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

28. 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
```

```
37              freeRooms--;
38              occupiedRooms++;
39
40              return true;
41          } else {
42              // House full
43              return false;
44          }
45      }
46
47      /**
48       * Trying to move out of the house
49       *
50       * @return True if successful, False if house is empty
51       */
52      public boolean moveOutHouse() {
53          if (occupiedRooms > 0) {
54              freeRooms++;
55              occupiedRooms--;
56
57              return true;
58          } else {
59              // House empty
60              return false;
61          }
62      }
63
64      /**
65       * @return True if house is empty, False if not
66       */
67      public boolean isEmpty() {
68          return occupiedRooms == 0;
69      }
70
71      /**
72       * @param currentYear The current year to compute how old the building is
73       * @return True if house is older than its lifespan
74       */
75      public boolean isOverdue(int currentYear) {
76          return (currentYear - buildYear) >= lifeSpan;
77      }
78
79      /**
80       * @return Number of occupied rooms
81       */
82      public int getOccupiedRooms() {
83          return occupiedRooms;
84      }
85      /**
86       * @return Number of free rooms
87       */
88      public int getFreeRooms(){
89          return freeRooms;
90      }
91  }
```

Listing 2: The Street Class organizes the houses

```
0  package u9.src;
1
2  /**
3   * The Street class holds houses and deals with incoming and leaving
4   * neighbours. It also demolishes houses if they're overdue.
5   */
6  public class Street {
7      private House[] houses;
8
9      /**
```

```
     * 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;
        for (int i = 0; i < houses.length; i++) {
            if (houses[i] != null) {
                sum += houses[i].getOccupiedRooms();
            }
        }
        return sum;
    }
```

```
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     }
157     /**
158      * @return True street holds no empty flat
159      */
160     public boolean isFull(){
161         return NumberFreeFlats() == 0;
162     }
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("");
185         for (int i = 0; i < houses.length; i++) {
186             System.out.print(" - -");
187         }
188         System.out.println("");
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;
2  import java.lang.Math;
4  /**
5   * Tests the house simulation
6   */
7  public class Main {
8      public static void main(String[] args) {
9          //simulate 15 years of street
10         int year = 2017;
11         int properties = 10;
12         int flat_per_house = 5;
13         int lifespan = 2;
```

```
14          int simulation_time = 6;
15
16          // Print simulation setup
17          System.out.println("\n=======Simulation parameters============\n");
18          System.out.println("Properties: " + properties);
19          System.out.println("Flats per house: " + flat_per_house);
20          System.out.println("Lifespan of the houses:" + lifespan);
21          System.out.println("Years simulated:" + simulation_time);
22
23          // Initial setup with one house
24          Street hessmann_weg = new Street(properties);
25          hessmann_weg.buildNewHouse(year, flat_per_house, lifespan);
26
27          // Print initial Setup
28          System.out.println("\n============Initial Setup==============\n");
29          hessmann_weg.printStreet(year);
30
31          System.out.println("\n=========Start of Simulation============\n");
32          for (int i = 0; i < simulation_time; i++, year++) {
33              // Create random moves in and out
34              int n_moveIn = (int) (Math.random() * 10);
35              hessmann_weg.moveIn(n_moveIn, year, flat_per_house, lifespan);
36              int n_families = hessmann_weg.NumberOccupiedFlats();
37              int n_moveOut = (int) (Math.random() * n_families);
38
39              System.out.println("\nYear:\t" + year + "\tFamilies moving in:\t" + n_moveIn +
    "\tmoving out:\t" + n_moveOut);
40
41              System.out.println("\n=======Street after moving in===========\n");
42              hessmann_weg.printStreet(year);
43              hessmann_weg.moveOut(n_moveOut);
44
45              System.out.println("\n=======Street after moving out==========\n");
46              // Remove old buildings and build a new one if there are no empty flats anymore
47              hessmann_weg.cleanStreet(year, year, flat_per_house, lifespan);
48              hessmann_weg.printStreet(year);
49
50
51          }
52
53
54      }
55  }
```

2 Aufgabe 2

2.1 Beobachtungen

- a) ist immer 1.0, da der erste Summand 1.0 ist und die darauf folgenden auf 0 gerundet werden durch die division zweier integers.
- die Summe für b) ist erst 1.0 und ab $n = 10^6$ erfolgt ein Abbruch, da der Wertebereich für Integers in Java zyklisch implementiert ist, sodass irgendwann $i * i = 0$ ist und die Division durch 0 für Integer nicht definiert ist.
- Im Gegensatz dazu ist die Division durch 0 für double definiert als `infinity`, sodass ab $n = 10^6$ im Falle d) und f) das Ergebnis `infinity` ist.
- Die anderen Varianten funktionieren, aber weisen zum Teil leicht unterschiedliche Abweichungen auf.

2.2 Ergebnisse der verschiedenen Durchläufe

Variante	Endwert	Abweichung vom Grenzwert
a)	1.0	0.6449340668482264
b)	1.0	0.6449340668482264
c)	1.6349839001848923	0.009950166663334148
d)	1.6349839001848923	0.009950166663334148
e)	1.6349839001848923	0.009950166663334148
f)	1.6349839001848923	0.009950166663334148
g)	1.6349839032409363	0.009950163607290063
h)	1.6349839001848923	0.009950166663334148

Tabelle 1: n = 100

Variante	Endwert	Abweichung vom Grenzwert
a)	1.0	0.6449340668482264
b)	1.0	0.6449340668482264
c)	1.6448340718480652	9.999500016122376E-5
d)	1.6448340718480652	9.999500016122376E-5
e)	1.6448340718480652	9.999500016122376E-5
f)	1.6448340718480652	9.999500016122376E-5
g)	1.644834074928367	9.999191985943234E-5
h)	1.6448340718480652	9.999500016122376E-5

Tabelle 2: n = 10000

Variante	Endwert	Abweichung vom Grenzwert
a)	1.0	0.6449340668482264
b)	Abgebrochen	Abgebrochen
c)	1.64493306684877	9.999994563525405E-7
d)	Infinity	-Infinity
e)	1.64493306684877	9.999994563525405E-7
f)	Infinity	-Infinity
g)	1.6449330699290232	9.969192031888952E-7
h)	1.64493306684877	9.999994563525405E-7

Tabelle 3: n = 1000000

Variante	Endwert	Abweichung vom Grenzwert
a)	1.0	0.6449340668482264
b)	Abgebrochen	Abgebrochen
c)	1.644934057834575	9.013651380840315E-9
d)	Infinity	-Infinity
e)	1.644934057834575	9.013651380840315E-9
f)	Infinity	-Infinity
g)	1.6449340609148324	5.933393998347469E-9
h)	1.644934057834575	9.013651380840315E-9

Tabelle 4: n = 100000000