UNIVERSITY OF Hull

Creating a Bank Class

Rob Miles

Department of Computer Science

Containers

- Often in programs you want to store more than one item
  - Cricket scores
  - Bank Accounts
- We have seen that we can use arrays to hold multiple objects
- An array is a container object
  - It contains a number of elements
- A Windows Presentation Foundation (WPF) page is a container object
  - It contains a number of display elements

An Account Container

- Accounts are held in a `Bank`
- We need to create a `Bank` class that can manage a number of `Account` objects
- This will be another of our "business classes" which will have the job of looking after the data that our bank business uses
- We will of course create tests for it

Who Does What?

- Up until now, when we needed an `Account` we made one
- This is not what would really happen
- The `Bank` is actually the only thing that can make `Account` instances
  - It must create a unique account number for each `Account`
  - It must store accounts and allow programs to get hold of them

Account Numbers

- We can't use the name of a customer to identify a particular bank account
  - This is because there are lots of people with the same name
- Instead we have to give each account an unique number to identify it
- The Bank must create these account numbers
  - It knows which numbers have already been used

The Bank as an object

- We have seen that when we made an `Account` we decided what it needs to store and the methods it must provide to those who wish to use it
- Now we are doing exactly the same with the `Bank`
- We will also have to do things like create constructors for the `Bank` class

Bank Properties and Behaviours

- The `Bank` will have a name property
- It will also contain a set of methods to provide the behaviours
- We need to be able to use the `Bank` to:
  - Get the name of the bank
  - Create and store a new `Account`
  - Find an account by account number
  - Delete an account we don't want any more

Bank Abilities as C# Methods

```
Account AddAccount(string accountName,
        string accountAddress,
        decimal initialBalance);

bool DeleteAccount(int accountNumber);

Account FindAccount(int accountNumber);

string GetBankName();
```

- These are all the things a `Bank` needs to do
- Anything that can do this things is a `Bank`
- As users of the `Bank` we don't need to care how it works

Constructing a Bank

- When we construct a bank we must give the bank a name
- We might also need to tell the bank the maximum number of Accounts that the bank will ever hold
- Then the `Bank` can reserve space for these
- The `Bank` could contain an array of `Account` references, with an element to hold each of the accounts in it

Storing Account References

- We know that classes are managed by *reference*
- We know that `Account` is a class
- Therefore the `bankAccounts` list just holds references to `Account` instances
- When the bank "gives" another part of the program an `Account` to work on it really gives it a reference to that `Account`

Constructing a Bank

```
Bank friendlyBank = new Bank ("Friendly Bank);
```

- This constructor creates a new `Bank` instance
- The bank name is set to "Friendly Bank"
- Note that we don't know how the bank actually works, we just know how to use it
- This is a fundamental principle of object oriented programming

The Bank Class

```
class Bank
{
    private string bankName;
    private List<Account> bankAccounts;
    public Bank(string newBankName)
    {
        bankName = newBankName;
        bankAccounts = new List<Account>();
    }
}
```

- This constructor sets the name of the bank and creates a list to hold references to the `Account` objects in the bank
- Note that the bank is the **only** way we can get hold of accounts

Creating a Bank

```
Bank friendlyBank = new Bank("The Friendly Bank");
```

- This is how we would create a new bank
- The bank is given a name
- The reference `friendlyBank` refers to the bank object
- We would normally only create the bank once, after that we would load it from a saved file
- We will look at how to store the bank a little later on

AddAccount method

```
Account rob = friendlyBank.AddAccount("Rob", "Hull", 100);
```

- This method is used to add an account to the bank
- It returns a reference to the `Account` it adds to the bank
- If it fails it an exception will be thrown by the `AddAccount` method
- This would indicate that some of the initial Account settings were invalid
  - Empty account name or address for example

Account Number Question

- Why don't we set the account number when we create a new `Account`?

Account Number Question

- Why don't we set the account number when we create a new `Account`?
- This is because we never set the account number
- It is set by the bank
- We can read back the account number from an `Account` that has been created, but we can't set the number itself

How AddAccount works

```
static int newAccountNumber = 1;
public Account AddAccount(string inName, string inAddress,
                          decimal inBalance)
{
    Account result = new Account(inName, inAddress, inBalance,
                                 newAccountNumber);
    bankAccounts.Add(result);
    newAccountNumber = newAccountNumber + 1;
    return result;
}
```

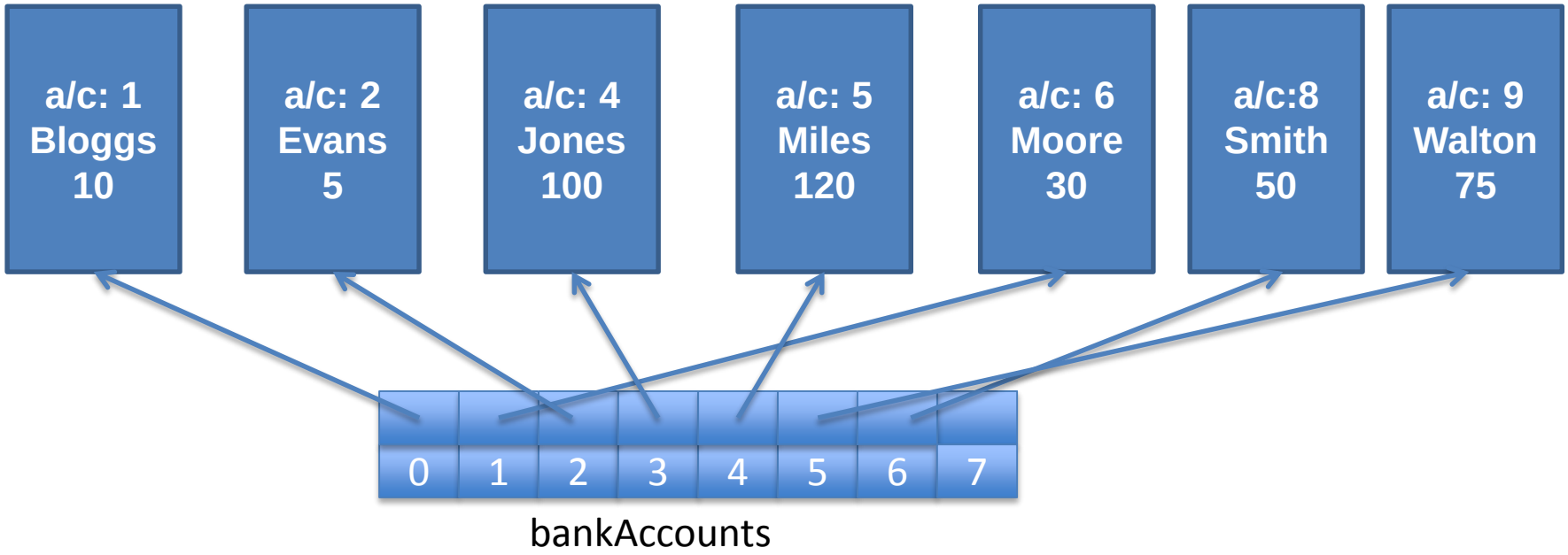
- `AddAccount` creates a new account and then adds it to the list of the accounts held in the bank

How AddAccount works

```
static int newAccountNumber = 1;
public Account AddAccount(string inName, string inAddress,
                          decimal inBalance)
{
    Account result = new Account(inName, inAddress, inBalance,
                                 newAccountNumber);
    bankAccounts.Add(result);
    newAccountNumber = newAccountNumber + 1;
    return result;
}
```

- The `newAccountNumber` variable is a static class member that is increased each time a new account is created
- Each account gets an account number one bigger than the previous one

The Bank Account Storage



- The bank storage is actually an list of references to `Account` objects
- Each element of the `bankAccounts` list refers to a particular `Account` object

Finding Accounts

- When a customer wants to perform some transactions the bank must find their account details
- This is exactly what happens when you put your card into a cash dispenser
- Your account information is used to obtain your bank details
- The bank class will require a behaviour that will allow customers to be located
- This could be called the `FindAccount` method

Using the FindAccount method

```
Account a = friendlyBank.FindAccount(1);
if (a == null)
{
    Console.WriteLine ("Account not found");
}
```

- This method searches the bank storage for an account with a particular account number
- If the account is not found the method will return `null`

FindAccount method

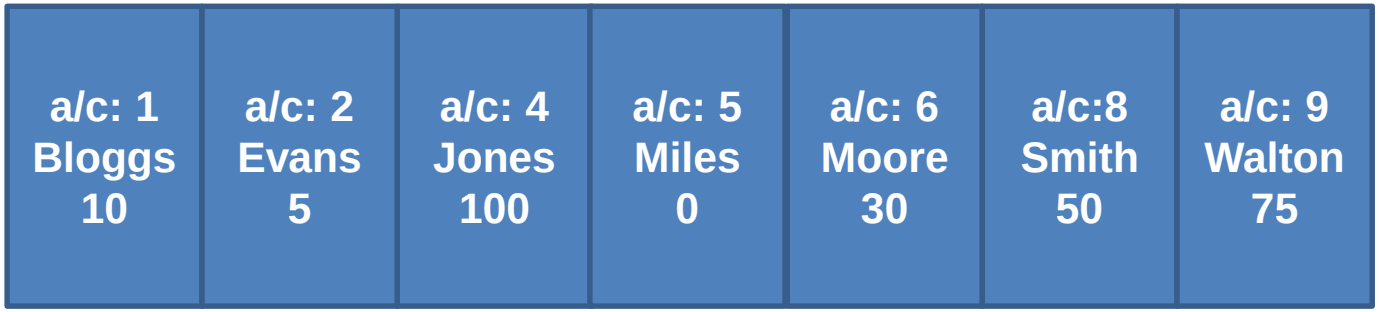
```
public Account FindAccount(int searchNumber)
{
    foreach (Account a in bankAccounts)
    {
        if (a.AccountNumber == searchNumber)
            return a;
    }
    return null;
}
```

- This is the inside of the `FindAccount` method
- It looks through the list of accounts until it finds one with the matching account number
- It then returns the reference in this element

Working on a Bank Account

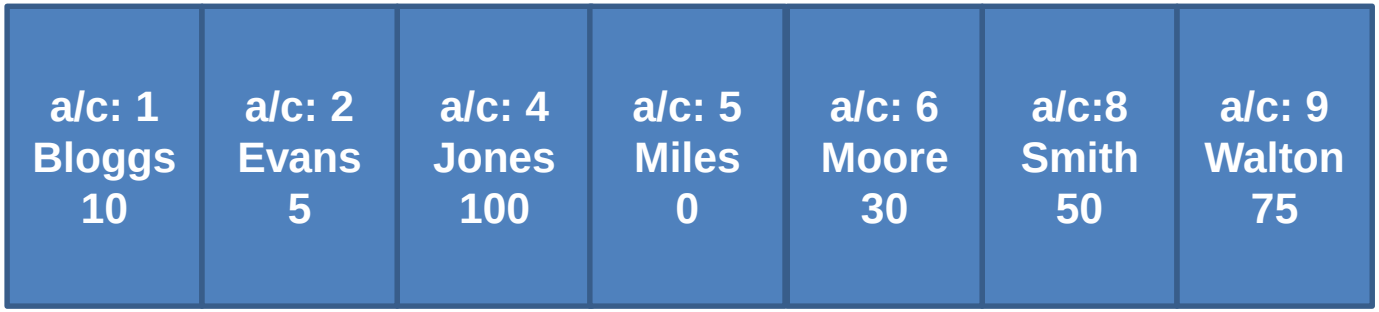
- Note that there is no method provided by the bank to "put an account back" when we have finished working with it
- This is because we never actually take the account out of the bank storage
- Instead we have a reference to the account that we are working on
- Any changes to the account will directly change the one in the bank
- This is the way that references work

An Example Bank



- This sample bank contains a number of accounts
- Each account has an account number, name and balance

Working with the Bank

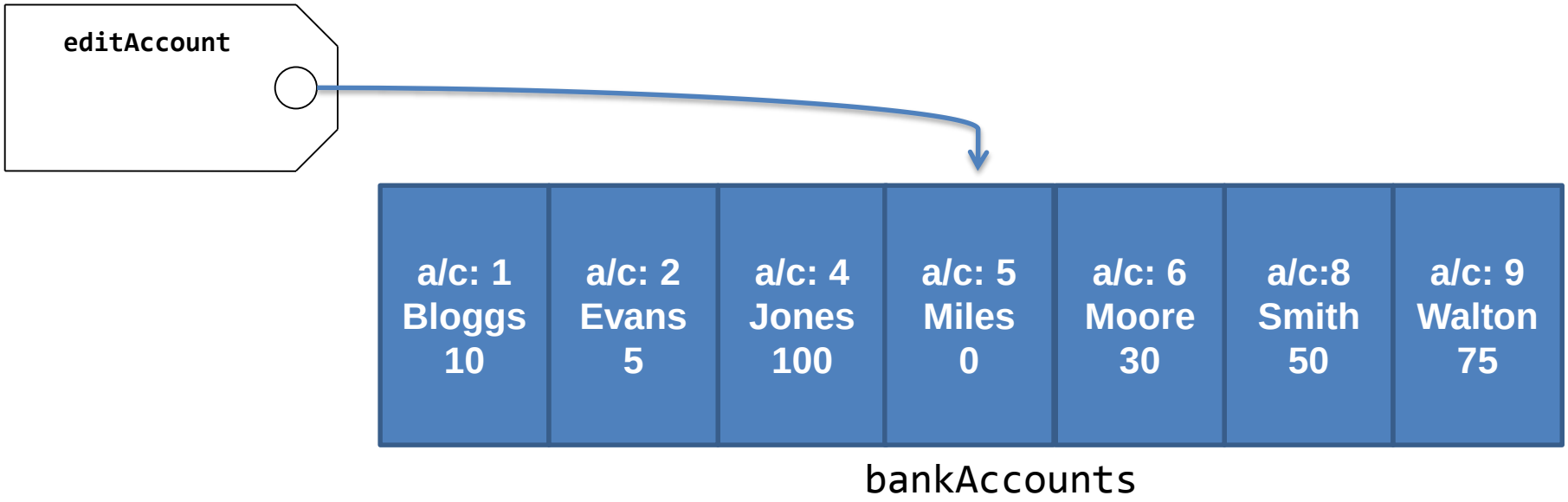


bankAccounts

```
Account editAccount;
```

- The `editAccount` variable is a reference that can refer to `Account` instances

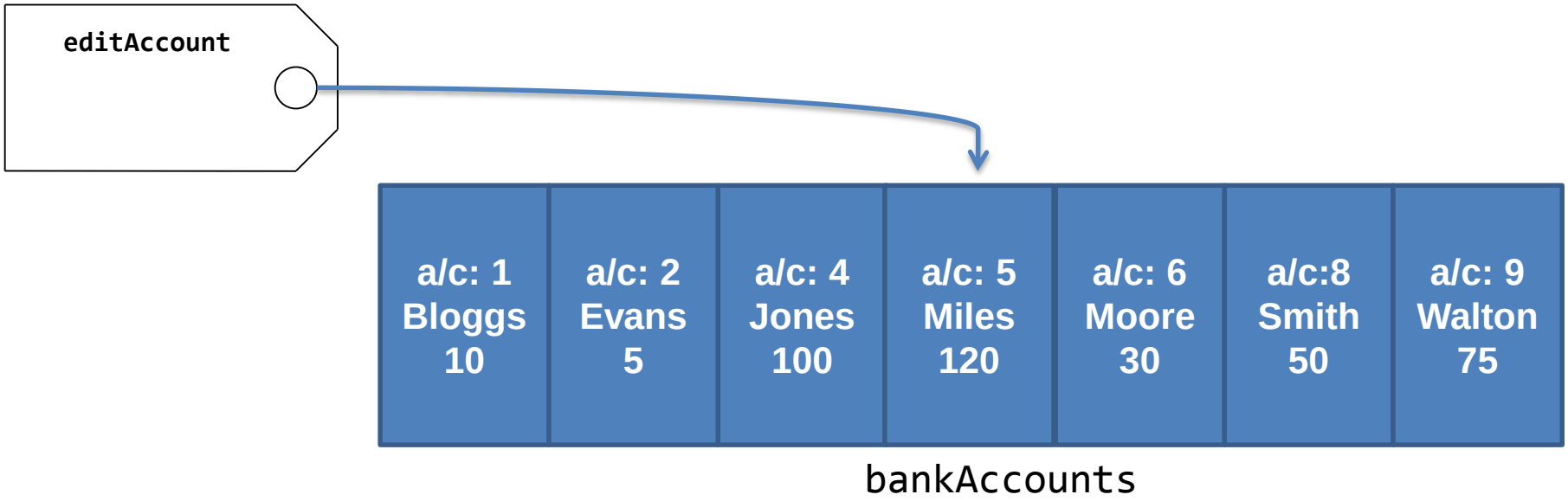
Working with the Bank




```
editAccount = BankStorage.FindAccount (5);
```

- The `FindAccount` method will ask the bank to return a reference to the account with the given account number

Working with the Bank




```
editAccount.PayInFunds (120);
```

- Changes to the account referred to by `editAccount` will directly affect that account in the bank

Using the DeleteAccount method

```
bool result = friendlyBank.DeleteAccount(1);
if (!result)
{
    Console.WriteLine ("Account not deleted");
}
```

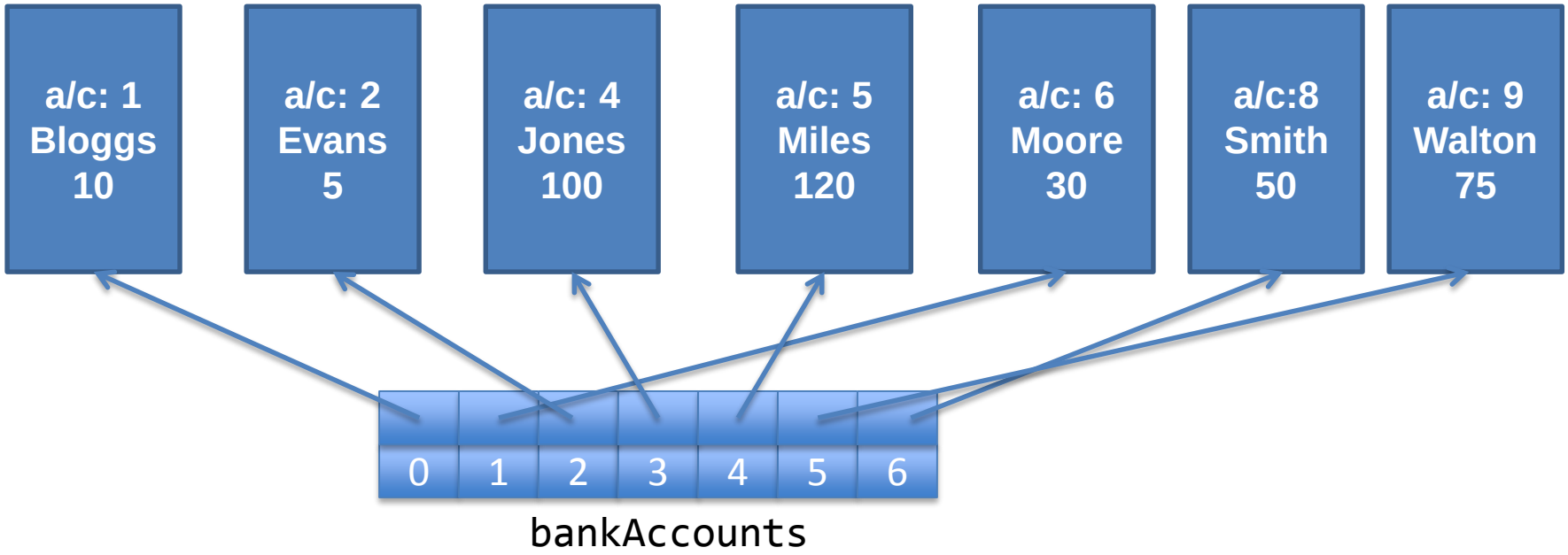
- This method is used to delete an account with a particular account number
- If the account is successfully deleted the method returns true to indicate that it has worked
- It is important that methods that do things return whether they worked or not

The DeleteAccount method

```
public bool DeleteAccount(int deleteNumber)
{
    Account del = FindAccount(deleteNumber);
    if (del != null)
    {
        bankAccounts.Remove(del);
        return true;
    }
    return false;
}
```

- To remove an account from the bank we just need to remove it from the account list
- It will then play no further part in our program and will eventually be removed by the Garbage Collector

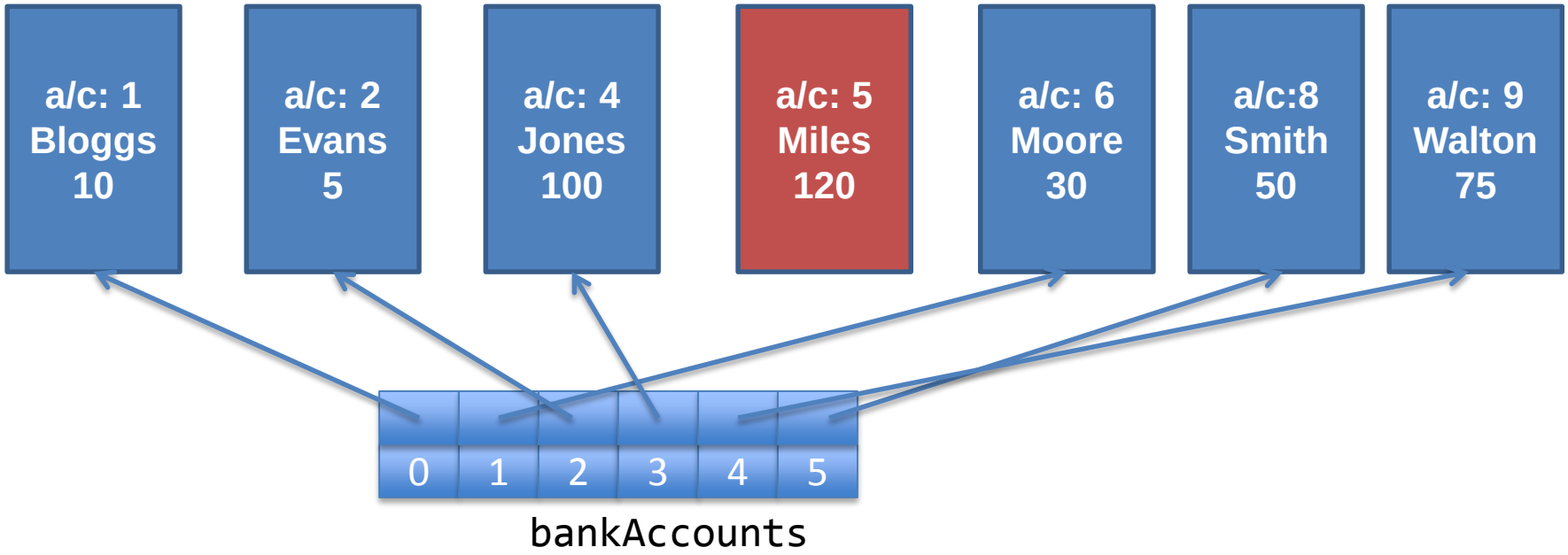
Deleting an Account using References




```
friendlyBank.DeleteAccount("5");
```

- The `DeleteAccount` method will search for the element that refers to the account with account number 5

Deleting an Account using References




```
friendlyBank.DeleteAccount(5);
```

- When the reference becomes `null` the account is no longer in the bank, as there are no references to it

BankName property

```
Console.WriteLine (friendlyBank. BankName);
```

- This property is used to obtain the name of the bank
- It is returned as a string which can then be printed
- Note that there is no property to set the name of a bank
- Once the bank has been created the name cannot be changed
- The bank name is a *read only* property

Building Banks and Other Things

- A bank is a good place to explore how to create and manage large amounts of related data
- It is very easy to understand what needs to be done, as we all have bank accounts
- The structure and organisation techniques behind the management of the bank data can also be used in lots of other contexts