CS102A Spring 2020 Assignment6

Designer: ZHU Yueming (General Design)

DENG Songhang (Junit), TANG Jiahao (Document), WANG Lishuang (Test)

Keyword Annotation:

Field(s), **Constructor(s)**, **Method(s)** You can see something you need to define or modify after them.

Hint(s) You can see other hints about designing classes.

Part 1. Simple Cinema System

General Description of Movie Reservation System

The movie reservation system manages **OrdinaryMovie**s, **ThreeDMovie**s and other kinds of **Movie**. You can check out all the movies in theaters and reserve tickets with this system. This system will arrange the times for the newly added movies. It also records incomes of this cinema.

It is only a simple Movie Reservation System for exercising the interface, abstract class, inheritance,
polymorphism, etc. The implement of Movie Reservation System in real world would be more complex than our assignment.

You need to define or modify following classes

1. class Time

Use 24-hour clock

Fields:

- `private int hour`
- `private int minute`

Constructor:

- `public Time(int hour, int minute)`

  Initializes a newly created Time object using specific hour and minute.

Methods:

- `public String toString()`

Override method which **return**s a format type of **String** as "**hour:minute**" such as 01:15, 10:02 or 23:59.

Hints:

- Do not modify or remove any methods or fields that have been already defined.
- You can add other methods or attributes that you think are necessary.

2. class Movie

It is an **abstract** class, which is the *super* class of other Concrete Movie classes.

Fields:

- `private int id`

  Each movie has its own unique ID, which starts from **1** and increased by 1 in each instantiating.

- `private String name`

  The name of the movie

- `private Time startTime`

  The opening time of the movie

- `private int runtime`

  The length of a movie (minutes)

- `private double price`

  Ticket price of the movie

- `protected int ticketsLeft`

  Spare tickets for the movie, and the initial value of which should be the **capacity of corresponding movie hall**.

Methods:

- `public abstract double purchase(int arg)`

  Represent the ticket operation and return the price, the parameters will be explained in its subclasses.

- `public String toString()`

  Use following code to override `toString()` method

```
@Override
public String toString() {
    return
        "id=" + id + ", name='" + name +
        "', startTime:" + startTime +
        ", runtime=" + runtime +
        ", price=" + price +
        ", ticketsLeft=" + ticketsLeft;
}
```

Hints:

- Do not modify or remove any methods or fields that have been already defined.
- You can add other methods or attributes that you think are necessary.

3. Class OrdinaryMovie and ThreeDMovie

Fields:

- `private final int GLASS_PRICE = 20`

  Only defined in class **ThreeDMovie**.

Methods:

- `public String toString()`

  Override this method in each subclass. The outputs must append the class name at the end of the `toString()` method in super class. (separated by only *one* **space**)

  For example:

  ```
  id=2, name='name2', startTime:06:25, runtime=125, price=60.0,
  ticketsLeft=15 OrdinaryMovie
  id=3, name='name3', startTime:16:15, runtime=125, price=58.5,
  ticketsLeft=4 ThreeDMovie
  ```

- `public double purchase(int arg)`

  Override this abstract method in each subclass. The way to calculate purchase is shown below:

  - Ordinary Movie

    `arg` represents the number of tickets which will be bought. If there are no enough tickets, it will buy all the tickets left, remaining orders will not be considered.

    Example: There are 10 tickets, if you want to buy 3 of them, it will return $3 * times price$, but if you want to buy 20 tickets, it will only return $10 * times price$.

  - Three-D Movie

`arg` only has two possible values: 0 and 1 which represent the need for 3D glasses (1: need, 0: not need). That is, invoking this method one time can only buy one ticket. The return value is price plus the cost of glasses or only the price.

4. class ConcreteCinema

It is a concrete class of the interface **Cinema**, in which you need to implement all abstract methods that are declared in the interface Cinema. Please read the annotations in *Cinema.java* carefully and all parameters, return values, functions of each method should be strictly followed the description in annotations. In class **ConcreteCinema**, the following important parameters must be defined. Beyond that, you can define any attributes that you think are important.

Field:

- `List<Movie> movies`

  Contain all Movies, including all different types such as **OrdinaryMovie** and **ThreeDMovie**.

5. You can add other Classes that you think are necessary.

If possible, you need add a class that represents **movie hall**, but whether add the class is determined by your design.

Other Important parameters and Test cases

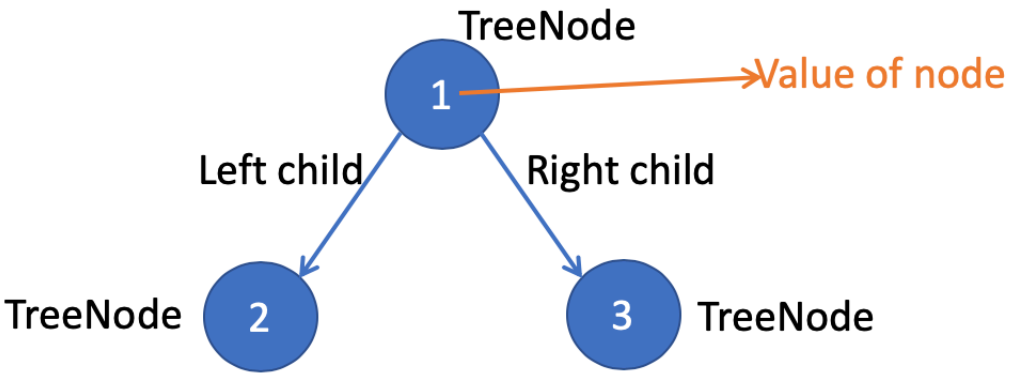
1. Start time of movies in all test cases are between 00:00 and 24:00, viz. all operations take place in **one** day only.
2. All test cases of arguments that represent the hall number or movie id will not be exceeded the max number of current halls and current movie id.

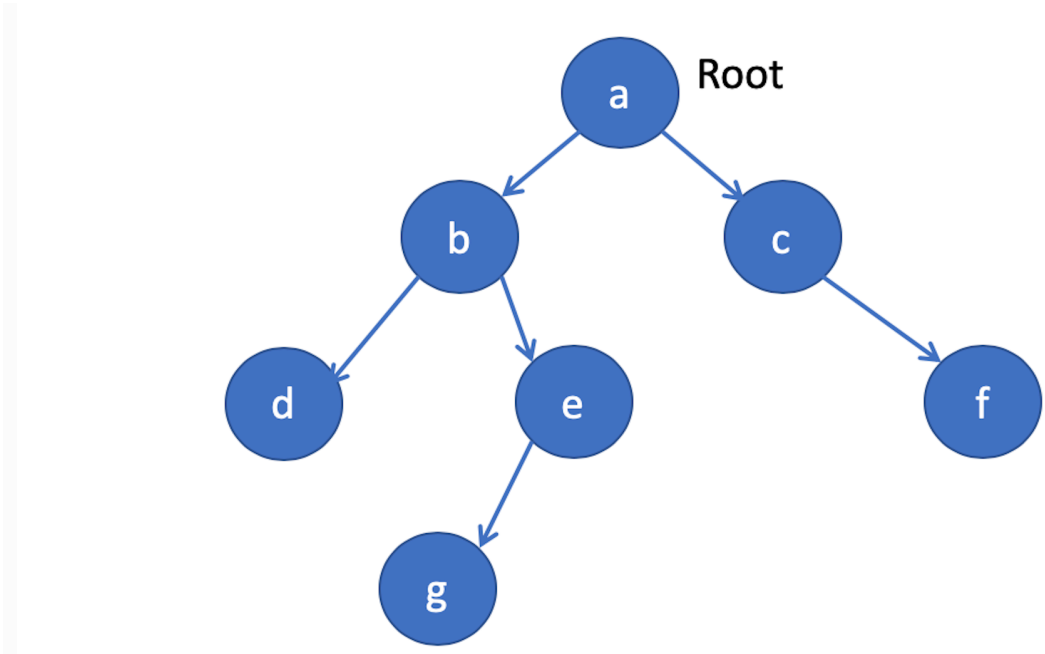
Part2 Design Your Binary Tree

General Description

A tree structure is an important data structure in computer science which you can image as an inverted tree. A binary tree is a simple and useful kind of tree structure.

A binary tree has many simple structures called node which stores its **value** and **child node**. Each node in a binary tree only has two child node at most, which are called left child and right child. You can understand a node in binary tree from the graph below:





1. Class MyBinaryTree <T>

Fields:

- `private [your node type] root;`

  A private field named **root** represents the root node of the binary tree. It is a node type (defined by yourself), which contains the value of node, the left child and right child, but not a generic type of T

- `private int size`

  A field represents how many nodes in the binary tree.

Constructor:

- `public MyBinaryTree(T root)`

  a one-parameter constructor, only one argument which represents the value of root node in the binary tree.

We can invoke constructors as follows:

```
MyBinaryTree<String> stringTree = new MyBinaryTree<>("root");
MyBinaryTree<Integer> intTree = new MyBinaryTree<>(1);
```

Methods:

- Getter methods of *root* and *size*

  `public [your node type] getRoot()`, `public int getSize()` which can return those two private fields of `MyBinaryTree<T>` respectively.

  We can invoke those two methods as follows:

  ```
  System.out.println(stringTree.getRoot())
  System.out.println(intTree.getRoot())
  ```

- `public void addTreeNodes(String directions, T[] values)`

  A method with a void return type used for adding and changing values.

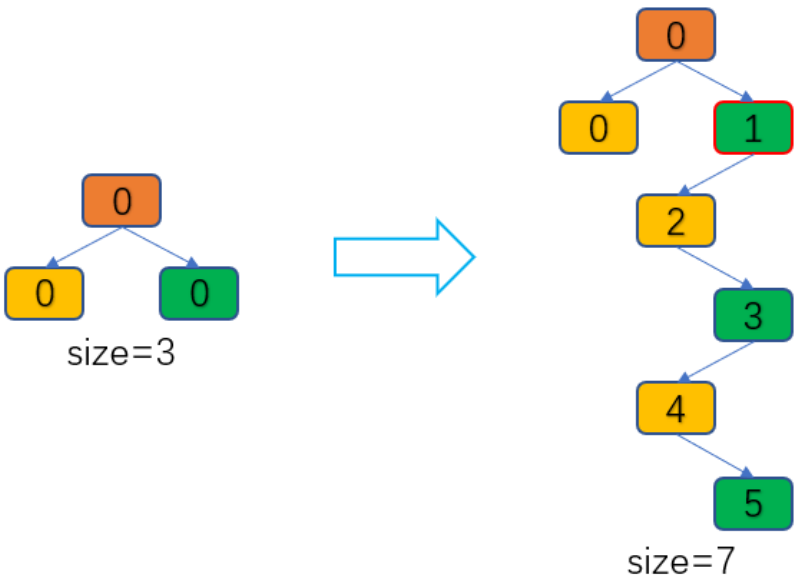
  `directions` is a string of numbers, only contains **0** and **1** in test cases.

  - '0': means visit the left node
  - '1': means visit the right node
  - start from the root node; if there is no child node in required direction, create a new child node (do not forget to expand the **size**), else change the value of the node according to corresponding index of **values**.

  `values` is an array which length is `directions.length()` and contains values need to be assigned

  example:

  ```
  intTree.addTreeNodes("10101", {1, 2, 3, 4, 5})
  ```



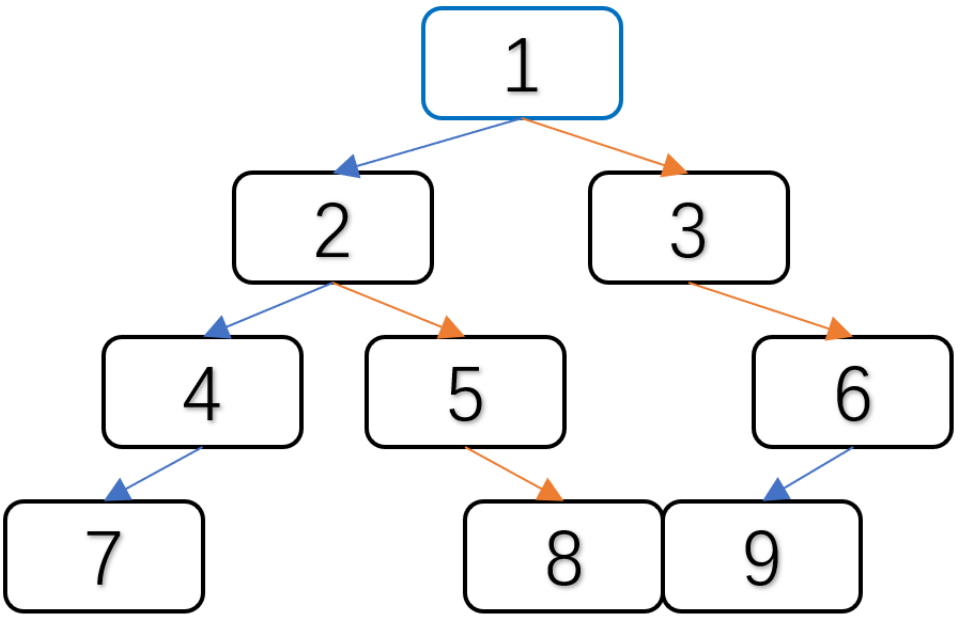
- `public String levelTraverse()` **BONUS**

This method returns values of the tree nodes following the level order from the root to leaves.

The return type is a String with values of tree nodes in level order separated by only one space.

This algorithm is called BFS in computer science. You can get more information from this page.

example:



The level traversal results of this tree is `1 2 3 4 5 6 7 8 9`.

2. You can add other Classes that you think are necessary

Maybe, it is necessary to add a class that represent `TreeNode<T>`, which contains the value, its left child and right child.

Submission of Assignment

1. You should submit all the source code files (with an extension **".java"**).

2. You need submit at least those java files: **Cinema.java**, **Movie.java**, **OrdinaryMovie.java**, **ThreeDMovie.java**, **ConcreteCinema.java**, **Time.java** and **MyBinaryTree.java**.

3. You should submit all source code **directly** into your OJ system, **do not compress** them into one folder.

4. **No Chinese characters** are allowed to appear in your code.

5. **No package** included.

6. The output must strictly follow the description and the sample in this document and the LocalJudgeA6Test.java, and you can only get points for each task only when you pass the test case.