4
Adding a New Class

In this Chapter:

- ❑ **Defining a New Class**
- ❑ **Adding a New Class Attribute to the *app* Class**
- ❑ ***app* Class: Attribute Restrictions**

The `Die` Class

Introduction

To see how we can tackle our games programming using the class-oriented aspects of C++, we'll start by creating a new project which, as well as defining the `app` class, will also define a *Die* (dice) class which we can use to display and "roll" a six-sided die (see FIG-4.1).

FIG-4.1

A Visual Representation of a *Die* Class Object

Activity 4.1

Make a copy of the folder *CoreMethods* and rename the copy *UsingADie*.

Features

Properties of the `Die` Class

Since this is to be a visual component, we'll need to know the ID of the image and sprite used for the die as well as the last value that has been rolled.

When the die is created, we'll want to load, position and size the image it uses within a sprite. Positioning and sizing and even using a new image should also be available after the die has been created. And, of course, we want to be able to roll the die. This last operation will create both a visual effect and set the value thrown to a random integer between 1 and 6.

Class Diagrams

In its simplest form a class diagram is a visual representation of the attributes and operations of a class.

The following features can be added when describing the various elements of the class:

+	at the start of a property indicates that it is public.
-	at the start of a property indicates that it is private
= value	after a property shows that it is a constant attribute. The property name should be given in uppercase OR after a parameter, giving it's default value.
underlined	below a static property.

«constructor»name for constructors.

«destructor»name for destructors.

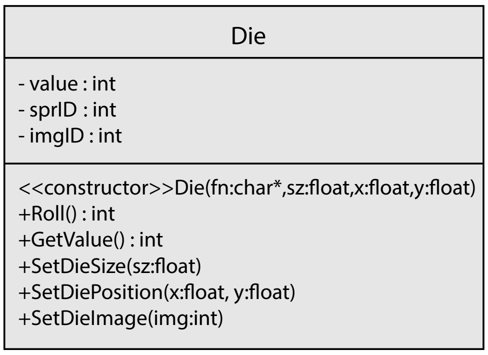
«friend»name for functions which are friends of the class.

When naming the constructors and the destructor in a class diagram, you can use the C++ convention of naming them after the class or you can use other terms such as *Create* and *Destroy* or *New* and *Delete*.

The class diagram for *Die* is shown in FIG-4.2.

FIG-4.2

The Class Diagram for Die



```
Die
- value : int
- sprID : int
- imgID : int
<<constructor>>Die(fn:char*,sz:float,x:float,y:float)
+Roll() : int
+GetValue() : int
+SetDieSize(sz:float)
+SetDiePosition(x:float, y:float)
+SetDieImage(img:int)
```

Implementing the Class

The *Die* Header File

To implement and test our new class, we'll start by creating a header file which contains the code shown in FIG-4.3.

FIG-4.3

The Header File for the Die Class

```
class Die
{
   private:
      int value;         //Value thrown
      int sprID;         //ID of sprite used to show dice;
      int imgID;         //ID of image used for dice;
   public:
      Die(char*, float, float, float);
      int Roll();
      int GetValue();
      void SetDieSize(float);
      void SetDiePosition(float, float);
      void SetDieImage(int);
};
```

While Visual Studio will automatically include the line

```
#pragma once
```
(code shown in FIG-4.3 will go here)

at the start of our new header file to ensure that the header file is copied no more than once into the final build before compilation, when using Android Studio creating the new header file will automatically insert the following lines to achieve the same safeguard:

```
#ifndef USINGADIE_DIE_H
#define USINGADIE_DIE_H
```

(code shown in FIG-4.3 will go here)

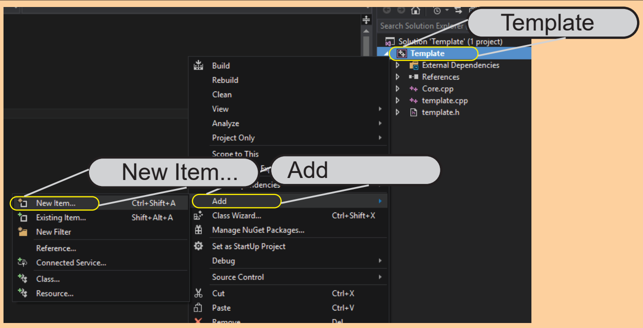
```
#endif //USINGADIE_DIE_H
```

The steps needed to add the new header file to our existing project using Visual Studio are shown in FIG-4.4. For Android Studio see FIG-4.5.

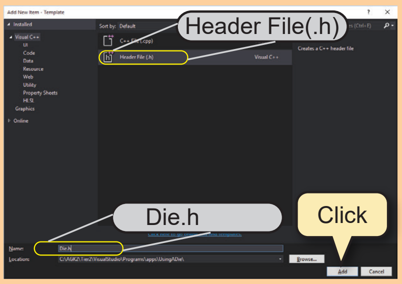
FIG-4.4

Creating the Header File for the *Die* Class in Visual Studio

With a new copy of *CoreMethods* renamed as *UsingADie*, we need to right click on the project name (*Template*), choose *Add* and *New Item*.



From the dialog box, choose *Header File (.h)* and name the file *Die.h* then click Add.



The new file will automatically open up in the edit window, where we can add the necessary code.

```
class Die
{
        private:
                int value;                      //Value thrown
                int sprID;                      //ID of sprite used to show dice;
                int imgID;                      //ID of image used for dice;
        public:
                Die(char*, float, float, float);
                int Roll();
                int GetValue();
                void SetDieSize(float);
                void SetDiePosition(float, float);
                void SetDieImage(int);
};
```

FIG-4.5

Creating the Header File for the *Die* Class in Android Studio

After making a copy of *CoreMethods* and renaming it *UsingADie*, we need to display the contents of *UsingADie\AGK2Template\src\main\jni*.
Right-clicking on *jni* displays a context menu. Selecting **New** and **C/C++ Header File** will create the empty file we require.

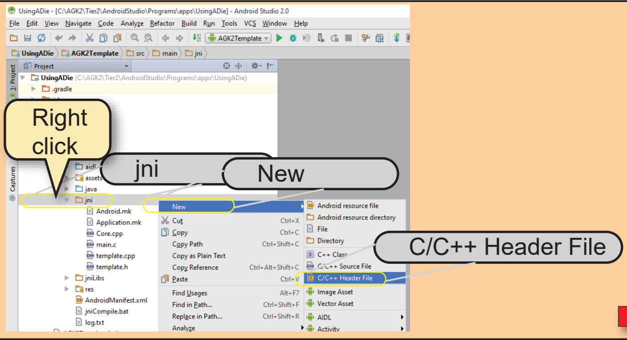
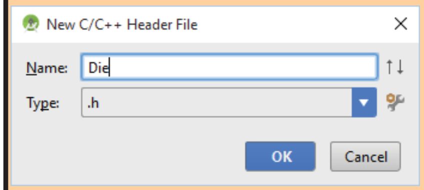
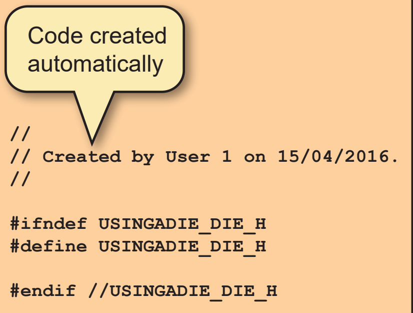


FIG-4.5
(continued)

Creating the Header File for the *Die* Class in Android Studio

In the dialog box, name the file *Die* (don't add the *.h* extension - Android Studio will do this automatically) then click **OK**.	The new file will be listed within the *jni* folder and opened in the edit window, with comments and the necessary guard code.





```
//
// Created by User 1 on 15/04/2016.
//

#ifndef USINGADIE_DIE_H
#define USINGADIE_DIE_H

#endif //USINGADIE_DIE_H
```

Activity 4.2

Load project *UsingADie* and create a new file called *Die.h*.

Enter the code given in FIG-4.3 in the new file.

Next, we need to create the source code for the new class's methods.

The constructor loads the image used by the die before creating, sizing and positioning the sprite used to display the die. It also "rolls" the die to assign it an initial value.

The *Roll()* method not only assigns the die a new random value between 1 and 6 but also creates a visual effect of the die being shaken before finally displaying the rolled value. This method also returns the value that has been generated.

The *GetValue()* method returns the value currently assigned to the die by the previous roll.

New AGK2 functions used in these routines are shown in FIG-4.6.

FIG-4.6 AGK2 Functions Used in *Die.cpp*

Function Name	Parameters	Returned	Description	Example
SetSpriteAnimation	int sprID int width int height int frames	None	Separates the sprites image into a set of frames. Frames are *width* by *height* pixels and the total number of frames is *frames*.	SetSpriteAnimation(1,20,40,3) Sprite 1's image is split into 3 frames each 20 by 40 pixels
GetImageWidth	int *ImgID*	int	Returns the width of an image in pixels.	int w = GetImageWidth(3) Returns the width of image 3
GetImageHeight	int *imgID*	int	Returns the height of an image in pixels.	int h = GetImageHeight(3) Returns the height of image 3
SetSpriteFrame	int *sprID* int *frame*	None	Sets the sprite's displayed frame to frame number *frame* (Frame numbers start at 1).	SetSpriteFrame(1,3) Sets sprite 1 to display frame 3
Random2	int *low* int *high*	int	Returns a random value lying between *low* and *high* (inclusive).	int num = Random2(1,6) num is set to a random value between 1 and 6