```java
//Annika Boehme 23361825, Lukas Aumann 23323010

public class Color {
    //The RGB values are stored in a single int value.
    private int rgb;

    //Public static constant Color objects
    public static final Color BLACK = new Color(0, 0, 0);
    public static final Color WHITE = new Color(255, 255, 255);
    public static final Color GRAY = new Color(128, 128, 128);
    public static final Color RED = new Color(255, 0, 0);
    public static final Color GREEN = new Color(0, 255, 0);
    public static final Color BLUE = new Color(0, 0, 255);

    //Constructor which includes the RGB value.
    public Color(int rgbValue) {
        this.rgb = rgbValue;
    }

    //Constructs a Color object with specified red, green, and blue values.
    public Color(int red, int green, int blue) {

        // Validate and set red, green, and blue values. If the values are outside of the 0-255 range, they are clamped to the nearest valid value and an error message is printed.
        if (red < 0) {
            red = 0;
            System.err.println("The number You have put in for red was to low\nRed has been set to 0");
        } else if (red > 255) {
            red = 255;
            System.err.println("The number You have put in for red was to high\nRed has been set to 255");
        }
        if (green < 0) {
            green = 0;
            System.err.println("The number You have put in for green was to low\nGreen has been set to 0");
        } else if (green > 255) {
            green = 255;
            System.err.println("The number You have put in for green was to high\nGreen has been set to 255");
        }
        if (blue < 0) {
            blue = 0;
            System.err.println("The number You have put in for blue was to low\nblue has been set to 0");
        } else if (blue > 255) {
            blue = 255;
```

```
43              System.err.println("The number You have put in for blue was to high\nblue has been set to 255");
44          }
45          //Combine the RGB values into a single integer.
46          this.rgb = (red << 16) | (green << 8) | blue;
47      }
48
49      //Default constructor that creates a black color.
50      public Color() {
51          this.rgb = 0; //Black.
52      }
53
54      //Getter-Methode for the RGB value.
55      public int getRgb() {
56          return this.rgb;
57      }
58
59      //Getter-Methode for the red color channel.
60      public int getRed() {
61          return (this.rgb >> 16) & 0xFF; //Shifts the red bits to the right and isolates the last 8 bits.
62      }
63
64      //Getter-Methode for the green color channel.
65      public int getGreen() {
66          return (this.rgb >> 8) & 0xFF; //Shifts the green bits to the right and isolates the last 8 bits.
67      }
68
69      //Getter-Methode for the blue color channel.
70      public int getBlue() {
71          return this.rgb & 0xFF; //Isolates the last 8 bits.
72      }
73
74      //Getter-Methode for the hex value.
75      public String getHex() {
76          //Extracting the red, green, and blue values.
77          int red = getRed();
78          int green = getGreen();
79          int blue = getBlue();
80
81          //Converting the values to hexadecimal.
82          String hexRed = Integer.toHexString(red);
83          String hexGreen = Integer.toHexString(green);
84          String hexBlue = Integer.toHexString(blue);
85
```

```
86          //Ensuring each component is two digits.
87          hexRed = hexRed.length() == 1 ? "0" + hexRed : hexRed;
88          hexGreen = hexGreen.length() == 1 ? "0" + hexGreen : hexGreen;
89          hexBlue = hexBlue.length() == 1 ? "0" + hexBlue : hexBlue;
90
91          //Returning the hex string.
92          return "#" + hexRed.toUpperCase() + hexGreen.toUpperCase() + hexBlue.toUpperCase();
93      }
94
95      //Additional constructor that accepts a hex string with a '#' prefix.
96      public Color(String hex) {
97          hex = hex.replace("#", ""); //Remove the '#' prefix.
98          this.rgb = Integer.parseInt(hex, 16); //Convert hex string to an integer.
99      }
100
101     //Method to create the complementary color.
102     public Color complementaryColor() {
103
104         //Calculate the complementary colors for each color channel
105         int compRed = 255 - this.getRed();
106         int compGreen = 255 - this.getGreen();
107         int compBlue = 255 - this.getBlue();
108
109         //Create and return a new Color object with the complementary values
110         return new Color(compRed, compGreen, compBlue);
111     }
112
113     //Method to mix two colors.
114     public Color mixColor(Color otherColor) {
115
116         // Mitteln der Farbkan?le der aktuellen und der ?bergebenen Farbe
117         int mixedRed = (this.getRed() + otherColor.getRed()) / 2;
118         int mixedGreen = (this.getGreen() + otherColor.getGreen()) / 2;
119         int mixedBlue = (this.getBlue() + otherColor.getBlue()) / 2;
120
121         //Return a new Color object with the mixed values.
122         return new Color(mixedRed, mixedGreen, mixedBlue);
123     }
124
125     //Overriding the toString method.
126     @Override
127     public String toString() {
128         return getHex();
```

```
129     }
130
131
132     //Implementation of the main method.
133     public static void main(String[] args) {
134
135         //Test cases for the constructors.
136         Color black = new Color(); //Black.
137         Color white = new Color(255, 255, 255); //White.
138         Color red = new Color(255, 0, 0); //Red.
139         Color green = new Color(0, 255, 0); //Green.
140         Color blue = new Color(0, 0, 255); //Blue.
141         //Color invalidColor = new Color(-1, 256, 500); //Invalid color.
142
143         //Output of the constructor test cases.
144         System.out.println("Black (rgb): " + black.getRgb()); //Output: 0.
145         System.out.println("White (rgb): " + white.getRgb()); //Output: 16777215.
146         System.out.println("Red (rgb): " + red.getRgb()); //Output: 16711680.
147         System.out.println("Green (rgb): " + green.getRgb()); //Output: 65280.
148         System.out.println("Blue (rgb): " + blue.getRgb()); //Output: 255.
149         //System.out.println("Invalid Color (rgb): " + invalidColor.getRgb()); //Output: Correction and error messages in stderr.
150
151         //PeachPuff as a hexadecimal value.
152         //int peachPuffRgb = 0xFFDAB9;
153         //Color peachPuff = new Color(peachPuffRgb);
154         Color peachPuff = new Color(0xFFDAB9); //Customization for the ColorVisualizer.
155         //Output of the hexadecimal value of PeachPuff.
156         System.out.println("PeachPuff Hex-Value: " + peachPuff.getHex());
157
158         //Visualizing the colors with the ColorVisualizer class.
159         ColorVisualizer visualizerBlack = new ColorVisualizer(black);
160         ColorVisualizer visualizerWhite = new ColorVisualizer(white);
161         ColorVisualizer visualizerRed = new ColorVisualizer(red);
162         ColorVisualizer visualizerGreen = new ColorVisualizer(green);
163         ColorVisualizer visualizerBlue = new ColorVisualizer(blue);
164         ColorVisualizer visualizerPeachPuff = new ColorVisualizer(peachPuff);
165
166         //Additional test colors.
167         Color cyan = new Color("#00FFFF"); // Cyan.
168         Color aqua = new Color("#7FFFD4"); // Aqua.
169         Color gold = new Color("#FFD700"); // Gold.
170         Color magenta = new Color("#FF00FF"); // Magenta.
171
```

```
172         //Complementary color.
173         String colorHex = "#006400";
174         Color color = new Color(colorHex);
175
176         //Call, output and visualizing the complementaryColor method.
177         Color complementary = color.complementaryColor();
178         System.out.println("The complementary color of" + colorHex + " is " + complementary.getHex());
179         ColorVisualizer visualizerComplementary = new ColorVisualizer(complementary);
180
181         //Testing complementaryColor method.
182         Color complementaryCyan = cyan.complementaryColor();
183         Color complementaryAqua = aqua.complementaryColor();
184         Color complementaryGold = gold.complementaryColor();
185         Color complementaryMagenta = magenta.complementaryColor();
186         System.out.println("Cyan Complementary color (rgb): " + complementaryCyan.getRgb());
187         System.out.println("Aqua Complementary color (rgb): " + complementaryAqua.getRgb());
188         System.out.println("Gold Complementary color (rgb): " + complementaryGold.getRgb());
189         System.out.println("Magenta Complementary color (rgb): " + complementaryMagenta.getRgb());
190
191         //Testing mixColor method.
192         Color mixedColor1 = cyan.mixColor(aqua);
193         Color mixedColor2 = gold.mixColor(magenta);
194
195         System.out.println("Mixed-Color 1 (rgb): " + mixedColor1.getRgb());
196         System.out.println("Mixed-Color 2 (rgb): " + mixedColor2.getRgb());
197
198         //Test cases for the complementaryColor method.
199         Color colorFromHex = new Color("#FFA000");
200         //Output of the complementaryColor method.
201         System.out.println("RGB-Value of color #FFA000: " + colorFromHex.getRgb());
202         System.out.println("Hex-Value of color #FFA000: " + colorFromHex.getHex());
203         //Visualizing the complementary color.
204         ColorVisualizer visualizer = new ColorVisualizer(colorFromHex);
205
206         //Testing of the toString method.
207         //Color color = new Color("#FFA000");
208         //System.out.println("The toString() method returns: " + color.toString());
209
210         //Testing of the mixColor method.
211         Color color1 = new Color(255, 0, 0); //Red.
212         Color color2 = new Color(0, 0, 255); //Blue.
213         Color mixedColor = color1.mixColor(color2); //Purple.
214         System.out.println("Mixed Color (rgb): " + mixedColor.getRgb() + ", Hex: " + mixedColor.getHex());
```

```
215         //Visualizing the mixed color.
216         ColorVisualizer visualizerMixed = new ColorVisualizer(mixedColor);
217     }
218
219 }
```