

## INTERACTION, EVENTS

Cristian Bogdan  
cristi@kth.se

DH2641 Interaction Programming

## “Kinds” of interaction

- User interacts with a widget (component), producing an event, this lecture
- Application signaling an event (timer, sensor value change, new data from the net) to the user (notifications, mentioned in this lecture)
- Multitouch and Gestures (mentioned in this lecture)
- Drag and Drop (and copy paste) within an application or between applications (coming lectures)
- Automatic appearance changes on window size/font/skin change (previous lecture)
- Interaction Design patterns (mentioned today)

DH2641 Interaction Programming

## A button event - HTML

Layout and event linking:

```
<button onClick="onClickAction()">Press me</button>
```

"Listener"

```
function onClickAction() {  
  alert("button was pressed");  
}
```

Only one listener per event type (like in Android)



DH2641 Interaction Programming


## A button event – Java Swing

Layout/initialization:

```
JButton b = new JButton("Press me");  
...  
b.addActionListener(new ButtonPressDetector());
```

Listener:

```
public class ButtonPressDetector  
  implements ActionListener{  
  public void actionPerformed(ActionEvent e){  
    System.out.println("button was pressed! " + e);  
  }  
}
```



DH2641 Interaction Programming

## A button event - Android

Layout:


```
<Button id="@+id/my_button"  
  android:layout_height="wrap_content"  
  android:layout_width="wrap_content"  
  android:text="@string/pressMe" />
```

Attaching listener:

```
Button b = (Button) findViewById(R.id.my_button);  
...  
b.setOnClickListener(new ButtonPressDetector());
```

Listener implementation:

```
public class ButtonPressDetector  
  implements View.OnClickListener{  
  public void onClick(View v){ ... }  
}
```



DH2641 Interaction Programming

## A button event - HTML – better way

Layout and event linking:

```
<button id="myButton">Press me</button>
```

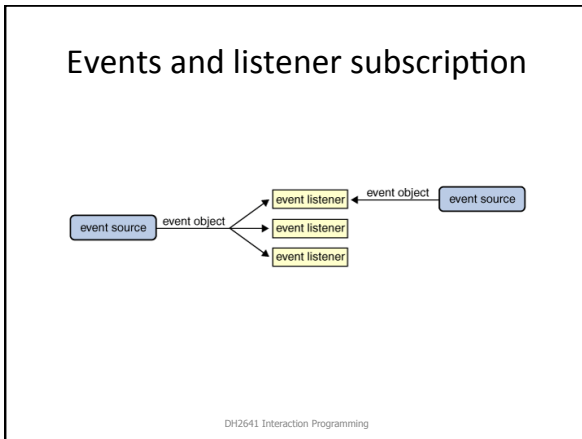
"Listener"

```
var onClickAction = function {  
  alert("button was pressed");  
}
```

```
$("#myButton").click(onClickAction);
```



DH2641 Interaction Programming



### Events at different levels

- Example: a button press can originate from one or more low-level events
  - Swing: MouseEvent (MouseListener), KeyEvent (KeyListener)
    - MouseEvent: mouse pressed + mouse released= mouse clicked
    - -> ActionEvent
  - Android: MotionEvent -> View.OnClickListener
  - HTML onClick, or onKeyDown followed by onKeyUp -> onClick
- Other examples (Swing high-level event shown):
  - moving the cursor of a textbox (ChangeEvent)
  - changing the keyboard focus (FocusEvent)
  - scrolling a textbox (AdjustmentEvent)
  - closing a window (WindowEvent)

DH2641 Interaction Programming

### Event source

- Swing event.getSource() returns the Object that produced the event
  - can be compared using == to known widgets
  - can be cast to the known origin type to get more information

```
((JButton)event.getSource()).getText()
```
- Android: the source View is passed as first parameter of all Listener methods
- HTML: pass “this” as parameter
 

```
<button onClick="clicked(this)" > ...</button>
```

DH2641 Interaction Programming

### Event recipients

- A class can implement many listeners
  - implements ActionListener, MouseListener
- A listener object can listen to more sources

```

    ButtonPressDetector detector=
        new ButtonPressDetector();
    b1.addActionListener(detector);
    b2.addActionListener(detector);
  
```

DH2641 Interaction Programming

### Key Low level

	onkeydown, onkeypress, onkeyup
	.keydown(); .keypress(); .keyup();
	javax.awt.event.KeyListener keyPressed(), keyReleased(), keyTyped()
	android.view.View.OnKeyListener onKey()

DH2641 Interaction Programming




### Mouse / Touch Low level

	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmouseout, onmousemove
	.click(), .dblclick(), .mousedown(), .mouseup(), .mouseover(), .mouseout() ...
	javax.awt.event.MouseListener mouseClicked() = mousePressed() + mouseReleased() mouseEntered(), mouseExited()
	android.view.View.OnClickListener    onClick() android.view.View.OnLongClickListener    onLongClick() android.view.View.OnTouchListener    onTouch()

DH2641 Interaction Programming

**Low level**





## (Mouse) Motion

	<code>ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop</code>
	<code>javax.awt.event.MouseMotionListener</code> <code>mouseDragged(), mouseMoved()</code>
	<code>android.view.View.OnDragListener</code> <code>onDragEvent()</code>

DH2641 Interaction Programming

**Mid level**





## (Keyboard) Focus

 	<code>onfocus, onblur</code> <code>.focus(), .blur(), .focusin(), .focusout()</code>
	<code>javax.awt.event.FocusListener</code> <code>focusGained(); focusLost()</code>
	<code>android.view.View.FocusChangeListener</code> <code>onFocusChange()</code>

DH2641 Interaction Programming

**High level**






## (Text) Change

 	<code>onchange</code> <code>.change()</code>
	<code>javax.awt.event.TextListener</code> <code>textValueChanged()</code>
	<code>android.text.TextWatcher</code> <code>onTextChanged(); afterTextChanged(); beforeTextChanged();</code>

DH2641 Interaction Programming

**High level**

## HTML specific

	<code>onselect</code> <code>.select()</code>	Selection of text in an element
	<code>onresize</code> <code>.resize()</code>	Size of element changes
	<code>onload</code> <code>.load()</code>	Object has loaded. Can be used for the whole page, but better use jQuery .ready()
	<code>onscroll</code> <code>.scroll()</code>	Scrolling to a different place in the element
	<code>onunload</code> <code>.unload()</code>	Before the browser closes the document

DH2641 Interaction Programming

**High level**

## Java specific

<code>javax.awt.event.ActionListener</code> <code>javax.awt.event.AdjustmentListener</code> <code>javax.awt.event.ItemListener</code>	<code>actionPerformed()</code> <code>adjustmentValueChanged()</code> <code>itemStateChanged()</code>
<code>javax.awt.event.WindowListener</code>	<code>windowActivated()</code> <code>windowClosed()</code> <code>windowOpened()</code> ...
<code>javax.awt.event.ComponentListener</code> <code>javax.awt.event.ContainerListener</code>	<code>componentHidden()</code> <code>componentMoved()</code> <code>componentResized()</code> <code>componentShown()</code> <code>componentAdded()</code> <code>componentRemoved()</code>

DH2641 Interaction Programming

**High level**

## Android specific

<code>android.hardware.SensorListener</code>	<code>onAccuracyChanged()</code> <code>onSensorChanged()</code>
<code>android.view.GestureDetector.OnGestureListener</code>	<code>onDown()</code> <code>onFling()</code> <code>onScroll()</code> ...
<code>android.view.View.OnCreateContextMenuListener</code>	<code>onCreateContextMenu()</code>

and many, many, many more...

DH2641 Interaction Programming

### One Java class for each event??

- Listener code (controller) should be separated from layout code (see next lecture)
- In any Java (Android, Swing...) ...
- A class can implement many interfaces, listening to many kinds of events
- A listener object can subscribe (addListener) to many event sources
- Still, listening to one single event type for one single source can be achieved by defining an **anonymous inner class**

```
b.addListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){...}
}); // end of addListener() call
```

DH2641 Interaction Programming

### Event consumption

- Low-level events can be consumed
- So they will not lead to higher-level events!
- Example: avoiding text being typed in a textbox
- Swing: event.consume()
- Android Long click, Key and Touch event can be consumed by returning true in their handler methods

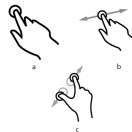
```
public class KeyConsumer implements View.KeyListener{
    public boolean
    onKeyDown (View v, Editable text, int keyCode, KeyEvent
    event ){
        ... return true; }
    // ... other KeyListener methods
}
```

DH2641 Interaction Programming

### Android Multitouch interaction

<http://www.zdnet.com/blog/burnette/how-to-use-multi-touch-in-android-2/1747>

- (a) Tap, (b) drag, (c) pinch zoom
- Multitouch is detected via the [View.OnTouchListener](#) public boolean onTouch(View v, [MotionEvent](#) event)
- [MotionEvent.getAction\(\)](#) type of action
  - ACTION\_DOWN first finger down
  - ACTION\_POINTER\_DOWN 2<sup>nd</sup>, 3<sup>rd</sup> finger down
  - ACTION\_MOVE move any number of fingers
  - ACTION\_POINTER\_UP one finger up
  - ACTION\_UP last finger up
- [MotionEvent.getPointerCount\(\)](#) returns the number of fingers
- [MotionEvent.getPointerId\(int\)](#) returns the pointer (finger)
- [MotionEvent.getX\(int\)](#), [getY\(int\)](#) accept an index argument to see where the respective finger is



DH2641 Interaction Programming

### Android design patterns

- Define “standard” ways of [designing interaction](#) on Android
- For consistency and predictability within and between apps
- And possible implementations
- E.g. the “Swipe view” pattern and [implementation](#)
- [Gesture pattern](#) and [implementation](#)
- Many other patterns

DH2641 Interaction Programming

### WHEN EXISTING COMPONENTS ARE NOT ENOUGH

DH2641 Interaction Programming

### Drawing your own component

- All Swing components are “lightweight” (drawn in Java)
  - you can make your own, subclass of JComponent
- <http://java.sun.com/products/jfc/tsc/articles/painting>
- Implement the paint (Graphics g) method
  - called automatically when the parent becomes visible
  - never call paint()! You can ask for repaint()

DH2641 Interaction Programming

## Drawing your own component

- You can override the paint() inline

```
new JComponent(){ // or JButton or whatever
    public void paintComponent(Graphics g){...}
};
```
- To draw on top of the “normal” appearance (if any), call super.paintComponent() first
- Swing paint() calls paintComponent(), paintBorder(), and paintChildren()
  - You can override paint() directly but make sure that super.paint() is called, otherwise children will not be painted!

DH2641 Interaction Programming

## Graphics

- Change colors: foreground, background
- Draw
  - shapes: polyline, eclipse, ... (and fill them)
  - images from various sources
  - text
- Clipping rectangle (if only a part needs to be drawn)
- More details about the concepts: the DGI course (DH2323)
- Can be cast to Graphics2D for more functionality
  - Graphics2D g2=(Graphics2D)g;

DH2641 Interaction Programming

## Special components in Swing

- Transparent components
  - Set the `opaque` property to false
- Non-rectangular components
  - implement the needed processXXX and addXXXListener
  - contains() for non-rectangular shapes
  - as usual, paint() to draw your own component shape
  - if the component is opaque, paint() will need to cover all the area for which contains() is true

DH2641 Interaction Programming

## Drawing in Android

- Android supports both 2D and 3D graphics.
- Override the onDraw(Canvas) method of any View
- Canvas offers similar functionality to Swing Graphics

<http://developer.android.com/guide/topics/graphics/index.html>

DH2641 Interaction Programming

## HTML5 canvas

```
<canvas id="myCanvas" width="200" height="100" style="border:1px solid #c3c3c3;">
Your browser does not support the canvas element.
</canvas>
```

```
<script type="text/javascript">
```

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
var grd=ctx.createLinearGradient(0,0,175,50);
grd.addColorStop(0,"#FF0000");
grd.addColorStop(1,"#00FF00");
ctx.fillStyle=grd;
ctx.fillRect(0,0,175,50);
```

```
</script>
```



DH2641 Interaction Programming

## NOTIFICATIONS


DH2641 Interaction Programming

### Notifications

```

// Let's check if the browser supports notifications
if (!("Notification" in window)) {
  alert("Browser doesn't support desktop notification");
}

// Let's check if the user is okay to get notification
else if (Notification.permission === "granted") {
  // If it's okay let's create a notification
  var notification = new Notification("Hi there!");
}
else {
  Notification.requestPermission(function (permission) {
    // Handle the permission
  });
}
    
```



DH2641 Interaction Programming

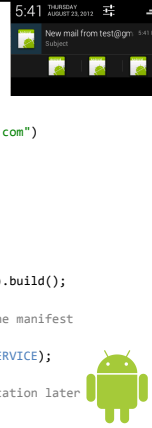

### Notifications

```

// Build the notification
Notification n = new Notification.Builder(this)
    .setContentTitle("New mail from " + "test@gmail.com")
    .setContentText("Subject")
    .setSmallIcon(R.drawable.icon)
    .setContentIntent(pIntent)
    .setAutoCancel(true)
    // add extra buttons
    .addAction(R.drawable.icon, "Call", pIntent)
    .addAction(R.drawable.icon, "More", pIntent)
    .addAction(R.drawable.icon, "And more", pIntent).build();

// Get the manager, don't forget to ask permission in the manifest
NotificationManager notificationManager =
    (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

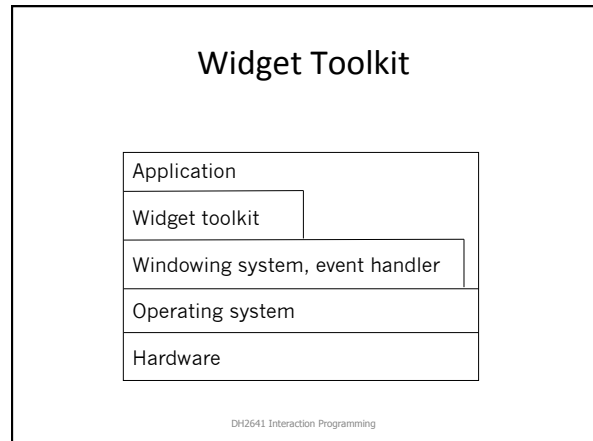
// Notify, pass an id (0), so you can cancel the notification later
notificationManager.notify(0, n);
    
```

DH2641 Interaction Programming

## TOOLKITS AND FRAMEWORKS

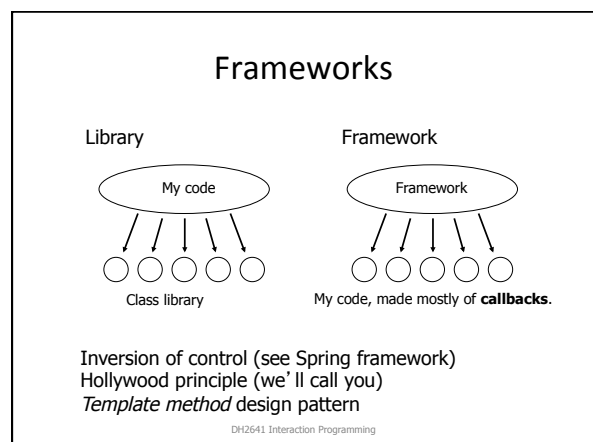
DH2641 Interaction Programming



### Frameworks

- Toolkits are like normal libraries
  - Application runs and calls the toolkit when needed
- Experience shows that it is hard to manage toolkits
- If everything is allowed, applications can easily become inconsequent
- A **framework** enforces a certain way of working
- Examples
  - Java Swing
  - IBM/Eclipse SWT
  - Microsoft MFC
  - Microsoft .NET

DH2641 Interaction Programming



### Frameworks

- The most important types of Swing callbacks
  - event listener methods like actionPerformed()
  - paint() method
  - Other callbacks in Java: finalize(), Observer's notify()
- Fundament:
  - we implement the callbacks, but never call them.
  - we wait for the framework to call them.

DH2641 Interaction Programming

### USER INPUT ARCHITECTURES

DH2641 Interaction Programming

### Input strategies

- Events are the current stage in an evolution
  - Sampling
  - Interrupt
  - Event
- Some input devices can only be read through sampling (and maybe interrupts)
  - Very old, or recent/experimental
- Events require multitasking

DH2641 Interaction Programming

### Sampling architectures

- AKA polling
- the application needs to ask whether there are changes in the input
- read/write data and continue execution
- leads often to active loops in the applications

DH2641 Interaction Programming

### Interrupt based architectures

- the application registers to be notified when a device state changes, and it provides an interrupt handler procedure (callback)
- the operating system calls the procedure when the device state changes
- leads to complicated asynchronous situations

DH2641 Interaction Programming

### Event-based architectures

- the operating system takes care of the changes in the device states
- an **event object (instance)** is generated for every state change
  - contains information about the device change (e.g. the left mouse button was pressed at time T at the 100, 200 coordinates)
- the applications register (**subscribe**) methods to be called when a certain event occurs (**callback**, see Inversion of Control later)

DH2641 Interaction Programming

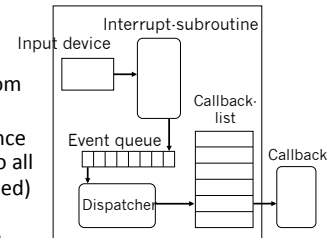
### Event-based architectures (cont)

- after its generation, the event is placed in an event queue
- in another thread, an event **dispatcher** takes the events from the queue and calls the registered (subscribed) callback routines

DH2641 Interaction Programming

### Event architecture

- The system places changes in a queue
- The framework consumes events from the queue regularly
- Send an event instance with event details to all interested (subscribed) parties (dispatch)
- Each subscribed party will register a callback



DH2641 Interaction Programming

If we have time

### UML

DH2641 Interaction Programming

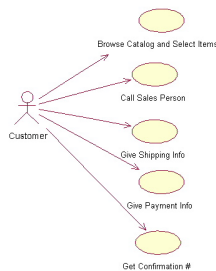
### UML

- Both a software design process and a series of diagram types
- Developed by the Rational company in the mid 1990s
- Standardized today and used overall in the industry
- Gives very limited information on how the user interfaces will look like

DH2641 Interaction Programming

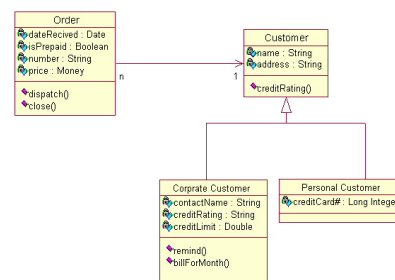
### UML: Use case diagrams

- An abstraction of one or more processes: a **scenario**.
- Shows **actors** and **use cases**, i.e. what actors do.
- One or more scenarios may be generated from a use case



DH2641 Interaction Programming

### UML: Class diagram

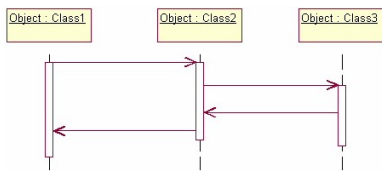


DH2641 Interaction Programming



### UML: Interaction diagram

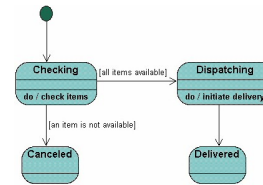
- Shows how messages (method calls) are sent between objects in the system
- Gives an overview of the system processes



DH2641 Interaction Programming

### UML: State diagram

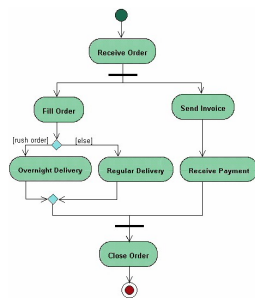
- Shows how the system moves between different states



DH2641 Interaction Programming

### UML: Activity diagram

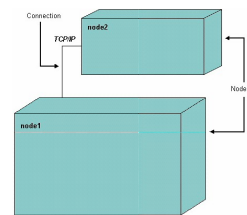
- Like the state diagram but focuses on activity
- An activity is not necessarily a state.
- Can correspond to features that the user activates



DH2641 Interaction Programming

### UML: Physical diagram

- Shows relations between hardware and software in the system
- Can also be used to show how the system is distributed between different machines



DH2641 Interaction Programming