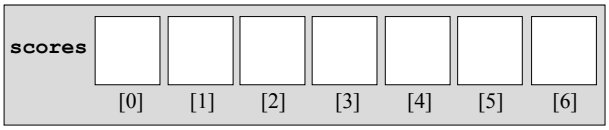
Arrays

How to store lots of data

Arrays

- An *array* is a sequence of variables of the same type
- It is used to group together a number of values that are going to be treated in the same way. For example:
  - the marks for all the students in a class.
  - the scores for all the games in a league.

Arrays



- Arrays are often represented as a sequence of slots as shown above
- Each slot can hold one item (or element).
- The slots are numbered so that the elements can be accessed.

Defining an array

We can define an array variable called scores using the statement:

```
int [] scores;
```

We can create an array object that can hold seven elements as follows

```
new int [7];
```

This would usually be assigned to an array variable:

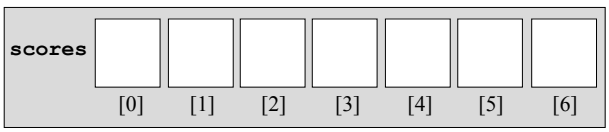
```
scores = new int [7];
```

Defining an array

So to create the seven slots called scores we have

```
int [] scores;
scores = new int [7];
```

Scores can now hold 7 integers.



Array Construction Syntax

Syntax	`new typeName[length]`
Example	`new int[7]`
Purpose	To construct an array with a given number of elements

Accessing an element of an array

- The scores array can now be treated as if it was seven integer variables.
- We can access variables using the index; the first element of the array is **`scores[0]`**, the last is **`scores[6]`**.
- We can use these as normal integer variables

```
scores[0] = 10;
scores[4] = scores[3] * 5;
```

Array Element Access

Syntax

```
arrayReference[index]
```

Example

```
scores[0] = 23;
```

Purpose

To access an element in an array

Accessing an element of an array

- We could initialise all the elements in the array as follows

```
scores[0] = 10;
scores[1] = 23;
scores[2] = 99;
scores[3] = 9;
scores[4] = 1;
scores[5] = 2;
scores[6] = 20;
scores[7] = 53;
```

← Out of bounds error

Java Checks Array Bounds For You

`scores[7]`

This does not exist, since the 7 elements in `scores` run from 0 to 6

If you try and use this nonexistent value, Java throws an `ArrayIndexOutOfBoundsException` error.

These kinds of errors (exceptions) will be covered later.

Index Vs Contents

- Many beginning programmers confuse the index and the contents

```
scores[0] = 10;
scores[1] = 23;
scores[2] = 99;
scores[3] = 9;
scores[4] = 1;
scores[5] = 2;
scores[6] = 20;
```

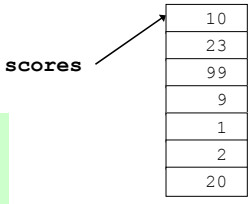
← contents

indices

Creating and initialising an array

An array can be initialised when it is defined:

```
int [] scores = {10, 23, 99, 9, 1, 2, 20};
```



An array is an object: like a String it can be created without the new operator.

Creating and initialising an array

- This is equivalent to

```
int [] scores = new int[7];

scores[0] = 10;
scores[1] = 23;
scores[2] = 99;
scores[3] = 9;
scores[4] = 1;
scores[5] = 2;
scores[6] = 20;
```

Another example

```
int[] primes = {2,3,5,7,11,13,17,19,23,29};
```

Note: it must be done in the definition statement; the following is illegal in Java.

```
int [] primes;
primes = {2,3,5,7,11,13,17,19,23,29};
```

Back to the index

- The index can be any integer expression

```
scores[2] = 99;
```

any integer expression

- For example

```
int i = 3;
scores[i] = 6;

scores[i] = scores[i + 1];
scores[i - 1] = 2 * i;
```

More Examples

- Some example definitions

```
int [] scores;
double [] amounts;
String [] names;
```

More Examples

- Creating arrays

```
scores = new int[10];
amounts = new double[15];
names = new String[7];
```

- In general, new *type*[*size*] creates an array of *size* variables of type *type*.

More Examples

- As with other variable types, you can define and initialise an array in one line:

```
int [] scores = new int[10];
double [] amounts = new double[15];
String [] names = new String[7];
```

- Now we have array objects, called scores (that can hold 10 ints), amounts (that can hold 15 doubles), and names (that can hold 7 String objects)

More points about array creation

```
new type[size]
```

- *size* can be any integer or integer expression:

```
int i = 3;
char [] name = new char[i];
```

is equivalent to:

```
char [] name = new char[3];
```

More points about array creation

- The memory in an array is automatically initialized for us (to 0 for numbers, false for booleans, or null for objects)

Example Program

- Read in 10 integer values and print them out.

```
public class Simple
{
   public static void main(String[] args)
   {
      int [] scores = new int[10];

      System.out.println("Enter ten scores: ");
      scores[0] = Console.readInt( );
      scores[1] = Console.readInt( );
      scores[2] = Console.readInt( );
      scores[3] = Console.readInt( );
      scores[4] = Console.readInt( );
      scores[5] = Console.readInt( );
      scores[6] = Console.readInt( );
      scores[7] = Console.readInt( );
      scores[8] = Console.readInt( );
      scores[9] = Console.readInt( );

      print(scores[0] + " " + scores[1] + " "
       + scores[2] + " " + scores[3] + " " + scores[4]
       + " " + scores[5] + " " + scores[6] + " "
       + scores[7] + " " + scores[8] + " " + scores[9]);
   }
}
```

Of course we can do this with a loop

Implementing the loop

```
scores[0] = Console.readInt( );
scores[1] = Console.readInt( );
scores[2] = Console.readInt( );
scores[3] = Console.readInt( );
scores[4] = Console.readInt( );
scores[5] = Console.readInt( );
scores[6] = Console.readInt( );
scores[7] = Console.readInt( );
scores[8] = Console.readInt( );
scores[9] = Console.readInt( );
```

This is basically one statement executed 10 times

```
scores[ x ] = Console.readInt();
```

with x varying from 0 to 9

Implementing the loop

```
scores[0] = Console.readInt( );
scores[1] = Console.readInt( );
scores[2] = Console.readInt( );
scores[3] = Console.readInt( );
scores[4] = Console.readInt( );
scores[5] = Console.readInt( );
scores[6] = Console.readInt( );
scores[7] = Console.readInt( );
scores[8] = Console.readInt( );
scores[9] = Console.readInt( );
```

So we use a for loop

```
for(x = 0; x < 10; x++)
   scores[ x ] = Console.readInt();
```

```
public class Simple
{
   public static void main(String[] args)
   {
      int [] scores = new int[10];

      System.out.println("Enter ten scores: ");
      for(int i = 0; i < 10; i++)
         scores[i] = Console.readInt( );

      for(int i = 0; i < 10; i++)
         System.out.print(scores[i] + " ");
   }
}
```

Arrays and for loops are partners

- Every array object has a length attribute which tells how many elements are in the array.
- If **a** is an array, **a.length** says how many elements it has.
- To print all elements of the array scores:

```
for(i = 0; i < scores.length; i++)
   System.out.println(scores[i]);
```

Read 10 numbers and print them in reverse order

```
class Reverse
{
   public static void main(String[] args)
   {
      int[] scores = new int[10];

      println("Enter ten numbers: ");
      for(int i = 0; i < 10; i++)
         scores[i] = Console.readInt();

      for(int i = 0; i < 10; i++)
         System.out.print(scores[?????] + " ");
   }
}
```

Another Way to Do It

```
public class Reverse2
{
   public static void main(String[] args)
   {
      int[] counts = new int[10];

      println("Enter ten numbers: ");
      for(int i = 0; i < 10; i++)
         counts[i] = Console.readInt();

      for(int i = 9; i >= 0; i--)
         System.out.print(counts[i] + " ");
   }
}
```

Things to remember about an Array

1. The array's *name*
2. The *type* of values that it can hold
3. The number of values it can hold

   An array object has a length attribute

   (E.g. **scores.length**)

An Array's Size is Fixed When Created

- An array cannot grow. Its size is determined when it is created.
- What happens if we later want to increase it?
- One simple way of dealing with this is to make a large array and only partially fill it.

Old Example

```
public class Scores
{
   public static void main(String [] args)
   {
      int count = 0;
      double total = 0;
      int score = Console.readInt();
      while(score != -1)
      {
         count++;
         total = total + score;
         score = Console.readInt();
      }
      println("There were " + count + " scores");
      println("Average is " + total / count);
   }
}
```

Used and Forgotten

- Each time we read in a value, we count it, add it to the total and then forget it.
- But suppose we had a different problem
  - Read in the scores.
  - print out only the above average scores.
- We would have to remember the scores so that we could later print them.
  - We would use an ***array*** to store the values.
  - We would then calculate the average and print out any scores that were above the average.

Example: Printing above average scores

```
int [] score = {25, 82, 38, 40, 2, 77, 65};

// now find the average
int sum = 0;
for(i = 0; i < score.length; i++)
   sum = sum + score[i];
average = sum / score.length;

// Print out any scores bigger than average
for(i = 0; i < score.length; i++)
   if(score[i] > average)
      println(score[i]);
```

Method to convert an integer to a day of the week

```
final String [] NAME = { "Monday",
 "Tuesday", "Wednesday", "Thursday",
   "Friday",  "Saturday", "Sunday"};

// day must be in range 0..6
public void printName(int day)
{
   System.out.print(NAME[day]);
}
```

Summary

- An array holds a group of related variables
- Defined using `int [] marks`
- Created using `new type[size]`
- Accessed using the index, e.g. `marks[4]`
- Commonly used with for loops