**Design Elevator**

Describe the approach to design the elevator using OOPS concepts. Consider all edge cases in your design.

package Elevator;

public class Constants {

public static final int FLOORS = 40;

public static final int ELEVATORS = 16;

public static final int NUM\_PEOPLE = 10;

public static final int MAX\_PEOPLE = 5;

public static final int MIN\_FLOOR = FLOORS + 1;

public static final int MAX\_FLOOR = -1;

}

package Elevator;

public interface IElevatorController {

public void status ();

public Elevator getElevator(int elevatorID);

public void update(int elevatorId, int floor);

public void pickup(int floor , boolean direction);

public void reset(int elevatorId, int floor);

public void step();

}

package Elevator;

public interface ElevatorEventListener {

public void onStopped(Object sender);

}

package Elevator;

public class Elevator {

public enum DIRECTION {

NONE, UP, DOWN

}

private DIRECTION direction = DIRECTION.NONE;

private Boolean move = false;

private boolean [] floors;

private int countUp = 0;

private int countDown = 0;

private int cf = 0;

private int min = Constants.MIN\_FLOOR;

private int max = Constants.MAX\_FLOOR;

private int numFloors;

private ElevatorEventListener elEventListener;

public Elevator(int numFloors) {

if(numFloors<0) throw new IllegalArgumentException();

this.numFloors = numFloors;

floors = new boolean [numFloors];

}

public Integer getCurrentFloor() {

return cf;

}

public int getGoalFloor() {

if(direction == DIRECTION.UP ) return max;

if(direction == DIRECTION.DOWN ) return min;

return -1;

}

public void moveNext(){

if(!move) {

move = (direction != DIRECTION.NONE);

return;

}

if(direction == DIRECTION.UP) {

if(floors[++cf]) {

floors[cf] = false;

if(--countUp == 0) {

direction = (countDown == 0)?(DIRECTION.NONE):(DIRECTION.DOWN);

max = Constants.MAX\_FLOOR;

}

move = false;

if(elEventListener != null) elEventListener.onStopped(this);

}

return;

}

if (direction == DIRECTION.DOWN) {

if(floors[--cf]) {

floors[cf] = false;

if(++countDown == 0) {

direction = (countUp == 0)?(DIRECTION.NONE):(DIRECTION.UP);

min = Constants.MIN\_FLOOR;

}

move = false;

if(elEventListener != null) elEventListener.onStopped(this);

}

}

}

public void setGoalFloor(int gf) {

if((gf<0) || (gf >= numFloors ) ) throw new IllegalArgumentException();

if(cf == gf) return;

if(floors[gf]) return;

floors[gf] = true;

if(gf>cf) { countUp++; max = (gf>max)?(gf):(max); }

if(gf<cf) { countDown--; min = (gf<min)?(gf):(min); }

if(direction == DIRECTION.NONE)

direction = (gf>cf)?(DIRECTION.UP):(DIRECTION.DOWN);

}

public void reset() {

cf = countUp = countDown = 0;

move = false;

direction = DIRECTION.NONE;

floors = new boolean [numFloors];

max = Constants.MAX\_FLOOR; min = Constants.MIN\_FLOOR;

}

public void moveToFloor(int floor) {

if((floor<0) || (floor >= numFloors ) ) throw new IllegalArgumentException();

reset();

cf = floor;

}

public boolean getMove() {

return move;

}

public DIRECTION getDirection() {

return direction;

}

public void setElEventListener(ElevatorEventListener elEventListener) {

this.elEventListener = elEventListener;

}

}

package Elevator;

import java.util.ArrayList;

import java.util.NoSuchElementException;

import edu.princeton.cs.algs4.Queue;

import edu.princeton.cs.introcs.StdOut;

import edu.princeton.cs.introcs.StdRandom;

public class ElevatorController implements IElevatorController, ElevatorEventListener {

private int numElevators;

private int numFloors;

private Elevator [] elevators = null;

private ArrayList<Queue<Integer>> passengers = null;

private void initElevators(int numElevators, int numFloors) {

if(numElevators<0) throw new IllegalArgumentException();

elevators = new Elevator [numElevators];

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

Elevator el = new Elevator(numFloors);

el.setElEventListener(this);

elevators[i] = el;

}

}

private void initFloors(int numFloors) {

if(numFloors<0) throw new IllegalArgumentException();

passengers = new ArrayList<Queue<Integer>> (numFloors);

for(int i=0;i<numFloors;i++) passengers.add(i, new Queue<Integer>());

}

public ElevatorController (int numElevators, int numFloors) {

initFloors(numFloors);

initElevators(numElevators, numFloors);

this.numElevators = numElevators;

this.numFloors = numFloors;

}

private int calculateRoute(int afloor, int bfloor) {

return Math.abs(afloor - bfloor);

}

private int calculateRoute(int xfloor, int xefloor, int tfloor) {

return calculateRoute(xefloor, tfloor) + calculateRoute(xfloor, tfloor);

}

public Queue<Integer> getPassengers(int floor) {

if((floor<0) || (floor >= numFloors ) ) throw new IllegalArgumentException();

return passengers.get(floor);

}

public void setPassengers(Queue<Integer> pssgrs, int floor) {

if(pssgrs == null) throw new NullPointerException();

if((floor<0) || (floor >= numFloors ) ) throw new IllegalArgumentException();

this.passengers.add(floor, pssgrs);

}

@Override

public void status() {

int i = 0;

for(Elevator el:elevators) {

StdOut.print( " elID = " + i++ + " CurrentFloor = " + el.getCurrentFloor() + " Moving = " + el.getMove() + " DIRECTION = " + el.getDirection() + "\n");

}

}

@Override

public Elevator getElevator(int elevatorId) {

if((elevatorId<0) || (elevatorId >= numElevators ) ) throw new NoSuchElementException();

return elevators[elevatorId];

}

@Override

public void reset(int elevatorId, int floor) {

if((elevatorId<0) || (elevatorId >= numElevators ) ) throw new NoSuchElementException();

Elevator elevator = elevators[elevatorId];

elevator.moveToFloor(floor);

}

@Override

public void pickup(int floor, boolean direction) {

if((floor<0) || (floor >= numFloors )) throw new IllegalArgumentException();

// shufling the order of elevators in the case where most of the elevators are on the same floors and they are picked up at the same time from different floors

// this is going to run several elevators instead of only the first !

int [] elevatorIDs = new int[numElevators];

for(int i=0; i<numElevators; i++) elevatorIDs[i] = i;

StdRandom.shuffle(elevatorIDs);

Elevator.DIRECTION userDirection = (direction) ? (Elevator.DIRECTION.UP):(Elevator.DIRECTION.DOWN);

int minDistance = numFloors;

Elevator closestElevator = null;

int d;

for (int elID : elevatorIDs ) {

Elevator elevator = elevators[elID];

if( (elevator.getMove() == false) ||

((userDirection == Elevator.DIRECTION.UP) && (elevator.getDirection() == Elevator.DIRECTION.UP) && (floor >= elevator.getCurrentFloor())) ||

((userDirection == Elevator.DIRECTION.DOWN) && (elevator.getDirection() == Elevator.DIRECTION.DOWN) && (floor <= elevator.getCurrentFloor())) )

d = calculateRoute(floor,elevator.getCurrentFloor());

else

d = calculateRoute(floor, elevator.getCurrentFloor(), elevator.getGoalFloor());

if(d<minDistance) {

minDistance = d;

closestElevator = elevator;

}

}

closestElevator.setGoalFloor(floor);

}

@Override

public void update(int elevatorId, int floor) {

if((floor<0) || (floor >= numFloors )) throw new IllegalArgumentException();

if((elevatorId<0) || (elevatorId >= numElevators ) ) throw new NoSuchElementException();

Elevator el = elevators[elevatorId];

el.setGoalFloor(floor);

}

@Override

public void step() {

for(Elevator elevator : elevators) elevator.moveNext();

}

@Override

public void onStopped(Object sender) {

/// onboarding waiting people

Queue<Integer> psQueue = getPassengers(((Elevator)sender).getCurrentFloor());

if(psQueue == null) return;

if(psQueue.isEmpty()) return;

for(Integer goalFloor : psQueue){

((Elevator)sender).setGoalFloor(goalFloor);

}

}

}

**Design Parking Lot**

Describe the approach to design the Parking System to park cars. Consider all edge cases in your design.

package Garage;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.Hashtable;

import java.util.LinkedList;

import java.util.Queue;

import java.util.Random;

import Vehicles.Car;

import Vehicles.Motorbike;

import Vehicles.Vehicle;

public class Garage {

public static class Pair {

private int level;

private int lot;

public Pair(int level , int lot) {

this.level = level;

this.lot = lot;

}

public int getLevel() {

return level;

}

public int getLot() {

return lot;

}

}

private int [] parkingSpaces;

private LinkedList<Vehicle> [] garage;

private Hashtable<Integer , Pair> idToLots = new Hashtable<Integer, Pair> ();

@SuppressWarnings("unchecked")

public Garage(int [] parkingSpaces) throws IllegalArgumentException {

if( parkingSpaces == null || parkingSpaces.length < 1 ) throw new IllegalArgumentException("Invalid parking space!");

for(Integer i:parkingSpaces)

if(i<=0)

throw new IllegalArgumentException();

this.parkingSpaces = parkingSpaces;

garage = new LinkedList [parkingSpaces.length];

for(int i = 0; i<parkingSpaces.length;i++)

garage[i] = new LinkedList<Vehicle>();

}

public boolean enter(Vehicle v) {

if(v == null) throw new IllegalArgumentException("Invalid Vehcle argument");

if(idToLots.containsKey(v.getId())) throw new IllegalArgumentException("A Vehicle with this id already exists in the garage!");

for(int i = 0;i<garage.length;i++) {

if((parkingSpaces[i] - garage[i].size()) >0) {

garage[i].add(v);

idToLots.put(v.getId(), new Pair(i,garage[i].size()));

return true;

}

}

return false;

}

public Vehicle exit() {

for(int i = garage.length-1;i>=0;i--) {

if(garage[i].size() > 0) {

idToLots.remove( garage[i].getFirst().getId());

return garage[i].remove();

}

}

return null;

}

public Pair getParkingPlace(int vehicleId) {

return idToLots.get(vehicleId);

}

public int getFreeParkingLots() {

int sum = 0;

for(int i = 0; i< garage.length;i++) {

sum += parkingSpaces[i] - garage[i].size();

}

return sum;

}

public static final String EXIT\_COMMAND = "exit";

public static void main(String[] args) {

try {

int [] parkingSpaces = {5,4,3};

Garage g = new Garage(parkingSpaces);

Queue<Vehicle> vq = new LinkedList<Vehicle>();

final int size = 20;

for(int i = 0;i<size;i+=2) {

vq.add(new Car(1000 + i));

vq.add(new Motorbike(1000 +i+1));

}

Random r = new Random();

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

while (true) {

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

for (int i = 0; i<r.nextInt(vq.size());i++) {

System.out.println(" Vehicle with id = " + vq.peek().getId() + " is entering the garage");

if(!g.enter(vq.poll())) break;

}

System.out.println(" Available lots in the garage are: " + g.getFreeParkingLots());

int id = 1000 + r.nextInt(size);

Pair p = g.getParkingPlace(id);

if(p == null) {

System.out.println(" The vehicle with id = " + id + "is not in the garage!");

} else {

System.out.println(" The place of the vehicle with id = " + id + " is on level : " + p.getLevel() + " and lot : " + p.getLot());

}

for (int i = 0; i<r.nextInt(size-g.getFreeParkingLots()-1);i++) {

Vehicle v = g.exit();

if(v== null) break;

vq.add(v);

System.out.println(" Vehicle with id = " + v.getId() + " is leaving the garage");

}

System.out.println();

System.out.println();

System.out.println("Press enter to continue or type exit!");

System.out.print(">>> ");

String input = br.readLine();

if (input.length() == EXIT\_COMMAND.length() && input.toLowerCase().equals(EXIT\_COMMAND)) {

System.out.println("Terminating .... ");

return;

}

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

} catch (IllegalArgumentException ex) {

System.err.println(ex.getLocalizedMessage());

} catch (Exception ex) {

// TODO :: general case

System.err.println(ex.getLocalizedMessage());

}

}

}

package Vehicles;

public abstract class Vehicle {

private int id;

public int getId() {

return id;

}

public Vehicle(int id) throws IllegalArgumentException {

if(id<=0 ) throw new IllegalArgumentException();

this.id = id;

}

public static void main(String[] args) {

// TODO Auto-generated method stub

}

}

package Vehicles;

public class Car extends Vehicle {

public Car(int id) throws IllegalArgumentException {

super(id);

// TODO Auto-generated constructor stub

}

public static void main(String[] args) {

// TODO Auto-generated method stub

}

}

package Vehicles;

public class Motorbike extends Vehicle {

public Motorbike(int id) throws IllegalArgumentException {

super(id);

// TODO Auto-generated constructor stub

}

public static void main(String[] args) {

// TODO Auto-generated method stub

}

}