CS2030 Lab #3: Rubik's Cube

Tags & Categories

Tags:

Categories:

Related Tutorials

Task Content

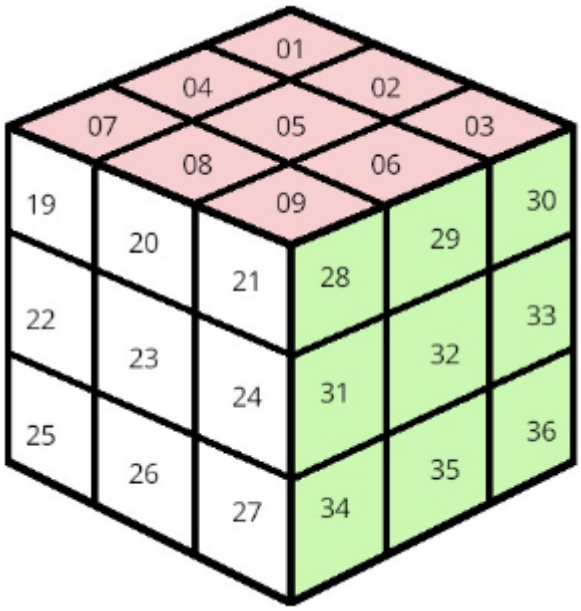
CS2030 Rubik's Cube

Topic Coverage

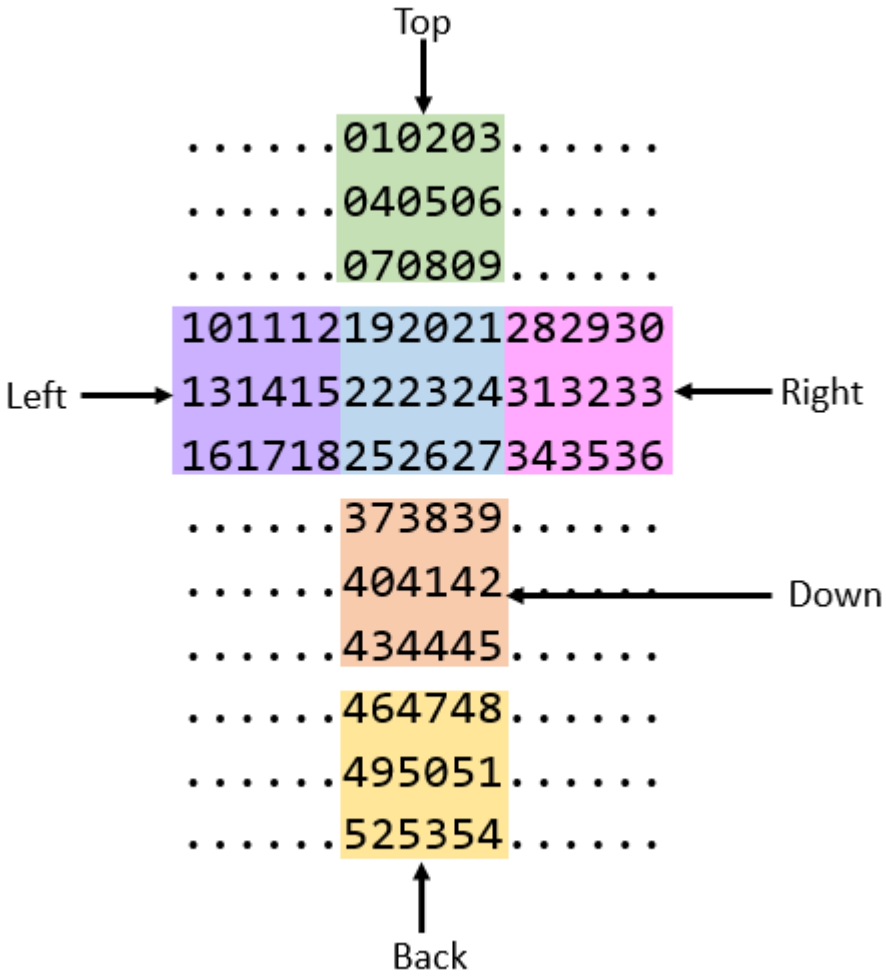
- Inheritance and polymorphism
- Abstract classes and interfaces
- Immutability and deep-copying

Problem Description

The Rubik's cube is a three-dimensional combination puzzle that is considered the world's best-selling toy.



The 3 × 3 rubik's cube above can be flattened so that all six faces are shown.



Your task is to write a program that reads the cube faces as input, followed by the turns. It then manipulates the cube by turning the faces clockwise (`F/R/U/L/B/D`), anti-clockwise (`F'/R'/U'/L'/B'/D'`) or halfway around (`F2/R2/U2/L2/B2/D2`). The final cube is then output. As an example, the input

```
1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45
46 47 48 49 50 51 52 53 54
F
R'
U2
```

will give the output

```
......541518......
......510504......
......480201......
303336455352101137
131438262306293235
161739272412070809
......343119......
......404120......
......434421......
......464728......
......495042......
......032225......
```

Take note of the following assumptions:

- Input always begins with a set of `54` values specifying the face values of the `3 × 3` cube in order (top, left, front, right, down, back),
- The numbers of the cube faces are positive (`> 0`) integers less than `100`
- Input turns are always valid

This task is divided into several levels. Read through all the levels to see how the different levels are related. **You need to complete all levels**.

Level 1

Since a cube comprises six faces, let's begin by developing the `Face` class. The cube face's nine values can be represented using a `3 × 3` array of integers. Each face can also be rotated right or left.

Develop the following methods for the `Face` class:

- `Face(int[][] grid)`: the constructor that takes in the `3 × 3` array of integers
- `Face right()`: performs a quarter rotation to the right and returns a new `Face`
- `Face left()`: performs a quarter rotation to the left and returns a new `Face`
- `Face half()`: performs a half rotation and returns a new `Face`
- `int[][] toIntArray()`: returns the `3 × 3` integer grid of the `Face` object
- `String toString()`: returns a String representing the `Face` object

In addition, a `Face` object should be cloneable. That is, there is a `clone()` method that creates an immutable copy. Note that `Face newFace.grid = this.grid` is not enough, as both will reference the same grid. Write a `Cloneable` interface that enforces the definition of the `clone` method. This will be useful in a later level.

It is also worth noting that the `Object` class defines a similar `clone()` method with a `protected` access modifier. While we can override this with another `clone()` method in the Face class, the accessibility cannot be less restrictive than `protected`. Only by implementing the `Cloneable` interface, would the overriding `clone()` method be `public` accessible.

In addition, use `String.format("%02d", ...)` to output the cube face's values.

```
$ javac *.java
$ jshell -q your_java_files_in_bottom-up_dependency_order < test1.jsh
jshell> Face f = new Face(new int[][]{{1,2,3},{4,5,6},{7,8,9}})
jshell> Face g = f.clone()
jshell> Cloneable c = g
jshell> c.clone()
$.. ==>
010203
040506
070809

jshell> f.right()
$.. ==>
070401
080502
090603

jshell> f.left()
$.. ==>
030609
020508
010407

jshell> f.half()
$.. ==>
090807
060504
030201

jshell> g
g ==>
010203
040506
070809

jshell> f.toIntArray()
$.. ==> int[3][] { int[3] { 1, 2, 3 }, int[3] { 4, 5, 6 }, int[3] { 7, 8, 9 } }
jshell> /exit
```

Level 2

With the `Face` class ready, we can proceed to develop `Rubik`. We define `Rubik` as a rubik's cube with one face that can be turned using the following methods:

- `right()`: turns the face clockwise
- `left()`: turns the face anti-clockwise

- `half()`: turns the face half a revolution

`RubikFront` is a `Rubik` where the turns are applied to the front face. Define the `RubikFront` constructor that takes in a three-dimensional (`6 × 3 × 3`) integer array of the six face's values.

Since all turn methods return an immutable object, it should be cloneable too.

```
$ javac *.java
$ jshell -q your_java_files_in_bottom-up_dependency_order < test2.jsh
jshell> int[][][] grid = new int[6][3][3]
jshell> int d = 1
jshell> for (int k = 0; k < 6; k++)
   ...>     for (int i = 0; i < 3; i++)
   ...>         for (int j = 0; j < 3; j++) grid[k][i][j] = d++;
jshell> Rubik r = new Rubik(grid)
|  Error:
|  Rubik is abstract; cannot be instantiated
|  Rubik r = new Rubik(grid);
|            ^-------------^
jshell> Rubik r = new RubikFront(grid)
jshell> Rubik s = r.clone();
jshell> Cloneable c = s
jshell> c.clone()
$.. ==>
......010203......
......040506......
......070809......
101112192021282930
131415222324313233
161718252627343536
......373839......
......404142......
......434445......
......464748......
......495051......
......525354......

jshell> r.left()
$.. ==>
......010203......
......040506......
......283134......
101109212427392930
131408202326383233
161707192225373536
......121518......
......404142......
......434445......
......464748......
......495051......
......525354......

jshell> r.right()
$.. ==>
......010203......
......040506......
......181512......
101137252219072930
131438262320083233
161739272421093536
......343128......
......404142......
......434445......
......464748......
......495051......
......525354......

jshell> r.half()
$.. ==>
......010203......
......040506......
......393837......
101134272625182930
131431242322153233
```

```
161728212019123536
......090807......
......404142......
......434445......
......464748......
......495051......
......525354......

jshell> s
s ==>
......010203......
......040506......
......070809......
101112192021282930
131415222324313233
161718252627343536
......373839......
......404142......
......434445......
......464748......
......495051......
......525354......

jshell> /exit
```

Level 3

A `Rubik` can also be oriented to any one of the six sides using the following methods:

- `rightView()`: orientates the cube so that the right-side faces front
- `leftView()`: orientates the cube so that the left-side faces front
- `upView()`: orientates the cube so that the top-side faces front
- `downView()`: orientates the cube so that the bottom-side faces front
- `backView()`: orientates the cube so that the back-side faces front. Although you can view the back of the cube by either orientating right/left or up/down, for ease of correctness checking, we stipulate that you can **only orientate right/left**
- `frontView()`: no orientation needed

```
$ javac *.java
$ jshell -q your_java_files_in_bottom-up_dependency_order < test3.jsh
jshell> int[][][] grid = new int[6][3][3]
jshell> int d = 1
jshell> for (int k = 0; k < 6; k++)
   ...>     for (int i = 0; i < 3; i++)
   ...>         for (int j = 0; j < 3; j++) grid[k][i][j] = d++;
jshell> Rubik r = new RubikFront(grid)
jshell> r.upView()
$.. ==>
......464748......
......495051......
......525354......
161310010203303336
171411040506293235
181512070809283134
......192021......
......222324......
......252627......
......373839......
......404142......
......434445......

jshell> r.rightView()
$.. ==>
......070401......
......080502......
......090603......
192021282930545352
222324313233515049
252627343536484746
......394245......
......384144......
......374043......
......181716......
......151413......
```

```
......121110......

jshell> r.downView()
$.. ==>
......192021......
......222324......
......252627......
121518373839343128
111417404142353229
101316434445363330
......464748......
......495051......
......525354......
......010203......
......040506......
......070809......

jshell> r.leftView()
$.. ==>
......030609......
......020508......
......010407......
545352101112192021
515049131415222324
484746161718252627
......434037......
......444138......
......454239......
......363534......
......333231......
......302928......

jshell> r.backView()
$.. ==>
......090807......
......060504......
......030201......
282930545352101112
313233515049131415
343536484746161718
......454443......
......424140......
......393837......
......272625......
......242322......
......212019......

jshell> r.frontView()
$.. ==>
......010203......
......040506......
......070809......
101112192021282930
131415222324313233
161718252627343536
......373839......
......404142......
......434445......
......464748......
......495051......
......525354......

jshell> /exit
```

Level 4

We are now *really* ready to turn the other faces. Just like `RubikFront`, we need to implement five other classes:

- `RubikLeft` to turn the left face
- `RubikRight` to turn the right face
- `RubikBack` to turn the back face
- `RubikUp` to turn the top face
- `RubikDown` to turn the bottom face

You might think that you need to implement right, left and half turns separately for each of the above classes. However, there is no need to. With only front-face turns implemented in `RubikFront`, all you have to do is to orientate the turning side so that it is facing the front, make the front-face turn, and then orientate it back.

If the design has been done correctly, we can simply extend our implementation with left/right/half turns for each of the remaining classes.

```
$ javac *.java
$ jshell -q your_java_files_in_bottom-up_dependency_order < test4.jsh
jshell> int[][][] grid = new int[6][3][3]
jshell> int d = 1
jshell> for (int k = 0; k < 6; k++)
   ...>     for (int i = 0; i < 3; i++)
   ...>         for (int j = 0; j < 3; j++) grid[k][i][j] = d++;
jshell> Rubik rubik = new RubikFront(grid);
jshell> ((Rubik) new RubikUp(rubik)).left()
$.. ==>
......030609......
......020508......
......010407......
545352101112192021
131415222324313233
161718252627343536
......373839......
......404142......
......434445......
......464748......
......495051......
......302928......

jshell> ((Rubik) new RubikUp(rubik)).right()
$.. ==>
......070401......
......080502......
......090603......
192021282930545352
131415222324313233
161718252627343536
......373839......
......404142......
......434445......
......464748......
......495051......
......121110......

jshell> ((Rubik) new RubikUp(rubik)).half()
$.. ==>
......090807......
......060504......
......030201......
282930545352101112
131415222324313233
161718252627343536
......373839......
......404142......
......434445......
......464748......
......495051......
......212019......

jshell> ((Rubik) new RubikRight(rubik)).left()
$.. ==>
......010248......
......040551......
......070854......
101112192003303336
131415222306293235
161718252609283134
......373821......
......404124......
......434427......
......464739......
......495042......
......525345......
```

```
jshell> ((Rubik) new RubikRight(rubik)).right()
$.. ==>
......010221......
......040524......
......070827......
101112192039343128
131415222342353229
161718252645363330
......373848......
......404151......
......434454......
......464703......
......495006......
......525309......

jshell> ((Rubik) new RubikRight(rubik)).half()
$.. ==>
......010239......
......040542......
......070845......
101112192048363534
131415222351333231
161718252654302928
......373803......
......404106......
......434409......
......464721......
......495024......
......525327......

jshell> ((Rubik) new RubikDown(rubik)).left()
$.. ==>
......010203......
......040506......
......070809......
101112192021282930
131415222324313233
252627343536484746
......394245......
......384144......
......374043......
......181716......
......495051......
......525354......

jshell> ((Rubik) new RubikDown(rubik)).right()
$.. ==>
......010203......
......040506......
......070809......
101112192021282930
131415222324313233
484746161718252627
......434037......
......444138......
......454239......
......363534......
......495051......
......525354......

jshell> ((Rubik) new RubikDown(rubik)).half()
$.. ==>
......010203......
......040506......
......070809......
101112192021282930
131415222324313233
343536484746161718
......454443......
......424140......
......393837......
......272625......
......495051......
```

```
......525354......

jshell> ((Rubik) new RubikLeft(rubik)).left()
$.. ==>
......190203......
......220506......
......250809......
121518372021282930
111417402324313233
101316432627343536
......463839......
......494142......
......524445......
......014748......
......045051......
......075354......

jshell> ((Rubik) new RubikLeft(rubik)).right()
$.. ==>
......460203......
......490506......
......520809......
161310012021282930
171411042324313233
181512072627343536
......193839......
......224142......
......254445......
......374748......
......405051......
......435354......

jshell> ((Rubik) new RubikLeft(rubik)).half()
$.. ==>
......370203......
......400506......
......430809......
181716462021282930
151413492324313233
121110522627343536
......013839......
......044142......
......074445......
......194748......
......225051......
......255354......

jshell> ((Rubik) new RubikBack(rubik)).left()
$.. ==>
......161310......
......040506......
......070809......
431112192021282901
441415222324313202
451718252627343503
......373839......
......404142......
......363330......
......485154......
......475053......
......464952......

jshell> ((Rubik) new RubikBack(rubik)).right()
$.. ==>
......303336......
......040506......
......070809......
031112192021282945
021415222324313244
011718252627343543
......373839......
......404142......
......101316......
......524946......
```

```
......535047......
......545148......

jshell> ((Rubik) new RubikBack(rubik)).half()
$.. ==>
......454443......
......040506......
......070809......
361112192021282916
331415222324313213
301718252627343510
......373839......
......404142......
......030201......
......545352......
......515049......
......484746......

jshell> /exit
```

Level 5

Finally, read the Rubik's cube as input, followed by the turns and output the final Rubik's cube after all the turns have been made.

A skeleton `Main` class has been provided for you that parses the input and creates the first `RubikFront` object. You will only need to modify the `oneMove` method to construct the appropriate `Rubik` object based on the first character, and then invoke the correct turn.

```
import java.util.Scanner;

class Main {
    static final int NFACES = 6;
    static final int NROWS = 3;
    static final int NCOLS = 3;

    static Rubik oneMove(Rubik rubik, String move) {
        System.out.print("Face " + move.charAt(0));

        if (move.length() == 1) {
            System.out.println(" right turn");
        } else {
            if (move.charAt(1) == '\'') {
                System.out.println(" left turn");
            } else {
                System.out.println(" half turn");
            }
        }

        return rubik;
    }


    public static void main(String[] args) {
        int[][][] grid = new int[NFACES][NROWS][NCOLS];

        Scanner sc = new Scanner(System.in);
        for (int k = 0; k < NFACES; k++) {
            for (int i = 0; i < NROWS; i++) {
                for (int j = 0; j < NCOLS; j++) {
                    grid[k][i][j] = sc.nextInt();
                }
            }
        }
        Rubik rubik = new RubikFront(grid);

        while (sc.hasNext()) {
            rubik = oneMove(rubik, sc.next());
        }
        System.out.println(rubik);
    }
}
```

The following is a sample run of the program. User input is underlined.

```
1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45
46 47 48 49 50 51 52 53 54
F
R'
U2

......541518......
......510504......
......480201......
303336455352101137
131438262306293235
161739272412070809
......343119......
......404120......
......434421......
......464728......
......495042......
......032225......
```

```
1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45
46 47 48 49 50 51 52 53 54
F R U L B D
F' R' U' L' B' D'
F2 R2 U2 L2 B2 D2

......281118......
......220508......
......270648......
211539342936452037
401451332344473253
014207192603543143
......123830......
......354124......
......104916......
......521346......
......175002......
......090425......
```

Check the format correctness of the output by typing the following Unix command

```
$ java Main < test5.in
```