

Java

Arrays and exceptions

It is a capital mistake to theorise before one has the data

Sherlock Holmes

Lecture objectives

To be able to understand the following fundamental concepts of the Java programming language:

- Arrays
- Exceptions

Arrays

- An array is a simple but powerful programming construct
- Consider this cumbersome expression

```
avgSales = (janSales + febSales + marSales + aprSales + maySales +  
            junSales + julSales + augSales + sepSales + octSales + novSales +  
            decSales) / 12;
```

- If the *type* of each field is identical we can create an *array* or *list of values*
- Instead of defining 12 variables, we define one and access it with an *index*

Arrays

- An *array* is an ordered list of values

**The entire array
has a single name**

Each value has a numeric *index*



	0	1	2	3	4	5	6	7	8	9
scores	79	87	94	82	67	98	87	81	74	91

An array of size N is indexed from zero to N-1

This array holds 10 values that are indexed from 0 to 9

Arrays

- A particular value in an array is referenced using the array name followed by the index in square brackets
- For example, the expression

`scores[2]`

refers to the value 94 (which is the 3rd value in the array)

	0	1	2	3	4	5	6	7	8	9
scores	79	87	94	82	67	98	87	81	74	91

Arrays

- An array stores multiple values of the same type
- That type can be primitive types or objects
- Therefore, we can create
 - an array of `integers`
 - an array of `doubles`
 - an array of `characters`, or
 - an array of `String` objects etc.
- In Java, the array itself is an object

Declaring Arrays

- The `scores` array could be declared as follows:

```
int[ ] scores = new int[10];
```

- The *type* of the variable `scores` is `int[]` (an array of integers)
- ie. a new array object that can hold 10 integers

```

public class BasicArray
{
    final static int MULTIPLE = 10;
    //-----
    // Creates an array, fills it, modifies one value, then prints them out.
    //-----
    public static void main (String[] args)
    {
        int[ ] list = new int [3];

        for (int index = 0; index < 3; index++)
            list[index] = index * MULTIPLE;

        list[1] = 999;                                // change one array value

        for (int index = 0; index < 3; index++)
            System.out.print (list[index] + " ");
    }
}

```

0 10 20

0 999 20

Practice

- Create an array that hold 5 integers
- Write a For loop to loop 5 times, each time prompting the user to enter an integer
- Store each integer in the array
- Now create another For loop (the same)
- Access each value in the array, print them and total them
- Print the total

```
public class AddNumbers
{
    public static void main (String[] args)
    {
        int total;
        int[ ] numbers = new int[ ];
        String inputNum;

        for (int index = 0; index < 5; index++)
        {
            inputNum = JOptionPane.showInputDialog
                ("Enter an integer: ");
            numbers[index] = Integer.parseInt(inputNum);
        }

        for (int index = 0; index < 5; index++)
        {
            System.out.println(numbers[index] + " ");
            total = total + numbers[index];
        }
        JOptionPane.showMessageDialog (null, "total = " + total);
    }
}
```

```
Enter an integer: 10
Enter an integer: 20
Enter an integer: 30
Enter an integer: 40
Enter an integer: 50
10
20
30
40
50
total = 150
```

Bounds Checking

- Once an array is created, it has a fixed size
- An index used in an array reference must specify a valid element
- That is, the index value must be in bounds (0 to $n-1$)
- The Java interpreter will throw an exception if an array index is out of bounds

Bounds Checking

- Each array object has a public constant called `length` that stores the size of the array ie the number of elements
- It is referenced using the array name (just like any other object):

`scores.length`

```
public class ReverseNumbers
{
    public static void main (String[] args)
    {
        double[] numbers = new double[3 ];
        String inputNum;

        System.out.println ("Array size = " + numbers.length);

        for (int index = 0; index < numbers.length; index++)
        {
            inputNum =JOptionPane.showInputDialog
                ("Enter number " + (index+1) + ": ");
            numbers[index] = Double.parseDouble(inputNum);
        }

        System.out.println ("The numbers in reverse:");

        for (int index = numbers.length-1; index >= 0; index--)
            JOptionPane.showMessageDialog (null,
                numbers[index] + " ";
        }
    }
}
```

```
Array size = 3
Enter number 1: 1
Enter number 2: 2
Enter number 3: 3
The numbers in reverse:
3.0 2.0 1.0
```

Array Declarations

- The brackets of the array type can be associated with the *type* or with the name of the array
- Therefore the following declarations are equivalent:

```
double[] prices;
```

```
double prices[];
```

- The first format is more common

Initializer Lists

- An *initializer list* can be used to instantiate and initialize an array in one step
- Values are delimited by braces and separated by commas
- Examples:

```
int[ ] units = {147, 323, 89, 933, 540, 269, 97, 114};
```

```
char[ ] letterGrades = {'A', 'B', 'C', 'D', 'F'};
```

Initializer Lists

- Note that when an initializer list is used:
 - the `new` operator is not used
 - no size value is specified
- The size of the array is determined by the number of items


```
//*****  
// GradeRange.java  
// Demonstrates the use of an array of String objects.  
//*****
```

```
public class GradeRange  
{  
    public static void main (String[] args)  
    {  
        String[] grades = {"HD", "D", "CR", "P", "N"};  
  
        int[] cutoff = {80, 70, 60, 50, 0};  
  
        for (int level = 0; level < cutoff.length; level++)  
            System.out.println (grades[level] + "\t" + cutoff[level]);  
    }  
}
```

HD	80
D	70
CR	60
P	50
N	0

Passing Array elements to Methods

- You can pass a single array element to a method in exactly the same way as you pass a variable
- The variables are local to the method and any changes to variables passed into methods are not permanent
 - ie changes are not reflected in the main() program

```

public class PassArrayElement
{
    public static void main (String [ ] args)
    {
        int [ ] nums = {5, 10, 15};
        for (int index = 0; index < nums.length; index++)
            System.out.println("In main method " + nums[index]);

        for (int index = 0; index < nums.length; index++)
            methodGetOneInteger(nums[ index]);

        for (int index = 0; index < nums.length; index++)
            System.out.println("At end of main method " + nums[index]);
    }

    public static void methodGetOneInteger (int num)
    {
        System.out.println("In methodGetOneInteger " + num);
        num = 999;
        System.out.println("After change " + num);
    }
}

```

```

In main method 5
In main method 10
In main method 15
In methodGetOneInteger 5
After change 999
In methodGetOneInteger 10
After change 999
In methodGetOneInteger 15
After change 999
At end of main method 5
At end of main method 10
At end of main method 15

```

Passing Arrays to Methods

- Arrays are objects
- Arrays are passed by reference
 - the method receives the memory address of the array and has access to the actual values in the array elements
 - the method can therefore change values in the array

```
public class PassArray
{
    public static void main (String [ ] args)
    {
        int [ ] nums = {5, 10, 15};
        for (int index = 0; index < nums.length; index++)
            System.out.println("In main method " + nums[index]);

        methodGetsArray(nums);

        for (int index = 0; index < nums.length; index++)
            System.out.println("At end of main method " + nums[index]);
    }

    public static void methodGetsArray (int [ ] arr)
    {
        for (int index = 0; index < arr.length; index++)
        {
            System.out.println("In methodGetsArray " + arr [index]);
            arr [index] = 999;
        }
    }
}
```

In main method 5
In main method 10
In main method 15
In methodGetsArray 5
In methodGetsArray 10
In methodGetsArray 15
At end of main method 999
At end of main method 999
At end of main method 999

Exceptions

- When a Java program performs an illegal operation,
 - an *exception* happens
- Exceptions occur during program execution – not compilation ie errors the compiler cannot detect
- If a program has no special provisions for dealing with exceptions, it will behave badly if one occurs
 - the program will terminate immediately
- Java provides a way for a program to detect that an exception has occurred and execute statements that are designed to deal with the problem.

```
//*****  
// ZeroTest.java    Author: Lewis and Loftus  
// Demonstrates an exception - Division by zero  
//*****
```

```
public class ZeroTest  
{  
    public static void main (String[] args)  
    {  
        int numerator = 10;  
        int denominator = 0;  
  
        System.out.println (numerator / denominator); //causes an exception  
                                                        // program will crash.  
  
        System.out.println ("This text will not be printed.");  
    }  
}
```

Handling Exceptions

- When an exception occurs (is *thrown*), the program has the option of *catching* it.
- In order to catch an exception, the code in which the exception might occur must be enclosed in a `try` *block*.
- After the `try` block comes a `catch` *block* that catches the exception (if it occurs) and performs the desired action.

```
try
    block
catch (exception-type identifier)
    block
```


Handling Exceptions

```
try
{
    // one or more statements at least one of which may
    // be capable of causing an exception
}
catch (exceptionName argument)
{
    // one or more statements to execute if the exception
    // occurs
}
```

- If an exception is thrown within the `try` block, and the exception matches the one named in the `catch` block, the code in the `catch` block is executed
- If the `try` block executes normally—without an exception—the `catch` block is ignored

ArithmeticException

- `ArithmeticException`
 - an illegal arithmetic operation is performed
 - eg trying to divide by zero

```
int divisor = 0;
try
{
    quotient = dividend / divisor;
}

catch (ArithmeticException e)
{
    System.out.println("Error: Division by zero");
}
```

NumberFormatException

- `NumberFormatException`
 - Occurs when an attempt is made to convert a string that does not contain the appropriate characters
 - eg trying to convert alpha characters or spaces to an integer

```
try
{
    int num = Integer.parseInt("doh");
}

catch (NumberFormatException e)
{
    System.out.println("Error: Not an integer");
}
```

Valid data entry

- A robust program will not allow erroneous data input by a user to cause it to crash
- If invalid data is entered it needs to be caught
- Put the `try` and `catch` blocks in a loop
- Allow the user multiple attempts to enter a valid data

```
boolean validNumber = false;
```

```
while (!validNumber)
```

```
{
```

```
    String userInputStr = JOptionPane.showInputDialog ("Enter an integer: ");
```

```
    try
```

```
    {
```

```
        userInput = Integer.parseInt(userInputStr);
```

```
        validNumber = true;
```

```
    }
```

```
    catch (NumberFormatException e)
```

```
    {
```

```
        JOptionPane.showMessageDialog ("That number is not an integer");
```

```
    }
```

```
}
```

Practice

- Write a while loop that
 - prompts a user to enter an integer between 1 and 10 inclusive
 - Repeats the prompt while the number is not in that range

```
String numStr;  
int num = 0;  
while (num < 1 || num > 10)  
{  
    numStr = JOptionPane.showInputDialog ("Enter an integer between 1 and 10");  
    num = Integer.parseInt(numStr);  
}
```

- Now use a try/ catch block to catch a `NumberFormatException`

A solution

```
String numStr;  
int num = 0;  
while (num < 1 || num > 100)  
{  
    numStr = JOptionPane.showInputDialog  
        ("Enter an integer between 1 and 10");  
    try  
    {  
        num = Integer.parseInt(numStr);  
    }  
    catch (NumberFormatException e)  
    {  
        JOptionPane.showMessageDialog(null, "Not an integer");  
    }  
}
```


Multiple catch Blocks

```
try
{
    quotient = Integer.parseInt(str1) / Integer.parseInt(str2);
}
catch (NumberFormatException e)
{
    System.out.println("Error: Not an integer");
}
catch (ArithmeticException e)
{
    System.out.println("Error: Division by zero");
}
```

- When an exception is thrown, the first matching `catch` block will handle the exception.

Lecture Outcomes

Today we have covered:

- arrays
 - initializer lists
 - passing individual array elements to methods
 - passing arrays to methods
 - exceptions
-
- Questions?