CS109 Fall 2024 Assignment6 Cinema System

> Designer: ZHU Yueming (General Design)
>
> DENG Songhang (Junit), TANG Jiahao (Document), WANG Lishuang (Test)

What to Submit

Time.java

Movie.java

OrdinaryMovie.java

ThreeDMovie.java

Cinema.java

ConcreteCinema.java

Introduction

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.

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.

classes

1. class Time

> - 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.

Use 24-hour clock

Fields:

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

Methods:

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

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

- `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.

2. class Movie

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

> - 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.

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;
  }
  ```

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 $3 * timesprice$, but if you want to buy 20 tickets, it will only return $10 * timesprice$.

- 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. interface Cinema

We provide an interface named Cinema, which only defines abstract methods with no specific implementations. We recommend that you do not modify this interface. The specific implementations need to be pushed down to the implementation class, such as ConcreteCinema.java.

```java
import java.util.List;

public interface Cinema {

    public void addMovieHall(int capacity);

    public List<String> getAllMovieHallsCapacity();

    public void addMovie(String name, int runtime, int hallNumber, double
price, int type, Time startTime);

    public List<Movie> getAllMovies();

    public List<Movie> getMoviesFromMovieHallOrderByStartTime(int hallNumber);

    public double reserveMovie(int movieId, int arg);

    public Movie getMovieById(int movieId);

    public double getOneMovieIncome(int movieId);

    public double getTotalIncome();

    public List<Movie> getAvailableMoviesByName(Time currentTime, String name);
}
```

5. class ConcreteCinema

It is an implement class of the interface **Cinema**, in which you need to implement all abstract methods that are declared in the interface **Cinema**. In class **ConcreteCinema**, the following important parameters must be defined. Beyond that, you can define any attributes that you think are important.

> 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.

Fields:

- `List<Movie> movies`

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

Methods:

- **Constructor** with null parameter.

  ```
  public ConcreteCinema()
  ```

- **addMovieHall**

  ```
  public void addMovieHall(int capacity);
  ```

  This method is used to add a movie hall with `capacity` seats to the cinema.
  the id of the hall starts from `1` and increases one by one.

  Parameters:

  `capacity` - the capacity of the hall.

- **getAllMovieHallsCapacity**

  ```
  public List<String> getAllMovieHallsCapacity();
  ```

  This method is used to get all movie halls with capacities in the cinema.

  Returns:

  a `List<String>` in ascending order of the number of movie hall, which contains the information of each hall.
  the format of each String element in the list is "(index of the hall)-(capacity of the hall)"
  Example: "1-17" in {"1-17", "2-13", "3-23"} 1 represents the hall number, 17 represents the capacity.

- **addMovie**

  ```
  public void addMovie(String name, int runtime, int hallNumber, double
  price, int type, Time startTime);
  ```

  This method is used to add movie to the cinema.
  Each movie added successfully has its own id starting from 1 and increasing one by one.
  Caution:
  The movie hall needs `10` minutes to prepare before playing a movie and also needs `10` minutes to clean after playing a movie,

so there should be at least `20` minutes between the two movies.
(ending time of last movie + `20` min) <= (starting time of adding movie)
And
(ending time of adding movie + `20` min) <= (starting time of next movie)
If the adding fails, you do not need to deal with this adding, just discard it.
After successfully adding a movie, the number of tickets of this movie equals the `capacity` of the movie hall.

Parameters:

`name` - the name of the movie

`runtime` - the movie duration counted in minutes

`hallNumber` - the id of the movie hall used to play films

`price` - the price of the movie

`type` - an Integer (0 or 1) representing the type of the movie :

  - 0 --> OrdinaryMovie
  - 1 --> ThreeDMovie

`startTime` - a `Time` type of the start time of the movie

- **getAllMovies**

```
public List<Movie> getAllMovies();
```

This method is used to get all movies in the cinema.

Returns:

a `List<Movie>` in ascending order of the id.

- **getMoviesFromMovieHallOrderByStartTime**

```
 public List<Movie> getMoviesFromMovieHallOrderByStartTime(int hallNumber);
```

This method is used to get all movies in specific movie hall sorted by starting time of the movies.

Parameters:

`hallNumber` - the id of the movie hall

Returns:

a `List<Movie>` sorted by starting time in ascending order

- **reserveMovie**

```
public double reserveMovie(int movieId, int arg);
```

This method is used to reserve movie.

Parameters:

`movieId` - the id of the movie

`arg` - an Integer of the reservation number or whether need 3D glasses.

- the movie is ordinary movie, arg is the reservation number.
  If there are no enough tickets, it will buy all the tickets left, remaining orders will not be considered.
- the movie is 3D movie, arg (0 or 1) represents whether need 3D glasses, and only reserve 1 ticket.
  - 0 --> do not need glasses
  - 1 --> need glasses

Returns:

the price needed to pay for reservation
If there are insufficient tickets of this movie, return 0.

- **getMovieById**

```
public Movie getMovieById(int movieId);
```

This method is used to get the movie by its id.

Parameters:

`movieId` - the id of the movie

Returns:

a `Movie` type of specific movie of given id.

- **getOneMovieIncome**

```
 public double getOneMovieIncome(int movieId);
```

This method is used to get total income of specific movie.

Parameters:

`movieId` - the id of the movie.

Returns:

the income of specific movie.

- **getTotalIncome()**

```
public double getTotalIncome();
```

This method is used to get total income of the cinema.

Returns:

the total income of the current cinema.

- **getAvailableMoviesByName**

```java
public List<Movie> getAvailableMoviesByName(Time currentTime, String name);
```

This method is used to get the movie which can be reserved.
The same movie could be shown at different times.
If the start time of a movie is after `currentTime` and rest of the ticket is larger than 0, the movie will be reserved in the list.

Parameters:

`currentTime` - `Time` type representing current time

`name` - the movie name

Returns:

a `List<Movie>` of available movies, sorted by its startTime in ascending order