Psychological Task Design & Development

A Programming Workshop
Part II*c* – Advanced Programming

Wouter Boendermaker, M.Sc.

Johanna Quist, M.Sc.
Soraya Sanchez Maceiras, B.Sc.

University of Amsterdam

EPP Programming Workshop – February 12-13, 2015

4-Step Programme to Programming

Programming: Manipulating stuff through code

I. **Variables** (store values & complex Objects)

II. **Operations** (manipulate variables)

III. **Decisions** (make your program dynamic)

IV. **Repetitions** (i.e., how to avoid them!)

IV. Repetitions - Functions

Functions (or **methods**) are a combined set of operations.

First we **describe** a **function**:

```
function <functionName>(
    <param1>:<dataType>,
    <param2>:<dataType> = defaultValue ) :returnType {
       // code block
}
```

<u>Note</u>:

- Only the <u>order</u> of the parameters is important
- Next we can **call** the **function** from another function to dynamically execute a piece of code:

```
functionName( <argumentA>, <argumentB> );
```

IV. Repetitions - Example

```
function Test() :void {
    var myText:String = "Hello world!";
    saySomething();            // "nothing"
    saySomething( myText );  // "Hello world!“
    trace( double( 2 ) );    // 4
}

function saySomething( sayMe:String = "nothing" ) :void {
    trace( sayMe );
}

function double( doubleMe:Number ) :Number {
    return 2 * doubleMe;
}
```

Note:

- The return type `void` means nothing is returned
- The order of the **function descriptions** doesn’t matter; the **function calls** do!

Exercise 5 - Functions

1. Download **epw_ex5.zip** from www.wouboe.nl
2. Open **Exercise5.as** and **Exercise5.fla** in Flash

3. Create a global variable called myName (`String`) and with value *your name*. Make a `function` called testMe that accepts two `String`s as **arguments**: one called testName, and one called letter with default value `"a"`. The `function` returns a value of type `Boolean`.
4. Make this `function` return `true` if the letter is in the provided testName, and `false` if it's not. Call the `function` from your **constructor** `function`, once without the second argument and once with, and use `trace( )` to view the results.

IV. Repetitions – Classes (1)

Classes are even more elaborate **descriptions** (**blueprints**) for (sometimes very complex) **Objects**.

Objects can be **variables**, **functions**, data structures, etc., as well as **instances** of a **Class**.

Flash has many **Classes** available, but also allows us to write our own.

Using **Classes** and **Objects**, we can apply **Object-Oriented Programming techniques** to our designs.

e.g., one Car **Class** using 4 separate instances of the Wheel **Class**.

IV. Repetitions – Classes (2)

A **Class description** can hold any number of

- Properties (variables)
- Methods (functions)

To make a new **instance** of a **Class**, the `new` operator is used:

```
var myArray:Array             = new Array();
var myTextField:TextField     = new TextField();
var myCustomClass:IAT         = new IAT();
```

Note:

- There is an important difference between the **data type** and the **instantiation** of a **Class**.

IV. Repetitions - Example

```
Class Car extends MovieClip {
    function Car() :void {
        var wheel1:Wheel = new Wheel();
        var wheel2:Wheel = new Wheel();
        wheel1.radius = 12; // sets only wheel1 to 12
        trace( wheel2.whatIsMySpeed() ); // 10
    }
}

Class Wheel extends MovieClip {
    var radius:uint;    // global vars
    var speed:int = 10;
    function Wheel() :void {
        // ...
    }
    function whatIsMySpeed() :int {
        return speed;
    }
}
```

Note:
- The Classes should be in separate files, each bearing their respective names.
- Each instance of the Wheel class behaves completely separately.

Exercise 6 - Objects

1. Download **epw_ex6.zip** from www.wouboe.nl
2. Open **Exercise6.as** and **Exercise6.fla** in Flash

3. Make a new global variable called myArray with type `Array` (mind the capital A!). In your **constructor** function, **instantiate** the `Array` with

   ```
   myArray = new Array();
   ```
4. Use the `push()` function on your new `myArray` Object to add some elements to it (see online help file).
5. Make a new `function` called `sum()` that accepts an `Array` as input and `return`s a value of type `Number`, which is the sum of the elements in the received `Array`. Hint: use a loop.
6. Challenge: make two `function`s that (similarly) return the mean and the SD. Have each of these functions use the previous ones if applicable.

V. Visuals – Sprite & MovieClip

So far we've only used the **output panel**. Let's see how we can make some visuals inside the actual **.swf** file!

Flash has several types of displayable Object types, the most common of which are called the `Sprite` and the `MovieClip` Classes.

To make a new **instance** of the `MovieClip` Class, use

```
var myVisual:MovieClip = new MovieClip();
```

Now we can use all kinds of built-in functions to draw on the canvas:

```
myVisual.graphics.beginFill( 0x0099FF );
myVisual.graphics.drawRect( 0, 0, 50, 50 );
```

V. Visuals - stage

Flash uses an infinite number of **canvases** to show its visual assets. The **main canvas** is called the **stage**. To make things available on this **stage**, we use the `addChild( )` function:

```
stage.addChild( myVisual );
```

Similarly, we could add a second visual to the first:

```
myVisual.addChild( mySecondVisual );
```

VI. Interaction - Events

Next, to interact with the user, we can make use of user **Events**.

NB: **Flash** is **event-based** and uses many different types of **Events** (also e.g., **time** based).

To make our myVisual clickable, we use **EventListeners**.
EventListeners use three properties:

1. Which Object listens?
2. What type of Event do we listen to?
3. What do we do when we hear it?

```
myVisual.addEventListener( MouseEvent.CLICK, onClick );

function onClick( evt:MouseEvent ) :void {
   trace( evt );
}
```

VI. Interaction - Events

Next, to interact with the user, we can make use of user **Events**.

NB: **Flash** is **event-based** and uses many different types of **Events** (also e.g., **time** based).

To make our myVisual clickable, we use **EventListeners**.
EventListeners use three properties:

1. Which Object listens?
2. What type of Event do we listen to?
3. What do we do when we hear it?

```
myVisual.addEventListener( MouseEvent.CLICK, onClick );

function onClick( evt:MouseEvent ) :void {
    trace( evt );
}
```

Exercise 7 – Interactive Visuals

1. Download **epw_ex7.zip** from www.wouboe.nl
2. Open **Chessboard.as** and **Chessboard.fla** in Flash

3. Finish the code to make an interactive chess board.

Hints:

- Use a **nested** loop (a loop within a loop, like `if` within an `if`) to make a two dimensional board.
- Use the **modulo operator** to get the chessboard pattern
- Every space on the board should be an individual **MovieClip**, with its own **EventListener**.
- From the listener function, **trace** the coordinates of the space.
- Challenge: Make a **TextField** to show the coordinates on the stage instead. When a white square is clicked, make it change color. Reset the text and color after 1000 ms.

NB: An example of the finished task can be found online.

Additional Exercises

Explore the following:

- Showing assets:
  - Embed a picture and a sound
  - When the task starts, present a button
  - When the button is clicked, show the picture (and remove the button)
  - Present the sound when the picture is clicked
  - When the sound is done playing, remove the picture

- Dynamically loading in pictures (at runtime)

- Randomisation

- Making a trial structure

- Make a little memory game ☺