OOP, felhantering, events, och andra trevliga saker

Design av interaktiv multimedia

OOP

vs

procedural programming

OOP

- Vokabulär:
  - ~~Class~~
  - ~~Properties~~
  - ~~Methods~~
  - Encapsulation
  - Getters/setters
  - Inheritance och composition
  - Interface

Encapsulation

- Anger vem/vad som har tillgång till properties och metoder.
  - Public (överallt)
  - Private (inom klassen)
  - Protected (inom samma klass och ärvande klasser)
  - Internal (inom samma package)
  - Static (hör till klassen istället för instanser)
- **Ska** anges vid variabeldeklaration och när du skriver egna metoder!

Encapsulation i praktiken

```
public var myInt:int;
protected var myInt:int;
internal var myInt:int;
static var myInt:int;
private var _myOtherInt:int;
```

Notera _

```
private function myFunction():void {

}
```

Getters & setters

- Getter lämnar ut värdet på en privat variabel.

```
public function get myInt():int {
        return _myInt;
}
```

- Setter sätter ett nytt värde på en privat variabel.

```
public function set myInt(newValue:int):void {
        _myInt = newValue;
}
```

Använda getters och setters

```
public function get score():int {
        return _score;
}

public function set score(n:int):void {
        if(n < 10) {
                _score = n;
        }
}
```

```
package {
        import flash.display.MovieClip;
        public class Game extends MovieClip {
                public var player:Player;
                …
                player.score = 5;
                trace(player.score);
                …
        }
}
```

setter

getter

Constants

- Constants har ett fast värde som inte kan ändras.
  - Ge värdet vid deklarationen (deklareras globalt).
  - Använd inte `var`.
- Namnges **alltid** med VERSALER.

inget var!

```
const MY_VALUE:int = 5;
```

- Nås med dot-notation:

```
myClass.MY_VALUE;
```

Inheritance

- En klass ärver egenskaper från en annan klass.

```
public class MyClass extends MovieClip {
        public function MyClass() {
                …
        }
}
```

- MovieClip i sin tur ärver också:

MovieClip → Sprite → DisplayObjectContainer → InteractiveObject → DisplayObject → EventDispatcher → Object

Composition

- Composition säger att ett objekt är **byggt av** vissa objekt, snarare än att det **är** ett objekt.

```
public class Seagull {
        private var _wing:Wing;
        public function Seagull() {
                _wing = new Wing(25);
        }
}
```

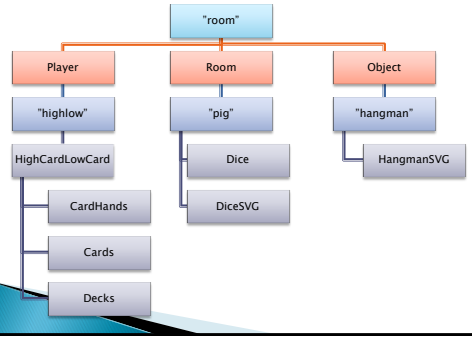
```
public class Wing {
        private var _size:int;
        public function Wing(size:int) {
                _width = _size;
        }
        public function get width():int {
                return _width;
        }
}
```

Interface

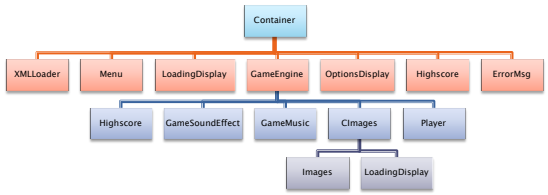
- Tillåter gemensamma metoder utan att använda inheritance.
- T.ex. IEventDispatcher som har addEventListener och removeEventListener.
- En klass kan implementera flera interface (till skillnad från inheritance som bara går en gång).

```
public class Main implements IEventDispatcher
```

Exempel: DragonSlayer



Exempel: Spot the Differences



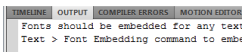
Felhantering

Felhantering

- **Compiler error:** Uppstår när din Flash-applikation publiceras.

- **Runtime error:** Uppstår efter att din fil publicerats, under tiden som den används.

- **Varningar:** Uppstår under publicering, men stoppar inte publikationen. Applikationen kan fortfarande fungera.



- Compiler errors:
  - `Error #1013: The private attribute may be used only on class property definitions.`
  - `Error #1021: Duplicate function definition.`
  - `Error #1051: Return value must be undefined.`
  - `Error #1059: Property is read-only.`
  - `Error #1120: Access of undefined property %s.`
- Run-time errors:
  - `Error #1009: Cannot access a property or method of a null object reference`
  - `Error #1065: Variable %1 is not defined`
  - `Error #2047: Security sandbox violation: %1: %2 cannot access %3.`
  - `Error #1071: The Stage class does not implement this property or method.`

Sandbox violation

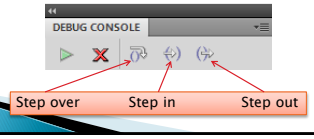
`Error #2047: Security sandbox violation`

- Flash har inbyggda säkerhetsgränser – en s.k. sandbox (sandlåda).
- Du får aldrig sandbox violation om du kör filerna lokalt på datorn.
- Löses med en crossdomain-fil (**crossdomain.xml**) i root-mappen:

```
<?xml version="1.0"?>
<cross-domain-policy>
        <allow-access-from domain="*.example.com" />
        <allow-access-from domain="www.friendOfExample.com" />
        <allow-access-from domain="192.0.34.166" />
</cross-domain-policy>
```

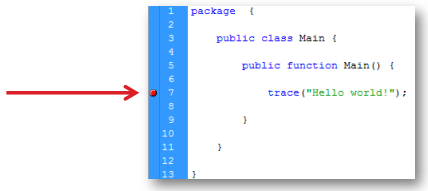
Debugging

- **Step over**: gå till nästa rad kod.
- **Step in**: gå till nästa rad kod – om en funktion används så hoppar debuggern in i funktionen och fortsätter där.
- **Step out**: hoppar till det ställe där funktionen returnerar ett värde.



Debugging

- Lägg in en brytpunkt (breakpoint) där du vill att koden ska stanna.



Try – catch

- Används för att testa och hantera eventuella fel som uppstår.

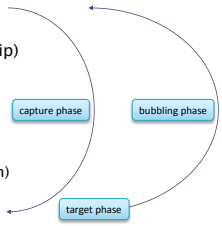
```
try {
        //Här testar vi något
} catch (error:Error) {
        //Här anger vi vad som händer om det blev fel
}
```

- Är en försäkring att applikationen inte lägger av vid ett visst runtime error.

Events

Eventflödet

- Display list:
  - Stage
    - Child1 (t.ex. MovieClip)
      - Child1:1
    - Child2
      - Child2:1 (t.ex. Button)
      - Child2:2



Eventflödet

```
addEventListener(type:String, listener:Function, useCapture:Boolean = false)
```

- **Capture phase** lyssnar på föräldern. T.ex.:
  - myParent.addChild(myChild);
  - myChild.addEventListener(…, …, true);
- **Bubbling phase** lyssnar på barnet. T.ex.:
  - myParent.addChild(myChild);
  - myParent.addEventListener(…, …, false);
- Bubbling phase är default.

Custom events

- Events implementeras egentligen med textsträngar (constants).
  - MouseEvent.CLICK = "click"
  - KeyboardEvent.KEY_DOWN = "keyDown"
- Det går att dispatcha (skicka iväg) och lyssna efter egna textsträngar:

```
dispatchEvent(new Event("myOwnEvent"));
```

```
addEventListener("myOwnEvent", myOwnEventHandler);
```

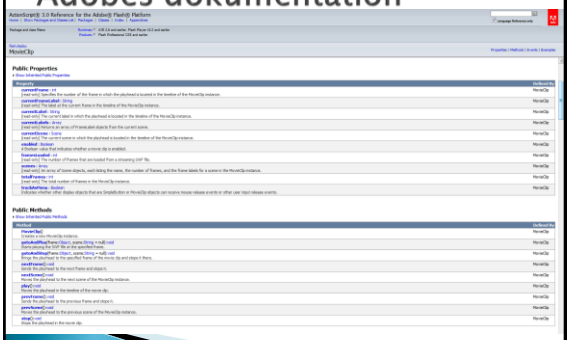
Custom event class

```
package {
   import flash.events.Event;

   public class CustomEvent extends Event {

      public static const CUSTOM:String = "custom";

      public var arg:*;

      public function CustomEvent(type:String, customArg:*=null,
                                  bubbles:Boolean=false,
                                  cancelable:Boolean=false) {

         super(type, bubbles, cancelable);

         this.arg = customArg;
      }

      public override function clone():Event {
         return new CustomEvent(type, arg, bubbles, cancelable);
      }

      public override function toString():String {
         return formatToString("CustomEvent", "type", "arg",
                               "bubbles", "cancelable", "eventPhase");
      }
   }
}
```

Här går att lägga in egna parametrar

Andra bra saker

Adobes dokumentation



Exempel hitTestObject()

Shared object

- Skapa ett nytt shared object:
  ```
  private var _mySO:SharedObject;
  ```
- Hämta ett existerande SO/döp det nya:
  ```
  _mySO = SharedObject.getLocal("TotalTimeAtThisPage");
  ```
- Läs data från SO:
  ```
  _myInt = _mySO.data.myNumber;
  ```
- Lägg till data på SO:
  ```
  _mySO.data.myNumber = 2;
  ```
- Spara data lokalt:
  ```
  _mySO.flush();
  ```
- Stäng ditt SO:
  ```
  _mySO.close();
  ```

Arrayer

- En array är en samling värden.
- Alla värden tilldelas indexnummer (från 0).
- Deklareras på flera sätt:

```
private var _arr:Array = new Array();
private var _arr:Array = [];
private var _arr:Array = ["Hello", "world", "again", "!"];

_arr[0] = "Hello";
_arr.push("world");
```

0	Hello
1	world
2	again
3	!

Arrayer ❤ For loops

```
for (var i:int = 0; i < _arr.length; i++) {
      trace(_arr[i]);
}
```

DisplayObjectContainer

- DisplayObject är basklassen för alla objekt som kan placeras i displaylistan (d.v.s. på scenen).
- DisplayObjectContainers är en typ av DisplayObject som kan ha ”barn”.
- Typer av DisplayObjectContainers:
  - Stage
  - Sprite (som ett MC men utan tidslinje)
  - Loader (laddar externa objekt till display list)
  - MovieClip

Garbage Collector

- GC städar bort objekt som inte längre används.
- Bara objekt som är tomma tas bort – töm därför dina objekt så fort de inte behövs!
  - delete(o:Object); tar bort objekt som inte används.
  - Sätt objektreferenser = *null*.
  - Ta bort event listeners (removeEventListener()).
  - Stoppa animeringar.
  - Ta bort timers.

Vad mer kan man göra?

- Flex
  - För utvecklare
  - Användargränssnitt
- Adobe AIR
  - Desktopapplikationer (mac, windows, linux)
  - Mobila enheter (ska läggas ner?)
- 3D
- Integrera webbkamera
- Streama live-video
- Animera filmer