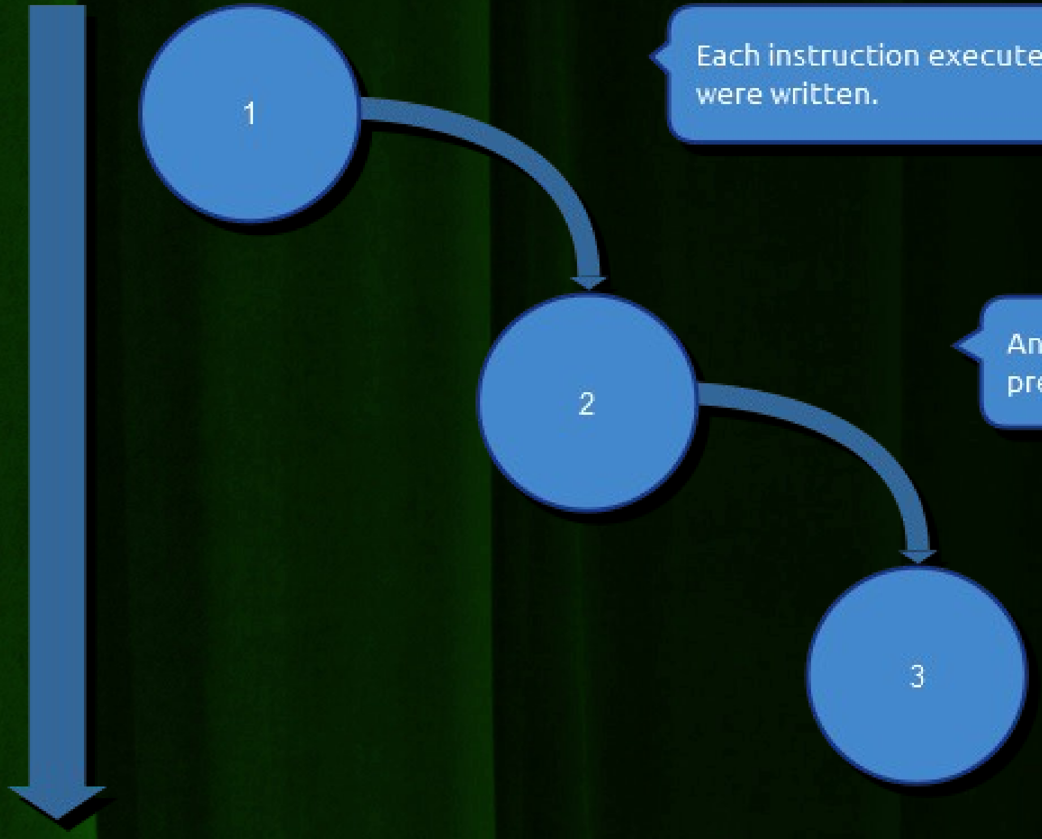
Actionscript 3.0 Basics

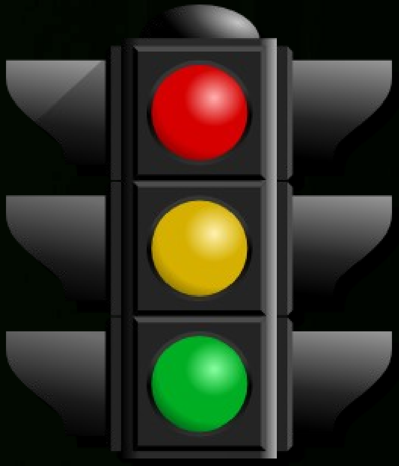
2 – Event-driven Programming

Synchronous Programming

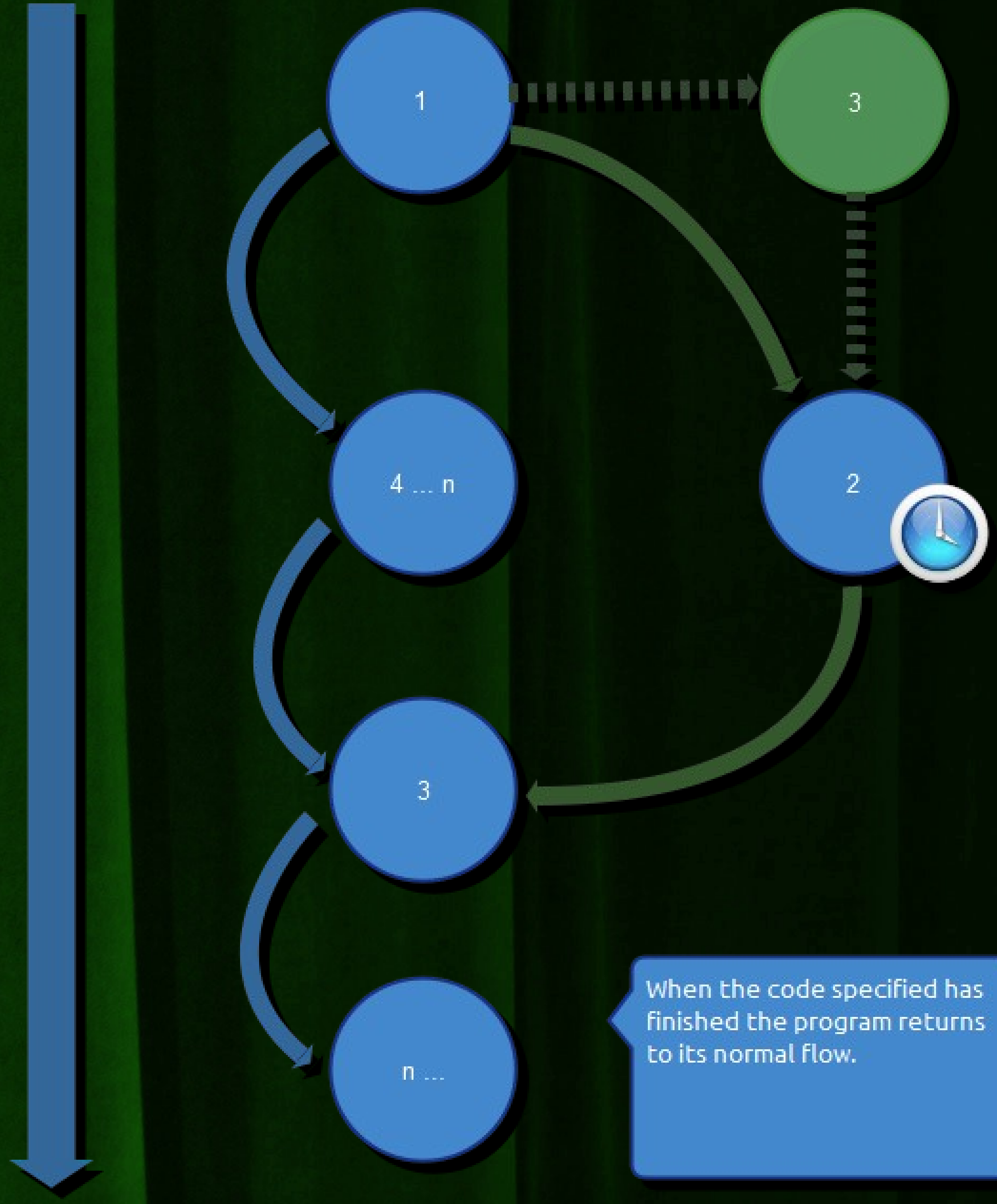


Each instruction executes in the order they were written.

An instruction is executed only if the previous instruction has finished.



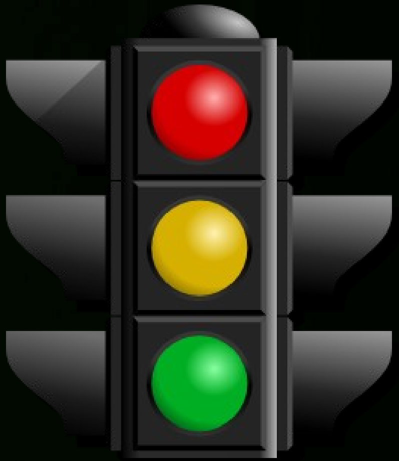
Asynchronous Programming



An instruction initiates a time-consuming process and indicates witch instruccions should be executed when the process finishes

The process runs in paralel with the rest of the program and when it finishes, it interrupts the normal flow of the program to execute the code specified.

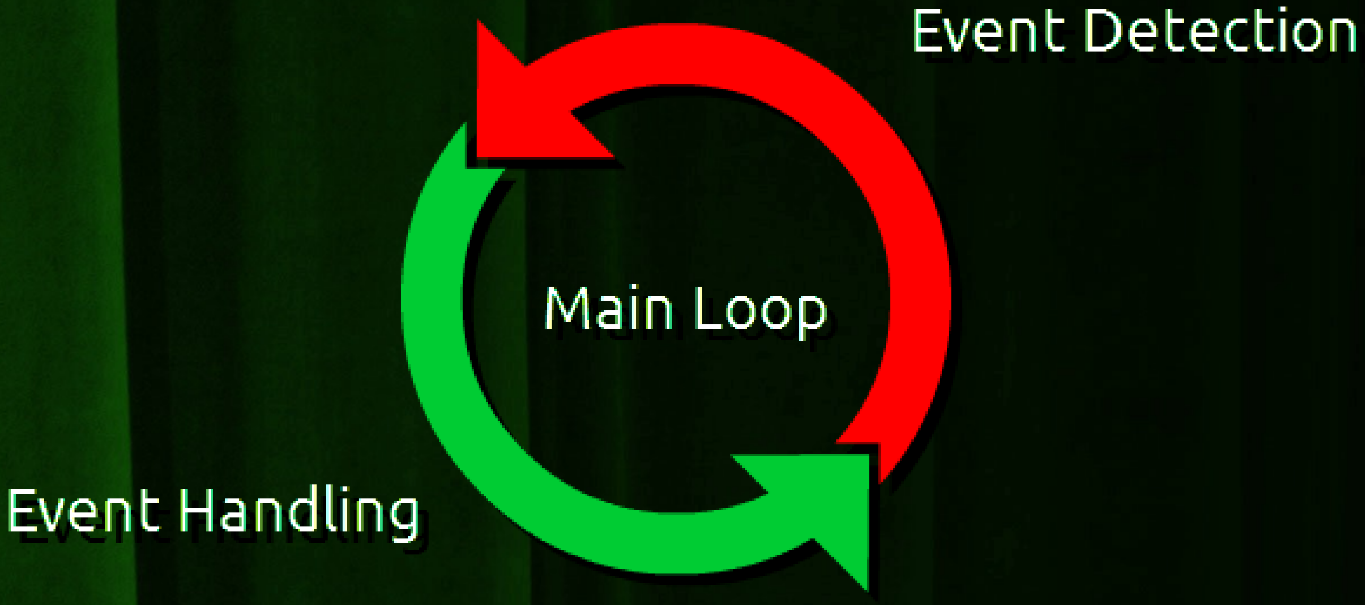
When the code specified has finished the program returns to its normal flow.



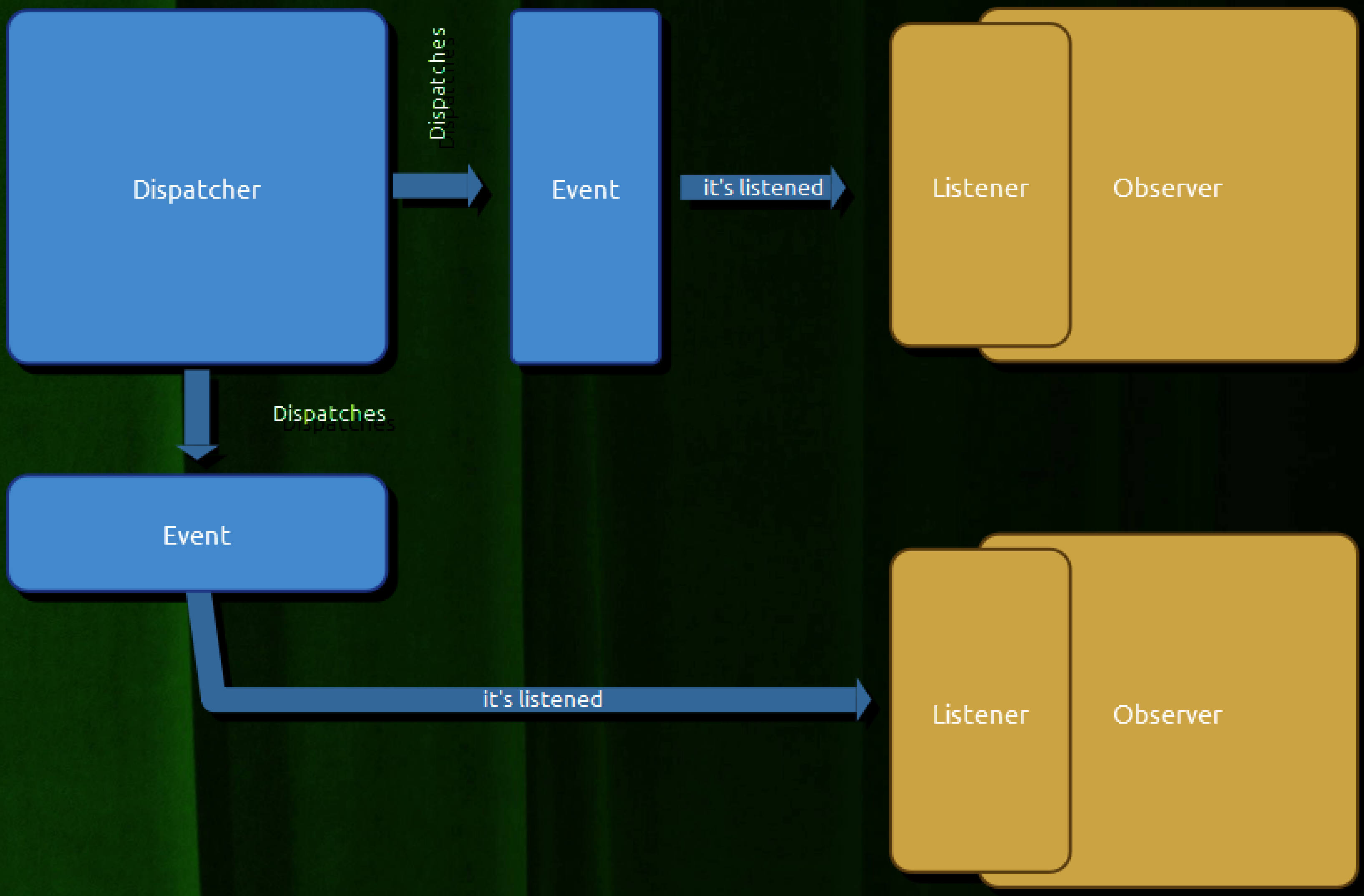
Event-Driven Programming

Is a programming paradigm in which the flow of the program is determined by events or messages from other programs or threads.
Event-driven programming can also be defined as an application architecture technique in which the application has a main loop which is clearly divided down to two sections: the first is event selection (or event detection), and the second is event handling.

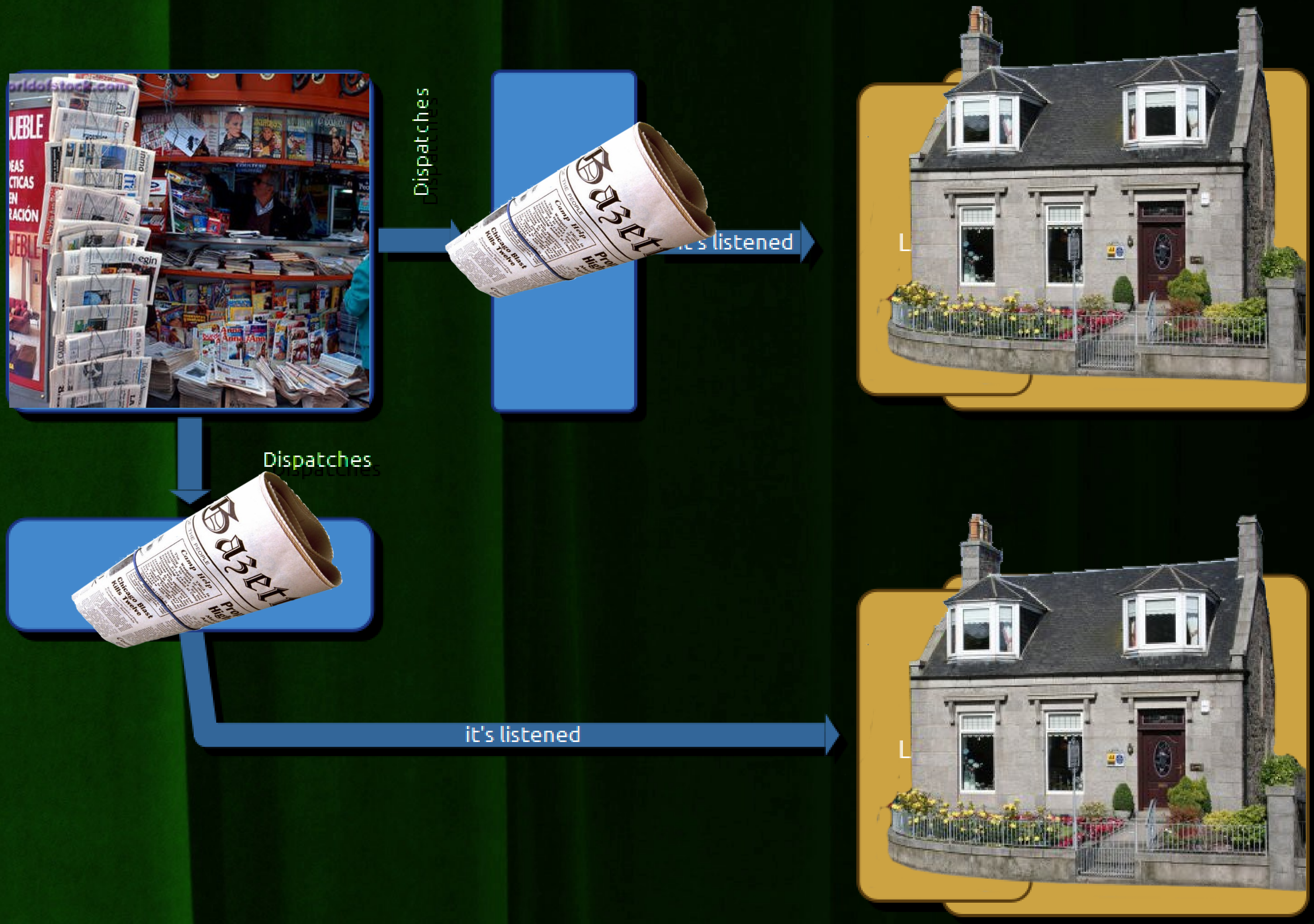
[http://en.wikipedia.org/wiki/Event-driven_programming]



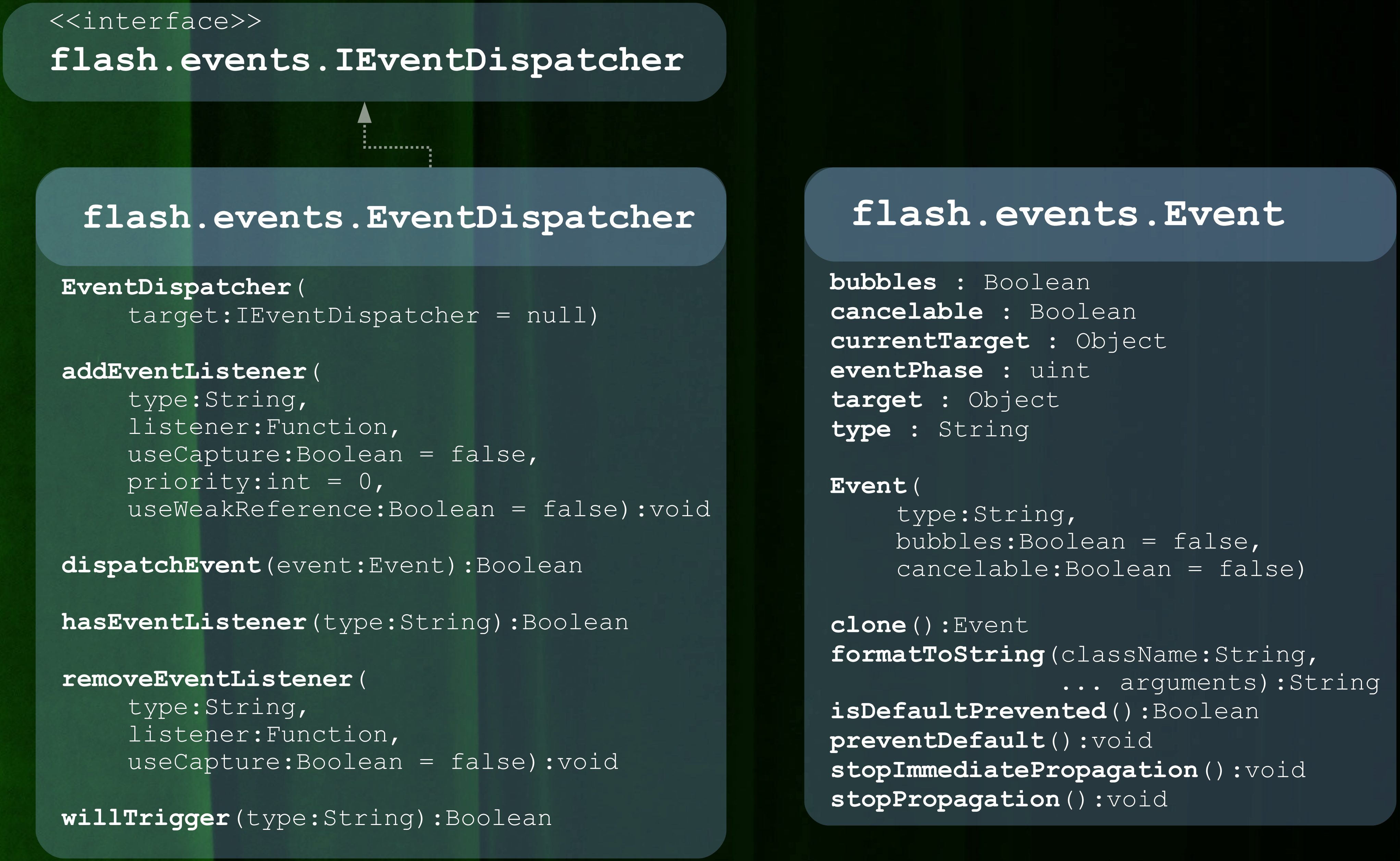
Observer-Dispatcher Pattern



Observer-Dispatcher Pattern



Flash Event System

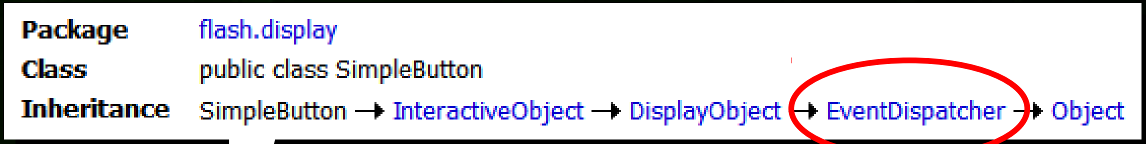


```
<<interface>>
flash.events.IEventDispatcher
```

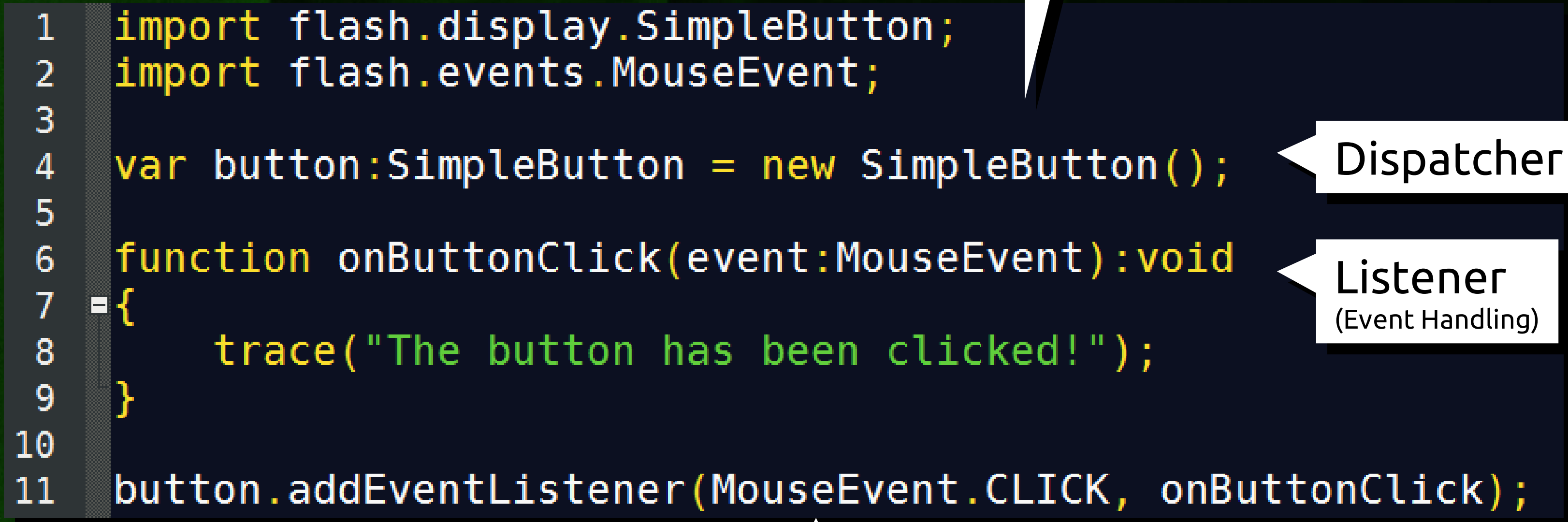
```
flash.events.EventDispatcher

EventDispatcher(
    target:IEventDispatcher = null)

addEventListener(
    type:String,
    listener:Function,
    useCapture:Boolean = false,
    priority:int = 0,
    useWeakReference:Boolean = false):void

dispatchEvent(event:Event):Boolean

hasEventListener(type:String):Boolean

removeEventListener(
    type:String,
    listener:Function,
    useCapture:Boolean = false):void

willTrigger(type:String):Boolean
```

```
flash.events.Event

bubbles : Boolean
cancelable : Boolean
currentTarget : Object
eventPhase : uint
target : Object
type : String

Event(
    type:String,
    bubbles:Boolean = false,
    cancelable:Boolean = false)

clone():Event
formatToString(className:String,
               ... arguments):String
isDefaultPrevented():Boolean
preventDefault():void
stopImmediatePropagation():void
stopPropagation():void
```

Flash Event System



Package	flash.display
Class	public class SimpleButton
Inheritance	SimpleButton → InteractiveObject → DisplayObject → EventDispatcher → Object



```
import flash.display.SimpleButton;
import flash.events.MouseEvent;

var button:SimpleButton = new SimpleButton();

function onButtonClick(event:MouseEvent):void
{
    trace("The button has been clicked!");
}

button.addEventListener(MouseEvent.CLICK, onButtonClick);
```

Dispatcher

Listener
(Event Handling)

Event Subscription

Flash Event System

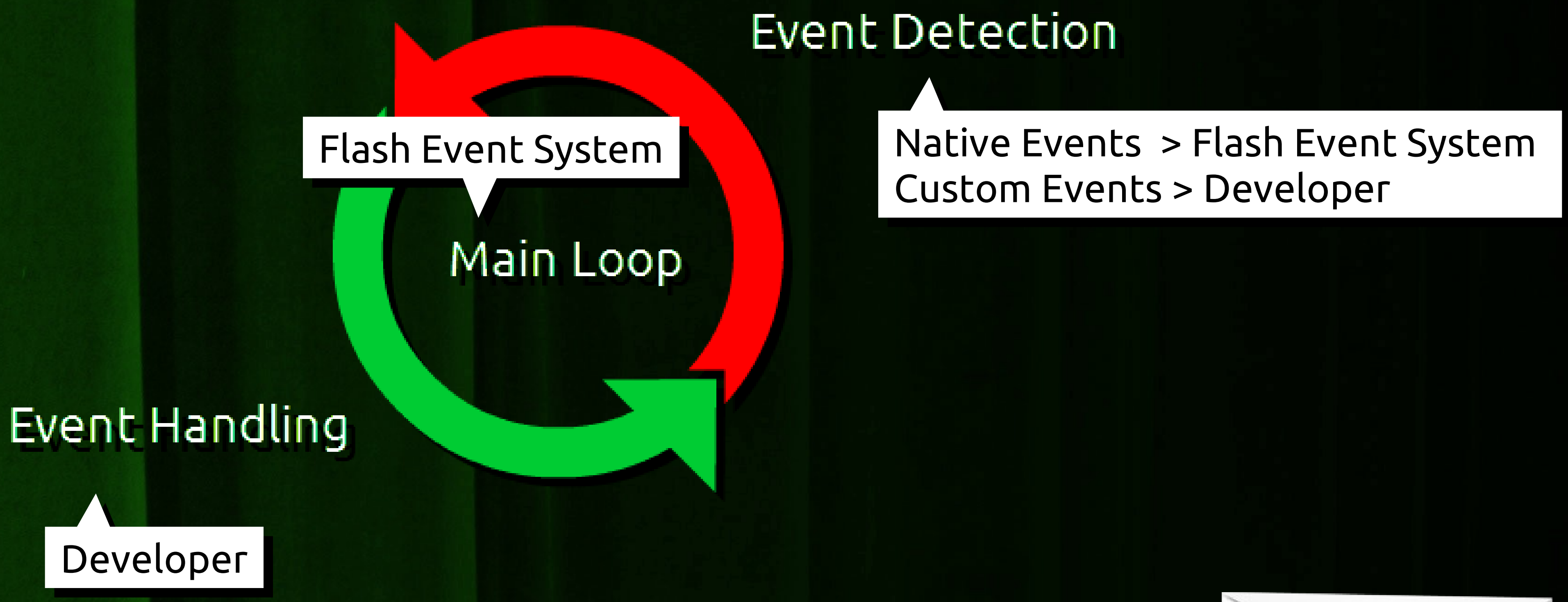
```
import flash.display.SimpleButton;
import flash.events.MouseEvent;

var button:SimpleButton = new SimpleButton();

function onButtonClick(event:MouseEvent):void
{
    var x:Number = event.mouseX;
    var y:Number = event.mouseY;

    trace("The button was clicked at x:",x,"y:",y);
}

button.addEventListener(MouseEvent.CLICK, onButtonClick);

```

Event Data

Flash Event System

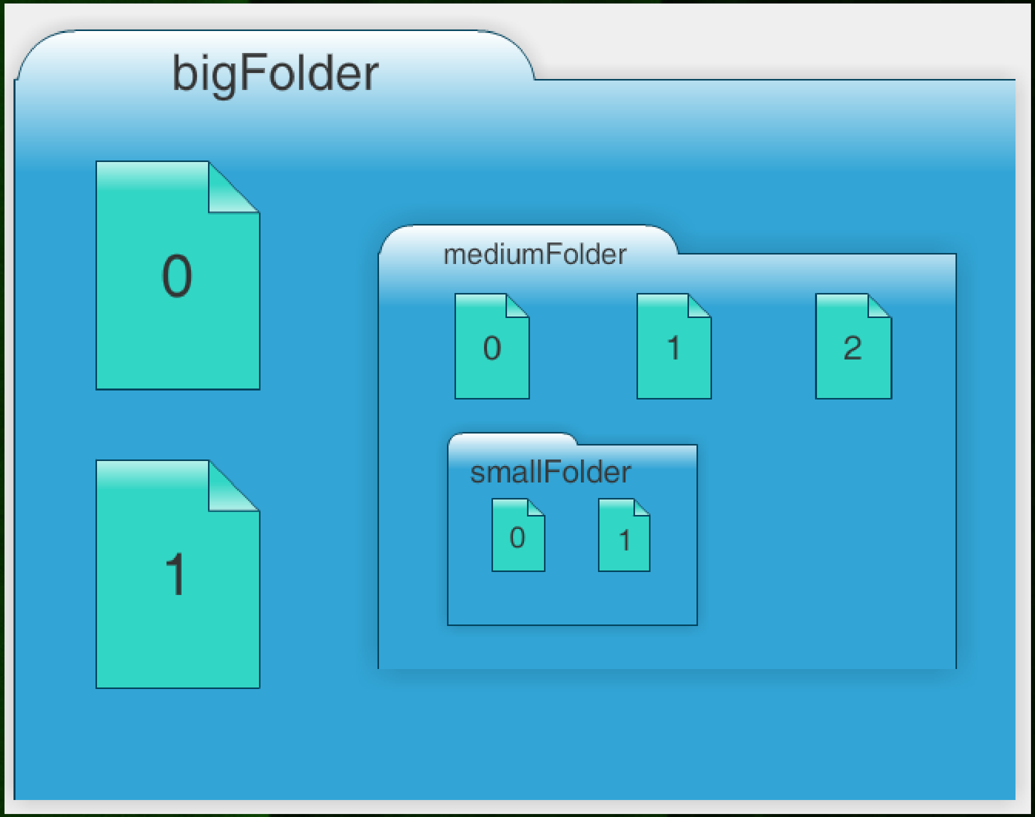
```
//Listen when a mouse click is performed and execute onButtonClick()
button.addEventListener("click", onButtonClick);
button.addEventListener(MouseEvent.CLICK, onButtonClick);

//Is any suscriber listening this event?
button.hasEventListener(MouseEvent.CLICK); // true

//Stop listening to the event
button.removeEventListener(MouseEvent.CLICK,onButtonClick);

button.hasListener(MouseEvent.CLICK); // false
```

Flash Event System


Responsibility Distribution
Event Detection
Native Events > Flash Event System
Custom Events > Developer
Flash Event System
Main Loop
Event Handling
Developer

Event Propagation

Document Object Model

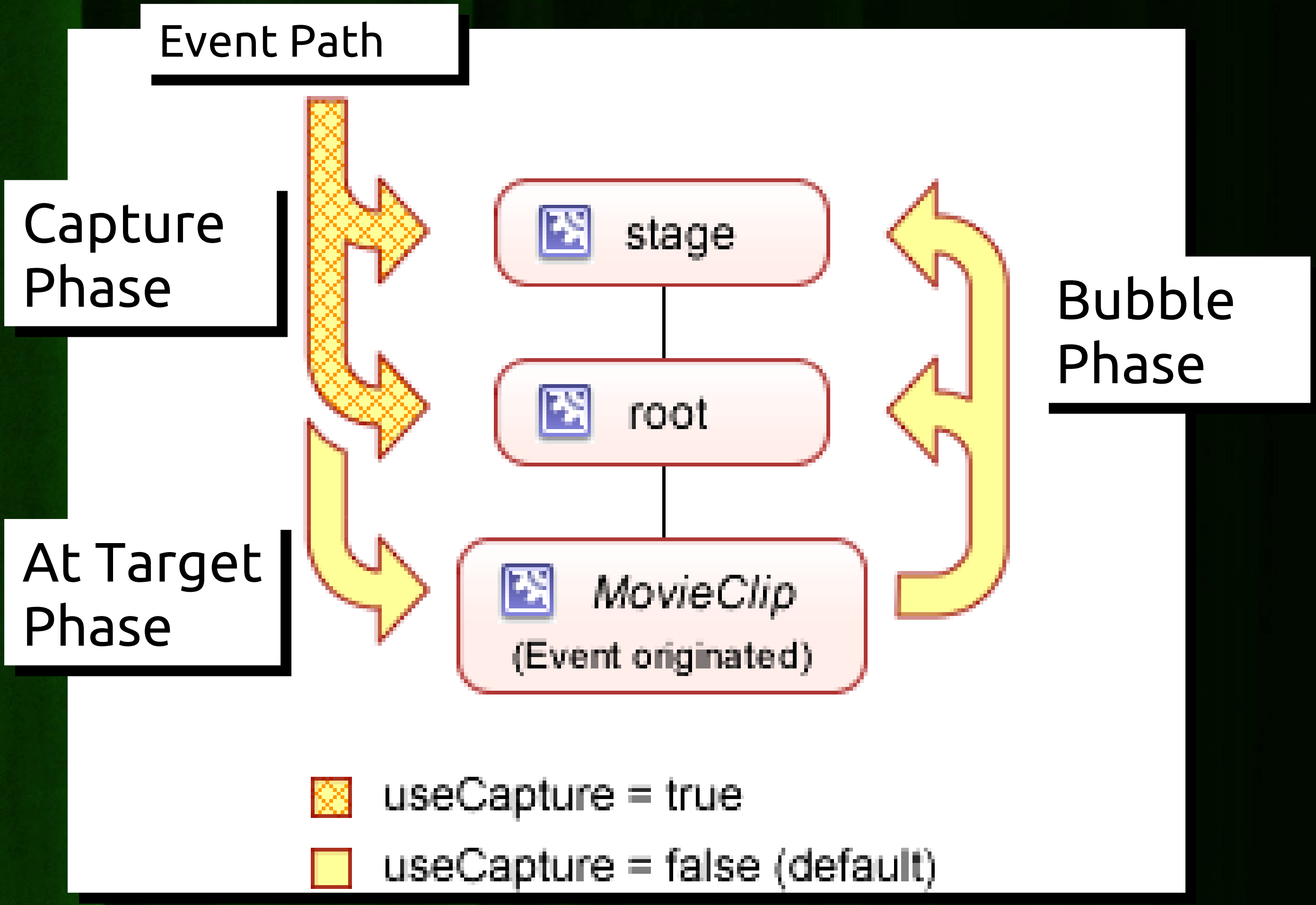


```
<root>
 ↳ BigFolder
     ↳ 0
     ↳ 1
     ↳ MediumFolder
         ↳ 0
         ↳ 1
         ↳ 2
         ↳ SmallFolder
             ↳ 0
             ↳ 1
```

↳ Contains

Event Propagation

Event Phases



Event Propagation

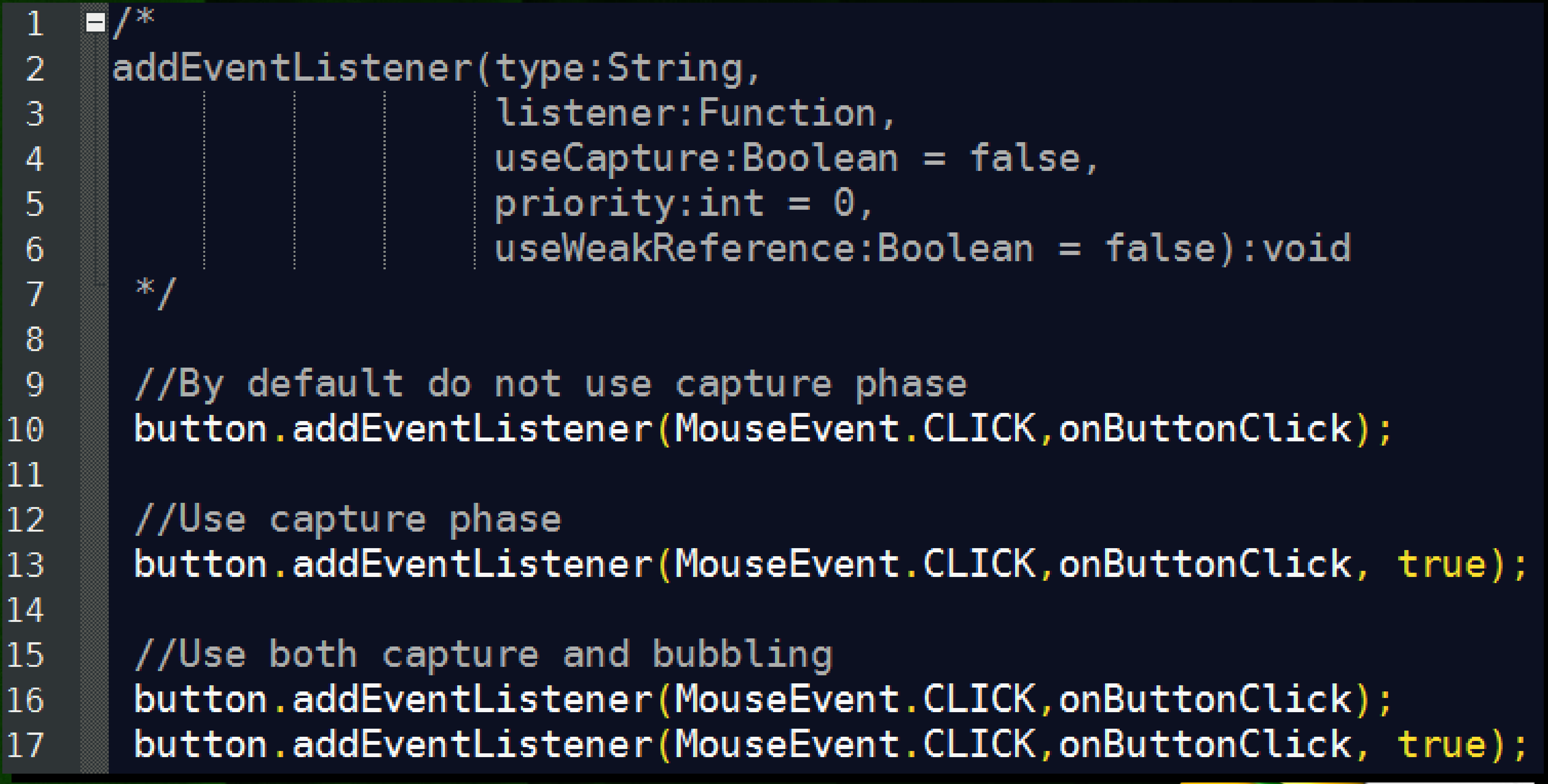
Event Phases

Event Propagation

Event Phases

Event Propagation

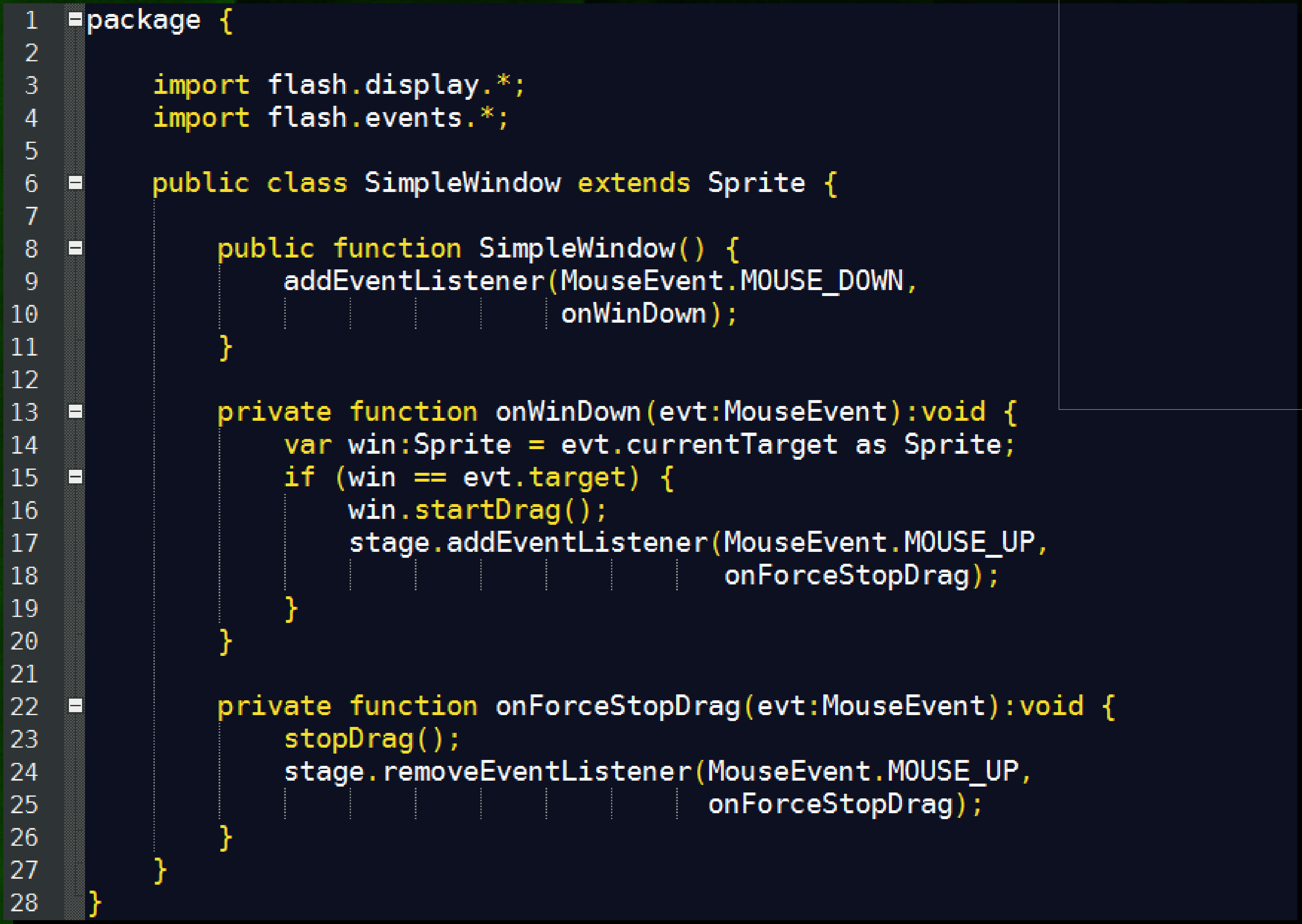
Event Phases



```
/*
addEventListener(type:String,
                 listener:Function,
                 useCapture:Boolean = false,
                 priority:int = 0,
                 useWeakReference:Boolean = false):void
 */

 //By default do not use capture phase
 button.addEventListener(MouseEvent.CLICK,onButtonClick);

 //Use capture phase
 button.addEventListener(MouseEvent.CLICK,onButtonClick, true);

 //Use both capture and bubbling
 button.addEventListener(MouseEvent.CLICK,onButtonClick);
 button.addEventListener(MouseEvent.CLICK,onButtonClick, true);
```

Event Propagation

Event Targets



```
package {

    import flash.display.*;
    import flash.events.*;

    public class SimpleWindow extends Sprite {

        public function SimpleWindow() {
            addEventListener(MouseEvent.MOUSE_DOWN,
                                    onWinDown);
        }

        private function onWinDown(evt:MouseEvent):void {
            var win:Sprite = evt.currentTarget as Sprite;
            if (win == evt.target) {
                win.startDrag();
                stage.addEventListener(MouseEvent.MOUSE_UP,
                                            onForceStopDrag);
            }
        }

        private function onForceStopDrag(evt:MouseEvent):void {
            stopDrag();
            stage.removeEventListener(MouseEvent.MOUSE_UP,
                                            onForceStopDrag);
        }
    }
}
```

Custom Events

```
public class MarioEvent extends Event {
    public static const HURT:String ="hurt";
    public static const DIED:String ="died";

    public var lifesLeft:int = 0;

    public function MarioEvent(type:String, lifesLeft:int){
        super(type);
        this.lifesLeft = lifesLeft;
    }

    override public function clone():Event
    {
        return new MyEvent(this.type, this.lifesLeft);
    }
}

public class Mario extends EventDispatcher {
    public var isSmall:Boolean = true;
    public var lifesLeft:int = 3;

    public function Mario()
    {
        super();
    }

    public function hurt():void {
        if(isSmall){
            this.lifesLeft--;
            this.dispatchEvent(new MarioEvent(MarioEvent.DIE, this.lifesLeft));
        }
        else{
            this.isSmall = true;
            this.dispatchEvent(new MarioEvent(MarioEvent.HURT, this.lifesLeft));
        }
    }
}
```

Custom Event

Event Name Constants

Custom Dispatcher

Event Detection

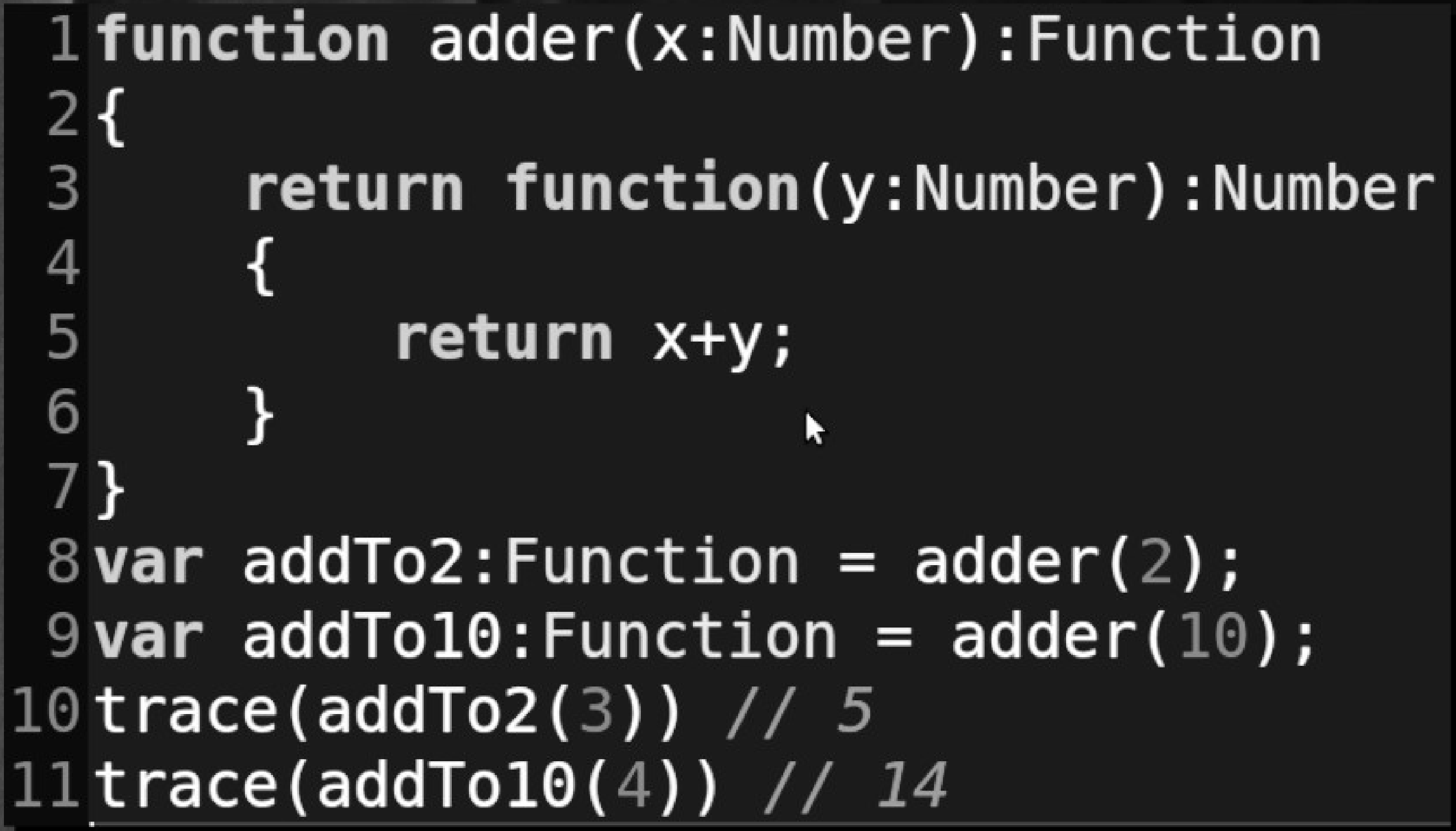
Advantages

- Promotes decoupling and encapsulation of objects.
- Cleaner code, consistent with flash apis.
- Faster since it's implemented as a native language feature.
- Standard, consistent with many language implementations.

Functions

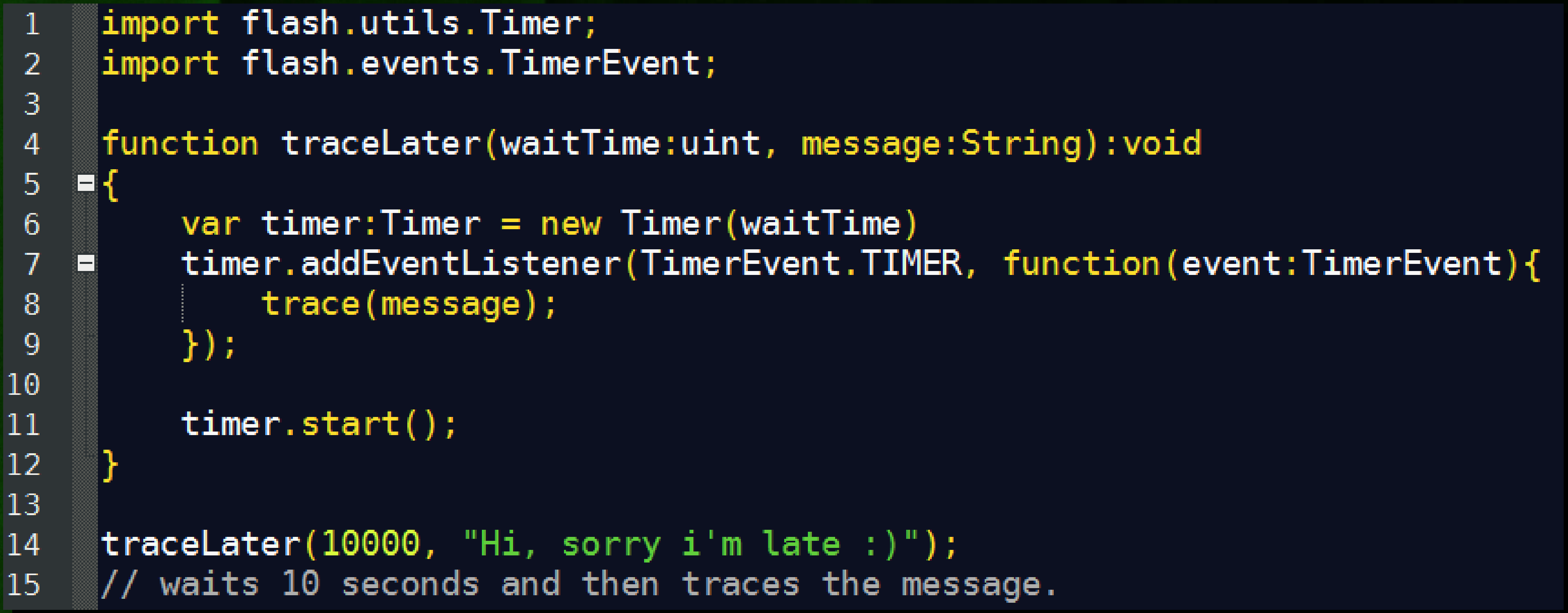
Closures:

A closure is a functional language feature that allows functions to retain the references from its lexical environment even after that environment is no longer in execution.



```
function adder(x:Number):Function
{
    return function(y:Number):Number
    {
        return x+y;
    }
}
var addTo2:Function = adder(2);
var addTo10:Function = adder(10);
trace(addTo2(3)) // 5
trace(addTo10(4)) // 14
```

Events + Closures



```
import flash.utils.Timer;
import flash.events.TimerEvent;

function traceLater(waitTime:uint, message:String):void
{
    var timer:Timer = new Timer(waitTime)
    timer.addEventListener(TimerEvent.TIMER, function(event:TimerEvent){
        trace(message);
    });

    timer.start();
}

traceLater(10000, "Hi, sorry i'm late :)");
// waits 10 seconds and then traces the message.
```

Thanks!

github.com/matix/as3basics

matias.figueroa@globant.com
@matixfigueroa