Contents

1 Basic Test Results

```
1  Logins: aviadle
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!
```

2 README

```
aviadle


=======================================
=  README for ex5:  File Processing   =
=======================================

==================================
=     List Of Submitted Files     =
==================================

  README - This file

  Package: oop.ex6.filescript
    FatalErrorException.java  -  Exception that handle Type II Error
    MyFileScript.java  -  The manager that handle everything
    Parsing.java   -   The parser that parsing every line
    WarningException.java  -  Exception for Type I errors

  Package: oop.ex6.filescript.sections
    Section.java

  Package: oop.ex6.filescript.sections.filters   -  All Filters and Factory
    FileAll.java
    FileBetween.java
    FileExecutable.java
    FileGreaterThan.java
    FileHidden.java
    FileNameContains.java
    FileNameEquals.java
    FileNameSuffix.java
    FileNamePrefix.java
    FileSmallerThan.java
    FileWritable.java
    NegFilter.java
    FilterFactory.java
    Filter.java
    WarningBetweenVariablesException.java
    WarningFilterNameException.java
    WarningYesOrNoException.java
    YesOrNoFilterParent.java  -  parent for all YES or NO filters

  Package: oop.ex6.filescript.sections.orders  -  all Orders and Factory
    AbsoluteOrder.java
    SizeOrder.java
    TypeOrder.java
    ReverseOrder.java
    Order.java
    OrderFactory.java
    WarningInOrderNameException.java

================================
=     Question asked in pdf     =
================================
1.  I'll describe the design for this project below.
2.  I implement the design the way we've shown in class.
3.  In case of a Type I error:
        I did in both factories (Filter/Order) defalut case if non of the allowed
        sequence was entered, to throw warning, and also in some special filters that
```

need checking of the values i throw warning within the constructor. then i catched
all Warnings in the Parsing proccess and handle them this way:
  - warning in Filter: - remember the line we're now checking (with array).
                       - define Filter to default (Filter "all")
  - warning in Order: - remember the line we're now checking (with array).
                       - define Order to default (Order "abs")
then, continue Parsing. I used this handling because I wanted to keep all warnings
with its sections, so I'll be able to print as instructed (Warning section1,
section1, Warning section2, section2...).
In case of a Type II errors:
    I threw FatalError in Factory process, so if a Type II error has found,
    it will throw Exception to parsing, and then it will stop Parsing and
    immediately will throw again up to the main script, will print "ERROR",
    and stop program.
4. (I hope i understood the question correctly)
   I used Comparator in order to sort an array of files.

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

  The main script is MyFileScript that get 2 arguments, directory and command file.
then it send the command file for Parsing.
  The Parsing process use Scanner in order to iterate through all lines in file, and
according to the line it check and parse. if the string fo filter or order is found,
we send him to factory.
  Both Factories works pretty much the same: we get the string, split and check which
filter we've got, and if NOT/REVERSE is found. if filter is found, Great. if not,
use Exception as i described above.
  The parsing returned all sections after parsing in ArrayList (the easiest way i
found to work with unknown size of array. maybe there is better way, and i would be
happy to find out :-)).
  Then in the main file we iterate through all the sections: 1st check if any warning
was found in parsing, then print the section with the filter and order we have.
  THE END!!

3 oop/ex6/filescript/FatalErrorException.java

```java
package oop.ex6.filescript;

public class FatalErrorException extends Exception {
    private static final long serialVersionUID = 1L;

    /**
     * Constructor with massage as instructed.
     */
    public FatalErrorException() {
        super("ERROR");
    }
}
```

4 oop/ex6/filescript/MyFileScript.java

```java
package oop.ex6.filescript;

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

import oop.ex6.filescript.sections.Section;

public class MyFileScript {

    /**
     * The manager that run the program
     *
     * @param args string array with 2 arguments. the first is the directory
     * where all the files we need to filter and sort is in, and the 2nd is the
     * command file.
     */
    public static void main(String[] args) {
        try {
            File directory = new File(args[0]);
            File[] listOfFiles = directory.listFiles();
            File[] sortFile;
            //send the command file to parsing.
            ArrayList<Section> sections = Parsing.parseFile(new File(args[1]));
            //check if warning is found in this section
            for(int i=0; i<sections.size(); i++) {
                for(int k=0; k<sections.get(i).getWarnings().length; k++) {
                    if(sections.get(i).getWarnings()[k] != 0) {
                        System.out.println("Warning in line "+
                                String.valueOf(sections.get(i).getWarnings()[k]));
                    }
                }
                //send all files to sorting
                sortFile = sections.get(i).getOrder().sortFile(listOfFiles);
                //check if file pass filter, if yes print
                for(int j=0; j<sortFile.length; j++) {
                    if(sortFile[j].isFile()) {
                        if(sections.get(i).getFilter().isPass(sortFile[j])) {
                            System.out.println(sortFile[j].getName());
                        }
                    }
                }
            }
        } catch (FatalErrorException e) {
            System.err.println(e.getMessage());
        }

    }

}
```

5 oop/ex6/filescript/Parsing.java

```java
package oop.ex6.filescript;

import java.io.*;
import java.util.ArrayList;
import java.util.Scanner;

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


public class Parsing{

    private static ArrayList<Section> sections;
    private static Scanner scanner;
    private static final int LINE_IN_SECTION = 4;
    private static final String FILTER = "FILTER", ORDER = "ORDER";


    /**
     * @param commandFile The file contain sections we need to parse.
     * @return sections after parsing.
     * @throws FatalError in case of Type I Error.
     */
    public static ArrayList<Section> parseFile(File commandFile) throws FatalErrorException {
        try {
            sections  = new ArrayList<Section>();
            Filter filter = null;
            Order order;
            String line;
            int[] warnings = new int[2]; //Initialize to 2 cells, so if any Type II
                                         //error happens, keep its line in matched cell.

            //we'll use in case we skipped line (FILTER right after ORDER)
            int lineSkipped = 0;

            scanner = new Scanner(commandFile);
            //run until no next line and no next section.
            for(int lineNumber=1; scanner.hasNext() ||
                    lineNumber%LINE_IN_SECTION != 1; lineNumber++) {
                if(scanner.hasNext()) {
                    line = scanner.next();
                } else {     //in case of empty line in last section
                    line = "";
                }
                //check what part of section line we're in
                switch (lineNumber % LINE_IN_SECTION) {
                    case 1:
                        if(!line.equals(FILTER))
                            throw new FatalErrorException();
                        break;
                    case 2:
                        try {
                            filter = FilterFactory.createFilter(line);
                        } catch (WarningException e) {
                            warnings[0] = lineNumber - lineSkipped;
                            filter = new FileAll();
                        }
                        break;
```

```
60                      case 3:
61                          if(!line.equals(ORDER))
62                              throw new FatalErrorException();
63                          break;
64                      case 0:
65                          try{
66                              order = OrderFactory.createOrder(line);
67                          } catch(WarningException e) {
68                              order = new AbsoluteOrder();
69                              //check if no line after order
70                              if(line.equals(FILTER)) {
71                                  lineNumber++;  //so we'll not confuse the sections lines
72                                  lineSkipped++;  //let's remember we skipped line
73                              } else{  //if no FILTER that mean Type II error happened
74                                  warnings[1] = lineNumber - lineSkipped;
75                              }
76                          }
77                          sections.add(new Section(filter,order,warnings));
78                          warnings = new int[2];
79                          break;
80                  }
81              }
82
83              return sections;
84          } catch (FileNotFoundException e) {
85              throw new FatalErrorException();
86          }
87      }
88  }
```

6 oop/ex6/filescript/WarningException.java

```java
package oop.ex6.filescript;

public class WarningException extends Exception {

    private static final long serialVersionUID = 1L;

}
```

7 oop/ex6/filescript/sections/Section.java

```java
package oop.ex6.filescript.sections;

import oop.ex6.filescript.sections.filters.Filter;
import oop.ex6.filescript.sections.orders.Order;



public class Section {

    private Filter _filter;
    private Order _order;
    private int[] _warnings;

    /**
     * @param filter this section filter.
     * @param order this section order.
     * @param warnings if Type II error happened, the line is inside the array.
     */
    public Section(Filter filter, Order order, int[] warnings) {
        _filter = filter;
        _order = order;
        _warnings = warnings;
    }

    /**
     * @return the _filter
     */
    public Filter getFilter() {
        return _filter;
    }

    /**
     * @param _filter the _filter to set
     */
    public void setFilter(Filter _filter) {
        this._filter = _filter;
    }

    /**
     * @return the _order
     */
    public Order getOrder() {
        return _order;
    }

    /**
     * @param _order the _order to set
     */
    public void setOrder(Order _order) {
        this._order = _order;
    }

    /**
     * @return the _warnings
     */
    public int[] getWarnings() {
        return _warnings;
    }

```

```
60      /**
61       * @param _warnings the _warnings to set
62       */
63      public void setWarnings(int[] _warnings) {
64          this._warnings = _warnings;
65      }
66  }
```

8 oop/ex6/filescript/sections/filters/FileAll.java

```java
package oop.ex6.filescript.sections.filters;

import java.io.File;

public class FileAll implements Filter {

    /**
     * every file is passing, so always return true.
     *
     * @param file the file we need to check if pass the filter
     */
    @Override
    public boolean isPass(File file) {
        return true;
    }

}
```

9 oop/ex6/filescript/sections/filters/FileBetween.java

```java
package oop.ex6.filescript.sections.filters;

import java.io.File;

public class FileBetween implements Filter {

    private final static int CONVERT_BYTES = 1024;
    private double maxLimit, minLimit;

    public FileBetween(double firstNumber, double secondNumer) throws WarningBetweenVariablesException {
        if(firstNumber >= secondNumer)
            throw new WarningBetweenVariablesException();
        this.minLimit = firstNumber;
        this.maxLimit = secondNumer;
    }

    /**
     * @param file the file we need to check if pass the filter.
     */
    @Override
    public boolean isPass(File file) {
        return ((file.length() / CONVERT_BYTES) >= minLimit) &&
                ((file.length() / CONVERT_BYTES) <= maxLimit);
    }

}
```

10 oop/ex6/filescript/sections/filters/FileExecutable.ja

```java
package oop.ex6.filescript.sections.filters;

import java.io.File;

public class FileExecutable extends YesOrNoFilterParent implements Filter {

    /**
     * if YES, trueOrFalse is true. if NO trueOrFalse is false.
     *
     * @param yesNo the string of YES \ NO
     * @throws Warning
     */
    public FileExecutable(String yesNo) throws WarningYesOrNoException {
        super(yesNo);
    }

    /**
     * @param file the file we need to check if pass the filter.
     */
    @Override
    public boolean isPass(File file) {
        return (file.canExecute() == trueOrFalse);
    }
}
```

11 oop/ex6/filescript/sections/filters/FileGreaterThan.

```java
package oop.ex6.filescript.sections.filters;

import java.io.File;

public class FileGreaterThan implements Filter{

    private final static double CONVERT_BYTES = 1024;
    private double minimumLimit;

    public FileGreaterThan(double number) {
        this.minimumLimit = number;
    }

    /**
     * @param file the file we need to check if pass the filter.
     */
    @Override
    public boolean isPass(File file) {
        return (file.length() / CONVERT_BYTES) > minimumLimit;
    }



}
```

12 oop/ex6/filescript/sections/filters/FileHidden.java

```java
package oop.ex6.filescript.sections.filters;

import java.io.File;

public class FileHidden extends YesOrNoFilterParent implements Filter {

    /**
     * if YES, trueOrFalse is true. if NO trueOrFalse is false.
     *
     * @param yesNo the string of YES \ NO
     * @throws Warning
     */
    public FileHidden(String yesNo) throws WarningYesOrNoException {
        super(yesNo);
    }

    /**
     * @param file the file we need to check if pass the filter.
     */
    @Override
    public boolean isPass(File file) {
        return (file.isHidden() == trueOrFalse);
    }
}
```

13 oop/ex6/filescript/sections/filters/FileNameContain

```java
package oop.ex6.filescript.sections.filters;

import java.io.File;

public class FileNameContains implements Filter {

    private String nameContains;

    public FileNameContains(String name) {
        this.nameContains = name;
    }

    /**
     * @param file the file we need to check if pass the filter.
     */
    @Override
    public boolean isPass(File file) {
        return file.getName().contains(nameContains);
    }

}
```

14 oop/ex6/filescript/sections/filters/FileNameEquals.j

```java
package oop.ex6.filescript.sections.filters;

import java.io.File;

public class FileNameEquals implements Filter {

    private String nameEquals;

    public FileNameEquals(String name) {
        this.nameEquals = name;
    }

    /**
     * @param file the file we need to check if pass the filter.
     */
    @Override
    public boolean isPass(File file) {
        return nameEquals.equals(file.getName());
    }

}
```

15 oop/ex6/filescript/sections/filters/FileNamePrefix.j

```java
package oop.ex6.filescript.sections.filters;

import java.io.File;

public class FileNamePrefix implements Filter {

    private String preffix;

    public FileNamePrefix(String pref) {
        this.preffix = pref;
    }

    /**
     * @param file the file we need to check if pass the filter.
     */
    @Override
    public boolean isPass(File file) {
        return file.getName().startsWith(preffix);
    }

}
```

16 oop/ex6/filescript/sections/filters/FileNameSuffix.ja

```java
package oop.ex6.filescript.sections.filters;

import java.io.File;

public class FileNameSuffix implements Filter {

    private String suffix;

    public FileNameSuffix(String suff) {
        this.suffix = suff;
    }

    /**
     * @param file the file we need to check if pass the filter.
     */
    @Override
    public boolean isPass(File file) {
        return file.getName().endsWith(suffix);
    }
}
```

17 oop/ex6/filescript/sections/filters/FileSmallerThan.

```java
package oop.ex6.filescript.sections.filters;

import java.io.File;

public class FileSmallerThan implements Filter {

    private final static int CONVERT_BYTES = 1024;
    private double maximumLimit;

    public FileSmallerThan(double number) {
        this.maximumLimit = number;
    }

    /**
     * @param file the file we need to check if pass the filter.
     */
    @Override
    public boolean isPass(File file) {
        return (file.length() / CONVERT_BYTES) < maximumLimit;
    }

}
```

18 oop/ex6/filescript/sections/filters/FileWritable.java

```java
package oop.ex6.filescript.sections.filters;

import java.io.File;

public class FileWritable extends YesOrNoFilterParent implements Filter {

    /**
     * if YES, trueOrFalse is true. if NO trueOrFalse is false.
     *
     * @param yesNo the string of YES \ NO
     * @throws Warning
     */
    public FileWritable(String yesNo) throws WarningYesOrNoException {
        super(yesNo);
    }

    @Override
    public boolean isPass(File file) {
        return (file.canWrite() == trueOrFalse);
    }
}
```

19 oop/ex6/filescript/sections/filters/Filter.java

```java
package oop.ex6.filescript.sections.filters;

import java.io.File;

public interface Filter {
    boolean isPass(File file);
}
```

20 oop/ex6/filescript/sections/filters/FilterFactory.java

```java
package oop.ex6.filescript.sections.filters;

import oop.ex6.filescript.WarningException;

public class FilterFactory {

    private static final String SEPEATOR = "#",    NOT = "NOT";

    /**
     * @param filterString string we need to convert to Filter.
     * @return the Filter the string represent.
     * @throws Warning
     */
    public static Filter createFilter(String filterString) throws WarningException {
        boolean isNegative = false;
        Filter filter = null;

        String[] array = filterString.split(SEPEATOR); //split the string with #

        //check if NOT in the end of the string
        if(array[array.length - 1].equals(NOT))
            isNegative = true;

        //check all cases given and send object according to what needed.
        switch (array[0]) {
            case "greater_than":
                filter = new FileGreaterThan(Double.valueOf(array[1]));
                break;
            case "smaller_than":
                filter = new FileSmallerThan(Double.valueOf(array[1]));
                break;
            case "between":
                //check if the first value is smaller. if not filter remain null.
                try {
                    filter = new FileBetween(Double.valueOf(array[1]),
                            Double.valueOf(array[2]));
                } catch (WarningBetweenVariablesException e) {
                    throw new WarningException();
                }
                break;
            case "file":
                filter = new FileNameEquals( array[1]);
                break;
            case "contains":
                filter = new FileNameContains( array[1]);
                break;
            case "prefix":
                filter = new FileNamePrefix( array[1]);
                break;
            case "suffix":
                filter = new FileNameSuffix( array[1]);
                break;
            case "writable":
                //check if YES / NO is spelled correctly
                try {
                    filter = new FileWritable( array[1]);
                } catch (WarningYesOrNoException e) {
                    throw new WarningException();
                }
```

```
60                  break;
61              case "executable":
62                  //check if YES / NO is spelled correctly
63                  try {
64                      filter = new FileExecutable( array[1]);
65                  } catch (WarningYesOrNoException e) {
66                      throw new WarningException();
67                  }
68                  break;
69              case "hidden":
70                  //check if YES / NO is spelled correctly
71                  try {
72                      filter = new FileHidden( array[1]);
73                  } catch (WarningYesOrNoException e) {
74                      throw new WarningException();
75                  }
76                  break;
77              case "all":
78                  filter = new FileAll();
79                  break;
80              default:
81                  throw new WarningException();
82          }
83
84          if(isNegative && filter != null)
85              filter = new NegFilter(filter);
86
87          return filter;
88      }
89  }
```

21 oop/ex6/filescript/sections/filters/NegFilter.java

```java
package oop.ex6.filescript.sections.filters;

import java.io.File;

public class NegFilter implements Filter {

    private Filter filterToNegation;

    public NegFilter(Filter filterToNeg) {
        this.filterToNegation = filterToNeg;
    }

    /**
     * return the opposite of the filter we need to negation.
     *
     * @param file the file we need to check if pass the filter
     */
    @Override
    public boolean isPass(File file) {
        return !this.filterToNegation.isPass(file);
    }

}
```

22 oop/ex6/filescript/sections/filters/WarningBetween

```java
package oop.ex6.filescript.sections.filters;

import oop.ex6.filescript.WarningException;

public class WarningBetweenVariablesException extends WarningException {

    private static final long serialVersionUID = 1L;

}
```

23 oop/ex6/filescript/sections/filters/WarningInFilterN

```java
package oop.ex6.filescript.sections.filters;

import oop.ex6.filescript.WarningException;

public class WarningInFilterNameException extends WarningException {

    private static final long serialVersionUID = 1L;

}
```

24 oop/ex6/filescript/sections/filters/WarningYesOrNo

```java
package oop.ex6.filescript.sections.filters;

import oop.ex6.filescript.WarningException;

public class WarningYesOrNoException extends WarningException {

    private static final long serialVersionUID = 1L;

}
```

25 oop/ex6/filescript/sections/filters/YesOrNoFilterPa

```java
package oop.ex6.filescript.sections.filters;

public class YesOrNoFilterParent {

    protected boolean trueOrFalse;
    private static final String YES = "YES", NO = "NO";

    public YesOrNoFilterParent(String yesNo) throws WarningYesOrNoException {
        if(yesNo.equals(YES)) {
            trueOrFalse = true;
        } else if (yesNo.equals(NO)) {
            trueOrFalse = false;
        } else {
            throw new WarningYesOrNoException();
        }
    }
}
```

26 oop/ex6/filescript/sections/orders/AbsoluteOrder.ja

```java
package oop.ex6.filescript.sections.orders;

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

public class AbsoluteOrder implements Order {

    /**
     * @param files the array of files to sort.
     * @return Comparator that sort by name.
     */
    public static Comparator<File> sortByName(File[] files) {
        Comparator<File> comparator = new Comparator<File>() {

            @Override
            public int compare(File o1, File o2) {
                return (o1.getName().compareTo(o2.getName()));
            }
        };
        return comparator;
    }

    /**
     * @param files the files we sorting.
     */
    @Override
    public File[] sortFile(File[] files) {
        Arrays.sort(files,sortByName(files));
        return files;
    }
}
```

27 oop/ex6/filescript/sections/orders/Order.java

```java
package oop.ex6.filescript.sections.orders;

import java.io.File;

public interface Order {
    File[] sortFile(File[] files);
}
```

28 oop/ex6/filescript/sections/orders/OrderFactory.jav

```java
package oop.ex6.filescript.sections.orders;

public class OrderFactory {

    private static final String SEPEATOR = "#",    REVERSE = "REVERSE";

    /**
     * @param orderString string we need to convert to Order.
     * @return the Order the string represent.
     * @throws Warning
     */
    public static Order createOrder(String orderString) throws WarningInOrderNameException {
        boolean isReverse = false;
        Order order = null;

        String[] array = orderString.split(SEPEATOR); //split the string with #
        //check if REVERSE in the end of the string
        if(array[array.length - 1].equals(REVERSE))
            isReverse = true;

        //check all cases given and send object according to what needed.
        switch(array[0]) {
            case "abs":
                order = new AbsoluteOrder();
                break;
            case "type":
                order = new TypeOrder();
                break;
            case "size":
                order =  new SizeOrder();
                break;
            case "":  //if no line entered in section after ORDER.
                order = new AbsoluteOrder();
                break;
            default:
                throw new WarningInOrderNameException();
        }

        if(isReverse)
            order = new ReverseOrder(order);

        return order;
    }
}
```

29 oop/ex6/filescript/sections/orders/ReverseOrder.jav

```java
package oop.ex6.filescript.sections.orders;

import java.io.File;

public class ReverseOrder implements Order {

    private Order orderToReverse;

    public ReverseOrder(Order reverse) {
        this.orderToReverse = reverse;
    }

    /**
     * @param files the files we sorting.
     */
    @Override
    public File[] sortFile(File[] files) {
        File[] reverse = new File[files.length];
        files = this.orderToReverse.sortFile(files);
        //reverse the order of the files.
        for(int i=0; i<files.length;i++) {
            reverse[i] = files[files.length - i - 1];
        }
        return reverse;
    }
}
```

30 oop/ex6/filescript/sections/orders/SizeOrder.java

```java
package oop.ex6.filescript.sections.orders;

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

public class SizeOrder implements Order {

    /**
     * @param files the array of files to sort.
     * @return Comparator that sort by size.
     */
    public static Comparator<File> sortBySize(File[] files) {
        Comparator<File> comparator = new Comparator<File>() {

            @Override
            public int compare(File o1, File o2) {
                if(o1.length() - o2.length() > 0)
                    return 1;
                else if(o1.length() - o2.length() < 0)
                    return -1;
                else
                    return 0;
            }
        };
        return comparator;
    }

    /**
     * first sort Alphabetic, then by size.
     *
     * @param files the files we sorting.
     */
    @Override
    public File[] sortFile(File[] files) {
        Arrays.sort(files, AbsoluteOrder.sortByName(files));
        Arrays.sort(files,sortBySize(files));
        return files;
    }
}
```

31 oop/ex6/filescript/sections/orders/TypeOrder.java

```java
package oop.ex6.filescript.sections.orders;

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

public class TypeOrder implements Order {

    private static final char DOT = '.';

    /**
     * @param files the array of files to sort.
     * @return Comparator that sort by type.
     */
    public static Comparator<File> sortByType(File[] files) {
        Comparator<File> comparator = new Comparator<File>() {

            @Override
            public int compare(File o1, File o2) {
                return (getType(o1).compareTo(getType(o2)));
            }
        };
        return comparator;
    }

    /**
     * @param name file we want its type.
     * @return the type of the file
     */
    private static String getType(File name) {
        String type = "";
        if(!name.getName().endsWith(Character.toString(DOT))) { //check if no ending, just dot
            //run from the end of the file name until we meet '.'
            for(int i=name.getName().length() - 1; i >= 0; i--) {
                type = String.valueOf(name.getName().charAt(i)) + type;
                if(name.getName().charAt(i) == DOT)
                    break;
            }
        }
        return type;
    }

    /**
     * first sort Alphabetic, then by type.
     *
     * @param files the files we sorting.
     */
    @Override
    public File[] sortFile(File[] files) {
        Arrays.sort(files, AbsoluteOrder.sortByName(files));
        Arrays.sort(files,sortByType(files));
        return files;
    }
}
```

32 oop/ex6/filescript/sections/orders/WarningInOrderN

```java
package oop.ex6.filescript.sections.orders;

import oop.ex6.filescript.WarningException;

public class WarningInOrderNameException extends WarningException {

    private static final long serialVersionUID = 1L;

}
```