Contents

1 Basic Test Results 2

2 README 3

3 UML.pdf 4

4 ascii art/AsciiArtAlgorithm.java 5

5 ascii art/Shell.java 7

6 exceptions/InvalidActionException.java 11

7 exceptions/InvalidCommandException.java 12

8 exceptions/InvalidCommandFormatException.java 13

9 exceptions/UnknownCommandException.java 14

10 image/BrightnessCalculator.java 15

11 image/Image.java 17

12 image/ImageEditor.java 19

13 image char matching/CharConverter.java 21

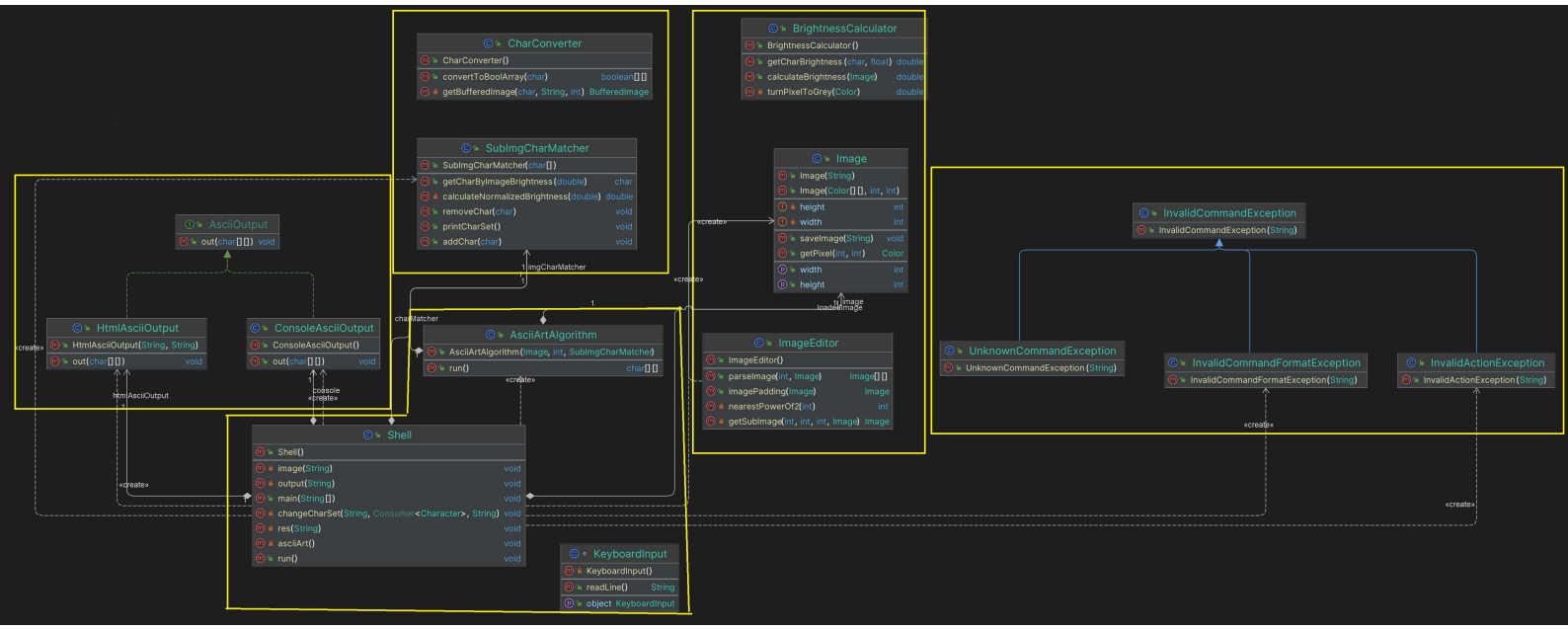
14 image char matching/SubImgCharMatcher.java 22

1 Basic Test Results

```
======================
===== EX3 TESTER =====
======================


===== EXTRACTING =====


===== CHECKING FILES =====


===== COPYING NECESSARY DIRECTORIES =====


===== ANALYZE README =====
How are you e342791191, archsak?


==== CHECKING LINE LENGTHS ====


===== COMPILE CODE =====


===== EXECUTE TESTS =====
************************ Test 1: Project Structure Test ************************
Description: Checks you submitted the required API
PASSED, Wow!
********************************************************************************
************************ Test 2: Sanity Check ************************
Description: Runs the example you were given in the exercise (section 1.6)
PASSED, Excellent!
**********************************************************************


==== CHECKING DOCUMENTATION ====


You passed all the tests, GGWP
```

2 README

```
e342791191,archsak
342791191,207600164

1. The Shell class handles everything that is linked to the user, gets the user input and calls the
right function accordingly. It creates an instance of SubImgCharMatcher to be passed to
AsciiArtAlgorithm. The SubImgMatcher is responsible of everything that has to do with the charset -
printing it, calculating chars brightness and normalized brightness, adding and removing chars. THe
AsciiAlgorithm class runs the algorithm and calls the right functions of SubImgBrightness and
ImageEditor when needed. ImageEditor is a utility class responsible of everything that has t do with
 editing the image - dividing it into subimages, padding it and such. BrightnessCalculator is also
 a utility class used by SubImgCharMatcher and AsciiArtAlgorithm to get the brightness of a char or
 a subimage. The Exceptions classes guide the user in understanding why his command didn't work.
 UnknownCommandException is used when command isn't recognised. InvalidCommandFormatException is
 used with a legal command but illegal format. InvalidActionException is used with valid command
 and valid format but action couldn't be performed due to another reason. They all extend
 InvalidCommandException.

2. Data structures
   private final Map<Character, Double> charset = new HashMap<>();
   We saved the charset in a hashedmap to be able to save a key-value couple of each char and
   its non-normalized brightness. Hashmap gave us the ability to save and access every value in
   an optimized time
   private final Map<Double, Character> normalizedCharset = new HashMap<>();
   We saved the normalized charset so it would allow us to save runtime when using the same charset
   twice in a row. The map allowed us to save key-value couples of each char and its normalized
   brightness
   private final List<Character> sortedChars = new ArrayList<>();
   We saved a sorted charset to save runtime, so that when we print the charset we don't have to
   sort it. Adding and removing chars after the initialization takes O(n) worst case and is
   therefore faster than creating and sorting a new List every time we want to print the chars. A
   List is more practical than an array and gave us the flexibility needed to add
   and remove chars as needed
   private static final Map<Map.Entry<Image, Integer>, Image[][]> parsedImages = new HashMap<>();
   We used a hash map to save the division into sub-images for each image and resolution
   combination, to save runtime if the user uses the same image and resolution twice.
   private final Map<Map.Entry<Image, Integer>, char[][]> subImagesBrightness = new HashMap<>();
   We used a map to save for each image and resolution, the result we get with the chars so we
   won't need to rerun the algorithm if the user wants the same picture with same resolution and
   charset twice


3. We created a package with Exceptions that answered our needs: the InvalidCommandException extends
 RuntimeException, and the three other exceptions extends InvalidCommandException. This allowed us
 to have the maximum amount of information concerning the command the user entered.

4. public void printCharSet()
   This function goes over the sorted charset and prints each char. It is needed in the Shell
   class when user wants to print the char in the charset. The alternative would be for Shell
   to access the charset, which is less convenient and takes up more memory space

5. XXX

```


CharConverter
CharConverter()
convertToBoolArray(char) boolean[][]
getBufferedImage(char, String, int) BufferedImage
SubImgCharMatcher
SubImgCharMatcher(char[])
getCharByImageBrightness(double) char
calculateNormalizedBrightness(double) double
removeChar(char) void
printCharSet() void
addChar(char) void
BrightnessCalculator
BrightnessCalculator()
getCharBrightness(char, float) double
calculateBrightness(image) double
turnPixelToGrey(Color) double
Image
Image(String)
Image(Color[][], int, int)
height int
width int
saveImage(String) void
getPixel(int, int) Color
width int
height int
AsciiOutput
out(char[][]) void
HtmlAsciiOutput
HtmlAsciiOutput(String, String)
out(char[][]) void
ConsoleAsciiOutput
ConsoleAsciiOutput()
out(char[][]) void
AsciiArtAlgorithm
AsciiArtAlgorithm(Image, int, SubImgCharMatcher)
run() char[][]
ImageEditor
ImageEditor()
parseImage(int, Image) Image[][]
imagePadding(Image) Image
nearestPowerOf2(int) int
getSubImage(int, int, int, Image) Image
Shell
Shell()
image(String) void
output(String) void
main(String[]) void
changeCharSet(String, Consumer<Character>, String) void
res(String) void
asciiArt() void
run() void
KeyboardInput
KeyboardInput()
readLine() String
object KeyboardInput
InvalidCommandException
InvalidCommandException(String)
UnknownCommandException
UnknownCommandException(String)
InvalidCommandFormatException
InvalidCommandFormatException(String)
InvalidActionException
InvalidActionException(String)
imgCharMatcher
charMatcher
loadedImage
image
htmlAsciiOutput
consoleAsciiOutput
«create»

4 ascii art/AsciiArtAlgorithm.java

```java
package ascii_art;

import image.BrightnessCalculator;
import image.Image;
import image.ImageEditor;
import image_char_matching.SubImgCharMatcher;

import java.util.AbstractMap;
import java.util.HashMap;
import java.util.Map;

/**
 * The AsciiArtAlgorithm class generates ASCII art from an image using a specified resolution and
 * character matcher.
 *
 * @author Elsa Sebagh and Aharon Saksonov
 */
public class AsciiArtAlgorithm {
    private final SubImgCharMatcher charMatcher;
    private final int resolution;
    private final Map<Map.Entry<Image, Integer>, char[][]> subImagesBrightness = new HashMap<>();
    private Image image;

    /**
     * Constructs an AsciiArtAlgorithm object with the given image, resolution,
     * and character matcher.
     *
     * @param image       The input image for generating ASCII art.
     * @param resolution  The resolution (number of characters per row) for the ASCII art.
     * @param charMatcher The character matcher used to match image brightness to ASCII characters.
     */
    public AsciiArtAlgorithm(Image image, int resolution, SubImgCharMatcher charMatcher) {
        this.image = image;
        this.resolution = resolution;
        this.charMatcher = charMatcher;
    }

    /**
     * Runs the ASCII art generation algorithm.
     *
     * @return A 2D char array representing the ASCII art generated from the image.
     */
    public char[][] run() {
        if (this.subImagesBrightness.containsKey(new AbstractMap.SimpleImmutableEntry<>(image,
                resolution)))
            return this.subImagesBrightness.get(new AbstractMap.SimpleImmutableEntry<>(image,
                    resolution));
        else {
            image = ImageEditor.imagePadding(image);
            Image[][] subImages = ImageEditor.parseImage(resolution, image);
            char[][] finalPicture = new char[subImages.length][subImages[0].length];
            for (int i = 0; i < subImages.length; i++) {
                for (int j = 0; j < subImages[i].length; j++) {
                    double imageBrightness =
                            BrightnessCalculator.calculateBrightness(subImages[i][j]);
                    finalPicture[i][j] = this.charMatcher.getCharByImageBrightness(imageBrightness);
                }
            }
            this.subImagesBrightness.put(new AbstractMap.SimpleImmutableEntry<>(image,
```

```
60                      resolution), finalPicture);
61              return finalPicture;
62          }
63      }
64  }
```

5 ascii art/Shell.java

```java
package ascii_art;

import ascii_output.ConsoleAsciiOutput;
import ascii_output.HtmlAsciiOutput;
import exceptions.InvalidActionException;
import exceptions.InvalidCommandFormatException;
import exceptions.UnknownCommandException;
import image.Image;
import image_char_matching.SubImgCharMatcher;

import java.io.IOException;
import java.util.function.Consumer;

/**
 * The Shell class provides a command-line interface for interacting with the ASCII art generator.
 * It allows users to perform various actions such as changing the character set, adjusting
 * resolution, loading images, and generating ASCII art.
 *
 * @author Elsa Sebagh and Aharon Saksonov
 */
public class Shell {
    // Constants for error messages

    private static final String INVALID_COMMAND_MESSAGE =
            "Did not execute due to incorrect format.";
    private static final String INVALID_RESOLUTION_COMMAND_MESSAGE =
            "Did not change resolution due to incorrect format.";
    private static final String INVALID_OUTPUT_COMMAND_MESSAGE =
            "Did not change output method due to incorrect format.";
    private static final String FAIL_IN_CHANGING_RESOLUTION_MESSAGE =
            "Did not change resolution due to exceeding boundaries.";
    private static final String COMMAND_DOESNT_EXIT = "Did not execute due to" +
            " incorrect command.";
    private static final String INVALID_ADD_MESSAGE = "Did not add due to" +
            " incorrect format.";
    private static final String INVALID_IMAGE_MESSAGE = "Did not change image due to" +
            " problem with image file.";


    // Character set matcher
    private static final SubImgCharMatcher imgCharMatcher = new SubImgCharMatcher(
            new char[]{'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'});
    // Prompt string
    private static final String PROMPT = ">>> ";
    // Default image file name
    private static final String DEFAULT_IMAGE = "cat.jpeg";
    private static final int ASCII_START = 32;
    private static final int ASCII_END = 126;
    private static final ConsoleAsciiOutput console = new ConsoleAsciiOutput();
    private static final HtmlAsciiOutput htmlAsciiOutput = new HtmlAsciiOutput("out.html",
            "Courier New");

    // Default to console output
    private static String selectedOutputStream = "console";
    private static int resolution = 128;
    private static Image loadedImage;

    /**
     * Main method to start the ASCII art shell.
```

```
60          *
61          * @param args Command-line arguments.
62          */
63         public static void main(String[] args) {
64             try {
65                 Shell.loadedImage = new Image(DEFAULT_IMAGE);
66                 Shell.run();
67             } catch (IOException error) {
68                 System.out.println(error.getMessage());
69             }
70 
71         }
72 
73 
74         /**
75          * Runs the ASCII art shell, allowing users to input commands and interact with the generator.
76          */
77         public static void run() {
78             System.out.print(PROMPT);
79             String command;
80             while (true) {
81                 try {
82                     command = KeyboardInput.readLine();
83                     String methodName = command.split(" ")[0];
84                     String params = command.split(" ").length > 1 ?
85                             command.split(" ")[1] : null;
86                     switch (methodName) {
87                         case "exit" -> {
88                             if (params != null)
89                                 throw new InvalidCommandFormatException(INVALID_COMMAND_MESSAGE);
90                             System.exit(0);
91                         }
92                         case "chars" -> {
93                             if (params != null)
94                                 throw new InvalidCommandFormatException(INVALID_COMMAND_MESSAGE);
95                             Shell.imgCharMatcher.printCharSet();
96                         }
97                         case "add" -> Shell.changeCharSet(params,
98                                 Shell.imgCharMatcher::addChar, "add");
99                         case "remove" -> Shell.changeCharSet(params,
100                                Shell.imgCharMatcher::removeChar, "remove");
101                        case "res" -> Shell.res(params);
102                        case "image" -> Shell.image(params);
103                        case "output" -> Shell.output(params);
104                        case "asciiArt" -> {
105                            if (params != null)
106                                throw new InvalidCommandFormatException(INVALID_COMMAND_MESSAGE);
107                            Shell.asciiArt();
108                        }
109                        default -> throw new UnknownCommandException(COMMAND_DOESNT_EXIT);
110                    }
111                    System.out.print(PROMPT);
112                } catch (InvalidCommandFormatException | UnknownCommandException err) {
113                    System.out.println(err.getMessage());
114                    System.out.print(PROMPT);
115                }
116            }
117        }
118 
119        /*
120         * Changes the character set according to the given parameters.
121         *
122         * @param param   The parameter indicating the character set modification.
123         * @param consumer The consumer function for character set modification.
124         * @param action  The action performed (add or remove).
125         * @throws InvalidCommandFormatException If an invalid action is encountered.
126         */
127        private static void changeCharSet(String param, Consumer<Character> consumer,
```

```
                                    String action)
            throws InvalidCommandFormatException {
        if (param == null || param.trim().isEmpty())
            throw new InvalidCommandFormatException(INVALID_ADD_MESSAGE);
        else if (param.equals("all")) {
            for (int i = ASCII_START; i <= ASCII_END; i++) {
                consumer.accept((char) i);
            }
        } else if (param.equals("space"))
            consumer.accept(' ');
        else if (param.length() == 1)
            consumer.accept(param.charAt(0));
            // Handle commands of the form p-m
        else if (param.matches(".-.")) {
            // Make sure p-m = m-p
            char char1 = param.charAt(0);
            char char2 = param.charAt(2);
            for (int i = Math.min(char1, char2); i <= Math.max(char1, char2); i++) {
                consumer.accept((char) i);
            }
        } else {
            throw new InvalidCommandFormatException("Did not " + action +
                    " due to incorrect format.");
        }
    }


    /*
     * Changes the resolution of the ASCII art.
     *
     * @param change The change in resolution ("up" or "down").
     * @throws IllegalArgumentException If the change parameter is invalid.
     */
    private static void res(String change) throws InvalidActionException,
            InvalidCommandFormatException {
        if (change == null)
            throw new InvalidCommandFormatException(INVALID_RESOLUTION_COMMAND_MESSAGE);
        int minCharsInRows = Math.max(1,
                Shell.loadedImage.getWidth() / Shell.loadedImage.getHeight());
        switch (change) {
            case "up":
                if (Shell.resolution * 2 > Shell.loadedImage.getWidth())
                    throw new InvalidActionException(
                            FAIL_IN_CHANGING_RESOLUTION_MESSAGE);
                else Shell.resolution *= 2;
                break;
            case "down":
                if (Shell.resolution / 2 < minCharsInRows)
                    throw new InvalidActionException(FAIL_IN_CHANGING_RESOLUTION_MESSAGE);
                else Shell.resolution /= 2;
                break;
            default:
                throw new InvalidCommandFormatException(INVALID_RESOLUTION_COMMAND_MESSAGE);
        }
        System.out.println("Resolution set to " + Shell.resolution);
    }

    private static void image(String path) throws InvalidCommandFormatException {
        try {
            if (path == null || path.isEmpty()) {
                throw new InvalidCommandFormatException(INVALID_IMAGE_MESSAGE);
            }
            Shell.loadedImage = new Image(path);
        } catch (IOException e) {
            //TODO Correct the exception here
            throw new InvalidActionException(INVALID_IMAGE_MESSAGE);
        }
    }
```

```
196
197         /*
198          * Changes the output method.
199          *
200          * @param outputStream The output method to be changed to.
201          * @throws InvalidActionException If an invalid action is encountered.
202          */
203         private static void output(String outputStream) throws InvalidCommandFormatException {
204             if (outputStream == null)
205                 throw new InvalidCommandFormatException(INVALID_OUTPUT_COMMAND_MESSAGE);
206             switch (outputStream) {
207                 case "console":
208                     selectedOutputStream = "console";
209                     break;
210                 case "html":
211                     selectedOutputStream = outputStream;
212                     break;
213                 default:
214                     throw new InvalidCommandFormatException(INVALID_OUTPUT_COMMAND_MESSAGE);
215             }
216         }
217
218         /*
219          * Generates ASCII art from the loaded image and displays it using the selected output method.
220          */
221         private static void asciiArt() {
222             AsciiArtAlgorithm algo = new AsciiArtAlgorithm(Shell.loadedImage,
223                     Shell.resolution, imgCharMatcher);
224             char[][] asciiArt = algo.run();
225             if (selectedOutputStream.equals("console")) {
226                 console.out(asciiArt);
227             } else if (selectedOutputStream.equals("html")) {
228                 htmlAsciiOutput.out(asciiArt);
229             }
230         }
231     }
232
233
```

6 exceptions/InvalidActionException.java

```java
package exceptions;

/**
 * A runtime exception indicating that the command action is invalid.
 */
public class InvalidActionException extends InvalidCommandException {
    /**
     * Constructor for InvalidAction
     *
     * @param message message to print
     */
    public InvalidActionException(String message) {
        super(message);
    }
}
```

7 exceptions/InvalidCommandException.java

```java
package exceptions;

/**
 * A runtime exception indicating that a command is invalid.
 */
public class InvalidCommandException extends RuntimeException {
    /**
     * Constructor for InvalidCommandException
     *
     * @param message message to print
     */
    public InvalidCommandException(String message) {
        super(message);
    }
}
```

8 exceptions/InvalidCommandFormatException.java

```java
package exceptions;

/**
 * A runtime exception indicating that the command format is invalid.
 */
public class InvalidCommandFormatException extends InvalidCommandException {
    /**
     * Constructor for InvalidCommandFormatException
     *
     * @param message message to print
     */
    public InvalidCommandFormatException(String message) {
        super(message);
    }
}
```

9 exceptions/UnknownCommandException.java

```java
package exceptions;


/**
 * A runtime exception indicating that the command is unknown.
 */
public class UnknownCommandException extends InvalidCommandException {
    /**
     * Constructor for UnknownCommandException
     *
     * @param message message to print
     */
    public UnknownCommandException(String message) {
        super(message);
    }
}
```

10 image/BrightnessCalculator.java

```java
package image;

import image_char_matching.CharConverter;

import java.awt.*;

/**
 * This is a utility class whose only role is to calculate the brightness of a given picture.
 */
public class BrightnessCalculator {
    private static final double RED_WEIGHT = 0.2126;
    private static final double GREEN_WEIGHT = 0.7152;
    private static final double BLUE_WEIGHT = 0.0722;

    /**
     * This function receives an image or a sub-image and calculates its brightness
     *
     * @param image the image to calculate the brightness of - an array of arrays of Colors
     * @return the overall brightness of the image
     */
    public static double calculateBrightness(Image image) {
        int totalPixels = 0;
        double totalBrightness = 0;
        double maxBrightness = 0;
        double pixelBrightness = 0;
        // Go over image and turn each pixel to grey
        for (int i = 0; i < image.getHeight(); i++) {
            for (int j = 0; j < image.getWidth(); j++) {
                pixelBrightness = turnPixelToGrey(image.getPixel(i, j));
                totalBrightness += pixelBrightness;
                maxBrightness = Math.max(maxBrightness, pixelBrightness);
                totalPixels += 1;
            }
        }
        return totalBrightness / (totalPixels * maxBrightness);
    }

    /**
     * This function gets a char and returns its brightness
     *
     * @param c            the char we want to calculate the brightness of
     * @param total_pixels number of pixels in the char - default is 16*16
     * @return a double between 0 and 1 representing the brightness
     */
    public static double getCharBrightness(char c, float total_pixels) {
        boolean[][] boolArray = CharConverter.convertToBoolArray(c);
        int countTrue = 0;
        for (boolean[] booleans : boolArray) {
            for (boolean bool : booleans) {
                if (bool) countTrue += 1;
            }
        }
        return (double) countTrue / total_pixels;
    }

    /* Helper function to calculate Brightness. This function receives a colored pixel and
    turns it grey;
     */
    private static double turnPixelToGrey(Color pixel) {
```

```
60          return pixel.getRed() * RED_WEIGHT + pixel.getGreen() * GREEN_WEIGHT
61                  + pixel.getBlue() * BLUE_WEIGHT;
62      }
63
64
65  }
```

11 image/Image.java

```java
package image;

import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

/**
 * A package-private class of the package image.
 *
 * @author Dan Nirel
 */
public class Image {

    private final Color[][] pixelArray;
    private final int width;
    private final int height;

    /**
     * Constructs an Image object from the specified file.
     *
     * @param filename The path to the image file.
     * @throws IOException If an error occurs while reading the image file.
     */
    public Image(String filename) throws IOException {
        BufferedImage im = ImageIO.read(new File(filename));
        width = im.getWidth();
        height = im.getHeight();

        pixelArray = new Color[height][width];
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                pixelArray[i][j] = new Color(im.getRGB(j, i));
            }
        }
    }

    /**
     * Constructs an Image object from the specified pixel array.
     *
     * @param pixelArray The pixel array representing the image.
     * @param width      The width of the image.
     * @param height     The height of the image.
     */
    public Image(Color[][] pixelArray, int width, int height) {
        this.pixelArray = pixelArray;
        this.width = width;
        this.height = height;
    }

    /**
     * Gets the width of the image.
     *
     * @return The width of the image.
     */
    public int getWidth() {
        return width;
    }
```

```
60
61      /**
62       * Gets the height of the image.
63       *
64       * @return The height of the image.
65       */
66      public int getHeight() {
67          return height;
68      }
69
70      /**
71       * Gets the color of the pixel at the specified coordinates.
72       *
73       * @param x The x-coordinate of the pixel.
74       * @param y The y-coordinate of the pixel.
75       * @return The color of the pixel.
76       */
77      public Color getPixel(int x, int y) {
78          return pixelArray[x][y];
79      }
80
81      /**
82       * Saves the image to a file with the specified filename.
83       *
84       * @param fileName The filename of the saved image.
85       */
86      public void saveImage(String fileName) {
87          // Initialize BufferedImage, assuming Color[][] is already properly populated.
88          BufferedImage bufferedImage = new BufferedImage(pixelArray[0].length, pixelArray.length,
89                  BufferedImage.TYPE_INT_RGB);
90          // Set each pixel of the BufferedImage to the color from the Color[][].
91          for (int x = 0; x < pixelArray.length; x++) {
92              for (int y = 0; y < pixelArray[x].length; y++) {
93                  bufferedImage.setRGB(y, x, pixelArray[x][y].getRGB());
94              }
95          }
96          File outputfile = new File(fileName + ".jpeg");
97          try {
98              ImageIO.write(bufferedImage, "jpeg", outputfile);
99          } catch (IOException e) {
100             throw new RuntimeException(e);
101         }
102     }
103
104 }
```

12 image/ImageEditor.java

```java
package image;

import java.awt.*;
import java.util.AbstractMap;
import java.util.HashMap;
import java.util.Map;


/**
 * This class regroups every function in charge of editing the image in any way whatsoever -
 * padding it, dividing it to subImages and so on. Every function is static.
 */
public class ImageEditor {
    private static final Map<Map.Entry<Image, Integer>, Image[][]> parsedImages = new HashMap<>();

    /**
     * This function receives an image and pads it so each side will be the size of a power of 2
     *
     * @param image The image to pad.
     */
    public static Image imagePadding(Image image) {
        //Calculate the correct dimensions of the image
        int newWidth = nearestPowerOf2(image.getWidth());
        int newHeight = nearestPowerOf2(image.getHeight());
        int widthDifference = newWidth - image.getWidth();
        int heightDifference = newHeight - image.getHeight();
        Color[][] paddedImage = new Color[newHeight][newWidth];
        // Apply white padding to the necessary areas
        for (int row = 0; row < newHeight; row++) {
            for (int col = 0; col < newWidth; col++) {
                if (row < heightDifference / 2 ||
                        row >= newHeight - (heightDifference / 2) ||
                        col < widthDifference / 2 ||
                        col >= newWidth - (widthDifference / 2)) {
                    paddedImage[row][col] = Color.WHITE;
                } else {
                    paddedImage[row][col] =
                            image.getPixel(row - heightDifference / 2,
                                    col - widthDifference / 2);
                }
            }
        }
        return new Image(paddedImage, newWidth, newHeight);
    }


    /**
     * This function separates the image into subImages according to the resolution
     *
     * @param resolution the desired resolution
     * @return an array of arrays of subImages
     */
    public static Image[][] parseImage(int resolution, Image image) {
        // Check if we already calculated the sub-images for this image and resolution
        if (ImageEditor.parsedImages.containsKey(new AbstractMap.SimpleImmutableEntry<>(image,
                resolution)))
            return ImageEditor.parsedImages.get(new AbstractMap.SimpleImmutableEntry<>(image,
                    resolution));
        // Else, calculate it and save it
```

```
60          int size = image.getWidth() / resolution;
61          Image[][] subImages = new Image[resolution][resolution];
62          for (int i = 0; i < resolution; i++) {
63              for (int j = 0; j < resolution; j++) {
64                  subImages[i][j] = getSubImage(size, i * size, j * size, image);
65              }
66          }
67          ImageEditor.parsedImages.put(new AbstractMap.SimpleImmutableEntry<>(image,
68                  resolution), subImages);
69          return subImages;
70      }
71
72
73      /*
74      This function return the subimage corresponding to a row and column
75       */
76      private static Image getSubImage(int size, int row, int col, Image image) {
77          // Create a Color[][] array to store the pixels of the sub-image
78          Color[][] subImagePixels = new Color[size][size];
79
80          // Copy the pixels from the original image to the sub-image array
81          for (int i = 0; i < size; i++) {
82              for (int j = 0; j < size; j++) {
83                  int originalRow = row + i;
84                  int originalCol = col + j;
85
86                  // Ensure the originalRow and originalCol are within bounds
87                  if (originalRow < image.getHeight() && originalCol < image.getWidth()) {
88                      subImagePixels[i][j] = image.getPixel(originalRow, originalCol);
89                  }
90              }
91          }
92          // Create and return a new Image object for the sub-image
93          return new Image(subImagePixels, size, size);
94      }
95
96      /*
97      This function returns the nearest power of two of a given number
98       */
99      private static int nearestPowerOf2(int value) {
100         return (int) Math.pow(2, Math.ceil(Math.log(value) / Math.log(2)));
101     }
102 }
```

13 image char matching/CharConverter.java

```java
package image_char_matching;

import java.awt.*;
import java.awt.image.BufferedImage;

/**
 * Inspired by, and partly copied from
 * https://github.com/korhner/asciimg/blob/95c7764a6abe0e893fae56b3b6b580e09e1de209/src/main/java
 * /io/korhner/asciimg/image/AsciiImgCache.java
 * described in the blog:
 * https://dzone.com/articles/ascii-art-generator-java
 * Adaptations made by Dan Nirel and again by Rachel Behar.
 * The class converts characters to a binary "image" (2D array of booleans).
 */
public class CharConverter {
    /**
     * The default resolution
     */
    public static final int DEFAULT_PIXEL_RESOLUTION = 16;
    private static final double X_OFFSET_FACTOR = 0.2;
    private static final double Y_OFFSET_FACTOR = 0.75;
    private static final String FONT_NAME = "Courier New";

    /**
     * Renders a given character, according to how it looks in the font specified in the
     * constructor, to a square black&white image (2D array of booleans),
     * whose dimension in pixels is specified.
     */
    public static boolean[][] convertToBoolArray(char c) {
        BufferedImage img = getBufferedImage(c, FONT_NAME, DEFAULT_PIXEL_RESOLUTION);
        boolean[][] matrix = new boolean[DEFAULT_PIXEL_RESOLUTION][DEFAULT_PIXEL_RESOLUTION];
        for (int y = 0; y < DEFAULT_PIXEL_RESOLUTION; y++) {
            for (int x = 0; x < DEFAULT_PIXEL_RESOLUTION; x++) {
                matrix[y][x] = img.getRGB(x, y) == 0; //is the color black
            }
        }
        return matrix;
    }

    private static BufferedImage getBufferedImage(char c, String fontName, int pixelsPerRow) {
        String charStr = Character.toString(c);
        Font font = new Font(fontName, Font.PLAIN, pixelsPerRow);
        BufferedImage img = new BufferedImage(pixelsPerRow, pixelsPerRow,
                BufferedImage.TYPE_INT_ARGB);
        Graphics g = img.getGraphics();
        g.setFont(font);
        int xOffset = (int) Math.round(pixelsPerRow * X_OFFSET_FACTOR);
        int yOffset = (int) Math.round(pixelsPerRow * Y_OFFSET_FACTOR);
        g.drawString(charStr, xOffset, yOffset);
        return img;
    }

}
```

14 image char matching/SubImgCharMatcher.java

```java
package image_char_matching;

import image.BrightnessCalculator;

import java.util.*;

/**
 * This class is responsible for matching between a charset with a certain brightness and the
 * corresponding ASCII character in the set.
 *
 * @author Elsa Sebagh and Aharon Saksonov
 */
public class SubImgCharMatcher {
    private static final String DID_NOT_EXECUTE_CHARSET_IS_EMPTY = "Did not execute. Charset is " +
            "empty.";
    private final static int TOTAL_SQUARES =
            (int) Math.pow(CharConverter.DEFAULT_PIXEL_RESOLUTION, 2);
    private final Map<Character, Double> charset = new HashMap<>();
    private final Map<Double, Character> normalizedCharset = new HashMap<>();
    private final List<Character> sortedChars = new ArrayList<>();
    private double minBrightness = Double.MAX_VALUE;
    private double maxBrightness = -1;


    /**
     * Constructor for the SubImgCharMatcher class.
     *
     * @param charset the charset we have to choose from
     */
    public SubImgCharMatcher(char[] charset) {
        for (char c : charset) {
            this.addChar(c);
        }
    }

    /**
     * This function returns the ASCII symbol whose brigthness it the closest to the brightness
     * we've been given
     *
     * @param brightness the brightness we're looking for
     * @return the ASCII symbol to be printed
     */
    public char getCharByImageBrightness(double brightness) {
        if (charset.isEmpty() || this.minBrightness == this.maxBrightness)
            throw new IllegalStateException(DID_NOT_EXECUTE_CHARSET_IS_EMPTY);
        if (this.normalizedCharset.isEmpty() || !this.normalizedCharset.containsKey(brightness)) {
            double difference = Double.MAX_VALUE;
            char closestChar = 0;
            for (Map.Entry<Character, Double> currentChar : charset.entrySet()) {
                // Calculate char brightness according to the rest of the chars in the set
                double newBrightness = this.calculateNormalizedBrightness(currentChar.getValue());
                double newDifference = Math.abs(newBrightness - brightness);
                // Update char closest to brightness
                if (newDifference < difference) {
                    difference = newDifference;
                    closestChar = currentChar.getKey();
```

```
                }
                // If the difference is the same, return the char with the lowest ASCII value
                else if (newDifference == difference) {
                    closestChar = (char) Math.min(closestChar, currentChar.getKey());
                }
            }
            normalizedCharset.put(brightness, closestChar);
            return closestChar;
        } else {
            return normalizedCharset.get(brightness);
        }
    }

    /**
     * Add a char to the charset map with its corresponding value
     *
     * @param c the char to be added
     */
    public void addChar(char c) {
        if (this.charset.containsKey(c)) {
            return;
        }
        double charBrightness = BrightnessCalculator.getCharBrightness(c, TOTAL_SQUARES);
        this.charset.put(c, charBrightness);
        sortedChars.add(c);
        Collections.sort(sortedChars);
        // Updates the min and max brightness if needed
        if (charBrightness > maxBrightness) {
            maxBrightness = charBrightness;
            this.normalizedCharset.clear();
        } else if (charBrightness < minBrightness) {
            minBrightness = charBrightness;
            this.normalizedCharset.clear();
        } else {
            double normalizedBrightness = this.calculateNormalizedBrightness(charBrightness);
            Character identicalChar = this.normalizedCharset.putIfAbsent(normalizedBrightness, c);
            if (identicalChar != null) {
                this.normalizedCharset.replace(normalizedBrightness, (char) Math.min(c,
                        identicalChar));
            }
        }
    }

    /**
     * Remove a char from the charset map
     *
     * @param c the char to be removed
     */
    public void removeChar(char c) {
        if (!charset.containsKey(c)) return;
        Double charBrightness = this.charset.get(c);
        this.charset.remove(c);
        this.sortedChars.removeIf(currentChar -> currentChar.equals(c));
        // Update min and max brightness if needed
        if (charBrightness == this.minBrightness || charBrightness == this.maxBrightness) {
            minBrightness = Double.MAX_VALUE;
            maxBrightness = -1;
            this.normalizedCharset.clear();
            for (Map.Entry<Character, Double> currentChar : charset.entrySet()) {
                Double currentBrightness = currentChar.getValue();
                if (currentBrightness > maxBrightness) maxBrightness = currentBrightness;
                else if (currentBrightness < minBrightness) minBrightness = currentBrightness;
            }
        }
    }

    /*
    This function normalizes the given brightness according to min and max
```

```
125         */
126        private double calculateNormalizedBrightness(double brightness) {
127            double maxMinusMin = this.maxBrightness - this.minBrightness;
128            return (brightness - this.minBrightness) / maxMinusMin;
129        }
130
131        /**
132         * This function prints the charset in the ASCII order
133         */
134        public void printCharSet() {
135            // Iterate over the sorted list and print the entries
136            for (Character key : sortedChars) {
137                System.out.print(key);
138                //We used get method because getLast didn't work in the presubmit.
139                if (!sortedChars.get(sortedChars.size() - 1).equals(key)) System.out.print(" ");
140            }
141            System.out.println();
142        }
143
144    }
```