UNIVERSITY OF Hull

Properties

Rob Miles

Department of Computer Science

08120 Programming 2

Objects and Properties

- A program is made up of classes
- A class can contain data and behaviours
- The data is held in member variables
- The behaviours are provided by member methods
- When you design an application you decide what classes are required and the data and behaviours they have

Private and Public

- Data in a class is usually made `private`
- Methods are usually `public`
- You call methods to make changes to the data inside the class
- This means that all changes to data in the class is controlled
  - Programmers love having control……

Managing Class Data

- The `Account` class will have lots of data items that are stored about each customer
- Each of these items will map onto a data member of the class
- The programmer must then provide methods to get and set the values of the items

A Bank Example

- Some customers insist on having an old style chequebook
- There are three ways that a customer can get their chequebook printed:
  - Standard chequebook
  - Picture chequebook
  - Braille chequebook
- This information must be stored in the account

ChequeBookStyle Type

```
enum ChequeBookStyles
{
    standard,
    picture,
    braille
}
```

- We can create a type which represents the range of values that are needed
- The `ChequeBookStyles` type holds a value for each possible type of chequebook that can be ordered

The Power of Enum

```
enum AlienSpriteState
{
    asleep,
    hiding,
    hunting,
    damaged,
    dead
}
```

- We can use an enum anywhere we want to hold a value which can occupy a particular set of states
- They are particularly powerful when used with the switch construction

ChequeBookStyle Type

```
enum ChequeBookStyles
{
    standard,
    picture,
    braille
}
```

- In the case of the bank we can add as many enum values as the bank has different styles of chequebook
- We will add a variable of type `ChequeBookStyles` into the Account class to keep track of the style of chequebook used in that account

Chequebook Property

```
public class Account
{
    private ChequeBookStyles chequebook;
    /// rest of account here
}
```

- The member chequebook holds the style of chequebook for use on this account
- This is a `private` member of the `CurrentAccount` Class
- It is not directly visible to code running outside the `Account` class

Managing the Chequebook

```
public class Account
{
    private ChequeBookStyles chequebook;
    public void SetChequeBook (ChequeBookStyles inChequeBook)
    {
        chequebook = inChequeBook;
    }
    public ChequeBookStyles GetChequeBook()
    {
        return chequebook;
    }
}
```

- We can provide public get and set methods for this property

Managing the Chequebook

```
CurrentAccount a = new CurrentAccount();
a.SetChequeBook(ChequeBookStyles.standard);
```

- When we want to control the chequebook property we can call the `SetChequeBook` method
- We can use the `GetChequeBook` method to read the chequebook setting for this account

Get and Set Methods

- Creating `Get` and `Set` methods is a standard programming pattern
- Frequently they will perform validation of incoming data and reject out of range values
- I have not added any validation to the set method at the moment
  - Perhaps only customers with an overdraft greater than 1,000 are allowed picture `ChequeBookStyles`

Using Properties

- The C# language provides *properties* to make getting and setting data easier
- They do not make things possible that we couldn't do before
- They must make the programs easier to write
- Some programmers call this *"syntactic sugar"*

Chequebook Property

```
public class CurrentAccount : Account
{
    private ChequeBookStyles chequebookValue;
    public ChequeBookStyles ChequeBook
    {
        get
        {
            return chequebookValue;
        }
        set
        {
            chequebookValue = value;
        }
    }
}
```

Property Keywords

- The block of code after the get keyword is obeyed when the property is read by a program
  - It must return a value of the property type
- The block of code after the set keyword is obeyed when the property is written
  - Within the set block the keyword `value` means "The value being assigned to the property"

Using the ChequeBook Property

```
CurrentAccount a = new CurrentAccount();
a.ChequeBook = ChequeBookStyles.picture;
```

- When a program uses a property it looks just like direct access to a data member
- The difference is that the get or set behaviours are used the blocks of code in the property are used
- This makes the code much tidier

Restricted properties

- By leaving out the set part of a property you can create ones which are “read only”
- This can be very useful
  - We could have a `Balance` property in the Bank application to read the value of the balance
- You can also create “write only” properties, but these are less common

Design with properties

- You can have multiple “read only” properties which provide different versions of the data inside
  - Provide the height in metres or feet, depending on the property used
- Each different get method can perform calculation to generate results
- You can also make use of multiple set properties if you want several ways to put a value into an object

Properties and the C# Libraries

- You can see properties in use in some of the C# library methods:

```
int [] scores = new int [20];
Console.WriteLine ( scores.Length);
```

- You get the number of elements in an array by using the `Length` property of it
- This is read only, you can't change the length of an array by assigning to this property

Property Issues

- Properties can make code clearer but use them with care
- They do not have a way of indicating that a set action has failed
- When a property is used it raises the possibility of lots of code running as a result of what looks like a simple assignment or read
- The Refactor commands in Visual Studio can convert member variables into properties for you by creating the get and set behaviours