Practice Lab Assignment 5

Practice Lab Assignment 5

For this Practice Lab Assignment, you will write programs based on the concepts of Inheritance, Abstract Class, Interface and Packages.

Instructions

- There are 4 questions in this assignment.
- Do not share your work with anyone.
- Discuss with TA in case of any further clarifications.

Due Date: Show your codes to TA to get the attendance and Submit your remaining codes on BB.

Submission Format: A zip file containing all .java files.

Grading Criteria

No Grading Criteria.

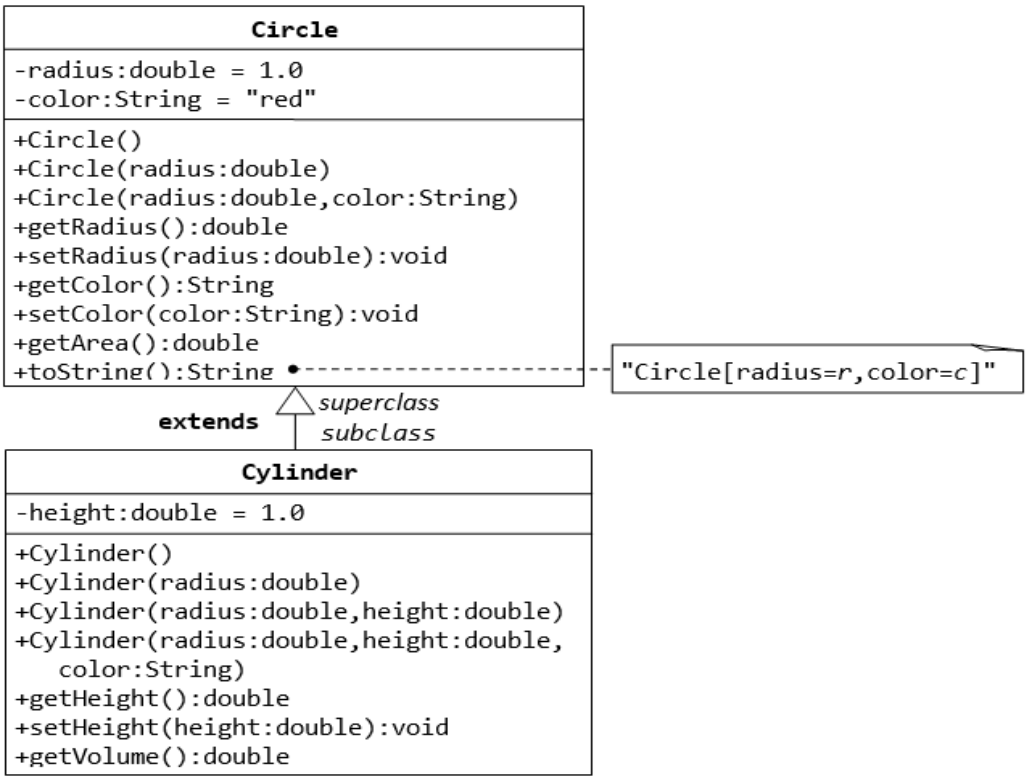
Questions

1. A stack is a data structure that holds objects in a last-in, first-out fashion. A queue is a data structure that holds objects in a first-in, first-out fashion. Design a queue class for integers, named **QueueOfIntegers**, with the following features:

- A private data field of type **int[]** to store data in the queue.
- A method named **dequeue()** to retrieve and remove the first element from the queue.
- A method named **enqueue(int)** to add an element to the queue.

Implement the class. Write a client program that tests all methods in the class.

2. Design the class Circle, and Class Cylinder, as shown in the figure. Write a test program (says TestCylinder) to test the Cylinder class created.



Also, Override the toString method in Cylinder class to display the properties of Cylinder.

3. Design a class named **Person** and its two subclasses named ***Student*** and ***Employee***. Make ***Faculty*** and ***Staff*** subclasses of ***Employee***. A person has a name, address, phone number, and email address. A student has a class status (freshman, sophomore, junior, or senior). Define the status as a constant. An employee has an office, salary, and date hired. A faculty member has office hours and a rank. A staff member has a title. Override the toString method in each class to display the class name and the person's name.

Implement these classes and write a test program that creates a Person, Student, Employee, Faculty, and Staff, and invokes their toString() methods.

4. A gaming console allows you to plug in various games and play any of them using a standard interface that has a set of buttons or similar input mechanisms. We cannot afford to buy one of these boxes, and so we will try a simple software simulation of such a console.

Consider a Console that has a Start button and 4 arrow keys - up, down, left, right. The player chooses a game to play, presses Start, and then does a series of actions by pressing

one of the 4 arrow keys. The console sends these commands to the chosen Game by invoking an appropriate method of the Game. The game object processes these commands and makes the appropriate changes to its state. In the process, it checks if the player has won/lost after each move, or if the game is over.

Note that the Console does not know anything about the specific game being played - for it, all are variations of Game, and it knows that all of them support a standard set of methods. Games are identified by their name - assume that the names are unique.

The Game object supports the following behaviors:

1. Creating an instance specifying its name
2. Start the game
3. Process an action - takes an argument that specifies which of the 4 arrow keys are pressed - a character which is one of 'L', 'R', 'U', 'D' representing left, right, up, down key presses. This method returns a String which is one of "Won", Lost", "Game Over" or "None"
4. Keeping track of the number of keystrokes/commands so that it can detect when the game is over

We have 2 games implemented, called "FlightSimple" and "RandomWalk". (Note: these are toy games and not to be analyzed for their content!). FlightSimple is a simple flight simulator. The objective is to fly a plane and land it back safely within the number of keystrokes allowed. It interprets the 4 keys as follows:

- R: increase speed by 1 unit
- L: reduce speed by 1 unit. Speed cannot go below 0. Reducing the speed to 0 when you are flying causes the plane to crash and you have lost the game.
- U: increase altitude by 1 unit. Altitude can be increased from 0 only if the speed is greater than 2 units. Otherwise, this command is ignored.
- D: reduce height by 1 unit. Reducing below 0 results in a crash and you have lost the game
- Getting the height to 0 and then the speed to 0 means you have landed successfully and have won the game.

RandomWalk is a simple and boring game where the user is on a 20x20 grid (of tiles), and starts at position 10,10 (rows and columns of tiles are numbered 0 to 19 in each direction). Each key press moves the player by 1 unit in the appropriate direction - left, right, up or down. If you hit the top limit (i.e. go past 19) moving to the top or to the

right, then you have lost the game. If you touch the bottom limit (row 0), you have won the game. Hitting the left limit is not an issue, though you cannot go further left.

To play any of these games, the user informs the console about the choice of game, as well as the max number of key presses (you have to pay for each keystroke, so you might want to limit this to some reasonable number). Then the user starts the game, and presses one of the 4 keys in any order. The console reports the status when the player either wins or loses the game, or the game is over (when the limit on keystrokes is reached), and then moves on to the next game if one exists

Model this as a set of Java classes:

1. GameConsole which reads the user input and invokes appropriate methods on the Games

2. Game: that exposes the interfaces to start and process a key, and returns one of the results mentioned above.

3. Classes to implement RandomWalk and FlightSimple, which provide the specific methods to handle keystrokes

4. A driver/main class that sets up the Console, RamdomWalk, and FlightSimple classes, and starts the console (which, in turn, starts the game and reads the input).

Note that the Console class should not know about specific types of games, but only thinks of them all as games. It uses the name of the game to identify the game to be invoked. An obvious requirement is that we should be able to add new kinds of games to a console, and the only code that should change is the main/driver class, and, of course, the implementation of the new game. The user actions are specified through the input as defined below. Each round starts by specifying the name of the game and the number of keystrokes allowed. This is followed by a series of characters, one of L, R, U, D.

An End as the name of the game implies the end of input

Notes:

The intent is that you will define the base and derived classes, and work out which, if any, methods are overridden. Which base class methods, if any, should be abstract. Also, what functionality will be handled at the base class, and what will be done by the derived classes.

Sample Input:

FlightSimple 10
R R R U U R D L D D
RandomWalk 11

R R R U U R D L D D R
RandomWalk 20
R R R U U R D L D U R R U U U U U U U U U
FlightSimple 20
R R U R U R R U D D L D L L L L L L L L
End

Output

FlightSimple: Lost
RandomWalk: Game Over
RandomWalk: Lost
FlightSimple: Lost