface. Write a test program with the following method:
   ```java
   private static List<USState>
   filter( List<USState> list, String start )
   The method examines the contents of list for any state with a name that starts with start.
   5. Write a method called filterOut which is similar to the above, but returns a list of all states with names that do not begin with start.
   6. Advanced: write an application like (3), above, but store the states in the list in alphabetical order.
HashMaps

- A HashMap is a collection of objects that can be accessed by key.
- The key must be unique for every object.
- The declaration of a HashMap includes the type of the key and the type of the value.

```java
public class ArrayListDemo1 {
    public static void main(String[] args) {
        ArrayList<Person> personList = new ArrayList<>();
        personList.add(new Person("Mickey", "Mantle"));
        personList.add(new Person("Sandy", "Koufax"));
        personList.add(new Person("Willy", "Mays"));
        personList.add(1, new Person("Roger", "Maris"));
        for (int inx = 0; inx < personList.size(); ++inx)
            System.out.println(personList.get(inx));
    }
}
```
public class HashMapDemo1
{
    public static void main(String[] args)
    {
        HashMap<String, USState> states = new HashMap<>();
        states.put( "Connecticut", new USState( "Connecticut" ) );
        states.put( "New York", new USState( "New York" ) );
        states.put( "Arizona", new USState( "Arizona" ) );

        USState state = states.get( "New York" );
        if ( state != null )
            System.out.println( state );
    }
}
Autoboxing

- When using primitives with a collection Java will automatically convert between the primitive (such as int) and its associated class (such as Integer). This is called autoboxing.

```java
public class AutoboxingDemo {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(45);
        list.add(100);
        for (int inx : list) {
            System.out.println(inx);
        }
    }
}
```
More About Expressions

• Recall that the type of a comparison expression is either true or false.
• Recall that, in Java, assignment (=) is an operator.
• Be aware that the assignment operator has a lower precedence than the comparison operators.
• This gives us the ability to form complex expressions like this one:

```java
while ( (str = askString()) != null )
    list.add( str );
```

Parentheses are required to alter the order of evaluation.
public class Test
{
    public static void main( String[] args )
    {
        ArrayList<String> list = new ArrayList<>();
        String str;
        while ( (str = askString()) != null )
        {
            list.add( str );
        }
        for ( String var : list )
        {
            System.out.println( var );
        }
    }

    private static String askString()
    {
        String str =
            JOptionPane.showInputDialog( null, "Enter a string" );
        return str;
    }
}
Exercises (1 of 2)

1. Write and test the application depicted in the slide Expression Example.

2. If you haven’t already done so, implement the class OpCanceledException as shown:
   
   ```java
   // OpCanceledException.java
   public class OpCanceledException
      extends Exception
   {
   
   }
   ```

3. Write an application that:
   
   a) Contains the method `String askModelName()`. This method will ask the operator for a string and return it. If the operator cancels the operation, throw an `OpCanceledException`.
   
   b) Contains the method `int askModelNumber()`. This method will ask the operator to enter a four digit model number. If the user fails to enter a proper four digit number, display an error message and reprompt. Convert the operator’s string to an int and return it. If the operator cancels the operation, throw an `OpCanceledException`.
   
   c) Repeatedly asks the operator for a model name and model number, and adds them to a `HashMap<Integer, String>`. When the operator cancels the operation, go to step (d).
d) Repeatedly asks the operator for a model number, and displays the model name associated with it. If there is no mapping from model number to name, display “none found.” When the operator cancels the operation, exit the program.
Managing Collections (1)

Common operations for ArrayList<E>:

- \texttt{add( E elem )} \textit{add elem to end of list.}
- \texttt{add( int pos, E elem )} \textit{add elem at position pos; pos must be \( \leq \) list.size().}
- \texttt{set( int pos, E elem )} \textit{changes position pos to elem; pos must be \(<\) list.size(); returns the element formerly at position pos.}
- \texttt{get( int pos )} \textit{returns the element at position pos; pos must be \(<\) list.size()}
- \texttt{remove( int pos )} \textit{removes the element at position pos; pos must be \(<\) list.size(); returns the element formerly at position pos.}
- \texttt{remove( Object obj )} \textit{removes the first element that matches obj; returns true if the object was in the list.}
- \texttt{clear()} \textit{empties the list.}
Managing Collections (2)

Common operations for HashMap<K, V>:

- **put(K key, V value)**
  
  *Key already exists in map: replaces the value associated with key.*
  
  *Key not in map: K/V added to map*

- **get(Object obj)**
  
  *returns the value that maps to key obj; null if none.*

- **remove(Object obj)**
  
  *removes the key/value that maps to key obj; returns the value formerly mapped to the key, or null if none.*

- **clear()**
  
  *empties the map.*

- **containsKey(Object obj)**
  
  *returns true if key obj exists in the map.*

- **containsValue(Object obj)**
  
  *returns true if value obj exists in the map.*
HashMap.get

Consider this code for populating/interrogating a HashMap:

```java
public static void main( String[] args ) {
    HashMap<String, USState> map = new HashMap<>();
    map.put( "a", new USState( "florida" ) );
    map.put( "b", new USState( "connecticut" ) );
    map.put( "c", new USState( "new jersey" ) );
    USState state = map.get( "B" );
    if ( state == null )
        System.out.println( "B isn't mapped to anything" );
    else
        System.out.println( state );
}
```

.get returns null means “B” isn’t in the map.

continued on next slide
HashMap.get

continued

Now look at this code; what is the problem?

```java
public static void main( String[] args )
{
    HashMap<String, USState> map = new HashMap<>();
    map.put( "a", new USState( "florida" ) );
    map.put( "b", null );
    map.put( "c", new USState( "new jersey" ) );
    USState state = map.get( "b" );
    if ( state == null )
        System.out.println( "b isn't mapped to anything" );
    else
        System.out.println( state );
}
```

.get returns null; does that mean "b" isn’t in the map?

continued on next slide
HashMap.get

continued

Use HashMap.containsKey( Object ) to determine if a key appears in the map.

public static void main( String[] args )
{
    HashMap<String, USState> map = new HashMap<>();
    map.put( "a", new USState( "florida" ) );
    map.put( "b", null );
    map.put( "c", new USState( "new jersey" ) );
    if ( !map.containsKey( "b" ) )
        System.out.println( "b isn't mapped to anything" );
    else
        System.out.println( map.get( "b" ) );
}
Private Classes

- A java file that contains a public class may also contain one or more private classes.
- Private classes are not nested inside other classes.
- No modifier is placed on the class declaration.
- Private classes can only be used inside the file that declares them.

```java
public class A
{
    ...
}

// private class
class B
{
    ...
}
```
public class PrivateClassDemo1
{
    ...
    private static String askString() throws CanceledException
    {
        String str = JOptionPane.showInputDialog( null, "Enter a string" );
        if ( str == null )
            throw new CanceledException();
        return str;
    }
}

class CanceledException extends Exception
{
}

public class OptionDialogDemo1 {
    public static void main(String[] args) {
        String[] options = { "moveOn", "backUp", "dance", "Quit" };  
        int result = JOptionPane.showOptionDialog( null,
                "What do you want to do?",
                "OptionDialog Demo 1",
                JOptionPane.DEFAULT_OPTION,
                JOptionPane.QUESTION_MESSAGE,
                null,
                options,
                "moveOn"
        );

        System.out.println( result );
    }
}
Option Dialogs

JOptionPane.showMessageDialog(  
  Component parent, // typically null  
  Object message, // message to display  
  String title, // title to display in dialog title bar  
  int optionType, // typically JOptionPane.DEFAULT_OPTION  
  int messageType, // e.g. JOptionPane.QUESTION_MESSAGE  
  Icon icon, // typically null  
  Object[] options, // array of options to display as buttons  
  Object initialValue // initially selected option
)

Returns:
  the index of the option selected, or -1 if none.
The NamedTurtle Class

```java
public class NamedTurtle {
    private final String name_; 
    public NamedTurtle() {
        name_ = "Default";
    }
    public NamedTurtle( String name) {
        name_ = name;
    }
    public String getName() {
        return name_; 
    }
    public String toString() {
        return name_; 
    }
}
```

We’ll be using this class in our exercises.
Exercise

1. Create an array of four NamedTurtles; give them any names you like. Use the array to display an option dialog. Print out the NamedTurtle that was selected in the dialog, if any.

2. Write the HashTurtle application that maintains a HashMap<String, NamedTurtle> where the key is the turtle’s name. Display an option dialog showing the options Add and Quit. Repeatedly ask the user to select an option; if the user selects Add ask for the name of a turtle, then add a NamedTurtle to the HashMap. Continue until the operator selects Quit.

3. Modify HashTurtle so that it will display an error message if the user tries to add a turtle with a name that is already in the map.

4. Add a Delete option to the HashTurtle option dialog. If the user selects Delete ask for a turtle name, then delete that turtle from the map. If the name does not exist in the map display an error message.
Event Processing In Swing

An event occurs when a Java GUI component changes state. Examples:

- A window opens or closes.
- A pushbutton is pressed.
- The mouse moves.
- The user types a character in a text box.
- A component changes size.
- A component becomes visible.
- The font in a text box changes.
Handling Events

- When an event occurs your application can, optionally, *handle* the event.
- The *event handler* is always an instance of a class that implements a particular interface.
- The interface contains a method that will be invoked when the corresponding event is *triggered, or dispatched*.
- You can have a single event handler that handles many events, or you may have many event handlers registered for a single event.
- Few applications have handlers registered for all events; most applications are concerned with just a few.
Techniques For Handling Events: Using The Main Class

```java
public class Test
    extends JFrame
    implements Runnable, ActionListener
{
    ...
    public void run()
    {
        setTitle( "JPanel Demo 1" );
        setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        Container pane = getContentPane();
        JButton date = new JButton( "Show Date" );
        date.addActionListener( this );
        pane.add( date );
        pack();
        setVisible( true );
    }
}

public void actionPerformed( ActionEvent evt )
{
    ...
}
```
Techniques For Handling Events: Using Nested Classes

```java
public class Test
    extends JFrame
    implements Runnable
{
    ...
    public void run()
    {
        setTitle( "JPanel Demo 1" );
        setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        Container pane = getContentPane();
        JButton date = new JButton( "Show Date" );
        date.addActionListener( new ButtonListener() );
        pane.add( date );
        pack();
        setVisible( true );
    }

    private class ButtonListener implements ActionListener
    {
        ...
    }
}
```
Getting The Event Source

- Any event listener can determine the source of an event.
- The event source is stored as type Object; to treat it as a particular type, cast it to that type.

```java
public class Test
    extends JFrame
    implements Runnable
{
    ...
    private class ButtonListener
        implements ActionListener
    {
        public void actionPerformed( ActionEvent evt )
        {
            JButton button = (JButton)evt.getSource();
            System.out.println( button.getText() );
        }
    }
}
```

If source isn’t a JButton your program will crash.
Listener Parameters

• Every listener method has a parameter that stores information about the event; *ActionEvent* includes:
  – The source of the event (*.getSource()*)
  – Which shift/control/alt/etc. keys were pressed at the time of the event (*.getModifiers*)
  – The time that the event occurred (*.getWhen()*).

• The *MouseEvent* parameter includes the above, plus:
  – Which, if any, mouse button was pressed (*.getButton()*
  – The location of the mouse on the screen when the event occurred (*.getLocationOnScreen()*
  – The location of the mouse in the window where the event occurred (*.getPoint()*).
public class OneEventDemo1
    extends JFrame
    implements Runnable, ActionListener
{
    public static void main(String[] args)
    {
        Test demo    = new Test();
        SwingUtilities.invokeLater( demo );
    }
    public void run()
    {
        setTitle( "JPanel Demo 1" );
        setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        Container pane = getContentPane();
        JButton date = new JButton( "Show Date" );
        date.addActionListener( this );
        pane.add( date );
        pack();
        setVisible( true );
    }
    ...
continued on next slide
Listening For One Event: Example 1

continued

```java
...  
  public void actionPerformed( ActionEvent evt )  
  {  
      SimpleDateFormat date = new SimpleDateFormat( "dd-MMM-yyyy" );  
      final String str = date.format( new Date() );
      JOptionPane.showMessageDialog( null, str );  
  }
}  
```
public class OneEventDemo2 extends JFrame
    implements Runnable
{
    public static void main(String[] args)
    {
        OneEventDemo2 demo = new OneEventDemo2();
        SwingUtilities.invokeLater( demo );
    }
    public void run()
    {
        setTitle( "JPanel Demo 1" );
        setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        Container pane = getContentPane();
        JButton date = new JButton( "Show Date" );
        date.addActionListener( new ButtonListener() );
        pane.add( date );
        pack();
        setVisible( true );
    }
    ...
continued on next slide

private class ButtonListener implements ActionListener
{
    public void actionPerformed( ActionEvent evt )
    {
        SimpleDateFormat date = new SimpleDateFormat( "dd-MMM-yyyy" );
        final String str = date.format( new Date() );
        JOptionPane.showMessageDialog( null, str );
    }
}"
Exercise

1. Write an application that displays one pushbutton in a frame with the label “Push Me.” Add an ActionListener to the button which will pop-up the message “Don’t Push That Again!” when the user clicks it. For an ActionListener, use your main class.

2. Write an application that displays one toggle button in a frame with the label “Tickle Me” (new JToggleButton( “Tickle Me” )). Add an ActionListener to the toggle button that will change the foreground color to yellow if selected, and blue if not selected. Hints:
   - Use eventobject.getSource() to get the toggle button; don’t forget you have to cast it: 
     
     ```java
     JToggleButton button = (JToggleButton)evt.getSource();
     ```
   - Use button.setForeground( Color ) to set the foreground color. Use Color.YELLOW and Color.BLUE; you will have to import java.awt.Color.
   - Use button.isSelected() to determine if the toggle button is selected:
     ```java
     if ( button.isSelected() )
     ```
public class MultipleEventsDemo1
   extends JFrame
   implements Runnable, ActionListener
{
   private static JButton date_ = new JButton( "Show Date" );
   private static JButton time_ = new JButton( "Show Time" );

   public static void main(String[] args)
   {
      MultipleEventsDemo1 demo    = new MultipleEventsDemo1();
      SwingUtilities.invokeLater( demo );
   }
...

One event handler using the main class as the ActionListener.

Note: date_ and time_ buttons stored as class variables.
Listening For Multiple Events: Example 1

```java
public void run(){
    setTitle( "MultipleEvents Demo 1" );
    setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    Container pane = getContentPane();
    pane.setLayout( new GridLayout( 1, 2 ) );
    date_.addActionListener( this );
    pane.add( date_ );
    time_.addActionListener( this );
    pane.add( time_ );
    pack();
    setVisible( true );
}
```

continued on next slide
Listening For Multiple Events:

Example 1

```java
... public void actionPerformed( ActionEvent evt ) {
    Object obj = evt.getSource();
    SimpleDateFormat date;
    if ( obj == time_ )
        date = new SimpleDateFormat( "HH:mm:ss" );
    else if ( obj == date_ )
        date = new SimpleDateFormat( "dd-MMM-yyyy" );
    else
        throw new IllegalArgumentException( "Button not found" );
    final String str = date.format( new Date() );
    JOptionPane.showMessageDialog( null, str );
}
```
Listening For Multiple Events: Example 2

```java
public class MultipleEventsDemo2
    extends JFrame
    implements Runnable
{
    private static JButton date_ = new JButton( "Show Date" );
    private static JButton time_ = new JButton( "Show Time" );

    public static void main(String[] args)
    {
        MultipleEventsDemo2 demo = new MultipleEventsDemo2();
        SwingUtilities.invokeLater( demo );
    }
...}
```

Note: date_ and time_ buttons stored as class variables.

One event handler implemented by a nested class.

continued on next slide
Listening For Multiple Events: Example 2

```java
... public void run() {
    setTitle( "MultipleEvents Demo 2" );
    setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    Container pane = getContentPane();
    pane.setLayout( new GridLayout( 1, 2 ) );
    date_.addActionListener( new ButtonHandler() );
    pane.add( date_ );
    time_.addActionListener( new ButtonHandler() );
    pane.add( time_ );
    pack();
    setVisible( true );
}
...
Listening For Multiple Events: Example 2

... private class ButtonHandler implements ActionListener {
    public void actionPerformed( ActionEvent evt ) {
        Object obj = evt.getSource();
        SimpleDateFormat date;
        if ( obj == time_ )
            date = new SimpleDateFormat( "HH:mm:ss" );
        else if ( obj == date_ )
            date = new SimpleDateFormat( "dd-MMM-yyyy" );
        else
            throw new IllegalArgumentException( "Button not found" );
        final String str = date.format( new Date() );
        JOptionPane.showMessageDialog( null, str );
    }
}

Remember: a nested class can access its parent’s variables.
public class MultipleEventsDemo3
    extends JFrame
    implements Runnable
{
    public static void main(String[] args)
    {
        MultipleEventsDemo3 demo = new MultipleEventsDemo3();
        SwingUtilities.invokeLater( demo );
    }
...

Two event handlers implemented by nested classes.

continued on next slide
Listening For Multiple Events: Example 3

```java
... public void run(){
    JButton date = new JButton( "Show Date" );
    JButton time = new JButton( "Show Time" );

    setTitle( "MultipleEvents Demo 3" );
    setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    Container pane = getContentPane();
    pane.setLayout( new GridLayout( 1, 2 ) );
    date.addActionListener( new TimeButtonHandler() );
    pane.add( date );
    time.addActionListener( new DateButtonHandler() );
    pane.add( time );
    pack();
    setVisible( true );
}
... continued on next slide
Listening For Multiple Events:
Example 3

```
...  
private class TimeButtonHandler implements ActionListener
{
    public void actionPerformed( ActionEvent evt )
    {
        Object            obj = evt.getSource();
        SimpleDateFormat date = new SimpleDateFormat( "HH:mm:ss" );
        final String      str = date.format( new Date() );
        JOptionPane.showMessageDialog( null, str );
    }
}
private class DateButtonHandler implements ActionListener
{
    public void actionPerformed( ActionEvent evt )
    {
        Object        obj = evt.getSource();
        SimpleDateFormat date = new SimpleDateFormat( "dd-MMM-yyyy" );
        final String    str = date.format( new Date() );
        JOptionPane.showMessageDialog( null, str );
    }
}
```
Listening For Multiple Events: Example 4

```java
public class MultipleEventsDemo4
    extends JFrame
    implements Runnable
{
    public static void main(String[] args)
    {
        MultipleEventsDemo4 demo = new MultipleEventsDemo4();
        SwingUtilities.invokeLater(demo);
    }
    ...
```

Complete encapsulation of Java components using nested classes.

**continued on next slide**
public void run()
{
    setTitle( "MultipleEvents Demo 4" );
    setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    Container pane = getContentPane();
    pane.setLayout( new GridLayout( 1, 2 ) );
    pane.add( new DateButton() );
    pane.add( new TimeButton() );
    pack();
    setVisible( true );
}

...
private class TimeButton
    extends JButton
    implements ActionListener
{
    public TimeButton()
    {
        super( "Time" );
        addActionListener( this );
    }
    public void actionPerformed( ActionEvent evt )
    {
        Object          obj = evt.getSource();
        SimpleDateFormat date = new SimpleDateFormat( "HH:mm:ss" );
        final String     str = date.format( new Date() );
        JOptionPane.showMessageDialog( null, str );
    }
}
...
Listening For Multiple Events:
Example 4

```java
private class DateButton
    extends JButton
    implements ActionListener
{
    public DateButton()
    {
        super( "Date" );
        addActionListener( this );
    }
    public void actionPerformed( ActionEvent evt )
    {
        Object           obj = evt.getSource();
        SimpleDateFormat date = new SimpleDateFormat( "dd-MMM-yyyy" );
        final String     str = date.format( new Date() );
        JOptionPane.showMessageDialog( null, str );
    }
}
```
Exercise

1. Write, compile and test *MultipleEventsDemo2* as shown in the slides.

2. To *MultipleEventsDemo2* add a third button labeled “Quit,” and add a *ButtonHandler* action listener to it. Enhance *ButtonHandler.actionPerformed* so that if the user pokes the quit button, the application will exit (use *System.exit(0)*).

3. Write, compile and test *MultipleEventsDemo4* as shown in the slides.

4. Add a *quit* button to *MultipleEventsDemo4* as in (2), above, but this time fully encapsulate the quit button as we did the *date* and *time* buttons.
JComboBox

• The **JComboBox** is used to create drop-down lists.

```java
{ // ComboBoxDemo1.java
   ... public void run()
   {
      setTitle( "ComboBox Demo 1" );
      setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      Container pane = getContentPane();
      pane.setLayout( new FlowLayout() );
      JComboBox box = new JComboBox();
      box.addItem( "Uruguay" );
      box.addItem( "Argentina" );
      box.addItem( "Bhutan" );
      box.addItem( "Nepal" );
      pane.add( box );
      pack();
      setVisible( true );
   }
}
JComboBox\langle E\rangle(E[]) 

- As of Java 7 JComboBox is a generic type.
- JComboBox\langle E\rangle(E[]) is a constructor that initializes a JComboBox from an array.

```java
{ // ComboBoxDemo2.java
...
| public void run()
| {
|     setTitle( "ComboBox Demo 2" );
|     setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
|     Container pane = getContentPane();
|     pane.setLayout( new FlowLayout() );
|     String[] strs = { "Uruguay", "Argentina", "Bhutan" };
|     JComboBox<String> box = new JComboBox\langle String\rangle( strs );
|     pane.add( box );
|     pack();
|     setVisible( true );
| }
}```
JComboBox\langle E \rangle(\ E[\])

- As of Java 7 JComboBox is a *generic type*.
- `JComboBox\langle E \rangle(\ E[\])` is a constructor that initializes a JComboBox from an array.

```java
{ // ComboBoxDemo2.java
...
  public void run()
  {
    setTitle( "ComboBox Demo 2" );
    setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    Container pane = getContentPane();
    pane.setLayout( new FlowLayout() );
    String[] strs = { "Uruguay", "Argentina", "Bhutan" };
    JComboBox\langle String \rangle box = new JComboBox\langle\rangle( strs );
    pane.add( box );
    pack();
    setVisible( true );
  }
}
```
List.toArray( E[] arr )

- `List.toArray( E[] arr )` fills an array with elements from a `List`...
- ... if `arr` is big enough it fills and returns `arr`...
- ... otherwise a new array of the same type as `arr` is created, filled and returned.

```java
public class ToArrayDemo1 {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.add( "Uruguay" );
        list.add( "Argentina" );
        list.add( "Bhutan" );
        list.add( "Nepal" );
        String[] arr = list.toArray( new String[0] );
        for ( String str : arr )
            System.out.println( str );
    }
}
```

This array will be discarded and a new one created.

continued on next slide
List.toArray( E[] arr )

continued

public class ToArrayDemo2
{
    public static void main(String[] args)
    {
        ArrayList<String> list = new ArrayList<>();
        list.add( "Uruguay" );
        list.add( "Argentina" );
        list.add( "Bhutan" );
        list.add( "Nepal" );
        for ( String str : list.toArray( new String[list.size()] ) )
            System.out.println( str );
    }
}
Using a JComboBox with an ArrayList

Example 1

```java
public class ComboBoxDemo3
    extends JFrame
    implements Runnable
{
    public static void main( String[] args )
    {
        ComboBoxDemo3 demo = new ComboBoxDemo3();
        SwingUtilities.invokeLater( demo );
    }

    public void run()
    {
        setTitle( "ComboBox Demo 3" );
        setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        Container pane = getContentPane();
        pane.setLayout( new FlowLayout() );
        ...
    }
}
```

continued on next slide
Using a JComboBox with an ArrayList

Example 1

continued

```java
...  
ArrayList<String> list = new ArrayList<String>();  
list.add( "Uruguay" );  
list.add( "Argentina" );  
list.add( "Bhutan" );  
list.add( "Nepal" );  
pane.add( getComboBox( list ) );  
  
pack();  
setVisible( true );
}

private JComboBox<String> getComboBox( ArrayList<String> list )
{
    String[] arr = list.toArray( new String[list.size()] );  
    JComboBox<String> box = new JComboBox( arr );  
    return box;
}
```
Using a JComboBox with an ArrayList
Example 2

```java
public class ComboBoxDemo4
    extends JFrame
    implements Runnable
{
    public static void main( String[] args )
    {
        ComboBoxDemo4 demo = new ComboBoxDemo4();
        SwingUtilities.invokeLater( demo );
    }

    public void run()
    {
        setTitle( "ComboBox Demo 4" );
        setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        Container pane = getContentPane();
        pane.setLayout( new FlowLayout() );
        ... continue on next slide
    }
```
Using a JComboBox with an ArrayList

Example 2

continued

ArrayList<String> list = new ArrayList<String>();
list.add( "Uruguay" );
list.add( "Argentina" );
list.add( "Bhutan" );
list.add( "Nepal" );
pane.add( new StrComboBox( list ) );

pack();
setVisible( true );

private class StrComboBox extends JComboBox<String>
{
    public StrComboBox( ArrayList<String> list )
    {
        super( list.toArray( new String[list.size()] ) );
    }
}

...
Common JComboBox Operations

- `addItem( E item )` adds the item to the box.
- `getItemAt( int inx )` returns the item at index inx; null if out-of-range.
- `getItemCount()` returns the number of items in the box.
- `getSelectedIndex()` returns the index of the first selected item in the box; -1 if none.
- `getSelectedltem()` returns the first selected item (as an object) in the box.
- `removeItem( Object item )` removes item from the box.
- `removeItemAt( int inx )` removes the item at index inx.
public class ComboBoxExercise1
    extends JFrame
    implements Runnable
{
    private static ArrayList<String> names_ = new ArrayList<>();

    public static void main( String[] args )
    {
        init();
        ComboBoxExercise1 frame = new ComboBoxExercise1();
        SwingUtilities.invokeLater( frame );
    }
    private static void init()
    {
        // Write a loop asking the operator to enter a name, and add it to names_. Stop when the operator cancels the operation.
    }
    ...

    continued on next slide
ComboBox Exercise 1

private static String askName() {
    String str = JOptionPane.showInputDialog( "Enter a name" );
    return str;
}

public void run() {
    setTitle( "ComboBox Exercise 1" );
    setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    Container pane = getContentPane();
    pane.add( getPanel() );
    pack();
    setVisible( true );
}
ComboBox Exercise 1

continued

private JPanel getPanel()
{
    JPanel panel = new JPanel( new GridBagLayout() );
    GridBagConstraints gbc = new GridBagConstraints();
    // Create a panel with a combobox and two pushbuttons as shown;
    // for the combobox use new StrComboBox().
    return panel;
}

private class StrComboBox extends JComboBox<String>
{
    public StrComboBox()
    {
        // Use List.toArray to get an array of strings from the names_
        // list; pass it to the super class constructor to create and
        // initialize the combobox.
    }
}
Exercises

1. Build the *ComboBoxExercise1* dialog as shown in the slides.

2. Add an event listener to the *Show Selected* button; when pushed, pop up a JOptionPane message box that displays the combo box’s selected item, if any.

3. Add an event listener to the *Delete Selected* button; when pushed, delete the combo box’s selected item, if any.
The `String.split(String)` method splits a string into tokens based on a regular expression:

```java
public class SplitDemo1 {
    public static void main(String[] args) {
        String str = "Dick Jane Sally";
        String[] tokens = str.split(" ");
        System.out.println(tokens.length);
        for (String token : tokens) {
            System.out.println(token);
        }
    }
}
```

Output: 3
Dick
Jane
Sally

String consists of three tokens.
Tokens are separated by delimiters (spaces in this case).
String.split( String regex ) Example 2

Remember that the expression you are splitting on is a *regular expression*. In regular expressions some characters have special meaning. If you’re not familiar with regular expressions think twice before you use this method.

```java
class SplitDemo2 {
    public static void main(String[] args) {
        String str = "Dick.Jane.Sally";
        String[] tokens = str.split( "." );
        System.out.println( tokens.length );
        for ( String token : tokens )
            System.out.println( "">" + token + "<" );
    }
}
```

Output: 0

Regular expression.
**java.util.StringTokenizer Class**

`StringTokenizer` is a utility class that splits strings into tokens.

```java
public class StringTokenizerDemo1 {
    public static void main(String[] args) {
        String str = "Dick.Jane, Sally,Spot";
        StringTokenizer tizer = new StringTokenizer(str, ", . " );
        System.out.println( tizer.countTokens() );
        while ( tizer.hasMoreTokens() )
            System.out.println( tizer.nextToken() );
    }
}
```

Output:

```
4
Dick
Jane
Sally
Spot
```

String to tokenize.

Each character in this string is treated as a delimiter.

Output:

4 Dick Jane Sally Spot
Common StringTokenizer Methods

- `StringTokenizer( String str, String delims )` constructor to tokenize `str` using `delims` as delimiters.

- `.nextToken()` returns the next token from `str`.

- `.hasMoreTokens()` returns true if there is a token to retrieve using `nextToken()`.
Exercises

1. Use a JOptionPane input dialog to ask the operator for a string. Parse the string using StringTokenizer; for delimiters, use space, comma, colon and semicolon.

2. Use a JOptionPane input dialog to ask the operator for a string of the form:
   
   \[
   \text{name} = \text{integer value}
   \]

   For example:
   
   \[
   \text{inx} = 5
   \]

   Assume that there may or may not be spaces at either end of the string, and on either side of the equal sign. Parse the string into name and value tokens, and convert the value to an integer. Print something descriptive, for example:

   \[
   \text{name: inx value: 5}
   \]
ActionTurtle1: The Dialog

Frame

Parameters

JComboBox

Move
Box
Paint
Circle
Swing
Say

MasterPanel with GridLayout(2, 1)

buttonPanel with GridLayout(2, 1)

continued on next slide
public class ActionTurtle1
    implements Runnable
{
    static private  ActionTurtleFrame frame_  = new ActionTurtleFrame();

    private              Turtle              sam_    = new Turtle();
    private              JButton             move_   = new JButton( "Move" );
    private              JButton             paint_  = new JButton( "Paint" );
    private              JButton             swing_  = new JButton( "Swing" );
    private              JButton             box_    = new JButton( "Box" );
    private              JButton             circle_ = new JButton( "Circle" );
    private              JButton             say_    = new JButton( "Say" );
    private              JComboBox           combo_  = new JComboBox();

    public static void main( String[] args )
    {  
        ActionTurtle1 app = new ActionTurtle1();
        SwingUtilities.invokeLater( app );
    }
}
ActionTurtle1 : The Dialog

...
ActionTurtle1 : Setting Up The Event Handler

Add an event handler to each button.

```java
... public void run()
{
    frame_.run();
    ...
    ButtonListener buttonListener = new ButtonListener();
    move_.addActionListener( buttonListener );
    paint_.addActionListener( buttonListener );
    swing_.addActionListener( buttonListener );
    box_.addActionListener( buttonListener );
    circle_.addActionListener( buttonListener );
    say_.addActionListener( buttonListener );

    frame_.pack();
    frame_.setVisible( true );
}
... continued on next slide
ActionTurtle1 : Setting Up The Event Handler

continued

... 
private class ButtonListener
    implements ActionListener
{
    public void actionPerformed( ActionEvent evt )
    {
    }
}
}
Exercise


2. Write the application ActionTurtle2:
   a) Start with the code for ActionTurtle1. Change class ButtonListener to class ActionButton, which extends JButton and implements ButtonListener.
   b) Give ActionButton a constructor that takes a string, the label to display on the button. Pass the string to the ActionButton’s superclass constructor.
   c) Make ActionButton register itself as an ActionListener.

3. Modify the actionPerformed method of ActionButton; if the Say button is pushed, get the text from the TurtleFrame and make sam_ print it:
   ```java
   if ( button == say_ )
   {
       String whatHeSaid = frame_.getParameters();
       sam_.say( whatHeSaid );
   }
   ```
The TurtleParameters Class

The TurtleParameters class will hold a counter and up to two double parameters parsed from the ActionTurtleFrame’s text field.

```java
public class TurtleParameters {
    public final int numParams;
    public final double param1;
    public final double param2;
    ...
}
```

- Notice that the instance variables are final, so we can make them public without worrying about corruption.

*continued on next slide*
The TurtleParameters Class

*continued*

```java
... public final int numParams;
public final double param1;
public final double param2;

public TurtleParameters( double param1 )
{
    numParams = 1;
    this.param1 = param1;
    this.param2 = 0;
}

public TurtleParameters( double param1, double param2 )
{
    numParams = 2;
    this.param1 = param1;
    this.param2 = param2;
}
...```

There are two constructors for initializing the instance variables with either one or two parameters.

*continued on next slide*
The TurtleParameters Class

*continued*

The class has a utility method that parses a string into one or two parameters, creates a TurtleParameters object, and returns it.

```java
... public static TurtleParameters getTurtleParameters( String paramString ) {
    ... 
} ... continued on next slide
```
The TurtleParameters Class

...
ActionTurtle: Validating The Parameters

• Before executing a Turtle action, the parameters entered by the user must be validated:
  – For each action the correct number of parameters must be entered.
  – For each action except “Say” the parameters must be numeric.
• We could validate the parameters immediately before executing the action, for example:
  if ( action == move )
      validate: two parameters entered
      validate: parameters are numeric
  else if ( action == circle )
      validate: one parameter entered
      validate: parameter is numeric
• Or we could encapsulate the validation requirements in the action’s declaration and perform validation before beginning action execution.
Encapsulating Validation Requirements
In Declaration

- In the ActionTurtle code, make the `ActionButton` constructor specify how many double arguments are required for the action:

```java
private class ActionButton extends JButton
    implements ActionListener
{
    private final int numParams_;
    public ActionButton( String label, int numParams )
    {
        super( label );
        numParams_ = numParams;
        addActionListener( this );
    }
...
Encapsulating Validation In Event Handler

• In the ActionTurtle code, revise the ActionButton declarations to reflect the number of required double arguments:

  ```java
  private ActionButton box_   = new ActionButton( "Box", 2 );
  private ActionButton say_   = new ActionButton( "Say", 0 );
  ```

• Perform validation at the top of the event handler. Use a try/catch block, and throw an exception if the parameters are invalid.

  ```java
  public void actionPerformed( ActionEvent evt )
  {
      JButton       button  = (JButton)evt.getSource();
      String        sparams = frame_.getParameters();
      TurtleParameters params = null;
      try
      {
          ...
  
  continued on next slide
Encapsulating Validation In Event Handler

... try {
  if ( numParams_ > 0 ) {
    params = TurtleParameters.getTurtleParameters( sparsms );
    if ( params.numParams != numParams_ ) {
      String err = "Invalid Parameters";
      throw new IllegalArgumentException( err );
    }
  }
  if ( button == say_ ) {
    sam_.say( sparsms );
  }
...
Encapsulating Validation In Event Handler

continued

... else if ( button == move_ )
    sam_.move( params.param1, params.param2 );
...
}
catch ( NumberFormatException exc )
{
    postMessage( "Not a valid number" );
}
catch ( IllegalArgumentException exc )
{
    postMessage( exc.getMessage() );
}
Exercises

1. Revise your ActionTurtle code to implement the changes to theActionButton constructor and declarations as shown in the slides (this is ActionTurtle3.java in the sample code).

2. Add the postMessage method to your ActionTurtle code; note that this should go directly in the ActionTurtle class, not the ActionButton class:
   ```java
   private static void postMessage( String msg )
   {
       JOptionPane.showMessageDialog( null, msg );
   }
   ```

3. Can you think of any reason(s) why you might want to write a postMessage method instead of just writing JOptionPane.showMessageDialog every time you need it?

4. Revise the ActionButton.actionPerformed method to implement validation as shown in the slides.

5. Implement the code for all Turtle actions in the ActionButton.actionPerformed method.
Instead of just a JComboBox, substitute a JPanel containing a JComboBox and add and delete buttons; encapsulate the new panel in the TurtlePanel class.
The TurtlePanel Class

The `TurtlePanel` class will maintain a `JComboBox` of `NamedTurtles`. Upon instantiation, the combo box will contain one turtle named `Default`.

```java
public class TurtlePanel
    extends JPanel
{
    private JComboBox<NamedTurtle> comboBox_ = new JComboBox<>();
    private NamedTurtle defTurtle_ = new NamedTurtle( "Default" );
    private JButton addButton_ = new JButton( "Add" );
    private JButton delButton_ = new JButton( "Delete" );
    public TurtlePanel()
    {
        // Note: JPanels have the FlowLayout layout manager by default
        comboBox_.addItem( defTurtle_ );
        add( addButton_ );
        add( comboBox_ );
        add( delButton_ );
    }
    ...
}
```

*continued on next slide*
The TurtlePanel Class

The *TurtlePanel* class will have an accessor to return the currently selected turtle.

```java
... public NamedTurtle getSelected()
{
    NamedTurtle turtle = (NamedTurtle)comboBox_.getSelectedItem();
    return turtle;
}
```
Exercises

1. Implement the *TurtlePanel* class as shown in the slides.

2. In the *ActionTurtle* class, substitute a *TurtlePanel* for the combo box; delete the instance variable for the original combo box.

3. In the *ActionTurtle* class delete the instance variable *sam_*; instead, use the currently selected turtle in the turtle panel.
Adding To The TurtlePanel

In the *TurtlePanel* class:

1. When the user pushes the *Add* button ask for a new turtle name; if the user fails to enter a name terminate the operation...

2. ... otherwise add the new turtle to the combo box, and make it the currently selected turtle.

```java
public class TurtlePanel
    extends JPanel
    implements ActionListener
{
    ...
```
Adding To The TurtlePanel

...
private void addTurtle()
{
    String name = askName();
    if ( name != null )
    {
        NamedTurtle turt = new NamedTurtle( name );
        comboBox_.addItem( turt );
        comboBox_.setSelectedItem( turt );
    }
}

// Return null if the operation is canceled
private String askName()
{
    String name = // The usual JOptionPane.showInputDialog stuff.
    return name;
}
Exercises

1. Implement the *add* logic in *TurtlePanel* as shown in the slides.

2. Add logic for the *delete* button in the *TurtlePanel* class, which will delete the currently selected turtle unless the selected turtle is *Default*; post an error message if the user tries to delete *Default*. 
The `JColorChooser` class contains a `showDialog` method that displays a fairly sophisticated dialog for choosing a color.
Launching JColorChooser

• The showDialog method takes three arguments: null, dialog title and initial color.
• The method returns the selected color, or null if the user abandons the operation.

```java
Color color =
    JColorChooser.showDialog( null, "Select a color", Color.RED );
if ( color != null )
    System.out.println( color );
```
public class ColorChooserDemo1
    extends JFrame
    implements ActionListener, Runnable
{
    private JTextField rect_ = new JTextField();
    public static void main(String[] args)
    {
        ColorChooserDemo1 demo = new ColorChooserDemo1();
        SwingUtilities.invokeLater(demo);
    }

    public void actionPerformed(ActionEvent evt)
    {
        Color color =
            JColorChooser.showDialog(null, "Select a color", Color.RED);
        if (color != null)
            rect_.setBackground(color);
    }

    ...

continued on next slide
public void run()
{
    setTitle( "ColorChooserDemo 1" );
    setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    JButton button = new JButton( "Change color" );
    Container pane = getContentPane();

    button.addActionListener( this );
    pane.setLayout( new GridLayout( 2, 1 ) );
    pane.add( rect_ );
    pane.add( button );
    pack();
    setVisible( true );
}
ColorComboBox

• The ColorComboBox class implements a JComboBox that displays colors.
• Constructors include ColorComboBox(Color[]) and ColorComboBox(ArrayList<Color>)
• The ColorComboBox class can be downloaded from the class web pages.

```java
public class ColorComboBox
    extends JComboBox<Color>
{
    ...
```
The ColorComboBox class implements a JComboBox that displays colors.

Constructors include `ColorComboBox(Color[])` and `ColorComboBox(ArrayList<Color>)`.

The ColorComboBox class can be downloaded from the class web page.

```java
public class ColorComboBox
    extends JComboBox<Color>
{
    ...
```
Adding Color To ActionTurtle

- Add a ColorComboBox to the list of ActionTurtle’s instance variables:

```java
private ActionButton circle_ = new ActionButton( "Circle", 1 );
private ActionButton say_ = new ActionButton( "Say", 0 );
private TurtlePanel turtles_ = new TurtlePanel();
private ColorComboBox colors_;  // added
```

- Add a constructor to initialize the ColorComboBox:

```java
private ActionTurtle5()
{
    Color[] cArray = { Color.RED, Color.BLUE, Color.GREEN };  // added
    colors_ = new ColorComboBox( cArray );
}
```

*continued on next slide*
Adding Color To ActionTurtle

continued

• In the run method, add a new row to the GridLayout, and add the ColorComboBox immediately after the TurtlePanel:

```java
public void run()
{
    frame_.run();
    Container pane = frame_.getMasterPanel();
    pane.setLayout( new GridLayout( 3, 1 ) );
    pane.add( turtles_ );
    pane.add( colors_ );
    ...  

    continued on next slide
Adding Color To ActionTurtle

continued

• In the ActionButton.ActionPerformed method, after validating the TurtleParameters, switch the selected turtle to the selected color:

```java
try {
    if ( numParams_ > 0 ) {
        params = TurtleParameters.getTurtleParameters( sparams );
        if ( params.numParams != numParams_ ) {
            String err = "Invalid Parameters";
            throw new IllegalArgumentException( err );
        }
        Turtle tutt = turtles_.getSelected();
        tutt.switchTo( (Color)colors_.getSelectedItem() );
        ...
    }
}
```
Exercises

1. Write a `ColorPanel` class based on the architecture of the `TurtlePanel` class. In the class’s constructor initialize the ColorComboBox to `Color.BLACK`, `Color.WHITE`, `Color.RED`, `Color.BLUE` and `Color.GREEN`.

2. Replace the `ColorComboBox` in `ActionTurtle` with a `ColorPanel`. Hint: `ActionTurtle` won’t need a constructor any more.
The Dimension Class

- *Dimension* is a simple class that stores a width and height in public variables.
- *Dimension* is found in java.awt.

```java
Dimension dim = new Dimension(100, 200);
int oldHeight = dim.height;
dim.height = dim.width;
dim.width = oldHeight;
```

See the Dimension documentation.
Setting A Component’s Preferred Size

- Sometimes you would like to specify the specific size of a component; to do this, set the component’s *preferred size*.
- The GUI manager is *not* required to honor the preferred size, but it will typically try.

```java
public void run()
{
    setTitle( "PreferredSize Demo 1" );
    setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    Container pane = getContentPane();
    pane.setLayout( new FlowLayout() );
    JButton butt1   = new JButton( "OK" );
    JButton butt2   = new JButton( "Go Away" );
    pane.add( butt1 );
    pane.add( butt2 );
    pack();
    setVisible( true );
}
```

*continued on next slide*
Setting A Component’s Preferred Size

continued

• Make the two buttons in the previous dialog the same width.

```java
public void run()
{
    ...;
    JButton butt1 = new JButton( "OK" );
    JButton butt2 = new JButton( "Go Away" );
    Dimension dim1 = butt1.getPreferredSize();
    Dimension dim2 = butt2.getPreferredSize();
    if ( dim1.width > dim2.width )
    {
        dim2.width = dim1.width;
        butt2.setPreferredSize( dim2 );
    }
    else
    {
        dim1.width = dim2.width;
        butt1.setPreferredSize( dim1 );
    }
    ...;
}
```
Tweaking The ActionTurtle Size

• Add the following method to both the TurtlePanel and ColorPanel:

```java
public Component getListBox()
{
    return comboBox_;}
```

• Complete this activity in the next exercise.
Exercises

1. In *ActionTurtle* make the combo box in the turtle dialog the same size as the combo box in the color panel.

2. Why did we make *Component* the return type of `getListBox()`?
Borders

The `javax.swing.borders` package contains several classes for "decorating" components, among them:

- **EtchedBorder**: decorates a component with a thin, 3D-effect line, drawn either raised or lowered.
- **TitledBorder**: takes a previously created border and adds a title to it.
public class BordersDemo1
    implements Runnable
{
    public static void main( String[] args )
    {
        BordersDemo1 test = new BordersDemo1();
        SwingUtilities.invokeLater( test );
    }
    public void run()
    {
        JFrame frame = new JFrame( "BorderDemo 1" );
        JPanel panel = new JPanel();
        panel.setLayout( new GridLayout( 2, 3 ) );
        panel.add( new JButton( "A Button" ) );
        panel.add( new JButton( "B Button" ) );
        panel.add( new JButton( "C Button" ) );
        panel.add( new JButton( "D Button" ) );
        panel.add( new JButton( "E Button" ) );
        panel.add( new JButton( "F Button" ) );
        ...
        
        // continued on next slide
... // First create a raised, etched border, // then add a title to it.
EtchedBorder etched =
    new EtchedBorder( EtchedBorder.RAISED );
TitledBorder titled =
    new TitledBorder( etched, "Many Buttons" );
panel.setBorder( titled );
frame.getContentPane().add( panel );
frame.pack();frame.setVisible( true );

Can be RAISED or LOWERED

Previously created border

Title to add to border
Adding Borders To ActionTurtle

Add an etched, titled border to the ActionTurtle’s color panel. The title should be “Colors.”

```java
public void run()
{
    frame_.run();
    Container pane = frame_.getMasterPanel();
    pane.setLayout( new GridLayout( 3, 1 ) );
    pane.add( turtles_ );
    pane.add( colors_ );
    Component turtleList = turtles_.getListBox();
    Component colorList = colors_.getListBox();
    turtleList.setPreferredSize( colorList.getPreferredSize() );

    EtchedBorder etched =
    new EtchedBorder( EtchedBorder.RAISED );
    TitledBorder titled =
    new TitledBorder( etched, "Colors" );
    colors_.setBorder( titled );
    ...
```
The `GridLayout` class allows you to specify horizontal and vertical gaps between the components that it lays out. You can do this using accessor methods, or specify the horizontal and vertical gaps in a constructor:

```java
JPanel looseLayout = new JPanel( new GridLayout( 2, 3, 5, 5 ) );
```

**horizontal gap  vertical gap**
public class GridLayoutGapDemo
    extends JFrame
    implements Runnable
{
    public static void main(String[] args)
    {
        GridLayoutGapDemo demo = new GridLayoutGapDemo();
        SwingUtilities.invokeLater(demo);
    }

    private void addButtons(JPanel panel)
    {
        panel.add( new JButton( "a1 button" ) );
        panel.add( new JButton( "a2 button" ) );
        panel.add( new JButton( "a3 button" ) );
        panel.add( new JButton( "b1 button" ) );
        panel.add( new JButton( "b2 button" ) );
        panel.add( new JButton( "b3 button" ) );
    }
    ...

    // continued on next slide
GridLayout Constructor Example

continued

...  
public void run()
{
    setTitle( "GridLayout GAp Demo" );
    setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    Container pane = getContentPane();
    pane.setLayout( new GridLayout( 2, 1, 0, 5 ) );
    JPanel tightLayout = new JPanel( new GridLayout( 2, 3 ) );
    JPanel looseLayout = new JPanel( new GridLayout( 2, 3, 5, 5 ) );
    addButtons( tightLayout );
    addButtons( looseLayout );
    pane.add( tightLayout );
    pane.add( looseLayout );

    pack();
    setVisible( true );
}
The EmptyBorder Class

The *EmptyBorder* puts a margin around the outside of a container; the parameters correspond to *top, left, bottom, right*:

```java
looseLayout.setBorder( new EmptyBorder( 5, 5, 5, 5 ) );
```
Exercises

In ActionTurtle:
1. Put an etched/titled border around the turtle panel (it should say “Turtles”).
2. Switch the turtle and color panels so that the color panel is on top.
3. Add an inner class to encapsulate the button panel:

   ```java
   private class ButtonPanel
       extends JPanel
   {
   }
   ```

4. In the button panel, change the grid to display the buttons in three rows of two columns each.

5. In the button panel, change the GridLayout to

   ```java
   GridLayout( 2, 3, 5, 5 );
   ```

   This will put a five pixel gap around each button in the panel.

6. In the button panel, use an EmptyBorder to put 5 pixel margins all the way around the panel.