UNIVERSITY OF Hull

Objects and Structures

Rob Miles

Department of Computer Science

28b 08120 Programming 2

The "Friendly Bank"

- The examples we are going to use are for the creation of a management system for "The Friendly Bank"
- The manager has asked us to create a C# program that will keep track of information held about each account holder
- The bank has told you what information needs to be stored
- This is metadata about the system to be created
- For the purpose of demonstration we are going to work with the name, account number and balance information for each customer

Objects vs Structures

- It is very important that you appreciate the difference between objects and structures
- Objects are managed by *reference*
  - A variable is a tag which is connected to the instance which is held somewhere in memory
- Structures are managed by *value*
  - A variable is a box which holds the actual data for the structure

The Account Structure

```
struct AccountStruct
{
    public string Name;
    public int AccountNumber;
    public int Balance;
}
```

- We have seen structures before
  - They allow us to create “lumps of data”
- If all we need to store is the name of the account holder, their account number and the amount of money they have we can create a structure like the one above

Using a struct

- You create a variable of type `struct` by declaring it

```
AccountStruct rob;
rob.Name = "Rob Miles";
```

- This creates an account structure which can hold an account value
- This structure is managed by value, in that once we have declared the variable we get space in memory where we can store a value of type `AccountStruct`

Creating a Bank

```
AccountStruct [] Bank = new AccountStruct [100];
```

- Once we have our structure type we can create an array of that type
- This will reserve space in the memory to store a large number of accounts
- The memory will be reserved as a single large block of memory with enough room to hold 100 account items

The Account Class

```
class AccountClass
{
    public string Name;
    public int AccountNumber;
    public int Balance;
}
```

- Structures and classes both describe objects
- The code above creates a class called `AccountClass` that has exactly the same content as `AccountStruct`
- However, because it is a class it is managed by reference

Using a class variable

- Making a class variable is more complicated than a structure

```
AccountClass rob;
rob.Name = "Rob Miles";
```

- When a variable of `AccountClass` type is declared you only get a reference to objects of that type
  - This is because classes describe objects that are managed by reference
- If I try to use this reference the program will fail at run time

Creating an Account Reference

```
Account RobsAccount;
```

- Declaring a variable of type `Account` creates a tag that can refer to `Account` instances
- It does **not** create anything that can store account information

Creating an Instance

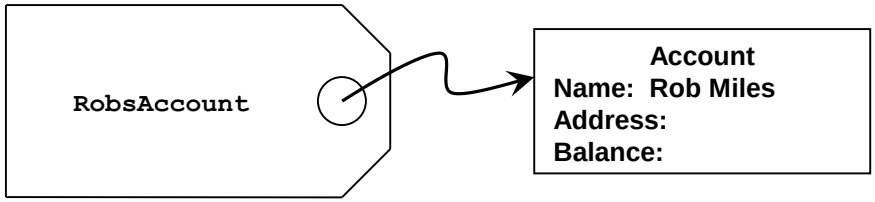
- Before you can use a class variable you need to create an instance

```
AccountClass rob;
rob = new AccountClass();
rob.Name = "Rob Miles";
```

- We have seen the new construction before
- We used it to create new instances of arrays
- This means that an array is actually implemented by a class

Creating an Account Instance

```
Account RobsAccount;
RobsAccount = new Account();
RobsAccount.Name = "Rob Miles";
```



- If we want an `Account` instance we have to create it and set the reference to refer to it
- Note that this is different from structures (and other types managed by value)

Assignment

- The effect of assignment is different when considering value and reference types
  - A value type assignment copies the values from one struct to another
  - A reference type assignment makes the two references point at the same instance
- It is impossible to state the effect of an assignment without knowing whether the variables are reference or value

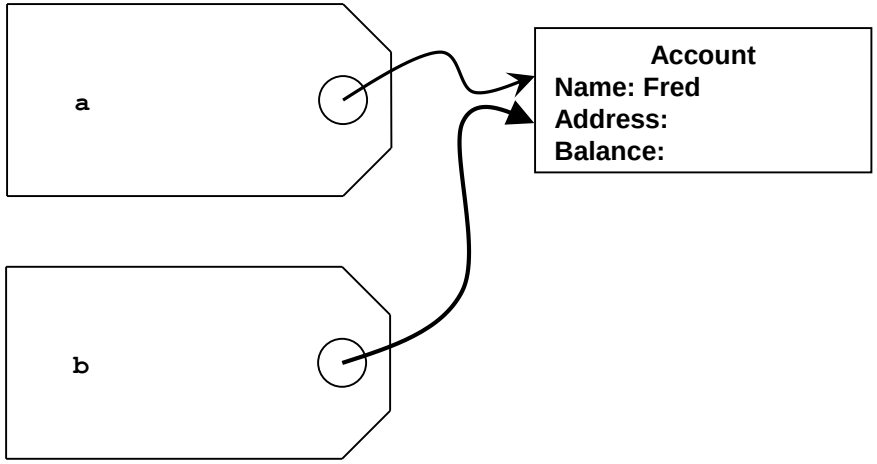
Assignment Problems

```
a = b;
a.Name = "Fred";
```

- If `a` and `b` are value types the assignment will have no effect on the value held in `a`
- If `a` and `b` are reference types the assignment will make them both refer to the same instance
- This means that there are two ways of accessing one object

References to the same object

```
a = b;
a.Name = "Fred";
```



- An object in memory can have multiple references referring to it
- In the above code we have changed the `Name` property of the object referred to by `a`
- We could have changed the name of `b`

Garbage Collection

```
AccountClass a = new AccountClass();
AccountClass b = new AccountClass();
a = b;
```

- At the end of the above sequence both **a** and **b** refer to the same instance
- The original instance referred to by **a** now has no references connected to it
- The memory it occupies will be recovered by the Garbage Collector

Why Bother with References?

- References don't seem to add much value to our programs
  - You have to use the new keyword to create them
  - You get this strange behaviour when you perform assignments
  - You can waste blocks of memory which need to be garbage collected

Useful References

- We can "move" data around by simply moving the reference
- We can have arrays of references which can be ordered in different ways
  - A bank account list ordered on customer name
  - A bank account list ordered on account balance
- We can also use references to build up data structures
- We will see how to do this in the next session