

UNIVERSITY OF Hull

Generics, Lists and Dictionaries

Rob Miles

Department of Computer Science

Storing Collections

- You often want to store a number of things in your program
- We could do this by using an array

```
// Storing bank accounts for 10 people
Account [] accounts = new Account [10];
accounts [0] = new Account ("Rob");
```

- Arrays are useful but they have limitations
 - Fixed size means that if we need to store 11 Accounts we have to rebuild the program

The ArrayList as an improvement

- The first improvement on the array is the ArrayList
- This holds a list of references to objects
- The great advantage of the ArrayList is that it grows and contracts automatically
- The ArrayList provides methods you can use to remove elements as well as add them
 - You can add and remove items from the list as you need
 - There is no limit to the upper size of the list

Creating an ArrayList

```
// Storing bank accounts for any number of people
ArrayList accounts = new ArrayList ();
accounts.Add(new Account ("Fred"));
accounts.Add(new Account ("Jim")); // two elements
```

- This creates an ArrayList called accounts and then adds two Account instances to it
- The Add method is given a reference to the thing to be stored in the ArrayList
- Each time it is called the ArrayList gets one larger
- You don't need to know how this works

4

Reading ArrayList Elements

```
for (int i = 0; i < accounts.Count; i++)
{
    Account a = (Account)accounts[i];
    Console.WriteLine(a.GetName());
}
```

- The program can use subscripts to get the values out of an ArrayList
- The ArrayList also has a property called Count which gives the number of items in the list
- However you have to cast the value you get out of the ArrayList

5

ArrayList Elements as object references

```
for (int i = 0; i < accounts.Count; i++)
{
    Account a = (Account)accounts[i];
    Console.WriteLine(a.GetName());
}
```

- Each item in the ArrayList is managed by reference
- An ArrayList may be used to hold references to any kind of object
 - the references in the list must refer to object instances (which are the parent class of every type)

6

Casting ArrayList Element References

```
for (int i = 0; i < accounts.Count; i++)
{
    Account a = (Account)accounts[i];
    Console.WriteLine(a.GetName());
}
```

- If we want to use an item in a list as an Account we have to cast the object reference into an Account reference
- This is so that we can do “account” type things with the object on the end of the reference
- Of course, if the object on the reference isn’t an Account instance our program will fail at run time

7

Generics Lists and Dictionaries

Using foreach

```
foreach (Account a in accounts)
{
    a.Save(textOut);
}
```

- The `foreach` construction makes it much easier to work through any collection (including arrays)
- It automatically casts the elements to the type that it has been told to work with
- This construction works with any collection, including arrays

8

Generics Lists and Dictionaries

Enter Generics

- Later versions of C# have been extended to include support for *generics*
- Generics are another *abstraction* technology
 - They provide another way of ‘stepping back’
- In this case they let you design code that has general behaviours which are applied to a type the programmer specifies
- They are very useful for managing collections

9

Generics Lists and Dictionaries

Generic Theory

- The things that an array of integers does are just the same as an array of floats
 - It is just that the elements of each array are different types
- Generics let a programmer separate the actions (putting things in and out of elements) from the types they work on (`int`, `float`, `string`, `Account`, `Banjo`)
- We can separate the design and behaviours of the container from the type of objects that it contains
- A container that uses generics can work with references of any type

10

Generics Lists and Dictionaries

Storing Accounts with Generics

```
List<Account> accountList = new List<Account>();
accountList.Add(new Account ("Fred"));
accountList.Add(new Account ("Jim")); // two elements
```

- `accountList` is a `List` of `Account` references
 - You can use it in just the same way as an `ArrayList`
- The `List` collection class is supplied with the type of list you want using the notation shown above
 - This is given between the `<` and `>` characters
- You can make a `List` that holds any type

11

Generics Lists and Dictionaries

Reading List Elements

```
for (int i = 0; i < accountList.Count; i++)
{
    Console.WriteLine(accountList[i].GetName());
}
```

- Because we have set the type of elements the `List` can contain there is no need to cast the result
- We can also be sure that a `List` of `Account` type can only hold references to `Account` instances
- We can use subscripts to access elements in the list exactly as for an array
 - Note that the size of the list is given by a `Count` property

12

Generics Lists and Dictionaries

Lists and References

- Remember that a List just contains a bunch of references, not the object itself
- This means that a given object can be on several List instances
 - Each List just contains another reference to the object
- When an object is removed from a list the reference is just removed from that list
- The bank could have one list ordered on account number, another on account balance and so on

13

Generics Lists and Dictionaries

Removing List items

```
Account firstAccount = accountList [0];
accountList.Remove(firstAccount);
```

- The List (and the ArrayList) provide a Remove method that will remove a reference from a list
- The Remove method is given a reference to the item to be removed
- The List is automatically reduced in size when the reference is removed
- The code above removes the item at the start of the list

14

Generics Lists and Dictionaries

Dictionaries

- The List is very powerful, it gives you an array that is always the right size
- However, it is not always what you want
- Sometimes you want to find items based on a key of some kind, rather than looking through elements in a List
- The Dictionary collection is very useful in this situation

15

Generics Lists and Dictionaries

Dictionary Storage

- In Real Life (tm) a dictionary is something you can use to look words up
 - i.e. Given a word and a dictionary you can look in the dictionary and find the entry for that word
- We could use something similar in the bank
 - Given an account number we could look in the 'bank dictionary' and find the account with that number
- This behaviour is used a lot in programs

16

Generics Lists and Dictionaries

Keys and Values

- In database terms the number of the account is the key, and the account data is the value
- Given the key (my account number) I want to find the value (my account)
- If we use a List of accounts the only way to find the account is by searching through the accounts until we find the one with the required account number
 - There are other searching and ordering techniques that can speed this up, but we still have to implement them

17

Generics Lists and Dictionaries

Making a Dictionary

```
Dictionary<string,Account> accountDictionary =
    new Dictionary<string,Account>();
Account a = new Account("Fred");
accountDictionary.Add(a.GetName(),a);
```

- This code makes a dictionary of accounts, indexed on a string – which might be the name of the account holder
 - each key must be unique for this to work, otherwise Add will throw an exception
- It then creates an Account and adds it to the dictionary
- The name of the account holder is used as the key
- We can now use the name to find this account

18

Generics Lists and Dictionaries

Looking up Items

```
// Pay in 50 pounds to Fred's account
accountDictionary["Fred"].PayInFunds(50);
```

- This code looks up the account with the name "Fred" and then pays it 50 pounds
- Note how easy it is to index on the key value
 - It has to be a string because that is how the Dictionary was declared
- Unfortunately if the accountDictionary does not contain a key with the value "Fred" this statement will throw an exception

19

Generics Lists and Dictionaries

Looking for Keys

```
if (accountDictionary.ContainsKey("Fred"))
{
    Console.WriteLine("Account Fred is in the bank");
}
```

- You could catch the `KeyNotFoundException` if a key is not found in a Dictionary
- Alternatively the method `ContainsKey` can be used to look up a key and find out if the dictionary contains it

20

Generics Lists and Dictionaries

Multiple Dictionaries

- You can create multiple dictionaries if you want to index on different elements in the data
- The program only holds one copy of each object, adding a new dictionary does not actually mean storing the data twice, you just need enough space for the index, and your program must now keep track of the two places that store data
 - But remember that this will not work if any items have the same keys – for example two people with the same name
 - You could put a number on the end of additional ones and then search for these – but this can get messy

21

Generics Lists and Dictionaries

Namespaces

```
using System.Collections; // ArrayList  
using System.Collections.Generic; // List and Dictionary
```

- If you want to use these resources you need to add the appropriate namespaces
- If you create an application using the New Project wizard in Visual Studio the using statement for Generic collections is included automatically
- The List and Dictionary collections have taken over from the ArrayList and HashTable collections that used to be used before Generics

Lists, Dictionaries and Generics

- Generics introduces the idea that you can write code that manipulates objects without worrying about the precise object type
- This is particularly useful when managing collections
 - It is also used by the XNA content manager
- Lists can hold collections, and Dictionary can hold references managed by a key
