

# Contents

1	Basic Test Results	2
2	README	3
3	oop/ex6/filescript/MyFileScript.java	5
4	oop/ex6/filescript/Parsing.java	6
5	oop/ex6/filescript/Print.java	9
6	oop/ex6/filescript/Section.java	10
7	oop/ex6/filescript/SectionErrorException.java	11
8	oop/ex6/filescript/UsageException.java	12
9	oop/ex6/filters/AllFilter.java	13
10	oop/ex6/filters/BetweenFilter.java	14
11	oop/ex6/filters/BooleanFilters.java	15
12	oop/ex6/filters/ContainsFilter.java	16
13	oop/ex6/filters/ExecutableFilter.java	17
14	oop/ex6/filters/FileFilter.java	18
15	oop/ex6/filters/Filter.java	19
16	oop/ex6/filters/FilterFactory.java	20
17	oop/ex6/filters/FilterNameErrorException.java	22
18	oop/ex6/filters/FilterParameterErrorException.java	23
19	oop/ex6/filters/GreaterThanFilter.java	24
20	oop/ex6/filters/HiddenFilter.java	25
21	oop/ex6/filters/NotFilter.java	26
22	oop/ex6/filters/PrefixFilter.java	27

23	<a href="#">oop/ex6/filters/SizeFilters.java</a>	28
24	<a href="#">oop/ex6/filters/SmallerThanFilter.java</a>	29
25	<a href="#">oop/ex6/filters/StringFilters.java</a>	30
26	<a href="#">oop/ex6/filters/SuffixFilter.java</a>	31
27	<a href="#">oop/ex6/filters/WritableFilter.java</a>	32
28	<a href="#">oop/ex6/orders/AbsOrder.java</a>	33
29	<a href="#">oop/ex6/orders/Order.java</a>	34
30	<a href="#">oop/ex6/orders/OrderErrorException.java</a>	35
31	<a href="#">oop/ex6/orders/OrderFactory.java</a>	36
32	<a href="#">oop/ex6/orders/ReverseOrder.java</a>	37
33	<a href="#">oop/ex6/orders/SizeOrder.java</a>	38
34	<a href="#">oop/ex6/orders/TypeOrder.java</a>	39

# 1 Basic Test Results

```
1 Logins: roigreenberg
2
3
4
5 compiling with
6     javac -cp ./cs/course/2013/oop/lib/junit4.jar *.java oop/ex6/filescript/*.java
7
8
9 tests output :
10     Perfect!
```

good =)

## 2 README

```
1  roigreenberg
2
3  #####
4  # File Description #
5  #####
6
7  package filesript:
8      MyFileScript - the main class
9      Parsing - class that create section according to command file
10     Print - filter, order and print the files
11     Section - class that have filter, order and array for warning
12     SectionErrorException - throw error in case of bad section
13     UsageExcaption - throw error in case of bad usage parameters
14 package filters:
15     FilterFactory - create the filter instance
16     FilterNameErrorException - throw error in case of bad name
17     FilterParameterErrorException - throw error in case of bad parameters
18     Filter - interface for the follow classes:
19         SizeFilter - abstract class for the follow classes
20             GreaterThanFilter
21             SmallerThanFilter
22             BetweenFilter
23         BooleanFilter - abstract class for the follow classes
24             ExecutableFilter
25             WritableFilter
26             HiddenFilter
27         StringFilter - abstract class for the follow classes
28             ContainsFilter
29             FileFilter
30             PrefixFilter
31             SuffixFilter
32         AllFilter - default filter
33         NotFilter - decoretor to opposite filters
34 package orders:
35     OrderFactory - create the order instance
36     OrderErrorException - throw error in case of bad name
37     Order - interface for the follow classes:
38         AbsOrder - the default order
39         TypeOrder
40         SizeOrder
41         ReverseOrder - decoretor to opposite orders
42
43 README - this file
44 #####
45 # Design + Answer #
46 #####
47
48 I follow the design offer by the course teem, so I explain it short
49 I seperate the work for indevidual classes.
50 The main class 'MyFileScript' working in that way:
51     First, The 'Parsing' class read the command file and create array of sections.
52     Then the 'Print' class get this array, and for each section filter the files according
53     to the spesific filter then print them in the spesific order.
54 The parser using 2 factories, 1 for the filters and the other for the orders.
55 each factory get a String with name and parameters(not always).
56 In case there a problem with the name or parameters the line number will be saved in an array
57 at the section variable and an default filter/order will be create.
58 otherwise, the factory will create filter/order as needed.
59 The factory are the only classes that know and care which filter/order there are.
```

```

60 All other classes use the master-class Filter/Order and don't care what the exact filter/order.
61
62 For the filters/orders I done an interface so all the filters/order will implement the same type
63 and I would be able to use all filters/orders without the need to know which spesific
64 filter/order I have
65 Also, I done filter classes for same types of filters.
66 for example, all the filters using compration to numbers extend 'SizeFilter' in that way
67 I could use 1 constuctor for all the filters.
68 In addition I implement 2 decorator,(1 filter, 1 order) that return the opposite of the
69 filter/order they got as parameter.
70 All of the filters are implement one method: 'isPass()' which get file and return if it pass
71 the filter or not.
72 All of the orders are implement Comparator<> and for that have a single method 'compare()'
73 which get 2 files and return the sequeense of those file so the class can use as a comparator
74 for the method 'Collections.sort(List<T>, Comparator<T>)' on the filtered file array.
75
76
77
78 The Excaption -
79 I create several exceptions.
80 For Error type 1:
81     In case of bad filter/order name the follow exceptions will throw:
82         FilterNameErrorException
83         OrderErrorException
84     In case of bad parameter for filter the follow exceptions will throw:
85         FilterParameterErrorException
86     In this case, I catch the exception at the parser, saved the problematic line and
87     create the default filter/order and continue.
88
89 For Error type 2:
90     In case of bad Section headlines the follow exceptions will throw:
91         SectionErrorException
92     In case of bad usage parameters the follow exceptions will throw:
93         UsageErrorException
94     In this case, I catch the exception at the main class, print "ERROR" and finish.
95
96 As I explain above the order classes are implement Comparator so I use ArrayList to hold
97 the filtered files and use 'Collections.sort(List<T>, Comparator<T>)' to sort them.
98
99 #####
100 # Implementation Issues #
101 #####
102
103 I don't remember having a serious implementation issue.

```

### 3 oop/ex6/filescript/MyFileScript.java

```
1  /**
2   *
3   */
4  package oop.ex6.filescript;
5
6  import java.io.File;
7  import java.util.ArrayList;
8  import java.util.Collections;
9
10
11 /**
12  * the main class, the Manager
13  * @author roigreenberg
14  *
15  */
16 public class MyFileScript {
17
18     /**
19     * the manager, takes the directory and command file, read it and then print
20     * the files at the directory according to the command file.
21     * @param args - path of source directory and command file
22     */
23     public static void main(String[] args) {
24         // Parsing Parser = null;
25
26         try {
27             String sourceDirPath = args[0];
28             String commandFilePath = args[1];
29
30
31             Parsing Parser = new Parsing(commandFilePath);
32             ArrayList<Section> sections = Parser.CreateSections();
33
34             Print.printFiles(sections, sourceDirPath);
35
36         } catch (Exception e) {
37             System.err.println("ERROR");
38
39             // throw new ErrorIIException();
40         }
41
42     }
43
44
45 }
```

-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

```
1  /**
2   *
3   */
4  package oop.ex6.filescript;
5
6  import java.io.FileNotFoundException;
7  import java.io.FileReader;
8  import java.io.IOException;
9  import java.util.ArrayList;
10 import java.util.Arrays;
11 import java.util.Scanner;
12
13 import oop.ex6.filters.*;
14 import oop.ex6.orders.*;
15
16 /**
17  * the parser. read the command file and create sections
18  * @author roigreenberg
19  *
20  */
21 public class Parsing {
22
23     private ArrayList<Section> sections = new ArrayList<Section>();
24     private FileReader file = null;
25     private Scanner commands;
26     private static final String DEAFULT_ORDER = "abs";
27     private static final String FILTER = "FILTER";
28     private static final String ORDER = "ORDER";
29
30     /**
31      * the constructor - open the file and scan it's lines
32      * @param commandFile - path to command file
33      * @throws IOException - in case read the file failed.
34      */
35     public Parsing(String commandFile)
36         throws IOException {
37         try {
38             file = new FileReader(commandFile);
39             commands = new Scanner(file).useDelimiter("\\s*\\n\\s*");
40         } catch (FileNotFoundException e) {
41             throw new IOException();
42         }
43
44     }
45
46
47     /**
48      * the method run over the commands and create section as command
49      * @return ArrayList of sections
50      * @throws SectionErrorException - in case filter heads("FILTER","ORDER"
51      *         are wrote wrong.
52      * @throws IOException
53      */
54     public ArrayList<Section> CreateSections()
55         throws SectionErrorException, IOException{
56         int lineNo = 1;
57         String command = commands.next();
58         while (commands.hasNext()){
59             ArrayList<Integer> warnings = new ArrayList<>();
```

```

60     warnings.clear();
61
62     // default filter and order
63     Filter filter = new AllFilter();
64     Order order = new AbsOrder();
65
66     String filterName = null, orderName = null;
67
68     if (!command.equals(FILTER))
69         throw new SectionErrorException();
70
71     filterName = commands.next();
72     lineNo++;
73
74     try{
75         filter = FilterFactory.createFilter(filterName);
76     } catch (Exception e){
77         warnings.add(lineNo);
78     }
79
80     if (commands.hasNext()){
81         command = commands.next();
82         lineNo++;
83     } else {
84         throw new SectionErrorException();
85     }
86
87
88
89
90     if (command.equals(ORDER)){
91         if (commands.hasNext()){
92             orderName = commands.next();
93             lineNo++;
94         } else {
95             orderName = DEAFULT_ORDER ;
96         }
97     } else {
98         throw new SectionErrorException();
99     }
100
101
102     if (!orderName.equals(FILTER)){
103         try{
104             order = OrderFactory.createOrder(orderName);
105         } catch (OrderErrorException e){
106             order = new AbsOrder();
107             warnings.add(lineNo);
108         }
109
110         if (commands.hasNext()){
111             command = commands.next();
112             lineNo++;
113         }
114
115     } else {
116         command = orderName;
117     }
118
119
120     sections.add(new Section(filter, order, warnings));
121
122
123 }
124 commands.close();
125 file.close();
126 return sections;
127 }

```

-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')

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





## 5 oop/ex6/filescript/Print.java

```
1  /**
2   *
3   */
4  package oop.ex6.filescript;
5
6  import java.io.File;
7  import java.util.ArrayList;
8  import java.util.Collections;
9
10 /**
11  * this class handle the printing of the files
12  * @author roigreenberg
13  *
14  */
15 public class Print {
16     static File[] files;
17
18     /**
19     * this method print the files name.
20     * it run of every section, filtering the files than sorted according
21     * to the order then print the warning and the files
22     * @param sections - the sections created by the parser
23     * @param onlyFiles - ArrayList of the files needed to be handled
24     */
25     public static void printFiles(ArrayList<Section> sections, String sourceDirPath){
26         File[] files = new File(sourceDirPath).listFiles();
27         ArrayList<File> onlyFiles = new ArrayList<>();
28         //take only the files
29         for (File file: files)
30             if (file.isFile())
31                 onlyFiles.add(file);
32
33         ArrayList<File> filteredFiles = new ArrayList<File>();
34         for (Section section: sections){
35             filteredFiles.clear();
36             for (File file: onlyFiles){
37                 if (section.filter.isPass(file)){
38                     filteredFiles.add(file);
39                 }
40             }
41
42             Collections.sort(filteredFiles, section.order);
43             for (int w: section.warnings){
44                 System.out.println("Warning in line "+w);
45             }
46             for (File file : filteredFiles){
47                 System.out.println(file.getName());
48             }
49         }
50     }
51 }
52
53
54 }
```

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

-0.5/-2.5 (code='general\_error') Should be private

## 6 oop/ex6/filescript/Section.java

```
1  /**
2   *
3   */
4  package oop.ex6.filescript;
5
6  import java.util.ArrayList;
7
8  import oop.ex6.filters.*;
9  import oop.ex6.orders.*;
10
11 /**
12  * the Section class
13  * create section with filter, order and warnings lines
14  * @author roigreeberg
15  *
16  */
17 public class Section {
18
19     Filter filter;
20     Order order;
21     ArrayList<Integer> warnings;
22     /**
23      * the constructor
24      * @param filter - the filter for this section
25      * @param order - the order for this section
26      * @param warnings - the lines with warnings.
27      */
28     public Section(Filter filter, Order order, ArrayList<Integer> warnings){
29         this.filter = filter;
30         this.order = order;
31         this.warnings = warnings;
32     }
33 }
34
35 }
```

-1/-2.5 (code='general\_error') Should be private

## 7 oop/ex6/filescript/SectionErrorException.java

```
1  /**
2   *
3   */
4  package oop.ex6.filescript;
5
6  /**
7   * Exception in case there a problem with the section heads
8   * @author roigreenberg
9   *
10  */
11  public class SectionErrorException extends Exception {
12      /**
13       * default constructor
14       */
15      public SectionErrorException() {
16          super("Bad Section heads");
17      }
18      /**
19       * constructor
20       * @param exception - messege for the exception
21       */
22      public SectionErrorException(String exception) {
23          super(exception);
24      }
25  }
```

## 8 oop/ex6/filescript/UsageException.java

```
1  /**
2   *
3   */
4  package oop.ex6.filescript;
5
6  /**
7   * Exception in case there a problem with the files
8   * @author roigreenberg
9   *
10  */
11  public class UsageException extends Exception {
12
13      /**
14       * default constructor
15       */
16      public UsageException() {
17          super("Bad Usage");
18      }
19
20      /**
21       * constructor
22       * @param exception - messege for the exception
23       */
24      public UsageException(String exception) {
25          super(exception);
26      }
27  }
```

## 9 oop/ex6/filters/AllFilter.java

```
1  /**
2   *
3   */
4  package oop.ex6.filters;
5
6  import java.io.File;
7
8  /**
9   * default filter
10  * @author roigreenberg
11  *
12  */
13  public class AllFilter implements Filter {
14
15      /**
16       * @see oop.ex6.filters.Filter#isPass(java.io.File)
17       * @return true always
18       */
19      public boolean isPass(File file){
20          return true;
21      }
22  }
```

## 10 oop/ex6/filters/BetweenFilter.java

```
1  package oop.ex6.filters;
2  /**
3   *
4   */
5
6
7  import java.io.File;
8
9  /**
10   * filter file according size between parameters
11   * @author roigreenberg
12   *
13   */
14  public class BetweenFilter extends SizeFilters implements Filter {
15      double param2;
16      /**
17       * the constructor
18       * @param param1 - lower bound
19       * @param param2 - upper bound
20       */
21      public BetweenFilter(double param1, double param2){
22          super(param1);
23          this.param2 = param2*KILO;
24      }
25      /**
26       * @see oop.ex6.filters.Filter#isPass(java.io.File)
27       * @return true iff file size between or equal to lower and upper bounds
28       */
29      public boolean isPass(File file){
30          return ((file.length() >= param) && (file.length() <= param2));
31      }
32  }
```

-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

```
1  /**
2   *
3   */
4  package oop.ex6.filters;
5
6  import java.io.File;
7
8  /**
9   * abstract class for filters with YES/NO parameters
10   * @author roigreenberg
11   *
12   */
13  public abstract class BooleanFilters implements Filter {
14      boolean yes = false;
15      static final String YES = "YES";
16      public BooleanFilters (String param){
17          if (param.equals(YES))
18              yes = true;
19      }
20      /** (non-Javadoc)
21       * @see oop.ex6.filters.Filter#isPass(java.io.File)
22       */
23      @Override
24      public boolean isPass(File file) {
25          // TODO Auto-generated method stub
26          return false;
27      }
28  }
29  }
```

-0.5/-2.5 (code='general\_error') Should be abstract



## 12 oop/ex6/filters/ContainsFilter.java

```
1  /**
2   *
3   */
4  package oop.ex6.filters;
5
6  import java.io.File;
7
8  /**
9   * filter file according to name contain given String
10   * @author roigreenberg
11   *
12   */
13  public class ContainsFilter extends StringFilters implements Filter {
14      public ContainsFilter(String param){
15          super(param);
16      }
17      /**
18       * @see oop.ex6.filters.Filter#isPass(java.io.File)
19       * @return true iff file name contain the given parameter string
20       */
21      @Override
22      public boolean isPass(File file) {
23          return (file.getName().contains(param));
24      }
25  }
```

-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

```
1  /**
2   *
3   */
4  package oop.ex6.filters;
5
6  import java.io.File;
7
8  /**
9   * filter file according to it's executability
10   * @author roigreenberg
11   *
12   */
13  public class ExecutableFilter extends BooleanFilters implements Filter {
14      public ExecutableFilter(String param){
15          super(param);
16      }
17      /**
18       * @see oop.ex6.filters.Filter#isPass(java.io.File)
19       * @return true iff file is/isn't (according to parameter) executable
20       */
21      @Override
22      public boolean isPass(File file) {
23          if (yes){
24              return (file.canExecute());
25          } else {
26              return (!file.canExecute());
27          }
28      }
29  }
```

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

```
1  /**
2   *
3   */
4  package oop.ex6.filters;
5
6  import java.io.File;
7
8  /**
9   * filter file according to name contain equal to String
10   * @author roigreenberg
11   */
12  */
13  public class FileFilter implements Filter {
14      String param;
15      public FileFilter(String param){
16          this.param = param;
17      }
18      /**
19       * @see oop.ex6.filters.Filter#isPass(java.io.File)
20       * @return true iff file name equal to the given parameter String
21       */
22      @Override
23      public boolean isPass(File file) {
24          return (file.getName().equals(param));
25      }
26  }
```

-0/-2.5 (code='general\_error') should be under StringFilters

## 15 oop/ex6/filters/Filter.java

```
1  /**
2   *
3   */
4  package oop.ex6.filters;
5
6  import java.io.File;
7
8  /**
9   * interface for all the filter
10   * @author roigreenberg
11   *
12   */
13  public interface Filter {
14
15      /**
16       * determine if a file pass the filter
17       * @param file - file to filter
18       * @return true if the file pass the filter
19       */
20      public boolean isPass(File file);
21  }
```

You could have just used `io.FlleFilter` of Java

## 16 oop/ex6/filters/FilterFactory.java

```
1  /**
2   *
3   */
4  package oop.ex6.filters;
5
6  /**
7   * this class create a filter
8   * @author roigreenberg
9   *
10  */
11  public class FilterFactory {
12      static String filterName, param1, param2;
13      /**
14       * this method create filter according to the command file
15       * @param args - the command line for creating a filter
16       * @return filter - the filter as wrote in the command or AllFilter
17       * if a problem has found at the command line
18       * @throws FilterNameErrorException - in case of problem in filter name
19       * @throws FilterParameterErrorException - in case of problem in parameters
20       * at the command line
21       */
22      public static Filter createFilter(String args)
23          throws FilterNameErrorException, FilterParameterErrorException{
24          Filter filter = new AllFilter();
25          String[] params = args.split("#");
26          filterName = params[0];
27          try {
28              param1 = params[1];
29          } catch (ArrayIndexOutOfBoundsException e){
30              if (!filterName.equals("all"))
31                  throw new FilterParameterErrorException();
32          }
33          double doubleParam1;
34          switch (filterName){
35              case("all"): {
36                  break;
37              }
38              case("greater_than"): {
39                  doubleParam1 = Double.parseDouble(param1) ;
40                  if (doubleParam1 >= 0){
41                      filter = new GreaterThanFilter(doubleParam1);
42                      break;
43                  } else {
44                      throw new FilterParameterErrorException();
45                  }
46              }
47          }
48          case("between"): {
49              doubleParam1 = Double.parseDouble(param1);
50              param2 = params[2];
51
52              double doubleParam2 = Double.parseDouble(param2);
53              if ((doubleParam1 >= 0) && (doubleParam1<=doubleParam2)){
54                  filter = new BetweenFilter(doubleParam1,doubleParam2);
55                  break;
56              } else {
57                  throw new FilterParameterErrorException();
58              }
59          }
```

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

```

60     }
61     case("smaller_than"): {
62         doubleParam1 = Double.parseDouble(param1) ;
63         if (doubleParam1 >= 0){
64             filter = new SmallerThanFilter(doubleParam1);
65             break;
66         } else {
67             throw new FilterParameterErrorException();
68         }
69     }
70 }
71 case("file"): {
72     filter = new FileFilter(param1);
73     break;
74 }
75 case("contains"): {
76     filter = new ContainsFilter(param1);
77     break;
78 }
79 case("prefix"): {
80     filter = new PrefixFilter(param1);
81     break;
82 }
83 case("suffix"): {
84     filter = new SuffixFilter(param1);
85     break;
86 }
87 case("writable"): {
88     if (param1.equals("YES") || param1.equals("NO")){
89         filter = new WritableFilter(param1);
90         break;
91     } else {
92         throw new FilterParameterErrorException();
93     }
94 }
95 case("executable"): {
96     if (param1.equals("YES") || param1.equals("NO")){
97         filter = new ExecutableFilter(param1);
98         break;
99     } else {
100         throw new FilterParameterErrorException();
101     }
102 }
103 case("hidden"): {
104     if (param1.equals("YES") || param1.equals("NO")){
105         filter = new HiddenFilter(param1);
106         break;
107     } else {
108         throw new FilterParameterErrorException();
109     }
110 }
111 default:
112     throw new FilterNameErrorException();
113 }
114 if (!args.endsWith("NOT")){
115     return filter;
116 } else {
117     return new NotFilter(filter);
118 }
119 }
120 }

```

## 17 oop/ex6/filters/FilterNameErrorException.java

```
1  /**
2   *
3   */
4  package oop.ex6.filters;
5
6  /**
7   * Exception in case there a problem with the filter name
8   * @author roigreenberg
9   *
10  */
11 public class FilterNameErrorException extends Exception {
12     /**
13      * default constructor
14      */
15     public FilterNameErrorException() {
16         super("Not Such filter");
17     }
18     /**
19      * constructor
20      * @param exception - messege for the exception
21      */
22     public FilterNameErrorException(String exception) {
23         super(exception);
24     }
25 }
```

## 18 oop/ex6/filters/FilterParameterErrorException.java

```
1  /**
2   *
3   */
4  package oop.ex6.filters;
5
6  /**
7   * Exception in case there a problem with the filter parametres
8   * @author roigreenberg
9   *
10  */
11  public class FilterParameterErrorException extends Exception {
12      /**
13       * default constructor
14       */
15      public FilterParameterErrorException() {
16          super("wrong parameters");
17      }
18      /**
19       * constructor
20       * @param exception - messege for the exception
21       */
22      public FilterParameterErrorException(String exception) {
23          super(exception);
24      }
25  }
```



## 19 oop/ex6/filters/GreaterThanFilter.java

```
1  /**
2   *
3   */
4  package oop.ex6.filters;
5
6  import java.io.File;
7
8  /**
9   * filter file according to size greater than parameter
10   * @author roigreenberg
11   *
12   */
13  public class GreaterThanFilter extends SizeFilters implements Filter {
14      public GreaterThanFilter(double param){
15          super(param);
16      }
17      /**
18       * @see oop.ex6.filters.Filter#isPass(java.io.File)
19       * @return true iff file size greater than parameter
20       */
21      public boolean isPass(File file){
22          return (file.length() > param);
23      }
24  }
25 }
```

## 20 oop/ex6/filters/HiddenFilter.java

```
1  /**
2   *
3   */
4  package oop.ex6.filters;
5
6  import java.io.File;
7
8  /**
9   * filter file according to it's hidden state
10   * @author roigreenberg
11   *
12   */
13  public class HiddenFilter extends BooleanFilters implements Filter {
14      public HiddenFilter(String param){
15          super(param);
16      }
17      /**
18       * @see oop.ex6.filters.Filter#isPass(java.io.File)
19       * @return true iff file is/isn't (according to parameter) hidden
20       */
21      @Override
22      public boolean isPass(File file) {
23          if (yes){
24              return (file.isHidden());
25          } else {
26              return (!file.isHidden());
27          }
28      }
29  }
```

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

```
1  /**
2   *
3   */
4  package oop.ex6.filters;
5
6  import java.io.File;
7
8  /**
9   * decorator to opposite filters
10   * @author roigreenberg
11   *
12   */
13  public class NotFilter implements Filter {
14      Filter filter;
15      public NotFilter(Filter filter){
16          this.filter = filter;
17      }
18      /**
19       * @see oop.ex6.filters.Filter#isPass(java.io.File)
20       * @return - true if file DID NOT pass the filter
21       */
22      @Override
23      public boolean isPass(File file) {
24          return (!filter.isPass(file));
25      }
26
27  }
```

## 22 oop/ex6/filters/PrefixFilter.java

```
1  /**
2   *
3   */
4  package oop.ex6.filters;
5
6  import java.io.File;
7
8  /**
9   * filter file according to name start with given String
10   * @author roigreenberg
11   *
12   */
13  public class PrefixFilter extends StringFilters implements Filter {
14      public PrefixFilter(String param){
15          super(param);
16      }
17      /**
18       * @see oop.ex6.filters.Filter#isPass(java.io.File)
19       * @return true iff file name start with the given parameter String
20       */
21      @Override
22      public boolean isPass(File file) {
23
24          return (file.getName().startsWith(param));
25      }
26
27
28  }
```

## 23 oop/ex6/filters/SizeFilters.java

```
1  /**
2   *
3   */
4  package oop.ex6.filters;
5
6  import java.io.File;
7
8  /**
9   * abstract class for filter with numeric parameters
10   * @author roigreenberg
11   *
12   */
13  public abstract class SizeFilters implements Filter {
14      double param;
15      public static final int KILO = 1024;
16      public SizeFilters(double param) {
17          this.param = param*KILO;
18      }
19      /**
20       * @see oop.ex6.filters.Filter#isPass(java.io.File)
21       */
22      @Override
23      public boolean isPass(File file) {
24          // TODO Auto-generated method stub
25          return true;
26      }
27
28  }
```

should have been  
abstract

## 24 oop/ex6/filters/SmallerThanFilter.java

```
1  /**
2   *
3   */
4  package oop.ex6.filters;
5
6  import java.io.File;
7
8  /**
9   * filter file according to size smaller than parameter
10   * @author roigreenberg
11   *
12   */
13  public class SmallerThanFilter extends SizeFilters implements Filter {
14      public SmallerThanFilter(double param){
15          super(param);
16      }
17      /**
18       * @see oop.ex6.filters.Filter#isPass(java.io.File)
19       * @return true iff file size smaller than parameter
20       */
21      public boolean isPass(File file){
22
23          return (file.length() < param);
24      }
25  }
```

## 25 oop/ex6/filters/StringFilters.java

```
1  /**
2   *
3   */
4  package oop.ex6.filters;
5
6  import java.io.File;
7
8  /**
9   * abstract class for filters with String parameter
10   * @author roigreenberg
11   *
12   */
13  public abstract class StringFilters implements Filter {
14      String param;
15      public StringFilters(String param){
16          this.param = param;
17      }
18      /**
19       * @see oop.ex6.filters.Filter#isPass(java.io.File)
20       */
21      @Override
22      public boolean isPass(File file) {
23          // TODO Auto-generated method stub
24          return false;
25      }
26
27  }
```

## 26 oop/ex6/filters/SuffixFilter.java

```
1  /**
2   *
3   */
4  package oop.ex6.filters;
5
6  import java.io.File;
7
8  /**
9   * filter file according to name end with given String
10   * @author roigreenberg
11   *
12   */
13  public class SuffixFilter extends StringFilters implements Filter {
14      public SuffixFilter(String param){
15          super(param);
16      }
17      /**
18       * @see oop.ex6.filters.Filter#isPass(java.io.File)
19       * @return true iff file name end with the given parameter String
20       */
21      @Override
22      public boolean isPass(File file) {
23
24          return (file.getName().endsWith(param));
25      }
26  }
```



## 27 oop/ex6/filters/WritableFilter.java

```
1  /**
2   *
3   */
4  package oop.ex6.filters;
5
6  import java.io.File;
7
8  /**
9   * filter file according to it's writability
10   * @author roigreenberg
11   *
12   */
13  public class WritableFilter extends BooleanFilters implements Filter {
14      public WritableFilter(String param){
15          super(param);
16      }
17      /**
18       * @see oop.ex6.filters.Filter#isPass(java.io.File)
19       * @return true iff file is/isn't (according to parameter) writable
20       */
21      @Override
22      public boolean isPass(File file) {
23          if (yes){
24              return (file.canWrite());
25          } else {
26              return (!file.canWrite());
27          }
28      }
29  }
```

## 28 oop/ex6/orders/AbsOrder.java

```
1  /**
2   *
3   */
4  package oop.ex6.orders;
5
6  import java.io.File;
7  import java.util.ArrayList;
8  import java.util.Arrays;
9  import java.util.Comparator;
10
11 /**
12  * order file according to file's path
13  * @author roigreenberg
14  *
15  */
16 public class AbsOrder implements Order, Comparator<File> {
17
18     /**
19      * @see java.util.Comparator#compare(java.lang.Object, java.lang.Object)
20      * @see oop.ex6.orders.Order#compare(java.lang.Object, java.lang.Object)
21      * the comparison is according to the path of the file
22      * @return result - number indicate the order between the files
23      */
24     @Override
25     public int compare(File file1, File file2) {
26         return String.valueOf(file1.getPath())
27             .compareTo(file2.getPath());
28     }
29
30 }
```

redundant to implement both Order  
and Comparator

## 29 oop/ex6/orders/Order.java

```
1  /**
2   *
3   */
4  package oop.ex6.orders;
5
6  import java.io.File;
7  import java.util.ArrayList;
8  import java.util.Comparator;
9
10 /**
11  * interface for all the orders
12  * extends Comparator<> in order to be comparator for sorting an array
13  * @author roigreenberg
14  *
15  */
16 public interface Order extends Comparator<File>{
17     /**
18      * compare between 2 file
19      * @see java.util.Comparator#compare(java.lang.Object, java.lang.Object)
20      */
21     public int compare(File file1, File file2);
22 }
```

## 30 oop/ex6/orders/OrderErrorException.java

```
1  /**
2   *
3   */
4  package oop.ex6.orders;
5
6  /**
7   * Exception in case there a problem with the order name
8   * @author roigreenberg
9   *
10  */
11  public class OrderErrorException extends Exception {
12      /**
13       * default constructor
14       */
15      public OrderErrorException() {
16          super("Not Such Order");
17      }
18      /**
19       * constructor
20       * @param exception - messege for the exception
21       */
22      public OrderErrorException(String exception) {
23          super(exception);
24      }
25  }
```

## 31 oop/ex6/orders/OrderFactory.java

```
1  /**
2   *
3   */
4  package oop.ex6.orders;
5
6  import oop.ex6.filters.*;
7
8
9  /**
10   * this class create a order
11   * @author roigreenberg
12   *
13   */
14  public class OrderFactory {
15      static String orderName;
16      /**
17       * this method create order according to the command file
18       * @param args - the command line for creating a order
19       * @return order - the order as wrote in the command or default order
20       * if a problem has found at the command line
21       * @throws OrderErrorException -in case of problem in filter name
22       */
23      public static Order createOrder(String args)
24          throws OrderErrorException {
25          Order order = new AbsOrder();
26          String[] params = args.split("#");
27          orderName = params[0];
28          switch(orderName){
29              case("abs"):{
30                  break;
31              }
32              case("type"):{
33                  order = new TypeOrder();
34                  break;
35              }
36              case("size"):{
37                  order = new SizeOrder();
38                  break;
39              }
40              default:
41                  throw new OrderErrorException();
42              }
43          if (!args.endsWith("REVERSE")){
44              return order;
45          } else {
46              return new ReverseOrder(order);
47          }
48      }
49  }
```

## 32 oop/ex6/orders/ReverseOrder.java

```
1  /**
2   *
3   */
4  package oop.ex6.orders;
5
6  import java.io.File;
7  import java.util.Collections;
8  import java.util.Comparator;
9
10 /**
11  * decorator to reverse orders
12  * @author roigreenberg
13  *
14  */
15 public class ReverseOrder implements Order, Comparator<File> {
16     Order order;
17     public ReverseOrder (Order order){
18         this.order = order;
19     }
20     /**
21      * @see java.util.Comparator#compare(java.lang.Object, java.lang.Object)
22      * @see oop.ex6.orders.Order#compare(java.lang.Object, java.lang.Object)
23      * @return the opposite of the result from the original order
24      */
25     @Override
26     public int compare(File file1, File file2) {
27         return -1*order.compare(file1, file2);
28     }
29 }
30
31 }
```

## 33 oop/ex6/orders/SizeOrder.java

```
1  /**
2   *
3   */
4  package oop.ex6.orders;
5
6  import java.io.File;
7  import java.util.ArrayList;
8  import java.util.Arrays;
9  import java.util.Collections;
10 import java.util.Comparator;
11
12 import org.omg.PortableInterceptor.SUCCESSFUL;
13
14 /**
15  * order file according to file's size
16  * @author roigreenberg
17  *
18  */
19 public class SizeOrder implements Order, Comparator<File> {
20
21     /**
22      * @see java.util.Comparator#compare(java.lang.Object, java.lang.Object)
23      * @see oop.ex6.orders.Order#compare(java.lang.Object, java.lang.Object)
24      * the comparison is according to the size of the file
25      * in case of even, the comparison will be like absOrder
26      * @return result - number indicate the order between the files
27      */
28     @Override
29     public int compare(File file1, File file2) {
30         int result = Long.valueOf(file1.length())
31             .compareTo(file2.length());
32
33         if (result != 0)
34             return result;
35         else
36             return new AbsOrder().compare(file1, file2);
37     }
38 }
39
40
41
42
43
44
45
46
47
48 }
```

## 34 oop/ex6/orders/TypeOrder.java

```
1  /**
2   *
3   */
4  package oop.ex6.orders;
5
6  import java.io.File;
7  import java.util.ArrayList;
8  import java.util.Comparator;
9
10 /**
11  * order file according to file's type
12  * @author roigreenberg
13  *
14  */
15 public class TypeOrder implements Order, Comparator<File> {
16     /**
17      * @see java.util.Comparator#compare(java.lang.Object, java.lang.Object)
18      * @see oop.ex6.orders.Order#compare(java.lang.Object, java.lang.Object)
19      * the comparison is according to the type of the file
20      * in case of even, the comparison will be like absOrder
21      * @return result - number indicate the order between the files
22      */
23     @Override
24     public int compare(File file1, File file2){
25
26         int index1 = file1.getName().lastIndexOf(".") + 1;
27         int index2 = file2.getName().lastIndexOf(".") + 1;
28         int result = String.valueOf(file1.getName().substring(index1))
29             .compareTo(file2.getName().substring(index2));
30
31         if (result != 0)
32             return result;
33         else
34             return new AbsOrder().compare(file1, file2);
35     }
36
37 }
```