Computer Programming Assignment 3

Due date: 2017-11-22 23:59:59

Checkpoints

1. You should do the assignment in your own. You are not allowed to share code with others and/or copy code from other resources. If you are caught, as in the syllabus, you will get a failing grade.
2. Grading will be done in Linux environment using java 1.8.
3. Program failed to compile/run will result 0.
4. Do not infinite loop your program to repeat unless you are told so.
5. Do not change input/output format unless you are told so.
6. Do not color console.
7. Write your name and student number at top of program as a comment

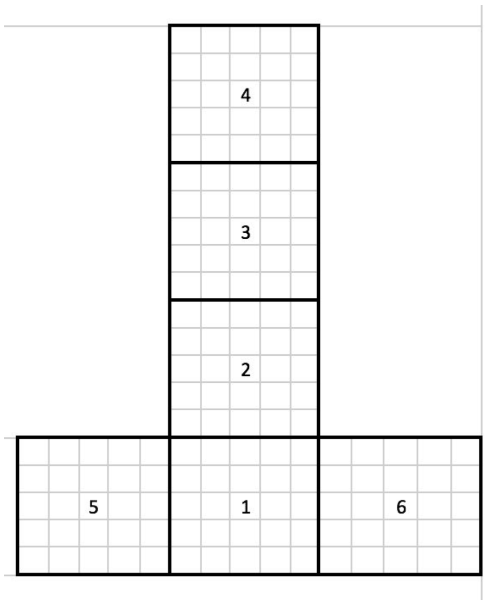
Submission

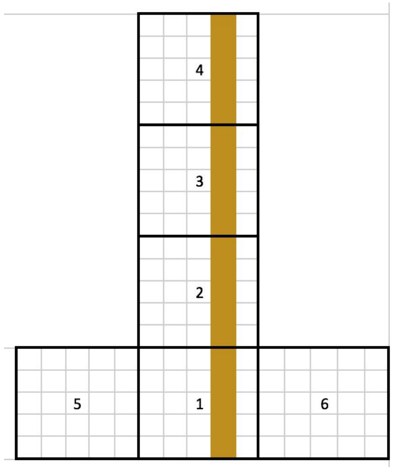
1. Submit your assignment on eTL.
2. Zip your file (or tar) as '<Student ID>-assign3.zip'
3. No late submission is allowed

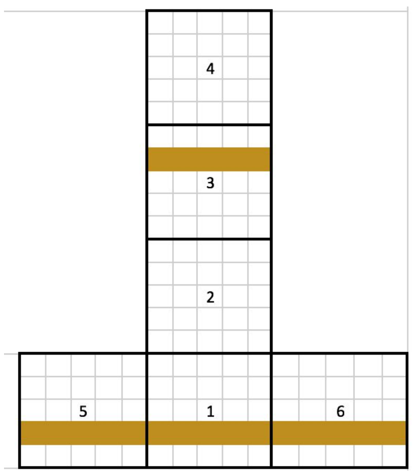
Simulation of n x n x n Rubik's Cube

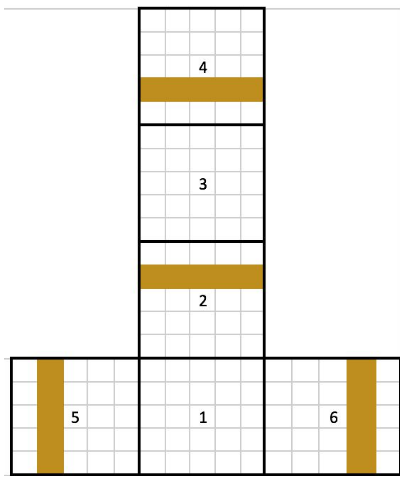
- Write a class Assignment3_Cube.java and Assignment3_Side.java. (Two separated file)
- Assignment3_Cube.java
  - Constructor takes a String, which is the input file name.
    - First line contains dimension, n, of the cube (n x n x n)
      - Create 6 "Assignment3_Side" objects, passing n to the constructor
      - All the squares on `side[i]` will be filed with integer i+1, for 0 <= i <= 5.
    - Remaining lines in the input file contain the rotate operations you need to simulate. Each operation has the following format:
      - [side] [row/column] [direction] [location]
        - Separated by spaces
        - Side tells which side will be rotated
          - Will be an integer between 1 and 6
        - row/column tells if operation rotates by row or column
          - r if row
          - c if column
        - Direction tells which direction it will be rotated
          - If rotating by row, direction will be l or r, meaning left or right
          - If rotating by column, direction will be u or d, meaning up or down
        - Location tells which row/column will be rotated
          - If rotating by row, it will rotate the [location]th row
          - If rotating by column, it will rotate the [location]th column
          - [location] is an integer between 1 to n
  - public int[][] report(int a)
    - This method returns 2-D int array of [a]'s side.
    - Left top would be [0][0] and right bottom will be [n-1][n-1]
    - a is an integer between 1 and 6.
- Assignment3_Side.java
  - Constructor takes two int parameters, n and v
    - One int will determine size of n and object will hold 2-D int array [n][n]
    - v will be initial value for array [0~n-1][0~n-1], where v is between 1 and 6
- Rotate
  - Refer to the images below
  - There will be 3 cases to handle
    - Column operations of side 1, 2, 3, 4
    - Row operations of side 1, 3, 5, 6
    - Column operations of side 5, 6, and row operations of 2, 4

    - See the figures and the explanation given below.
- Main(driver)
    - Main will be created by TA and used for grading. Do not include main in your submission. (Include only Assignment3_Cube.java and Assignment3_Side.java)









More detail on rotate

- Case 1 (1 <= [side] <= 4 and [column])
    - [direction] is u
        - 1's [location] column = 4's [location] column
        - ...
        - 4's [location] column = 3's [location] column
    - [direction] is d
        - 1's [location] column = 2's [location] column
        - ...
        - 4's [location] column = 1's [location] column

- There are cases where you need store [row/column] reverse
- There are cases where [location] is reverse
    - ex.) `1 r r 4`, where `[size]` = 5
    - If 6’s 4th is `[1 2 3 4 5]` then 3’s 2nd will become `[5 4 3 2 1]`
- There are cases where you should handle direction
    - ex.) `1 r r 4` and `3 r l 2`, where `[size]` = 5, will be same operation

ABC zoo management

Management System of a zoo will be simulated here. **You should only implement classes, not 'main' method. (You may test with your own main method before submission.)** 'main' method will be in a class with which TA will use to grade.
"Manage" class will get information from "ManageInfo" class, and will return management result through an object of class "Report". "Manage" class should consider the following conditions and report those "Room"'s that does not satisfy these conditions.

1. A room that has any animal with aggression level 1 should be marked as "careful treatment".
   Report Name: CareTreat
2. A room that has an animal with aggression level 2 should be reported as "dangerous".
   Report Name: Dangerous
3. A Carnivore eats only 'C' type(true) foods, Herbivore eats only 'H' type(false) foods, and an Omnivore can eat any type.
   Report Name: FoodType

Aggression level: 0, 1, 2

- 0 : Not hostile
- 1 : Hostile when attacked by other species
- 2 : Hostile to any other species

When reporting unsatisfied rooms, you should list all room's index and their unsatisfied conditions.
General report format will be like the following: (*[(description)]*: format string, _ : a space)
[Total number of rooms]
Room_*[i]*_:_*[cond j1]*,_*[cond j2]*,_*[cond_j3]*
…
Room_*[n]*_:_*[unsatisfied conditions]*…

e.g.) A room contains two carnivores and one of them has aggression level 2, and foods are 'H' type.
=>
1
Room 1 : Dangerous, FoodType

Class information is specified in detail in the java files which can be downloaded from eTL.
Here is a brief description of essential contents provided by the files. Methods written here must be included in your class without changing parameters, name, and return type, but you can add more just as you follow this rule.

Given classes (you do not have to implement but should know methods):

```
class ManageInfo{
  //Fields: List of Rooms and lists of each room's Animals and Foods.
```

```
   //Only methods that you need to know are listed
  public ManageInfo(){
      //default initialization
  }
  Animal getAnimal(int i, int j){
      //returns an Animal that is located in room[i] and [j]-th
location in the room[i];
  }
  Room getRoom(int i){
      //returns 'i'-th room that is located in room[i] and [j]-th
location in the room[i];
  }
  boolean getFood(int i, int j){
      //returns a food - true means 'C' type, the other means 'H'
type.
  }
}

class Report{
//fields
//Constructors (including default)
}
```

Classes you should implement:

```
class Manage{
//you can add more
   public Report getReport(){
      //your final report result must be returned by this method.
   }
}
class Animal{
   char type;    //Carnivore, Herbivore, or Omnivore
   String speciesName;  //Name of species
   String Name;  //Name of individual

   public String getSpeciesName(){
      //should return speciesName
   }
   public String getName(){
     //should return Name
   }
   public char getType(){
      //should return Type of animal
   }
   //you can add more here
}

class Room{
  //should contains lists of Animals and Foods
  //You can add more fields or methods as you want.

  public Animal getAnimal(int index){
    //you can add more without changing method parameters and name
  }
  public boolean getFood(int index){
    //you can add more without changing method parameters and name
  }
}

class Carnivore extends Animal{
  //you can add more without changing specified method parameters and
name
  public String getSpeciesName(){
    //should add "(Carni)" at the end of original speciesName
    //e.g.) "Lion" -> "Lion(Carni)"
  }
}
class Herbivore extends Animal{
   //you can add more without changing method parameters and name

   public String getSpeciesName(){
    //should add "(Herbi)" at the end of original speciesName
    //e.g.) "Elephant" -> "Elephant(Herbi)"
  }
}
class Omnivore extends Animal{
  //you can add more without changing method parameters and name

   public String getSpeciesName(){
    //should add "(Omni)" at the end of original speciesName
    //e.g.) "Chimpanzee" -> "Chimpanzee(Omni)"
   }
}
```

Submission

- You should make a zip file named as “Assign3_2.zip” that contains all your java files including essential java files.

Essential java files are all files specified above such as Manage,java, Animal.java etc.

ManageInfo.java and Report.java does not have to be included.

- e.g.) Assign3_2.zip : Manage.java, Room.java, Animal.java, Carnivore.java, Herbivore.java, Omnivore.java, Food.java