Contents

1 Basic Test Results

```
Logins: roigreenberg



compiling with
    javac -cp .:/cs/course/2013/oop/lib/junit4.jar  *.java oop/ex6/filescript/*.java


tests output :
        Perfect!
```

good =)

2 README

```
roigreenberg

####################
# File Description #
####################

package filescript:
    MyFileScript - the main class
    Parsing - class that create section according to command file
    Print - filter, order and print the files
    Section - class that have filter, order and array for warning
    SectionErrorException - throw error in case of bad section
    UsageExcaption - throw error in case of bad usage parameters
package filters:
    FilterFactory - create the filter instance
    FilterNameErrorException - throw error in case of bad name
    FilterParameterErrorException - throw error in case of bad parameters
    Filter - interface for the follow classes:
        SizeFilter - abstract class for the follow classes
            GreaterThanFilter
            SmallerThanFilter
            BetweenFilter
        BooleanFilter - abstract class for the follow classes
            ExecutableFilter
            WritableFilter
            HiddenFilter
        StringFilter - abstract class for the follow classes
            ContainsFilter
            FileFilter
            PrefixFilter
            SuffixFilter
        AllFilter - default filter
        NotFilter - decoretor to opposite filters
package orders:
    OrderFactory - create the order instance
    OrderErrorException - throw error in case of bad name
    Order - interface for the follow classes:
        AbsOrder - the default order
        TypeOrder
        SizeOrder
        ReverseOrder - decoretor to opposite orders

README - this file
###################
# Design + Answer #
###################

I follow the design offer by the course teem, so I explain it short
I seperate the work for indevidual classes.
The main class 'MyFileScript' working in that way:
    First, The 'Parsing' class read the command file and create array of sections.
    Then the 'Print' class get this array, and for each section filter the files according
    to the spesific filter then print them in the spesific order.
The parser using 2 factories, 1 for the filters and the other for the orders.
each factory  get a String with name and parameters(not always).
In case there a problam with the name or parameters the line number will be saved in an array
at the section variable and an default filter/order will be create.
otherwise, the factory will create filter/order as needed.
The factory are the only classes that know and care which filter/order there are.
```

```
All other classes use the master-class Filter/Order and don’t care what the exact filter/order.

For the filters/orders I done an intarface so all the filters/order will implement the same type
and I would be able to use all filters/orders without the need to know which spesific
filter/order I have
Also, I done filter classes for same types of filters.
for example, all the filters using compration to numbers extend ’SizeFilter’ in that way
I could use 1 constuctor for all the filters.
In addition I implement 2 decorator,(1 filter, 1 order) that return the opposite of the
filter/order they got as parameter.
All of the filters are implement one method: ’isPass()’ which get file and return if it pass
the filter or not.
All of the orders are implement Comparator<> and for that have a single method ’compare()’
which get 2 files and return the sequense of those file so the class can use as a comparator
for the method ’Collections.sort(List<T>, Comparator<T>)’ on the filtered file array.



The Excaption -
I create several exceptions.
For Error type 1:
        In case of bad filter/order name the follow exceptions will throw:
            FilterNameErrorException
            OrderErrorException
        In case of bad parameter for filter the follow exceptions will throw:
            FilterParameterErrorException
    In this case, I catch the exception at the parser, saved the problematic line and
    create the default filter/order and continue.

For Error type 2:
        In case of bad Section headlines the follow exceptions will throw:
            SectionErrorException
        In case of bad usage parameters the follow exceptions will throw:
            UsageErrorException
    In this case, I catch the exception at the main class, print "ERROR" and finish.

As I explain above the order classes are implement Comparator so I use ArrayList to hold
the filtered files and use ’Collections.sort(List<T>, Comparator<T>)’ to sort them.

#########################
# Implementation Issues #
#########################

I don’t remember having a serious implementation issue.
```

3 oop/ex6/filescript/MyFileScript.java

```java
/**
 *
 */
package oop.ex6.filescript;

import java.io.File;
import java.util.ArrayList;
import java.util.Collections;


/**
 * the main class, the Manager
 * @author roigreenberg
 *
 */
public class MyFileScript {

    /**
     * the manager, takes the directory and command file, read it and then print
     * the files at the directory according to the command file.
     * @param args - path of source  directory and command file
     */
    public static void main(String[] args) {
//        Parsing Parser = null;

            try {
            String sourceDirPath = args[0];
            String commandFilePath = args[1];


            Parsing Parser = new Parsing(commandFilePath
            ArrayList<Section> sections = Parser.CreateS

            Print.printFiles(sections, sourceDirPath);

            } catch (Exception e) {
                System.err.println("ERROR");

//                  throw new ErrorIIException();
            }

    }


}
```

-1/-2 Catching only the general Exception class is a bad idea (it hides other errors, such as run time errors) (code='catch_Exception_problem')

4 oop/ex6/filescript/Parsing.java

```java
/**
 *
 */
package oop.ex6.filescript;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Scanner;

import oop.ex6.filters.*;
import oop.ex6.orders.*;

/**
 * the parser. read the command file and create sections
 * @author roigreenberg
 *
 */
public class Parsing {

    private ArrayList<Section> sections = new ArrayList<Section>();
    private FileReader file = null;
    private Scanner commands;
    private static final String DEAFULT_ORDER = "abs";
    private static final String FILTER = "FILTER";
    private static final String ORDER = "ORDER";

    /**
     * the constructor - open the file and scan it's lines
     * @param commandFile - path to command file
     * @throws IOException - in case read the file failed.
     */
    public Parsing(String commandFile)
            throws IOException {
        try {
            file = new FileReader(commandFile);
            commands = new Scanner(file).useDelimiter("\\s*\n\\s*");
        } catch (FileNotFoundException e) {
            throw new IOException();
        }


    }

    /**
     * the method run over the commands and create section as command
     * @return ArrayList of sections
     * @throws SectionErrorException - in case filter heads("FILTER","ORDER"
     *          are wrote wrong.
     * @throws IOException
     */
    public ArrayList<Section> CreateSections()
            throws SectionErrorException, IOException{
        int lineNo = 1;
        String command = commands.next();
        while (commands.hasNext()){
            ArrayList<Integer> warnings = new ArrayList<>();
            warnings.clear();

            // deafult filter and order
            Filter filter = new AllFilter();
            Order order = new AbsOrder();

            String filterName = null, orderName = null;

            if (!command.equals(FILTER))
                throw new SectionErrorException();

            filterName = commands.next();
            lineNo++;

            try{
                filter = FilterFactory.createFilter(filterName);
            } catch (Exception e){
                warnings.add(lineNo);
            }
```

-1/-2 Catching only the general Exception class is a bad idea (it hides other errors, such as run time errors) (code='catch_Exception_problem')

```


            if (commands.hasNext()){
                command = commands.next();
                lineNo++;
            } else {
                throw new SectionErrorException();

            }


            if (command.equals(ORDER)){
                if (commands.hasNext()){
                    orderName = commands.next();
                    lineNo++;
                } else {
                    orderName = DEAFULT_ORDER ;
                }
            } else {
                throw new SectionErrorException();

            }

            if (!orderName.equals(FILTER)){
                try{
                    order = OrderFactory.createOrder(orderName);
                } catch (OrderErrorException e){
                    order = new AbsOrder();
                    warnings.add(lineNo);
                }
```

-0.5/-2.5 (code='general_error') Section shouldn't be familiar with the differnt types of filters and orders. You should have used some sort of method that returns the default filter/order

```

                if (commands.hasNext()){
                    command = commands.next();
                    lineNo++;
                }

            } else {
                command = orderName;
            }


            sections.add(new Section(filter, order, warnings));


        }
        commands.close();
        file.close();
        return sections;
    }
```

```
}
```

5 oop/ex6/filescript/Print.java

This class is redundant and could have been inserted in another class

-0.5/-2.5 (code='general_error') Should be private

```java
/**
 *
 */
package oop.ex6.filescript;

import java.io.File;
import java.util.ArrayList;
import java.util.Collections;

/**
 * this class handle the printing of the files
 * @author roigreenberg
 *
 */
public class Print  {
    static File[] files;

    /**
     * this method print the files name.
     * it run of every section, filtering the files than sorted according
     * to the order then print the warning and the files
     * @param sections - the sections created by the parser
     * @param onlyFiles - ArrayList of the files needed to be handled
     */
    public static void printFiles(ArrayList<Section> sections, String sourceDirPath){
        File[] files = new File(sourceDirPath).listFiles();
        ArrayList<File> onlyFiles = new ArrayList<>();
        //take only the files
        for (File file: files)
            if (file.isFile())
                onlyFiles.add(file);

        ArrayList<File> filteredFiles = new ArrayList<File>();
        for (Section section: sections){
            filteredFiles.clear();
            for (File file: onlyFiles){
                if (section.filter.isPass(file)){
                    filteredFiles.add(file);
                }
            }

            Collections.sort(filteredFiles, section.order);
            for (int w: section.warnings){
                System.out.println("Warning in line "+w);
            }
            for (File file : filteredFiles){
                System.out.println(file.getName());
            }

        }

    }

}
```

6 oop/ex6/filescript/Section.java

```java
/**
 *
 */
package oop.ex6.filescript;

import java.util.ArrayList;

import oop.ex6.filters.*;
import oop.ex6.orders.*;

/**
 * the Section class
 * create section with filter, order and warnings lines
 * @author roigreeberg
 *
 */
public class Section {

    Filter filter;
    Order order;
    ArrayList<Integer> warnings;
    /**
     * the constructor
     * @param filter - the filter for this section
     * @param order - the order for this section
     * @param warnings - the lines with warnings.
     */
    public Section(Filter filter, Order order, ArrayList<Integer> warnings){
        this.filter = filter;
        this.order = order;
        this.warnings = warnings;

    }

}
```

-1/-2.5 (code='general_error') Should be private

7 oop/ex6/filescript/SectionErrorException.java

```java
/**
 *
 */
package oop.ex6.filescript;

/**
 * Execption in case there a problem with the section heads
 * @author roigreenberg
 *
 */
public class SectionErrorException extends Exception {
    /**
     * default constructor
     */
    public SectionErrorException() {
        super("Bad Section heads");
    }
    /**
     * constructor
     * @param exception  - messege for the exception
     */
    public SectionErrorException(String exception) {
        super(exception);
    }
}
```

8 oop/ex6/filescript/UsageException.java

```java
/**
 *
 */
package oop.ex6.filescript;

/**
 * Execption in case there a problem with the files
 * @author roigreenberg
 *
 */
public class UsageException extends Exception {

    /**
     * default constructor
     */
    public UsageException() {
        super("Bad Usage");
    }
    /**
     * constructor
     * @param exception  - messege for the exception
     */
    public UsageException(String exception) {
        super(exception);
    }
}
```

9 oop/ex6/filters/AllFilter.java

```java
/**
 *
 */
package oop.ex6.filters;

import java.io.File;

/**
 * default filter
 * @author roigreenberg
 *
 */
public class AllFilter implements Filter {

    /**
     * @see oop.ex6.filters.Filter#isPass(java.io.File)
     * @return true always
     */
    public boolean isPass(File file){
        return true;
    }
}
```

10 oop/ex6/filters/BetweenFilter.java

```java
package oop.ex6.filters;
/**
 *
 */


import java.io.File;

/**
 * filter file according size between parameters
 * @author roigreenberg
 *
 */
public class BetweenFilter extends SizeFilters implements Filter {
    double param2;
    /**
     * the constructor
     * @param param1 - lower bound
     * @param param2 - upper bound
     */
    public BetweenFilter(double param1, double param2){
        super(param1);
        this.param2 = param2*KILO;
    }
    /**
     * @see oop.ex6.filters.Filter#isPass(java.io.File)
     * @return true iff file size between or equal to lower and upper bounds
     */
    public boolean isPass(File file){
        return ((file.length() >= param) && (file.length() <= param2));
    }
}
```

-0/-2.5 (code='general_error') no need to implement filter since SizeFilters already implements it

-0.5/-2.5 (code='general_error') Should be private

-1/-2.5 (code='general_error') you didn't check that param1<=param2

11 oop/ex6/filters/BooleanFilters.java

```java
/**
 *
 */
package  oop.ex6.filters;

import java.io.File;

/**
 * abstract class for filters with YES/NO parameters
 * @author roigreenberg
 *
 */
public abstract class BooleanFilters implements Filter {
    boolean yes = false;
    static final String YES = "YES";
    public BooleanFilters (String param){
        if (param.equals(YES))
            yes = true;
    }
    /** (non-Javadoc)
     * @see oop.ex6.filters.Filter#isPass(java.io.File)
     */
    @Override
    public boolean isPass(File file) {
        // TODO Auto-generated method stub
        return false;
    }

}
```

-0.5/-2.5 (code='general_error') Should be abstract

12 oop/ex6/filters/ContainsFilter.java

```java
/**
 *
 */
package oop.ex6.filters;

import java.io.File;

/**
 * filter file according to name contain given String
 * @author roigreenberg
 *
 */
public class ContainsFilter extends StringFilters implements Filter {
    public ContainsFilter(String param){
        super(param);
    }
    /**
     * @see oop.ex6.filters.Filter#isPass(ja
     * @return true iff file name contain the given parameter String
     */
    @Override
    public boolean isPass(File file) {
        return (file.getName().contains(param));
    }
}
```

-2/-5 Some parts of your code are redundant (code='redundant_code_problem') it's redundant to call super c-tor in this case

13 oop/ex6/filters/ExecutableFilter.java

```java
/**
 *
 */
package oop.ex6.filters;

import java.io.File;

/**
 * filter file according to it's executability
 * @author roigreenberg
 *
 */
public class ExecutableFilter extends BooleanFilters implements Filter {
    public ExecutableFilter(String param){
        super(param);
    }
    /**
     * @see oop.ex6.filters.Filter#isPass(java.io.File)
     * @return true iff file is/isn't(according to parameter) executable
     */
    @Override
    public boolean isPass(File file) {
        if (yes){
            return (file.canExecute());
        } else {
            return (!file.canExecute());
        }
    }
}
```

0/-5 Some parts of your code are redundant (code='redundant_code_problem') it's redundant to call super c-tor in this case

14 oop/ex6/filters/FileFilter.java

```java
/**
 *
 */
package oop.ex6.filters;

import java.io.File;

/**
 * filter file according to name contain equal to String
 * @author roigreenberg
 *
 */
public class FileFilter implements Filter {
    String param;
    public FileFilter(String param){
        this.param = param;
    }
    /**
     * @see oop.ex6.filters.Filter#isPass(java.io.File)
     * @return true iff file name equal to the given parameter String
     */
    @Override
    public boolean isPass(File file) {
        return (file.getName().equals(param));
    }
}
```

-0/-2.5 (code='general_error') should be under StringFilters

15 oop/ex6/filters/Filter.java

```java
/**
 *
 */
package oop.ex6.filters;

import java.io.File;

/**
 * interface for all the filter
 * @author roigreenberg
 *
 */
public interface Filter {

    /**
     * derermine if a file pass the filter
     * @param file - file to filter
     * @return true if the file pass the filter
     */
    public boolean isPass(File file);
}
```

You could have just used io.FIleFilter of Java

16 oop/ex6/filters/FilterFactory.java

```java
/**
 *
 */
package oop.ex6.filters;

/**
 * this class create a filter
 * @author roigreenberg
 *
 */
public class FilterFactory {
    static String filterName, param1, param2;
    /**
     * this method create filter according to the command file
     * @param args - the command line for creating a filter
     * @return filter - the filter as wrote in the command or AllFilter
     * if a problem has found at the command line
     * @throws FilterNameErrorException - in case of problem in filter name
     * @throws FilterParameterErrorException - in case of problem in parameters
     *          at the command line
     */
    public static Filter createFilter(String args)
            throws FilterNameErrorException, FilterParameterErrorException{
        Filter filter = new AllFilter();
        String[] params = args.split("#");
        filterName = params[0];
        try {
            param1 = params[1];
        } catch (ArrayIndexOutOfBoundsException e){
            if (!filterName.equals("all"))
                throw new FilterParameterErrorException();
        }
        double doubleParam1;
        switch (filterName){
        case("all"): {
            break;
        }
        case("greater_than"): {
            doubleParam1 = Double.parseDouble(param1) ;
            if (doubleParam1 >= 0){
                filter = new GreaterThanFilter(doubleParam1);
                break;
            } else {
                throw new FilterParameterErrorException();
            }

        }
        case("between"): {
            doubleParam1 = Double.parseDouble(param1);
            param2 = params[2];

            double doubleParam2 = Double.parseDouble(param2);
            if ((doubleParam1 >= 0) && (doubleParam1<=doubleParam2)){
                filter = new BetweenFilter(doubleParam1,doubleParam2);
                break;
            } else {
                throw new FilterParameterErrorException();
            }

```

-1.5/-2.5 (code='general_error') checking specific filters and throwing specific exceptions about them should be done in their class

```java
        }
        case("smaller_than"): {
            doubleParam1 = Double.parseDouble(param1) ;
            if (doubleParam1 >= 0){
                filter = new SmallerThanFilter(doubleParam1);
                break;
            } else {
                throw new FilterParameterErrorException();
            }

        }
        case("file"): {
            filter = new FileFilter(param1);
            break;
        }
        case("contains"): {
            filter = new ContainsFilter(param1);
            break;
        }
        case("prefix"): {
            filter = new PrefixFilter(param1);
            break;
        }
        case("suffix"): {
            filter = new SuffixFilter(param1);
            break;
        }
        case("writable"): {
            if (param1.equals("YES") || param1.equals("NO")){
                filter = new WritableFilter(param1);
                break;
            } else {
                throw new FilterParameterErrorException();
            }
        }
        case("executable"): {
            if (param1.equals("YES") || param1.equals("NO")){
                filter = new ExecutableFilter(param1);
                break;
            } else {
                throw new FilterParameterErrorException();
            }
        }
        case("hidden"): {
            if (param1.equals("YES") || param1.equals("NO")){
                filter = new HiddenFilter(param1);
                break;
            } else {
                throw new FilterParameterErrorException();
            }
        }
        default:
            throw new FilterNameErrorException();
        }
        if (!args.endsWith("NOT")){
            return filter;
        } else {
            return new NotFilter(filter);
        }
    }
}
```

17 oop/ex6/filters/FilterNameErrorException.java

```java
/**
 *
 */
package oop.ex6.filters;

/**
 * Execption in case there a problem with the filter name
 * @author roigreenberg
 *
 */
public class FilterNameErrorException extends Exception {
    /**
     * default constructor
     */
    public FilterNameErrorException() {
        super("Not Such filter");
    }
    /**
     * constructor
     * @param exception  - messege for the exception
     */
    public FilterNameErrorException(String exception) {
        super(exception);
    }
}
```

18 oop/ex6/filters/FilterParameterErrorException.java

```java
/**
 *
 */
package oop.ex6.filters;

/**
 * Execption in case there a problem with the filter parametres
 * @author roigreenberg
 *
 */
public class FilterParameterErrorException extends Exception {
    /**
     * default constructor
     */
    public FilterParameterErrorException() {
        super("wrong parameters");
    }
    /**
     * constructor
     * @param exception  - messege for the exception
     */
    public FilterParameterErrorException(String exception) {
        super(exception);
    }
}
```

19 oop/ex6/filters/GreaterThanFilter.java

```java
/**
 *
 */
package oop.ex6.filters;

import java.io.File;

/**
 * filter file according to size greater than parameter
 * @author roigreenberg
 *
 */
public class GreaterThanFilter extends SizeFilters implements Filter {
    public GreaterThanFilter(double param){
        super(param);
    }
    /**
     * @see oop.ex6.filters.Filter#isPass(java.io.File)
     * @return true iff file size greater than parameter
     */
    public boolean isPass(File file){
        return (file.length() > param);
    }

}
```

20 oop/ex6/filters/HiddenFilter.java

```java
/**
 *
 */
package oop.ex6.filters;

import java.io.File;

/**
 * filter file according to it's hidden state
 * @author roigreenberg
 *
 */
public class HiddenFilter extends BooleanFilters implements Filter {
    public HiddenFilter(String param){
        super(param);
    }
    /**
     * @see oop.ex6.filters.Filter#isPass(java.io.File)
     * @return true iff file is/isn't(according to parameter) hidden
     */
    @Override
    public boolean isPass(File file) {
        if (yes){
            return (file.isHidden());
        } else {
            return (!file.isHidden());
        }
    }
}
```

0/-5 Some parts of your code are redundant (code='redundant_code_problem') it's redundant to call super c-tor in this case

21 oop/ex6/filters/NotFilter.java

```java
/**
 *
 */
package oop.ex6.filters;

import java.io.File;

/**
 * decorator to opposite filters
 * @author roigreenberg
 *
 */
public class NotFilter implements Filter {
    Filter filter;
    public NotFilter(Filter filter){
        this.filter = filter;
    }
    /**
     * @see oop.ex6.filters.Filter#isPass(java.io.File)
     * @return - true if file DID NOT pass the filter
     */
    @Override
    public boolean isPass(File file) {
        return (!filter.isPass(file));
    }

}
```

22 oop/ex6/filters/PrefixFilter.java

```java
/**
 *
 */
package oop.ex6.filters;

import java.io.File;

/**
 * filter file according to name start with given String
 * @author roigreenberg
 *
 */
public class PrefixFilter extends StringFilters implements Filter {
    public PrefixFilter(String param){
        super(param);
    }
    /**
     * @see oop.ex6.filters.Filter#isPass(java.io.File)
     * @return true iff file name start with the given parameter String
     */
    @Override
    public boolean isPass(File file) {

        return (file.getName().startsWith(param));
    }


}
```

23 oop/ex6/filters/SizeFilters.java

```java
/**
 *
 */
package oop.ex6.filters;

import java.io.File;

/**
 * abstract class for filter with numeric parameters
 * @author roigreenberg
 *
 */
public abstract class SizeFilters implements Filter {
    double param;
    public static final int KILO = 1024;
    public SizeFilters(double param) {
        this.param = param*KILO;
    }
    /**
     * @see oop.ex6.filters.Filter#isPass(java.io.File)
     */
    @Override
    public boolean isPass(File file) {
        // TODO Auto-generated method stub
        return true;
    }

}
```

should have been abstract

24 oop/ex6/filters/SmallerThanFilter.java

```java
/**
 *
 */
package oop.ex6.filters;

import java.io.File;

/**
 * filter file according to size smaller than parameter
 * @author roigreenberg
 *
 */
public class SmallerThanFilter extends SizeFilters implements Filter {
    public SmallerThanFilter(double param){
        super(param);
    }
    /**
     * @see oop.ex6.filters.Filter#isPass(java.io.File)
     * @return true iff file size smaller than parameter
     */
    public boolean isPass(File file){

        return (file.length() < param);
    }
}
```

25 oop/ex6/filters/StringFilters.java

```java
/**
 *
 */
package oop.ex6.filters;

import java.io.File;

/**
 * abstarct class for filters with String parameter
 * @author roigreenberg
 *
 */
public abstract class StringFilters implements Filter {
    String param;
    public StringFilters(String param){
        this.param = param;
    }
    /**
     * @see oop.ex6.filters.Filter#isPass(java.io.File)
     */
    @Override
    public boolean isPass(File file) {
        // TODO Auto-generated method stub
        return false;
    }

}
```

26 oop/ex6/filters/SuffixFilter.java

```java
/**
 *
 */
package oop.ex6.filters;

import java.io.File;

/**
 * filter file according to name end with given String
 * @author roigreenberg
 *
 */
public class SuffixFilter extends StringFilters  implements Filter {
    public SuffixFilter(String param){
        super(param);
    }
    /**
     * @see oop.ex6.filters.Filter#isPass(java.io.File)
     * @return true iff file name end with the given parameter String
     */
    @Override
    public boolean isPass(File file) {

        return (file.getName().endsWith(param));
    }
}
```

27 oop/ex6/filters/WritableFilter.java

```java
/**
 *
 */
package oop.ex6.filters;

import java.io.File;

/**
 * filter file according to it's writability
 * @author roigreenberg
 *
 */
public class WritableFilter extends BooleanFilters implements Filter {
    public WritableFilter(String param){
        super(param);
    }
    /**
     * @see oop.ex6.filters.Filter#isPass(java.io.File)
     * @return true iff file is/isn't(according to parameter) writable
     */
    @Override
    public boolean isPass(File file) {
        if (yes){
            return (file.canWrite());
        } else {
            return (!file.canWrite());
        }
    }
}
```

28 oop/ex6/orders/AbsOrder.java

```java
/**
 *
 */
package oop.ex6.orders;

import java.io.File;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;

/**
 * order file according to file's path
 * @author roigreenberg
 *
 */
public class AbsOrder implements Order, Comparator<File> {

    /**
     * @see java.util.Comparator#compare(java.lang.Object, java.lang.Object)
     * @see oop.ex6.orders.Order#compare(java.lang.Object, java.lang.Object)
     * the compration is according to the path of the file
     * @return result - number indicate the order between the files
     */
    @Override
    public int compare(File file1, File file2) {
            return String.valueOf(file1.getPath())
                    .compareTo(file2.getPath());
    }

}
```

redundant to implement both Order and Comparator

29 oop/ex6/orders/Order.java

```java
/**
 *
 */
package oop.ex6.orders;

import java.io.File;
import java.util.ArrayList;
import java.util.Comparator;

/**
 * interface for all the orders
 * extends Comparator<> in order to be comparator for sorting an array
 * @author roigreenberg
 *
 */
public interface Order extends Comparator<File>{
    /**
     * compare between 2 file
     *@see java.util.Comparator#compare(java.lang.Object, java.lang.Object)
     */
    public int compare(File file1, File file2);
}
```

30 oop/ex6/orders/OrderErrorException.java

```java
/**
 *
 */
package oop.ex6.orders;

/**
 * Execption in case there a problem with the order name
 * @author roigreenberg
 *
 */
public class OrderErrorException extends Exception {
    /**
     * default constructor
     */
    public OrderErrorException() {
        super("Not Such Order");
    }
    /**
     * constructor
     * @param exception  - messege for the exception
     */
    public OrderErrorException(String exception) {
        super(exception);
    }
}
```

31 oop/ex6/orders/OrderFactory.java

```java
/**
 *
 */
package oop.ex6.orders;

import oop.ex6.filters.*;


/**
 * this class create a order
 * @author roigreenberg
 *
 */
public class OrderFactory {
    static String orderName;
    /**
     * this method create order according to the command file
     * @param args - the command line for creating a order
     * @return order - the order as wrote in the command or default order
     * if a problem has found at the command line
     * @throws OrderErrorException -in case of problem in filter name
     */
    public static Order createOrder(String args)
            throws OrderErrorException {
        Order order = new AbsOrder();
        String[] params = args.split("#");
        orderName = params[0];
        switch(orderName){
        case("abs"):{
            break;
        }
        case("type"):{
            order = new TypeOrder();
            break;
        }
        case("size"):{
            order =  new SizeOrder();
            break;
        }
        default:
            throw new OrderErrorException();
        }
        if (!args.endsWith("REVERSE")){
            return order;
        } else {
            return new ReverseOrder(order);
        }
    }
}
```

32 oop/ex6/orders/ReverseOrder.java

```java
/**
 *
 */
package oop.ex6.orders;

import java.io.File;
import java.util.Collections;
import java.util.Comparator;

/**
 * decorator to reverse orders
 * @author roigreenberg
 *
 */
public class ReverseOrder implements Order,Comparator<File> {
    Order order;
    public ReverseOrder (Order order){
        this.order = order;
    }
    /**
     * @see java.util.Comparator#compare(java.lang.Object, java.lang.Object)
     * @see oop.ex6.orders.Order#compare(java.lang.Object, java.lang.Object)
     * @return the opposite of the result from the original order
     */
    @Override
    public int compare(File file1, File file2) {
        return -1*order.compare(file1, file2);

    }

}
```

33 oop/ex6/orders/SizeOrder.java

```java
/**
 *
 */
package oop.ex6.orders;

import java.io.File;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;

import org.omg.PortableInterceptor.SUCCESSFUL;

/**
 * order file according to file's size
 * @author roigreenberg
 *
 */
public class SizeOrder implements Order,Comparator<File> {

    /**
     * @see java.util.Comparator#compare(java.lang.Object, java.lang.Object)
     * @see oop.ex6.orders.Order#compare(java.lang.Object, java.lang.Object)
     * the compration is according to the size of the file
     * in case of even, the compration will be like absOrder
     * @return result - number indicate the order between the files
     */
    @Override
    public int compare(File file1, File file2) {
        int result = Long.valueOf(file1.length())
                .compareTo(file2.length());

        if (result != 0)
            return result;
        else
            return new AbsOrder().compare(file1, file2);

}










}
```

34 oop/ex6/orders/TypeOrder.java

```java
/**
 *
 */
package oop.ex6.orders;

import java.io.File;
import java.util.ArrayList;
import java.util.Comparator;

/**
 * order file according to file's type
 * @author roigreenberg
 *
 */
public class TypeOrder implements Order, Comparator<File> {
    /**
     * @see java.util.Comparator#compare(java.lang.Object, java.lang.Object)
     * @see oop.ex6.orders.Order#compare(java.lang.Object, java.lang.Object)
     * the compration is according to the type of the file
     * in case of even, the compration will be like absOrder
     * @return result - number indicate the order between the files
     */
    @Override
    public int compare(File file1, File file2){

        int index1 = file1.getName().lastIndexOf(".")+1;
        int index2 = file2.getName().lastIndexOf(".")+1;
        int result = String.valueOf(file1.getName().substring(index1))
                .compareTo(file2.getName().substring(index2));

        if (result != 0)
            return result;
        else
            return  new AbsOrder().compare(file1, file2);
    }

}
```