UNIVERSITY OF Hull

Object Ettiquette

Rob Miles

Department of Computer Science

Objects and Programs

- We are now using objects in our programs to represent items
  - The object contains data it manages
  - The object provides behaviours we can use
- We are creating our `Account` class on this basis
- Now we need to consider some other things that we can do to make our objects better

Objects and Strings

```
int i = 99;
Console.WriteLine (i);
```

- We are very used to the idea that when we want to print out a value we can just do this
- However, it probably shouldn't work:
  - `WriteLine` wants to print a string, and `i` is an integer

The Magic of ToString

- We have seen that to get from a string to a number we have to use `Parse`
- But to get from a number to a string seems to happen automatically
- This is because all the number classes provide a "`ToString`" method which returns a string which describes them

Accounts and ToString

```
int i = 99;
Console.WriteLine (i);
```

- When the system needs the string version of an instance it calls the `ToString` method on that instance
- This happens automatically
- All the number classes have this behaviour built in

Accounts and ToString

```
Account a = new Account("Rob", "Hull", 100, 1);
Console.WriteLine (a);
```

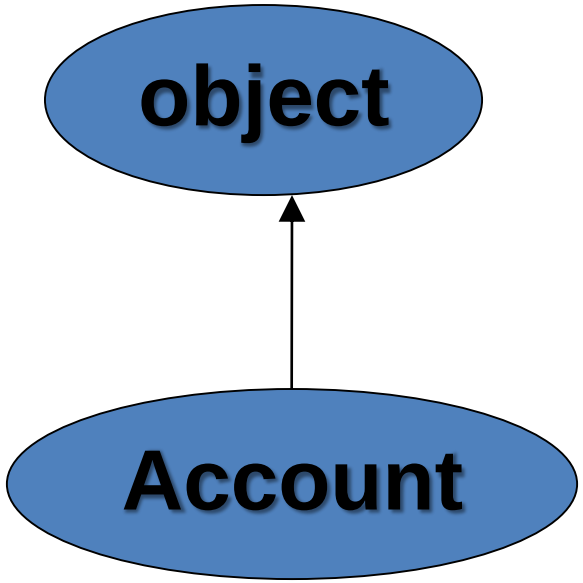
- When an `Account` instance is printed it doesn't have a useful `ToString` behaviour
- Instead it prints out the name of the class
- What we want to do is print out the account information

Making our own ToString

- If you don't provide a `ToString` method you get the one provided by the parent class
- This just returns the fully qualified name of the class
- We want to create our own `ToString` method that returns account information
- To do this we must *override* the method in our parent class

Class Hierarchies

- When you create a new class it is actually based on a *parent* class
- The `Account` class is based on the `object` class
- It is called the *child* of object
- An `Account` instance can do everything an `object` instance can do



What is an Object?

```
object o = new object();
Console.WriteLine (o);
```

- The `object` class is built into C#
- You can create instances of it if you like
- You can't use it for much, but it does provide the basis of all the classes that you create
  - When you declare a new class you are actually extending the `object` class
- We will discuss extending classes later in the course

Overriding ToString

```
public override string ToString()
{
  return "Account: " + accountNumber +
         " Name: " + name +
         " Address: " + address +
         " Balance: " + balance;
}
```

- This version of `ToString` returns a string that describes the content of an `Account`
- It *overrides* the `ToString` method in `object`

Overriding

- Overriding means that rather than using the method in the parent class, the method in the child is called instead
  - The child class can have behaviour which is appropriate to that particular class – this is more useful than the parent behaviour
- The keyword `override` is used to tell the compiler that the method is overriding one in the parent
- Note that this is quite different from overloading a method

Override and Overload

- Override:
  - Provide a method in a child class with the same name and same signature as one in the parent
  - This method is used instead of the one in the parent
  - It *overrides* it
- Overload:
  - Provide a method in a class with the same name but different signature as others in that class

Overriding and Class Design

- We will take a look at overriding in more detail later, when we consider how to design systems using class hierarchies
- For now you should know that when you create a class it is considered good manners to create a `ToString` method
- Then it can be printed out if required

The Equals Method

- The `object` class also has an `Equals` method which can be used to compare two objects to see if they contain the same values
- If we wanted to allow users of the `Account` class to compare two accounts and see if they contained the same data we could add our own `Equals` method to do this
- The equals behaviour is used a lot in testing of our programs
- It is how we can prove that our load/save methods are working correctly

Using the Equals method

```
Account a = new Account ("Adam", 0);
Addount b = new Account ("Adam", 0);
if (a.Equals(b))
{
    Console.WriteLine("The same");
}
```

- The `Equals` method is used to compare two objects to see if they contain the same data
- It is called on one instance and passed a reference to the other

Writing our own Equals

- It would be useful to have our own `Equals` method for the `Account` class
- Then we can test our program can save `Account` values and retrieve them intact
- To do this we must override the `Equals` method in the parent `object` class

An Equals Method for Account

```
public override bool Equals(object obj)
{
    Account compareWith = (Account) obj;
    if (name != compareWith.name)
    {
        return false;
    }
    if (address != compareWith.address)
    {
        return false;
    }
    return true;
}
```

Casting References

- The `Equals` method is always given a reference to an object
- The `Equals` method must cast this to a reference to an `Account`
- Then we can get hold of members of the account and use them to compare with the ones in the current instance

Etiquette Summary

- All classes are children of the `object` class
- The `object` class provides a `ToString` behaviour we can override
- This allows us to get text descriptions of the content of our classes
- We can also override the `Equals` method in the `object` class to allow instances to be compared