

```

1  package ir.shayandaneshtar;
2
3  import ...
4
15
16  public class Main extends Application {
17      public static void main(String[] args) { launch(args); }
18
19
20
21      @FXML
22      @Override
23      public void start(Stage stage) throws URISyntaxException, IOException {
24          AnchorPane root = FXMLLoader
25              .load(getClass().getResource("/Main.fxml"))
26              .toURI()
27              .toURL();
28          Scene scene = new Scene(root, root.getPrefWidth(), root.getPrefHeight(),
29              false, SceneAntialiasing.BALANCED);
30          stage.setResizable(false);
31          stage.setScene(scene);
32          stage.initStyle(StageStyle.UTILITY);
33          stage.setTitle("Box Toolkit");
34          stage.setX(Screen.getPrimary().getBounds().getMaxX() - 470);
35          stage.show();
36      }
37  }
38
39

```

class Main:

همانند سوال قبل، ساختار اولیه فرم از فایل main.fxml خوانده می شود که ریشه آن یک AnchorPane است، مابقی تنظیمات پنجره ای است که باز می شود مثله نوع پنجره و مکان آن.

```

1  package ir.shayandaneshtar;
2
3  public enum Side {
4      // x || y
5      //ordinals 0 to 5 => Side(x) + Side(y) = 5
6      S1, S2, S3, S4, S5, S6;
7
8      @ public static Side getTheOtherSide(Side s) {
9          return Side.values()[5 - s.ordinal()];
10     }
11
12     public Side getTheOtherSide() {
13         return getTheOtherSide(this);
14     }
15 }

```

enum Side: هر وجه مکعب با عددی متناظر است طوری که مجموع عدد هر دو برابر 7 می شود (اگر از 1 شروع کنیم) و اگر از 0 شروع کنیم برابر 5 - متد های موجود هم برای گرفتن طرف دیگراند.

```

8  /**
9      * -----
10     * --/   2   /|
11     * -/_____/ |
12     * |       |3|
13     * |   1   | |
14     * |_____|/
15     * <p>
16     * x || y : Side(x) + Side(y) = 7
17     *
18     * @author shayan daneshvar
19     *
20     */
21     public class Box {
22         private final double weight;
23         private final HashMap<Side, Color> sides = new HashMap<>();
24
25         public boolean containsColor(Color color) {...}
26
27
28
29         private Box getS1OnTop() {...}
30
31         private Box getS5OnTop() {...}
32
33         private Box getS6OnTop() {...}
34
35         private Box getS3OnTop() {...}
36
37         private Box getS4OnTop() {...}
38
39
40
41         public Box(double weight) { this.weight = weight; }
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59     @
60     public Box setSideOnTop(Side side) {...}
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83

```

Class box:

weight همان وزن جعبه است، sides یک نگاشت از هر روی مکعب به رنگ آن است.

متد containsColor بررسی می کند که آیا رنگ خاصی در این مکعب موجود است یا نه.

متد های getSOnTop: این متد ها طوری مکعب را مچرخانند که روی Sn در بالای مکعب قرار گیرد، یعنی Sn به S2 برسد.

و بقیه عملیات لازم برای حفظ جای رو های مکعب صورت گیرد.

متد `setSideOnTop` : این متد ابتدا روی داده شده را گرفته بر اساس آن یکی از متد های `getS1OnTop` را صدا می زند و نسخه ای از جعبه را بر میگرداند که `side` داده شده در آن در بالای جعبه قرار دارد.

```
84 public List<Side> getSide_s(Color color) {...}
90
91 public double getWeight() { return weight; }
94
95 public Color getFront() { return sides.get(Side.S1); }
98
99 public Color getRear() { return sides.get(Side.S6); }
102
103 public Color getTop() { return sides.get(Side.S2); }
106
107 public Color getBottom() { return sides.get(Side.S5); }
110
111 public Color getRight() { return sides.get(Side.S3); }
114
115 public Color getLeft() { return sides.get(Side.S4); }
118
119 public Map<Side, Color> getSides() { return sides; }
122
123 public void addSide(Side side, Color color) { sides.put(side, color); }
126
127 public Box addSides(Color... colors) {...}
138
139 public String toString() {
140     return "Box(weight=" + this.getWeight() + ", sides=" + this.getSides() + ")";
141 }
142 }
143 }
```

متد `getSide_s` تمامی رو هایی که رنگ داده شده اند رو برمیگرداند.

متد های گت برای وزن و گرفتن رنگ روی خاصی از مکعب. و همچنین کل رو ها

متد `addSide` و `addSides` برای افزودن رو ها و رنگشان به مکعب پس از ساختن شی از آن است.

class Cube:

کلاس برای نشان دادن مکعب، مکعب پیشفرض `javafx` امکان رنگ کردن رو های مختلف را نمیدهد، برای همین این کلاس نوشته شده و هر روی مکعب با استفاده از یک مکعب پیش فرض `javafx` (کلاس `Box` جاوا افیکس) مدل شده است.

سیستم مختصات سه بعدی `javafx` به این صورت اصلا که محور `x` ها از چپ به راست زیاد می شود، محور `y` ها از بالا به پایین و محور `z` ها هر چه از بیننده دورتر می شود بیشتر می شود.

`animation timer`: برای اینکه همه وجوه مکعب را بتوانیم ببینم برای آن انیمیشن مینویسیم تا در هر فریم نسب به بردار (1و1و1) دوران داده شود و همه وجه های آن دیده شوند.

```

11 public class Cube extends Group {
12     private Box box1;
13     private Box box2;
14     private Box box3;
15     private Box box4;
16     private Box box5;
17     private Box box6;
18
19     public Cube(double length) {
20         box1 = new Box(length, length, v2: 0.5);
21         box2 = new Box(length, v1: 0.5, length);
22         box3 = new Box(v: 0.5, length, length);
23         box4 = new Box(v: 0.5, length, length);
24         box5 = new Box(length, v1: 0.5, length);
25         box6 = new Box(length, length, v2: 0.5);
26         box1.setTranslateZ(-length / 2);
27         box2.setTranslateY(-length / 2);
28         box3.setTranslateX(length / 2);
29         box4.setTranslateX(-length / 2);
30         box5.setTranslateY(length / 2);
31         box6.setTranslateZ(length / 2);
32         getChildren().addAll(box1, box2, box3, box4, box5, box6);
33     }
34     public void enableRotationAnimation() {
35         (AnimationTimer) (l) → {
37             setRotationAxis(new Point3D(v: 1, v1: 1, v2: 1));
38             setRotate(getRotate() + 3);
39         }.start();
41     }

```

متد های ست برای ست کردن رنگ هر وجه مکعب.

متد `getGraphics` که مدل `box` در برنامه را میگیرد و شی 3 بعدی متناظر با آن را تولید میکند.

فرم طراحی شده موجود در `fxml`:

**Box Stacking Problem**

Color S1: #FFFFFF

Color S2: #FFFFFF


Color S3: #FFFFFF

Color S4: #FFFFFF

Color S5: #FFFFFF

Color S6: #FFFFFF

Weight:



```

43 public void setColorSide1(Color color) {...}
46
47 public void setColorSide2(Color color) {...}
50
51 public void setColorSide3(Color color) {...}
54
55 public void setColorSide4(Color color) {...}
58
59 public void setColorSide5(Color color) {...}
62
63 public void setColorSide6(Color color) {...}
66
67 @ public static Cube getGraphics(ir.shayandaneshvar.Box box, int length) {...}
77
78 @ public static Cube getGraphics(ir.shayandaneshvar.Box box) {...}
81 }
82

```

```

29 public class Presenter implements Initializable {
30 </> @FXML private AnchorPane root;
31 </> @FXML private JFXColorPicker s1Color;
32 </> @FXML private JFXColorPicker s2Color;
33 </> @FXML private JFXColorPicker s3Color;
34 </> @FXML private JFXColorPicker s4Color;
35 </> @FXML private JFXColorPicker s5Color;
36 </> @FXML private JFXColorPicker s6Color;
37 </> @FXML private JFXTextField weightField;
38 private List<Box> boxList;
39 private Group view;
40 private Stage stage;
41
42 @FXML void addBox() {...}
63 private void update3DView() {...}
78 @FXML void reset() {...}
82 private void clearWindow() { view.getChildren().clear(); }
85 @FXML void startStacking() {...}
91 @ private void drawStack(List<Box> boxes) {...}
118 @Override
119 public void initialize(URL url, ResourceBundle resourceBundle) {...}
134 private void show3DView() throws URISyntaxException, MalformedURLException {...}
144 private Function<List<Box>, List<Box>> getTallestBoxStack
145 = (list) -> {...};
201 private void extractAnswer(List<Box> list, HashMap<Side, Integer>[]
202 valuesList, int maxIndex, List<Box> result, Color color) {...}
230
231 }

```

class Presenter:

این کلاس کنترلر فرم هست و الگوریتم در این کلاس موجود می باشد.

SnColor انتخاب گر های رنگ هستند که در فرم هستند، روت – ریشه فرم است. weightField هم برای ورودی گرفتن وزن است.

هرکدام از این مکعب ها در listBox نگاه داشته می شوند. Group و Stage برای پنجره دیگری هستند که در کنار فرم ابتدایی برای نمایش سه بعدی باز می شود، دلیل این کار این است که اگر یک scene سه بعدی باشد دیگر فرم ها از کار میفتند به همین دلیل از روی اجبار ، فرم و نمایش سه بعدی را جدا کردیم.

متد initialize: که این ها در متد initialize که کاربردی شبیه به سازنده دارد ساخته می شوند. برای سه بعدی کردن باید scene یک camera داشته باشد تا محور z برای آن تعریف شود. علاوه بر آن برای مشاهده اشیای سه بعدی باید scene دارای depth buffer باشد که متغیر چهارم سازنده آن می باشد. Scene Antialiasing نیز برای واضح تر شدن اشیاء اضافه شده است.

متد addBox: این متد که با دکمه متناظر با خود بایند شده، ابتدا متد show3Dview را فراخوانی میکند تا در صورتی که بخش 3 بعدی در حال نمایش نبود، نمایش داده شود سپس جعبه داده شده را به لیست می افزاید و متد update3DView را فراخوانی میکند که این مدت کل جعبه هارا در پنجره سه بعدی رسم میکند، همچنین انیمیشن نوشته شده را برای آن ها فعال میکند تا بتوانیم همه وجوه را ببینیم.

متد ریست که به عکس موجود در فرم بایند شده، صفحه را پاک میکند و listBox را نیز خالی میکند.

متد clearWindow صفحه را پاک میکند.

متد drawStack: پس از حل،(جواب) جعبه ها را روی هم میچیند و انیمیشن روی آن ها تعریف میکند تا مطمئن شویم بالای هر جعبه ، جعبه ای قرار گرفته که پایین آن به رنگ بالای جعبه زیرین است.

متد startStacking: متد که با دکمه متناظر با خودش بایند شده ، هنگامی که صدا زده می شود، فانکشن getTallestStack را صدا می زند که جواب را به صورت لیستی از باکس ها برمیگرداند و سپس این لیست به drawStack داده می شود تا رسم شوند.

فانکشن getTallestBoxStack که به صورت لامبدا نوشته شده، بخش اصلی الگوریتم حل مسئله است که برای حل از الگوریتم LIS الهام گرفته شده.

توضیح الگوریتم: ابتدا همه باکس ها را بر اساس وزن مرتب میکنیم(مسئله را از چیدن از بالا به پایین حل میکنیم که فرقی ندارد). و یک آرایه ای به طول کل باکس ها در نظر میگیریم که درون هر عضو از این آرایه همه رو های باکس (که 6 تا است) به همراه عددی که بیانگر بلند ترین برج قابل ساخت با این روی مکعب است، بنابراین در ابتدا مقدار همه این هارا برابر یک قرار می دهیم، چرا که در ابتدا بلند ترین برج، خود همین جعبه است که ارتفاع آن یک است. حال باید به ازای هر مکعب و به ازای هر روی آن اندازه بزرگترین برج قابل ساختن را پیدا کنیم که بسیار مشابه LIS است با این تفاوت که به جای شرط صعودی بودن دو سیکونس و بزرگتر بودن فعلی از قبلی ها، شرط این را داریم که اول باید از ابتدا تا خود برج بباییم و هرگاه به جعبه ای خوردیم که رنگ این روی مکعب فعلی در آن موجود بود شرایط زیر را بررسی کنیم:

اول باید ببینیم که آیا برج ساخته شده از این مکعب قبلی اگر روی مقابل این رویی که هم رنگ روی فعلی (رویی که الان داریم بالا میگذاریم) مکعب فعلی است، طولش چقدر است، اگر طول مکعب ساخته شده وقتی که روی دیگر آن بالا قرار گیرد بیشتر یا مساوی حالتی باشد که روی هم رنگ با مکعب فعلی بالا قرار گیرد به این معنی است که اگر مکعب فعلی را زیر مکعب قبلی قرار دهیم به حالتی با طول بیشتری دست پیدا می کنیم، چرا که اگر اینطور نباشد (این شرط بررسی نشود) دیگر برج با آن طول ساخته نمی شود.

حال در هر مرحله این فرایند را از ابتدا تا مکعب قبل از مکعب فعلی می رویم و هر گاه شرایط برقرار بود طول برج ساخته شده با این روی جعبه را یکی افزایش می دهیم، این فرایند را ادامه می دهیم تا آرایه اندازه برج ها کامل شود و بتوانیم ترتیب را از آن استخراج کنیم.

```

for (int i = 1; i < list.size(); i++) {
    for (int k = 0; k < i; k++) {
        for (int j = 0; j < 6; j++) {
            int curValue = valuesList[i].get(Side.values()[j]);
            Color color = list.get(i).getSides().get(Side.values()[j]);
            List<Side> sides = list.get(k).getSide_s(color);
            for (Side s : sides) {
                int value = valuesList[k].get(s);
                if (value <= valuesList[k]
                    .get(s.getTheOtherSide())) {
                    curValue++;
                    break;
                }
            }
            valuesList[i].put(Side.values()[j], curValue);
        }
    }
}
}

```

حال به دنبال مکعبی می گردیم که بیشترین طول را برای ساختن برج به ما می داد و وجهی از آن که این بیشترین طول را می داد استخراج میکنیم و این مکعب را طوری که وجه بهینه آن بالا قرار گیرد را درون لیست `result` قرار می دهیم و آن را به متد `extractAnswers` می دهیم تا همین فرایند را با مابقی مکعب ها انجام دهد با این تفاوت که از بین حداکثر ها باید آن وجهی را انتخاب کند که پایین آن همرنگ قبلی است و سپس رنگ بالای آن را استخراج کرده، مکعب را طوری بچرخاند که رنگ این وجه که دارای این رنگ است در بالا قرار گیرد و این فرایند طی شود.

در انتها چون همه مکعب ها همواره به انتهای لیست افزوده میشوند، باید کل لیست را برعکس کنیم تا اولین عنصر لیست، بالاترین آن باشد. (بهرتر بود از `stack` استفاده می کردیم که دیگر نخواهیم برعکس کنیم اما چون پشته موجود در `java` از نظر پرفورمانس ضعیف بود فعلا از لیست استفاده کردیم، تا بعدا پشته سریعی را خودمان پیاده سازی کنیم و در اینجا استفاده کنیم)

حال جواب نهایی درون لیست `result` موجود است به این صورت که عنصر آخر موجود در لیست پایین ترین عنصر است.

همانطور که در متد `drawStack` گفته شد این نتیجه به این متد داده میشود تا جعبه های سه بعدی را به همراه وزنشان بکشد

و انیمیشنی برای آن تعریف می کند که مکعب ها را بالا پایین کند تا ببینیم که سوال درست حل شده یا خیر.

مثال:

ورودی:

S1	S6
S2	S5
S3	S4

به فرم رو به رو است و وزن آن را زیر آن می نویسیم.

```

private void extractAnswer(List<Box> list, HashMap<Side, Integer>[]
    valuesList, int maxIndex, List<Box> result, Color color) {
    if (maxIndex == 0) return;
    int index = 0;
    int maxVal = 0;
    Side top = null;
    for (int i = 0; i < maxIndex; i++) {
        if (!list.get(i).containsColor(color)) continue;
        int temp = valuesList[i].values()
            .stream().mapToInt(Integer::intValue).max().getAsInt();
        if (temp > maxVal) {
            int finalI = i;
            top = valuesList[i].entrySet().stream()
                .filter(z -> z.getValue().equals(temp))
                .map(Map.Entry::getKey)
                .filter(x -> list.get(finalI).getSides()
                    .get(x.getTheOtherSide()).equals(color))
                .findFirst().get();
            index = i;
            maxVal = temp;
        }
    }
    Box wantedBox = list.get(index).setSideOnTop(top);
    Color topColor = wantedBox.getTop();
    result.add(wantedBox);
    if (maxVal == 1) return;
    extractAnswer(list, valuesList, index, result, topColor);
}

```



```

173     int maxIndex = 0;
174     int maxValue = 0;
175     Side topSide = null;
176     for (int i = 0; i < list.size(); i++) {
177         int temp = valuesList[i].values().stream()
178             .mapToInt(Integer::intValue)
179             .max().orElse(0);
180         if (temp > maxValue) {
181             topSide = valuesList[i].entrySet().stream()
182                 .filter(z -> z.getValue().equals(temp))
183                 .map(Map.Entry::getKey)
184                 .findAny().get();
185             maxIndex = i;
186             maxValue = temp;
187         }
188     }
189     List<Box> result = new ArrayList<>();
190     Box box = list.get(maxIndex).setSideOnTop(topSide);
191     Color color = box.getTop();
192     result.add(box);
193     extractAnswer(list, valuesList, maxIndex, result, color);
194     Collections.reverse(result);
195     return result;
196 };

```

ورودی:

red	green	orange	red	khaki	orange	magenta	khaki
blue	wheat	cyan	yellow	red	blue	red	blue
yellow	purple	blue	green	green	yellow	cyan	green
5		10		12		14	

حل:

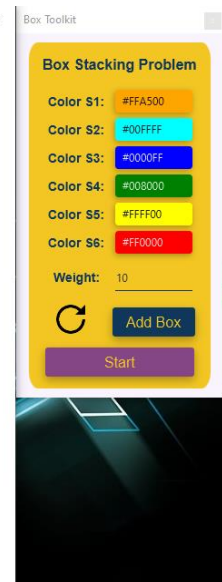
1	1	1	2	1	2	1	2
1	1	1	2	3	3	4	4
1	1	2	2	3	2	2	2

این به این معنی است که پایین ترین عنصر باید رنگ بالای آن یا آبی باشد و یا قرمز.

نمونه اجرای این ورودی ها در برنامه:

طریقه انتخاب رنگ به اسم: روی colorChooser موجود برای وجه دلخواه کلیک کنید، روی custom color بزنید

چرخه رنگ باز می شود، نام رنگ را تایپ کنید (باید املاي آن درست باشد) سپس enter بزنید:



این بخش مربوط به رنگ از کتابخانه jfoenix استفاده شده است که سهولت انتخاب رنگ را افزایش دهد، خود javafx نیز colorChooser دارد ولی به مراتب ساده تر است.

پس اضافه کردن هر چهار جعبه:

این عکس در حال چرخش مکعب ها، زمانی گرفته شده که رو های  $s_1, s_2, s_3$  قابل مشاهده اند:

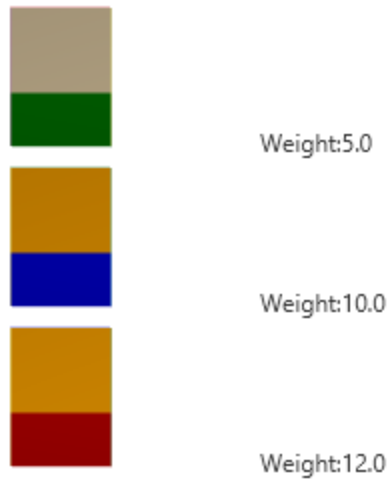


با زدن روی start الگوریتم اجرا می شود و بلند ترین برج از جعبه ها ساخته می شود.

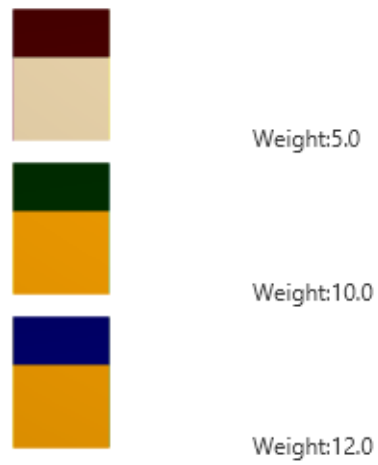
عکس از هر دو نما گرفته شده تا بالا و پایین هر مکعب نشان داده شود.

همانطور که مشاهده می شود رنگ روی بالایی هر مکعب با رنگ روی پایینی مکعب بالای خود برابر است.

Colored Box Stacking Problem



Colored Box Stacking Problem



نیازمندی ها برای اجرا برنامه مشابه برنامه مثلث بندی است.