Data Input

Entering Data at the Keyboard

Learning Objectives

- **Introduce and become familiar with:**
  - **Entering Data at the Keyboard**
  - **Input Buffer**
  - **Input Prompts**
  - **Constants**
  - **Char Data Types**

Hard Coding (1)

- Consider a program to add together 2 integers and output their total e.g. HardCode (overleaf …)
- We can HARD CODE the assignment of the 2 integers to variables within a program HOWEVER EVERY TIME we run the program, the output/results will be identical.
  - This is because the program is running with identical data at all times
- **Hard Coding Values is thus generally undesirable**

```
public class HardCode {

    public static void main(String[] args) {

        int number1, number2;
        int total;

        number1 = 20;
        number2 = 30;

        total = number1 + number2;

        System.out.println("The total is:\t" + total);
    }//main
}//class
```

```
The total is:    50
```

Hard Coding (2)

- If we wish to run the program using different data we have to:
  - Load up the program into an editor like IntelliJ, and
  - Change the 2 integer values assigned to the relevant variables

In other words we need to edit the program every time we wish to have 2 new integer values added together.

- The program uses specific values that can only be changed by editing the program

MUCH BETTER
Data Entry via the Keyboard

- Nearly all programming languages have a facility which allows a user to enter data via the keyboard and store this data in variables
- There is a **pre-written Java class** to help you read values from the keyboard
- This is called the Scanner class

Using the Scanner Class

- To use this pre-written class we **must** include the following statement above our header comment (at the start of the Java program):

```
import java.util.Scanner;

//     This command sets up a link to the
//           pre-written Scanner class

//     This allows the program to accept values
//           from the keyboard
```

More information

- The programs we write can use different types of data (ints, doubles, floats, Strings etc.)
- The Scanner class provides us with several useful commands to read from the keyboard
- The command that we need to use depends on the **TYPE** of data that you wish

Read carefully

Once we have set up the link to the Scanner program we need to create an object of that type in our **main method** to use as part of the connection

```
Scanner keyboard = new Scanner(System.in);
//    Sets up an object of the Scanner class
//          allowing us to use it to
//          read values from the keyboard
```

NB

The following 2 statements are ALWAYS implemented TOGETHER

```
import java.util.Scanner;

        ……

Scanner keyboard = new Scanner(System.in);
```

Example (ReadIntVariables.java)

Prompt the user for an integer value for length

Read and store the number int length

Prompt the user for an integer value for breadth

Read and store the number int breadth

Output “The value entered for length is “ + length

Output “The value entered for breadth is “ + breadth

Code (1)

```
1 import java.util.Scanner;


   /**
   * Created by: Martin
   * Created on: 09/01/2019
   * Program to read integer values from the
              keyboard and print them out
   */
```

Code (2)

```
10   public class ReadIntVariables {
11     public static void main(String [] args) {
12
13       Scanner keyboard = new Scanner(System.in);
         //  Sets up an object of the Scanner class
         //         allowing us to use it to read
         //         values from the keyboard
15        // Declare 2 int variables length and breadth
16        int length, breadth;
```

Code (3)

```
        //  ALWAYS prompt the user for a value first
        //      then read from the keyboard
        //  FIRST - the PROMPT (use a print)
18      //  Read in values for length and breadth
19      System.out.print("Enter a value for length: ");
        //   This returns the next keyboard input as
        //        a value of type int to variable length
        // NEXT - the READ
20      length = keyboard.nextInt();
```

Code (4)

```
        //  REPEAT …. FIRST - the PROMPT
22      System.out.print("Enter a value for breadth: ");
        //  NEXT - the READ
23      breadth = keyboard.nextInt();
25      //  Output values entered for length and breadth
26      System.out.println("Length value entered was: "
                           + length);
27      System.out.println("Breadth value entered was: "
                          + breadth);
29    }//main
30 }//class
```

Output

```
Enter a value for length 10
Enter a value for breadth 5
Length value entered was 10
The value entered for breadth was 5
```

prompt displayed and values (in red) entered by user

2 lines of informative text and values displayed on the screen

Some Comments

- A prompt to the user to enter a value MUST ALWAYS be FOLLOWED by a read statement

- REMEMBER we created a communication link to the keyboard via the Scanner class:

```
Scanner keyboard = new Scanner(System.in);
```

- The read statements use the **keyboard** object to identify what type of data we expect to read from the keyboard

More Comments

```
//    ALWAYS prompt the user for a value first
//          then read from the keyboard
System.out.print("Enter a value for breadth: ");
breadth = keyboard.nextInt();
```

The above fragment of Java code, prompts the user for the breadth.

It expects to read in an integer from the keyboard - **`nextInt()`** **-** to store in the variable **breadth**

print() and println() methods

Note the difference between print and println:

```
System.out.println("Enter a value for breadth: ");
breadth = keyboard.nextInt();
```

```
Enter a value for breadth:
```

```
System.out.print("Enter a value for breadth: ");
breadth = keyboard.nextInt();
```

```
Enter a value for breadth:
```

Inputting a Real Number

- Suppose you want to read in and store the price of an item e.g. 19.99?

- Similar to previous:
  - Declare a variable of type **`double`**
  - Use **`nextDouble()`** **to** read the number from the keyboard and store in the double variable

Example (ReadDoubleVariables.java)

Prompt the user for a value for price1

Read and store the number double price1

Prompt the user for a value for price2

Read and store the number double price2

Output “The value entered for price 1 is “ + price1

Output “The value entered for price 2 is “ + price2

Inputting a Double

```
16  // Declare two double variables for price1 and price2
17  double price1, price2;

    // ALWAYS prompt the user for a value first
    // then read from the keyboard

20  System.out.print("Please enter a value for price 1: ");
21  price1 = keyboard.nextDouble();
22
23  System.out.print("Please enter a value for price 2: ");
24  price2 = keyboard.nextDouble();
```

Output

```
Please enter a value for price 1: 5.6
Please enter a value for price 2: 2

The value entered for price 1 is 5.60

The value entered for price 2 is 2.00
```

prompt displayed and values (in red) entered by user

informative text and values displayed on the screen

NOTE

- Nothing typed in at the keyboard is accepted by the computer until the user entering the data presses the **ENTER** key
- As with output there is an input **buffer**
- Data entered at the keyboard passes initially into an intermediate store called an **input buffer** from where it can be read and used by the program
- The program does not start to read the data until the ENTER key has been pressed

Poor Programming

- It is extremely poor programming practice to expect a user to enter data at the keyboard without first providing some form of instruction about what kind of data is expected
- A user SHOULD NEVER be left not knowing what to do next or what the program expects them to do
- It is normal practice to display a message onscreen inviting or instructing the user to enter whatever data is required
- This type of message is known as an **input prompt**

Example (HolidayCost.java)

Write a program to

- read in the number of people going on holiday
- read in the cost of each flight
- read in the cost of each transfer

The program should use this information to calculate (including VAT at 20%)

- total cost of flights
- total cost of transfers
- total overall cost

and output these details to 2 decimal places

CONSTANT VALUES

Use of finals in Java

Constant Values

- Most programs are written so that they can be run with different data on each occasion
- Hence almost all of the data is stored in variables whose values can be changed
- Sometimes a program may wish to use an item of data which will remain the same regardless of how many times the program is run

Example (contd.)

Note the extra word - final
and the use of uppercase VATRATE

```
17      // Set the VAT rate
18      final double VATRATE = 0.20;

19      // Declare variable for the number of people travelling
20      int noOfPeople;

21      // Declare variables for flight and transfer costs
22      double flightCost, transferCost;

23      // Declare the variables for the totals
24      double flightTotal, transferTotal, total;
```

Example (contd.)

```
26      // Enter the number of people travelling
27      System.out.print("How many people are travelling? ");
28      noOfPeople = keyboard.nextInt();
29
30      // Enter the costs
31      System.out.print("\nEnter the cost of each flight: £ ");
32      flightCost = keyboard.nextDouble();
33      System.out.print("Enter the cost of each transfer: £ ");
34      transferCost = keyboard.nextDouble();
```

Example (contd.)

```
36          //Calculate the totals
37          flightTotal = noOfPeople * (flightCost * (1 + VATRATE));
38          transferTotal = noOfPeople * (transferCost * (1 + VATRATE));
39          total = flightTotal + transferTotal;
40
41          //Output the result of the calculations
42          System.out.println("\nTotal Cost for flights:\t\t\t£" +
43                  df.format(flightTotal));
44          System.out.println("Total Cost for transfers:\t\t£" +
45          df.format(transferTotal));
46          System.out.println("\nTotal Cost \t\t\t\t\t\t£" +
                    df.format(total));
```

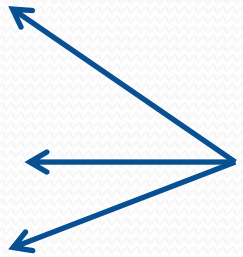
Example - Interaction

The interaction of the above program would be as follows:

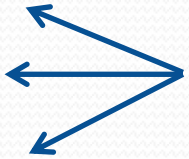
```
How many people are travelling?    4

Enter the cost of each flight:     £400
Enter the cost of each transfer:   £50

Total cost for flights      : £1920.00
Total cost for transfers    : £240.00
Total Cost                  : £2160.00
```



prompt displayed and values (in red) entered by user



informative text and values displayed on the screen

Constant Values

- The value of **VAT** is a defined constant value
  - It WILL NEVER CHANGE regardless of how many times the program is run
- We may wish to run the program with different values of the `noOfPeople`, `flightCost` and `transferCost`
- Java allows us to declare an identifier and assign a value to it in such a way that the value CANNOT be changed
- This kind of identifier is called a CONSTANT IDENTIFIER which is frequently shortened to CONSTANT (or final)

Constant Values

- There is a convention in Java (and C) that identifiers chosen to represent constants should always be in UPPER CASE so a suitable identifier for VAT is VATRATE

- The full declaration of the constant **VATRATE is**:

```
final double VATRATE = 0.20;
```

- The value of VAT is a real number, i.e. it has digits after the decimal point so it must be declared as a double (not an int)

- The reserved word **`final`** indicates that the identifier is a CONSTANT and the value assigned to it CANNOT be changed

Why use finals?

- The use of a constant (or final) is particularly useful if the VAT value is used frequently in a program
- If we wish to use a different value for VAT (say 17.5% or 23%) we need only change the value of the constant once
- This will have a ripple (or knock-on) effect.
- All references to the identifier VAT throughout the program (which are made via the final/constant) will automatically reflect this (revised) value

Inputting a String

- Suppose you want to read in someone's name
  - Declare a variable of type **`String`**
  - Use **`nextLine()`** **to** read the String from the keyboard and store in the String variable

```
Scanner keyboard = new Scanner(System.in);
String name;

System.out.print("Enter your name: ");
name = keyboard.nextLine();

System.out.println("Hello " + name);
```

Output:

```
Enter your name: Martin
Hello Martin
```

Example for you!

DESIGN and DEVELOP a program called **TicketCost.java**

This program should prompt the user for:

- Their name
- The number of adult tickets required
- The number of child tickets required

The program should store this information as variables.

The program should store the cost of an adult ticket (£24.95) and the cost of a child ticket (£14.95) as **CONSTANTs** (**finals**) and use these constants to calculate the user's total bill

The program should output informative text including the user's name and print out their total bill.

CHARACTER SETS

Useful

SELF READING

Character Sets

- Each computer system has an associated **character set**, which may be described as:

  "all the individual symbols that may be typed in at the keyboard"

- This means:
  - ALL the upper case letters of the alphabet (e.g. 'A', 'B' … 'Z')
  - ALL the lower case letters of the alphabet (e.g. 'a', 'b' … 'z')
  - The numerals (or digits) 0..9
  - A range of symbols used in punctuation, mathematics, currency etc. (';', '*', '£' etc.)

- The character set also includes a number of keys which might not, at first sight, be thought of as characters e.g. the ENTER key, the SPACE BAR, the ESC key

Character Sets

- Character variables are capable of holding a single character
- In Java a character value must be enclosed INSIDE SINGLE QUOTES
- Thus typical assignment statements with character variable might be as follows:

```
// Declare three character variables
char char1, char2, char3;

char1 = 'A';    // put letter 'A' into the first
char2 = '7';    // put numeral '7' into the second
char3 = '$';    // put symbol '$' into the third
```

ASCII

- Each character in a set is identified by a unique pattern of eight bits (1's and 0's) known as its binary code
- Until recently almost all computing systems use the same coding system known as the

American Standard Code for Information Interchange

- This is usually referred to as **ASCII Code**
- Every character in the set is mapped onto a binary number with values in the range 0..127
- A table giving the ASCII code for each character may be found in at the back of most good Java books

More on ASCII

- More modern computers use an extension of ASCII known as UNICODE
- This involves adding a second set of 8 bits onto the original ASCII code and then using the value of the second set of 8 bits to vary the nature of the character set to be used
- This allows the computer to be used in a range of languages e.g. Greek, Hebrew, Arabic, Russian, Chinese, Japanese etc. which do not use the same letters as English
- The first 128 UNICODE characters are identical to the ASCII characters
- For all practical purposes however, it is the ASCII codes which are important

Casting Explored

- As each character has an associated number it is possible to assign a character variable an integer value representing an ASCII code

```
char char1 = 65;     //     Gives char1 the value 'A'
```

- You can then easily switch from character data to numerical data and vice versa.
- This can be done using **casting**

- Placing **(int)** before a character value will return its associated decimal value

```
(int)'A'     gives     value 65
(int)'B'     gives     value 66
(int)'0'     gives     value 48
(int)'1'     gives     value 49
(int)'a'     gives     value 97
```

Conversely

- Placing **(char)** before a number in the range 0..127 will return its character value

```
(char)46      gives value   '0'
(char)49      gives value   '1'
                ...
(char)65      gives value   'A'
(char)66      gives value   'B`
                ...
(char)90      gives value   `Z'
                ...
(char)97      gives value   'a`
(char)98      gives value   `b`
```

Conversely

Note 1

Numeric codes for letters are arranged in ascending order so that character data can be sorted into alphabetical order

Note 2

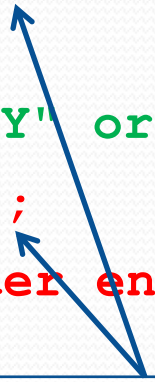
It is important to distinguish between an integer value in the range 0 .. 9 and the set of numerals or digits which are characters in the range '0' .. '9'

Input of Single Characters

- Sometimes you want to read a single character from the keyboard
- For example your program might ask the user a yes/no question and allow them to enter **'Y'** for yes or **'N'** for no
- **The Scanner class does NOT have a method to read single characters directly from the keyboard**
- We must use the **next()** or **nextLine()** methods from the Scanner class to read a String from the keyboard and then convert it to a character

Example

```
15  String input;          // Declare a String variable
16  char letter;           // Declare a character variable letter
17
18  //  Prompt for and accept a single character typed in
19  //    at the keyboard and assign it to the variable input
20  System.out.print("Are you happy? (Y=yes, N=no): ");
21  input = keyboard.nextLine();
22
23  //  Convert the String "Y" or "N" to a character 'Y' or 'N'
24  letter = input.charAt(0);
25  System.out.println("Letter entered was " + letter);
```



NB Lines 21 and 24 could be combined into one statement:

```
letter = keyboard.nextLine().charAt(0);
```

For now .....

- Just get used to keying in this code and using it to accept characters and print them to the screen

- If a user enters a lower case letter into a program from the keyboard how would I convert it to uppercase?

- What is the difference between the

  `keyboard.next()` method and the
  `keyboard.nextLine()` method?

Using JOptionPane to Input Data

- The JOptionPane class can be used for input from the keyboard

- Example: Addition Tutor

  - Suppose we wish to create an addition tutor which prompts the user to type in the answer to the addition of two numbers

  - The program indicates whether or not the user has typed in the correct answer

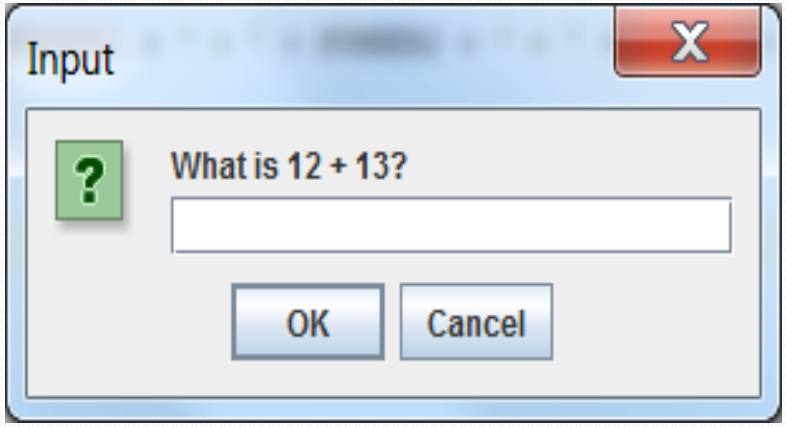
AdditionTutor1.java

```
1    import javax.swing.*;
     …
9    public class AdditionTutor1 {
10      public static void main(String[] args) {
11
12         final int NUMBER1 = 12, NUMBER2 = 13;
13         int answer;
14         String answerString;
15
16         answerString = JOptionPane.showInputDialog("What is "
                              + NUMBER1  + " + " + NUMBER2 + "?");
17
18         answer = Integer.parseInt(answerString);
19
20         JOptionPane.showMessageDialog(null, NUMBER1 + " + "
                              + NUMBER2  + " = " + answer + " is "
                              + ((NUMBER1 + NUMBER2) == answer));
21      }//main
22   }//class
```

Program Output

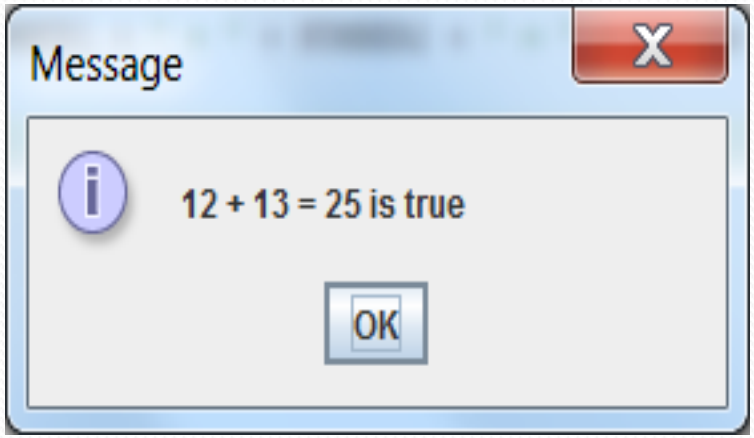
```
answerString = JOptionPane.showInputDialog ("What is " +

                    NUMBER1 +  " + " + NUMBER2 + "?");
```



Program Output

```
JOptionPane.showMessageDialog(null, NUMBER1 + " + " +
                              NUMBER2 + " = " + answer + " is " +
                              ((NUMBER1 + NUMBER2) == answer));
```



Questions

- What type of variable would you use for:
  - an address
  - someone's age
  - the cost of a T-shirt
  - the examination grade 'A', 'B', 'C' or 'D'
- When would you use a constant?
- How do you declare a constant?
- **When (int) is inserted** before a character value, what is returned?
- What is the output from the following section of code?

```
char letter = 'D';
System.out.println((char)((int)letter + 32));
```


53