

CS 175

Action Script

Classes 1

What's a class?

- A class is an abstract representation of an object. It stores information about the types of data that an object can hold and the behaviors that an object can exhibit.
- When creating a class you are creating your custom type that contains other types and functions.
- The usefulness of such an abstraction may not be apparent when you write small scripts that contain only a few objects interacting with one another.
- As the scope of a program grows, however, and the number of objects that must be managed increases, you may find that classes allow you to better control how objects are created and how they interact with one another.

Class Definition

```
public class ClassName
{
    attributes.....
}
```

Proper syntax for a class definition:

- **class keyword**
- **class name**
- *The class body, which is enclosed by curly braces ({})*

NB: Don't worry about the “public” keyword for now. I will explain all about that in another chapter.

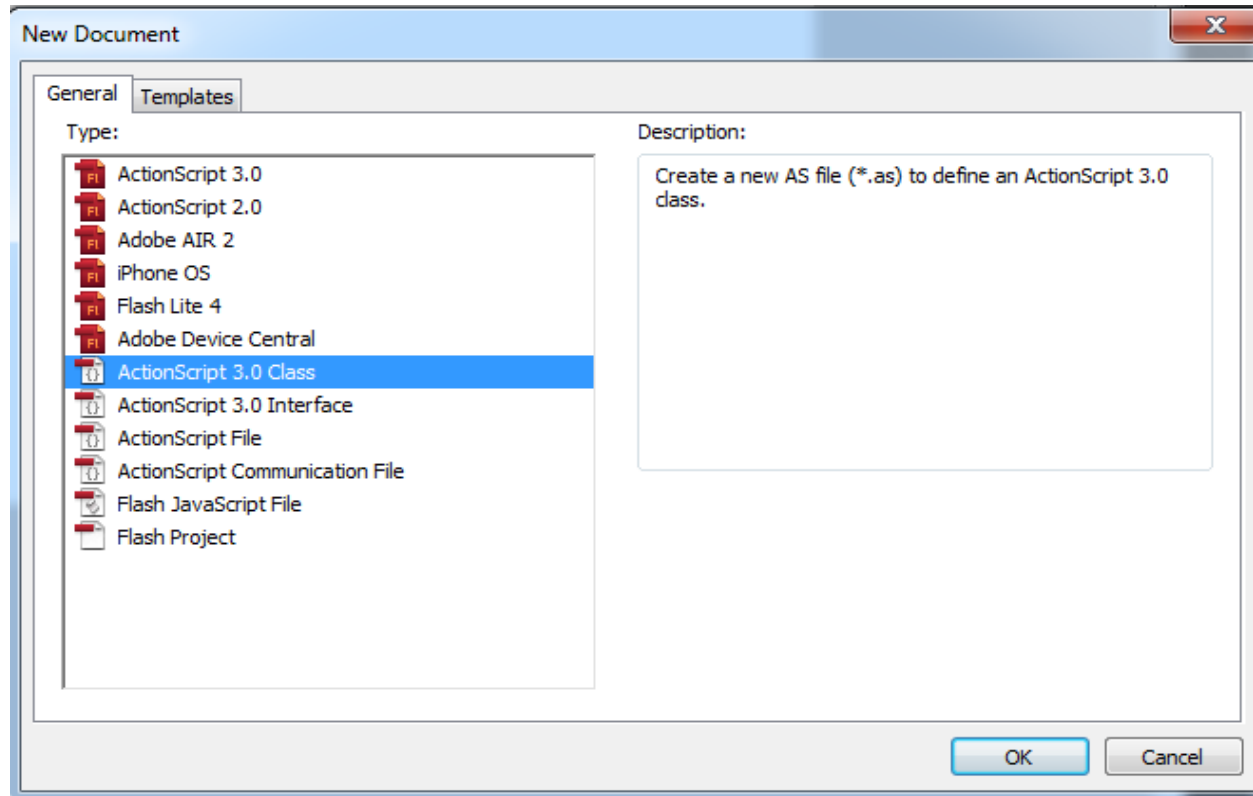
Class Definition

For example, the following code creates a class named Hero that contains multiple attributes (properties and methods)

```
public class Hero  
{  
    public var nPosX:Number;           /* Hero's x position */  
    public var nPosY:Number;         /* Hero's y position */  
    public var iHealth:int;          /* Hero's health variable */  
  
    public function Initialize()  
    {  
        /* Function's body */  
    }  
}
```

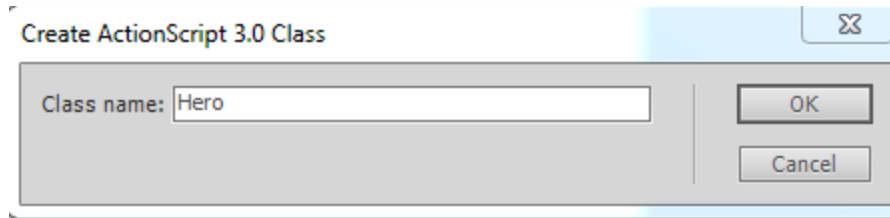
Class Creation

- Every class will be created in its separate ActionScript file (.as)
- To create an ActionScript file in **CS5 or CS5.5**
 - File > New > “ActionScript 3.0 Class”



Class Creation

- It will ask you for the class name (*In our case “Hero”*)

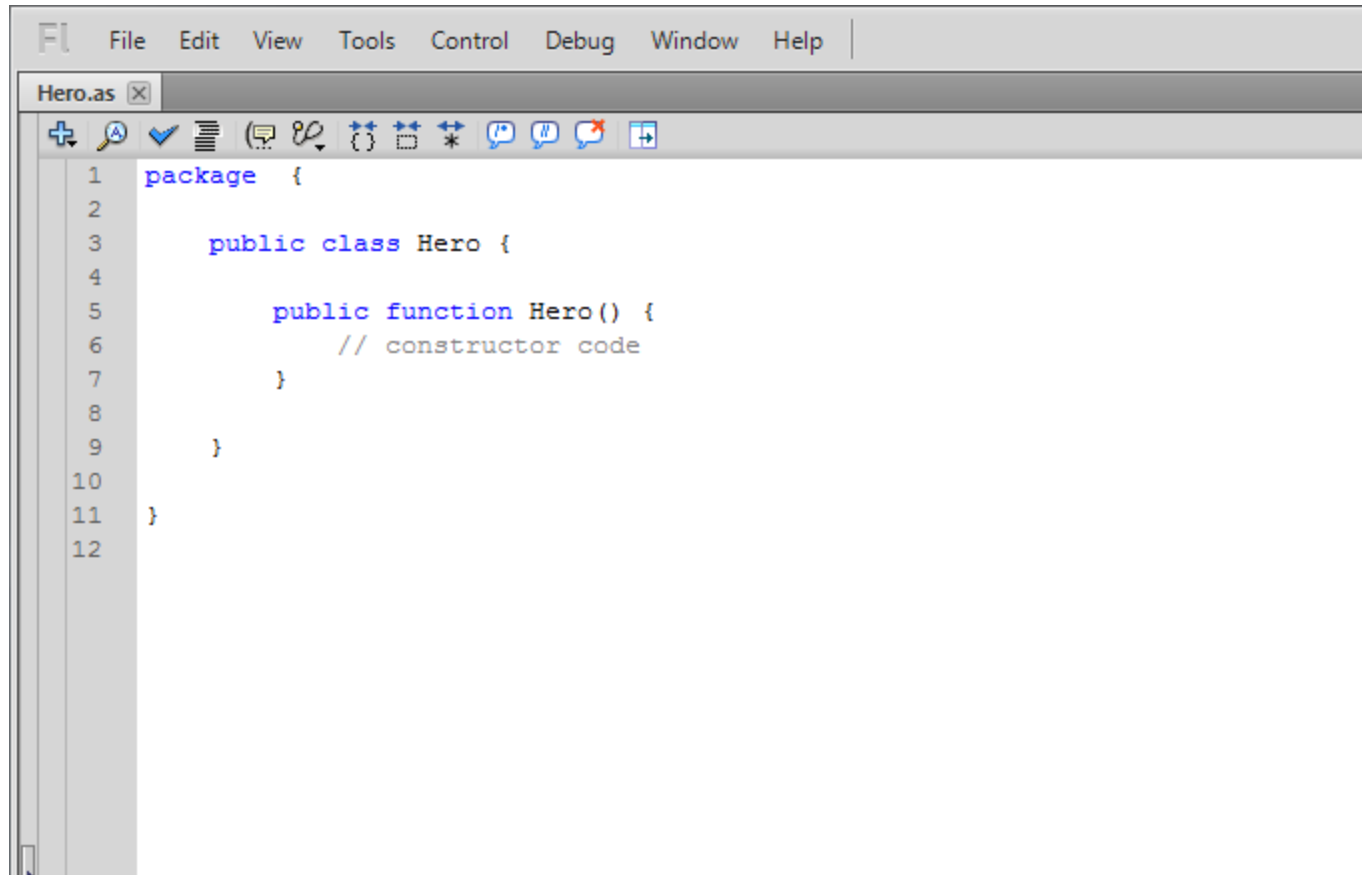


- Save the “.as” file in the same folder as your “.fla”

NB: ALWAYS name the “.as” file the same name as the class you are creating!!!!

Class Creation

- The following should be the result:

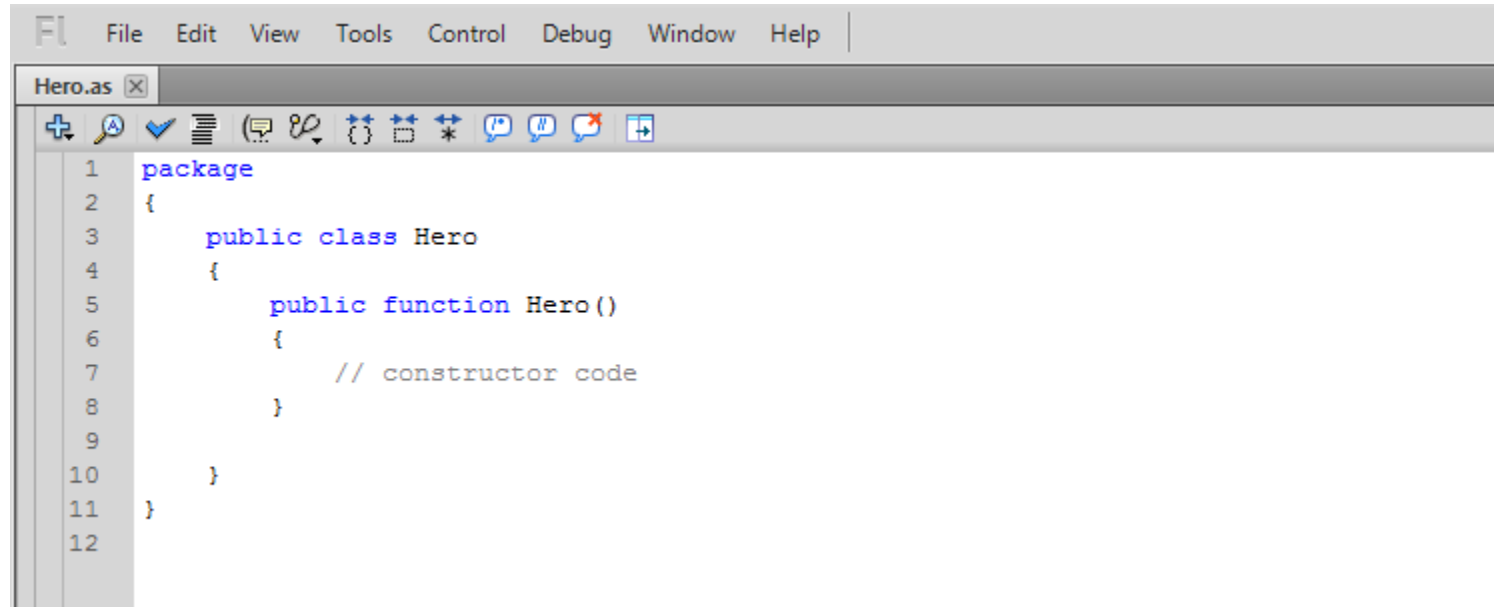
A screenshot of an IDE window titled 'Hero.as'. The window has a menu bar with 'File', 'Edit', 'View', 'Tools', 'Control', 'Debug', 'Window', and 'Help'. Below the menu bar is a toolbar with various icons. The code is displayed in a text area with line numbers 1 through 12 on the left. The code is as follows:

```
1 package {  
2  
3     public class Hero {  
4  
5         public function Hero() {  
6             // constructor code  
7         }  
8  
9     }  
10  
11 }  
12
```

Note: Restructure the code so that it follows the conventions I taught you in CS116.

Class Creation

- The code after it was restructured

A screenshot of a code editor window titled 'Hero.as'. The editor has a menu bar with 'File', 'Edit', 'View', 'Tools', 'Control', 'Debug', 'Window', and 'Help'. Below the menu is a toolbar with various icons for editing and development. The code is as follows:

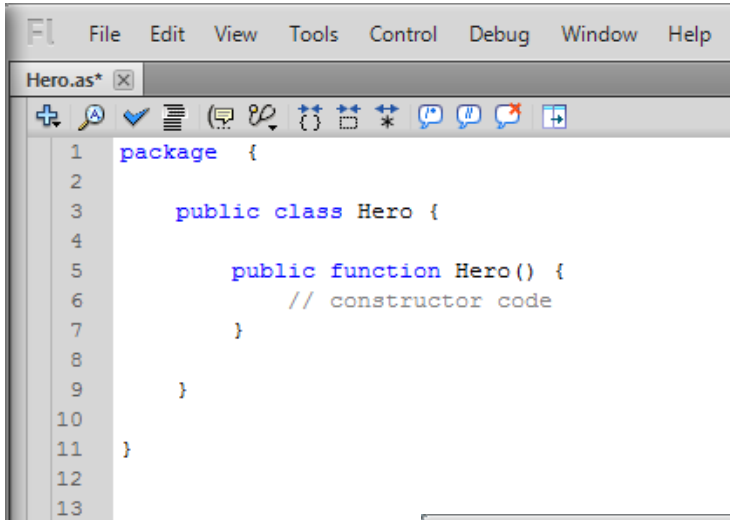
```
1 package
2 {
3     public class Hero
4     {
5         public function Hero()
6         {
7             // constructor code
8         }
9     }
10 }
11
12
```

- I will teach you about “packages” and “constructors” very soon but for now let’s just take the code that we previously wrote and add it to the class.

Class Creation

- Small trick to restructure

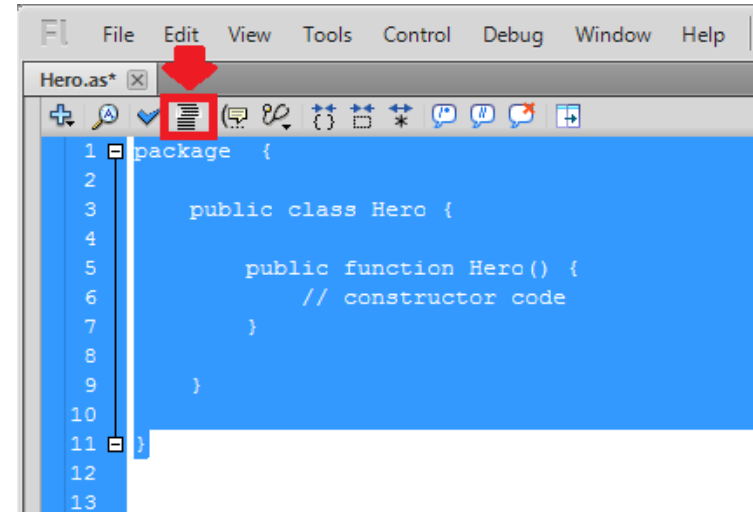
1



```

1 package {
2
3     public class Hero {
4
5         public function Hero() {
6             // constructor code
7         }
8
9     }
10
11 }
12
13
  
```

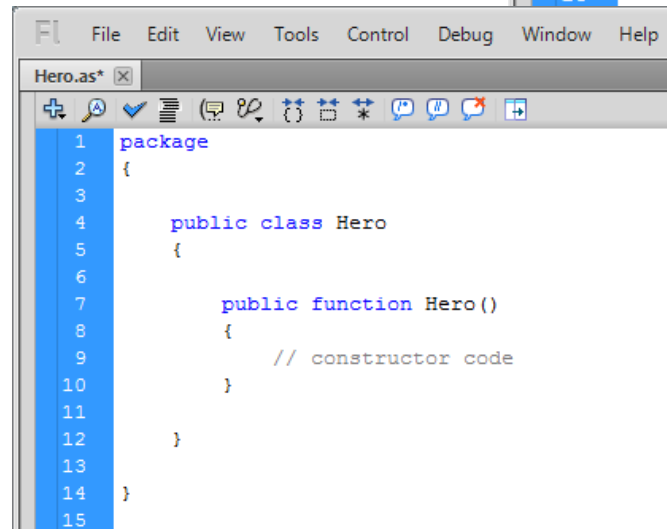
2



```

1 package {
2
3     public class Hero {
4
5         public function Hero() {
6             // constructor code
7         }
8
9     }
10
11 }
12
13
  
```

3

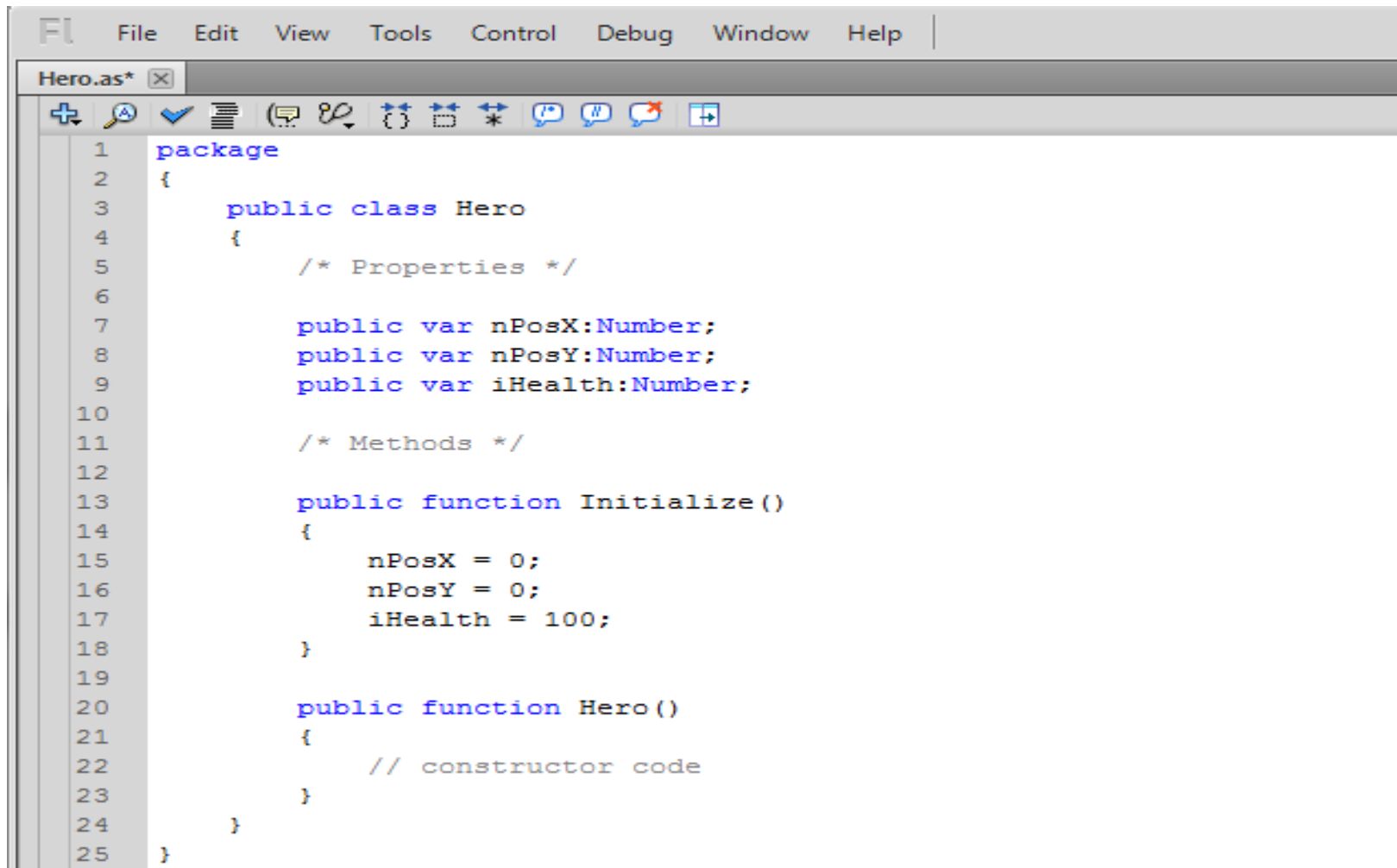


```

1 package
2 {
3
4     public class Hero
5     {
6
7         public function Hero()
8         {
9             // constructor code
10        }
11
12    }
13
14 }
15
  
```

Class Creation

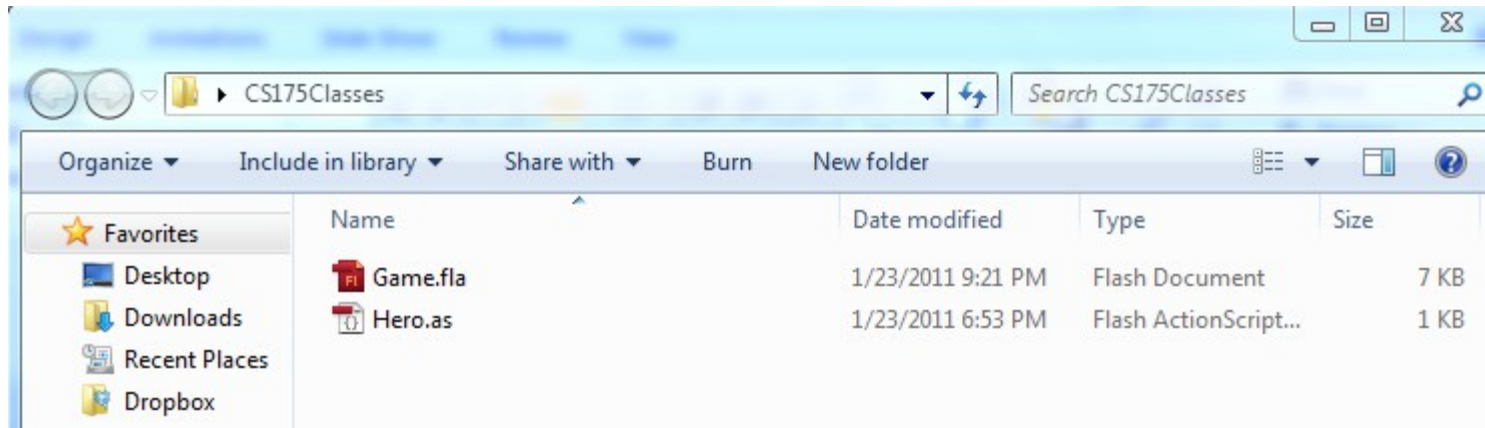
- This is our class so far:



```
1 package
2 {
3     public class Hero
4     {
5         /* Properties */
6
7         public var nPosX:Number;
8         public var nPosY:Number;
9         public var iHealth:Number;
10
11         /* Methods */
12
13         public function Initialize()
14         {
15             nPosX = 0;
16             nPosY = 0;
17             iHealth = 100;
18         }
19
20         public function Hero()
21         {
22             // constructor code
23         }
24     }
25 }
```

Class Instance Creation

- At this point, we have a Hero.as file that contains our Hero class.
- Let's create a ".fla" file and name it Game.fla
 - File > New > "ActionScript 3.0"
 - Make sure you save the ".fla" file in the same folder as your ".as" file
- Our folder should look as follows:



Class Instance Creation

- Since both files are next to each other in the folder, the “.fla” file can see the “.as” file. That will allow us to create variables from the new type (Hero) that we just created.
- Write the following code in the Action’s tab inside the “.fla”:

```
var heroCharacter:Hero = new Hero();
```

- Now that we created a variable of type “Hero”, we can access all the attributes inside it.

Accessing Class Attributes

The “.” operator will let you access properties and methods inside the class (after you create it of course).

- Write the following code in the Action's tab inside the “.fla”:

```
var heroCharacter:Hero = new Hero();
```

```
/* Accessing the “Initialize” function */  
heroCharacter.Initialize();
```

```
/* Accessing and tracing the iHealth_ value */  
trace(heroCharacter.iHealth);
```

Class Constructor

Why do we use **new ClassName()** to create a class?

```
var VariableName:ClassName = new ClassName();
```

By doing that we are creating an object of type “ClassName” (in the example below the type is Hero) and calling its constructor.

Eg:

```
var heroCharacter:Hero = new Hero();
```

Class Constructor

What's a **Constructor**? What are the rules to follow?

- A constructor is a function inside a class that has the same name as the class.
- It will be called only once, when you create an object of type that class.
- The constructor can take parameters but can never have a return type(since by default it should return an object of that class' type).
- Usually the constructor is used to initialize the properties of a class.
- If no constructors are created, ActionScript will add a default constructor to the class, which looks like this `ClassName()` and doesn't do anything.
- You can only have **ONE** constructor in a class (**this rule doesn't apply for other languages like C++**).

Using The Constructor

In our code now, we are using the default constructor (which doesn't do anything) and an “**Initialize**” function to initialize the properties inside the class.

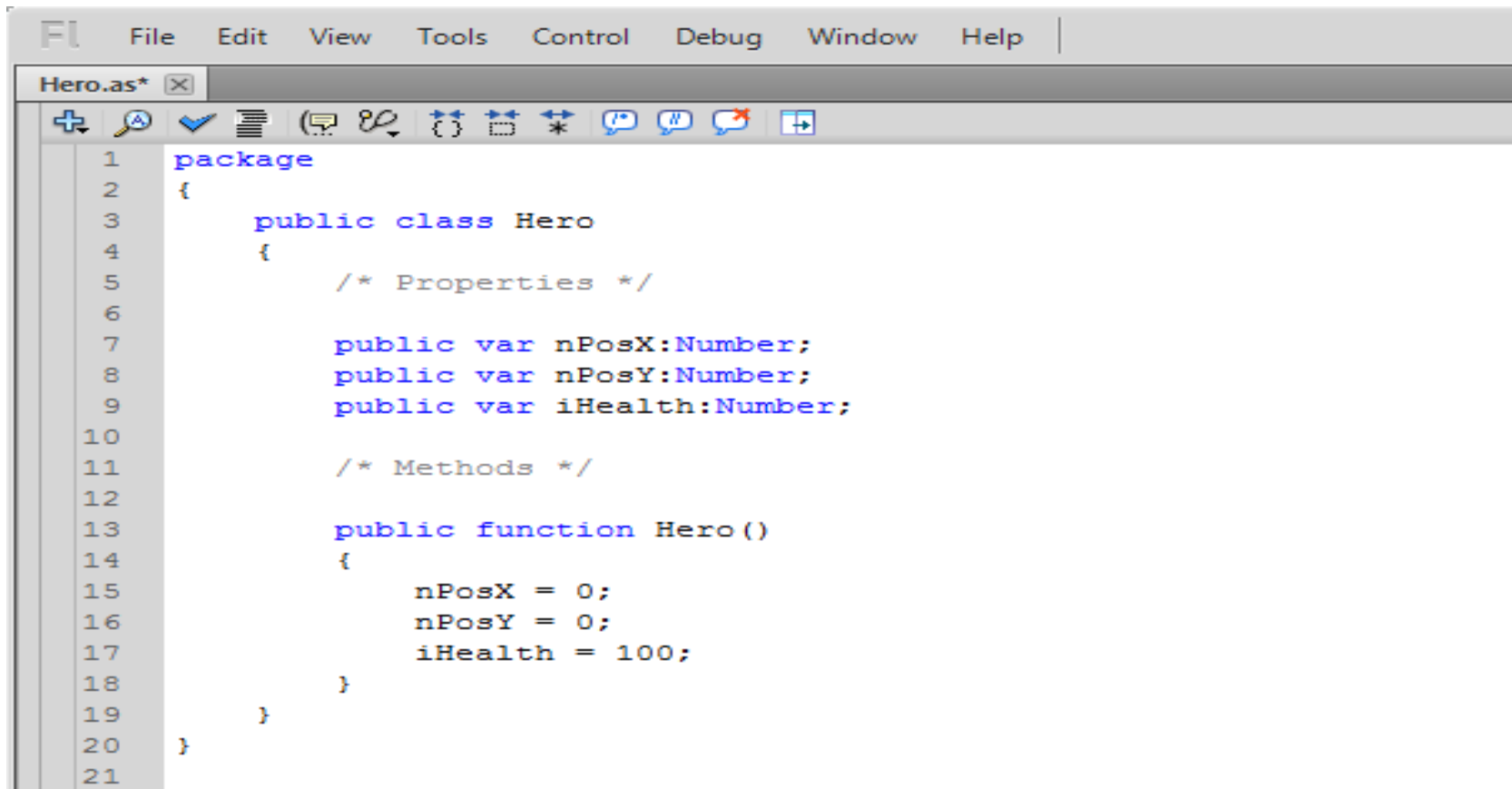
But wait!! Isn't that the constructor's role??

We need to get rid of the “Initialize” function and let the constructor do its job.

- Take the code from the “Initialize” function and put it inside the constructor
- Delete the Initialize function since we don't need it anymore.

Using The Constructor

Below is our class with the new changes:



The screenshot shows a code editor window titled 'Hero.as*' with a menu bar (File, Edit, View, Tools, Control, Debug, Window, Help) and a toolbar. The code is as follows:

```
1 package
2 {
3     public class Hero
4     {
5         /* Properties */
6
7         public var nPosX:Number;
8         public var nPosY:Number;
9         public var iHealth:Number;
10
11        /* Methods */
12
13        public function Hero()
14        {
15            nPosX = 0;
16            nPosY = 0;
17            iHealth = 100;
18        }
19    }
20 }
21
```

Using The Constructor

The code in the “.fla” becomes:

```
var heroCharacter:Hero = new Hero();  
trace(heroCharacter.iHealth);
```

The constructor is initializing all the variables now.

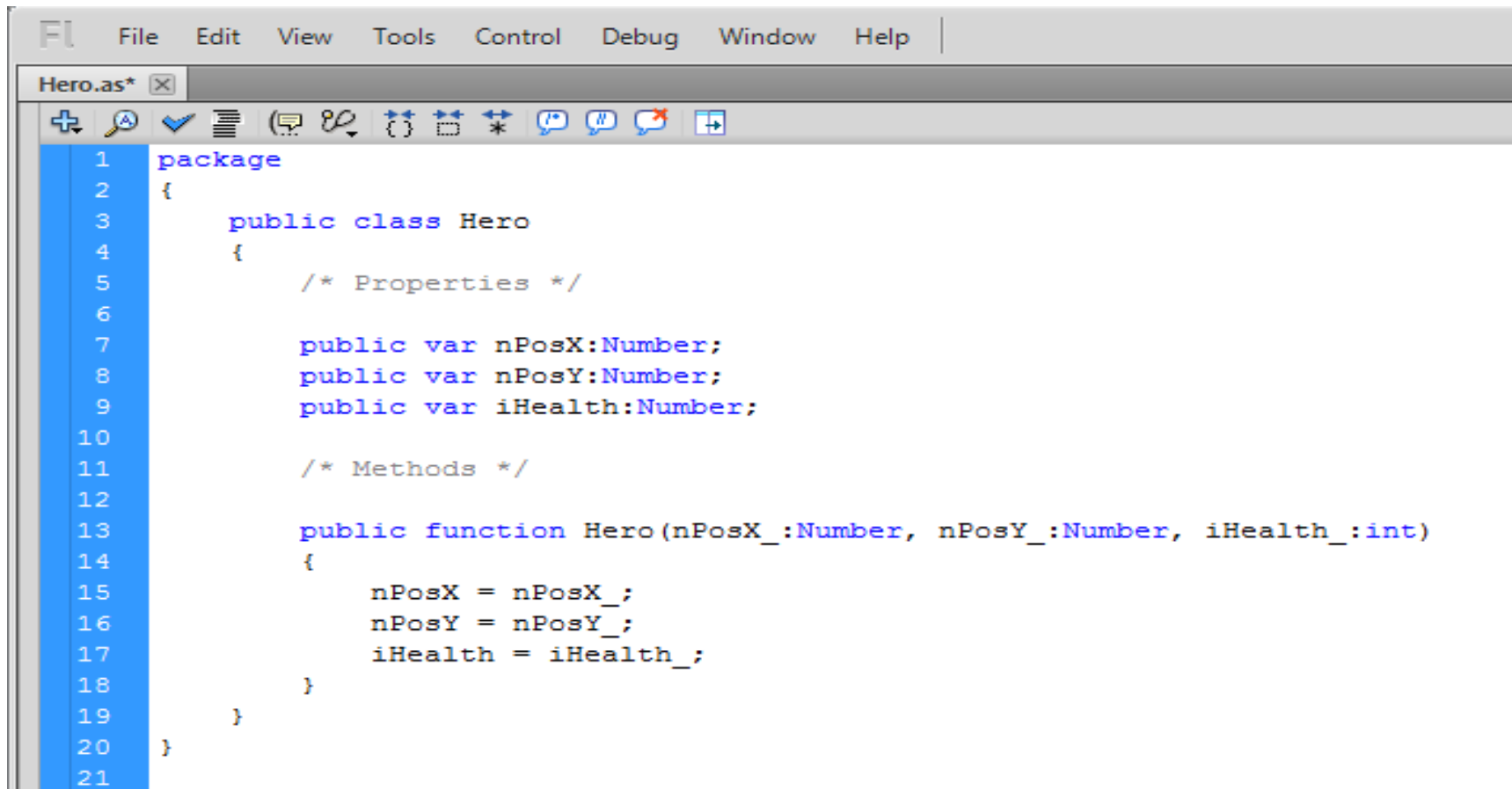
What's next?!

How about we add parameters to the constructor to give the user more flexibility when creating instances from this class...

We will let the user enter the position (X & Y) and health desired values at creation time.

Using The Constructor

Our new constructor:



The screenshot shows a code editor window titled 'Hero.as*' with a menu bar (File, Edit, View, Tools, Control, Debug, Window, Help) and a toolbar. The code is as follows:

```
1 package
2 {
3     public class Hero
4     {
5         /* Properties */
6
7         public var nPosX:Number;
8         public var nPosY:Number;
9         public var iHealth:Number;
10
11         /* Methods */
12
13         public function Hero(nPosX_:Number, nPosY_:Number, iHealth_:int)
14         {
15             nPosX = nPosX_;
16             nPosY = nPosY_;
17             iHealth = iHealth_;
18         }
19     }
20 }
21
```

Using The Constructor

The code in the “.fla” becomes:

```
var heroCharacter:Hero = new Hero(10,55,100);  
trace(heroCharacter.nPosX); /* Output: 10 */  
trace(heroCharacter.nPosY); /* Output: 55 */  
trace(heroCharacter.iHealth); /* Output: 100 */
```

- As you can see in the code above, the user now has the ability to specify the initial values for the properties inside the class.
- By doing that, we can specify different initial values for each instance created from our class.

```
/* nPosX = 10   nPosY = 55   iHealth = 100 */  
var heroCharacter:Hero = new Hero(10,55,100);  
  
/* nPosX = 30   nPosY = 155   iHealth = 80 */  
var heroCharacter2:Hero = new Hero(30,155,80);
```

Packages

- A package is used to specify in which folder your .as file can be located at.

Eg:

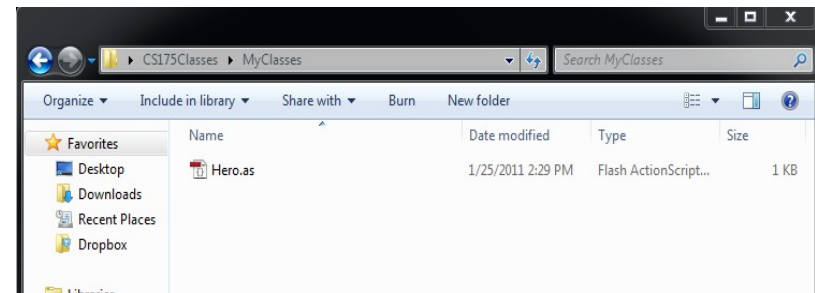
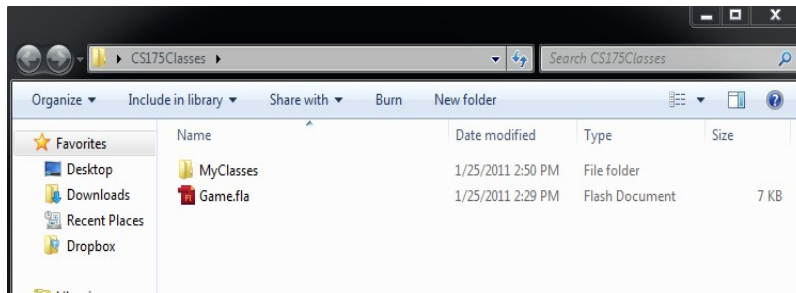
```
package DirectoryHierarchy
{
    public class ClassName
    {
        .....
    }
}
```

- If the “.as” file is located next to the “.fla” file (which our case right now) then we do not to specify anything for the **“DirectoryHierarchy”** (you will just leave it blank)

Packages

Example:

Let's say, in order to better organize our files, we put our .as files (classes) in a folder called MyClasses next to our .fla file



So when creating your class you will specify that next to the package keyword

Eg:

```
package MyClasses
{
    public class Hero
    {
        .....
    }
}
```

Packages

Example:

- Do the same hierarchy, update the Hero class package and run it... If you followed the previous steps correctly you should get the following error:

TIMELINE	OUTPUT	COMPILER ERRORS	MOTION EDITOR	ACTIONS - FRAME
Location		Description		
Scene 1, Layer 'Layer 1', Frame 1, Line 1		1046: Type was not found or was not a compile-time constant: Hero.		
Scene 1, Layer 'Layer 1', Frame 1, Line 1		1180: Call to a possibly undefined method Hero.		

- This is saying that the “.fla” file doesn’t see the Hero class, so the type doesn’t exist.

Packages

Solution:

- Previously we had the “.as” file next to the “.fla” file so everything was fine.
- But since we made changes and restructured our files, the “.fla” doesn’t see the “.as” file anymore.
- In order to fix the problem, we need to tell the “.fla” where our file is, and we do that by importing (using the **import** keyword).
- Add the following at the beginning of the “.fla” code:

import MyClasses.Hero;

- Now the “.fla” knows where to find the Hero class.

The End 😊