



Introduction to Programming

17. TextIO and Files

- Some of this is revision: for more details read section 2.4 of the textbook

1



Text Output

- n Java provides a class called *System* that contains a constant static field called *out* which allows data to be displayed on the standard output (on most systems, this is the screen)
 - n `System.out.print(x);`
 - n `System.out.println(x);`
- n *System.out* is a *PrintStream*

2



Output methods

- n The methods `print()` and `println()` in the *PrintStream* class are *overloaded*
 - n `println(int x)`, `println(boolean x)`,
`println(char x)`, `println(double x)`,
`println(String x)`, *etc.*
 - n Similarly for `print()`
- n What is displayed is text
 - n The values are converted to *String* and then displayed

3



`System.out.println()`

- n The overloaded methods called `println()` all write out their parameters to the standard output and then write out a new line
 - n `println()` - print line
- n There is also a version of `println()` which takes no parameters and just writes out a new line

4



Text Input

- n System has another field, called *in*, which can be used for input from the standard input (on most systems this is the keyboard)
- n `System.in` is an *InputStream*
- n Unfortunately, an `InputStream` reads input as a series of bytes, not as text

5



Text Input in Java

- n It is possible to use `System.in` for text input by passing it as a parameter to the constructor for a class that can handle text input
- n Java SE5 simplified text input by providing additional classes but these still need to be passed `System.in` as a parameter
- n Text input remains more complicated than text output in standard Java

6



TextIO

- n For simplicity, we have avoided using the input classes in standard Java directly and instead use the class, `TextIO`
- n As `TextIO` is not built-in to Java we have needed to ensure that it is included in any Java project that we write

7



TextIO continued

- n `TextIO` is an input/output class written by David Eck and used in the course textbook
 - n In addition to using the standard input and output `TextIO` also supports writing to and reading from text files
 - n This is another advantage in using the `TextIO` class
- n All of its methods are static

8



Output in TextIO

- n Output to the screen in TextIO is very similar to that in System.out
 - n There is some additional functionality
- n There are overloaded methods called *TextIO.put()* which are equivalent to System.out.print() and overloaded methods called *TextIO.putln()* which are equivalent to System.out.println()

9



Output in TextIO

- n One overloaded version of TextIO.put() and of TextIO.putln() takes a single parameter and writes the value of its parameter to the current output destination (by default this is the standard output)
- n Another version of each method takes a second parameter, an `int`, which is the minimum number of characters to use for the output (this is different from the print methods of System.out)
 - n Right-justifies the output, padding with leading blanks if the value to be written needs fewer characters than this
- n As with System.out.println() there is also a TextIO.putln() method with no parameters

10



Formatting output

- n System.out also has a method called printf()
 - n There is an equivalent method in TextIO called putf()
- n The method allows you to format output using a *format string*
- n Format strings are described in section 2.4.1 of the textbook

11



Examples

```
int x = 100;
// The quotation marks in the comments below
// are not included in the output
TextIO.put(x);           // prints "100"
TextIO.put(x, 5);        // prints " 100"
TextIO.put(x==100);      // prints "true"
TextIO.put(x==0, 8);     // prints " false "
TextIO.put("£" + 100);   // prints "£100"
```

12



Input in TextIO

- n The TextIO methods that read something from the input source (by default, the keyboard) and return what they have read have names starting with *get* or *getln*:
 - n For primitive type values, method name ends in name of primitive type
 - n `getInt()`, `getByte()`, `getShort()`, `getChar()`, etc.
 - n `getlnInt()`, `getlnShort()`, `getlnChar()`, etc

13



Methods that return Strings

- n There is a method called `getln()` which returns all the characters on a line as a String (so reads a whole line in to a String)
- n There are methods called `getWord()` and `getlnWord()` which return a "word" (a sequence of chars that does not contain spaces, tabs or end of lines)

14



Input methods continued

- n Methods whose name starts with *get* skip whitespace (spaces and tabs) and return the value, retaining any other characters on the current line for the next *get*
 - n Except `getAnyChar()` which does not skip whitespace
- n Methods whose name starts with *getln* do the same but discard the rest of the current line so the next *get* reads from the next line
 - n Except `getln()` does not skip whitespace

15



`get()` versus `getln()` methods

- n The textbook recommends that you use methods starting with `getln()` rather than `get()` in most circumstances where you are reading from the keyboard
- n These assume user will press <Enter> after typing the data to be read
- n If you find when you prompt the user for input that your program is displaying a ? it can often be fixed by replacing a call to `getXxx()` with `getlnXxx()`
 - n Xxx being Int, Char, Boolean etc.

16



Names of input methods

- n Because the output methods take parameters it is possible to overload them
 - n the compiler can work out which method is being called by the type of the actual parameter that the caller supplies
- n The input methods do not take parameters and so must have different names
 - n in Java the return type is not part of the signature

17



Examples

```
int i = TextIO.getInInt();
double d = TextIO.getInDouble();
String line = TextIO.getIn();
String word = TextIO.getInWord();
boolean b = TextIO.getInBoolean();
// for this last method user can type yes
// or no or y or n as well as true or false
```

18



Other input methods

- n There is a method to test whether you are at the end of a line
public static boolean eoln()
- n There is a method to test whether you are at the end of a file
public static boolean eof()
- n There is a method to skip over any blanks
public static void skipBlanks()

19



TextIO.peek()

- n There is a peek() method which returns the next character in the input stream, without reading it
 - n That is, you can peek ahead to see what the next character in the input data is and it will still be there to read when you call one of the *get* or *getIn* methods.

20



Changing the input source

- n By default, TextIO takes its input from the keyboard
- n You can change the input source so that it reads from a file by calling the following method:

```
public static void readFile(String fileName)
```

- n You can change the input source back to the keyboard by calling the following method:

```
public static void readStandardInput()
```

21



Changing the output destination

- n By default, TextIO writes its output to its display console
- n You can change the output destination so that it writes to a file by calling the following method:

```
public static void writeFile(String fileName)
```

- n You can change the output destination back to the console by calling the following method:

```
public static void writeStandardOutput()
```

22



Points to note about files 1

- n `TextIO.writeFile("myfile.txt");`
 - n This creates the file myfile.txt and opens it for writing
 - n Writing starts at the beginning of the file, each piece of data written to the file is appended to the end of the file
 - n If a file with this name already exists it is overwritten!

23



Points to note about files 2

- n `TextIO.readFile("myfile.txt");`
 - n This opens the file, myfile.txt, for reading
 - n As you are reading from the file, it must already exist (readFile does not create a new file)
 - n Reading starts from the beginning of the file
 - n Reading is sequential, you cannot jump directly to a given point in the file but have to read it from the beginning

24



More on files

- n TextIO deals with text files
 - n We will assume that all the files we read from or write to will contain text (sequences of characters)
 - n This module does not deal with file formats that do not contain text and are intended only to be machine-readable
 - n You can create text files using applications such as Notepad (or by using TextIO.writeFile)


25



Demonstration program 1


- n This example is from the textbook (pages 44-45)
 - n Some spelling changes for UK English!
- n The example shows how to use TextIO to create a text file
- n You can open and read the file using an application such as Notepad or using TextIO.readFile()

26




```
public class CreateProfile {
    public static void main(String[] args) {
        String name; // The user's name.
        String email; // The user's email address.
        double salary; // the user's annual salary.
        String favColour; // The user's favourite colour.
        TextIO.putln("Good Afternoon! This program"
            + " will create");
        TextIO.putln("your profile file, if you will "
            + " just answer");
        TextIO.putln("a few simple questions.");
        TextIO.putln();
    }
}
```

27



```
/* Gather responses from the user. */
TextIO.put("What is your name? ");
name = TextIO.getln();
TextIO.put("What is your email address? ");
email = TextIO.getln();
TextIO.put("What is your annual income? ");
salary = TextIO.getlnDouble();
TextIO.put("What is your favourite colour? ");
favColour = TextIO.getln();
```

28




```

/* Write the user's information to the file
named profile.txt. */
TextIO.writeFile("profile.txt"); /* subsequent
output goes to the file */
TextIO.putln("Name: " + name);
TextIO.putln("Email: " + email);
TextIO.putln("Favourite Colour: " + favColour);
TextIO.putf( "Yearly Income: %1.2f\n", salary);
/* The "\n" in the previous line is a newline
character */

```

29



```

/* Print a final message to standard output. */
TextIO.writeStandardOutput();
TextIO.putln("Thank you. Your profile has "
            + "been written to profile.txt.");
    }
}

```


30



Demonstration program 2

- n This is a program that reads the profile created by the previous application
- n The logic used takes advantage of knowledge of the layout of the file, though some simple error checking is made to check that the layout is as expected

31



```

public class ReadProfile {
    public static void main(String[] args) {
        String name = "";
        String email = "";
        double salary = 0.0;
        String favColour = "";
        boolean error = false;
        TextIO.readFile("profile.txt"); /* subsequent
input comes from the file */
    }
}

```

32



```
String word = TextIO.getWord();
if (word.equals("Name:")) {
    name = TextIO.getWord();
} else {
    error = true;
}
TextIO.getln();

word = TextIO.getWord();
if (word.equals("Email:")) {
    email = TextIO.getWord();
} else {
    error = true;
}
TextIO.getln();
```

33



```
word = TextIO.getWord();
if (word.equals("Favourite")) {
    word = TextIO.getWord();
    favColour = TextIO.getWord();
} else {
    error = true;
}
TextIO.getln();
```

34



```
word = TextIO.getWord();
if (word.equals("Yearly")) {
    word = TextIO.getWord();
    salary = TextIO.getlnDouble();
} else {
    error = true;
}
TextIO.readStandardInput();
/* Closes the file, now subsequent
input comes from the keyboard */
```

35



```
if (error) {
    TextIO.putln("Something went wrong!");
} else {
    TextIO.putln("Your name is " + name);
    TextIO.putln("Your email address is "
        + email);
    TextIO.putln("Your favourite colour is "
        + favColour);
    TextIO.printf("Your salary is %,1.2f\n",
        salary);
}
}
```

36



Programs are on Moodle

- n Download these and experiment with reading and writing files and switching between file and standard input/output
- n Note that if you are using Eclipse the program will create the text file in the project folder (the folder above *src* if you are using the default settings)

37



Reading a file line by line

- n To read the whole of a text file line by line use the following idiom:

```
String currentLine = null;
TextIO.readFile(fileName);
while (!TextIO.eof()) {
    currentLine = TextIO.getln();
    ... // code here to process currentLine
}
TextIO.readStandardInput();
```

38



Reading and Writing Files

- n Always remember when you are finished writing to a file to close the file and set the output back to the console with a call to `TextIO.writeStandardOutput()`
- n Always remember when you are finished reading from a file to close the file and set the input back to the keyboard with a call to `TextIO.readStandardInput()`

39



Questions?

40