Design Document

For the tenth Object Oriented homework

Attention

The sequence of all classes are arranged by alphabetical order. All classes' attribute are public except the class MapException.

Content

一、 ChangeIndex

1. Overview

   Record the change of indexes.

2. Process Specifications

```
public boolean repOK() {
            /*
             * Requires: Nothing.
             * Modifies: Nothing.
             * Effects: Return the true if the rep variant holds for this.
    otherwise return false.
          */
(The repOK method in all class have the same specification so only write
once here)
 public ChangeIndex(Index index, int change)
          /*
          * Requires: Two Index variables.
          * Modifies: Nothing.
          * Effects: Construct a ChangeIndex.
          */
 public Index getIndex()
         /*
          * Requires: Nothing.
          * Modifies: Nothing.
          * Effects: Return the index.
          */
 public int getChange()
         /*
          * Requires: Nothing.
          * Modifies: Nothing.
          * Effects: Return the change.
          */
```

3. Indicated Object

   private Index index

   private int change

4. Abstract Function

   AF(c) = (index, change path), where index = c. index, change path = c. change.

5. Invariance

   c. index != null && 0<=change<=3

二、 Index

1. Overview

Record the index.

2. Process Specifications

```
public int getX()
        /*
         * Requires: Nothing.
         * Modifies: Nothing.
         * Effects: Return the x.
         */
public int getY() {
        /*
         * Requires: Nothing.
         * Modifies: Nothing.
         * Effects: Return the y.
         */
public Index(int x, int y) {
        /*
         * Requires: Two integer.
         * Modifies: Nothing.
         * Effects: Construct a index.
         */
```

3. Indicated Object

    private final int x

    private final int y

4. Abstract Function

    AF(c) = (x,y), where x = c. x, y = c. y.

5. Invariance

    c. x ∈R && c. y ∈R

三、 Light_ctl

1. Overview

    Control all the traffic lights on the simulative road.

2. Process Specifications

```
public Light_ctl(Traffic_light[][] light) {
        /*
         * Requires: two-dimensional array of Traffic_light.
         * Modifies: Nothing.
         * Effects: Initialize the light..
         */
```

3. Indicated Object

    private Traffic_light[][] light

4. Abstract Function

    AF(c) = (light), where light = c. light.

5. Invariance

c. light != null

四、 Map

1. Overview

   Simulate the roads and traffic lights.

2. Process Specifications

```
public Map()
        /*
         * Requires: Nothing.
         * Modifies: Nothing.
         * Effects: Initialize the flows, map_p, changeIndex, map, map2
         */
public static boolean isConnect(Index a, Index b)
        /*
         * Requires: Two indexes which is border upon.
         * Modifies: Nothing.
         * Effects: check this two indexes whether border upon.
         */
public static boolean isConnect_t(Index a, Index b)
        /*
         * Requires: Two indexes which is border upon.
         * Modifies: Nothing.
         * Effects: check this two indexes whether border upon.
         */
public static Vector<Passenger> findPasg(int x, int y)
        /*
         * Requires: Two integers which is an index.
         * Modifies: Nothing.
         * Effects: Find all the passengers near the index which passed
in then return a Vector contains them.
         */
public static Vector<Integer> shortestPath(int x1, int y1, int x2, int y2)
        /*
         * Requires: Four integers which are two indexes.
         * Modifies: Nothing.
         * Effects: Find the shortest path of this two indexes.
         */
public static Vector<Integer> shortestPath_t(int x1, int y1, int x2, int y2)
        /*
         * Requires: Four integers which are two indexes.
         * Modifies: Nothing.
         * Effects: Find the shortest path of this two indexes.
         */
public static int shortestPath2(int x1, int y1, int x2, int y2)
```

```
        /*
          * Requires: Four integers which are two indexes.
          * Modifies: Nothing.
          * Effects: Find the first step of the shortest and least car flow
path of this two indexes and return.
          */
public static int shortestPath2_t(int x1, int y1, int x2, int y2)
        /*
          * Requires: Four integers which are two indexes.
          * Modifies: Nothing.
          * Effects: Find the first step of the shortest and least car flow
path of this two indexes and return.
          */
public static void addReq(int x, int y, Passenger p)
        /*
          * Requires: Two integers which is an index and a passenger.
          * Modifies: Nothing.
          * Effects: Map the passenger into the map_p.
          */
public static void deleteReq(int x, int y, Passenger p)
        /*
          * Requires: Two integers which is an index and a passenger.
          * Modifies: Nothing.
          * Effects: delete the passenger in the map_p.
          */
private boolean init_map()
        /*
          * Requires: Nothing.
          * Modifies: Nothing.
          * Effects: Initialize the map by Map.txt.
          */
private void init_lights()
        /*
          * Requires: Nothing.
          * Modifies: Nothing.
          * Effects: Initialize the light.
          */
private int countConnect(int i, int j)
        /*
          * Requires: Two integers.
          * Modifies: Nothing.
          * Effects: Get the number of connected path.
          */
public synchronized static boolean deletePath(Index co, int num)
```

```
	/*
	 * Requires: An index in map which needs to be changed to the
num.
	 * Modifies: Nothing.
	 * Effects: Delete a path in the map..
	 */
public synchronized static void recoverPath(int i)
	/*
	 * Requires: A number which is a index of changeIndex.
	 * Modifies: Nothing.
	 * Effects: Recover a path in map.
	 */
public static Vector<ChangeIndex> getChanged()
	/*
	 * Requires: Nothing.
	 * Modifies: Nothing.
	 * Effects: Return the changeIndex.
	 */
public static void addFlow(int x, int y, int direction)
	/*
	 * Requires: An Index and a direction..
	 * Modifies: flows.
	 * Effects: Add the flow in corresponding edge.
	 */
public static void addFlow_t(int x, int y, int direction)
	/*
	 * Requires: An Index and a direction..
	 * Modifies: flows.
	 * Effects: Add the flow in corresponding edge.
	 */
public static void minusFlow(int x, int y, int direction)
	/*
	 * Requires: An Index and a direction..
	 * Modifies: flows.
	 * Effects: Minus the flow in corresponding edge.
	 */
public static void minusFlow_t(int x, int y, int direction)
	/*
	 * Requires: An Index and a direction..
	 * Modifies: flows.
	 * Effects: Minus the flow in corresponding edge.
	 */
public static int getFlow(int x, int y, int direction)
	/*
```

```
                    * Requires: An Index and a direction..
                    * Modifies: Nothing.
                    * Effects: Return the flow in corresponding edge.
                    */
        public static boolean haslight(int x, int y)
                /*
                    * Requires: Two integers.
                    * Modifies: Nothing.
                    * Effects: Return the light[x][y].isHas().
                    */
        public static boolean canPass(int x, int y, int di)
                /*
                    * Requires: Three integers.
                    * Modifies: Nothing.
                    * Effects: Return the whether can pass.
                    */
```

3. Indicated Object

```
            private static final int [][] map
            private static final int [][] map2
            private static final int [][] connect
            private static final Traffic_light[][] light
            private static Vector<Passenger>[][] map_p
            private static Vector<ChangeIndex> changeIndex
            private static AtomicIntegerArray flows
```

4. Abstract Function

   AF(c) = (map, connect, light, map_p, changeIndex, flows), where map = c. map + c.map2, connect = c. connect, light = c. light, map_p = c. map_p, changeIndex = c. changeIndex， flows = c. flows.

5. Invariance

   c. map != null && c. connect != null && c. light != null && c. map_p != null && c. changeIndex != null && c. flows != null && c.map2 != null

五、 MapException

1. Overview

   An user-defined exception.

2. Process Specifications

```
        public MapException(String msg)
                /*
                    * Requires: Nothing.
                    * Modifies: Nothing.
                    * Effects: Nothing
              */
```

3. Indicated Object

```
        private static final long serialVersionUID
```

4. Abstract Function
   nothing
5. Invariance
   nothing

六、 Passenger_Monitor

1. Overview
   Simulate the passenger.
2. Process Specifications

```
private void addPSG(Index loc, Index des)
        /*
         * Requires: Two Index variables which indicate the passenger
location and destination.
         * Modifies: Nothing.
         * Effects: Construct a passenger and then add the passenger
request into the passengers.
         */
public Passenger_Monitor(Taxi[] taxis)
        /*
         * Requires: An array of Taxi
         * Modifies: this.taxis
         * Effects: set the taxis
          */
```

3. Indicated Object
   private Taxi [] taxis
4. Abstract Function
   AF(c) = (taxis), where taxis = c. taxis.
5. Invariance
   c. taxis!= null

七、 Passenger

1. Overview
   A fake Passenger ☺.
2. Process Specifications

```
public Passenger(Index location, Index destination)
        /*
         * Requires: Two Indexes.
         * Modifies: Nothing.
         * Effects: Initialize the passenger.
         */
public boolean addTaix(Taxi taxi)
        /*
         * Requires: A taxi.
         * Modifies: Nothing.
```

```
		  * Effects: Add the taxi into taxis.
		  */
	public Taxi selectTaxi()
		/*
		  * Requires: Nothing.
		  * Modifies: Nothing.
		  * Effects: Arrange a taxi to serve this passenger.
		*/
	public Index getLocation()
		/*
		  * Requires: Nothing.
		  * Modifies: Nothing.
		  * Effects: Return the location.
		*/
	public Index getDestination()
		/*
		  * Requires: Nothing.
		  * Modifies: Nothing.
		  * Effects: Return the destination.
		  */
	public String toString()
		/*
		  * Requires: Nothing.
		  * Modifies: Nothing.
		  * Effects: Return the passenger's string.
		  */
```

3. Indicated Object

```
	private Index location
	private Index destination
	private Vector<Taxi> taxis
```

4. Abstract Function

   AF(c) = (location, destination, taxis), where taxis = c. taxis, destination = c. destination, location = c. location.

5. Invariance

   c. taxis!= null && c. location!= null && c. destination!= null

八、 PassengerQuene

1. Overview

   A container of all the fake passengers.

2. Process Specifications

```
	public static void pushPassenger(Passenger p)
		/*
```

```
            * Requires: A passenger.
            * Modifies: Nothing.
            * Effects: if passengers' size less than 400 then add the passenger into passengers.
            */
    public static Passenger pullPassenger()
           /*
            * Requires: Nothing.
            * Modifies: Nothing.
            * Effects: Push a passenger and return.
            */
    public static int getsize()
           /*
            * Requires: Nothing.
            * Modifies: Nothing.
            * Effects: Return the passengers' size now..
            */
```

3. Indicated Object

```
    private static Vector<Passenger> passengers
    private static int size
```

4. Abstract Function

   AF(c) = (passengers, size), where passengers = c. passengers, size = c. size.

5. Invariance

   c. size >= 0

九、 Schedule

1. Overview

   Schedule the passenger.

2. Process Specifications
3. Indicated Object

```
    private static int i = 0
```

4. Abstract Function

   Nothing.

5. Invariance

   c.i >= 0

十、 Taxi_main

1. Overview

   Initialize all the classes and make this program running.

2. Process Specifications

```
    public static void main(String[] args)
           /*
            * Requires: Nothing.
            * Modifies: Nothing.
```

```
        * Effects: Initialize all the classes and make this program running.
        */
```
3. Indicated Object
4. Abstract Function
5. Invariance

十一、 Taxi

1. Overview
   Simulate the taxi.
2. Process Specifications

```
public Taxi(int id)
        /*
         * Requires: Taxi id.
         * Modifies: Nothing.
         * Effects: Initialize a taxi.
         */
public void setPassenger(Passenger passenger)
        /*
         * Requires: A passenger.
         * Modifies: this.passenger and credit.
         * Effects: Allocate a passenger to this taxi.
         */
private void runTaxi(int di)
        /*
         * Requires: Nothing.
         * Modifies: Nothing.
         * Effects: run the taxi.
         */
public int getID()
        /*
         * Requires: Nothing.
         * Modifies: Nothing.
         * Effects: Return the taxi'ID.
         */
public int getCredit()
        /*
         * Requires: Nothing.
         * Modifies: Nothing.
         * Effects: Return the taxi' credit.
         */
public int getState()
        /*
         * Requires: Nothing.
```

```
                * Modifies: Nothing.
                * Effects: Return the taxi' state.
                */
    public int getNow_x()
              /*
                * Requires: Nothing.
                * Modifies: Nothing.
                * Effects: Return the taxi' x now.
                */
    public int getNow_y()
              /*
                * Requires: Nothing.
                * Modifies: Nothing.
                * Effects: Return the taxi' y now.
                */
    public int getTime()
              /*
                * Requires: Nothing.
                * Modifies: Nothing.
                * Effects: Return the time.
                */
    public abstract void fuckrun()
              /*
                * Requires: Nothing.
                * Modifies: Nothing.
                * Effects: Run the taxi.
                */
```

3. Indicated Object

```
private int now_x
private int now_y
private int state
private int ID
private int credit
private Passenger passenger
private int Direction
private int exDirection
private int time
private int rest_count
```

4. Abstract Function

   AF(c) = (now_x, now_y, state, ID, credit, passenger, Direction, exDirection, time, rest_count), where now_x = c. now_x, now_y = c. now_y , state = c. state, ID = c. ID, credit = c. credit, passenger = c. passenger, Direction = c. Direction, exDirection = c. exDirection, time = c. time, rest_count = c. rest_count.

5. Invariance

0<=now_x<80 && 0<=now_y<80 && 4<= state<=7 && 0<= ID<100 && credit >= 0 && -1 <= Direction<=3 && -1<= exDirection<=3 && time >= 0 && 0<=rest_count<=200

十二、 Traffic_light

1. Overview

   Simulate the traffic light.

2. Process Specifications

```
        public Traffic_light(boolean has)
                /*
                  * Requires: A boolean.
                  * Modifies: Nothing.
                  * Effects: Initialize a traffic light.
                  */
          public boolean isHas()
                /*
                  * Requires: Nothing.
                  * Modifies: Nothing.
                  * Effects: Return the has.
                  */
        public int getL_r()
                /*
                  * Requires: Nothing.
                  * Modifies: Nothing.
                  * Effects: Return the l_r.
                  */
        public int getU_d()
                /*
                  * Requires: Nothing.
                  * Modifies: Nothing.
                  * Effects: Return the u_d.
                  */
            public void change()
                /*
                  * Requires: Nothing.
                  * Modifies: Nothing.
                  * Effects: Change the light status..
                  */
```

3. Indicated Object

```
        private boolean has
        private int l_r
        private int u_d;
```

4. Abstract Function

   AF(c) = (has, l_r, u_d), where has = c. has, l_r = c. l_r, u_d = c. u_d.

5. Invariance

|c.l_r| == 1 && |c.u_d| == 1

十三、 Types

1. Overview

   Define all the base types in this project.
2. Process Specifications
3. Indicated Object

```
public static final int UP = 0
public static final int DOWN = 1
public static final int LEFT = 2
public static final int RIGHT = 3
public static final int size = 80
public static final int WAIT = 4
public static final int GETPSG = 5
public static final int SERVING = 6
public static final int REST = 7
public static final long BASE_TIME = 100
public static final long CALL_TIME = 3000
public static final int WAIT_TIME = 200
public static final int REST_TIME = 10
```

4. Abstract Function
5. Invariance

十四、 Normal_Taxi

1. Overview

   Simulate the normal taxi.
2. Process Specifications

```
public Normal_Taxi(int id)
        /*
          * Requires: Taxi id.
          * Modifies: Nothing.
          * Effects: Initialize a taxi.
     */
public void fuckrun()
        /*
          * Requires: Nothing.
          * Modifies: this.
          * Effects: run the taxi.
     */
```

   The others please find it in Taxi.
3. Indicated Object

   See Taxi.
4. Abstract Function

   See Taxi.

5. Invariance
   See Taxi.

十五、 Tracking_Taxi

1. Overview
   Simulate the trackable taxi.
2. Process Specifications

```
public Normal_Taxi(int id)
        /*
         * Requires: Taxi id.
         * Modifies: Nothing.
         * Effects: Initialize a taxi.
         */
public void fuckrun()
        /*
         * Requires: Nothing.
         * Modifies: this.
         * Effects: run the taxi.
         */
public Iterator<Index> iterator(int i)
        /*
         * Requires: Nothing.
         * Modifies: Nothing.
         * Effects: Return the iterator of ith path..
         */
 public int getServe_times()
        /*
         * Requires: Nothing.
         * Modifies: Nothing.
         * Effects: Return the serve times..
        */
```

   The others please find it in Taxi.
3. Indicated Object

```
private int serve_times;
private Vector<Index> [] serve_path
```

   The others please find it in Taxi.
4. Abstract Function
   AF(c) = (now_x, now_y, state, ID, credit, passenger, Direction, exDirection, time, rest_count, serve_times, serve_path), where now_x = c. now_x, now_y = c. now_y , state = c. state, ID = c. ID, credit = c. credit, passenger = c. passenger, Direction = c. Direction, exDirection = c. exDirection, time = c. time, rest_count = c. rest_count, serve_times = c. serve_times, serve_path = c. serve_path.
5. Invariance
   Taxi.invariance && serve_path != null

十六、 Proof of LSP Principle

The taxi abstract summarizes the two types of taxi’s moving module.

The two subclasses of the Taxi override this abstract method (public abstract void fuckrun()). The Normal_Taxi does nothing except that and the other methods added in Tracking_Taxi does not modify the superclass’ members. We can see those two subclasses do not break the constraint in superclass.

Other classes don’t extend any class.

So the LSP Principle is satisfied.