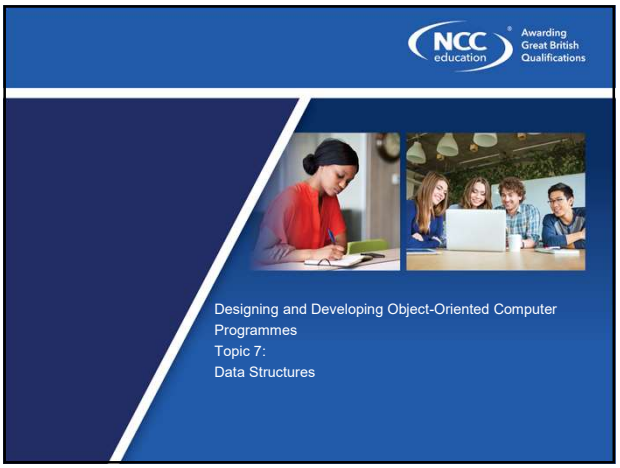
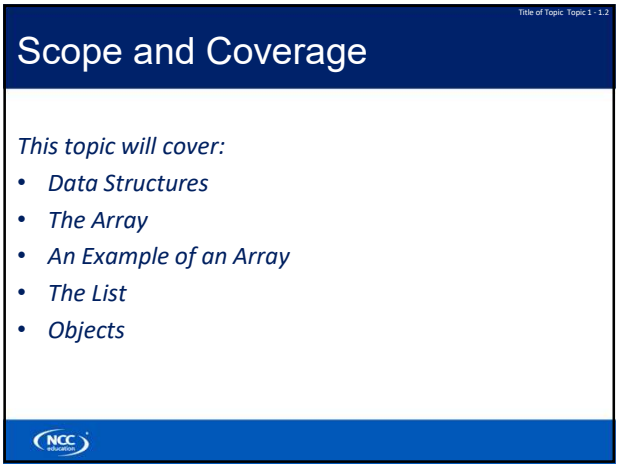
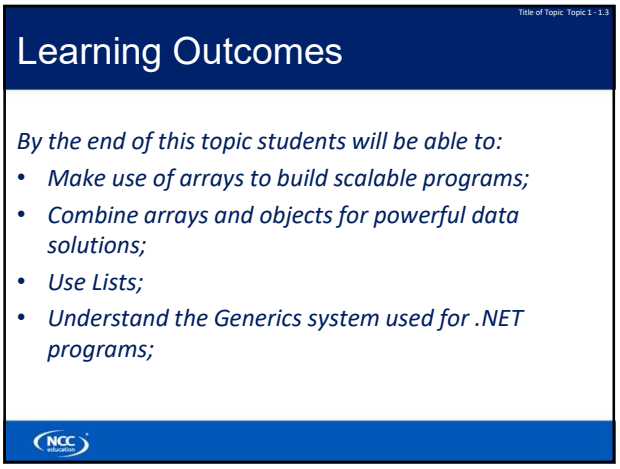

NCC education
Awarding Great British Qualifications
Designing and Developing Object-Oriented Computer Programmes
Topic 7:
Data Structures

1


Title of Topic Topic 1 - 1.2
Scope and Coverage
This topic will cover:
• Data Structures
• The Array
• An Example of an Array
• The List
• Objects

2


Title of Topic Topic 1 - 1.3
Learning Outcomes
By the end of this topic students will be able to:
• Make use of arrays to build scalable programs;
• Combine arrays and objects for powerful data solutions;
• Use Lists;
• Understand the Generics system used for .NET programs;

3

Title of Topic Topic 1 - 1.4

## Data Structures - 1

- In the tasks for Topic 6, you will have discovered that certain elements you might expect from such an application were impossible to implement in a sensible way.
- Most banks don't have three or four accounts – they might have three or four million.
- Most banks don't need you to carefully select between individual accounts to find the right one – they can locate them almost instantly.
- In this chapter, we are going to discuss the tremendous impact that data structures have on your programs.

4

Title of Topic Topic 1 - 1.5

## Data Structures - 2

- In a very real sense, a good data structure is what will determine the effectiveness of a program you write.
- A good data structure is easy to manipulate and quick to access – it makes everything you do easier, and as a result your program code will be cleaner, more efficient, and more maintainable.
- A poor data structure will need you to write complicated code to do simple things, with a corresponding negative impact on code quality.

5

Title of Topic Topic 1 - 1.6

## Data Structures - 3

- For the moment, we have largely been dealing with poor data structures – individual objects which we must check one at a time, for example.
- From now on, we're going to make our own lives much easier.

6

Title of Topic Topic 1 - 1.7

## The Array - 1

- Here's the problem we have at the moment – our variables live an isolated existence with no way, in code, to link them together.
- If we create five integers to represent age, we can't tell the program to group them together into 'a list of ages'.
- If we could, we'd be able to do things like 'read each age in this list and give me all of them that are greater than 18' or 'count how many ages in this list are equal to 30'.

7

Title of Topic Topic 1 - 1.8

## The Array - 2

- Since we can't do that, we have to write the code that groups them together ourselves, with individual comparisons.
- The more variables we have, the more comparisons we have to make.

8

Title of Topic Topic 1 - 1.9

## The Array - 3

- The good news is, we can create that list – we do it through the use of a data structure called an array.
- This allows us to create a single variable that has many compartments within it – in each compartment, we store an individual piece of data.
- Instead of using age1, age2, age3 and so on we can create an array called ages, and store each age in its own compartment:

| Ages |
|---|
| 20 |
| 25 |
| 18 |

9

Title of Topic Topic 1 - 1.10

## The Array - 4

- Each of these compartments also has an address that we can use to extract a specific age if we need it.
- In C#, we begin counting at zero (you've seen this in how we do for loop counters) and arrays are no different.
- Every compartment gets its own numerical address – this is known as an **index**:

| Ages | |
|---|---|
| 0 | 20 |
| 1 | 25 |
| 2 | 18 |

10

Title of Topic Topic 1 - 1.11

## The Array - 5

- When we make use of an array in code, we use a slightly different notation – we create a variable for the array in the usual way, but we put square brackets on the type information:

```
int[] ages;
```

11

Title of Topic Topic 1 - 1.12

## The Array - 6

- This creates a container for the array, but it doesn't put anything in it. We have to actually **create** the array at an appropriate point (such as in the constructor of an object, or the load event of a form).
- When we do this, we tell the program how big we want the array to be, like so:

```
ages = new int[3];
```

12

Title of Topic Topic 1 - 1.13

## The Array - 7

- The index means that we can access individual ages with the same precision as we do for a variable, but that we can do it with a single, consistent variable name.
- In C#, we access the contents of an array index (known as an element) with a square bracket notation.
- If we wanted to change the second age in our array to 40, we'd do this:

```
ages[1] = 40;
```

13

Title of Topic Topic 1 - 1.14

## The Array - 8

- At the end of this line of code our array would look like this:

| Ages | |
|---|---|
| 0 | 20 |
| 1 | 40 |
| 2 | 18 |

- If we wanted to put the third element in a text box, we'd do something similar:

```
txtOutput.text = "The third age is " +ages[2];
```

14

Title of Topic Topic 1 - 1.15

## The Array - 9

- The size of an array is **whatever we want it to be**, which means the code difference between an array of ten, twenty, or twenty-thousand elements is **nothing**.
- This means our programs, for the first time, become truly **scalable** – as we need larger amounts of data stored, our code stays as complex, or otherwise, as it was when we needed to store smaller amounts.
- That's because **you can use a variable to provide the index.**

15

Title of Topic Topic 1 - 1.16

## The Array - 10

- This is, you will find, an amazing benefit to someone writing a computer program. Let's see what we can do!

16

Title of Topic Topic 1 - 1.17

## An Example of an Array - 1

- We're going to create a program here that will give us the average of ten numbers we input.
- This is an amount of numbers we have never even dreamed of before.
- Consider a program making use of a textbox (txtToAdd) and two buttons (cmdAdd and CmdCalculate).
- We'll also use a label, lblNum that shows us how many numbers we've already got added.

17

Title of Topic Topic 1 - 1.18

## An Example of an Array - 2

- We need to add code to create our array, which we'll call **nums**:

```
int[] nums;

private void frmAverage_Load(object sender, EventArgs e)
{
  nums = new int[10];
}
```

18

Title of Topic Topic 1 - 1.19

## An Example of an Array - 3

- We'll need the code that lets us add a number to our array.
- We'll also need to keep track of the next index into which we will insert the element.
- We'll also make sure we can't overfill the array (if we do, the program will crash).

```
int[] nums;
int currentIndex;

private void frmAverage_Load(object sender, EventArgs e)
{
  nums = new int[10];
  currentIndex = 0;

}
```

19

Title of Topic Topic 1 - 1.20

## An Example of an Array - 4

- The array we have at this point looks like this:

| Nums | |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

20

Title of Topic Topic 1 - 1.21

## An Example of an Array - 5

- We want to start filling these compartments with what the user enters, so we implement the code for our add button:

```
private void cmdAdd_Click(object sender, EventArgs e)
    {
      int num;

      if (currentIndex == 10)
      {
        MessageBox.Show("You have filled the array.  Press calculate.");
        return;
      }

      num = Int32.Parse(txtToAdd.Text);

      nums[currentIndex] = num;

      currentIndex = currentIndex + 1;

      lblNum.Text = currentIndex + " numbers in array.";
    }
```

21

Title of Topic Topic 1 - 1.22

## An Example of an Array - 6

- If our user has the number 20 in the textbox, then this function checks to see if we have a currentIndex that does not equal ten – if we do, we'd be trying to put something into position 11 of the array (remember, we start counting from zero) and we only have ten.
- If that's not a problem, we parse the number out of the text box, and then put the number into the array at the position indicated by currentIndex.

22

Title of Topic Topic 1 - 1.23

## An Example of an Array - 7

- To begin with, we set currentIndex to zero – the end result is that our array now looks like this:

| Nums | |
|---|---|
| 0 | 20 |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

23

Title of Topic Topic 1 - 1.24

## An Example of an Array - 8

- At the end of this, currentIndex is increased by 1 (to one), and the label is updated to show how many numbers we've entered.
- The next time we press the button, the number will go into index 1, and then 2, and then 3, and then so on until we reach the end.

24

Title of Topic Topic 1 - 1.25

## An Example of an Array - 9

- Note here that we don't need to store the variable anywhere other than the array – the **num** variable we are using within the function is only to hold the temporary result of the integer parsing.
- The array is the only place we keep this data.

25

Title of Topic Topic 1 - 1.26

## An Example of an Array - 10

- That means that we can then easily calculate our average by summing up the numbers we have and then dividing them by the current index.
- If we were doing this the bad old way, it would look like this:

```
total = nums[0] + nums[1] + nums[2]... + nums[9];
```

26

Title of Topic Topic 1 - 1.27

## An Example of an Array - 11

- We're not going to do that though – we're going to make use of a loop to do it for us.
- The fact that we can index an array using a variable is incredibly powerful, because it reduces summing up to this:

```
private void cmdCalculate_Click(object sender, EventArgs e)
{
  int total;

  total = 0;

  for (int i = 0; i < currentIndex; i++)
  {
    total = total + nums[i];
  }

  MessageBox.Show("The average is " + total / currentIndex);

}
```

27

Title of Topic Topic 1 - 1.28

## An Example of an Array - 12

- But that's not all. Because here's why arrays are going to change your world – this code looks **exactly the same** if we're averaging ten numbers, or ten thousand.
- All we need to do is change the size of the array:

```
nums = new int[10000];
```

28

Title of Topic Topic 1 - 1.29

## An Example of an Array - 13

- And the check that stops us from overfilling:

```
if (currentIndex == 10000)
{
  MessageBox.Show("You have filled the array.  Press calculate.");
  return;
}
```

29

Title of Topic Topic 1 - 1.30

## An Example of an Array - 14

- Or even quicker still – an array is an object, and it has properties.
- One of these properties is **Length**, and that gives us back the size of the array:

```
if (currentIndex == nums.Length)
{
  MessageBox.Show("You have filled the array.  Press calculate.");
  return;
}
```

30

Title of Topic Topic 1 - 1.31

## An Example of an Array - 15

- All we need to change, whenever we want to scale up to massive amounts of data, is the code that deals with how many elements we want.
- Nothing else, not even the average calculation, needs to change at all.

31

Title of Topic Topic 1 - 1.32

## An Example of an Array - 16

- That's amazing, but sometimes even that isn't enough.
- Sometimes we don't even want to say how many things should be there in the first place.
- For that, we need to use a slightly more complex data structure – **the List**.

32

Title of Topic Topic 1 - 1.33

## The List - 1

- Think of the **List** as an array that grows and shrinks as you need it.
- This is a special, powerful object in C# that uses up some more computing resources (CPU and memory) but in exchange for a flexible data structure that doesn't need us to know when we create the code how many things we'll be dealing with.

33

Title of Topic Topic 1 - 1.34

## The List - 2

- Think of the Array above as a **bounded** data structure and the List as an **unbounded** structure and you'll understand the circumstances under which we might wish to use either.

34

Title of Topic Topic 1 - 1.35

## The List - 3

- The List is a more advanced kind of structure known as a Collection, and we need to tell .NET where to find it.
- We add the following line to the top of any program making use of a List:

```
using System.Collections;
```

35

Title of Topic Topic 1 - 1.36

## The List - 4

- However, when we declare a List we need to use a slightly odd syntax known as the **Generic** system.
- Think back to when we created our classes in a previous chapter – imagine if we could provide data types when we actually created a class, and those would be reflected in the variables we used inside it.
- We won't talk about how to do that, not here, but many classes in .NET do this through the use of generics.

36

Title of Topic Topic 1 - 1.37

## The List - 5

- We provide type information when we create the object, and when we declare its variable – like so:

```
List<int> nums;
```

- And when we create the object:

```
nums = new List<int>();
```

37

Title of Topic Topic 1 - 1.38

## The List - 6

- Otherwise, we make use of this the same way we do any object.
- We add things on to the end to the List using the Add method, but we can still use the square bracket notation if we want specific elements.
- What we don't need to do though is track what number of the index we're currently at, because the List does it for us.

38

Title of Topic Topic 1 - 1.39

## The List - 7

- If we want to find out how many things have been added, we make use of the Count property of the list.
- Since we don't set a size of the List, there's no need for us to have a check to make sure it isn't overfilled:

```
private void cmdAdd_Click(object sender, EventArgs e)
{
  int num;

  num = Int32.Parse(txtToAdd.Text);

  nums.Add(num);

  lblNum.Text = nums.Count + " numbers in array.";
}
```

39

Title of Topic Topic 1 - 1.40

## The List - 8

- And:

```
private void cmdCalculate_Click(object sender, EventArgs e)
{
  int total;

  total = 0;


  for (int i = 0; i < nums.Count ; i++)
  {
    total = total + nums[i];
  }

  MessageBox.Show("The average is " + total / nums.Count);

}
```

- This program works exactly the same way as the original one we did for arrays, but with a greater degree of flexibility because the List will grow as we need.

40

Title of Topic Topic 1 - 1.41

## Objects - 1

- Both of these data structures are very powerful when dealing with ints, strings and the other primitive data types.
- They become incredibly powerful when dealing with objects – and they can do exactly that.

41

Title of Topic Topic 1 - 1.42

## Objects - 2

- Consider the following class:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace _7._2
{
  class Person
  {
    private String name;
    private int age;

    public Person(String n, int a)
    {
      name = n;
      age = a;
    }

    public string Name
    {
      get
      {
        return name;
      }

      set
      {
        name = value;
      }
    }
    public int Age
    {
      get
      {
        return age;
      }

      set
      {
        age = value;
      }
    }
  }
}
```

42

Title of Topic Topic 1 - 1.43

## Objects - 3

- If we wanted to make a simple database making use of arrays, lists and objects, we can do that.
- Consider the following interface.
- There are three text boxes (txtSearch, txtName and txtAge), and two buttons (cmdAdd and cmdFind).

43

Title of Topic Topic 1 - 1.44

## Objects - 4

- Here, when we press Add Person we are going to take their name and age, create an object, and store it in a List.
- If we wanted to use an array, we'd create it like so:

```
Person[] people;

private void frmNamesAndAges_Load(object sender, EventArgs e)
{
  people = new Person[10];
}
```

44

Title of Topic Topic 1 - 1.45

## Objects - 5

- We're not going to do that, but the technique is very similar.
- The only difference is that we'd need to keep track of our currentIndex if we were doing this with a standard array.
- We'll be using a List, derived from Generic syntax:

```
List<Person> people;


private void frmNamesAndAges_Load(object sender, EventArgs e)
{
  people = new List<Person>();

}
```

45

## Objects - 6

- Creating the object follows the pattern we discussed in Topic 5, while putting it in the array based on the Add method we discussed above:

```
private void cmdAdd_Click(object sender, EventArgs e)
{
  string name;
  int age;
  Person p;

  name  = txtName.Text;
  age = Int32.Parse(txtAge.Text);

  p = new Person(name, age);

  people.Add(p);
}
```

46



## Objects - 7

- To find the right person, we'll go over each object in the list and see if their name matches the one that we entered into the text box.
- If it does, we'll display their age:

```
private void findPerson_Click(object sender, EventArgs e)
    {
      String name;

      name = txtSearch.Text;

      for (int i = 0; i < people.Count; i++)
      {
        if (people[i].Name.ToLower() == name.ToLower())
        {
          MessageBox.Show(name + " is " + people[i].Age);
        }
      }
    }
```

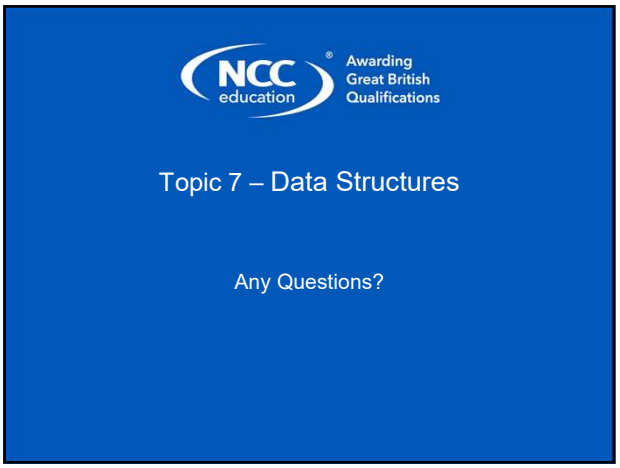
47



## Objects - 8

- Now, think how useful that would have been in the previous chapter when it came to finding an account that matched a PIN.
- This kind of system is hugely powerful, and we'll be seeing it a lot as we progress through this module.

48


NCC education
Awarding Great British Qualifications
Topic 7 – Data Structures
Any Questions?

49