1. Homework 8

Posted: October/13/2018
Due: October/21/2018 24.00

All homework solutions are due October/21/2018 24.00. I recommend to submit at least one version of all homework solutions long before due date.

1.1. Homework 8.1 (10 Points)

Objective: Reading a file

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to 80% if your solution is poorly designed
Testing: You can lose up to 50% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:

You have to implement a program which can read a file from stdin or from the file system. You must analyze your implementation based on speed.

Explanation:

Given are the following files:

```
% ssh glados
# ... password credentials
% ls -l /home/fac/hpb/public_html/Lectures/Src_28/
total 1253117
-rw-r--r-- 1 hpb fac 1000000002 Oct 12 09:13 pi-billion.txt
-rw-r--r-- 1 hpb fac  469229415 Oct 12 09:13 pi-billion.txt.gz
```

This file contains pi. Your program has to count how many even and odd numbers this file contains. The character '.' is not a number.

You program must be able to read a compressed/uncompressed file, if the file name is given as command-line. A compressed file name will end with '.gz'. You have to read from stdin, if no file name is specified on the command line. Files read from stdin are not compressed.

Your Work:

Requirements:

You have to name your file *PiEvenOddImprovement.java*. Your program has to throw an exception if error conditions arise; like an empty file (EmptyFileException), or the file does not incude any numbers (NoNumbersException). You have to create these exception classes.

Example:

An example of a solution execution:

Let's assume the following:

```
% cat pi.txt
3.14
% gzip pi.txt
% ls -l pi.t*
-rw-------  1 hpb  staff   5 Oct 12 09:36 pi.txt
-rw-------  1 hpb  staff  32 Oct 12 09:35 pi.txt.gz
% java PiEvenOddImprovement pi.txt.gz
even = 3
```

```
odd  = 2
odd/even  = 0.6666666666666666
% java PiEvenOddImprovement pi.txt
even = 3
odd  = 2
odd/even  = 0.6666666666666666
% cat pi.txt | java PiEvenOddImprovement
even = 3
odd  = 2
odd/even  = 0.6666666666666666

% echo "ab" | java PiEvenOddImprovement
NoNumbersException: null is empty
even = 0
odd  = 0
odd/even  = NaN

% echo "" | java PiEvenOddImprovement
args.length: 0
EmptyFileException: null is empty
even = 0
odd  = 0
odd/even  = NaN
```

My code executes like this (I deleted the results):

```
% time java PiEvenOddImprovement pi-billion.txt.gz
       14.08 real        12.67 user         1.38 sys
 ...
% time java PiEvenOddImprovement pi-billion.txt
        7.75 real         5.93 user         0.70 sys
 ...
% time cat pi-billion.txt | java PiEvenOddImprovement
 ...
real0m6.150s
user0m5.996s
sys 0m1.284s
```

Submission:

```
% ssh glados.cs.rit.edu # or use queeg.cs.rit.edu if glados is down
# password
# go to the directory where your solution is ...
% try hpb-grd lab8-1 ’All files required’
```

Solution:

(This solution serves as the basis for the discussion in class. Sometimes there will be errors introduced to show common mistakes)

```

import java.util.zip.GZIPInputStream;
import java.io.*;

public class PiEvenOddImprovement extends Thread {

    static int MAX_THREADS = 6; // 8;
    String fileName = null;
    final int MAX = 102400;
    char buf[] = new char[MAX];
    int soManyCharactersRead = 0;
    int index;
    long from = 0;
    long to  = 0;
    long even = 0;
    long odd  = 0;
    boolean   emptyFile = true;
    static long length     = 0;
    String id;


    public PiEvenOddImprovement() {
    }
    public PiEvenOddImprovement(String id, String fileName, long from, long to) {

        this();
        this.id = id;
        this.fileName = fileName;
        this.from = from;
        this.to   = to;
    }
    public PiEvenOddImprovement(String fileName) {
        this();
        this.fileName = fileName;
    }
    private int nextBlock(Reader input ) throws IOException     {
        return input.read(buf, 0, MAX);
    }
    public void processChar(char aChar) throws NoNumbersException       {
                if ( ! ( (int)aChar <= 0 ) && ( (int)aChar <= 9 ) )
                        throw new NoNumbersException("not a number " + aChar );
                if ( (int)aChar % 2 == 0 )
                        even++;
                else
                        odd++;
    }
    public  void setT(int nThreads) {
        MAX_THREADS = nThreads; // 8;
    }
```

```
        private long length(String name) {
            File aFile = new File(fileName);
            return aFile.length();
        }
        public String toString()    {
            return  "id:                      \n"   +
                    "even = " + even        + "\n"  +
                    "odd  = " + odd         + "\n"  +
                    "odd + even  = " + (odd + even) +  "\n" +
                    "odd/even  = " + (double)odd/(double)even ;
        }
        public Reader openFile(String fileName) throws IOException  {
            Reader input = null;

            if (  fileName == null  ) {
                    input = new InputStreamReader(System.in);
            } else {
                    if ( fileName.endsWith(".gz") ) {
                            input = new BufferedReader(new InputStreamReader( new GZIP
                    } else  {
                            System.out.println(length(fileName) );
                            System.exit(0);
                            input = new BufferedReader(new InputStreamReader( new File
                    }
            }
            return input;
        }

        public void run()   {
                    try     (
                        Reader input = new BufferedReader(new InputStreamReader( new F
                    ) {
                                    input.skip(from);
                                    long index = from;
                                    while (  ( soManyCharactersRead = nextBlock(input)
                                            for (int counter = 0; counter < soManyChar
                                                    index ++;
                                                    if ( index >= to ) {
                                                            return;
                                                    } else {
                                                            processChar(buf[counter]);
                                                    }
                                            }
                                    }
                    } catch ( NoNumbersException e )        {
                            System.err.println(e);
                    } catch ( IOException e )       {
                            System.err.println(e);
                    }
        }
        public void startProcesses(String fileName) {
            length = length(fileName);
            PiEvenOddImprovement[] theThreads = new PiEvenOddImprovement[MAX_THREADS];
            long delta = length / MAX_THREADS;
```

```
                long from = 0;
                long to = delta;
                for ( int index = 0; index < MAX_THREADS - 1; index ++ )        {
                        theThreads[index] = new PiEvenOddImprovement("" + index  , fileNam
                        from += delta;
                        to += delta;
                        theThreads[index].start();
                }
                theThreads[MAX_THREADS - 1 ] = new PiEvenOddImprovement( "" + (MAX_THREADS
                theThreads[MAX_THREADS - 1 ].start();
                try {
                        for ( int index = 0; index < MAX_THREADS; index ++ )    {
                                theThreads[index].join();
                                even += theThreads[index].even;
                                odd += theThreads[index].odd;

                        }
                } catch ( Exception e ) {}


            }
            public PiEvenOddImprovement doTheWork() {
                if ( ! ( ( fileName == null ) || fileName.endsWith(".gz") ) ) {
                        startProcesses(fileName);
                } else {
                        try     (
                            Reader input = openFile(fileName);
                        ) {
                                    while (  ( soManyCharactersRead = nextBlock(input)
                                            if ( soManyCharactersRead == 0 )
                                                    continue;
                                            for ( int index = 0; index < soManyCharact
                                                    processChar(buf[index]);
                                            }
                                    }
                        } catch ( NoNumbersException e )        {
                                System.err.println(e);
                                System.exit(1);
                        } catch ( IOException e )       {
                                System.err.println(e);
                                System.exit(1);
                        }
                }

                return this;
            }

            public static void test(String[] args) {
                for ( int nThreads = 1; nThreads < 22; nThreads ++ )    {
                        long start = System.currentTimeMillis();
                        PiEvenOddImprovement aPiEvenOddImprovement = new PiEvenOddImprovem
                        aPiEvenOddImprovement.setT(nThreads);
                        aPiEvenOddImprovement.doTheWork();
                        long end = System.currentTimeMillis();
                        System.out.println(nThreads + ":        " + (end - start) );
                }
            }
            public static void main(String[] args) {
                        // test(args);

                        PiEvenOddImprovement aPiEvenOddImprovement = new PiEvenOddImprovem
                        aPiEvenOddImprovement.doTheWork();
            }
        }
```

Source Code: Src/28_sol/PiEvenOddImprovement.java

1.2. Homework 8.2 (10 Points)

Objective: Reading a file and extracting information based on matches.

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to 80% if your solution is poorly designed
Testing: You can lose up to 50% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:

You have to implement a program which functions like Your program must only implement a subset of *grep's* functionlity.

Explanation:

(copied and slighly modified from the manual page) The grep utility searches any given input files, selecting lines that match one or more patterns. By default, a pattern matches an input line if the regular expression in the pattern matches the input line without its trailing newline. An empty expression matches every line. Each input line that matches at least one of the patterns is written to the standard output.

...

Patterns may consist of one or more lines, allowing any of the pattern lines to match a portion of the input.

Command-line arguments control the particalur funcitonlity of *grep*.

Your Work:

In the manual page, the line

```
grep [OPTIONS] PATTERN [FILE...]
```

describes how grep can be used. More than one argument might be used for a particular call.

You have to implmentent the following arguments:

- -c
- -l
- -w
- -q

Requirements:

You have to name your file *Grep.java*.

Example:

An example of a solution execution:

```
% cat input.txt
one one one one
one
two
two
oneoneoneone
% grep -c one input.txt
3
% grep -w -c one input.txt # -w specifies words
2
% grep -w -c -s one input.txt
2
% grep -q one input.txt
echo $?
0
% grep -q three input.txt
% echo $?
1
% grep -l one input.txt
input.txt
% grep one input.txt - input.txt
input.txt:one one one one
input.txt:one
input.txt:oneoneoneone
one
(standard input):one
input.txt:one one one one
input.txt:one
input.txt:oneoneoneone
```

Submission:

```
% ssh glados.cs.rit.edu # or use queeg.cs.rit.edu if glados is down
# password
# go to the directory where your solution is ...
% try hpb-grd lab8-2 ’All files required’
```

Solution:

(This solution serves as the basis for the discussion in class. Sometimes there will be errors introduced to show common mistakes)

```
import java.io.*;
import java.util.regex.Pattern;


public class Grep {

  boolean cValue                = false;
  boolean lValue                = false;
  boolean wValue                = false;
  boolean qValue                = false;
  int     nMinusArgs            = 0;
  String searchString           = "";
```

```

public BufferedReader openFile( String fileName) throws FileNotFoundException  {
      BufferedReader input;
      if ( fileName.equals("-") )
              input =  new BufferedReader(new InputStreamReader(System.in));
      else
              input =  new BufferedReader( new FileReader(fileName));
      return input;
}
public void processFile( String fileName) {
      int counter = 0;
      try (
              BufferedReader input = openFile(fileName);
      ) {

              String line;
              String thePattern;
              while ( ( line = input.readLine() )  != null ) {
                      thePattern = wValue ? ".*(^|\\W)" + searchString + "(\\W|$
                      if ( Pattern.matches(thePattern, line ) )       {
                              if ( qValue )   {
                                      System.exit(0);
                              }
                              if ( lValue )   {
                                      System.out.println(fileName);
                                      System.exit(0);
                              }
                              if ( cValue )   {
                                      counter ++;
                              } else {
                                      System.out.println(line);
                              }
                      }
              }
              if ( cValue )
                      System.out.println(counter);
              input.close();
              }
      catch ( FileNotFoundException e)        {
          System.out.println(e.getMessage());
       }
       catch ( IOException e) {
              System.out.println(e.getMessage());
       }
       catch ( Exception e)   {
           System.out.println("ExceptionType occurred: " +
              e.getMessage() );
       }
}
public void openFiles( String args[] ) {
      searchString = args[nMinusArgs - 1 ];
      for ( int index = nMinusArgs; index < args.length; index ++ )
              processFile(args[index] );
}
```

```
        public void parseArgs( String args[] ) {

              for ( int index = 0; index < args.length; index ++ )    {
                      if ( args[index].startsWith("-c") )     {
                              cValue = true; nMinusArgs++;
                      }
                      if ( args[index].startsWith("-l") )     {
                              lValue = true; nMinusArgs++;
                      }
                      if ( args[index].startsWith("-w") )     {
                              wValue = true; nMinusArgs++;
                      }
                      if ( args[index].startsWith("-q") )     {
                              qValue = true; nMinusArgs++;
                      }
              }


              System.out.println("nMinusArgs++ = " + nMinusArgs++ );
        }
        public String toString()      {
              return (
                      "\t\n" +
                      "\tcValue = " + cValue + "\n" +
                      "\tlValue = " + lValue + "\n" +
                      "\twValue = " + wValue + "\n" +
                      "\tqValue = " + qValue
                      );
        }
        public static void main( String args[] ) {

          Grep aGrep = new Grep();
          aGrep.parseArgs(args);
          System.out.println("aGrep = " + aGrep );
          aGrep.openFiles(args);
        }
      }
```

Source Code: Src/28_sol/Grep.java

1.3. Homework 8.3 (10 Points)

Objective: Converting a program to a multi threaded program

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to 80% if your solution is poorly designed
Testing: You can lose up to 50% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:

Given is a program which determines for every pixel a color value. The colors of two different pixels are independent of each other.

Explanation:

Take a look at the following class:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.image.ImageView;
import javafx.scene.image.PixelWriter;
import javafx.scene.image.WritableImage;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class MandelbrotFX extends Application {

    WritableImage mandelBrotSetImage;
    final int IMG_WIDTH         = 800;
    final int IMG_HEIGHT        = 800;
    long milliSeconds;

    public void init()  {
        milliSeconds = System.currentTimeMillis();
    }
    public void end(String s)   {
        System.err.println(s + ":       " + ( System.currentTimeMillis() - milliSe
        System.err.println(" # of cores" +   ":       " +
        Runtime.getRuntime().availableProcessors() );
    }

    public void start(Stage theStage) {

        MandelbrotSet aMandelbrotSet = new MandelbrotSet(IMG_WIDTH, IMG_HEIGHT);

        init();
                mandelBrotSetImage = aMandelbrotSet.createImage();
        end("Multiple Thread MandelbrotSet Test");


        ImageView aImage = new ImageView();
        aImage.setImage(mandelBrotSetImage);

        StackPane root = new StackPane();
        root.getChildren().add(aImage);

        Scene scene = new Scene(root, IMG_WIDTH, IMG_HEIGHT);

        theStage.setTitle("Mandelbrot Set");
        theStage.setResizable(false);
        theStage.setScene(scene);
        theStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}


class MandelbrotSet extends Thread {

    private static final int    MAX_COLORS      = 256;
    private static final double BOUNDERY = 1000;
    private static int    width;
    private static int    height;
    private static WritableImage mandelBrotSetImage;
    private static PixelWriter aPixelWriter;
    private static final Color[] colors = new Color[MAX_COLORS];
    private static double minR  = -2.4;
    private static double maxR  = 0.9;
    private static double minI  = -1.3;
    private static double maxI  = 1.28;
    private static MandelbrotSet[] allThreads;

    static {
        for (int index = 0; index < colors.length; index++) {
            colors[index] = Color.RED.interpolate(Color.BLUE, (( 1.0 / colors.leng
        }
    }

    public MandelbrotSet() {
    }
    public MandelbrotSet(int width,int height) {
        this.width = width;
        this.height = height;
        mandelBrotSetImage = new WritableImage(width, height);
        if ( allThreads == null )
                allThreads = new MandelbrotSet[width * height ];
    }
    private Color getColor(int count) {
            return count >= colors.length ?  Color.BLACK : colors[count];
    }
    private int calc(double re, double img ) {
        int    counter = 0;
        double length;
        double aComplexNumberRe = 0;
        double aComplexNumberImg = 0;
```

```
            double real = 0;
            double imaginary = 0;

            do {
                real       =  aComplexNumberRe * aComplexNumberRe -
                              aComplexNumberImg * aComplexNumberImg;
                imaginary  = aComplexNumberRe *  aComplexNumberImg +
                              aComplexNumberImg *  aComplexNumberRe;
                aComplexNumberRe   = real;
                aComplexNumberImg  = imaginary;
                aComplexNumberRe   += re;
                aComplexNumberImg  += img;
                length = aComplexNumberImg * aComplexNumberImg +
                         aComplexNumberRe * aComplexNumberRe;
                counter++;
            } while (counter < MAX_COLORS && ( length < BOUNDERY ) );
            return counter;
        }
        public Color determineColor(int x, int y)   {
            double re = (minR * (width - x) + x * maxR) / width;
            double img = (minI * (height - y) + y * maxI) / height;
            return getColor(calc(re, img));
        }
        public WritableImage createImage()  {
            mandelBrotSetImage = new WritableImage(width, height);
            aPixelWriter = mandelBrotSetImage.getPixelWriter();

            for (int x = 0; x < width; x++) {
                    for (int y = 0; y < height; y++) {
                            aPixelWriter.setColor(x, y, determineColor(x, y));
                    }
            }
            return mandelBrotSetImage;
        }
    }

```

Source Code: Src/28/MandelbrotFX.java

Your Work:

You need to find out how you can divide the work up so such *n* threads can work on the problem. It must be possible to specify *n* on the command line as an argument. If *n* is not specified your proigram has to select a reasonable *n* beased on cores. What is the best possible *n* for execution on gandalf. What is the best possible speed up you can achieve and why?

Requirements:

Yu need to name your file: MandelbrotFX.java

Submission:

```
% ssh glados.cs.rit.edu # or use queeg.cs.rit.edu if glados is down
# password
# go to the directory where your solution is ...
% try hpb-grd lab8-3 'All files required'
```

Solution:

(This solution serves as the basis for the discussion in class. Sometimes there will be errors introduced to show common mistakes)

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.image.ImageView;
import javafx.scene.image.PixelWriter;
import javafx.scene.image.WritableImage;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class MandelbrotFX extends Application {

    WritableImage mandelBrotSetImage;
    final int IMG_WIDTH         = 800;
    final int IMG_HEIGHT        = 800;
    long milliSeconds;

    public void init()  {
        milliSeconds = System.currentTimeMillis();
    }
    public void end(String s)   {
        System.err.println(s + ":       " + ( System.currentTimeMillis() - milliSe
        System.err.println(" # of cores" +   ":       " +
        Runtime.getRuntime().availableProcessors() );
    }

    public void start(Stage theStage) {

        MandelbrotSet aMandelbrotSet = new MandelbrotSet(IMG_WIDTH, IMG_HEIGHT);

        init();
                mandelBrotSetImage = aMandelbrotSet.createImage();
        end("Multiple Thread MandelbrotSet Test");


        ImageView aImage = new ImageView();
        aImage.setImage(mandelBrotSetImage);

        StackPane root = new StackPane();
        root.getChildren().add(aImage);

        Scene scene = new Scene(root, IMG_WIDTH, IMG_HEIGHT);

        theStage.setTitle("Mandelbrot Set");
        theStage.setResizable(false);
        theStage.setScene(scene);
        theStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}


class MandelbrotSet extends Thread {

    private static final int    MAX_COLORS      = 256;
    private static final double BOUNDERY = 1000;
    private static int    width;
    private static int    height;
    private static WritableImage mandelBrotSetImage;
    private static PixelWriter aPixelWriter;
    private static final Color[] colors = new Color[MAX_COLORS];
    private static double minR  = -2.4;
    private static double maxR  = 0.9;
    private static double minI  = -1.3;
    private static double maxI  = 1.28;
    private static int MAX_THREADS = 8;
    private static MandelbrotSet[] allThreads;
    private int startingXPoint;
    private int endXPoint;

    static {
        for (int index = 0; index < colors.length; index++) {
            colors[index] = Color.RED.interpolate(Color.BLUE, (( 1.0 / colors.leng
        }
    }

    public MandelbrotSet() {
    }
    public MandelbrotSet(int width,int height) {
        this.width = width;
        this.height = height;
        mandelBrotSetImage = new WritableImage(width, height);
        if ( allThreads == null )
                allThreads = new MandelbrotSet[width * height ];
    }
    private Color getColor(int count) {
            return count >= colors.length ?  Color.BLACK : colors[count];
    }
    private int calc(double re, double img ) {
        int    counter = 0;
        double length;
        double aComplexNumberRe = 0;
        double aComplexNumberImg = 0;
        double real = 0;
        double imaginary = 0;

        do {
            real       =  aComplexNumberRe * aComplexNumberRe -
                         aComplexNumberImg * aComplexNumberImg;
            imaginary  = aComplexNumberRe *  aComplexNumberImg +
                         aComplexNumberImg *  aComplexNumberRe;
            aComplexNumberRe   = real;
            aComplexNumberImg  = imaginary;
            aComplexNumberRe   += re;
            aComplexNumberImg  += img;
            length = aComplexNumberImg * aComplexNumberImg +
                     aComplexNumberRe * aComplexNumberRe;
            counter++;
        } while (counter < MAX_COLORS && ( length < BOUNDERY ) );
        return counter;
    }
    public Color determineColor(int x, int y)   {
        double re = (minR * (width - x) + x * maxR) / width;
        double img = (minI * (height - y) + y * maxI) / height;
        return getColor(calc(re, img));
    }
    public WritableImage createImage()  {
        mandelBrotSetImage = new WritableImage(width, height);
        aPixelWriter = mandelBrotSetImage.getPixelWriter();

        int xDelta = width / MAX_THREADS;       // strips
        for (int x = 0; x < MAX_THREADS; x++) {
                int startingXPoint = x * xDelta;
                int endXPoint = (x + 1) * xDelta > width? width : ( x + 1 ) * xDel
                allThreads[ x ] = new MandelbrotSet();
                allThreads[ x ].startingXPoint = startingXPoint;
                allThreads[ x ].endXPoint = endXPoint;
                allThreads[ x ].start();
        }
        synchronized ( mandelBrotSetImage )     {
                try {
                        mandelBrotSetImage.wait();
                } catch (  InterruptedException e ) {
                        System.err.println("Interrupted!");
                }
        }
        for (int x = 0; x < MAX_THREADS; x++) {
                try {
                        allThreads[ x ].join();
                } catch ( Exception e ) {
                        System.out.println("Something went wrong with join");
                        e.printStackTrace();
                }
        }
        return mandelBrotSetImage;
    }
    public void run()   {
        for (int x = startingXPoint; x < endXPoint; x++) {
                for (int y = 0; y < height; y++) {
                        aPixelWriter.setColor(x, y, determineColor(x, y));
                }
        }
        synchronized ( mandelBrotSetImage )     {
                try {
                        mandelBrotSetImage.notify();
```

```
                        } catch (  Exception e ) {
                                System.err.println("Interrupted!");
                }       }
        }
    }

```

Source Code: Src/28_sol/MandelbrotFX.java