360|Flex Atlanta

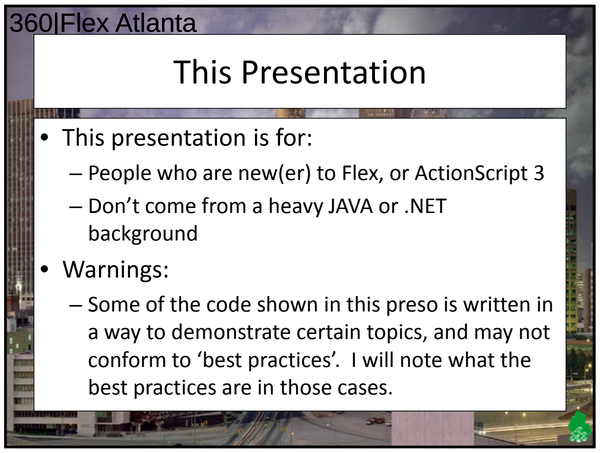
Introduction to Object Oriented Programming Fundamentals in Flex

Nick Kwiatkowski
Michigan Flex Users' Group
Michigan State University

360|Flex Atlanta

About Me

- Work for Michigan State University, in the Telecommunication Systems department
- Been developing Flex applications for just about 2 years; starting with Flex 1.5
- Co-Manager of the Mid-Michigan ColdFusion User's Group
- Manager of the Michigan Flex User's Group



360|Flex Atlanta

This Presentation

- This presentation is for:
  - People who are new(er) to Flex, or ActionScript 3
  - Don't come from a heavy JAVA or .NET background
- Warnings:
  - Some of the code shown in this preso is written in a way to demonstrate certain topics, and may not conform to 'best practices'. I will note what the best practices are in those cases.

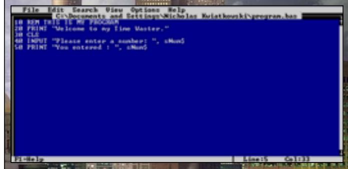
Two Styles of Programming

- In the modern day, there are two major styles of programming that are used:
  - Procedural Programming
  - OOP (Object-Oriented Programming)
- Some languages allow you to easily decide which style of programming you wish to use, others demand the use of one or the other.

Two Styles of Programming

Procedural programming mirrors the way a computer would execute a program.

The programmer would need to figure out how to break their application down into simple tasks that are executed in a 'script'.

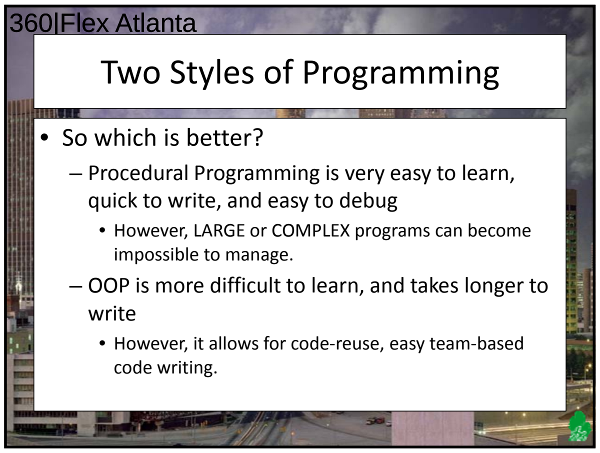

File Edit Search View Options Help
C:\Documents and Settings\Nicholas Kwiatkowski\program.bas
F1=Help
Line:5 Col:33

```
10 REM THIS IS MY PROGRAM
20 PRINT "Welcome to my Time Waster."
30 CLS
40 INPUT "Please enter a number: ", sNum$
50 PRINT "You entered : ", sNum$
```

Two Styles of Programming

Object Oriented Programming rather takes the approach that you should have "objects" available to you to perform certain tasks.

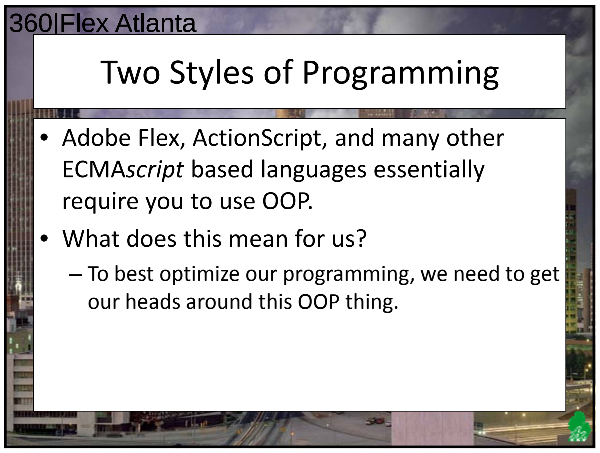
Your program does not follow a single script to do things, rather you empower objects to do them for you.



360|Flex Atlanta

Two Styles of Programming

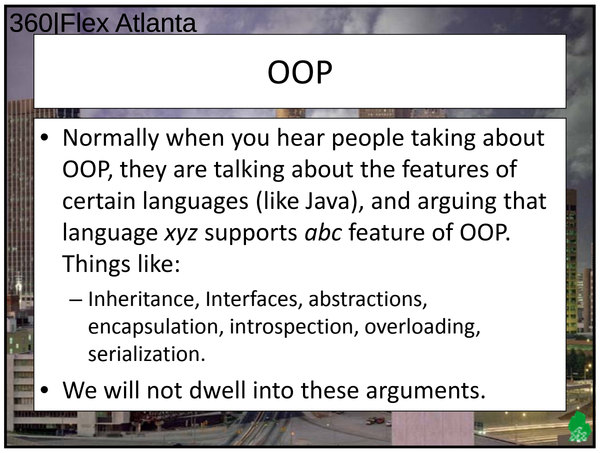
- So which is better?
  - Procedural Programming is very easy to learn, quick to write, and easy to debug
    - However, LARGE or COMPLEX programs can become impossible to manage.
  - OOP is more difficult to learn, and takes longer to write
    - However, it allows for code-reuse, easy team-based code writing.



360|Flex Atlanta

Two Styles of Programming

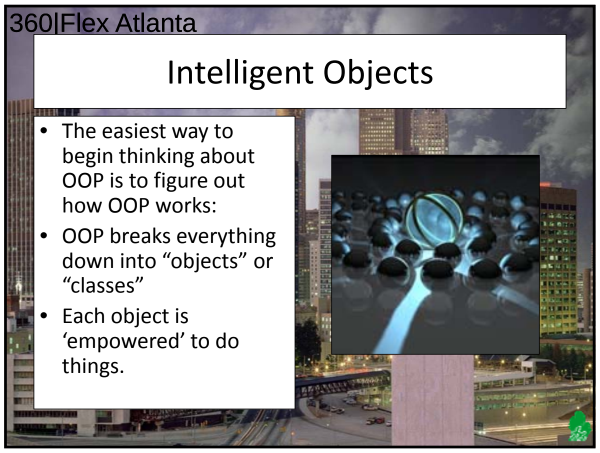
- Adobe Flex, ActionScript, and many other ECMA*script* based languages essentially require you to use OOP.
- What does this mean for us?
  - To best optimize our programming, we need to get our heads around this OOP thing.



360|Flex Atlanta

OOP

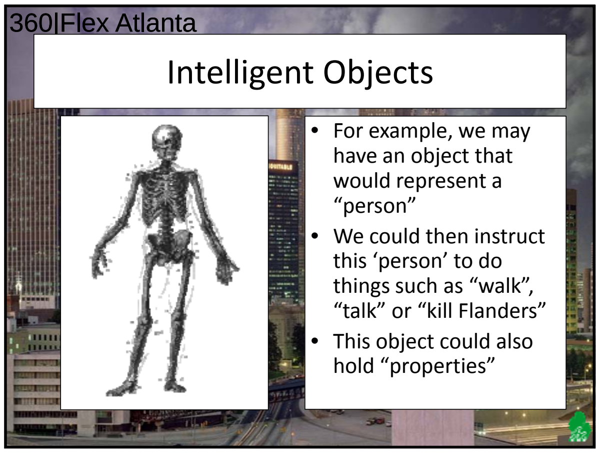
- Normally when you hear people taking about OOP, they are talking about the features of certain languages (like Java), and arguing that language *xyz* supports *abc* feature of OOP. Things like:
  - Inheritance, Interfaces, abstractions, encapsulation, introspection, overloading, serialization.
- We will not dwell into these arguments.



360|Flex Atlanta

Intelligent Objects

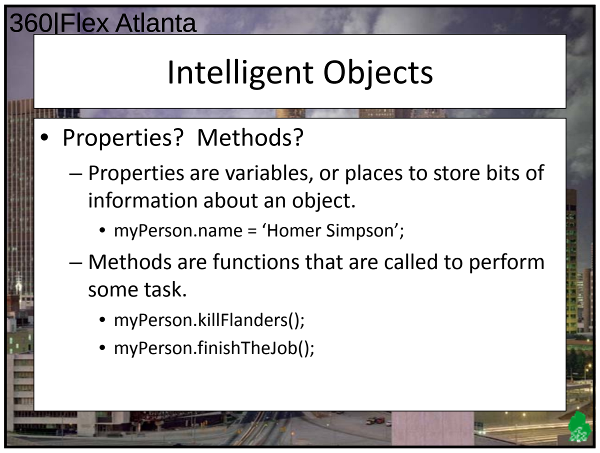
- The easiest way to begin thinking about OOP is to figure out how OOP works:
- OOP breaks everything down into “objects” or “classes”
- Each object is ‘empowered’ to do things.



360|Flex Atlanta

Intelligent Objects

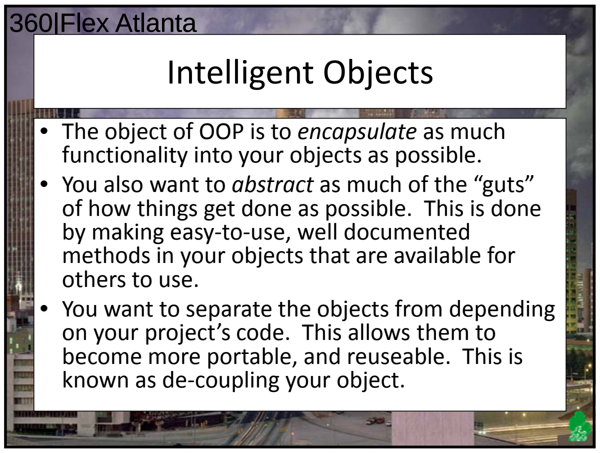
- For example, we may have an object that would represent a “person”
- We could then instruct this ‘person’ to do things such as “walk”, “talk” or “kill Flanders”
- This object could also hold “properties”



360|Flex Atlanta

Intelligent Objects

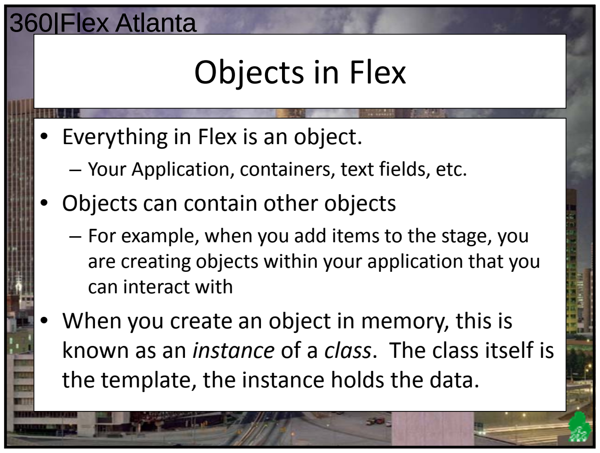
- Properties? Methods?
  - Properties are variables, or places to store bits of information about an object.
    - myPerson.name = ‘Homer Simpson’;
  - Methods are functions that are called to perform some task.
    - myPerson.killFlanders();
    - myPerson.finishTheJob();



360|Flex Atlanta

Intelligent Objects

- The object of OOP is to *encapsulate* as much functionality into your objects as possible.
- You also want to *abstract* as much of the "guts" of how things get done as possible. This is done by making easy-to-use, well documented methods in your objects that are available for others to use.
- You want to separate the objects from depending on your project's code. This allows them to become more portable, and reuseable. This is known as de-coupling your object.



360|Flex Atlanta

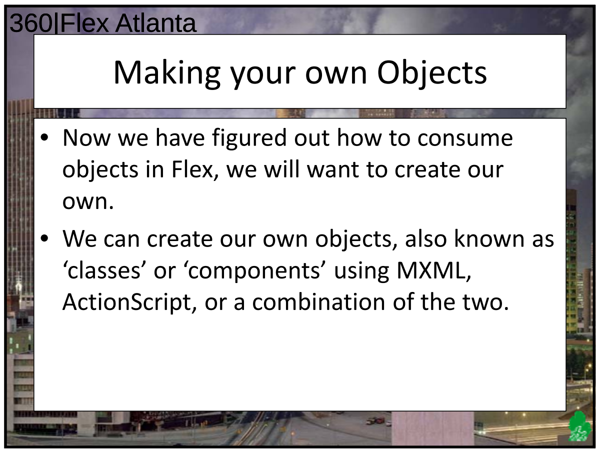
Objects in Flex

- Everything in Flex is an object.
  - Your Application, containers, text fields, etc.
- Objects can contain other objects
  - For example, when you add items to the stage, you are creating objects within your application that you can interact with
- When you create an object in memory, this is known as an *instance* of a *class*. The class itself is the template, the instance holds the data.

360|Flex Atlanta

Using the Person object in a simple project

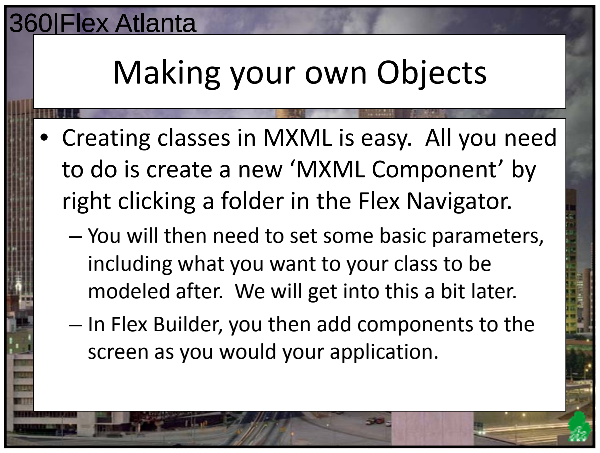
SIMPLE OBJECT DEMO



360|Flex Atlanta

Making your own Objects

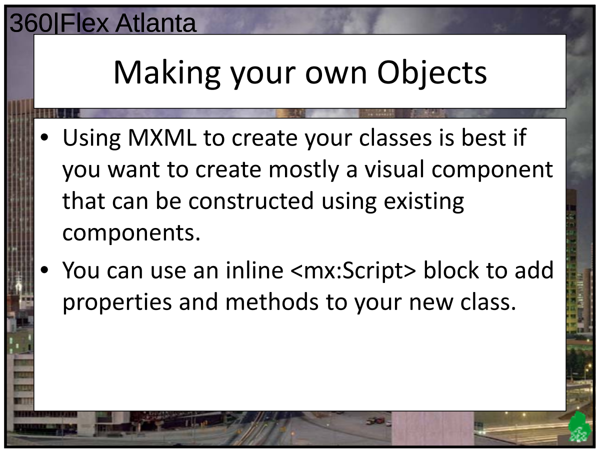
- Now we have figured out how to consume objects in Flex, we will want to create our own.
- We can create our own objects, also known as ‘classes’ or ‘components’ using MXML, ActionScript, or a combination of the two.



360|Flex Atlanta

Making your own Objects

- Creating classes in MXML is easy. All you need to do is create a new ‘MXML Component’ by right clicking a folder in the Flex Navigator.
  – You will then need to set some basic parameters, including what you want to your class to be modeled after. We will get into this a bit later.
  – In Flex Builder, you then add components to the screen as you would your application.



360|Flex Atlanta

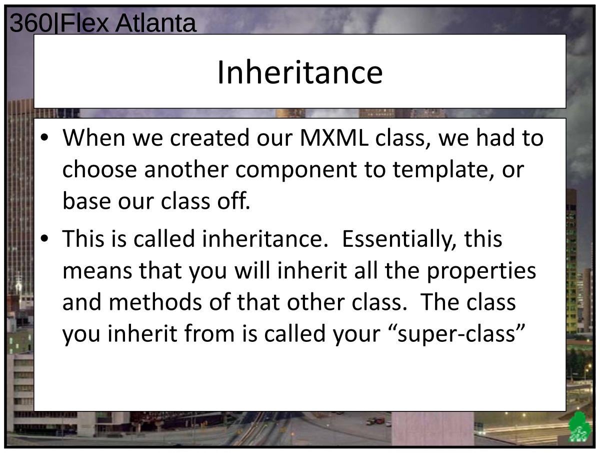
Making your own Objects

- Using MXML to create your classes is best if you want to create mostly a visual component that can be constructed using existing components.
- You can use an inline <mx:Script> block to add properties and methods to your new class.

360|Flex Atlanta

Creating a simple MXML visual-based class

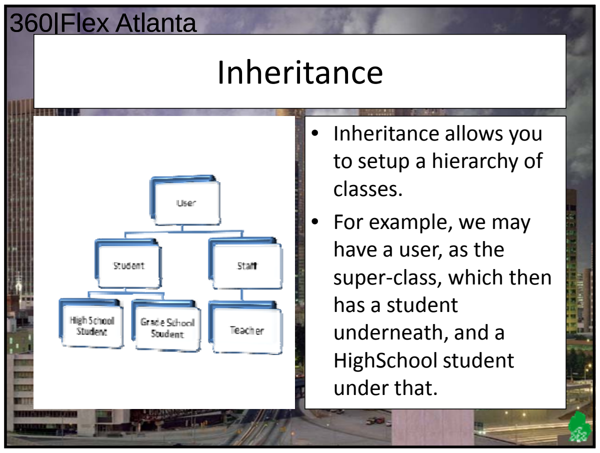
CREATING A SIMPLE CLASS



360|Flex Atlanta

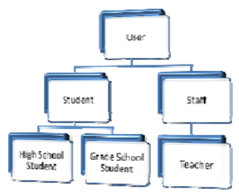
Inheritance

- When we created our MXML class, we had to choose another component to template, or base our class off.
- This is called inheritance. Essentially, this means that you will inherit all the properties and methods of that other class. The class you inherit from is called your “super-class”

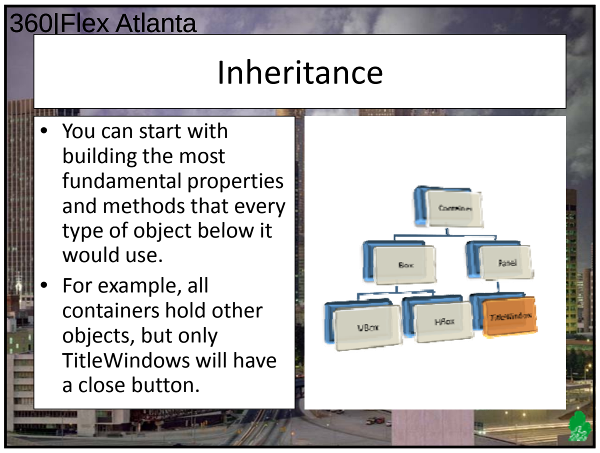


360|Flex Atlanta

Inheritance



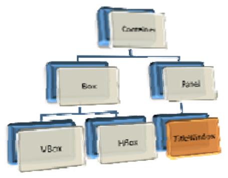
- Inheritance allows you to setup a hierarchy of classes.
- For example, we may have a user, as the super-class, which then has a student underneath, and a HighSchool student under that.

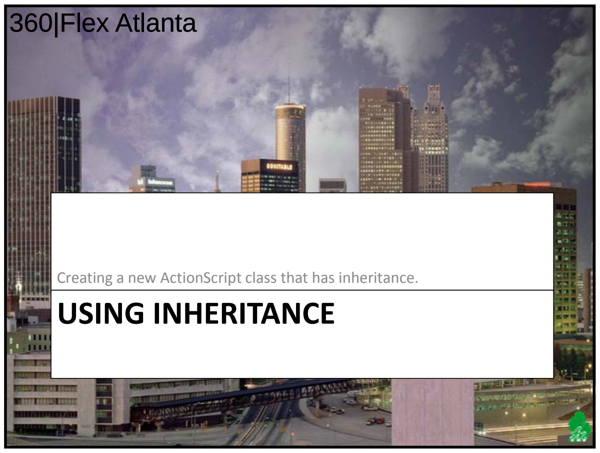


360|Flex Atlanta

Inheritance

- You can start with building the most fundamental properties and methods that every type of object below it would use.
- For example, all containers hold other objects, but only TitleWindows will have a close button.

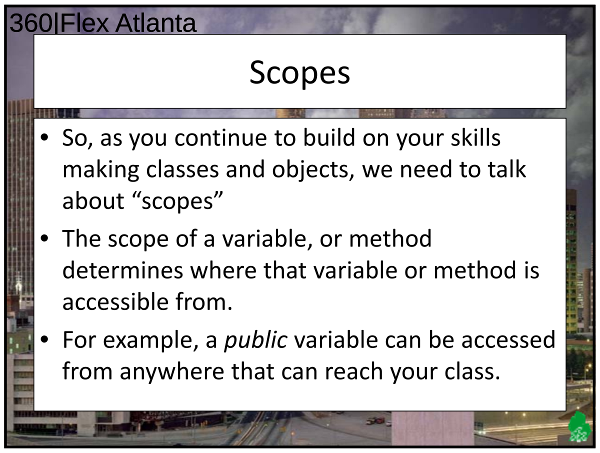




360|Flex Atlanta

Creating a new ActionScript class that has inheritance.

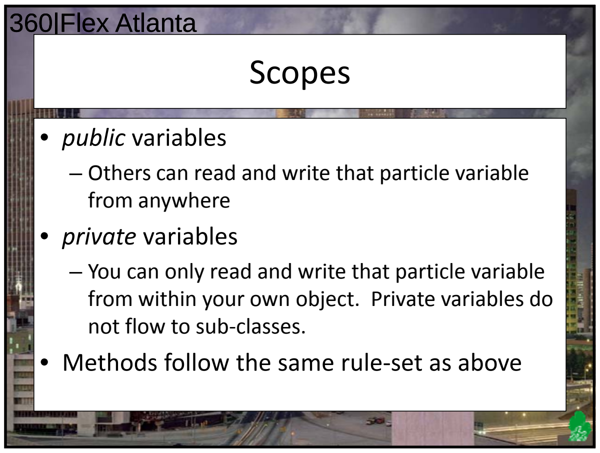
USING INHERITANCE



360|Flex Atlanta

Scopes

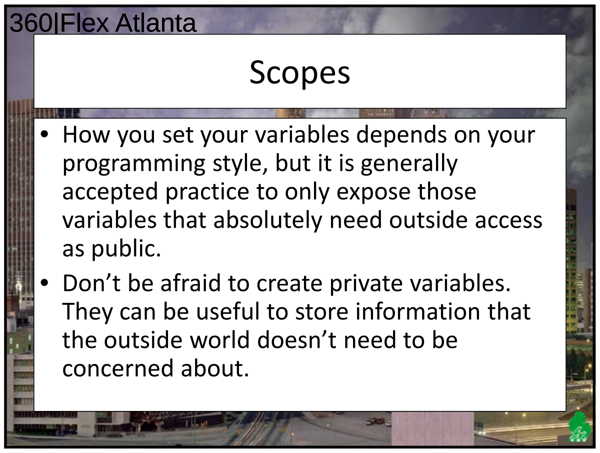
- So, as you continue to build on your skills making classes and objects, we need to talk about "scopes"
- The scope of a variable, or method determines where that variable or method is accessible from.
- For example, a *public* variable can be accessed from anywhere that can reach your class.



360|Flex Atlanta

Scopes

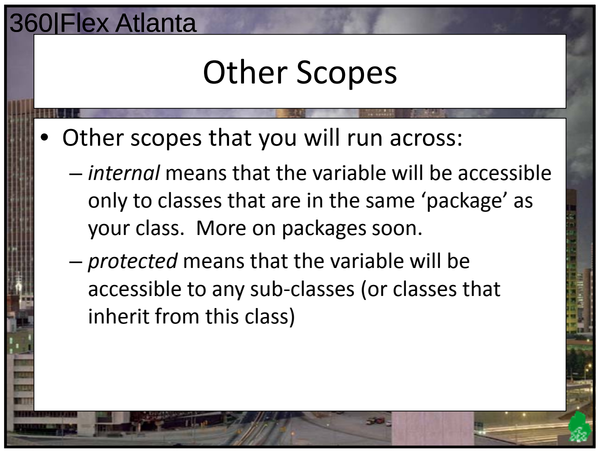
- *public* variables
  – Others can read and write that particle variable from anywhere
- *private* variables
  – You can only read and write that particle variable from within your own object. Private variables do not flow to sub-classes.
- Methods follow the same rule-set as above



360|Flex Atlanta

Scopes

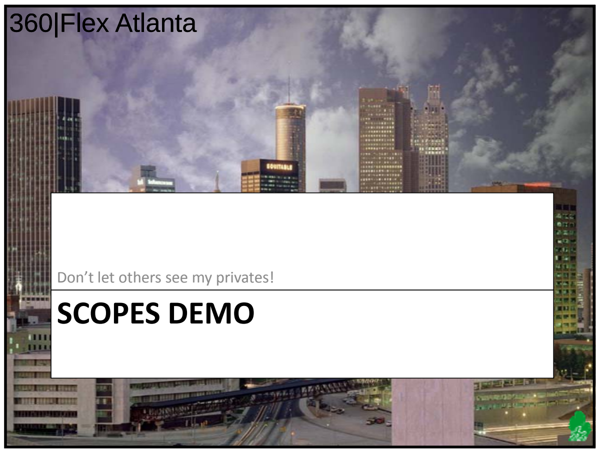
- How you set your variables depends on your programming style, but it is generally accepted practice to only expose those variables that absolutely need outside access as public.
- Don’t be afraid to create private variables. They can be useful to store information that the outside world doesn’t need to be concerned about.



360|Flex Atlanta

Other Scopes

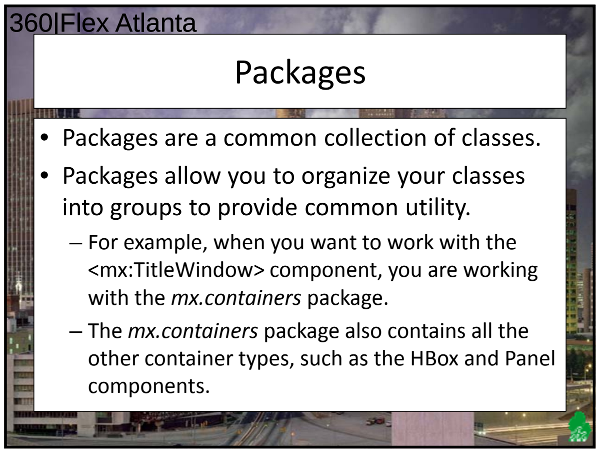
- Other scopes that you will run across:
  – *internal* means that the variable will be accessible only to classes that are in the same ‘package’ as your class. More on packages soon.
  – *protected* means that the variable will be accessible to any sub-classes (or classes that inherit from this class)



360|Flex Atlanta

Don't let others see my privates!

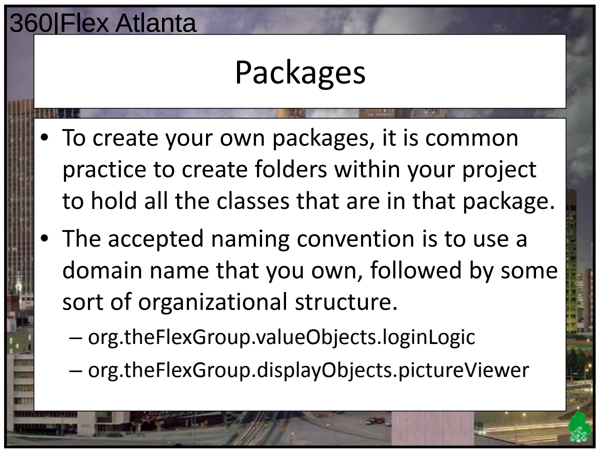
SCOPES DEMO



360|Flex Atlanta

Packages

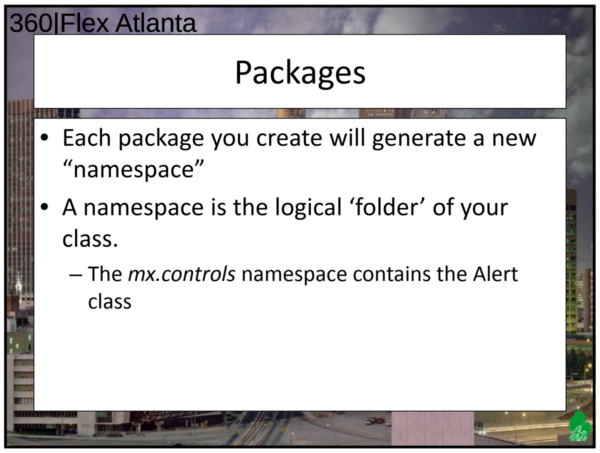
- Packages are a common collection of classes.
- Packages allow you to organize your classes into groups to provide common utility.
  - For example, when you want to work with the <mx:TitleWindow> component, you are working with the *mx.containers* package.
  - The *mx.containers* package also contains all the other container types, such as the HBox and Panel components.



360|Flex Atlanta

Packages

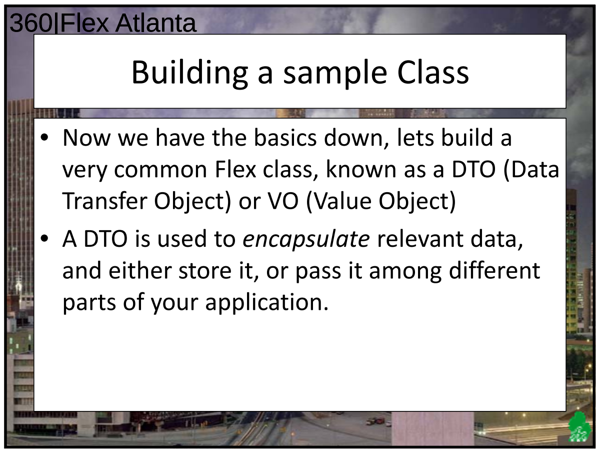
- To create your own packages, it is common practice to create folders within your project to hold all the classes that are in that package.
- The accepted naming convention is to use a domain name that you own, followed by some sort of organizational structure.
  - org.theFlexGroup.valueObjects.loginLogic
  - org.theFlexGroup.displayObjects.pictureViewer



360|Flex Atlanta

Packages

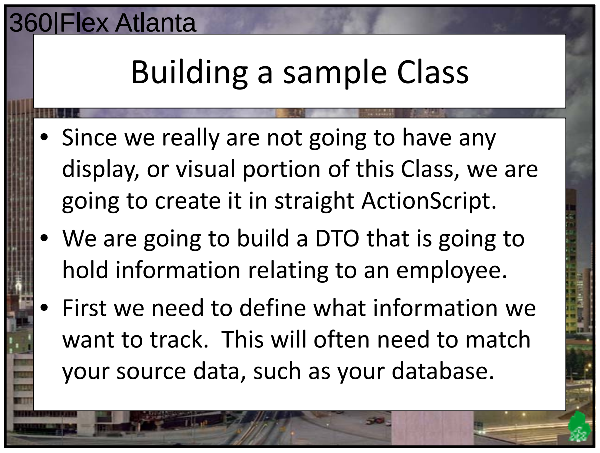
- Each package you create will generate a new "namespace"
- A namespace is the logical 'folder' of your class.
  - The *mx.controls* namespace contains the Alert class



360|Flex Atlanta

Building a sample Class

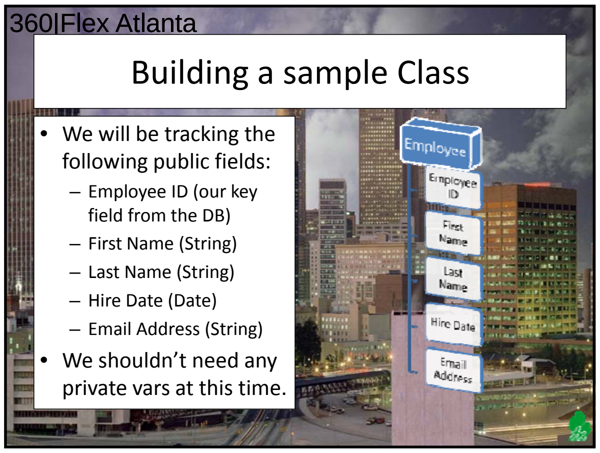
- Now we have the basics down, lets build a very common Flex class, known as a DTO (Data Transfer Object) or VO (Value Object)
- A DTO is used to *encapsulate* relevant data, and either store it, or pass it among different parts of your application.



360|Flex Atlanta

Building a sample Class

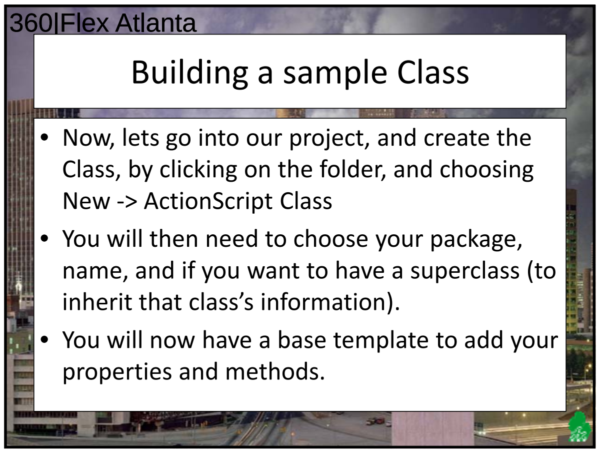
- Since we really are not going to have any display, or visual portion of this Class, we are going to create it in straight ActionScript.
- We are going to build a DTO that is going to hold information relating to an employee.
- First we need to define what information we want to track. This will often need to match your source data, such as your database.


Employee
Employee ID
First Name
Last Name
Hire Date
Email Address

360|Flex Atlanta

Building a sample Class

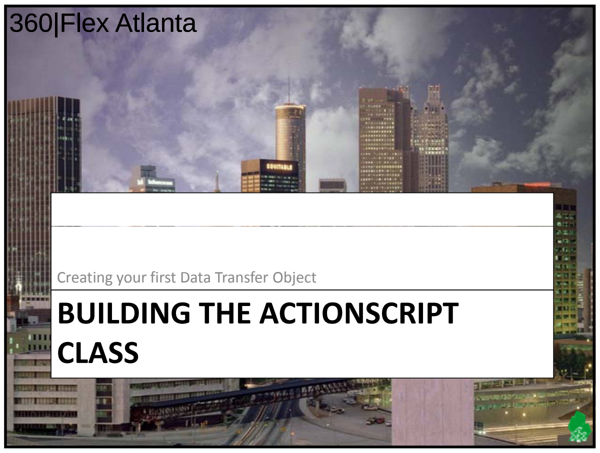
- We will be tracking the following public fields:
  - Employee ID (our key field from the DB)
  - First Name (String)
  - Last Name (String)
  - Hire Date (Date)
  - Email Address (String)
- We shouldn't need any private vars at this time.



360|Flex Atlanta

Building a sample Class

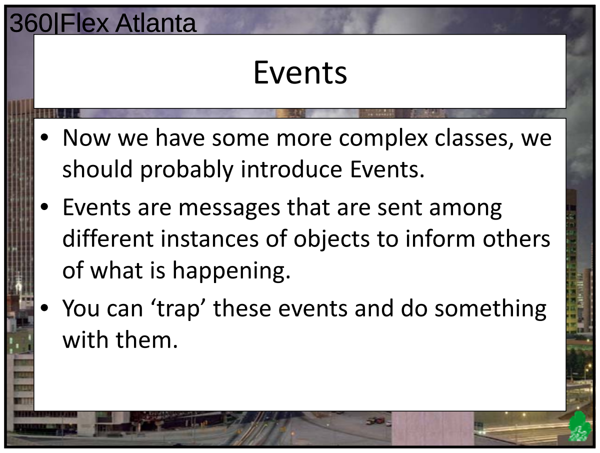
- Now, lets go into our project, and create the Class, by clicking on the folder, and choosing New -> ActionScript Class
- You will then need to choose your package, name, and if you want to have a superclass (to inherit that class's information).
- You will now have a base template to add your properties and methods.



360|Flex Atlanta

Creating your first Data Transfer Object

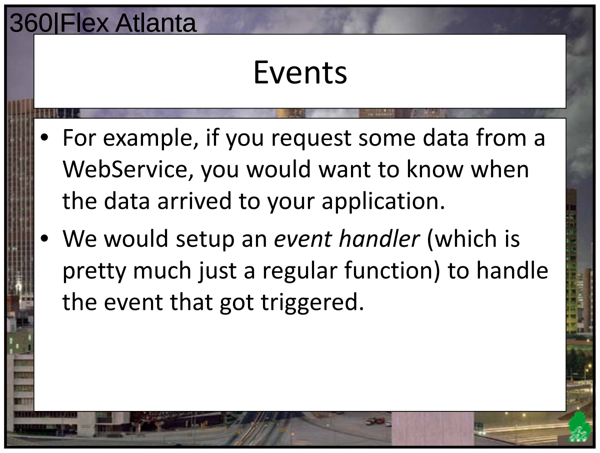
BUILDING THE ACTIONSCRIPT CLASS



360|Flex Atlanta

Events

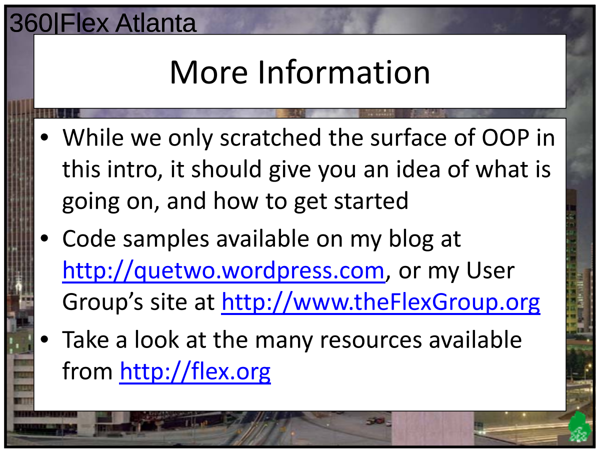
- Now we have some more complex classes, we should probably introduce Events.
- Events are messages that are sent among different instances of objects to inform others of what is happening.
- You can 'trap' these events and do something with them.



360|Flex Atlanta

Events

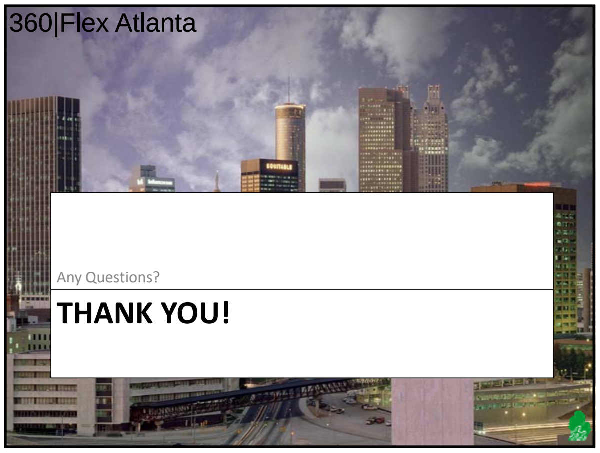
- For example, if you request some data from a WebService, you would want to know when the data arrived to your application.
- We would setup an *event handler* (which is pretty much just a regular function) to handle the event that got triggered.



360|Flex Atlanta

More Information

- While we only scratched the surface of OOP in this intro, it should give you an idea of what is going on, and how to get started
- Code samples available on my blog at http://quetwo.wordpress.com, or my User Group's site at http://www.theFlexGroup.org
- Take a look at the many resources available from http://flex.org



360|Flex Atlanta

Any Questions?

THANK YOU!