

CS 175

Action Script

Classes 2

Linking Symbols to Class

Linking a symbol to a class

- In CS116, we used to create symbols, representing characters on the screen, linked to the MovieClip Class. But the problem is that the Movie Clip Class doesn't contain specific attributes to all kinds of symbols like health, how many lives it has, different weapon types, different AI functions ...
- To solve our problem we used to open the symbol in the Library and add the extra attributes there.
- Now, since we know how to create classes, we are going to link the symbol to the class we created.
- Our created class will have its own properties and at the same time have the attributes found in the Movie Clip Class (in order to have access to the position, animation timeline and so that our object is rendered).

Our Hero Class

The class in our .as file looks something like this:

```
package MyClasses
{
  public class Hero
  {
    public var nPosX:Number;
    public var nPosY:Number;
    public var iHealth:int;

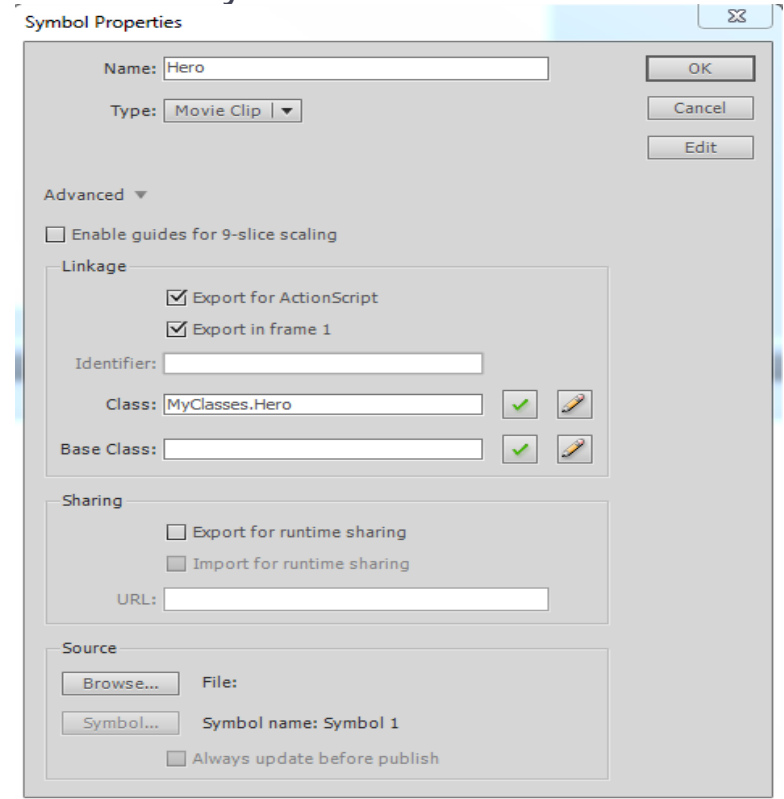
    public function Hero(nPosX_:Number , nPosY_:Number , iHealth_:int)
    {
      nPosX = nPosX_;
      nPosY = nPosY_;
      iHealth = iHealth_;
    }
  }
}
```

Linking The Symbol

- In your object's Library, you will find a list of all the objects that were created and converted to symbol.
- Right click on the name of the object you want to link to your class. Select Properties.
- Under Class, specify the class you want to link to : “ MyClasses.Hero”

NB: you need to specify the full path

- The base class used to be the MovieClip class but once we link it to our class it will become **empty**.



Creating An Instance

- In order to create a Hero instance and add it to the stage we do the following:

```
import MyClasses.Hero;
```

```
var heroCharacter:Hero = new Hero(200 , 250 , 100);  
stage.addChild(heroCharacter);
```

- Run the code. You should get the following errors:

TIMELINE	OUTPUT	COMPILER ERRORS	MOTION EDITOR	ACTIONS - FRAME
Location		Description		
Scene 1, Layer 'Layer 1', Frame 1, Line 4		1067: Implicit coercion of a value of type MyClasses:Hero to an unrelated type flash.display:DisplayObject.		
C:\Users\Elie\Desktop\Classes\MyClasses\Hero.as, Line 1		5000: The class 'MyClasses.Hero' must subclass 'flash.display.MovieClip' since it is linked to a library symbol of that type.		

- The errors are saying that our class needs to be a subclass from the **DisplayObject** for it to be added to the stage!!! (The MovieClip class is extended from the DisplayObject class)

Extending From MovieClip

- The MovieClip Class contains a lot of properties so that the user can control an object found on the screen using code...
- So now our problem is that, we want the object to have the properties found in our class and at the same time the properties found in the MovieClip class...
- A solution for that is found in ActionScript by **extending** a class from another class.

Simple Inheritance Example

```
public class A
{
    public var position:int;
    public var scale:int;
}

public class B extends A
{
    public var health:int;
}
```

Now when we create a variable of type B, it contains all 3 variables.

NB: This is only true in our case since everything is !!!PUBLIC!!!

Extending From MovieClip

Back to our class (add the lines in red to extend it from MovieClip):

```
package MyClasses
{
    import flash.display.MovieClip;
    public class Hero extends MovieClip
    {
        public var nPosX:Number;
        public var nPosY:Number;
        public var iHealth:int;

        public function Hero(nPosX_:Number , nPosY_:Number , iHealth_:int)
        {
            nPosX = nPosX_;
            nPosY = nPosY_;
            iHealth = iHealth_;
        }
    }
}
```

- Let's run the code again.
- Now even though we specified that our position is 200 on the X and 250 on the Y, our object was shown on the top left part of the window (which is 0,0)... Why is that?

var heroCharacter:Hero = new Hero(200 , 250 , 100);

- It's because we are assigning those values to the **nPositionX** and **nPositionY** variables found in the Hero class and not the x and y variables that come in the MovieClip class (which controls the object on stage)
- In other words, we don't need **nPosX**, **nPosY** since by extending from MovieClip we get (inherit) **x** , **y**.

Using the MovieClip Properties

This is how our class should look like now:

```
package MyClasses
{
    import flash.display.MovieClip;
    public class Hero extends MovieClip
    {
        public var iHealth:int;

        public function Hero(nPosX_:Number , nPosY_:Number , iHealth_:int)
        {
            x = nPosX_;
            y = nPosY_;
            iHealth = iHealth_;
        }
    }
}
```

Inheritance

Definition

- Inheritance is a form of code reuse that allows programmers to develop new classes that are based on existing classes.
- The existing classes are often referred to as **base** classes or **superclasses**, while the new classes are usually called **subclasses**.
- A key advantage of inheritance is that it allows you to reuse code from a base class yet leave the existing code unmodified.
- Many times, rather than modifying an existing class that may have been thoroughly tested or may already be in use, using inheritance you can treat that class as an integrated module that you can extend with additional properties or methods.
- You use the **extends** keyword to indicate that a class inherits from another class.

Property Attributes

- So far inheritance seems really simple, that's not fun!!!!
- Of course we have a lot of things that will complicate our lives when we use inheritance...
- First let's learn about “***Class Property Attributes***” then we will cover some tricky stuff in inheritance.

Class Property Attributes

- In ActionScript 3.0, there is a set of attributes that can be used with any attribute of a class. The following table lists this set of attributes.

Attribute	Definition
private	Visible to references in the same class.
protected	Visible to references in the same class and derived classes.
public	Visible to references everywhere.
static	Specifies that a property belongs to the class, as opposed to instances of the class.

Note: In the coming slide I will be using the word property to represent the class attributes (variables, functions ...)

The Public Attribute

- The **public** attribute makes a property visible anywhere in your script.
- For example, to make a method available to code outside its package, you must declare the method with the public attribute (this is true for any property, whether it is declared using the var, const, or function keywords).
- In the next slide, we will run some tests using our Hero class.

The Public Attribute

- In our Hero class we declared the iHealth variable as follows:

```
public var iHealth:int;
```

- Now in the .fla file do the following:

```
var heroCharacter:Hero = new Hero(200,250,100);  
trace(heroCharacter.iHealth); /* 100 */  
heroCharacter.iHealth = 50;  
trace(heroCharacter.iHealth); /* 50 */
```

- As you can see, we were able to access the iHealth variable and change it from outside the class package.

The Private Attribute

- The **private** attribute makes an attribute (property or method) visible only to callers within the property's defining class.
- Properties that are marked as private are unavailable outside the class at both compile time and run time.
- Let's change the iHealth variable from **public** to **private**

private var iHealth:int;

- Run the code

The Private Attribute

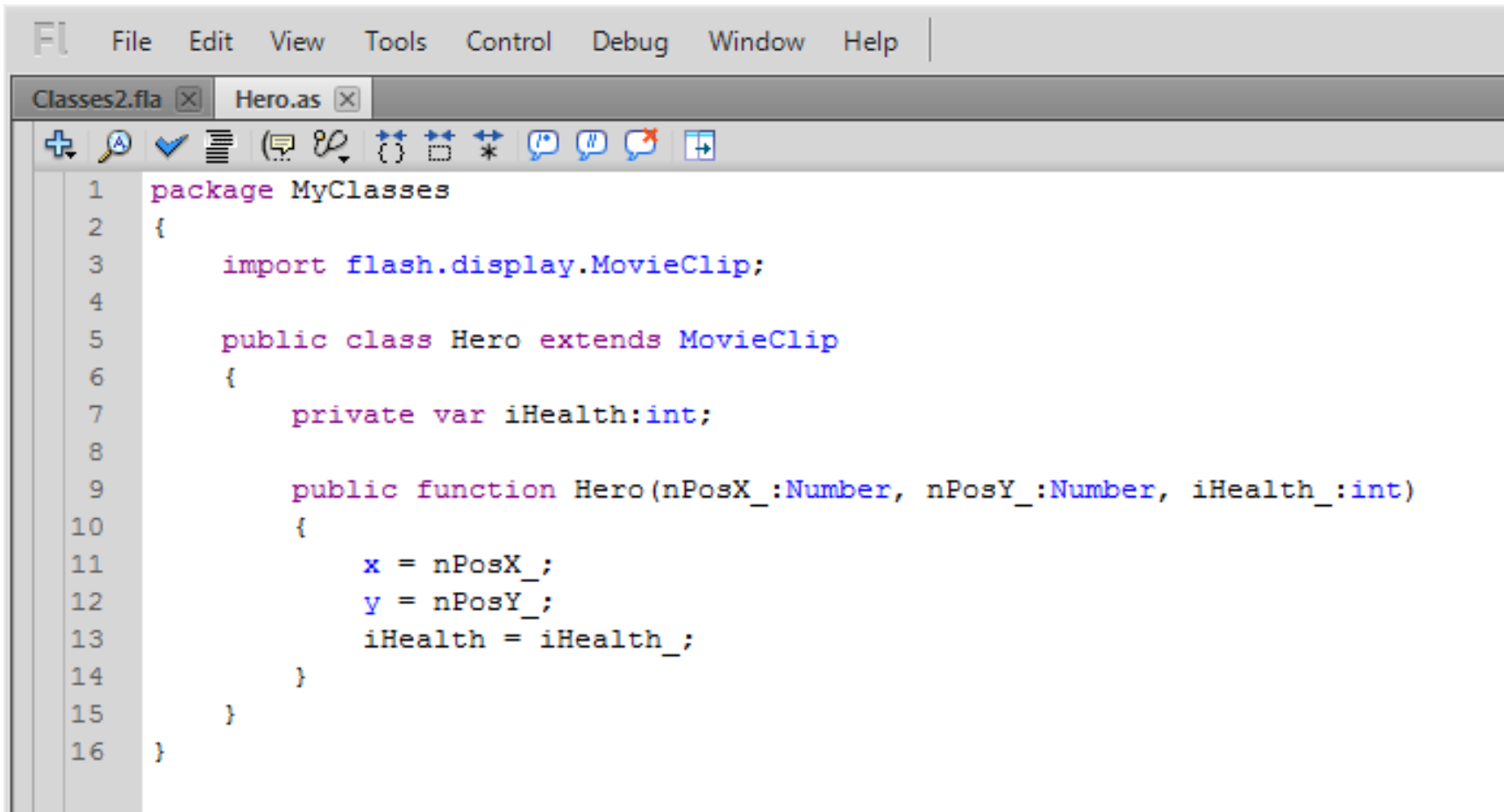
TIMELINE	OUTPUT	COMPILER ERRORS	MOTION EDITOR	ACTIONS - FRAME
Location		Description		
Scene 1, Layer 'Layer 1', Frame 1, Line 4		1178: Attempted access of inaccessible property iHealth through a reference with static type MyClasses:Hero.		
Scene 1, Layer 'Layer 1', Frame 1, Line 5		1178: Attempted access of inaccessible property iHealth through a reference with static type MyClasses:Hero.		
Scene 1, Layer 'Layer 1', Frame 1, Line 6		1178: Attempted access of inaccessible property iHealth through a reference with static type MyClasses:Hero.		

- We got the above errors saying that we are trying to access inaccessible property (we should expect that since we specifically created the “iHealth” variable as private.
- Now let's comment the following lines of code in the .fla file:

```
/*  
    trace(heroCharacter.iHealth);  
    heroCharacter.iHealth = 50;  
    trace(heroCharacter.iHealth);  
*/
```
- Run the code again, everything should be fine.

The Private Attribute

- It is important to keep in mind that, even if the variable is private you can still have access to it and change its value inside the class.



```
1 package MyClasses
2 {
3     import flash.display.MovieClip;
4
5     public class Hero extends MovieClip
6     {
7         private var iHealth:int;
8
9         public function Hero(nPosX_:Number, nPosY_:Number, iHealth_:int)
10        {
11            x = nPosX_;
12            y = nPosY_;
13            iHealth = iHealth_;
14        }
15    }
16 }
```

The Protected Attribute

- The **protected** attribute, makes a property visible to callers within its own class or in a subclass but not from the code outside the class.
- In other words, a **protected** property is available within its own class or to classes that lie anywhere below it in the inheritance hierarchy.
- The **protected** attribute is useful when you have a variable or method that your subclasses need but that you want to hide from code that is outside the inheritance chain.
- If we don't have inheritance the **protected** attribute works exactly like the **private** attribute

The Static Attribute

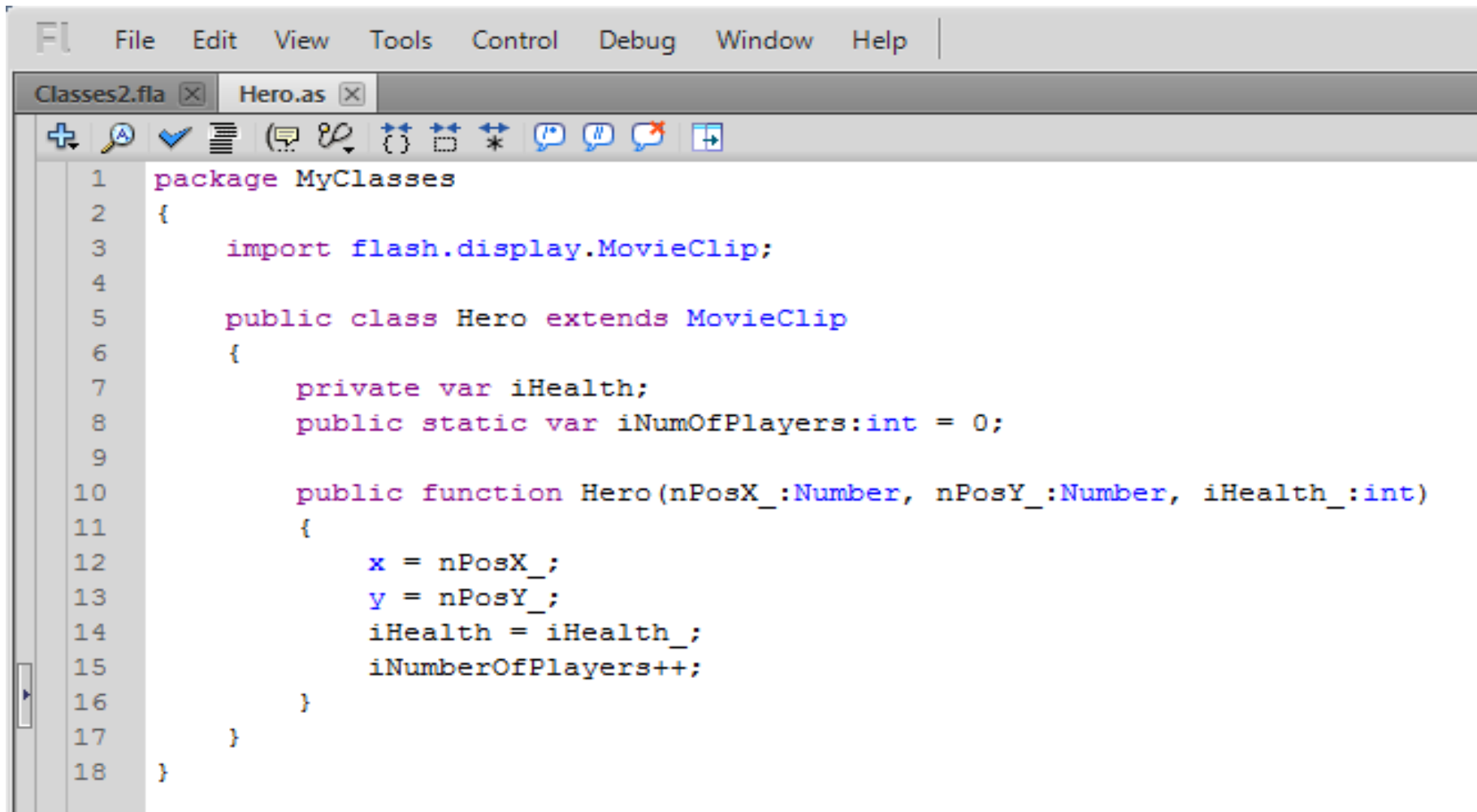
- The **static** attribute, which can be used with properties declared with the **var, const, or function keywords**, allows you to attach a property to the class rather than to instances of the class.
- Code external to the class must call static properties by using the class name instead of an instance name.
- Static properties are not inherited by subclasses, but the properties are part of a subclass's scope chain (***we will know what that means in few minutes***).

Note: The **static** attribute is used in conjunction with the “private”, “public”, “protected” and “internal” attributes.

```
public static var iNumOfPlayers:int = 0;
```

The Static Attribute

Example:

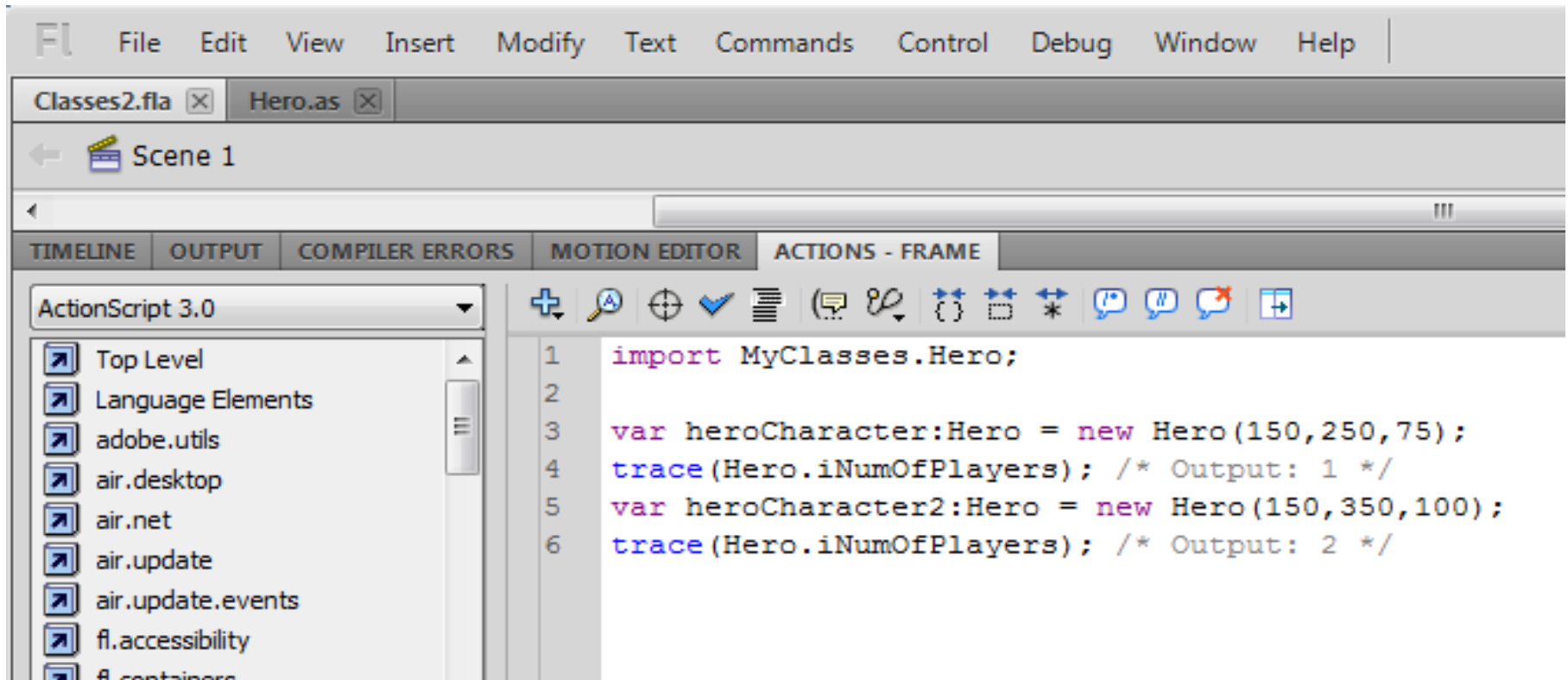


The screenshot shows an IDE window with two tabs: 'Classes2.fla' and 'Hero.as'. The 'Hero.as' tab is active, displaying the following ActionScript code:

```
1 package MyClasses
2 {
3     import flash.display.MovieClip;
4
5     public class Hero extends MovieClip
6     {
7         private var iHealth;
8         public static var iNumOfPlayers:int = 0;
9
10        public function Hero(nPosX_:Number, nPosY_:Number, iHealth_:int)
11        {
12            x = nPosX_;
13            y = nPosY_;
14            iHealth = iHealth_;
15            iNumberOfPlayers++;
16        }
17    }
18 }
```

The Static Attribute

Example (cont'd):



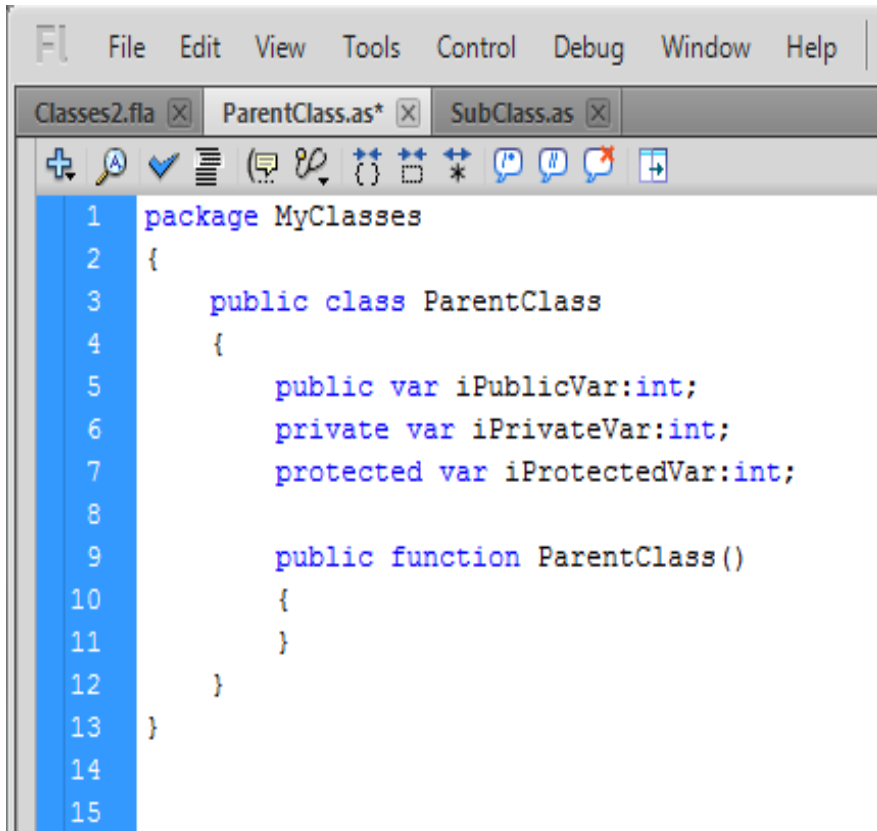
Note: As you can see, we access the static variable from the class itself and not from the instance of a class.

Attributes & Inheritance

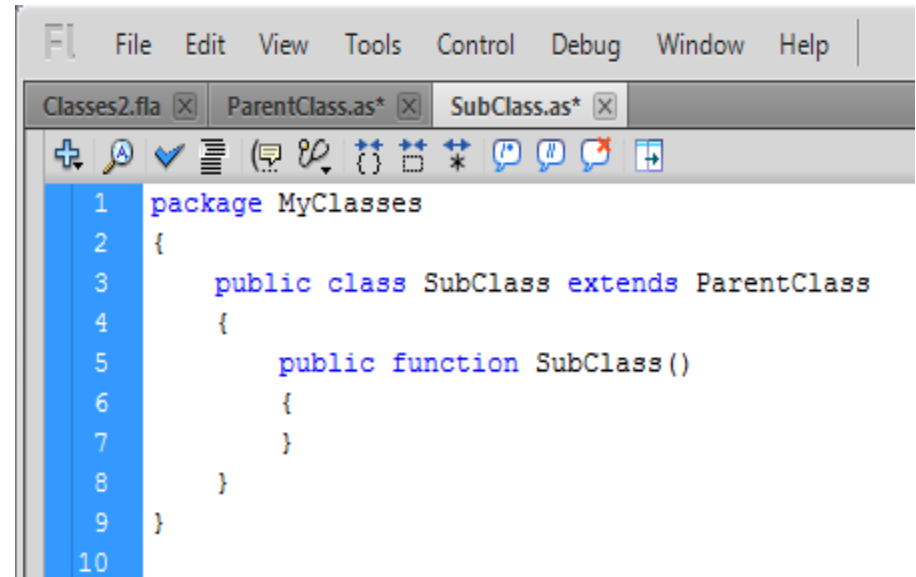
Accessibility

- If a property is declared with the **public** keyword, the property is visible to code anywhere. This means that the public keyword places no restrictions on property inheritance.
- If a property is declared with **private** keyword, it is visible only in the class that defines it, which means that it is not inherited by any subclasses.
- The **protected** keyword indicates that a property is visible not only within the class that defines it, but also to all subclasses. Again, the protected keyword does not make a property visible from outside the class but subclasses will inherit the property.

Example



```
1 package MyClasses
2 {
3     public class ParentClass
4     {
5         public var iPublicVar:int;
6         private var iPrivateVar:int;
7         protected var iProtectedVar:int;
8
9         public function ParentClass()
10        {
11        }
12    }
13 }
14
15
```



```
1 package MyClasses
2 {
3     public class SubClass extends ParentClass
4     {
5         public function SubClass()
6         {
7         }
8     }
9 }
10
```

So what did the SubClass inherit and what does it have access to??

Example (cont'd)

The SubClass inherited the following:

```
public var iPublicVar:int;  
protected var iProtectedVar:int;
```

But did NOT inherit:

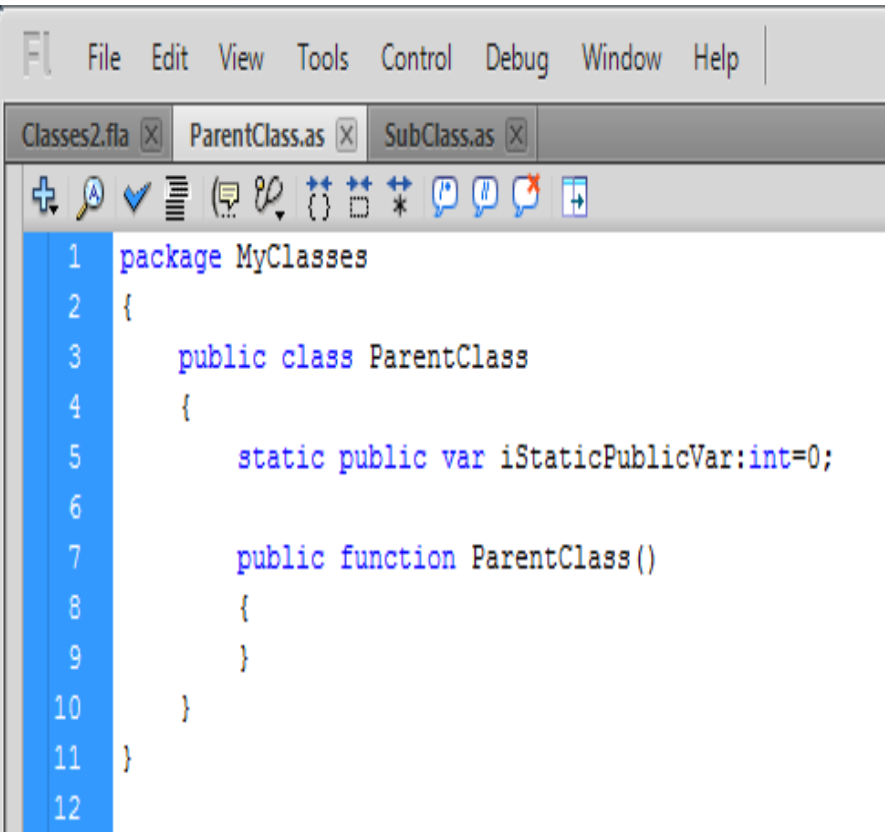
```
private var iPrivateVar:int;
```

Note: The same goes for “public”, “private” and “protected” functions if we had any in the Hero class

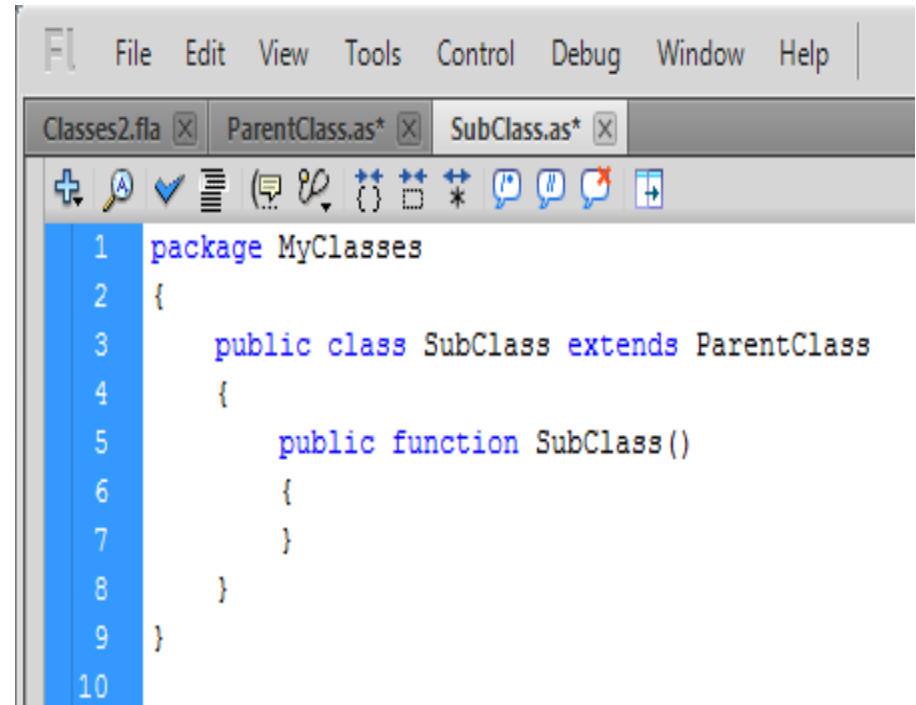
Static & Inheritance

- **Static** properties are not inherited by subclasses.
- A **static** property can be accessed only through the class object on which it is defined.
- Although **static** properties are not inherited, they are within the scope chain of the class that defines them and any subclass of that class. In other words, a static property is directly accessible within the body of the class that defines the static property and any subclass of that class.

Example On Static Inheritance



```
1 package MyClasses
2 {
3     public class ParentClass
4     {
5         static public var iStaticPublicVar:int=0;
6
7         public function ParentClass()
8         {
9         }
10    }
11 }
12
```



```
1 package MyClasses
2 {
3     public class SubClass extends ParentClass
4     {
5         public function SubClass()
6         {
7         }
8     }
9 }
10
```

Example On Static Inheritance (cont'd)

Even though, technically, the SubClass did not inherit the static variable

```
static public var iPublicStaticVar:int;
```

BUT it can access (get or set) this variable **ONLY** from inside the class because it is a public static variable. So now this variable is shared by both classes and if anyone changes it then it will be changed in all places (well, that is the purpose of the static type).

Note:

- **The same goes for static functions if we had any in the Hero class**
- **Don't forget that we can have "private" and "protected" static variables too, which will change the accessibility of the variables.**

Example On Static Inheritance (cont'd)

```

1 package MyClasses
2 {
3     public class ParentClass
4     {
5         static public var iStaticVar:int=0;
6
7         public function ParentClass()
8         {
9             iStaticVar = 10;
10        }
11    }
12 }
13

```

```

1 package MyClasses
2 {
3     public class SubClass extends ParentClass
4     {
5         public function SubClass()
6         {
7             iStaticVar++;
8             trace(iStaticVar);
9         }
10    }
11 }
12

```

In the fla:

```

1 import MyClasses.*;
2
3 var parentclass:ParentClass = new ParentClass();
4 trace(ParentClass.iStaticVar); /* Output: 10 */
5 var subclass:SubClass = new SubClass(); /* Output: 11 */
6

```


Example On Private Static Inheritance

What if we make the static variable private!!

.fla code:

```

1 package MyClasses
2 {
3     public class ParentClass
4     {
5         static private var iStaticVar:int=0;
6
7         public function ParentClass()
8         {
9             iStaticVar = 10;
10        }
11    }
12 }
13

```

```

1 import MyClasses.*;
2
3 trace(ParentClass.iStaticVar);
4
5

```

It will lead to the following Error:

TIMELINE	OUTPUT	COMPILER ERRORS	MOTION EDITOR	ACTIONS - FRAME
Location		Description		
Scene 1, Layer 'Layer 1', Frame 1, Line 3		1178: Attempted access of inaccessible property iStaticVar through a reference with static type Class.		

Attributes & Inheritance

It is up to you to play around with different attributes and check out how the accessibility changes and which variables get inherited !!!!

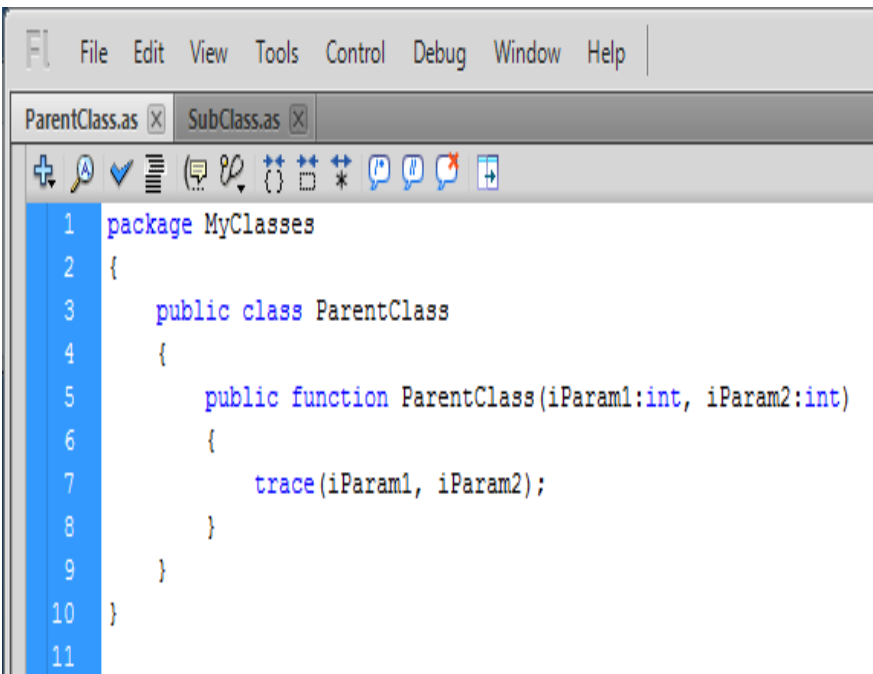
!!! HAVE FUN !!!

Constructors & Inheritance

The super keyword

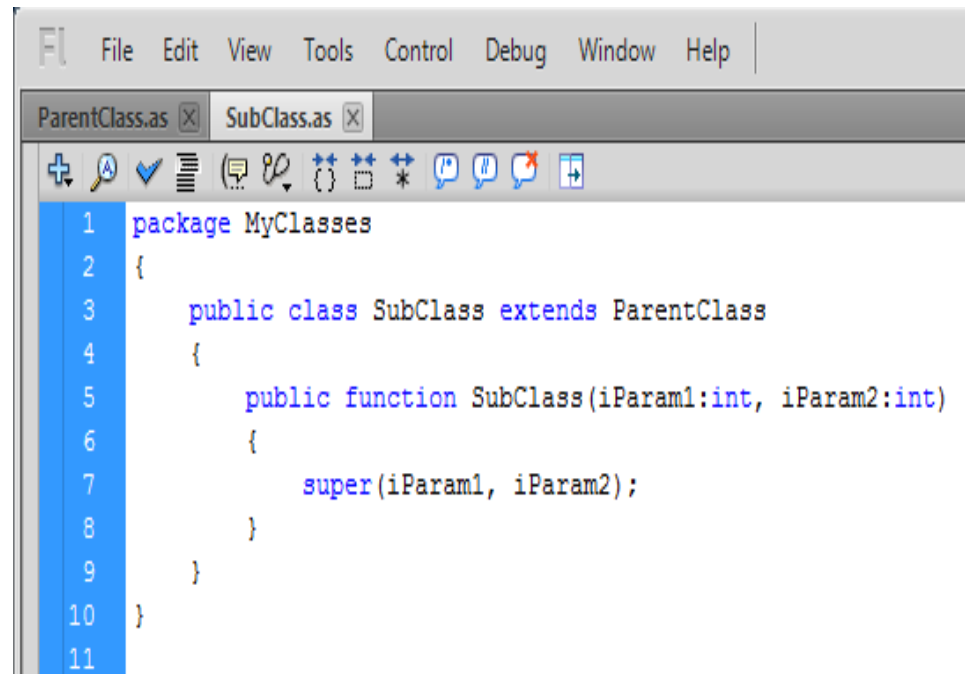
- A constructor can make an explicit call to the constructor of its direct superclass by using the **super()** statement.
- If the superclass constructor is not explicitly called, the compiler will consider that the superclass only has a default constructor and will automatically insert a call before the first statement in the subclass constructor body.
- You can also call methods of the superclass by using the **super** prefix as a reference to the superclass.

Example



A screenshot of an IDE window titled 'ParentClass.as'. The menu bar includes File, Edit, View, Tools, Control, Debug, Window, and Help. The toolbar contains icons for file operations, code navigation, and execution. The code is as follows:

```
1 package MyClasses
2 {
3     public class ParentClass
4     {
5         public function ParentClass(iParam1:int, iParam2:int)
6         {
7             trace(iParam1, iParam2);
8         }
9     }
10 }
11
```



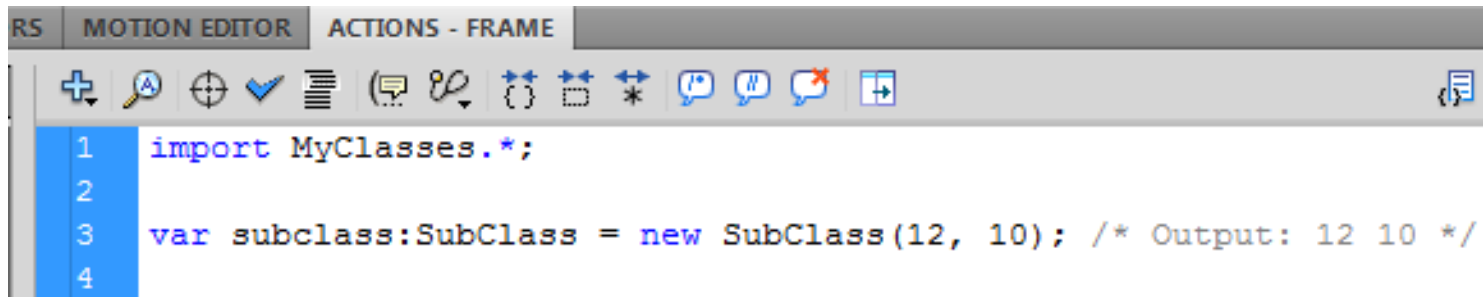
A screenshot of an IDE window titled 'SubClass.as'. The menu bar includes File, Edit, View, Tools, Control, Debug, Window, and Help. The toolbar contains icons for file operations, code navigation, and execution. The code is as follows:

```
1 package MyClasses
2 {
3     public class SubClass extends ParentClass
4     {
5         public function SubClass(iParam1:int, iParam2:int)
6         {
7             super(iParam1, iParam2);
8         }
9     }
10 }
11
```

In the SubClass we are calling the ParentClass constructor

super(iParam1, iParam2);

Example (cont'd)



```
1 import MyClasses.*;
2
3 var subclass:SubClass = new SubClass(12, 10); /* Output: 12 10 */
4
```

Where did the “12 10” output come from???!!! It’s because the SubClass is calling the ParentClass constructor where we actually trace the values sent.

No Default Constructor Error

- If you don't call the base class constructor you will get the following error:

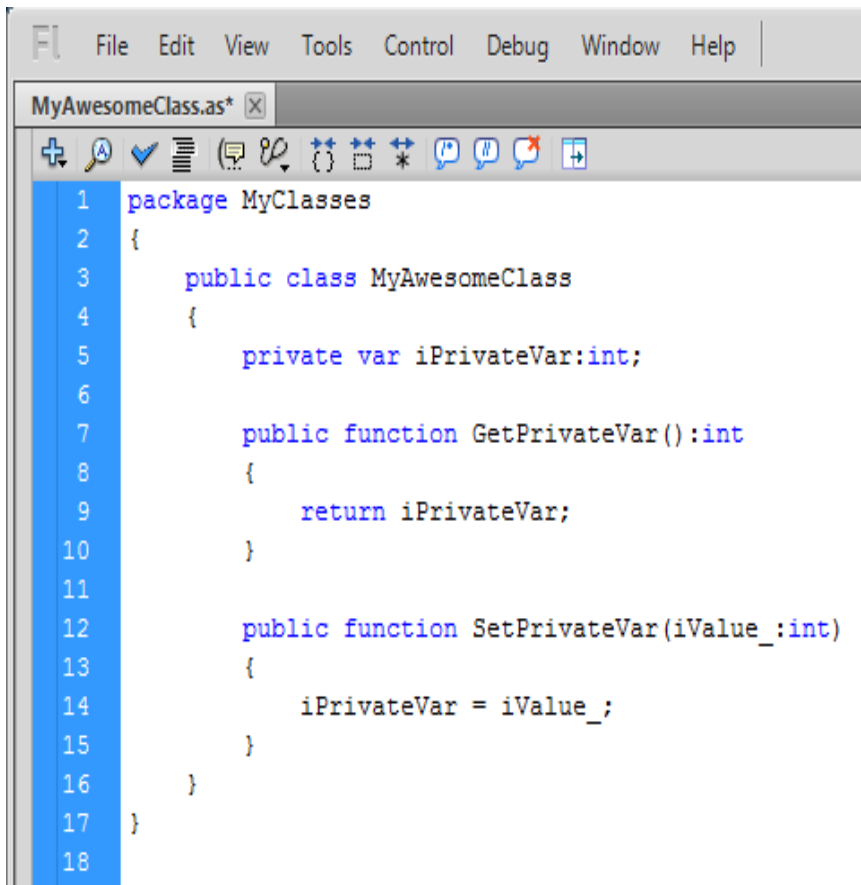
TIMELINE	OUTPUT	COMPILER ERRORS	MOTION EDITOR	ACTIONS - FRAME	
Location Description					
C:\Users\... 1203: No default constructor found in base class MyClasses:ParentClass.					

which means that the compiler is trying to call the **default constructor** in the ParentClass but is not finding one.

Getters & Setters

Getters & Setters

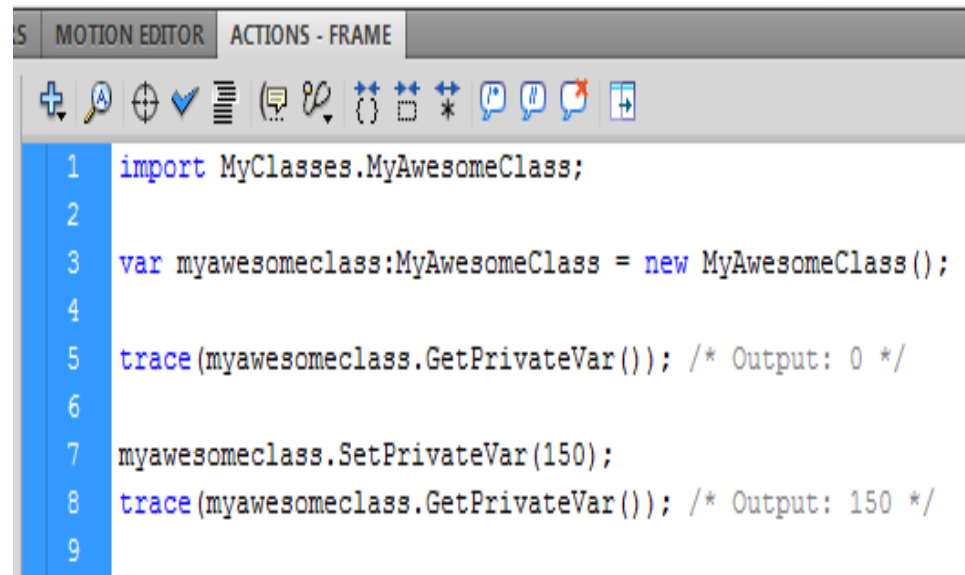
- In many languages (C++, C# ...) “**Getters**” and “**Setters**”, allow you to keep your class properties private to the class, but allow users of your class to access those properties.



```

1 package MyClasses
2 {
3     public class MyAwesomeClass
4     {
5         private var iPrivateVar:int;
6
7         public function GetPrivateVar():int
8         {
9             return iPrivateVar;
10        }
11
12        public function SetPrivateVar(iValue_:int)
13        {
14            iPrivateVar = iValue_;
15        }
16    }
17 }
18
  
```

Code in the class (.as file)



```

1 import MyClasses.MyAwesomeClass;
2
3 var myawesomeclass:MyAwesomeClass = new MyAwesomeClass();
4
5 trace(myawesomeclass.GetPrivateVar()); /* Output: 0 */
6
7 myawesomeclass.SetPrivateVar(150);
8 trace(myawesomeclass.GetPrivateVar()); /* Output: 150 */
9
  
```

Code in the layer (.fla file)

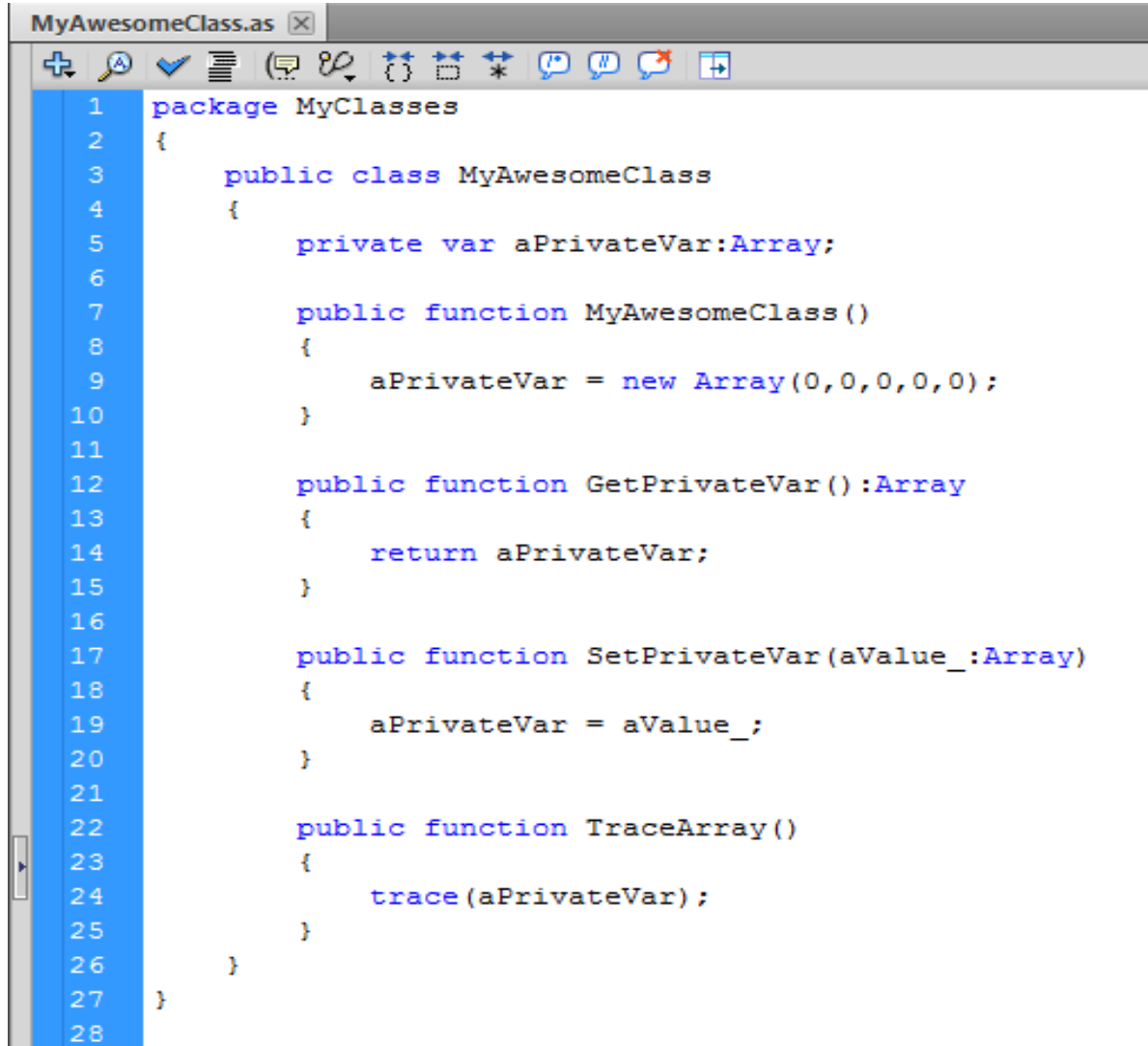
Getters & Setters

- Now the question is, why make the variable private with a getter & setter function to it ?
- By doing that we are giving the user access but the developer has control

```
MyAwesomeClass.as
1 package MyClasses
2 {
3     public class MyAwesomeClass
4     {
5         private var iPrivateVar:int;
6
7         public function GetPrivateVar():int
8         {
9             return iPrivateVar;
10        }
11
12        public function SetPrivateVar(iValue_:int)
13        {
14            if(iValue_ > 100)
15            {
16                iPrivateVar = 100;
17            }
18            else if(iValue_ < 0)
19            {
20                iPrivateVar = 0;
21            }
22            else
23            {
24                iPrivateVar = iValue_;
25            }
26        }
27    }
28 }
29
```

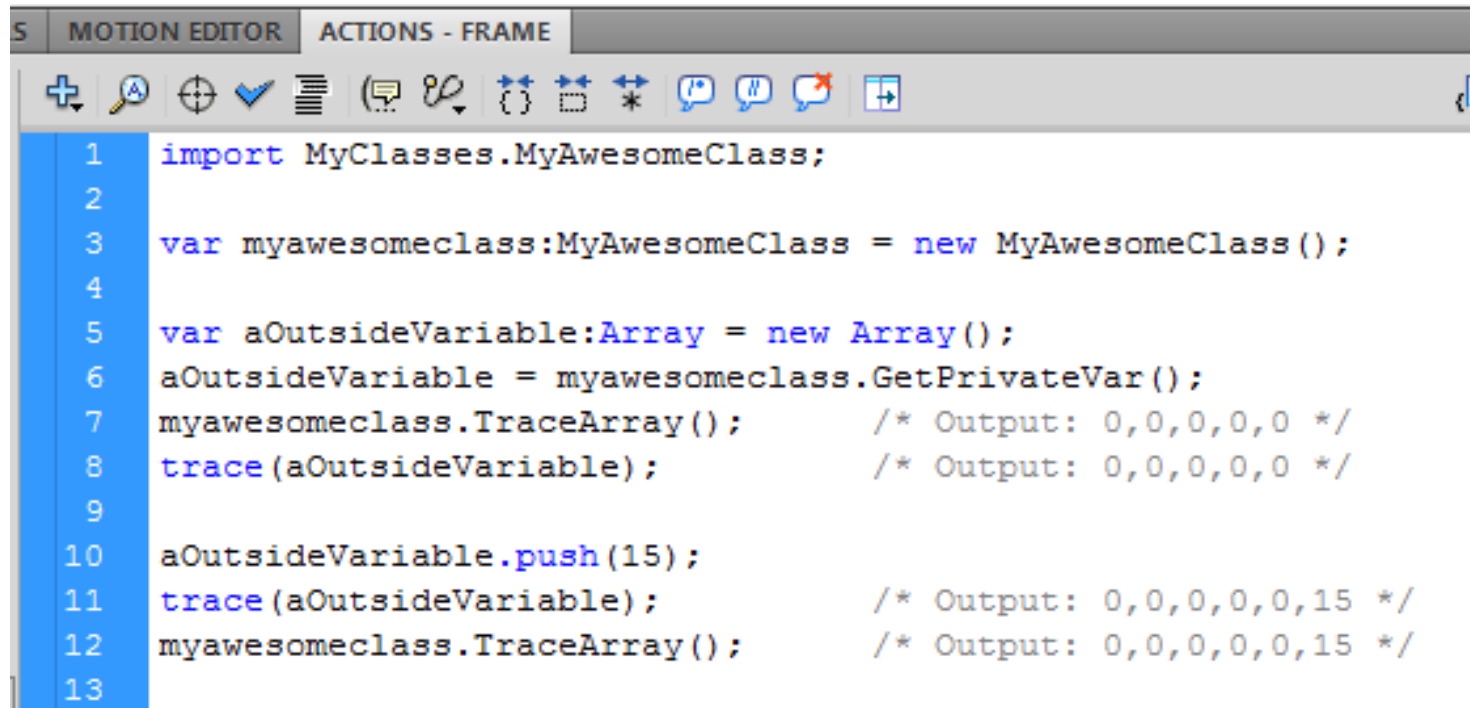
```
RS MOTION EDITOR ACTIONS - FRAME
1 import MyClasses.MyAwesomeClass;
2
3 var myawesomeclass:MyAwesomeClass = new MyAwesomeClass();
4
5 trace(myawesomeclass.GetPrivateVar()); /* Output: 0 */
6
7 myawesomeclass.SetPrivateVar(150);
8 trace(myawesomeclass.GetPrivateVar()); /* Output: 100 */
9
10 myawesomeclass.SetPrivateVar(-10);
11 trace(myawesomeclass.GetPrivateVar()); /* Output: 0 */
12
13 myawesomeclass.SetPrivateVar(50);
14 trace(myawesomeclass.GetPrivateVar()); /* Output: 50 */
15
```

Complex types with getters and setters



```
1 package MyClasses
2 {
3     public class MyAwesomeClass
4     {
5         private var aPrivateVar:Array;
6
7         public function MyAwesomeClass()
8         {
9             aPrivateVar = new Array(0,0,0,0,0);
10        }
11
12        public function GetPrivateVar():Array
13        {
14            return aPrivateVar;
15        }
16
17        public function SetPrivateVar(aValue_:Array)
18        {
19            aPrivateVar = aValue_;
20        }
21
22        public function TraceArray()
23        {
24            trace(aPrivateVar);
25        }
26    }
27 }
28
```

Complex types with getters and setters



```
1 import MyClasses.MyAwesomeClass;
2
3 var myawesomeclass:MyAwesomeClass = new MyAwesomeClass();
4
5 var aOutsideVariable:Array = new Array();
6 aOutsideVariable = myawesomeclass.GetPrivateVar();
7 myawesomeclass.TraceArray();          /* Output: 0,0,0,0,0 */
8 trace(aOutsideVariable);              /* Output: 0,0,0,0,0 */
9
10 aOutsideVariable.push(15);
11 trace(aOutsideVariable);              /* Output: 0,0,0,0,0,15 */
12 myawesomeclass.TraceArray();          /* Output: 0,0,0,0,0,15 */
13
```

- As you can see, we were able to get the private Array and change its content using a variable created outside the class “aOutsideVariable”.
- This is something very important to think about when adding getters and setters to a complex type variable.

Getters & Setters in ActionScript

- ActionScript has its special way to create **getters** and **setters** which will allow users of your class to access those properties as if they were accessing a class variable instead of calling a class method.

```

MyAwesomeClass.as
1 package MyClasses
2 {
3     public class MyAwesomeClass
4     {
5         private var iPrivateVar:int;
6
7         public function get PrivateVar():int
8         {
9             return iPrivateVar;
10        }
11
12        public function set PrivateVar(iValue_:int)
13        {
14            if(iValue_ > 100)
15            {
16                iPrivateVar = 100;
17            }
18            else if(iValue_ < 0)
19            {
20                iPrivateVar = 0;
21            }
22            else
23            {
24                iPrivateVar = iValue_;
25            }
26        }
27    }
28 }
29

```

```

15 MOTION EDITOR ACTIONS - FRAME
1 import MyClasses.MyAwesomeClass;
2
3 var myawesomeclass:MyAwesomeClass = new MyAwesomeClass();
4
5 trace(myawesomeclass.PrivateVar); /* Output: 0 */
6
7 myawesomeclass.PrivateVar = 150;
8 trace(myawesomeclass.PrivateVar); /* Output: 100 */
9
10 myawesomeclass.PrivateVar = -10;
11 trace(myawesomeclass.PrivateVar); /* Output: 0 */
12
13 myawesomeclass.PrivateVar = 50;
14 trace(myawesomeclass.PrivateVar); /* Output: 50 */
15

```

The End 😊