Lecture 3 - I/O and regular Expression

1. Input and Output(I/O)

Input sources include:

Keyboard

File

Network

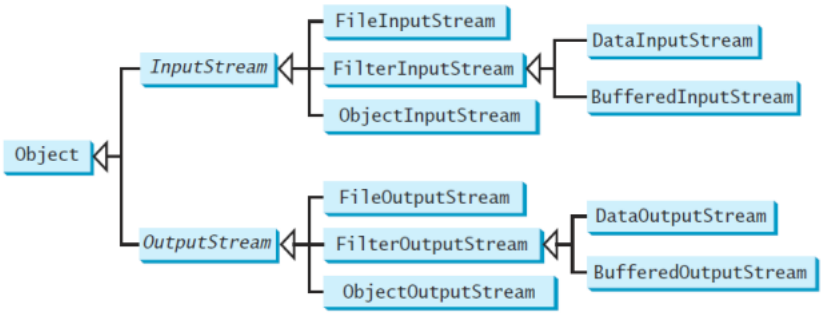
Output Destinations include:

Console

File

Network

Java handles i/o using streams



1.1 Standard I/O

System.in

Object of type InputStream

Typically refers to the keyboard

Reading data could be done using the Scanner class

Methods include:

- String next()
- String nextLine()
- int nextInt()

- double nextDouble()

```
import java.util.Scanner;

class xyz{
  public method(){
    Scanner scan = new  Scanner(System.in);
    sysout("String: ");

      // Reads only uptil a space (delmiter)
    String str = scan.next();
      // reads all but not able to exit through this loop
    while(scan.hasNext())
        String str = scan.next();


    sysout("Int: ");
    String str = scan.nextInt();

    sysout("Double: ");
    String str = scan.nextDouble();

    sysout("Float: ");
    String str = scan.nextFloat();
  }
}
```

System.out

Object of type PrintStream (SUBCLASS OF **filteroutputstream**)

Typically refers to console

```
class AssignmentOperator {
    public static void main(String[] args) {

        System.out.println("Java programming is interesting.");
    }
}
```

Difference between println(), print() and printf()

- `print()` - It prints string inside the quotes.
- `println()` - It prints string inside the quotes similar like `print()` method. Then the cursor moves to the beginning of the next line.
- `printf()` - It provides string formatting (similar to printf in C/C++ programming).

1.1.1 The File class

- Contains Methods for obtaining the properties of a file/directory and for renaming and deleting a file/directory
- Files could be specified using absolute / relative names
- Constructing a **File instance doesnt create a file on the machine**
- Methods include:
  - boolean createNewFile()
  - boolean delete()

- boolean exists()
- File [] listFiles()

```
class xyz{
  void trial(){
    File a = new File("C:/...../text.txt"); // in the directory of project
    f.createNewFile();
    f.delete();
    boolean i= f.exists();
    boolean b= f.isDirectory();
    File f= new File("C: ..... /abc.txt");

    if(f.exists() && f.isDirectory()){
      File [] files = d.listFiles();

      for (File f:files){
        if (!f.isHidden()) // hidden files are not displayed{
          sysout(f.getName()); // names of the files will be displayed
        }
      }
    }
  }
}
```

File I/O

Reading could be done using **Scanner** class

```
Scanner input = new Scanner(new File(filename));
```

Writing could be done using the **FileWriter** class

```
FileWriter output = new FileWriter((String) "filename", (boolean) append);

output.write("hello");
output.write("hello"); // will add hello next to previous hello not in the new line
output.close();
```

Extra

```
package lab3;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class HashtagClass {

    private static void findHashtags(File file, File outputDirectory){
        Pattern pattern = Pattern.compile("    Your REGEX   ");
        int hashtag = 0;


        File outputFile = new File(outputDirectory.getAbsolutePath() + '/' + file.getName().substring(0, file.getName().length()-4) + "_


        try{
          // Writing in file
          FileWriter myWriter = new FileWriter(outputFile.getAbsolutePath());

          // Scanner doesnt work properly in windows
            //!!!!!!Scanner sc = new Scanner(file);

          BufferedReader sc = new BufferedReader(new FileReader(file));
          String line = "";
```

```
          while ((line = sc.readLine()) != null) {
            //Saving all words in line in an array
              String[] words = line.split(" ");
            //Runnin a loop on words
            for (int i =0;  i< words.length; i++){
              Matcher m = pattern.matcher(words[i]);
              if (m.matches()){
                hashtag++;
                // Writing the word matched in the file
                  myWriter.write(words[i] + "\n");
                }
              }
            }

          //closing io
            myWriter.close();
            sc.close();

          System.out.println("File "+ outputFile.getName() +"Written with Total Hashtags: " + String.valueOf(hashtag));
        }

        catch(Exception e){
          System.out.println("Failed to read");
        }
    }


    public static void main(String[] args){
        Scanner reader = new Scanner(System.in);
        System.out.println("Enter the absolute path that conatains the files.");
        String inputPath = reader.nextLine();
        System.out.println("Enter the absolute path that you would like to store the output files.");
        String outputPath = reader.nextLine();


        //Making Directory
          File outputDirectory = new File(outputPath);
          outputDirectory.mkdir();
          reader.close();

        // Opening all files in an array of files
          File path = new File(inputPath);
          File[] files = path.listFiles();


        //Runnin loop on all the files in array
          for (File file: files){
            findHashtags(file, outputDirectory);
          }
        System.out.println("Completed");

    }
}
```

2. Regular Expression

Regular expression (regex) is a string that describes a pattern for matching a set of strings

The **Pattern class** can be used to define pattern

- The **compile** method takes a string representing the regular expression as an argument and compiles it into a pattern.

The **Matcher** class can be used to search for the pattern. its method include:

- boolean find()
- boolean matches()

Example:

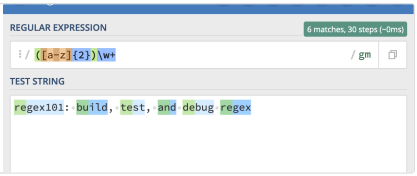
```
Pattern pattern = Pattern.compile(“H.*d");
Matcher matcher = pattern.matcher(“Hello World");
```

```
System.out.printl(matcher.matches());
System.out.printl(matcher.find());          //checks the subset of the string
```

regex101: build, test, and debug regex

Regular expression tester with syntax highlighting, explanation, cheat sheet for PHP/PCRE, Python, GO, JavaScript, Java. Features a regex quiz & library.

https://regex101.com/



Commonly used Regex

Regular Expression	*Matches*	*Example*
.	any single character	Java matches J..a
(ab|cd)	ab or cd	ten matches t(en|im)
[abc]	a, b, or c	Java matches Ja[uvwx]a
[^abc]	any character except a, b, or c	Java matches Ja[^ars]a
[a-z]	a through z	Java matches [A-M]av[a-d]
[^a-z]	any character except a through z	Java matches Jav[^b-d]
[a-e[m-p]]	a through e or m through p	Java matches [A-G[I-M]]av[a-d]

Regular Expression	*Matches*	*Example*
[a-e&&[c-p]]	intersection of a-e with c-p	Java matches [A-P&&[I-M]]av[a-d]
\d	a digit, same as [0-9]	Java2 matches "Java[\\d]"
\D	a non-digit	$Java matches "[\\D][\\D]ava"
\w	a word character	Java1 matches "[\\w]ava[\\w]"
\W	a non-word character	$Java matches "[\\W][\\w]ava"
\s	a whitespace character	"Java 2" matches "Java\\s2"
\S	a non-whitespace char	Java matches "[\\S]ava"

\w mean any letter digit or underscore

Regular Expression	*Matches*	*Example*
*p**	zero or more occurrences of pattern *p*	aaaabb matches "a*bb" ababab matches "(ab)*"
p+	one or more occurrences of pattern *p*	a matches "a+b*" able matches "(ab)+.*"
p?	zero or one occurrence of pattern *p*	Java matches "J?Java" Java matches "J?ava"
p{n}	exactly n occurrences of pattern *p*	Java matches "Ja{1}.*" Java does not match ".{2}"
p{n,}	at least n occurrences of pattern *p*	aaaa matches "a{1,}" a does not match "a{2,}"
p{n,m}	between n and m occurrences (inclusive)	aaaa matches "a{1,9}" abb does not match "a{2,9}bb"

https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html