**Practical Lab Week 7**

Objective: the objective of this lab is to practice abstract classes, interfaces, and inner classes.

1.

A.

public class Circle extends Shape {

private double radius;

public Circle(double radius){

this.radius = radius;

}

@Override

public double computeArea(){

return radius \* radius \* Math.*PI*;

}

@Override

public double computePeri(){

return 2 \* radius \* Math.*PI*;

}

@Override

public void print(){

System.*out*.println("The circle is at " + "(" + this.x + "," + this.y + ")");

}

}

public class Rectangle extends Shape {

private double width;

private double length;

public Rectangle(double width, double length){

this.width = width;

this.length = length;

}

@Override

public double computeArea(){

return this.width \* this.length;

}

@Override

public double computePeri(){

return 2 \* (this.width + this.length);

}

@Override

public void print(){

System.*out*.println("The rectangle is at " + "(" + this.x + "," + this.y + ")");

}

}

B.

public class MoveableCircle extends Circle implements Moveable {

public MoveableCircle(double x, double y, double radius){

super(radius);

this.x = x;

this.y = y;

}

@Override

public void move(double x, double y){

this.x += x;

this.y += y;

}

}

public class MoveableRectangle extends Rectangle implements Moveable {

public MoveableRectangle(double x, double y, double width, double length){

super(width, length);

this.x = x;

this.y = y;

}

@Override

public void move(double x, double y){

this.x += x;

this.y += y;

}

}

2.

A.

public class Movie implements Comparable<Movie> {

private double rating;

private String name;

private int year;

public Movie(String name, double rating, int year) {

this.name = name;

this.rating = rating;

this.year = year;

}

@Override

public int compareTo(Movie anotherMovie) {

return this.year - anotherMovie.year;

}

public double getRating() { return rating; }

public String getName() { return name; }

public int getYear() { return year; }

}

B.

public class NameComparator implements Comparator<Movie> {

@Override

public int compare(Movie movie1, Movie movie2) {

return movie1.getName().compareTo(movie2.getName());

}

}

C.

public class RatingComparator implements Comparator<Movie> {

@Override

public int compare(Movie movie1, Movie movie2) {

// Solution 1

double EPSILON = 1e-8;

if (Math.abs(movie1.getRating() - movie2.getRating()) < EPSILON) {

return 0;

} else if (movie1.getRating() < movie2.getRating()) {

return -1;

} else {

return 1;

}

// Solution 2

//return Double.compare(movie1.getRating(), movie2.getRating());

}

}

D.

import java.util.ArrayList;

import java.util.Collections;

public class Main {

public static void main(String[] args) {

ArrayList<Movie> list = new ArrayList<Movie>();

list.add(new Movie("Force Awakens", 8.3, 2015));

list.add(new Movie("Star Wars", 8.7, 1977));

list.add(new Movie("Empire Strikes Back", 8.8, 1980));

list.add(new Movie("Return of the Jedi", 8.4, 1983));

System.out.println("Sort by year : ");

Collections.sort(list);

for (Movie movie: list) {

System.out.println(movie.getName() + " " +

movie.getRating() + " " +

movie.getYear());

}

System.out.println("Sorted by rating : ");

RatingComparator ratingCompare = new RatingComparator();

Collections.sort(list, ratingCompare);

for (Movie movie: list)

System.out.println(movie.getRating() + " " +

movie.getName() + " " +

movie.getYear());

System.out.println("Sorted by name : ");

NameComparator nameCompare = new NameComparator();

Collections.sort(list, nameCompare);

for (Movie movie: list)

System.out.println(movie.getName() + " " +

movie.getRating() + " " +

movie.getYear());

}

}

3.

import java.util.ArrayList;

import java.util.List;

public class TextEditor {

private int lineLength;

private List<Line> lines;

private static class Line {

private final List<Character> chars;

private final int length;

Line(int length) {

this.chars = new ArrayList<Character>();

this.length = length;

}

// Return true if there is no more space in the line.

boolean isFull() {

return chars.size() == length;

}

// Return true if there are no characters in the line.

boolean isEmpty() {

return chars.size() == 0;

}

// Add a new character to the end of the line.

// When the line is full, exception is thrown.

void append(char t) {

if (isFull()) {

throw new RuntimeException("Line is full");

}

chars.add(t);

}

// Remove the last character at the end of the line.

// When the line is empty, exception is thrown.

void deleteLast() {

if (isEmpty()) {

throw new RuntimeException("Line is empty");

}

chars.remove(chars.size() - 1);

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

for (char t : chars) {

sb.append(t);

}

return sb.toString();

}

}

public TextEditor(int lineLength) {

this.lineLength = lineLength;

this.lines = new ArrayList<Line>();

}

public void append(char t) {

if (lines.isEmpty() || lines.get(lines.size() - 1).isFull()) {

lines.add(new Line(lineLength));

}

Line current = lines.get(lines.size() - 1);

current.append(t);

}

public void delete() {

if (!lines.isEmpty()) {

Line current = lines.get(lines.size() - 1);

current.deleteLast();

if (current.isEmpty()) {

lines.remove(current);

}

}

}

// Render the content of the text editor on the screen.

public void render() {

for (Line line : lines) {

System.out.println(line);

}

}

public static void main(String[] args) {

TextEditor editor = new TextEditor(10);

int numChars = 28;

System.out.println("\* Content after insertion:");

for (int i = 0; i < numChars; i++) {

int charCode = 'a' + (i % ('z' - 'a' + 1)); // ASCII code of the next char

editor.append((char) charCode);

}

editor.render();

System.out.println("\* Content after deletion:");

for (int i = 0; i < numChars - 5; i++) {

editor.delete();

}

editor.render();

}

}