Freie Universität Berlin

Prof. Dr. Claudia Müller-Birn, Barry Linnert

Objektorientierte Programmierung, SoSe 17

Übung 09

TutorIn: Thierry Meurers
Tutorium 10

Stefaan Hessmann, Jaap Pedersen, Mark Niehues

27. Juni 2017

---

1 Aufgabe 1

Listing 1: House Class

```
0  package u9.src;
1
2  /**
3   * Class represents a house that can hold a specific amount of people and
4   * will be demolished when the lifespan is exceeded and nobody is living
5   * int the building anymore
6   */
7  public class House {
8      // Attributes
9      private int buildYear;
10     private int lifeSpan;
11     private int freeRooms, occupiedRooms;
12
13     /**
14      * Constructor for the House Class with individual lifespan and number of rooms     *
15      *
16      * @param buildYear  Year in which the house was build
17      * @param totalRooms Number of rooms in the house
18      * @param lifeSpan   Lifespan of the building after which it gets destroyed
19      */
20     public House(int buildYear, int totalRooms, int lifeSpan) {
21         this.buildYear = buildYear;
22         this.lifeSpan = lifeSpan;
23
24         // All rooms empty
25         freeRooms = totalRooms;
26         occupiedRooms = 0;
27     }
28
29     /**
30      * Trying to move into the house
31      *
32      * @return True if success, False if house is full
33      */
34     public boolean moveInHouse() {
35         if (freeRooms > 0) {
36             // Free room available
```

```
            freeRooms--;
            occupiedRooms++;

            return true;
        } else {
            // House full
            return false;
        }
    }

    /**
     * Trying to move out of the house
     *
     * @return True if successful, False if house is empty
     */
    public boolean moveOutHouse() {
        if (occupiedRooms > 0) {
            freeRooms++;
            occupiedRooms--;

            return true;
        } else {
            // House empty
            return false;
        }
    }

    /**
     * @return True if house is empty, False if not
     */
    public boolean isEmpty() {
        if (occupiedRooms == 0) {
            return true;
        } else {
            return false;
        }
    }

    /**
     * @param currentYear The current year to compute how old the building is
     * @return True if house is older than its lifespan
     */
    public boolean isOverdue(int currentYear) {
        if ((currentYear - buildYear) >= lifeSpan) {
            return true;
        } else {
            return false;
        }
    }

    /**
     * @return Number of occupied rooms
     */
    public int getOccupiedRooms() {
        return occupiedRooms;
    }
    /**
     * @return Number of free rooms
     */
    public int getFreeRooms(){
        return freeRooms;
    }
}
```

Listing 2: The Street Class organizes the houses

```
package u9.src;
```

```java
/**
 * The Street class holds houses and deals with incoming and leaving
 * neighbours. It also demolishes houses if they're overdue.
 */
public class Street {
    private House[] houses;

    /**
     * Initialize empty Street with space for MaxHouse houses
     *
     * @param maxHouses Maximum number houses
     */
    public Street(int maxHouses) {
        houses = new House[maxHouses];
    }

    /**
     * Several people try to move out. This method overloads
     * the moveOut() method.
     *
     * @param numberOfPeople Number of people that try to move out
     */
    public void moveOut(int numberOfPeople) {
        for (int i = 0; i < numberOfPeople; i++) {
            if (!moveOut()) {
                // Street is empty
                System.out.println("\nStreet is empty.");
                break;
            }
        }
    }

    /**
     * One Person trying to move out.
     *
     * @return Success or not
     */
    public boolean moveOut() {
        for (int i = 0; i < houses.length; i++) {
            if (houses[i] != null) {
                if (houses[i].moveOutHouse())
                    // Moving out was successful
                    return true;
            }
        }
        // Moving out wasn't successful since the street is empty
        return false;
    }

    /**
     * Several people try to move in. If there a less flats than people
     * that try to move in, new buildings are build.
     *
     * @param numberOfPeople Number of people that try to move in
     * @param currentYear    The current year
     * @param flat_per_house Number of flats every house holds
     * @param lifespan       Lifespan of the house
     * @return False if street is full during the move, True else
     */
    public boolean moveIn(int numberOfPeople, int currentYear, int flat_per_house, int
    lifespan) {
        for (int i = 0; i < numberOfPeople; i++) {
            if (!moveIn()) {
                // Try to build new House
                if (!buildNewHouse(currentYear, flat_per_house, lifespan)) {
                    // Failed, street full
                    return false;
                }
```

```
                i--; // Decrease by one because nothing was done this time
            }
        }
        // Everyone could move in
        return true;
    }

    /**
     * Someone wants to move in
     *
     * @return Successful or not
     */
    public boolean moveIn() {
        // Try to find free Slot in existing House
        for (int i = 0; i < houses.length; i++) {
            if (houses[i] != null && houses[i].moveInHouse()) {
                return true;
            }
        }
        return false;
    }

    /**
     * Looking for a free slot and building a new house there
     *
     * @param buildYear  The current year
     * @param totalRooms Number of flats the house holds
     * @param lifeSpan   Lifespan of the house
     * @return True if it was successful, False if the street is already full
     */
    public boolean buildNewHouse(int buildYear, int totalRooms, int lifeSpan) {
        // Look for free slot
        for (int i = 0; i < houses.length; i++) {
            if (houses[i] == null) {
                houses[i] = new House(buildYear, totalRooms, lifeSpan);

                return true;
            }
        }
        // Street full
        return false;
    }

    /**
     * Looks for empty buildings that are overdue and deletes them. Also, if there are no
     * free flats anymore, a new house is build.
     */
    public void cleanStreet(int currentYear, int buildYear, int totalRooms, int lifeSpan) {
        for (int i = 0; i < houses.length; i++) {
            // Houses that are empty AND overdue should be removed
            if (houses[i] != null && houses[i].isEmpty() && houses[i].isOverdue(currentYear
    )) {
                houses[i] = null;
                System.out.println("\nHouse with housenumber: " + i + " was demolished");
            }
        }

        if (isFull()){
            // No free flats -> Build a new house if possible
            buildNewHouse(buildYear, totalRooms, lifeSpan);
        }
    }

    /**
     * @return The number of occupied flats in the street
     */
    public int NumberOccupiedFlats() {
        int sum = 0;
```

```
136         for (int i = 0; i < houses.length; i++) {
137             if (houses[i] != null) {
138                 sum += houses[i].getOccupiedRooms();
139             }
140         }
141         return sum;
142     }
143 
144     /**
145      * @return Number of overall free Flats in the street
146      */
147     public int NumberFreeFlats(){
148         int sum = 0;
149         for (int i = 0; i < houses.length; i++) {
150             if (houses[i] != null) {
151                 sum += houses[i].getFreeRooms();
152             }
153         }
154         return sum;
155     }
156 
157     /**
158      * @return True street holds no empty flat
159      */
160     public boolean isFull(){
161         return NumberFreeFlats() == 0;
162     }
163 
164     /**
165      * Creates print of the street
166      *
167      * @param currentYear
168      */
169     public void printStreet(int currentYear) {
170         System.out.println("");
171         for (House h : houses) {
172             if (h != null) {
173                 System.out.print(h.getOccupiedRooms());
174             } else {
175                 System.out.print("X");
176             }
177             System.out.print("   ");
178         }
179         System.out.println("");
180         for (int i = 0; i < houses.length; i++) {
181             System.out.print("----");
182         }
183         System.out.println("");
184 
185         for (int i = 0; i < houses.length; i++) {
186             System.out.print(" - -");
187         }
188         System.out.println("");
189 
190         for (int i = 0; i < houses.length; i++) {
191             System.out.print("----");
192         }
193     }
194 }
```

Listing 3: The Main routine that runs an example simulation with output

```
0 package u9.src;
1 
2 import java.lang.Math;
3 
4 /**
5  * Tests the house simulation
```

```
 */
public class Main {
    public static void main(String[] args) {
        //simulate 15 years of street
        int year = 2017;
        int properties = 10;
        int flat_per_house = 5;
        int lifespan = 2;
        int simulation_time = 6;

        // Print simulation setup
        System.out.println("\n=======Simulation parameters===========\n");
        System.out.println("Properties: " + properties);
        System.out.println("Flats per house: " + flat_per_house);
        System.out.println("Lifespan of the houses:" + lifespan);
        System.out.println("Years simulated:" + simulation_time);

        // Initial setup with one house
        Street hessmann_weg = new Street(properties);
        hessmann_weg.buildNewHouse(year, flat_per_house, lifespan);

        // Print initial Setup
        System.out.println("\n===========Initial Setup==============\n");
        hessmann_weg.printStreet(year);

        System.out.println("\n=========Start of Simulation===========\n");
        for (int i = 0; i < simulation_time; i++, year++) {
            // Create random moves in and out
            int n_moveIn = (int) (Math.random() * 10);
            hessmann_weg.moveIn(n_moveIn, year, flat_per_house, lifespan);
            int n_families = hessmann_weg.NumberOccupiedFlats();
            int n_moveOut = (int) (Math.random() * n_families);

            System.out.println("\nYear:\t" + year + "\tFamilies moving in:\t" + n_moveIn +
    "\tmoving out:\t" + n_moveOut);

            System.out.println("\n=======Street after moving in===========\n");
            hessmann_weg.printStreet(year);
            hessmann_weg.moveOut(n_moveOut);

            System.out.println("\n=======Street after moving out==========\n");
            // Remove old buildings and build a new one if there are no empty flats anymore
            hessmann_weg.cleanStreet(year, year, flat_per_house, lifespan);
            hessmann_weg.printStreet(year);


        }


    }
}
```