CMER

Centre for Mobile Education and Research

BlackBerry Event Handling

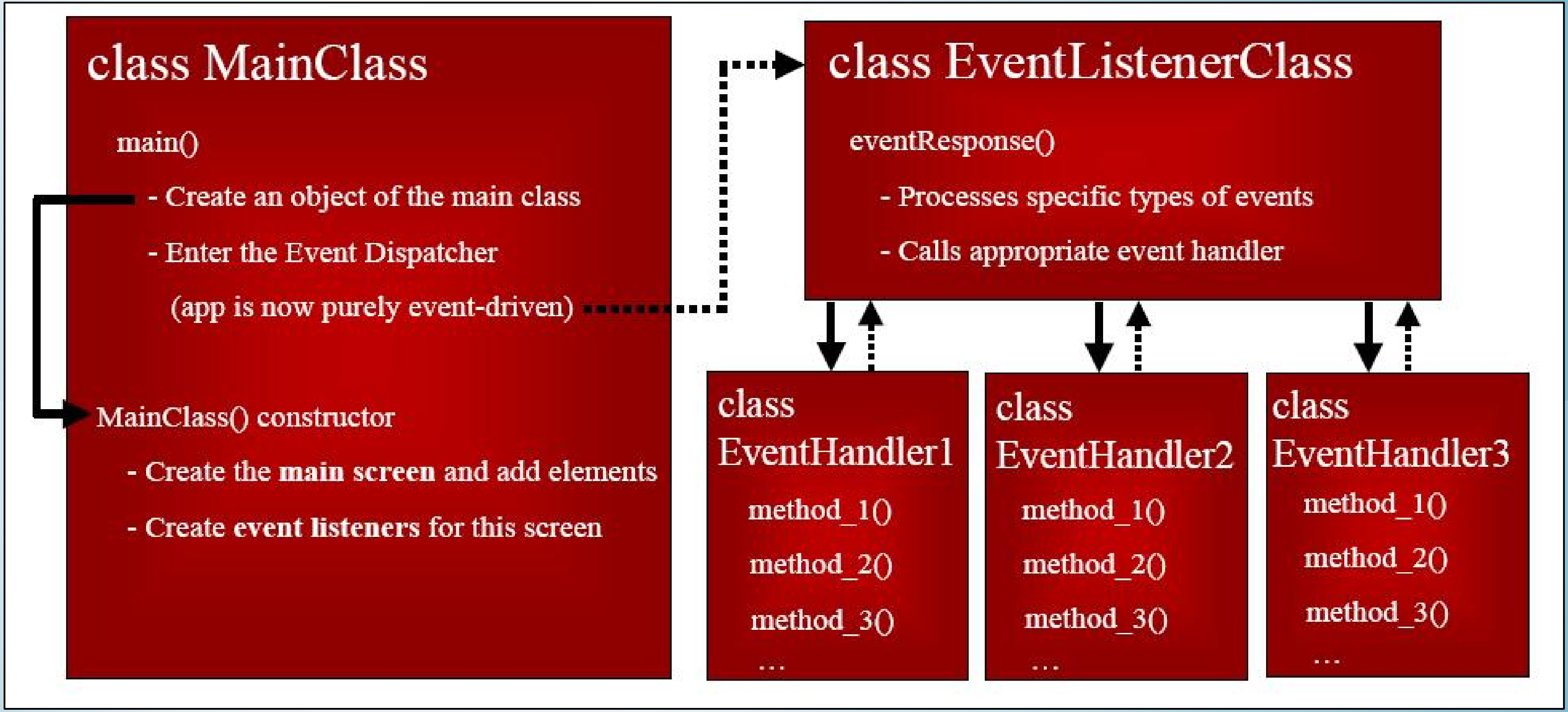
Overview

- Introduction
- Typical Application Model
- Event Listeners
- Responding to UI Events
- Touch Screen Events
- Touch Screen Gestures

Introduction

- Event handling deals with the interaction with UI components within an application.
- This is accomplished via BlackBerry event listeners.
- Similar implementation to event handling in Java.
- The net.rim.device.api.system package contains many useful listener classes.

Typical Application Model



class MainClass

main()

- Create an object of the main class
- Enter the Event Dispatcher
  (app is now purely event-driven)

MainClass() constructor

- Create the **main screen** and add elements
- Create **event listeners** for this screen

class EventListenerClass

eventResponse()

- Processes specific types of events
- Calls appropriate event handler

class EventHandler1

method_1()
method_2()
method_3()
…

class EventHandler2

method_1()
method_2()
method_3()
…

class EventHandler3

method_1()
method_2()
method_3()
…

Event Listeners

- Many types of listeners:
  - TrackwheelListener – ‘listens’ for trackwheel events
  - KeyListener – ‘listens’ for keyboard events
  - Others include:
    - AlertListener
    - GlobalEventListener
    - HolsterListener
    - IOPortListener
    - SystemListener
    - TouchEventListener
    - ChangeListeners (field, focus, scroll)
    - etc.
- See net.rim.device.api.system package for more.

TrackwheelListener

- A listener interface for receiving trackwheel events.
- Use of this interface is strongly discouraged now.
- Instead of using this interface, developers are strongly encouraged to use the "navigation" methods in the Screen class to receive such notifications.
  - This is done by extending the Screen class and providing custom implementations of the following 3 methods:
    - Screen.navigationClick(int, int)
    - Screen.navigationUnclick(int, int)
    - Screen.navigationMovement(int, int, int, int)

KeyListener

- The listener interface for receiving keyboard events.
- Used to handle events from the device hardware interface such as the keypad and trackwheel.
- Beneficial for game development where often the keys of the device become the controls.
- In order to support different locales in the future, apps should use the keyChar notification to determine which characters a user has pressed. Although in English there is a high correspondence between keys and characters, in other languages there might not be.
  - For example, using hirgana or katakana maps, it would often take two keys to generate one character

KeyListener (Cont.)

- keyChar(char key, int status, int time)
  - Invoked when a sequence of zero or more keyDowns generates a character.
- keyDown(int keycode, int time)
  - Invoked when a key has been pressed.
- keyRepeat(int keycode, int time)
  - Invoked when a key has been repeated.
- keyStatus(int keycode, int time)
  - Invoked when the ALT or SHIFT status has changed.
- keyUp(int keycode, int time)
  - Invoked when a key has been released.

KeyListener Example

- In this next example we will use the keypad and trackwheel to control specific functions of a typical game scenario.
- The five operations that can take place in this game scenario are:
  - Move Left
  - Move Right
  - Move Up
  - Move Down
  - Shoot
- Code sample provided on the next slide and will be discussed afterwards.

KeypadListener Example

```
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.KeypadListener;
import net.rim.device.api.system.KeyListener;

class KeypadListenerExample extends UiApplication {

   private static RichTextField command;

   KeypadListenerExample() {
      MainScreen mainScreen = new MyScreen();
      command = new RichTextField("Waiting for command...");
      mainScreen.add(command);
      pushScreen(mainScreen);
   }
```

KeypadListener Example (Cont.)

```
public static void main(String[] args) {
    KeypadListenerExample app = new KeypadListenerExample();
    app.enterEventDispatcher();
}

static class MyScreen extends MainScreen {
    public boolean keyChar(char key, int status, int time) {
        if(key == 'd'){
            command.setText("Move Left");
        }else if(key == 'j'){
            command.setText("Move Right");
        }else if(key == 't'){
            command.setText("Move Up");
        }else if(key == 'b'){
            command.setText("Move Down");
        }else if(key == 'g'){
            command.setText("Shoot!");
        }
        return true;
    }
```

KeypadListener Example (Cont.)

```
protected boolean navigationMovement(int dx, int dy, int status, int time) {
            if(dx < 0 && dy == 0) {
                command.setText("Move Left");
            } else if(dx > 0 && dy == 0) {
                command.setText("Move Right");
            } else if(dx == 0 && dy > 0) {
                command.setText("Move Up");
            } else if(dx == 0 && dy < 0) {
                command.setText("Move Down");
            }
            return true;
        }
        protected boolean navigationClick(int status, int time) {
            command.setText("Shoot!");
            return true;
        }
    }
}
```

KeypadListener Example Explained

- The keyChar method
  - Simply detects the character that was submitted to the device by comparing the key parameter.
  - The other input parameters are not needed for this occasion.
  - The status parameter can tell us information such as whether the shift or caps lock inputs are enabled.
  - The time parameter is the number of milliseconds since the device was turned on.
  - Our implementation of this method simply changes the RichTextField on the screen to read the command operation.
    - eg. If “d” was pressed then print out “Move left”
  - We return true because this informs that the event was consumed

KeypadListener Example Explained (Cont.)

- The navigationMovement method
  - Responds to the trackwheel events.
  - It takes a status and time parameter equivalent to the keyChar method previously discussed.
  - It also takes X and Y coordinates that specify the change in movement from the current position.
    - A positive X and Y value means right and down respectively.
    - A negative X and Y value means left and up respectively.
  - With this in mind we are able to detect the direction of the trackwheel and then specify the appropriate command.
  - We ignore X and Y values of zero which means that a roll of the trackwheel must be in the perfect direction (i.e left, right, up down) and that diagonal movements are not computed.
  - We return true to show that the event was consumed.

KeypadListener Example Explained (Cont.)

- The navigationClick method
  - Responds to the input from a trackwheel click.
  - Again, this method also takes a *status* and *time* parameter equivalent to the keyChar and navigationMovement methods previously discussed.
  - Invokes the “shoot” command after receiving a trackwheel click.
  - Like the previous methods we return true to indicate that the event has been consumed

AlertListener

- Provides functionality for receiving alert events.
- Useful in game development and media applications
- Use Application.addAlertListener(AlertListener) to receive notifications via this interface.
- audioDone(int reason)
  - Invoked when an audio alert ends.
- buzzerDone(int reason)
  - Invoked when a buzzer alert ends.
- vibrateDone(int reason)
  - Invoked when a vibrate alert ends.

GlobalEventListener

- The listener interface for receiving global events.
- Arbitrary applications may use global events for inter-process communication (IPC).
- The BlackBerry OS can also generate global events, such as those defined by the ServiceBook API.
- eventOccurred(long guid, int data0, int data1, Object object0, Object object1)
  - Invoked when the specified global event occurred.
  - The eventOccurred method provides two object parameters and two integer parameters for supplying details about the event itself. The developer determines how the parameters will be used.

GlobalEventListener (Cont.)

- For example, if the event corresponded to sending or receiving a mail message, the object0 parameter might specify the mail message itself, while the data0 parameter might specify the identification details of the message, such as an address value.
- Parameters:
  - guid - The GUID of the event.
  - data0 - Integer value specifying information associated with the event.
  - data1 - Integer value specifying information associated with the event.
  - object0 - Object specifying information associated with the event.
  - object1 - Object specifying information associated with the event.

HolsterListener

- The listener interface for receiving holster events.
- Useful in power management to increase battery life
- Implement this interface to listen for holster events, such as the insertion or removal of the BlackBerry device from the holster.
- inHolster()
  - Invoked when the device is put in the holster.
- outOfHolster()
  - Invoked when the device is removed from the holster.

IOPortListener

- The listener interface for receiving I/O port events.
- connected()
  - Invoked when the port is connected.
- dataReceived(int length)
  - Invoked when the port's receive queue has changed from empty to not empty.
- dataSent()
  - Invoked when the port's transmit queue becomes completely empty.
- disconnected()
  - Invoked when the port is disconnected.
- patternReceived(byte[] pattern)
  - Invoked when a registered pattern is received.
- receiveError(int error)
  - Invoked when a communication error has occurred.

SystemListener

- The listener interface for receiving system events.
- batteryGood()
  - Invoked when the internal battery voltage has returned to normal.
- batteryLow()
  - Invoked when the internal battery voltage falls below a critical level.
- batteryStatusChange(int status)
  - Invoked when the internal battery state has changed.
- powerOff()
  - Invoked when the user is putting the device into a power off state.
- powerUp()
  - Invoked when the device has left the power off state.

SystemListener

- The listener interface for receiving system events.
- Useful in developing application for accessories
- backlightStateChange(boolean on)
  - Invoked when the backlight state changes.
- cradleMismatch(boolean mismatch)
  - Invoked when a USB device has been placed in a serial cradle.
- fastReset()
  - Invoked when a fast reset occurs.
- powerOffRequested(int reason)
  - Invoked when the OS requests that the device power be turned off.
- usbConnectionStateChange(int state)
  - Invoked when the USB connection state changes.

Field Focus Changes

- The FocusChangeListener specifies what actions should occur when a field gains, loses, or changes focus.
- Implement FocusChangeListener to listen for field focus changes
- Your implementation of FocusChangeListener should specify what action occurs when the field gains, loses, or changes the focus by implementing focusChanged()
- Assign your implementation to a Field by invoking setChangeListener()
- Eg.

```
FocusListener myFocusChangeListener = new FocusListener();
myField.setFocusListener(myFocusChangeListener);
```

Field Focus Changes (Cont.)

Example focus listener class:

```
private class FocusListener implements FocusChangeListener {
    public void focusChanged(Field field, int eventType) {
            if (eventType == FOCUS_GAINED) {
                        // Perform action when this field gains the focus.
            }
            if (eventType == FOCUS_CHANGED) {
                        // Perform action when the focus changes for this
    field.
            }
            if (eventType == FOCUS_LOST) {
                        // Perform action when this field loses focus.}
            }
    }
}
```

Field Property Changes

- Similar to field focus change implementation
- Implement the FieldChangeListener interface.
- Assign your implementation to a field by invoking setChangeListener().
- Eg.

```
FieldListener myFieldChangeListener = new
FieldListener();
myField.setChangeListener(myFieldChangeListener);
```

Field Property Changes (Cont.)

```
private class FieldListener implements
    FieldChangeListener {
    public void fieldChanged(Field field, int context) {
            if (context != FieldChangeListener.PROGRAMMATIC)
    {
                        // Perform action if user changed field.
            } else {
                        // Perform action if application changed field.
            }
    }
}
// …
```

Responding to UI Events

- Manage navigation events by extending the net.rim.device.api.ui.Screen class (or one of its subclasses) and overriding the following navigation methods:
  - navigationClick(int status, int time)
  - navigationUnclick(int status, int time)
  - navigationMovement(int dx, int dy, int status, int time)
- Use the new Screen navigation methods and avoid using the TrackwheelListener

Responding to UI Events (Cont.)

- The status parameter of the navigation methods contains information about the event.
- To interpret this information, perform a bitwise AND operation on the status parameter in implementation of one of the navigationClick, navigationUnclick, or navigationMovement methods of the Screen or Field classes.
- See next slide for an example to determine the type of input mechanism that triggered an event.

Responding to UI Events (Cont.)

- In implementation of the navigationClick(int status, int time) method, create code such as the following:

```
public boolean navigationClick(int status, int time) {
        if ((status & KeypadListener.STATUS_TRACKWHEEL) ==
        KeypadListener.STATUS_TRACKWHEEL) {
                //Input came from the trackwheel
        } else if ((status & KeypadListener.STATUS_FOUR_WAY) ==
        KeypadListener.STATUS_FOUR_WAY) {
                //Input came from a four way navigation input device
        }
        return super.navigationClick(status, time);
}
```

Responding to UI Events (Cont.)

- Respond to BlackBerry® device user interaction
  - Use the Screen class and its subclasses to provide a menu for the BlackBerry device users to perform actions.
- Provide menu support
  - Extend the Screen class.

Responding to UI Events (Cont.)

- Provide screen navigation when using a FullScreen or Screen
  - Creating a MainScreen object provides default navigation to the application.
  - Avoid using buttons or other UI elements that take up space on the screen.
  - Specify the DEFAULT_MENU and DEFAULT_CLOSE parameters in the constructor to provide default navigation.

    FullScreen fullScreen = new FullScreen(DEFAULT_MENU | DEFAULT_CLOSE);

Responding to UI Events (Cont.)

- Provide menu support in an application that uses the TrackwheelClick() method of the TrackwheelListener
  - Use an extension of the Screen class.
  - In the constructor of the Screen class extension, invoke the Screen class constructor using the DEFAULT_MENU property.
  - Ensure that extension of the makeMenu() method of the Screen class invokes Screen.makeMenu() and adds the required menu items for the current UI application.

Responding to UI Events (Cont.)

- Manage selected menu items.
  - Two options
- Option 1
  - Override the onMenu()method.
  - In your extension of makeMenu() cache a reference to the “menu” arameter in the screen.
  - In your extension of OnMenu(), invoke Screen.OnMenu().
  - In your code, inspect the cached Menu object to determine which menu item the BlackBerry® device user selected.
  - Use the result of this inspection to trigger the appropriate menu action.

Responding to UI Events (Cont.)

- Option 2
  - Extend the MenuItem class.

    ```
    private MenuItem viewItem = new MenuItem("View
    Message", 100, 10);
    ```

  - Create a run() method that implements the behavior that you expect to occur when the BlackBerry device user clicks a menu item. When a BlackBerry device user selects a MenuItem, this action invokes the run() method.

    ```
    public void run() {
      Dialog.inform("This is today's message");
    }
    ```

  - If you do not use localization resources, override toString() to specify the name of the menu item.

Responding to UI Events (Cont.)

– When you add your own menu items, define a Close menu item explicitly.

```
private MenuItem closeItem = new MenuItem("Close", 200000, 10);
public void run() {
  onClose();
}
```

– To add the menu items to the screen, override Screen.makeMenu(), adding instances of the extended MenuItem class.

```
protected void makeMenu(Menu menu, int instance) {
  menu.add(viewItem);
  menu.add(closeItem);
}
```

– In your extension of the MenuItem class, do not override the onMenu()method.

Touch Screen Events

- New BlackBerry devices support touch screen events such as the Storm
- Can handle simple touch events:
  - Clicks, Up, Down, etc.
- Can also handle more complicated gestures:
  - Swipes, Hovering, etc.
- Classes:
  - net.rim.device.api.ui.TouchEvent
  - net.rim.device.api.ui.TouchGesture

Touch Screen Events

- Can be applied to:
  - Screens
  - Managers
  - Fields

Types of Touch Screen Events

Events

- Up
- Down
- Click
- Unclick
- Move
- Cancel

Gestures

- Tap
- Swipe North
- Swipe South
- Swipe East
- Swipe West
- Hover

Touch Screen Gestures

- Hover – Holding your finger on an item
- Swipe – Slide your finger from one point to another
- Tap – Touch the screen twice quickly