Contents

1 Basic Test Results

```
1  ******************** OOP pre-submission script for ex5 ********************
2
3  Logins: noa5
4
5
6
7  compiling with
8      javac -cp .:/cs/course/current/oop/lib/junit4.jar  *.java oop/ex5/filescript/*.java
9
10
11 tests error :
12     Executing Test: 001
13 Executing Test: 002
14 Executing Test: 003
15 Executing Test: 004
16 Executing Test: 005
17 Executing Test: 006
18 Executing Test: 007
19 Executing Test: 008
20 Executing Test: 009
21 Executing Test: 010
22 Executing Test: 011
23 Executing Test: 012
24 Executing Test: 013
25 Executing Test: 014
26 Executing Test: 015
27 Executing Test: 016
28 Executing Test: 017
29 Executing Test: 018
30 Executing Test: 019
31 Executing Test: 020
32 Executing Test: 021
33 Executing Test: 022
34 Executing Test: 023
35 Executing Test: 024
36 Executing Test: 025
37 Executing Test: 026
38 Executing Test: 027
39 Executing Test: 028
40 Executing Test: 029
41 Executing Test: 030
42 Executing Test: 031
43 Executing Test: 032
44 Executing Test: 033
45 Executing Test: 034
46 Executing Test: 035
47 Executing Test: 036
48 Executing Test: 037
49 Executing Test: 038
50 Executing Test: 039
51 Executing Test: 040
52 Executing Test: 041
53 Executing Test: 042
54 Executing Test: 043
55 Executing Test: 044
56 Executing Test: 045
57 Executing Test: 046
58 Executing Test: 047
59 Executing Test: 048
```

```
Executing Test: 049
Executing Test: 050
Executing Test: 051
Executing Test: 052
Executing Test: 053
Executing Test: 054
Executing Test: 055
Executing Test: 056
Executing Test: 057
Executing Test: 058
Executing Test: 059
Executing Test: 060

tests output :
        Tests Executed
Perfect!
***** Checking format on QUESTIONS file *****
QUESTIONS file has the right format.
```

2 README

```
noa5

none

============================
=      File description    =
============================

* MyFileScript.java - Represents the manager of the program
* Parser.java - Parsing a command File
* Section.java - Represents a section
* ErrorException.java - Represents an exception that is thrown in case of I/O
  problems in the command file, bad arguments given to the program or in case
  of bad sections' format.
* WarningException.java - Represents an exception that is thrown in case bad
  values, names and parameters are given to ORDER or FILTER.
* FilterFactory.java - Creates a Filter
* OrderFactory.java - Creates an Order
* Filter.java - Represents a FILTER
* NegFilter.java - Represents a negative filter
* AllFilter.java - Represents the filter: All files are matched
* GreaterThanFilter.java - Represents the filter: File size is strictly greater
  than the size value of the filter (in k-bytes)
* SmallerThanFilter.java - Represents the filter: File size is strictly less
  than the size value of the filter (in k-bytes)
* BetweenFilter.java - Represents the filter: File size is between (inclusive)
  the given size values of the filter (in k-bytes)
* PermissionFilter - Represents a filter that refers to a file permissions
* WritableFilter.java - Represents the filter: Does file have writing
  permission?
* ExecutableFilter.java - Represents the filter: Does file have execution
  permission?
* HiddenFilter.java - Represents the filter: Is file a hidden file?
* FileFilter.java - Represents the filter: the file name equals the value of
  the filter
* ContainsFilter.java - Represents the filter: the file name contains the value
  of the filter
* PrefixFilter.java - Represents the filter: the file name starts with the
  value of the filter
* SuffixFilter.java - Represents the filter: the file name ends with the value
  of the filter
* Order.java - Represents an ORDER
* ReversedOrder.java - Represents a reversed order
* AbsOrder.java - Singleton that Represents the order: abs
* SizeOrder.java - Singleton that Represents the order: size
* TypeOrder.java - Singleton that Represents the order: type

***
QUESTIONS
README
***


============================
=          Design          =
============================

My design follows the one showed in Tirgul 9;
* Filter is an interface, and any class (that represents some specific filter)
```

```
60    that implements Filter
61    must contain the method (=behavior) "isFilePassFilter".
62  * Order is also an interface. Moreover, It extends  the interface Comparator,
63    meaning any class
64    (that represents some specific order) that implements Order must contain the
65    method (=behavior) "compare".
66    I chose to implement the orders as singleton, since once we have created an
67    instance of some order, we won't need another one - what interests us is the
68    behavior - the comparation.
69  * Negative filter and reversed order are implemented as decorators, bu using
70    the specif filter/order and return the opposite.
71  * Using factories to create a/n filter/order.
72  * Parser is static, since there is no need to create an instance out of it (we
73    only use its static methods).
74
75  *** Exceptions ***
76  I chose to create two types of exceptions: WarningException and Error exception,
77  since there are two types of errors we should have dealt with.
78  WarningException deals with type-1 errors, and ErrorException deals with type-2
79  errors.
80  WarningExceptions are thrown from some of the filters, FilterFactory and
81  OrderFactory, and are caught only in Parser, in order to recognize in which
82  line the given command file contained bad values/parameters.
83  ErrorExceptions are thrown from the Parser (since they represent bad format of
84  command file) and from MyFileScript (in case of a problem with the given
85  arguments to the program), and are caught only in MyFileScript, because in case
86  of type-2 error, we want the program to stop (catching ErrorException in a
87  deeper level would prevent it to get to the most external level, thus the
88  program will continue).
89
90  *** Sorting, chosen data stracture and the reason for this choice***
91  In order to sort the files that have passed the given filters in the command
92  file, I implemented the Order class with extension to Comparator - as I said
93  above,each specific order class has its own "compare" method.
94  Because we can sort arrays in java by a given comparator, I put all of the
95  files that have passed some filter in an ArrayList - so not only there is no
96  need to determine a size at the begining (we have the method 'add' in
97  ArrayList), we can sort this data stracture by the current given order, which
98  is actually a comparator (meaning we sort by the "compare" method that was
99  defined in each specific order).
```

3 QUESTIONS

```
#Week 8 Questions
#================

# Choose the correct statement about the Factory design pattern
#1: Every Factory is a class that implements the Factory interface.
#2: All the objects that a factory returns are always the same object (all pointers
#   point to the same object).
#3: The reference returned by a Factory may be a static data member of the factory.
#4: If a Factory can produce some object, then it is the only way to produce that kind of object.

w8Answer1 = 3
# optional explanation.

#Choose the incorrect statement:
#1: The source code of a module designed according to the open-closed principle should not be
#   changed (or changed as little as possible).
#2: In a design that satisfies modular decomposability each module is independent of the others
#   and therefore can serve other purposes if combined with other modules.
#3: A Factory may be a class, an interface, or a method.
#4: The constructors of a Singleton class must not be public.

w8Answer2 = 3
# optional explanation.

#Choose all correct statements:
#1: if a software is composed of modules that can be used in other programs
#   as well, then we can say that this softwares design satisfies Modular Composability.
#2: The polymorphism mechanism supports the Open-Closed principle.
#3: If we want to create a sorting strategy, a good solution would be to create a class called
#   SimpleSort and the classes representing the different sorting strategies should extend this class.
#4: The Understandability principle says the code has to be readable (well documented, descriptive
#   variable names, etc').
#5: The open-closed principle says that a module should be open for modification but closed to extension.
#6: A class that all its fields and methods are static can also be called a Singleton.

w8Answer3 = (2,3)
# The answer should be a group of the chosen numbers, i.e. (1,7) or (2,6,1), etc'.
# optional explanation.


#Week 9 Questions
#============

# When reading bytes from a large file, we should use a buffered reader because:
#1: In Java, files with size greater than a certain size cannot be read without using a buffer.
#2: If we don't use a buffer, the system memory becomes overloaded.
#3: Using a buffered reader is the only way to read files in java, and it holds for non-large files as well.
#4:  buffered reading takes advantage of properties of the OS and the hardware to speed up the process

w9Answer1 = 4
# optional explanation.

#If we want to be able to read compressed files and also use a buffer to read them, a good solution
#would include:
#1: Creating a class that extends CompressedFileInputStream and add a buffered reading capability to it.
#2: Creating a class that extends BufferedFileInputStream and add a decompression capability to it.
#3: Creating a class that extends InputStream that reads to a buffer, and another class that
#   extends InputStream that uses decompression.
#4: Creating a class that extends InputStream and add buffered reading and a decompression capability to it.
```

```
60
61  w9Answer2 = 3
62  # optional explanation.
63
64  # If A decorates B then A also composes B.
65  #1: Correct
66  #2: Incorrect
67
68  w9Answer3 = 1
69  # optional explanation.
70
71  #Which of the following is not true?
72  #1: Scanner can receive a BufferedInputStream as a parameter.
73  #2: A BufferedInputStream can receive a Scanner as a parameter.
74  #3: BufferedInputStream's constructor can receive another BufferedInputStream as a parameter.
75  #4: In Java, when we read a files content, or we write content to a file, an I/O exception might occur.
76
77  w9Answer4 = 2
78  # optional explanation.
```

4 oop/ex5/filescript/AbsOrder.java

```java
package oop.ex5.filescript;

import java.io.File;

/**
 * Singleton that Represents the order: abs
 * @author noa5
 */
public class AbsOrder implements Order{

    private static AbsOrder instance = null;

    // Exists only to defeat instantiation.
    private AbsOrder() {

    }

    /**
     * @return the only instance of AbsOrder
     */
    public static AbsOrder getInstance() {
          if (instance == null) {
             instance = new AbsOrder();
          }
          return instance;
       }

    /**
     * @param o1 the first object to be compared.
     * @param o2 the second object to be compared.
     * @return a negative number, zero, or
     * a positive number as the first argument is less than, equal to, or
     * greater than the second.
     */
    public int compare(File o1, File o2) {
        return o1.getAbsolutePath().compareTo(o2.getAbsolutePath());
    }

}
```

5 oop/ex5/filescript/AllFilter.java

```java
package oop.ex5.filescript;

import java.io.File;

/**
 * Represents the filter: All files are matched
 * @author noa5
 */
public class AllFilter implements Filter{

    /**
     * Checks if a given file passes the filter
     * @param file the file we want to check
     * @return true if the file passes the filter, false otherwise
     */
    public boolean isFilePassFilter(File file) {
        return true;
    }

}
```

6 oop/ex5/filescript/BetweenFilter.java

```java
package oop.ex5.filescript;

import java.io.File;

/**
 * Represents the filter: File size is between (inclusive) the given size values
 * of the filter (in k-bytes)
 * @author noa5
 */
public class BetweenFilter implements Filter{

    private String[] filterName;
    private double lowerValue;
    private double upperValue;
    private int multToKB = 1024;

    /**
     * Constructor of the filter
     * @param name the name of the filter
     * @throws WarningException
     */
    public BetweenFilter(String[] name) throws WarningException {
        this.filterName = name;
        lowerValue = Double.parseDouble(filterName[filterName.length-2]);
        upperValue = Double.parseDouble(filterName[filterName.length-1]);
        if (lowerValue < 0 || upperValue < 0 || lowerValue >= upperValue) {
            throw new WarningException();
        }
    }

    /**
     * Checks if a given file passes the filter - meaning, if is between the
     * size values of the filter
     * @param file the file we want to check
     * @return true if the file passes the filter, false otherwise
     */
    public boolean isFilePassFilter(File file) {
        long fileLength = file.length()/multToKB;
        if (lowerValue <= fileLength && fileLength <= upperValue) {
            return true;
        }
        return false;
    }

}
```

7 oop/ex5/filescript/ContainsFilter.java

```java
package oop.ex5.filescript;

import java.io.File;

/**
 * Represents the filter: the file name contains the value of the filter
 * @author noa5
 */
public class ContainsFilter implements Filter{

    private String[] filterName;

    /**
     * Constructor of the filter
     * @param name the name of the filter
     */
    public ContainsFilter(String[] name) {
        this.filterName = name;
    }

    /**
     * Checks if a given file passes the filter - meaning, if the file name
     * contains the value of the filter
     * @param file the file we want to check
     * @return true if the file passes the filter, false otherwise
     */
    public boolean isFilePassFilter(File file) {
        if (this.filterName[this.filterName.length-1].indexOf(file.getName())
                != -1) {
            return true;
        }
        return false;
    }

}
```

8 oop/ex5/filescript/ErrorException.java

```java
package oop.ex5.filescript;

/**
 * Represents an exception that is thrown in case of I/O problems in the
 * command file, bad arguments given to the program or in case of bad sections'
 * format.
 * @author noa5
 */
public class ErrorException extends Exception {

    private static final long serialVersionUID = 1L;

}
```

9 oop/ex5/filescript/ExecutableFilter.java

```java
package oop.ex5.filescript;

import java.io.File;

/**
 * Represents the filter: Does file have execution permission?
 * (for the current user)
 * @author noa5
 */
public class ExecutableFilter extends PermissionFilter{

    /**
     * Constructor of the filter
     * @param name the name of the filter
     * @throws WarningException
     */
    public ExecutableFilter(String[] name) throws WarningException {
        super(name);
    }

    /**
     * Checks if a given file passes the filter - meaning, if has the wanted
     * permission for executing.
     * @param file the file we want to check
     * @return true if the file passes the filter, false otherwise
     */
    public boolean isFilePassFilter(File file) {
        String isExecutable = this.filterName[this.filterNameLength-1];
        if (isExecutable.equals("YES") && file.canExecute() ||
                isExecutable.equals("NO") && !file.canExecute()) {
            return true;
        }
        return false;
    }

}
```

10 oop/ex5/filescript/FileFilter.java

```java
package oop.ex5.filescript;

import java.io.File;

/**
 * Represents the filter: the file name equals the value of the filter
 * @author noa5
 */
public class FileFilter implements Filter{

    private String[] filterName;

    /**
     * Constructor of the filter
     * @param name the name of the filter
     */
    public FileFilter(String[] name) {
        this.filterName = name;
    }

    /**
     * Checks if a given file passes the filter - meaning, if the file name
     * equals the value of the filter
     * @param file the file we want to check
     * @return true if the file passes the filter, false otherwise
     */
    public boolean isFilePassFilter(File file) {
        if (this.filterName[this.filterName.length-1].equals(file.getName())) {
            return true;
        }
        return false;
    }

}
```

11 oop/ex5/filescript/Filter.java

```java
package oop.ex5.filescript;

import java.io.*;

/**
 * Represents a FILTER
 * @author noa5
 */
public interface Filter {

    /**
     * Checks if a given file passes the filter
     * @param file the file we want to check
     * @return true if the file passes the filter, false otherwise
     */
    public boolean isFilePassFilter(File file);
}
```

12 oop/ex5/filescript/FilterFactory.java

```java
package oop.ex5.filescript;

import java.util.Arrays;


/**
 * Creates a Filter
 * @author noa5
 */
public class FilterFactory {

    private final static String SEPERATOR = "#";
    private final static int INDEX_OF_FILTER_NAME = 0;
    private final static String NEG_FILTER_SUFFIX = "NOT";

    private final static String GREATER_THAN_FILTER = "greater_than";
    private final static String BETWEEN_FILTER = "between";
    private final static String SMALLER_THAN_FILTER = "smaller_than";
    private final static String FILE_FILTER = "file";
    private final static String CONTAINS_FILTER = "contains";
    private final static String PREFIX_FILTER = "prefix";
    private final static String SUFFIX_FILTER = "suffix";
    private final static String WRITABLE_FILTER = "writable";
    private final static String EXECUTABLE_FILTER = "executable";
    private final static String HIDDEN_FITLER = "hidden";
    private final static String ALL_FILTER = "all";

    /**
     * @return the default filter
     */
    public static Filter getDefaultFilter() {
        return new AllFilter();
    }

    /**
     * @param filterName a name of some filter
     * @return a specific filter
     * @throws WarningException
     */
    public static Filter createFilter(String filterName) throws WarningException {
        String[] filterNameArray = filterName.split(SEPERATOR);
        boolean isNegFilter = false;
        if (filterNameArray[filterNameArray.length-1].equals(NEG_FILTER_SUFFIX)) {
            isNegFilter = true;
            filterNameArray = Arrays.copyOf(filterNameArray,
                    filterNameArray.length-1);
        }
        Filter retFilter = null;

        if (filterNameArray[INDEX_OF_FILTER_NAME].equals(GREATER_THAN_FILTER)) {
            retFilter = new GreaterThanFilter(filterNameArray);
        } else if (filterNameArray[INDEX_OF_FILTER_NAME].equals(BETWEEN_FILTER)) {
            retFilter = new BetweenFilter(filterNameArray);
        } else if (filterNameArray[INDEX_OF_FILTER_NAME].equals(SMALLER_THAN_FILTER)) {
            retFilter = new SmallerThanFilter(filterNameArray);
        } else if (filterNameArray[INDEX_OF_FILTER_NAME].equals(FILE_FILTER)) {
            retFilter = new FileFilter(filterNameArray);
        } else if (filterNameArray[INDEX_OF_FILTER_NAME].equals(CONTAINS_FILTER)) {
            retFilter = new ContainsFilter(filterNameArray);
```

```
60          } else if (filterNameArray[INDEX_OF_FILTER_NAME].equals(PREFIX_FILTER)) {
61              retFilter = new PrefixFilter(filterNameArray);
62          } else if (filterNameArray[INDEX_OF_FILTER_NAME].equals(SUFFIX_FILTER)) {
63              retFilter = new SuffixFilter(filterNameArray);
64          } else if (filterNameArray[INDEX_OF_FILTER_NAME].equals(WRITABLE_FILTER)) {
65              retFilter = new WritableFilter(filterNameArray);
66          } else if (filterNameArray[INDEX_OF_FILTER_NAME].equals(EXECUTABLE_FILTER)) {
67              retFilter = new ExecutableFilter(filterNameArray);
68          } else if (filterNameArray[INDEX_OF_FILTER_NAME].equals(HIDDEN_FITLER)) {
69              retFilter = new HiddenFilter(filterNameArray);
70          } else if (filterNameArray[INDEX_OF_FILTER_NAME].equals(ALL_FILTER)) {
71              retFilter = new AllFilter();
72          } else {
73              throw new WarningException();
74          }
75
76          if (isNegFilter) {
77              retFilter = new NegFilter(retFilter);
78          }
79          return retFilter;
80
81      }
82
83  }
```

13 oop/ex5/filescript/GreaterThanFilter.java

```java
package oop.ex5.filescript;

import java.io.File;

/**
 * Represents the filter: File size is strictly greater than the size value of
 * the filter (in k-bytes)
 * @author noa5
 */
public class GreaterThanFilter implements Filter{

    private String[] filterName;
    private double comparingValue;
    private int multToKB = 1024;

    /**
     * Constructor of the filter
     * @param name the name of the filter
     * @throws WarningException
     */
    public GreaterThanFilter(String[] name) throws WarningException {
        this.filterName = name;
        this.comparingValue = Double.parseDouble(filterName[filterName.length-1]);
        if (comparingValue < 0) {
            throw new WarningException();
        }
    }

    /**
     * Checks if a given file passes the filter - meaning, if is greater than
     * the size value of the filter
     * @param file the file we want to check
     * @return true if the file passes the filter, false otherwise
     */
    public boolean isFilePassFilter(File file) {
        if (comparingValue < file.length()/multToKB) {
            return true;
        }
        return false;
    }

}
```

14 oop/ex5/filescript/HiddenFilter.java

```java
package oop.ex5.filescript;

import java.io.File;

/**
 * Represents the filter: Is file a hidden file?
 * @author noa5
 */
public class HiddenFilter extends PermissionFilter{

    /**
     * Constructor of the filter
     * @param name the name of the filter
     * @throws WarningException
     */
    public HiddenFilter(String[] name) throws WarningException {
        super(name);

    }

    /**
     * Checks if a given file passes the filter - meaning, if has the wanted
     * value for 'isHidden'.
     * @param file the file we want to check
     * @return true if the file passes the filter, false otherwise
     */
    public boolean isFilePassFilter(File file) {
        String isHidden = this.filterName[this.filterNameLength-1];
        if (isHidden.equals("YES") && file.isHidden() ||
                isHidden.equals("NO") && !file.isHidden()) {
            return true;
        }
        return false;
    }

}
```

15 oop/ex5/filescript/MyFileScript.java

```java
package oop.ex5.filescript;

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

/**
 * Represents the manager of the program
 * @author noa5
 */
public class MyFileScript {

    private static final int NUM_ARGUMENTS = 2;
    private static final int INDEX_OF_DIR_PATH = 0;
    private static final int INDEX_OF_COMMAND_FILE = 1;

    /**
     * @param args the given arguments, when the first one is a directory,
     * and the second is a file
     * @throws ErrorException
     * @throws FileNotFoundException
     * @throws WarningException
     */
    public static void main(String[] args) throws ErrorException,
            FileNotFoundException, WarningException {
        try {
            if (args.length != NUM_ARGUMENTS) {
                throw new ErrorException();
            }

            File dirPath = new File(args[INDEX_OF_DIR_PATH]);

            ArrayList<Section> sections = Parser.parseCommandFile(args[INDEX_OF_COMMAND_FILE]);

            for (Section curSection : sections) {
                ArrayList<File> filteredFiles = new ArrayList<File>();

                if (curSection.hasWarning()) {
                    System.out.println(curSection.getWarningMsg());
                }
                for (File curFile : dirPath.listFiles()) {
                    if (curFile.isFile() && curSection.getFilter().isFilePassFilter(curFile)) {
                        filteredFiles.add(curFile);
                    }
                }
                filteredFiles.sort(curSection.getOrder());

                for (File curFilteredFile : filteredFiles) {
                    System.out.println(curFilteredFile.getName());
                }
            }
        } catch (ErrorException e) {
            System.err.println("ERROR");
        }

    }

}
```

16 oop/ex5/filescript/NegFilter.java

```java
package oop.ex5.filescript;

import java.io.*;

/**
 * Represents a negative filter
 * @author noa5
 *
 */
public class NegFilter implements Filter{

    private Filter filter = null;

    /**
     * Constructs a new negative filter
     * @param filter
     */
    public NegFilter(Filter filter) {
        this.filter = filter;
    }

    /**
     * Checks if a given file passes the filter
     * @param file the file we want to check
     * @return true if the file does not pass the filter, false otherwise
     */
    public boolean isFilePassFilter(File file) {
        return !this.filter.isFilePassFilter(file);
    }

}
```

17 oop/ex5/filescript/Order.java

```java
package oop.ex5.filescript;

import java.io.*;
import java.util.Comparator;

/**
 * Represents an ORDER
 * @author noa5
 */
public interface Order extends Comparator<File>{

    final static int FIRST_OBJECT_IS_BIGGER = 1;
    final static int SECOND_OBJECT_IS_BIGGER = -1;
    final static int OBJECTS_ARE_EQUAL = 0;

    /**
     * @param o1 the first object to be compared.
     * @param o2 the second object to be compared.
     * @return 'SECOND_OBJECT_IS_BIGGER', 'OBJECTS_ARE_EQUAL', or
     * 'FIRST_OBJECT_IS_BIGGER' as the first argument is less than, equal to, or
     * greater than the second.
     */
    public int compare(File o1, File o2);

}
```

18 oop/ex5/filescript/OrderFactory.java

```java
package oop.ex5.filescript;

/**
 * Creates an Order
 * @author noa5
 */
public class OrderFactory {

    private final static String SEPERATOR = "#";
    private final static int INDEX_OF_ORDER_NAME = 0;
    private final static String REVERSE_ORDER_SUFFIX = "REVERSE";

    private final static String ABS_ORDER = "abs";
    private final static String TYPE_ORDER = "type";
    private final static String SIZE_ORDER = "size";

    /**
     * @return the default order
     */
    public static Order getDefaultOrder() {
        return AbsOrder.getInstance();
    }

    /**
     * @param orderName a name of some order
     * @return a specific order
     * @throws WarningException
     */
    public static Order createOrder (String orderName) throws WarningException {
        String[] orderNameArray = orderName.split(SEPERATOR);

        Order retOrder = null;

        if (orderNameArray[INDEX_OF_ORDER_NAME].equals(ABS_ORDER)) {
            retOrder = AbsOrder.getInstance();
        } else if (orderNameArray[INDEX_OF_ORDER_NAME].equals(TYPE_ORDER)) {
            retOrder = TypeOrder.getInstance();
        } else if (orderNameArray[INDEX_OF_ORDER_NAME].equals(SIZE_ORDER)) {
            retOrder = SizeOrder.getInstance();
        } else {
            throw new WarningException();
        }

        if (orderNameArray[orderNameArray.length-1].equals(REVERSE_ORDER_SUFFIX)) {
            retOrder = new ReversedOrder(retOrder);
        }

        return retOrder;

    }

}
```

19 oop/ex5/filescript/Parser.java

```java
package oop.ex5.filescript;

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

/**
 * Parsing a command File
 * @author noa5
 */
public class Parser {

    private static final String FILTER_TITLE = "FILTER";
    private static final String ORDER_TITLE = "ORDER";

    // Checks whether a line should be null if is, and returns the line
    private static String safeReadLine(BufferedReader reader, boolean expectNull, int lineNumber) throws IOException, ErrorE
        String line = reader.readLine();
        if (line == null && !expectNull) {
            throw new ErrorException();
        }
        return line;
    }

    // Checks if a given line is an expected title
    private static boolean isValidTitle(String line, String expectedTitle) {
        return line.equals(expectedTitle);
    }

/**
 * Creates Sections out of the file that is being parsed
 * @return a queue of type Section
 * @throws FileNotFoundException
 * @throws ErrorException
 */
    public static ArrayList<Section> parseCommandFile(String commandFile)
            throws ErrorException {
        int lineNumber = 0;
        ArrayList<Section> sections = new ArrayList<Section>();
        Filter curFilter = null;
        Order curOrder = null;
        try (BufferedReader reader = new BufferedReader(new FileReader(commandFile))){
            String line = Parser.safeReadLine(reader, false, lineNumber);
            lineNumber++;
            while (line != null) {

                Section curSection = new Section();

                if (!isValidTitle(line, FILTER_TITLE)) {
                    throw new ErrorException();
                }
                line = Parser.safeReadLine(reader, false, lineNumber);
                lineNumber++;
                try {
                    curFilter = FilterFactory.createFilter(line);
                }
                catch (WarningException e) {
                    curSection.addWarningMsg(lineNumber);
                    curFilter = FilterFactory.getDefaultFilter();
                }
```

```
60
61                  line = Parser.safeReadLine(reader, false, lineNumber);
62                  lineNumber++;
63                  if (!isValidTitle(line, ORDER_TITLE)) {
64                      throw new ErrorException();
65                  }
66
67                  line = Parser.safeReadLine(reader, true, lineNumber);
68                  lineNumber++;
69                  if (line == null || line.equals(FILTER_TITLE)) {
70                      curSection.setFilter(curFilter);
71                      curSection.setOrder(OrderFactory.getDefaultOrder());
72                      sections.add(curSection);
73                  } else {
74                      try {
75                          curOrder = OrderFactory.createOrder(line);
76                      }
77                      catch (WarningException e) {
78                          curSection.addWarningMsg(lineNumber);
79                          curOrder = OrderFactory.getDefaultOrder();
80                      }
81
82                      curSection.setFilter(curFilter);
83                      curSection.setOrder(curOrder);
84                      sections.add(curSection);
85
86                      line = Parser.safeReadLine(reader, true, lineNumber);
87                      lineNumber++;
88                  }
89
90
91              }
92
93          } catch (IOException e) {
94              throw new ErrorException();
95          }
96          return sections;
97      }
98
99
100
101 }
```

20 oop/ex5/filescript/PermissionFilter.java

```java
package oop.ex5.filescript;

import java.io.File;

/**
 * Represents a filter that refers to a file permissions
 * @author noa5
 */
public abstract class PermissionFilter implements Filter{

    protected String[] filterName;
    protected int filterNameLength;

    /**
     * Constructor of the filter
     * @param name the name of the filter
     * @throws WarningException
     */
    public PermissionFilter(String[] name) throws WarningException {
        this.filterName = name;
        this.filterNameLength = this.filterName.length;
        if (!this.filterName[this.filterNameLength-1].equals("NO") &&
                !this.filterName[this.filterNameLength-1].equals("YES")) {
            throw new WarningException();
        }

    }

    /**
     * Checks if a given file passes the filter
     * @param file the file we want to check
     * @return true if the file passes the filter, false otherwise
     */
    public abstract boolean isFilePassFilter(File file);


}
```

21 oop/ex5/filescript/PrefixFilter.java

```java
package oop.ex5.filescript;

import java.io.File;

/**
 * Represents the filter: the file name starts with the value of the filter
 * @author noa5
 */
public class PrefixFilter implements Filter{

    private String[] filterName;

    /**
     * Constructor of the filter
     * @param name the name of the filter
     */
    public PrefixFilter(String[] name) {
        this.filterName = name;
    }

    /**
     * Checks if a given file passes the filter - meaning, if the file name
     * starts with the value of the filter
     * @param file the file we want to check
     * @return true if the file passes the filter, false otherwise
     */
    public boolean isFilePassFilter(File file) {
        if (file.getName().startsWith(this.filterName[this.filterName.length-1])) {
            return true;
        }
        return false;
    }

}
```

22 oop/ex5/filescript/ReversedOrder.java

```java
package oop.ex5.filescript;

import java.io.*;

/**
 * Represents a reversed order
 * @author noa5
 */
public class ReversedOrder implements Order{

    private Order order = null;

    /**
     * Constructs a new reversed order
     * @param filter
     */
    public ReversedOrder(Order order) {
        this.order = order;
    }

    /**
     * @param o1 the first object to be compared.
     * @param o2 the second object to be compared.
     * @return a reversed order - meaning: 'FIRST_OBJECT_IS_BIGGER',
     * 'OBJECTS_ARE_EQUAL', or 'SECOND_OBJECT_IS_BIGGER' as the first argument is
     * less than, equal to, or greater than the second (the opposite than a 'regular' order).
     */
    public int compare(File o1, File o2) {
        return this.order.compare(o2, o1);
    }
}
```

23 oop/ex5/filescript/Section.java

```java
package oop.ex5.filescript;

/**
 * Represents a section
 * @author noa5
 */
public class Section {

    private final static String WARNING_MESSAGE = "Warning in line ";

    private Filter filter = null;
    private Order order = null;
    private String warningMsg = null;

    /**
     * Sets the filter
     */
    public void setFilter(Filter filter) {
        this.filter = filter;
    }

    /**
     * Sets the order
     */
    public void setOrder(Order order) {
        this.order = order;
    }

    /**
     * @return the filter of the section
     */
    public Filter getFilter() {
        return this.filter;
    }

    /**
     * @return the order of the section
     */
    public Order getOrder() {
        return this.order;
    }

    /**
     * @return true if the warning message of the section is not empty (meaning
     * if there was an error of type 1 in the section), false otherwise.
     */
    public boolean hasWarning () {
        return this.warningMsg != null;
    }

    /**
     * Updates the warning message of the section
     * @param lineWithError the line in which the error occurred
     */
    public void addWarningMsg(int lineWithError) {
        if (this.warningMsg == null) {
            this.warningMsg = WARNING_MESSAGE + Integer.toString(lineWithError);
        } else {
            this.warningMsg += "\n" + WARNING_MESSAGE +
```

```
60                      Integer.toString(lineWithError);
61          }
62      }
63
64      /**
65       * @return the warning message of the section
66       */
67      public String getWarningMsg() {
68          return this.warningMsg;
69      }
70
71  }
```

24 oop/ex5/filescript/SizeOrder.java

```java
package oop.ex5.filescript;

import java.io.File;

/**
 * Singleton that Represents the order: size
 * @author noa5
 */
public class SizeOrder implements Order {

    private static SizeOrder instance = null;

    // Exists only to defeat instantiation.
    protected SizeOrder() {

    }

    /**
     * @return the only instance of AbsOrder
     */
    public static SizeOrder getInstance() {
          if (instance == null) {
             instance = new SizeOrder();
          }
          return instance;
       }

    /**
     * @param o1 the first object to be compared.
     * @param o2 the second object to be compared.
     * @return 'SECOND_OBJECT_IS_BIGGER', 'OBJECTS_ARE_EQUAL', or
     * 'FIRST_OBJECT_IS_BIGGER' as the first argument is less than, equal to, or
     * greater than the second.
     */
    public int compare(File o1, File o2) {
        double o1Size = new Double(o1.length());
        double o2Size = new Double(o2.length());
        if (o1Size > o2Size) {
            return FIRST_OBJECT_IS_BIGGER;
        }
        if (o1Size < o2Size) {
            return SECOND_OBJECT_IS_BIGGER;
        }
        return AbsOrder.getInstance().compare(o1, o2);

    }

}
```

25 oop/ex5/filescript/SmallerThanFilter.java

```java
package oop.ex5.filescript;

import java.io.File;

/**
 * Represents the filter: File size is strictly less than the size value of the
 * filter (in k-bytes)
 * @author noa5
 */
public class SmallerThanFilter implements Filter{

    private String[] filterName;
    private double comparingValue;
    private int multToKB = 1024;

    /**
     * Constructor of the filter
     * @param name the name of the filter
     * @throws WarningException
     */
    public SmallerThanFilter(String[] name) throws WarningException {
        this.filterName = name;
        this.comparingValue = Double.parseDouble(filterName[filterName.length-1]);
        if (comparingValue < 0) {
            throw new WarningException();
        }
    }

    /**
     * Checks if a given file passes the filter - meaning, if is smaller than
     * the size value of the filter
     * @param file the file we want to check
     * @return true if the file passes the filter, false otherwise
     */
    public boolean isFilePassFilter(File file) {
        if (comparingValue > file.length()/multToKB) {
            return true;
        }
        return false;
    }

}
```

26 oop/ex5/filescript/SuffixFilter.java

```java
package oop.ex5.filescript;

import java.io.File;

/**
 * Represents the filter: the file name ends with the value of the filter
 * @author noa5
 */
public class SuffixFilter implements Filter{

    private String[] filterName;

    /**
     * Constructor of the filter
     * @param name the name of the filter
     */
    public SuffixFilter(String[] name) {
        this.filterName = name;
    }

    /**
     * Checks if a given file passes the filter - meaning, if the file name
     * ends with the value of the filter
     * @param file the file we want to check
     * @return true if the file passes the filter, false otherwise
     */
    public boolean isFilePassFilter(File file) {
        if (file.getName().endsWith(this.filterName[this.filterName.length-1])) {
            return true;
        }
        return false;
    }

}
```

27 oop/ex5/filescript/TypeOrder.java

```java
package oop.ex5.filescript;

import java.io.File;

/**
 * Singleton that Represents the order: type
 * @author noa5
 */
public class TypeOrder implements Order{

    private static final String SEPERATOR = ".";
    private static TypeOrder instance = null;

    // Exists only to defeat instantiation.
    protected TypeOrder() {

    }

    /**
     * @return the only instance of AbsOrder
     */
    public static TypeOrder getInstance() {
          if (instance == null) {
             instance = new TypeOrder();
          }
          return instance;
       }

    /**
     * @param o1 the first object to be compared.
     * @param o2 the second object to be compared.
     * @return 'SECOND_OBJECT_IS_BIGGER', 'OBJECTS_ARE_EQUAL', or
     * 'FIRST_OBJECT_IS_BIGGER' as the first argument is less than, equal to, or
     * greater than the second.
     */
    public int compare(File o1, File o2) {
        String o1Type = o1.getName().substring(
                o1.getName().lastIndexOf(SEPERATOR) + 1);
        String o2Type = o2.getName().substring(
                o2.getName().lastIndexOf(SEPERATOR) + 1);
        if (o1Type.compareTo(o2Type) > 0) {
            return FIRST_OBJECT_IS_BIGGER;
        }
        if (o1Type.compareTo(o2Type) < 0) {
            return SECOND_OBJECT_IS_BIGGER;
        }
        return AbsOrder.getInstance().compare(o1, o2);
    }

}
```

28 oop/ex5/filescript/WarningException.java

```java
package oop.ex5.filescript;

/**
 * Represents an exception that is thrown in case bad values, names and
 * parameters are given to ORDER or FILTER.
 * @author noa5
 */
public class WarningException extends Exception{

    private static final long serialVersionUID = 1L;

}
```

29 oop/ex5/filescript/WritableFilter.java

```java
 package oop.ex5.filescript;

import java.io.File;

/**
 * Represents the filter: Does file have writing permission?
 * (for the current user)
 * @author noa5
 */
public class WritableFilter extends PermissionFilter{

    /**
     * Constructor of the filter
     * @param name the name of the filter
     * @throws WarningException
     */
    public WritableFilter(String[] name) throws WarningException {
        super(name);
    }

    /**
     * Checks if a given file passes the filter - meaning, if has the wanted
     * permission for writing.
     * @param file the file we want to check
     * @return true if the file passes the filter, false otherwise
     */
    public boolean isFilePassFilter(File file) {
        String isWritablePermission = this.filterName[this.filterNameLength-1];
        if ((isWritablePermission.equals("YES") && file.canWrite()) ||
                (isWritablePermission.equals("NO") && !file.canWrite())) {
            return true;
        }
        return false;
    }

}
```