CMER

Centre for Mobile Education and Research

# GUI Components Part II

# Overview

- Creating Menu Items
- Layout Managers
- Creating Custom Fields
- Creating Custom Layout Managers
- Creating Custom Lists

# Creating Menu Items

- **The Application Menu Item API, in the *net.rim.blackberry.api.menuitem* package, lets you add menu items to BlackBerry Java Applications.**

- **The *ApplicationMenuItemRepository* class lets you add or remove menu items from BlackBerry Java Applications.**

# Creating Menu Items (Cont.)

- Define a menu item.
  - Extend the abstract ApplicationMenuItem class.

```
public class SampleMenuItem extends ApplicationMenuItem { ... }
```

- Specify the position of the menu item in the menu.
  - A higher number means that the menu item appears lower in the menu.
  - Invoke ApplicationMenuItem()

```
SampleMenuItem() {
    super(20);
}
```

- Specify the menu item text.
  - Implement toString().

```
public String toString() {
    return "Open the Contacts Demo application";
}
```

# Creating Menu Items (Cont.)

- Specify the behaviour of the menu item.
  - Implement run().

```
public Object run(Object context) {
    Contact c = (Contact)context; // An error occurs if this does not work.
    if ( c ! null ) {
      new ContactsDemo().enterEventDispatcher();
    } else {
      throw new IllegalStateException( "Context is null, expected a Contact
    instance");
    }
    Dialog.alert("Viewing a message in the messaging view");
    return null;
}
```

# Layout Managers

- To arrange components on a screen, use BlackBerry API layout managers .
- The following four classes extend the Manager class to provide predefined layout managers:
  - VerticalFieldManager
  - HorizontalFieldManager
  - FlowFieldManager
  - DialogFieldManager
- To create a custom layout manager, extend Manager.

# Creating Layout Managers

- On an instance of a Screen, complete the following actions:
  - Instantiate the appropriate Manager subclass.
  - Add UI components to the layout manager.
  - Add the layout manager to the screen.

```
VerticalFieldManager vfm = new
    VerticalFieldManager(Manager.VERTICAL_SCROLL);
vfm.add(bitmapField);
vfm.add(bitmapField2);
...
mainScreen.add(vfm)
```

# Creating Custom Fields

- You can only add custom context menu items and custom layouts to a custom field.
- Extend the Field class, or one of its subclasses, implementing the DrawStyle interface to specify the characteristics of the custom field and turn on drawing styles.

```
public class CustomButtonField extends Field implements DrawStyle {
    public static final int RECTANGLE = 1;
    public static final int TRIANGLE = 2;
    public static final int OCTAGON = 3;
    private String _label;
    private int _shape;
    private Font _font;
    private int _labelHeight;
    private int _labelWidth;
}
```

# Creating Custom Fields (Cont.)

- Implement constructors to define the label, shape, and style of the custom button.

```
public CustomButtonField(String label) {
    this(label, RECTANGLE, 0);
}
public CustomButtonField(String label, int shape) {
    this(label, shape, 0);
}
public CustomButtonField(String label, long style) {
    this(label, RECTANGLE, style);
}
public CustomButtonField(String label, int shape, long style) {
    super(style);
    _label = label;
    _shape = shape;
    _font = getFont();
    _labelHeight = _font.getHeight();
    _labelWidth = _font.getAdvance(_label);
}
```

# Creating Custom Fields (Cont.)

- Specify the arrangement of the objects in the field.
  - Implement layout(). Arrange field data so that you perform the most complex calculations in layout() instead of in paint().
  - Within your implementation, perform the following actions:
    - To calculate the available width and height, invoke Math.min() to return the smaller of the specified width and height and the preferred width and height of the field.
    - To set the required dimensions for the field, invoke Field.setExtent(int,int).
    - Recalculate the pixel layout, cached fonts, and locale strings.

```
protected void layout(int width, int height) {
    _font = getFont();
    _labelHeight = _font.getHeight();
    _labelWidth = _font.getAdvance(_label);
    width = Math.min( width, getPreferredWidth() );
    height = Math.min( height, getPreferredHeight() );
    setExtent( width, height );
}
```

- The manager of the field invokes layout() to determine how the field arranges its contents in the available space.

# Creating Custom Fields (Cont.)

- Define the preferred width of a custom component.
  - Implement getPreferredWidth(), using the relative dimensions to make sure that the label does not exceed the dimensions of the component.

```
public int getPreferredWidth() {
    switch(_shape) {
      case TRIANGLE:
        if (_labelWidth < _labelHeight) {
            return _labelHeight << 2;
         } else {
            return _labelWidth << 1;
        }
      case OCTAGON:
        if (_labelWidth < _labelHeight) {
            return _labelHeight + 4;
        } else {
            return _labelWidth + 8;
        }
      case RECTANGLE: default:
            return _labelWidth + 8;
    }
}
```

# Creating Custom Fields (Cont.)

- Define the preferred height of a custom component.
  - Implement getPreferredHeight(), using the relative dimensions of the field label to determine the preferred height.

```
public int getPreferredHeight() {
    switch(_shape) {
        case TRIANGLE:
            if (_labelWidth < _labelHeight) {
                return _labelHeight << 1;
            } else {
                return _labelWidth;
            }
        case RECTANGLE:
            return _labelHeight + 4;
        case OCTAGON:
            return getPreferredWidth();
    }
    return 0;
}
```

# Creating Custom Fields (Cont.)

- Define the appearance of the custom field.
  - Perform complex calculations in layout()instead of in paint().
  - Implement paint().

```
protected void paint(Graphics graphics) {
    int textX, textY, textWidth;
    int w = getWidth();
    switch(_shape) {
        case TRIANGLE:
            int h = (w>>1);
            int m = (w>>1)-1;
            graphics.drawLine(0, h-1, m, 0);
            graphics.drawLine(m, 0, w-1, h-1);
            graphics.drawLine(0, h-1, w-1, h-1);
            textWidth = Math.min(_labelWidth,h);
            textX = (w - textWidth) >> 1;
            textY = h >> 1;
            break;
```

# Creating Custom Fields (Cont.)

```
            case OCTAGON:
                int x = 5*w/17;
                int x2 = w-x-1;
                int x3 = w-1;
                graphics.drawLine(0, x, 0, x2);
                graphics.drawLine(x3, x, x3, x2);
                graphics.drawLine(x, 0, x2, 0);
                graphics.drawLine(x, x3, x2, x3);
                graphics.drawLine(0, x, x, 0);
                graphics.drawLine(0, x2, x, x3);
                graphics.drawLine(x2, 0, x3, x);
                graphics.drawLine(x2, x3, x3, x2);
                textWidth = Math.min(_labelWidth, w - 6);
                textX = (w-textWidth) >> 1;
                textY = (w-_labelHeight) >> 1;
                break;
            case RECTANGLE: default:
                graphics.drawRect(0, 0, w, getHeight());
                textX = 4;
                textY = 2;
                textWidth = w - 6;
                break;
        }
        graphics.drawText(_label, textX, textY, (int)( getStyle() & DrawStyle.ELLIPSIS |
        DrawStyle.HALIGN_MASK ), textWidth );
    }
```

- The fields manager invokes paint() to redraw the field whenever an area of the field is marked as invalid.

# Creating Custom Fields (Cont.)

- Paint a field only within the visible region.
  - Invoke Graphics.getClippingRect()
- Manage focus events.
  - Use the Field.FOCUSABLE style and implement Field.moveFocus()
- Change the appearance of the default focus indicator.
  - Override Field.drawFocus()

# Creating Custom Fields (Cont.)

- Add component capabilities.
  - Implement the Field set() and get() methods.

```
public String getLabel() {
    return _label;
}
public int getShape() {
    return _shape;
}
public void setLabel(String label) {
    _label = label;
    _labelWidth = _font.getAdvance(_label);
    updateLayout();
}
public void setShape(int shape) {
    _shape = shape;
    updateLayout();
}
```

# Creating Custom Context Menus

- Create the custom context menu items.
  - In your field class, create the custom context menu items.

```
private MenuItem myContextMenuItemA = new MenuItem( _resources,
    MENUITEM_ONE, 200000, 10) {
    public void run() {
      onMyMenuItemA();
    }
};

private MenuItem myContextMenuItemB = new MenuItem( _resources,
    MENUITEM_ONE, 200000, 10) {
    public void run() {
      onMyMenuItemB();
    }
};
```

# Creating Custom Context Menus (Cont.)

- Provide a context menu.
  - In your main BlackBerry® MDS Java Application class, override makeContextMenu().

```
protected void makeContextMenu(ContextMenu
    contextMenu) {
    contextMenu.addItem(myContextMenuItemA);
    contextMenu.addItem(myContextMenuItemB);
}
```

# Creating Custom Context Menus (Cont.)

- Create the BlackBerry® MDS Java Application menu.
  - In your main BlackBerry Java Application class, override Screen.makeMenu(), invoking Screen.getLeafFieldWithFocus() and Field.getContextMenu() on the return value to determine which fields receive custom menu items.

```
protected void makeMenu(Menu menu) {
    Field focus =
    UiApplication.getUiApplication().getActiveScreen().getLeafFieldWi
    thFocus();
    if (focus != null) {
            ContextMenu contextMenu = focus.getContextMenu();
            if (!contextMenu.isEmpty()) {
      menu.add(contextMenu);
      menu.addSeparator();
            }
    }
}
```

# Creating Custom Layout Managers

- Create a custom layout manager.
  - Extend the Manager class or one of its subclasses.

```
class DiagonalManager extends Manager {
    public DiagonalManager(long style){
      super(style);
    }
    ...
}
```

# Creating Custom Layout Managers (Cont.)

- Return a preferred field width.
  - Override getPreferredWidth() so that it returns the preferred field width for the manager.

```
public int getPreferredWidth() {
    int width = 0;
    int numberOfFields = getFieldCount();
    for (int i=0; i<numberOfFields; ++i) {
      width += getField(i).getPreferredWidth();
    }
    return width;
}
```

# Creating Custom Layout Managers (Cont.)

- Organize more than one TextField or Manager object.
  - Override the respective getPreferredWidth() methods for the TextField or Manager objects.
- Organize multiple TextFields horizontally.
  - Override layout().
- Return a preferred field height.
  - Override getPreferredHeight() so that it returns the preferred field height for the manager.

```
public int getPreferredHeight() {
    int height = 0;
    int numberOfFields = getFieldCount();
    for (int i=0; i<numberOfFields; ++i) {
      height += getField(i).getPreferredHeight();
    }
    return height;
}
```

# Creating Custom Layout Managers (Cont.)

- Specify the arrangement of the child fields.
  - Override sublayout() to retrieve the total number of fields in the manager.
  - Control how each child field is added to the screen by calling setPositionChild() and layoutChild() for each field that the manager contains.

```
protected void sublayout(int width, int height) {
    int x = 0;
    int y = 0;
    Field field;
    int numberOfFields = getFieldCount();
    for (int i=0; i<numberOfFields; ++i) {
       field = getField(i);
       layoutChild(field, width, height);
       setPositionChild(field, x, y);
       field.setPosition(x,y);
       x += field.getPreferredWidth();
       y += field.getPreferredHeight();
    }
    setExtent(width,height);
}
```

# Creating Custom Layout Managers (Cont.)

- Set the size of the child fields.
  - In sublayout(), invoke setExtent().
- Specify how the fields receive focus.
  - Override nextFocus().

```
protected int nextFocus(int direction, boolean alt) {
    int index = this.getFieldWithFocusIndex();
    if(alt) {
        if(direction < 0) {
            // action to perform if trackwheel is rolled up
        } else {
            // action to perform if trackwheel is rolled down
        }
    }
    if (index == this.getFieldWithFocusIndex()) {
        return super.nextFocus(direction, alt);
    } else {
        return index;
    }
}
```

# Creating Custom Layout Managers (Cont.)

- Repaint the fields when the visible region changes.
  - By default, the custom manager invokes paint() to repaint all of the fields without regard to the clipping region
  - Implement subpaint()

# Creating Custom Lists

- Let users select multiple items in a list.
  - Declare lists as MULTI_SELECT.

- Create a callback object.
  - Implement the ListFieldCallback interface.

```
private class ListCallback implements ListFieldCallback {
    // The listElements vector contain the entries in the list.
    private Vector listElements = new Vector();
    ...
}
```

# Creating Custom Lists (Cont.)

- Let the field repaint a row.
  - Implement ListFieldCallback.drawListRow(), invoking Graphics.drawText() using parameters that specify the row to paint.

```
public void drawListRow(ListField list, Graphics g, int
    index, int y, int w) {
    String text = (String)listElements.elementAt(index);
    g.drawText(text, 0, y, 0, w);
}
```

# Creating Custom Lists (Cont.)

- Let the field retrieve an entry from the list.
  - Implement ListFieldCallback.get().

```
public Object get(ListField list, int index) {
    return listElements.elementAt(index);
}
```

- Return a preferred width for the list.
  - In the implementation of getPreferredWidth(), return a preferred width for the list.

```
public int getPreferredWidth(ListField list) {
    return Graphics.getScreenWidth();
}
```

# Creating Custom Lists (Cont.)

- Assign the callback and add entries to the list.
  - Create the ListField and ListCallback objects for the list.

```
ListField myList = new ListField();
ListCallback myCallback = new ListCallback();
```

  - Invoke ListField.setCallback() to associate the ListFieldCallback with the ListField. This association lets the callback add items to the list.

```
myList.setCallback(myCallback);
```

  - To add entries to the list, create the entries, specify an index at which to insert each entry on the ListField object, and then insert each ListField object into the ListFieldCallback.

```
String fieldOne = new String("Field one label");
String fieldTwo = new String("Field two label");
String fieldThree = new String("Field three label");
myList.insert(0);
myList.insert(1);
myList.insert(2);
myCallback.insert(fieldOne, 0);
myCallback.insert(fieldTwo, 1);
myCallback.insert(fieldThree, 2);
mainScreen.add(myList);
```

# Complete Applications

- The BlackBerry API is capable of high-quality applications when you customize the UI components
- Example: Facebook for BlackBerry