

1 Example Technology: Macromedia Flash & ActionScript

1.1 Multimedia authoring tools - Example Macromedia Flash

1.2 Elementary concepts of ActionScript

Scripting in General + „History“ of ActionScript

Objects and Types in ActionScript

Animation with ActionScript

1.3 Interaction in ActionScript

1.4 Media classes in ActionScript

File Types in Flash Development

- Flash Project (.flp)
 - Bundles the information required for a specific development project
 - Easily readable XML file
 - Mainly: Links to involved files
- Flash Movie (.fla)
 - Contains the main animation (timelines and symbols)
 - Binary file, difficult to understand
 - Edited with the Flash authoring environment
- ActionScript (.as)
 - Contains an ActionScript class
 - Readable ActionScript ASCII file
 - Editable with any editor or with the built-in ActionScript editor of the Flash authoring environment
- Shockwave Flash (.swf)
 - Output format for Flash Player

Objects in Flash

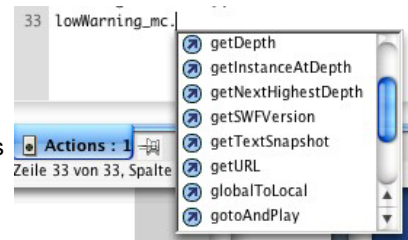
- Everything is an object.
- *Visual objects*: Can be created and manipulated in the graphical authoring environment (but also in other ways):
 - Objects of classes MovieClip, Button, TextField, Component, ...
 - Example: MovieClip object
 - » Has a TimeLine object where the class TimeLine defines methods like: `play()`, `stop()`, `gotoFrame()`
 - Dynamic creation of visual objects via method call
 - » Using specific methods like `createEmptyMovieClip`, `duplicateMovieClip`, `attachMovie`, ...
- *Non-visual objects*:
 - In particular objects of most developer-defined classes (“custom classes”)
 - Explicit instantiation
 - » Script contains new-statement like in Java
 - Example: “Account” objects

Strong vs. Weak Typing

- Weak Typing:
 - Variables and properties can be assigned different types of data at different times
 - Variables are declared without explicit type information
 - Example programming languages: BASIC, ActionScript 1.0
- Strong Typing:
 - Type information part of the variable declaration
 - All assigned values have to conform to the declared type at all time
 - Example programming languages: PASCAL, Java, ActionScript 2.0 (partially)
- Suffixing:
 - Only way in AS1 to get “code hinting”
 - See next slide

Type Hinting

- Naming convention for variables according to type of contained value
- Helpful mainly for weakly typed languages
 - “Hungarian notation” also used in C/C++, e.g. Microsoft standard
- Specific prefix or suffix of variable name indicates type
 - E.g. “variable names starting with ‘p’ indicate pointer values.”
 - E.g. “variable names ending with ‘_mc’ indicate MovieClip values“
- Information evaluated e.g. in programming environment
 - “Hinting” = interactive offer of adequate additions to currently edited programming text
 - For a variable named `xy_mc`, the special methods available for `MovieClip` objects are offered for selection



Types in ActionScript 2.0

- Types (= classes) for non-visual objects:
 - Array
 - Boolean
 - Number
 - Object
 - String
 - ...
 - + custom classes defined by the developer using `class { ... }`
 - Types (= classes) for visual objects:
 - MovieClip
 - Button
 - TextField
 - Component
- For visual objects, type information by suffixing is recommended !

A Full List of ActionScript 2.0 Data Types

- Accordion*
- Alert*
- Array
- Binding*
- Boolean
- Button**
- Camera**
- CheckBox*
- Color
- ComboBox*
- ComponentMixing*
- CustomActions*
- DataField*
- DataGrid*
- DataHolder*
- DataSet*
- DataType*
- Date
- DateChooser*
- Delta*
- DeltaItem*
- DeltaPacket*
- Endpoint*
- Error*
- Function**
- Label*
- LoadVars**
- LocalConnection**
- Log*
- MediaController*
- MediaDisplay*
- MediaPlayer*
- Menu*
- MenuBar*
- Microphone**
- MovieClip
- MovieClipLoader*
- NetConnection**
- NetStream**
- Number
- Object
- PendingCall*
- PopUpManager*
- PrintJob*
- ProgressBar*
- RadioButton*
- RadioButtonGroup*
- RDBMSResolver*
- ScrollPane*
- SharedObject**
- Slide*
- SOAPCall*
- Sound
- String
- TextArea*
- TextField**
- TextFormat**
- TextInput*
- TextSnapshot*
- Tree*
- TypedValue*
- Video**
- Void*
- WebServiceConnector*
- Window*
- XML
- XMLConnector*
- XMLNode
- XMLSocket
- XUpdateReceiver*

no sign = already contained in Flash 5 * = added in Flash MX ** = added in Flash MX 2004

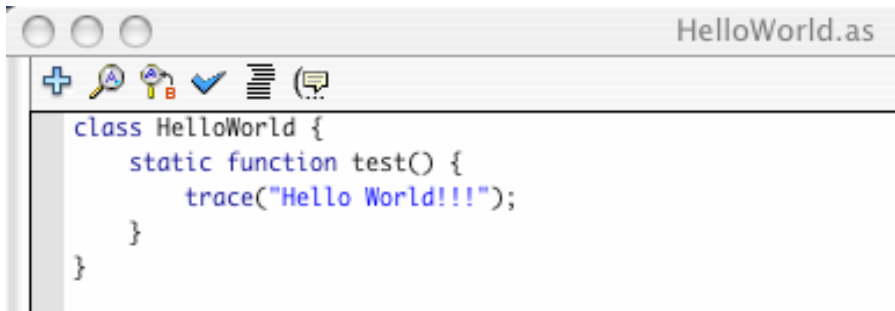
Type-hinting suffixes in ActionScript 2.0

Array: _array
Button: _btn
Camera: _cam
Color: _color
Date: _date
Error: _err
LoadVars: _lv
LocalConnection: _lc
Microphone: _mic
MovieClip: _mc
NetConnection: _nc
Sound: _sound
String: _str
TextField: _txt
Video: _video
XML: _xml
XMLNode: _xmlnode

Partial list !

A HelloWorld Program in ActionScript

- ActionScript class in file "HelloWorld.as"

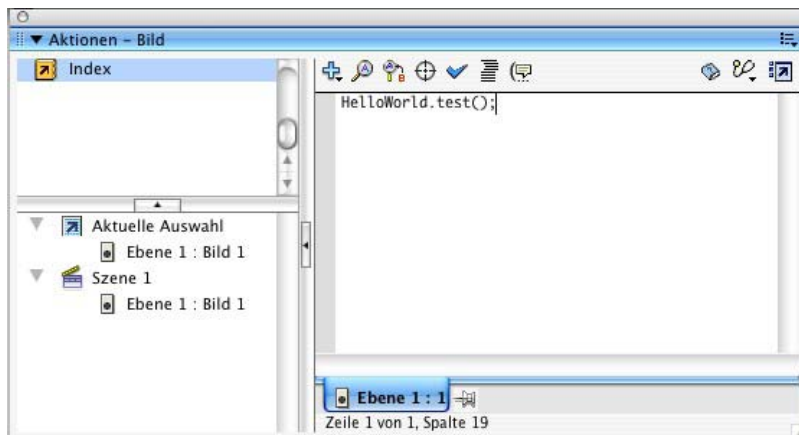


```
class HelloWorld {  
    static function test() {  
        trace("Hello World!!!");  
    }  
}
```

- **trace()**
 - Built-in function
 - Reports a message during runtime on the output console
 - Works only if debugger is present

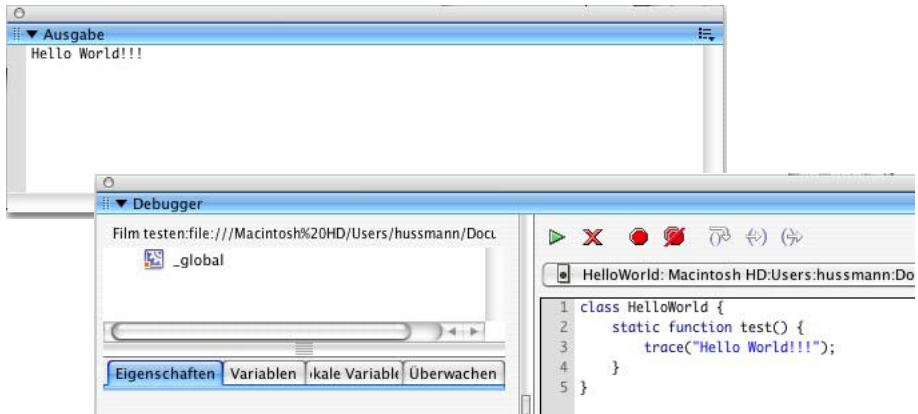
A Flash Movie Invoking the Hello World Program

- Flash movie "HelloWorld.fla"
 - Without any visible objects
 - ActionScript attached to Frame 1 of Scene 1



Running the Flash Hello World Movie

- Export as SWF file and start player
- Optional interactive debugger

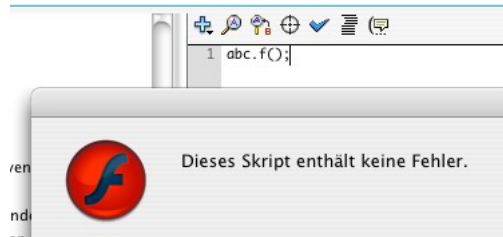


Flash Pattern: Start Frame Code

- **Problem:** A Flash movie needs to carry out some ActionScript code which cannot be easily defined in a local, object-oriented style
 - Creation of objects on an application-global scale
 - Invocation of methods defined in external “.as” files
 - Assignment of methods to visible objects instantiated from the standard library (e.g. TextField)
- **Solution:**
 - Keep the “global code” in the main timeline (`_root`).
 - Add a separate layer (e.g. “code” or “actions”) to the main timeline.
 - Add all “global” code to frame 1 of the newly created layer of the main timeline.
 - Advantage: There is just one place to inspect for the global code organisation.
- **Examples:**
 - Plenty found in literature

Undefined Variables & Methods in ActionScript

- Not recognized as errors:
 - Referencing an undefined variable
 - Calling a method not defined in the class/type of a variable



- Purpose of “sloppy” definition/typing rules in scripting languages for authoring systems:
 - Product can be tested and presented even in incomplete state
 - Danger: Error detection by tool checks (eg type check) does not work properly any more

Modifying Attributes in ActionScript

- All visible objects come with a predefined (more or less large) set of attributes
 - Example: “_x” and “_y” for screen position
- ActionScript code can e.g. move visible objects around the screen by modifying these attributes
- Example:
 - Modifying an object (with an independent timeline)
 - In Frame 1 (key frame): `inst_mc._x = 10; inst_mc._y = 10;`
 - In Frame 6 (key frame): `inst_mc._x = 20; inst_mc._y = 20;`
 - In Frame 11 (key frame): `inst_mc._x = 40; inst_mc._y = 40;`

Example RVML: Nested Timelines, ActionScript

```
...
<Definitions>
  <MorphShape id='inst_mc.MorphShape_1'> ...
</MorphShape>
  <MovieClip id='inst_mc'>
    <Timeline frameCount='5'>
      <Frame frameNo='1'>
        <Place name='inst_mc.MorphShape_1' depth='1' />
      </Frame>
      ...</Timeline>
    </MovieClip>
  </Definitions>
<Timeline frameCount='11'>
  <Frame frameNo='1'>
    <Place name='inst_mc' depth='1' instanceName='inst mc'>
      <Transform translateX='199.0' translateY='98.0' 7>
    </Place>
    <FrameActions><![CDATA[
inst_mc._x = 10;
inst_mc._y = 10;
]]></FrameActions>
  </Frame>
  ...
```

1 Example Technology: Macromedia Flash & ActionScript

1.1 Multimedia authoring tools - Example Macromedia Flash

1.2 Elementary concepts of ActionScript

Scripting in General + „History“ of ActionScript

Objects and Types in ActionScript

Animation with ActionScript

1.3 Interaction in ActionScript

1.4 Media classes in ActionScript

Animation as Attribute Modification

- Animation:
 - Modification of object attributes dependent on time / current frame
- Questions:
 - How to flexibly react on progress of time?
 - » Special events
 - How to program time-dependent code?
 - » Absolute computation of position
 - » Relative computation of position
- Most multimedia runtime systems have a notion of an event marking progress of time
 - Timer objects
 - Global clock
- **ActionScript**:
 - Special clip event **EnterFrame** is fired regularly at specified frame rate of the movie

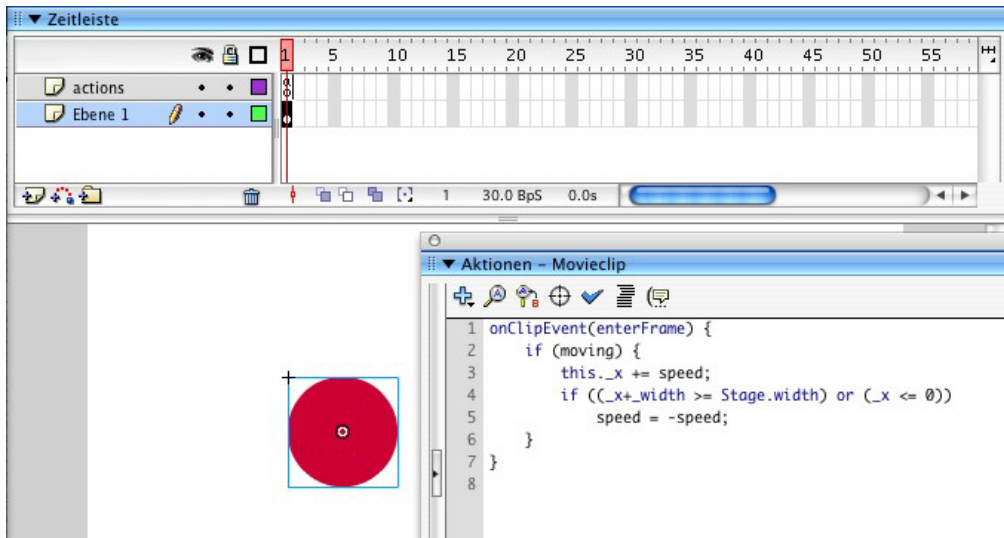
Events in ActionScript

- Clip events (affecting a whole movie clip):
 - Load
 - Unload
 - EnterFrame
 - Mouse...
 - Key..
 - Data

onClipEvent (...)
- Interaction events (caused by specific interaction objects, e.g. buttons):
 - Press
 - Release
 - ReleaseOutside
 - RollOut, RollOver
 - DragOut, DragOver
 - KeyPress

on (...)

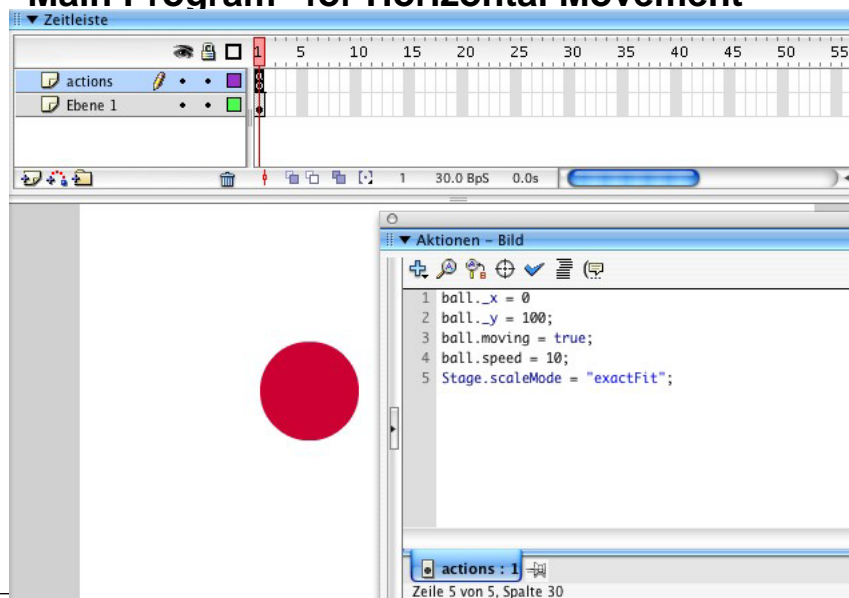
Horizontal Movement with EnterFrame-Events



The screenshot shows an animation software interface. At the top, there is a timeline labeled "Zeitleiste" with a scale from 0 to 55. Below the timeline, there are two layers: "actions" and "Ebene 1". The main workspace displays a red circle with a blue bounding box around it. To the right, a window titled "Aktionen - Movieclip" contains the following ActionScript code:

```
1 onClipEvent(enterFrame) {  
2     if (moving) {  
3         this._x += speed;  
4         if ((_x+_width >= Stage.width) or (_x <= 0))  
5             speed = -speed;  
6     }  
7 }  
8
```

“Main Program” for Horizontal Movement



The screenshot shows the same animation software interface. The main workspace displays a red circle. To the right, a window titled "Aktionen - Bild" contains the following ActionScript code:

```
1 ball._x = 0;  
2 ball._y = 100;  
3 ball.moving = true;  
4 ball.speed = 10;  
5 Stage.scaleMode = "exactFit";
```

At the bottom of the interface, there is a status bar that reads "aktionen : 1" and "Zeile 5 von 5, Spalte 30".

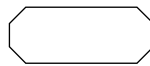
Visual Objects and Program Objects

Visual object
Manipulated with
Authoring system



```
class Xy  
new Xy
```

Program object
Written in
Script language

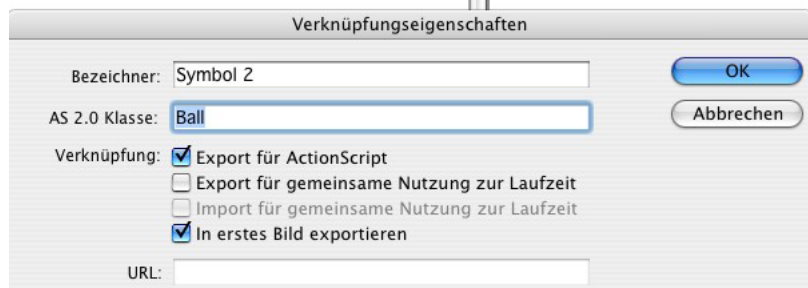
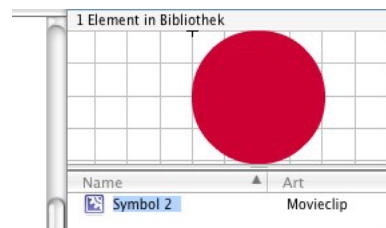


Joint abstraction:
"the object"

Has visual properties
Has program-defined properties

Flash: Linking AS2 Classes to Symbols

- In Flash, a symbol can be associated with a class by a special dialogue
 - "Linkage" / Verknüpfung



ActionScript 2 Class for Movement Example

```
class Ball extends MovieClip {
    public var speed:Number = 0;
    public var moving:Boolean = false;

    public function onEnterFrame() {
        if (moving) {
            this._x += speed;
            if ((_x+_width >= Stage.width) or (_x <= 0))
                speed = -speed;
        }
    }
}
```

Equivalent event handler declarations:

- attached to the object with generic keywords **on** and **onClipEvent**
- separate *callback* method (naming convention)

More powerful:

- listeners (see below)

Adding Vertical Movement

```
class Ball1 extends MovieClip {
    public var speed:Number = 0;
    public var jump:Number = 0;
    public var moving:Boolean = false;
    public var toRight = true;
    public var inLeftHalf:Boolean;

    public function onEnterFrame() {
        if (moving) {
            this._x += speed;
            if ((_x+_width >= Stage.width) or (_x <= 0)) {
                speed = -speed;
                toRight = !toRight;
            };
            inLeftHalf = (_x+_width)*2 <= Stage.width;
            if ((inLeftHalf && toRight) ||
                (!inLeftHalf && !toRight))
                _y -= jump;
            else
                _y += jump;
        }
    }
}
```

Absolute vs. Relative Movement Calculation

- Absolute calculation
 - Based on some base index
 - » Frame count, time, relative position on stage, ...
 - Base index to be provided by the programmer
 - » `_currentframe`, `_totalframe` etc. provide statically defined information
 - “Save” in terms of predictability of the effect
- Relative calculation
 - Based on most recent frame (“differential programming”)
 - Often easier (see example)
 - More flexible for changing situations
 - Problem: Rounding errors and other algorithmic problems may lead to unexpected effects (see example)

1 Example Technology: Macromedia Flash & ActionScript

1.1 Multimedia authoring tools - Example Macromedia Flash

1.2 Elementary concepts of ActionScript

Scripting in General + „History“ of ActionScript

Objects and Types in ActionScript

Animation with ActionScript

1.3 Interaction in ActionScript

Handling of Mouse Events

Classical Model-View-Controller Programming

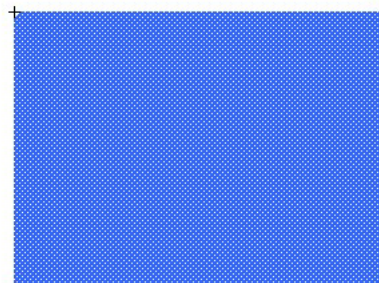
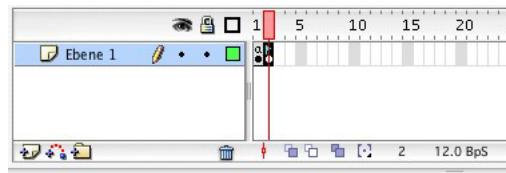
1.4 Media classes in ActionScript

What's Specific for an Animated (Flash) Interface?

- Traditional user interface elements:
 - Buttons, Text Fields, Menus, ...
 - All available also in Flash and other modern multimedia interface tools
- Animation in user interfaces:
 - Graphical feedback illustrating program actions
 - » E.g. direction of money transfer, strong warning: animation clips
 - Direct feedback “on touching”
 - » E.g. change of graphical representation on “mouse over”
- Direct interaction:
 - Drag and drop
 - Drawing-like actions
- Everything (in principle) realisable also by “normal” programming languages! (But often much more complex.)

Example: Highlighting a Region on “RollOver”

- Graphical element with AS event handler for “RollOver” event
 - E.g. changing the colour of a box
- “Traditional” solution with the Flash authoring tool:
 - Create a symbol with different key frames
 - Create an instance with an event handler switching between key frames



Event Handler for Frame Switching

```
on(rollOver) {  
    gotoAndStop("on");  
}  
on(rollOut) {  
    gotoAndStop("off");  
}
```

"on" and "off" are labels for the key frames of the symbol.
Not to be forgotten: `stop()` in first frame.

Flash Pattern: Graphical Response

- **Problem:** Dependent on some application-internal condition, we would like to show the user what the current status is, by selection among different graphical representations.
- **Solution:**
 - Create a `MovieClip` object and create different key frames showing the different graphical representations of status information. If the information is not to be shown sometimes, one key frame may remain empty.
 - Add a `stop()` ; action to the first key frame.
 - Optionally, assign labels to the key frames.
 - Place the `MovieClip` object on the stage
 - Show various status information by "`gotoAndStop()`" to the `MovieClip` object.
- **Examples:**
 - Realisation of the generic pre-defined `Button` class
 - Quiz example from `ActionScript 2.0 Dictionary`, pp. 8 ff.

A More Object-Oriented Solution

- Problems with the “traditional” solution:
 - Four different regions (with different highlighting colours) require four symbols
 - Event handling code has to be attached to *instance* of MovieClip symbol
 - Event handling code is duplicated
- The Macromedia partial solution:
 - Introduction of the special “Button” class
- A Programmer’s solution (next few slides):
 - Create a reusable class for a highlightable region
 - Make the color into a parameter settable from outside

Reusable Highlighting Color Block

```
class ColorBlock extends MovieClip {  
  
    private var myColor:Color;  
    public var myOnRgb:Number;  
  
    public function onLoad() {  
        myColor = new Color(this);  
    }  
  
    public function onRollOver() {  
        myColor.setRGB(myOnRgb);  
    }  
  
    public function onRollOut() {  
        myColor.setRGB(0xffffffff);  
    }  
}
```

Used built-in technology:

Color object controls the color of the movie clip.

Constructor assigns the new object to the given movie clip.

setRGB function actually changes the color.

Creating Instances of the Reusable Symbol

- There is *one* symbol with several instances (example: lo_mc, ro_mc, lu_mc, ru_mc)
- The symbol defines the graphical shape with irrelevant color.

- Initialisation code:

```
lo_mc.myOnRgb = 0xff0000; //red
ro_mc.myOnRgb = 0x0000ff; //blue
lu_mc.myOnRgb = 0x00ff00; //green
ru_mc.myOnRgb = 0xffff00; //yellow
```

1 Example Technology: Macromedia Flash & ActionScript

1.1 Multimedia authoring tools - Example Macromedia Flash

1.2 Elementary concepts of ActionScript

Scripting in General + „History“ of ActionScript

Objects and Types in ActionScript

Animation with ActionScript

1.3 Interaction in ActionScript

Handling of Mouse Events

Classical Model-View-Controller Programming

1.4 Media classes in ActionScript

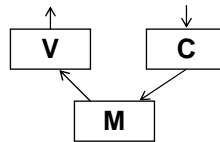
Creating a “Graphically Enhanced” User Interface

- Traditional programming
 - Example: Account with credit and debit function
- Additional “multimedia” features:
 - Auto-highlighting buttons
 - Visualization of money transfer direction
 - Visualization of “low” warning

The Account Class

```
class Account {  
    var saldo:Number = 0;  
    var num:Number;  
  
    function Account(accnum:Number) {  
        num = accnum;  
    }  
  
    function debit(n:Number) {  
        saldo -=n;  
    }  
  
    function credit(n:Number) {  
        saldo +=n;  
    }  
  
    function getNumber():Number {  
        return (num);  
    }  
  
    function getSaldo():Number {  
        return (saldo);  
    }  
}
```

Model-View-Controller (MVC) Paradigm



- Model:
 - Business model, mostly independent of user interface
 - Observable by arbitrary objects (application of *Observer* pattern)
- View:
 - Representation on user interface
 - Observes the model
 - Asks required data from the model
- Controller:
 - Modifies values in the model
 - Is driven by user interactions, therefore bound to elements of interface
 - Handles events mainly by calling methods of the model

Predefined Event Dispatcher

- Code base for library of predefined ActionScript classes:
 - In “Configuration/Classes” subdirectory
 - Contains readable ActionScript code (often undocumented)
- “mx” subdirectory:
 - Library functions for advanced use of ActionScript
 - E.g. “mx.events. ...”
 - Example class: **EventDispatcher**
- Usage by “import” statement as in Java
 - E.g. `import mx.events.EventDispatcher;`

Model: Account Class with Event Dispatching

```
import mx.events.EventDispatcher;

class Account extends EventDispatcher {

    var saldo:Number = 0;
    var accNum:Number;

    function Account(an:Number) {
        accNum = an;
    }

    function debit(n:Number) {
        if (n < 0) return;
        saldo -=n;
        if (n <> 0)
            dispatchEvent({type:"saldoLower"});
    }

    function credit(n:Number) {
        if (n < 0) return;
        saldo +=n;
        if (n <> 0)
            dispatchEvent({type:"saldoHigher"});
    } ...
}
```

View: User Interface Design

- Main output form is a (dynamic) text field
- However:
 - Text fields cannot carry ActionScript code
 - Text field cannot be easily associated with AS class
- How can we stay object-oriented?
- Idea: Add a new function to the text field object...

SuperBank

Your current account balance is:

€

Your action:

Amount:

€

Extending a TextField Object

- `saldo_txt` is a TextField object generated in the authoring tool
- Extension code (in main timeline):

```
saldo_txt.update = function() {
    var saldo: Number = myAccount.getSaldo();
    saldo_txt.text = saldo;
    if (saldo < 0)
        lowWarning_mc.gotoAndPlay("startAnim");
    else
        lowWarning_mc.gotoAndStop("stopAnim");
}
```

Connecting View to Model

- Using EventDispatcher
- Event handling code for updating view

```
var myAccount:Account = new Account(1234);
myAccount.addEventListener
    ("saldoLower", saldoLowerHandler);
myAccount.addEventListener
    ("saldoHigher", saldoHigherHandler);

function saldoLowerHandler(eventObj) {
    debit_mc.gotoAndPlay("startAnim");
    saldo_txt.update();
}

function saldoHigherHandler(eventObj) {
    credit_mc.gotoAndPlay("startAnim");
    saldo_txt.update();
}
```

Controller: User Event Handling

- Using Flash's built-in `Button` class makes highlighting easy.
- Event handling code (example "credit", "debit" is similar):

```
on (release) {  
    var amount:Number = Number(amount_txt.text);  
    if (isNaN(amount) or (amount < 0)) {  
        amount_txt.text += "?";  
    }  
    else {  
        myAccount.credit(amount);  
    }  
}
```