

# Lab 3

Object-Oriented Design, IV1350

Evan Saboo  
saboo@kth.se  
2015-05-06

# Innehåll

1	Introduktion	3
2	Metod	4
3	Resultat	5

# 1 Introduktion

Syftet med den tredje labb uppgiften var att skapa ett java program från klass diagrammet som vi utförde i andra labben. Syftet var också att lära sig att skriva Unit tester som är kopplad med programmet man utförde.

## 2 Metod

Jag arbetade med Emil Nordin på programmeringen av inspektionsprogrammet för att vi skulle bli klara snabbare och utföra koden på ett effektivt sätt. Eftersom vi hade två nästan likadana designer kunde vi gå igenom dem och implementera de bästa metoderna i programmet. Med hjälp klass diagrammen fick vi en stor bild på hur programmet skulle se ut och hur metoderna skulle fungera. Vi fokuserade mest på inspektionsprogrammet eftersom vi ville att alla metoder och klasser skulle funka utan att programmet kraschade på grund av t.ex. fel inmatning. I föreläsningar och handledningen fick vi veta att några klasser, som "Garage" och "Payment" skulle inte användas eftersom vi alla metoder skulle användas i "View" klassen och metoderna skulle gå igenom "Controller" klassen innan den kunde användas i View.

### 3 Resultat

I resultat finns det bilder på hela programmet med Unit testet.

```
1 package startup;
2
3 import controller.Controller;
4 /**
5  * Startup of the entire program.
6  *
7  */
8 public class StartUp {
9
10     /**
11      * Starts the program.
12      * @param args This takes nothing.
13      */
14     public static void main(String[] args) {
15         new Controller().controller();
16     }
17 }
18
19
```

Figur 1. En bild på klassen StartUp.

```

1 package view;
2
3 import java.util.Scanner;
4 import controller.Controller;
5
6 /**
7  * <code>View</code> handles all the inputs and calls on the controller to use them.
8  * Also prints to user.
9  */
10 public class View {
11
12     public String licenseNumber;
13     public String licenseNumberValid;
14     public boolean licensevalidation;
15     private boolean isReady;
16     private String ready;
17     private String doorCommand;
18     private boolean doorCommandCheck;
19     private String doorState;
20     private boolean correctinput = false;
21     private boolean continueprogram = false;
22     public int queuenumber = 0;
23     private String payment;
24     public int payCash;
25     public int totalChange;
26     private boolean updateCashRegister;
27     private boolean creditCardValidation;
28     public String creditCardInfo;
29     private String creditCardPin;
30     public String [] inspectionPart = new String [7];
31     public String [] inspect = new String [7];
32     public boolean [] inspectionResults = new boolean[inspect.length];
33     Scanner s = new Scanner(System.in);
34
35     /**
36      * Runs entire program.
37      * Prints out to user and takes inputs to then sent inputs to controller.
38      */
39     public void view()
40     {
41
42         while(continueprogram == false)
43         {
44
45             while(correctinput == false)
46             {
47                 System.out.println("Start new inspection? (y/n)");
48
49                 ready = s.nextLine();
50
51                 if(ready.equals("y") || ready.equals("n"))
52                     correctinput = true;
53
54                 else
55                 {
56                     correctinput = false;
57                     System.out.println("Invalid input");
58                 }
59             }
60
61             if(ready.equals("y"))
62             {
63                 isReady = true;
64                 continueprogram = true;
65             }
66             else
67             {
68                 isReady = false;
69                 continueprogram = false;
70                 correctinput = false;
71             }
72
73             queuenumber = new Controller().startNewInspection(isReady);
74
75         }
76     }
77 }

```

```
75     System.out.println("Current number is " + queuenumber);
76
77     continueprogram = false;
78
79     while(continueprogram == false)
80     {
81         correctinput = false;
82         while(correctinput == false)
83         {
84             System.out.println ("Press 'o' to open door");
85             doorCommand = s.nextLine();
86             if(doorCommand.equals("o") || doorCommand.equals("c"))
87                 correctinput = true;
88             else
89             {
90                 correctinput = false;
91                 System.out.println("Invalid input");
92             }
93         }
94         doorCommandCheck = new Controller().isOpen(doorCommand);
95
96         if(doorCommandCheck == true)
97         {
98             doorState = "open";
99             continueprogram = true;
100         }
101         else
102         {
103             doorState = "closed";
104             continueprogram = false;
105             System.out.println("Open the door to start inspection");
106         }
107     }
108
109     System.out.println ("The door is " + doorState);
110
111 }
112
113 continueprogram = false;
114 while(continueprogram == false)
115 {
116     correctinput = false;
117     while(correctinput == false)
```

```

118         {
119             System.out.println ("Press 'c' to close door");
120             doorCommand = s.nextLine();
121             if (doorCommand.equals("o") || doorCommand.equals("c"))
122                 correctinput = true;
123             else
124             {
125                 correctinput = false;
126                 System.out.println("Invalid input");
127             }
128         }
129         doorCommandCheck = new Controller().isOpen(doorCommand);
130
131         if (doorCommandCheck == true)
132         {
133             doorState = "open";
134             continueprogram = false;
135             System.out.println("Close the door to start inspection");
136         }
137         else
138         {
139             doorState = "closed";
140             continueprogram = true;
141         }
142
143         System.out.println ("The door is " + doorState);
144     }
145
146     continueprogram = false;
147     while(continueprogram == false)
148     {
149         System.out.println ("Enter vehicle license number:");
150         licenseNumber = s.nextLine();
151         licensevalidation = new Controller().enterVehicleInfo(licenseNumber);
152         if (licensevalidation == true)
153         {
154             licenseNumberValid = "valid";
155             continueprogram = true;
156         }
157         else
158         {
159             licenseNumberValid = "invalid";
160             continueprogram = false;
161         }
162     }
163
164     System.out.println ("The licenseNumber is " + licenseNumberValid);
165 }
166
167 continueprogram = false;
168 while(continueprogram == false)
169 {
170     System.out.println("Your inspection payment is " + new Controller().cashToPay + " dollars");
171
172     System.out.println ("Do you want pay with cash or credit? (cash/credit)");
173     payment = s.nextLine();
174
175     if (payment.equals("cash") || payment.equals("Cash"))
176     {
177         while(continueprogram == false)
178         {
179             System.out.println("How much do you want to pay? (in dollar bills)");
180
181             continueprogram = false;
182             while(continueprogram == false)
183             {
184                 boolean isNumber;
185                 do {
186                     if (s.hasNextInt())
187                     {
188                         payCash = s.nextInt();
189                         isNumber = true;
190
191                         updateCashRegister = new Controller().canCashRegisterPayChangeCheck(payCash);
192                         totalChange = new Controller().payWithCash(payCash);
193
194                         if (updateCashRegister == true)
195                         {
196                             continueprogram = true;
197                         }
198                     }
199                 }
200             }

```



```
201     }
202     else
203     {
204         System.out.println("Our cash register doesn't have enough cash for change, please pay with a
205         continueprogram = false;
206     }
207 }
208
209 else
210 {
211     System.out.println("Your payment is invalid or our cash register doesn't have enough cash for cha
212     isNumber = false;
213     s.next();
214 }
215
216 }while(!(isNumber));
217 }
218 s.nextLine();
219
220 if (totalChange >= 0)
221 {
222     System.out.println("Payemt Complete\n"
223     + "");
224     new Controller().cashReceipt(licenseNumber, new Controller().cashToPay, payCash, totalChange);
225     continueprogram = true;
226     System.out.println("-----");
227     System.out.println("Total amount cash in CashRegister is: " + new Controller().cashLeftInRegister);
228 }
229
230
231
232
233 else
234 {
235     System.out.println("You have not paid enough money, please try again");
236     continueprogram = false;
237 }
238 }
239
240 }
```

## Figur 2. Bilder på klassen View.

```

241
242     else if(payment.equals("credit") || payment.equals("Credit"))
243     {
244         while(continueprogram == false)
245         {
246             System.out.println("Enter your credit card info:");
247
248
249             correctinput = false;
250             while (correctinput == false)
251             {
252                 creditCardInfo = s.nextLine();
253
254                 if(creditCardInfo.length() == 16)
255                     correctinput = true;
256                 else
257                     System.out.println("Your credit card info is invalid, try again!");
258             }
259
260             System.out.println("Enter your pin code:");
261
262             correctinput = false;
263             while (correctinput == false)
264             {
265                 creditCardPin = s.nextLine();
266
267                 if(creditCardPin.length() == 4)
268                     correctinput = true;
269                 else
270                     System.out.println("Your credit card pin is invalid, try again!");
271             }
272
273             creditCardValidation = new Controller().sendPaymentAuthorizationRequest(creditCardInfo, creditCardPin);
274
275             if(creditCardValidation == true)
276             {
277                 payment = "Complete";
278                 continueprogram = true;
279                 new Controller().creditReceipt(creditCardInfo, licenseNumber, new Controller().cashToPay);
280             }
281
282             else
283             {
284                 payment = "invalid";
285                 correctinput = false;
286                 continueprogram = false;
287             }
288             System.out.println("Payment is " + payment
289                               + "\n" );
290         }
291     }
292     else
293     {
294         System.out.println("Wrong input, try again!");
295     }
296     System.out.println("Inspection will start now..." + "\n");
297
298     new Controller().parts(inspectionPart);
299
300     for(int i = 0; i < 7; i++)
301     {
302         System.out.println("Inspect:" + inspectionPart[i] + " (pass or fail)");
303         inspect[i] = s.nextLine();
304
305         continueprogram = false;
306         while(continueprogram == false)
307         {
308             if(inspect[i].equals("fail") || inspect[i].equals("Fail") || inspect[i].equals("pass") || inspect[i].equals("Pass"))
309                 continueprogram = true;
310
311             else
312             {
313                 System.out.println("Wrong input, try again!");
314                 continueprogram = false;
315                 inspect[i] = s.nextLine();
316             }
317         }
318     }
319
320     inspectionResults = new Controller().getInspections(inspect);
321     new Controller().printout(inspectionResults, inspectionPart);
322     System.out.println("Thank you for using Emils & Evan's Inspection agency!");
323 }

```

```

1 package controller;
2 import view.View;[]
11 /**
12  * The <code>Controller</code> class executes the requests from the <code>View</code> class.
13  * Calls to the model pass through the <code>Controller</code>
14  */
15
16 public class Controller {
17
18     public int cashToPay = new CashRegister().cashToPay;
19     public int cashLeftInRegister = new CashRegister().totalAmountCash + cashToPay;
20
21     /**
22      * <code>Controller</code> is used by the <code>StartUp</code> class and runs the <code>view</code> method.
23      */
24     public void controller()
25     {
26         new View().view();
27     }
28
29     /**
30      * <code>startNewInspection</code> gets the next queue number from the <code>displayNextNumber</code> method.
31      * @param Ready makes next number show if true.
32      * @return Returns the queue number.
33      */
34     public int startNewInspection(boolean ready)
35     {
36         int nextnumber;
37         nextnumber = new Display().displayNextNumber(ready);
38         return nextnumber;
39     }
40
41     /**
42      * @param command Door opening command entered by user.
43      * @return Returns true if door is open false if not.
44      */
45     public boolean isOpen(String command)
46     {
47         boolean checkOpen;
48         checkOpen = new GarageDoor().isOpen(command);
49         return checkOpen;
50     }
51
52     /**
53      * Gives result of the license number validation
54      * @param licensenumber The entered vehicle license number.
55      * @return Returns true if the license number is valid false if invalid.
56      */
57     public boolean enterVehicleInfo(String licensenumber)
58     {
59         if(new DbHandler().giveLicenseNumber(licensenumber) == true)
60             return true;
61         else
62             return false;
63     }
64
65     /**
66      * Gets change for payment.
67      * @param pay The paid amount.
68      * @return Returns the change amount.
69      */
70     public int payWithCash(int pay)
71     {
72         return new CashRegister().payAndReturnChange(pay);
73     }
74
75     /**
76      * Checks if there is enough cash for change.
77      * @param cashPaying Cash given by customer.
78      * @return Returns true if enough cash for change.
79      */
80     public boolean canCashRegisterPayChangeCheck(int cashPaying)
81     {
82
83         return new CashRegister().canCashRegisterPayChangeCheck(cashPaying);
84     }

```

```

85
86  /**
87   * Sends payment validation request.
88   * @param creditCardInfo Credit card number
89   * @param creditCardPin Credit card pin code.
90   * @return Credit card validation.
91   */
92  public boolean sendPaymentAuthorizationRequest(String creditCardInfo, String creditCardPin)
93  {
94      return new PaymentAuthorizationSystem().sendPaymentAuthorizationRequest(creditCardInfo, creditCardPin);
95  }
96
97  /**
98   * Receipt info for cash payment.
99   * @param licenseNumber The license number.
100   * @param cashToPay The inspection cost.
101   * @param payCash The amount paid.
102   * @param totalChange The change.
103   */
104  public void cashReceipt (String licenseNumber, int cashToPay, int payCash, int totalChange)
105  {
106      new Receipt().cashReceipt(licenseNumber, cashToPay, payCash, totalChange);
107  }
108
109  /**
110   * Receipt info for credit card payment.
111   * @param creditCardInfo Credit card information.
112   * @param licenseNumber License number.
113   * @param cashToPay The inspection cost.
114   */
115  public void creditReceipt(String creditCardInfo, String licenseNumber, int cashToPay)
116  {
117      new Receipt().creditReceipt(creditCardInfo, licenseNumber, cashToPay);
118  }
119
120  /**
121   * Makes inspection results into true/false from pass/fail.
122   * @param inspect Inspection results by user.
123   * @return Inspection results boolean.
124   */
125  public boolean [] getInspections(String [] inspect)
126  {
127      boolean[] checkInspection;
128
129      checkInspection = new Inspections().getInspections(inspect);
130      return checkInspection;
131  }
132
133  /**
134   * Gives parts to view.
135   * @param inspectionpart Part being inspected.
136   */
137  public void parts(String [] inspectionpart)
138  {
139      new Inspections().parts(inspectionpart);
140  }
141
142  /**
143   * Gives information to make printout.
144   * @param inspectionresults Results from inspection.
145   * @param testnames Names of inspections.
146   */
147  public void printout(boolean[] inspectionresults, String[] testnames)
148  {
149      Printout.printout(inspectionresults, testnames);
150  }
151
152 }
153

```

Figur 3. Bilder på klassen Controller.

```
1 package dbhandler;
2
3 /**
4  * Class to validate license number
5  */
6 public class DbHandler {
7     private boolean [] valid = new boolean [6];
8     private boolean validate = false;
9
10    /**
11     *
12     * @param licenseNumber The entered vehicle license number.
13     * @return Returns license number validation, true if valid false if not.
14     */
15    public boolean giveLicenseNumber(String licenseNumber)
16    {
17
18        if (licenseNumber.length() == 6)
19        {
20
21            for(int i = 0; i < 3; i++)
22            {
23                if(Character.isLetter(licenseNumber.charAt(i)))
24                    if(Character.isDigit(licenseNumber.charAt(i+3)))
25                        valid [i] = true;
26
27                else
28                    valid [i] = false;
29            }
30            if(valid[0] == true && valid[1] == true && valid[2] == true)
31                validate = true;
32
33            return validate;
34        }
35        else
36            return false;
37    }
38 }
39
```

Figur 4. En bild på klassen DbHandler.

```
1 package model;
2
3 /**
4  * Handles cash payment.
5  */
6 public class CashRegister {
7
8     public int cashToPay = 100;
9     public int totalAmountCash = 2000;
10    public int cashLeftInRegister = 0;
11    private boolean booltotalAmountCash;
12
13 /**
14  * @param payCash    Amount of cash paid.
15  * @return           Returns change.
16  */
17    public int payAndReturnChange(int payCash)
18    {
19        return payCash - cashToPay;
20    }
21 /**
22  * Checks if cash register can afford to pay change.
23  * @param cashPaying    Amount paid.
24  * @return             Returns true if register can afford to pay change.
25  */
26    public boolean canCashRegisterPayChangeCheck(int cashPaying)
27    {
28
29        if (totalAmountCash >= cashPaying)
30        {
31
32            booltotalAmountCash = true;
33        }
34        else
35            booltotalAmountCash = false;
36
37        return booltotalAmountCash;
38    }
39 }
40
```

Figur 5. En bild på klassen CashRegister.

```
1 package model;
2
3 /**
4  * Handles queue number.
5  */
6 public class Display {
7
8     /**
9      *
10     * @param ready    User input if ready for inspection.
11     * @return         Returns queue number.
12     */
13     public int displayNextNumber(boolean ready)
14     {
15         int currentnumber = 1336;
16         if (ready == true)
17             currentnumber += 1;
18         return currentnumber;
19     }
20 }
21
```

Figur 6. En bild på klassen Display.



```
1 package model;
2 /**
3  * Handles opening and closing of garage door.
4  */
5 public class GarageDoor {
6
7     /**
8      * @param command    Open/close command by user.
9      * @return           Returns door state.
10     */
11     public boolean isOpen(String command)
12     {
13
14         if(command.equals("o"))
15         {
16             return true;
17         }
18         else
19         {
20             return false;
21         }
22     }
23 }
```

Figur 7. En bild på klassen GarageDoor.

Figur 8. En bild på klassen Inspections.



```
1  package model;
2
3  /**
4   * Handles inspection results and names.
5   */
6  public class Inspections {
7      public boolean [] boolInspections = new boolean[7];
8      public String [] inspectionpart = new String[7];
9
10     /**
11      * <code>getInspections</code> turns the inspection results from pass/fail to true
12      * @param inspections    Inspection results string
13      * @return                Inspection result as boolean
14      */
15     public boolean [] getInspections (String [] inspections)
16     {
17         for(int i = 0; i < inspections.length; i++)
18         {
19             if(inspections[i].equals("pass") || inspections[i].equals("Pass"))
20             {
21                 boolInspections[i] = true;
22             }
23             else
24                 boolInspections[i] = false;
25         }
26         return boolInspections;
27     }
28     /**
29      * <code>parts</code> contains a list of the parts that should be inspected.
30      */
31     public void parts(String [] inspectionpart)
32     {
33
34         inspectionpart[0] = "Body";
35         inspectionpart[1] = "Glass";
36         inspectionpart[2] = "Suspension";
37         inspectionpart[3] = "Lights & lenses";
38         inspectionpart[4] = "Tires";
39         inspectionpart[5] = "Engine";
40         inspectionpart[6] = "Brakes";
41     }
42 }
```

```

1 package model;
2
3 /**
4  * Validates the credit card information.
5  */
6 public class PaymentAuthorizationSystem {
7
8
9
10 package model;
11 import java.util.Random;
12
13 /**
14  * <code>Receipt</code> creates a receipt and prints it out.
15  */
16 public class Receipt {
17
18     private Random rand = new Random();
19     private int receiptNumber = rand.nextInt(100);
20     private String creditCardNumberMasked= "XXXX XXXX XXXX ";
21
22     /**
23      * Created a receipt for cash payment and prints it out.
24      * @param licenseNumber Vehicle license number.
25      * @param cashToPay Inspection cost.
26      * @param cashPaid Amount paid in cash.
27      * @param totalChange Amount of change.
28      */
29     public void cashReceipt (String licenseNumber, int cashToPay, int cashPaid, int totalChange)
30     {
31         System.out.println("-----");
32         System.out.println("Receipt number: " + receiptNumber);
33         System.out.println("License number: " + licenseNumber);
34         System.out.println("Inspection cost: " + cashToPay + " dollars");
35         System.out.println("Cash paid: " + cashPaid + " dollars");
36         System.out.println("Change: " + totalChange + " dollars");
37     }
38
39     /**
40      * Creates a receipt for credit card payment.
41      * @param creditCardInfo Credit card information.
42      * @param licenseNumber Vehicle license number.
43      * @param cashToPay Inspection cost.
44      */
45     public void creditReceipt(String creditCardInfo, String licenseNumber, int cashToPay )
46     {
47         for(int i = 12; i < 16; i++)
48         {
49             creditCardNumberMasked += creditCardInfo.charAt(i);
50         }
51         System.out.println("-----");
52         System.out.println("Receipt number: " + receiptNumber);
53         System.out.println("License number: " + licenseNumber);
54         System.out.println("Inspection cost: " + cashToPay + " dollars");
55         System.out.println("Paid with credit card: " + creditCardNumberMasked);
56     }
57
58     for(boolean b : array) if(!b) return false;
59     return true;
60 }
61
62 }

```

Figur 9. En bild på klassen PaymentAuthorizationSystem.

Figur 10. En bild på klassen Receipt.

```
1 package model;
2
3 /**
4  * Creates the printout and prints it.
5  */
6 public class Printout {
7
8     public static String [] inspectionResults = new String [7];
9
10 /**
11  * Creates printout with inspection names and results of them.
12  * @param boolinspectionresults    Inspection results.
13  * @param testnames                Names of inspections.
14  */
15     public static void printout(boolean[] boolinspectionresults, String[] testnames)
16     {
17
18         System.out.println("Emil and Evan's Inspection agency - Printout\n"
19                             + "-----");
20         for(int i = 0; i < new Inspections().inspectionpart.length; i++)
21         {
22             if(boolinspectionresults[i] == true)
23                 inspectionResults[i] = "pass";
24
25             else
26                 inspectionResults[i] = "fail";
27
28             System.out.println("Result of " + testnames[i] + " test " + " = " + inspectionResults[i] );
29         }
30     }
31 }
32
```

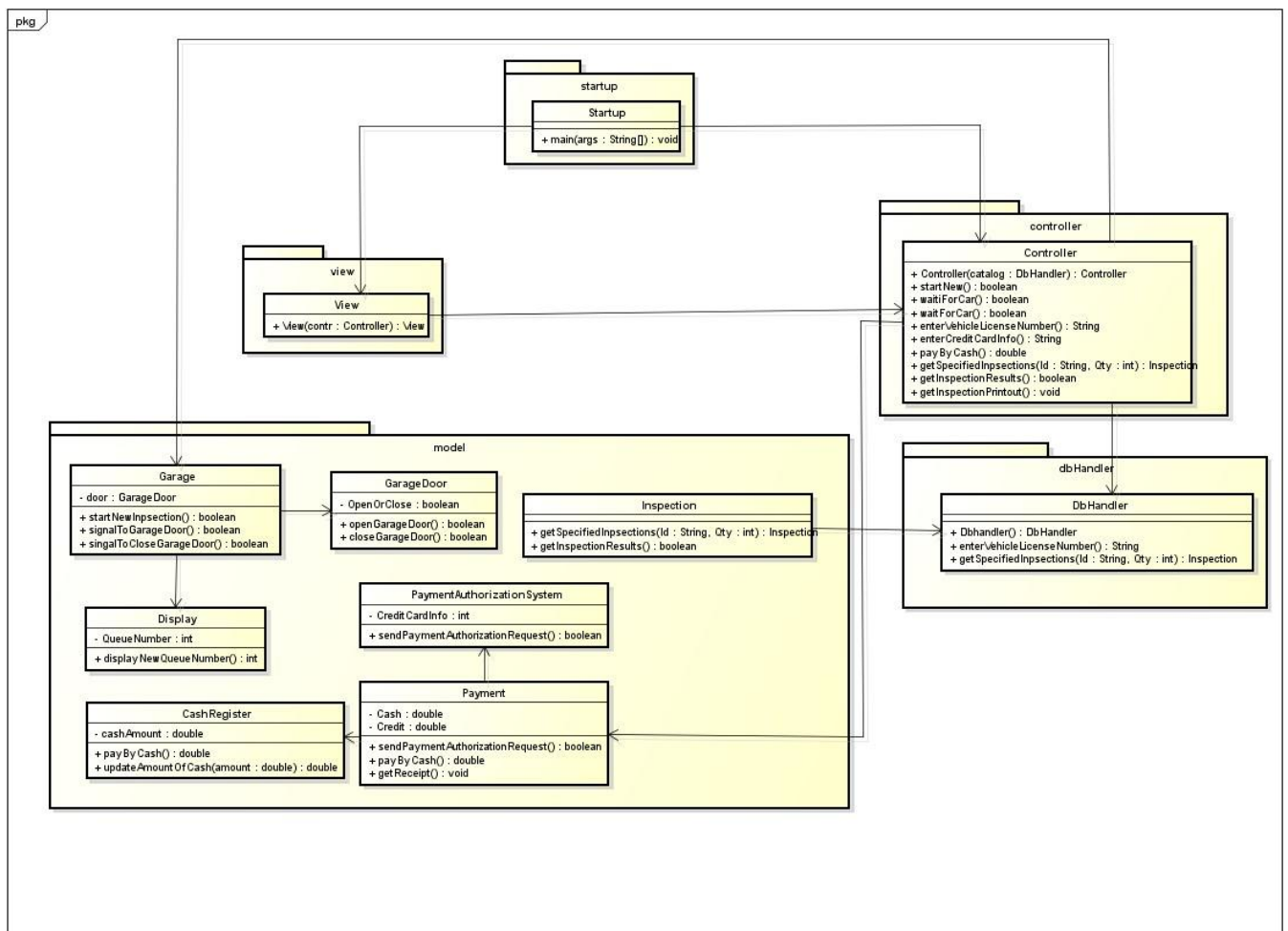
Figur 11. En bild på klassen Printout.

```

78 @Test
79 public void CashRegisterTest() {
80
81     for(int i = 0; i < 10000001; i++)
82     {
83         cashRegisterResults = cashRegister.payAndReturnChange(i);
84         assertEquals((i - cashToPay), cashRegisterResults);
85     }
86
87     for(int i = 0; i <= 2000; i++)
88     {
89         boolcashRegisterResults = cashRegister.canCashRegisterPayChangeCheck(i);
90         assertEquals(true, boolcashRegisterResults);
91     }
92
93     for(int i = 0; i > 2000; i++)
94     {
95         boolcashRegisterResults = cashRegister.canCashRegisterPayChangeCheck(i);
96         assertEquals(false, boolcashRegisterResults);
97     }
98 }
99
100 @Test
101 public void PaymentAuthorizationSystemTest(){
102     String testing = "1";
103     for(int i= 0; i <14; i++)
104     {
105         creditCardResults = paymentAuthorizationSystem.sendPaymentAuthorizationRequest(testing += "1", "7778");
106         assertEquals(false, creditCardResults);
107     }
108     creditCardResults = paymentAuthorizationSystem.sendPaymentAuthorizationRequest("1234123412341234", "7778");
109     assertEquals(true, creditCardResults);
110
111     creditCardResults = paymentAuthorizationSystem.sendPaymentAuthorizationRequest("15642", "7778");
112     assertEquals(false, creditCardResults);
113
114     creditCardResults = paymentAuthorizationSystem.sendPaymentAuthorizationRequest("1564233534630424", "778");
115     assertEquals(false, creditCardResults);
116
117     creditCardResults = paymentAuthorizationSystem.sendPaymentAuthorizationRequest("1564233534630", "778");
118     assertEquals(false, creditCardResults);
119
120     creditCardResults = paymentAuthorizationSystem.sendPaymentAuthorizationRequest("15642335a630", "7a78");
121     assertEquals(false, creditCardResults);
122
123     creditCardResults = paymentAuthorizationSystem.sendPaymentAuthorizationRequest("15642335a630", "778");
124     assertEquals(false, creditCardResults);
125
126     creditCardResults = paymentAuthorizationSystem.sendPaymentAuthorizationRequest("15642335346304a4", "1a11");
127     assertEquals(false, creditCardResults);
128
129     for(int i = 0; i < 10; i++)
130     {
131         allTrue[i] = true;
132     }
133     creditCardResults = paymentAuthorizationSystem.areAllTrue(allTrue);
134     assertEquals(true, creditCardResults);
135
136     for(int i = 0; i < 10; i++)
137     {
138         if(i < 4)
139             allTrue[i] = true;
140         else
141             allTrue[i] = false;
142     }
143     creditCardResults = paymentAuthorizationSystem.areAllTrue(allTrue);
144     assertEquals(false, creditCardResults);
145 }
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

Figur 12. Bilder på klassen ModelTest.



Figur 13. En bild på klass diagrammet som vi delvis använde.

