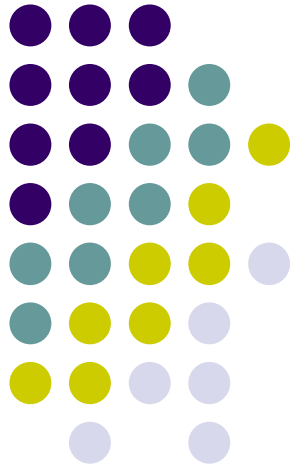
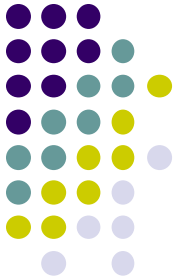
CSCI 1130
Introduction to Computing Using Java

Tutorial 7

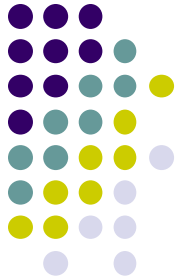


Topics



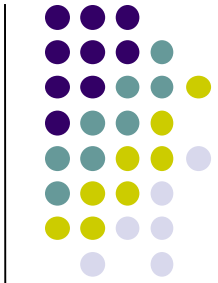
- File Input / Output & Exception handling
  - Basic idea
  - Filedownloader example
- Assignment 4
  - …

Keep Information Non-volatile



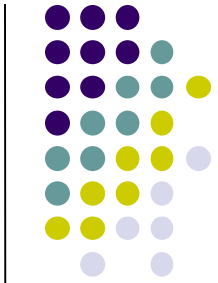
- Modern computers are electronic.
- Without power supply → Data Loss!
- To keep data non-volatile, ***save*** it to a file!

Retrieve Information



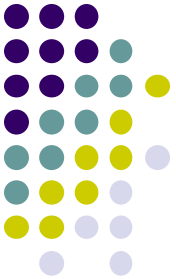
- Data on a file cannot be processed by a computer program directly.
- We have to ***load*** it from the file.
- Moreover, we don’t want to be typists who repeat the same data input every time.

File and Input/Output Operations



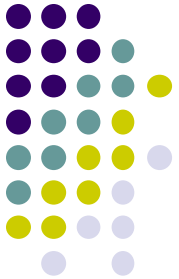
- There are classes to represent a file on disk, a file in network, or input/ output from console.
- They provide methods to make file and IO operations more convenient.
- Class PrintStream and Class Scanner

Class PrintStream



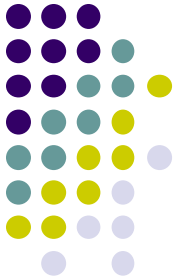
```
import java.io.*;

...main(...) throws IOException
{
  PrintStream myNewFile;

  myNewFile = new PrintStream("myWeb.txt");
  myNewFile.println("Hello World in a file!");

  System.out.println("Hello World on screen.");

  // myNewFile  is a PrintStream object reference
  // System.out is a PrintStream object reference
}
```

Class Scanner



```
import java.util.*;
import java.io.*;

...main(...) throws Exception
{
  Scanner markFile;
  markFile = new Scanner(new File("myWeb.txt"));

  int mark;
  if (markFile.hasNextInt())
    mark = markFile.nextInt();

}
```

Class Scanner

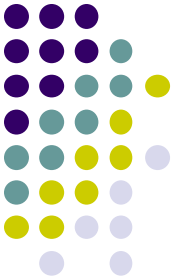


- The methods hasNextInt(), hasNextDouble(), hasNextXyz()… return us a boolean (true/ false) value that indicates if there is more data to read from the Scanner object.

- The methods nextInt(), nextDouble(), … reads a piece of data from the source for us.

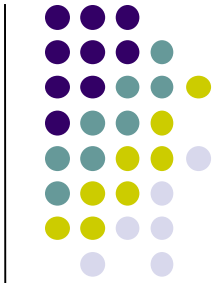
- Operations may fail, thus we add "throws Exception" to the **main()** method.

Class Scanner: line-by-line



```
import java.util.*;
import java.io.*;

...main(...) throws Exception
{
  Scanner markFile;
  markFile = new Scanner(new File("myWeb.txt"));

  String aLine;
  while (markFile.hasNextLine())
  {
    aLine = markFile.nextLine();
    System.out.println(aLine);
  }
}
```

Class Scanner: line-by-line

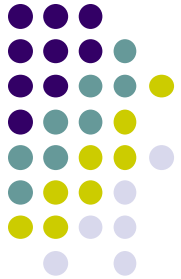


- The method nextLine() reads a line from the source for us. It returns a String.

- The source for the Scanner object could be the keyboard, a file, a web source.

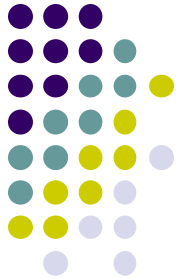
- System.in is a field in class System.
  It refers to a default InputStream object, representing the keyboard.

Class Scanner



- The source could be
  - the keyboard object
    - new Scanner(System.in);
  - a file object
    - new Scanner(new File(“filename”));
  - a web source
    - new Scanner(new URL(“http://...”).openStream( ));

File I / O Example: FileDownloader

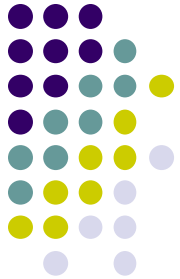


- Create a Java application for downloading plain text file from an URL source on the network.

```
Sample Plain Text File
Line 1 <Enter>
Line 2 Some Text<Enter>
Line 3 …
…
<End-Of-File/ EOF>
```

- The program performs exception handling with proper try-catch statements so that it will not terminate abnormally on an invalid URL or on other error/ unexpected conditions.

File I / O Example: FileDownloader



- Instance method framework

```
public void readDataFromURL ( /* possibly some parameters */ )
{
        // local variable declarations outside try-block

        try {
                // some error-prone I/O operations
        }
        catch (Exception   anExceptionObject) {

                // Exception is the general type to catch
                // it represents all kinds of Exception
                // we can print message from the exception object reference

                JOptionPane.showMessageDialog(null,
                                "Something wrong happened: " + anExceptionObject);

                // some remedial actions, perhaps

        }
}
```

File I/O

Exception handling

File I / O Example: FileDownloader

- Local variable declarations

Ask for URL address

```
String address;
address = (String) JOptionPane.showInputDialog(null, "Type an URL:", "Sample Program
FileDownloader", JOptionPane.QUESTION_MESSAGE, null, null,
"ftp://ftp.cse.cuhk.edu.hk");
```

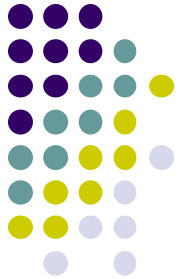
```
URL link = new URL(address);
```

Create URL object

```
String filename = link.getFile();
filename = filename.substring(filename.lastIndexOf('/') + 1);
if (filename == null || filename.trim().isEmpty())
    filename = "download.out";
else
    filename = "download_" + filename;
```

Determine file name

File I / O Example: FileDownloader



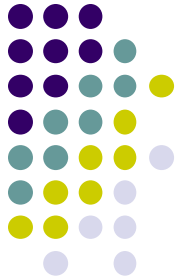
- File input / output

```
try {
    Scanner dataReader = new Scanner(link.openStream());
    PrintStream file = new PrintStream(new File(filename));

    int fileSize = 0;
    while (dataReader.hasNextLine())
    {
        String aLine = dataReader.nextLine();
        file.println(aLine);
        fileSize += aLine.length() + 2;
    }

    JOptionPane.showMessageDialog(null, "Download completed!");
}
```

File I/O object creation

File read and write

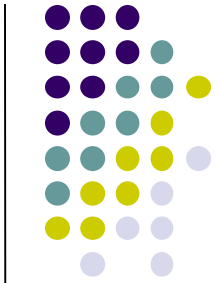
File I / O Example: FileDownloader



- Proxy setting for CSE lab

```
if (…)
{
    // Dept of CS&E proxy settings
    System.getProperties().put("proxySet", "true");
    System.getProperties().put("proxyHost", "proxy.cse.cuhk.edu.hk");
    System.getProperties().put("proxyPort", "8000");
}
```

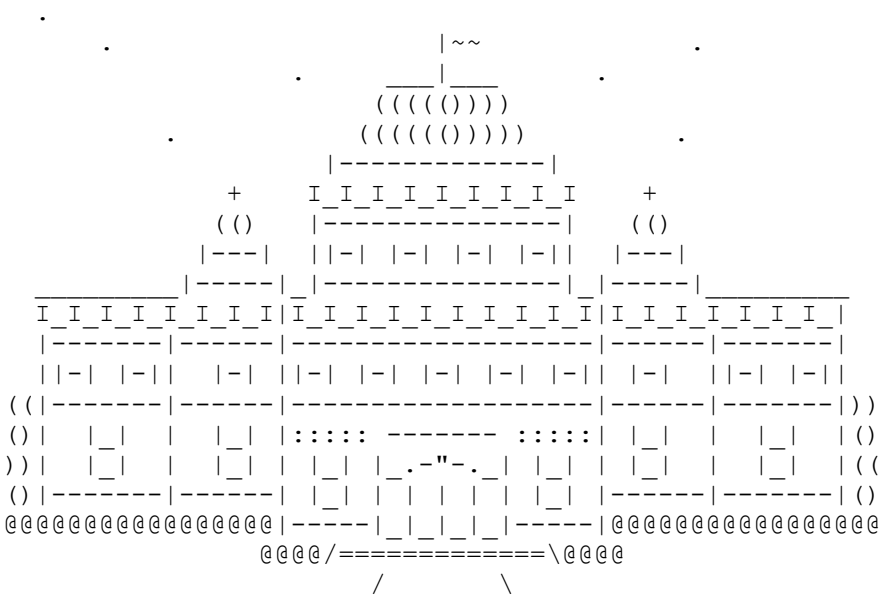
Assignment 4



- Aims
  - To encode pictures of ASCII art using run-length encoding
  - Practice reading/writing text files
  - Practice the use of String and related methods
- Problem Definition
  - In this assignment, you are required to write some classes to encode ASCII arts and decode the compressed ASCII art file.

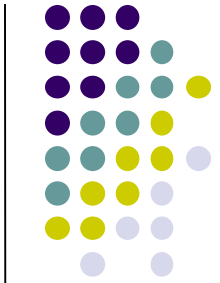
Assignment 4

- Background
  - **ASCII Arts:** A graphic design technique that uses printable ASCII characters to create pictures.



Source: www.asciiworld.com

Assignment 4



- Background
  - **Run-length Encoding:** A sequence of data in which the same data value occurs in many consecutive data elements are stored as a single pair of data value and count instead of the original form. It is quite effective in compressing data that contains frequent repetitions and repeated patterns, like ASCII Arts.

Assignment 4

- Task 1: Encoding a text file containing a picture of ASCII art
  - To encode the file, you need to open and read the original text file line-by-line. Then you need to figure out whether there are any occurrences of consecutive elements in a single line.
  - Syntax of the encoded line:
    - <negative integer> <positive integer> <ASCII character(s)> <negative integer> <positive integer> <ASCII character(s)> <positive integer> <ASCII character(s)> … repeats until the end of line.
    - <positive integer>: the number of times that the succeeding character(s) is repeated in the original line; <negative integer>: the number of times that the succeeding space(s) is repeated in the original line.
    - There is a space between each pair of “<>”s.
  - If the positive integer equals 1, the length of the succeeding ASCII character pattern can be larger than or equal to 1. If the positive integer is greater than 1, the length of the succeeding token should be 1.

Assignment 4

- Task 1 Sample 1

The original line:
@@@@@@|-----|_|_|_|_|-----|@@@@@@

@	@	@	@	@	@	|	-	-	-	-	-	|	_	|	_	|	_	|	_	|	-	-	-	-	-	|	@	@	@	@	@	@

The encoded line:
6 @ 1 | 5 – 1 |_|_|_|_| 5 – 1 | 6 @

6		@		1		|		5		–		1		|	_	|	_	|	_	|	_	|		5		–		1		|		6		@

Note:
1) The positive integer is the number of times that the succeeding character(s) is repeated in the original line;
2) The ASCII character(s) always follows a positive integer;
3) There should be a space between the positive integer and the character pattern.

Assignment 4

- Task 1 Sample 2

The original line:

. ((((()))))

The encoded line:

-10 1 . -11 6 (5)

-	1	0		1		.		-	1	1		6		(		5		)

Note:

1) As spaces are used as the delimiter in the encoded file, we use negative integers to encode spaces in the original file. For N spaces, the integer written is -N.

Assignment 4

- The suggested algorithm for Task 1:
  1. Read a single line of text from the file containing a picture of ASCII art.
  2. Process this line by first determining whether there is a space.
  3. If there are some spaces, count the number of spaces and write a negative integer to the encoded file.
  4. If it is not a space, determine whether there is a repetition of a single character.
  5. If repetition exists, write a positive integer to represent the number of the repeated character. Also write the repeated character to the file.
  6. If there is no repetition, write 1 and extract the character pattern up to a position where a space is present or repetition of a single character starts to occur.
  7. Repeat Step 2 to process the remaining characters in the same line.
  8. The encoding process ends when all lines in the original file are encoded.

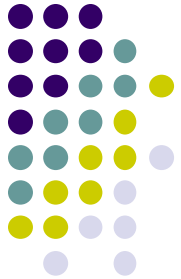
Assignment 4

- Task 2: Decoding the compressed ASCII art file
  - To decode the file, you need to open and read the run-length encoded text file line-by-line.
  - Once a positive number is read, you can write an appropriate number of the repeated characters following this number to the decoded file.
  - Once a negative number is read, you should be able to convert it to the correct number of spaces in the decoded file.
  - The content of the decoded file should be exactly the same as that of the original file.

Assignment 4

- The suggested algorithm for Task 2:
  1. Read a line of characters from the encoded ASCII art file.
  2. Scan this line for an integer.
  3. If it is negative, write a number of spaces in the corresponding line of the decoded file.
  4. If it is positive, scan for a string in the same line. Write an appropriate number of this character pattern in the decoded file.
  5. Repeat Step 2 until the end of line is reached.
  6. The decoding process ends when all lines in the encoded file are read.

Assignment 4



- Task 3: Increasing the efficiency of the encoded file
  - In task 1, we only consider the repetition of a single character in the original line. We can actually look for repeated string patterns to increase the efficiency in terms of storage space.

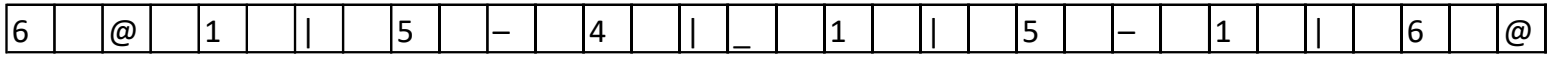
Assignment 4

- Task 3 Sample

The original line:
@@@@@@|-----|_|_|_|_|-----|@@@@@@

The encoded line (Task 1):
6 @ 1 | 5 – 1 |_|_|_|_| 5 – 1 | 6 @

The encoded line (Task 3):
6 @ 1 | 5 – 4 |_ 1 | 5 – 1 | 6 @



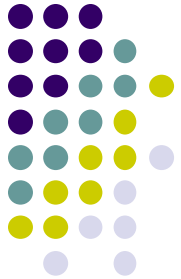
Note:
1) If we make use of the pattern “|_” in the original line, the encoded line becomes shorter than its counterpart in Task 1.
2) The shorter the encoded line is, the better the solution is.

Assignment 4

- Procedure:
  - Create a new project named Assignment4 in folder Assignment4. There are four source files:
    - Assignment4.java (containing Assignment4 class)
    - RunLengthEncoder.java (containing class RunLengthEncoder for Task 1)
    - RunLengthDecoder.java (containing class RunLengthDecoder for Task 2)
    - RunLengthEncoderAdvanced.java (containing class RunLengthEncoderAdvanced for Task 3)
  - In the main method of class Assignment4, you are required to read the original file's name (e.g. testcase1) from standard input by the user as follows:

The original ASCII art picture file: testcase1

  - If the input name is "testcase1" as shown above, the program should read the ASCII art file "testcase1.txt". Then, your program generates the following four files:
    1) "testcase1_e.txt": the encoding results from Tasks 1;
    2) "testcase1_d.txt": the results from decoding 1), it should be exactly same as the original file "testcase1";
    3) "testcase1_ae.txt": the encoding results from Tasks 3;
    4) "testcase1_ad.txt": the results from decoding 3), it should be exactly same as the original file "testcase1".



END