

Seminar 3

Object-Oriented Design, IV1350

Maher Jabbar, maherj@kth.se

2016-05-09

Contents

Introduction	3
Method	4
Result	5
Discussion	20

Introduction

Uppgiften för denna labb är att skriva ett program som implementerar allt i program specifikation som vi använde i lab1 & lab2. Programmet bedöms på om programmet är körbara, compilable och fick hög sammanhållning, låg koppling och en väl utformad offentligt gränssnitt.

Method

För detta labb Jag har använt Netbeans som min plattform. Jag använde mig också av seminar 1 och 2 och då fick jag hjälp från dem så skrev jag koden efter med hjälp av Netbeans. Att skriva enhetstester var svår faktisk i början eftersom den vart nått nytt för oss. Om metoden gjorde vad den var avsedd att göra alltså om metoden skickar tillbaka något, ett objekt inspektion till exempel vi bara jämförde kontroll returneras med en inspektion som är identisk med den som vi förväntas få från metoden. Metoden Open door eller Close door är void method då den tvungen att hämta tillståndet hos dörren och se att det är stängd eller öppen. Vi gjorde så att klasserna namn representerar det arbete de utför och inget annat för att vi ska hålla en hög sammanhållning i projektet för att det skulle vara lättförståeligt och lätt att följa. För att ha bra inkapsling så gjorde vi att användare inte skulle ha tillgång till metoder och klasser som användaren inte behövde göra direkta anrop till.

Result

figure 3.1 Testkör Av programmet

Page 6-7 View: säger till programmet nästa inspektion skall utföras. Programmet skriver ut aktuella könummer och öppnar dörrenx.

Page 8 View: säger till program för att stänga dörren.

Page 9-12 View: hämtar hämtar kostnaden för inspektion, skriver ut skrivaren en kvitto innehållande inspektion kostnader och delar att inspektera och granska retrieveres tillstånd för kreditkort.

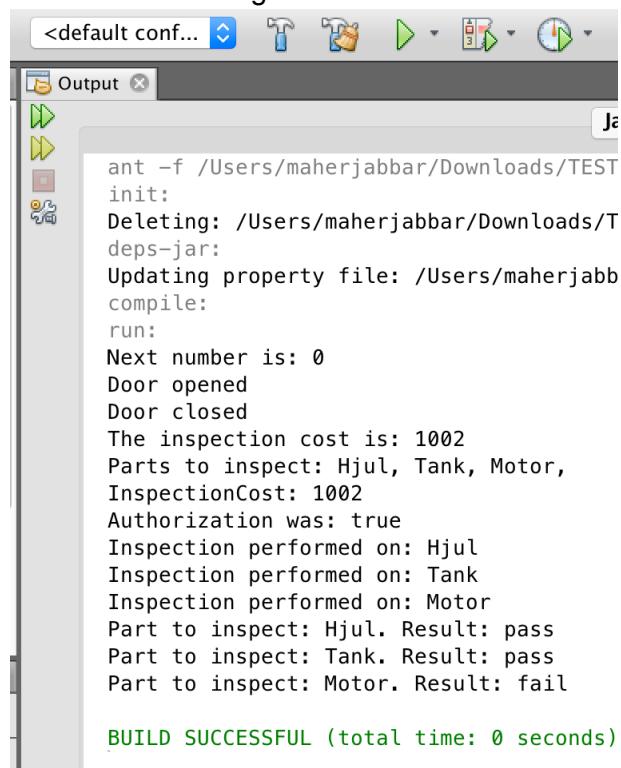
Page 13-15: Utskrift för samtliga inspektioner som utförts.

Page 16-18: Utskrift av alla inspektioner och deras resultat från skrivaren.

Efter utskriften testkörningen kommer en sektion som innehåller all kod för programmet.

Controller är involverad i alla åtgärder i programmet när controller anropas från view det vidarebefordra meddelandet till lämpliga klasser. InspektionCollector är ansvarig för att spara och hämta och vidarebefordra enstaka kontroller eller listor av inspektioner.

Figure 3.1



The screenshot shows the 'Output' tab of an IDE interface. The tab bar includes icons for file operations, a search field, and tabs for 'Output', 'Console', 'Tasks', and 'Markers'. The 'Output' tab displays the following log output:

```
<default conf... > [ ] Ja  
Output [ ]  
ant -f /Users/maherjabbar/Downloads/TEST  
init:  
Deleting: /Users/maherjabbar/Downloads/T  
deps-jar:  
Updating property file: /Users/maherjab  
compile:  
run:  
Next number is: 0  
Door opened  
Door closed  
The inspection cost is: 1002  
Parts to inspect: Hjul, Tank, Motor,  
InspectionCost: 1002  
Authorization was: true  
Inspection performed on: Hjul  
Inspection performed on: Tank  
Inspection performed on: Motor  
Part to inspect: Hjul. Result: pass  
Part to inspect: Tank. Result: pass  
Part to inspect: Motor. Result: fail  
  
BUILD SUCCESSFUL (total time: 0 seconds)
```

```

1 package vehicleinsp.main;
2
3 import vehicleinsp.controller.Controller;
4 import vehicleinsp.integration.Garage;
5 import vehicleinsp.integration.Printer;
6 import vehicleinsp.registry.RegistryCreator;
7 import vehicleinsp.view.View;
8 import vehicleinsp.model.PaymentController;
9
10 public class vehicleinsp {
11
12     public static void main(String[] args) {
13         Garage garage = new Garage();
14         Printer printer = new Printer();
15         RegistryCreator regCreator = new RegistryCreator(printer);
16         PaymentController paymContr = new PaymentController(printer);
17         Controller contr = new Controller(garage, regCreator, paymContr);
18         View view = new View(contr);
19         view.sampleExecution();
20     }
21 }
22 }
23
24 }
```

```

1 package vehicleinsp.view;
2 import vehicleinsp.controller.Controller;
3 import vehicleinsp.model.Inspection;
4
5 import vehicleinsp.controller.Controller;
6
7 public class View {
8     private Controller contr;
9     private String licenseNumber = "abc123";
10
11     /**
12      * Skapar en ny instans.
13      * @param contr används för alla operationer.
14      */
15     public View(Controller contr){
16         this.contr = contr;
17     }
18
19     /**
20      * Ett utförande prov
21      */
22     public void sampleExecution(){
23         contr.nextInspection();
24         contr.closeDoor();
25
26         // Skriv ut för kostnaden för inspektion
27         int inspectionCost = contr.getCost(licenseNumber);
28         System.out.println("The inspection cost is: " + inspectionCost);
29
30         // Skriv ut om authorization
31         boolean auth = contr.cardPaym(1002, licenseNumber, "sdfsdf", "dfsdf", "sdfsdf", 343);
32         System.out.println("Authorization was: " + auth);
33
34         // Utskrifter för när en inspektion har gjorts
35         Inspection nextInspection = contr.paymentDone(licenseNumber);
36         boolean lastInspection = nextInspection.getIsLast();
37         int inspNr = 1;
38         while(!(lastInspection)){
39             System.out.println("Inspection performed on: " + nextInspection.getInspectionPart());
40             nextInspection = contr.storeInspection(inspNr, true);
41             lastInspection = nextInspection.getIsLast();
42             inspNr++;
43         }
44         System.out.println("Inspection performed on: " + nextInspection.getInspectionPart());
45         nextInspection = contr.storeInspection(inspNr, false);
46     }
47 }
```

```
2 package vehicleinsp.controller;
3
4 import vehicleinsp.integration.Garage;
5 import vehicleinsp.registry.InspectionCollector;
6 import vehicleinsp.registry.RegistryCreator;
7 import vehicleinsp.model.PaymentController;
8 import vehicleinsp.model.ListOfInspections;
9 import vehicleinsp.model.CreditCard;
10 import vehicleinsp.model.Inspection;
11
12 public class Controller {
13     private Garage garage;
14     private RegistryCreator regCreator;
15     private InspectionCollector inspColl;
16     private PaymentController paymContr;
17
18     public Controller(Garage garage, RegistryCreator regCreator, PaymentController paymContr){
19         this.regCreator = regCreator;
20         this.inspColl = regCreator.getInspColl();
21         this.paymContr = paymContr;
22         this.garage = garage;
23     }
24
25     /**
26      * Metoden informerar programmet nästa inspektion ska göras
27      */
28     public void nextInspection(){
29         garage.nextInspection();
30     }
31
32     /**
33      * stängd door
34      */
35     public void closeDoor(){
36         garage.closeDoor();
37     }
38
39     /**
40      * Returns kostnaden för hela inspektion
41      * @param licenseNumber licensnummer på fordon som ska inspekteras
42      * @return en int kostnaden för hela inspektion göras
43      */
44     public int getCost(String licenseNumber){
45         int cost = inspColl.getCost(licenseNumber);
46         return cost;
47     }
48
49     /**
50      * Metod för kontant betalning returnerar den förändring som ska lämnas ut.
51      * @param mängd kontanter ges från kund
52      * @param licensnummer licensnummer från fordon som ska inspekteras
53      * @return mängd kontanter som ska ges tillbaka till kunden
54      */
55     public int payWithCash(int cashAmmount, String licenseNumber){
56         ListOfInspections list = inspColl.getInspections(licenseNumber);
57         int inspectionCost = inspColl.getCost(licenseNumber);
58         int change = paymContr.cashPaym(cashAmmount, inspectionCost, list);
59         return change;
60     }
61
62     /**
63      * Metoden utför ett tillstånd för kortbetalningar
64      * inspectionCost: Kostnaden för hela inspektion
65      * licenseNumber: fordon regNummer
66      * cardNumber: kunder kortnummer
67      * holder: Kunders Namn
68      * CVC : Kunders CVC
69      * expDate: kundkort utgångsdatum
70      * @return a boolean godkänd eller underkänd om kortbetalning har accepterats från
71      * externt betalningssystem.
72      */
73     public boolean cardPaym(int inspectionCost, String licenseNumber,
74                             String cardNumber, String holder, String expDate, int CVC){
75         CreditCard card = new CreditCard(CVC, cardNumber, holder, expDate);
76         ListOfInspections list = inspColl.getInspections(licenseNumber);
77         boolean authorization = paymContr.cardPaym(card, list, inspectionCost);
78         return authorization;
79     }
80
81     /**
82      * Informerar program som betalning har gjorts och returnerar den första
83      * inspektionen gör
84      * regNummer av fordonet för att utföra inspektioner på.
85      * @return den första inspektionen som skall utföras.
86      */
87     public Inspection paymentDone(String licenseNumber){
88         return inspColl.getFirstInspection(licenseNumber);
89     }
90
91     /**
92      * Lagrar resultatet på utförd kontroll
93      * index: index på inspektion som har utförts
94      * result: på inspektion godkänd / underkänd
95      * @return nästa kontroll som skall utföras
96      */
97     public Inspection storeInspection(int index, boolean result){
98         return inspColl.storeResult(index, result);
99     }
100 }
```

```

package vehicleinsp.registry;
import vehicleinsp.integration.Printer;

/**
 *
 * Denna klass är ansvarig för att skapa licenseNumReg och inspectionCollector
 *
 */
public class RegistryCreator {
    private Printer printer;
    private LicenseNumberRegistry licReg = new LicenseNumberRegistry();
    private InspectionCollector inspColl = new InspectionCollector();

    /**
     * Skapar en ny instans
     * @param printer
     */
    public RegistryCreator(Printer printer){
        this.printer = printer;
    }

    public LicenseNumberRegistry getLicNumpReg(){
        return licReg;
    }

    public InspectionCollector getInspColl(){
        return inspColl;
    }
}

1 package vehicleinsp.registry;
2 import vehicleinsp.integration.Printer;
3 import vehicleinsp.model.ListOfInspections;
4 import vehicleinsp.model.Inspection;
5
6 /**
7 * Denna klass är ansvarig för inspektioner hanterings
8 *
9 */
10 public class InspectionCollector {
11     private Printer printer = new Printer();
12     private InspectionRegistry inspReg = new InspectionRegistry();
13     private ListOfInspections insplist = new ListOfInspections();
14
15     /**
16     * skapar en ny instans
17     */
18     InspectionCollector(){
19
20     }
21
22     /**
23     * Denna metod skapar en lista över kontroller och fyller det med inspektioner
24     * fordonet med licensnumret som har gett
25     * @param licenseNumber antalet fordon licens
26     * @return en lista fylld med inspektioner
27     */
28     public ListOfInspections getInspections(String licenseNumber){
29         ListOfInspections list = new ListOfInspections();
30         inspReg.getInspections(licenseNumber, list);
31         setLastInspection(list);
32         return list;
33     }
}

```

```

34 /**
35 * Returnerar kostnaden för en lista över kontroller
36 * @param licenseNumber licensnummer på fordonet att listan hänvisningar till.
37 * @return kostnaden för alla inspektioner i listan.
38 */
39 public int getCost(String licenseNumber){
40     ListOfInspections list = getInspections(licenseNumber);
41     return list.getCost();
42 }
43 /**
44 * Denna metod ställer in boolean senaste inspektionen på inspektion med högst indexx
45 * i listan som skickas till denna metod.
46 * @param En lista över inspektioner.
47 */
48 private void setLastInspection(ListOfInspections list){
49     int len = list.getList().size();
50     list.getList().get(len - 1).setLastInspection(true);
51 }
52 /**
53 * Denna metod kallas när betalning har skett från controller och returnerar
54 * första inspektion måste göras.
55 * @param licenseNumber licensnummer på fordonet som ska inspekteras
56 * @return första inspektion måste göras.
57 */
58 public Inspection getFirstInspection(String licenseNumber){
59
60     inspReg.getInspections(licenseNumber, inspList);
61     Inspection inspection = inspList.getList().get(0);
62     return inspection;
63 }
64 /**
65 * Lagrar resultatet av en inspektion och returnerar nästa kontroll som
66 * kommer att utföras.
67 * @param index, Index - en i inspList är inspektionen att resultatet
68 * bör lagras i och inspektion med index ges är inspektionen
69 * som ska returneras .
70 * @param result resultatet av utförd kontroll, godkänd / underkänd .
71 * @return nästa kontroll som ska utföras.
72 */
73 public Inspection storeResult(int index, boolean result){
74     int size = inspList.getList().size() - 1;
75     inspList.getList().get(index - 1). setResult(result);
76
77     if(size == index - 1){
78         printResults();
79         return null;
80     }
81     return getNextInspection(index);
82 }
83
84 /**
85 * Skickar en lista över inspektion som ska skrivas ut till skrivaren
86 */
87
88 private void printResults(){
89     printer.printListOfInspections(inspList);
90 }
91
92 /**
93 *
94 * @param index index i listan
95 * @return en inspektion
96 */
97 private Inspection getNextInspection(int index){
98     return inspList.getList().get(index);
99 }
100 }
101

```

```
2 package vehicleinsp.model;
3 public class CreditCard {
4
5     private int CVC;
6     private String number;
7     private String holder;
8     private String expDate;
9
10    /**
11     * Skapar en ny instans
12     * @param CVC
13     * @param number
14     * @param holder
15     * @param expDate
16     */
17    public CreditCard(int CVC, String number, String holder, String expDate) {
18        this.CVC = CVC;
19        this.number = number;
20        this.holder = holder;
21        this.expDate = expDate;
22    }
23
24    public String getNumber(){
25        return number;
26    }
27
28    public String getHolder(){
29        return holder;
30    }
31
32    public String getExpDate(){
33        return expDate;
34    }
35
36    public int getCVC(){
37        return CVC;
38    }
}
```

```
2 package vehicleinsp.model;
3 public class Inspection {
4     private String partToInspect;
5     private String licenseNumber;
6     private int cost;
7     private boolean isLastInspection = false;
8     private boolean result = false;
9     /**
10      * Skapar en ny instans
11      * @param cost kostnaden för inspektion
12      * @param partToInspect del för att inspektera
13      * @param licenseNumber licens nummer på fordon
14      */
15     public Inspection(int cost, String partToInspect, String licenseNumber){
16         this.cost = cost;
17         this.partToInspect = partToInspect;
18         this.licenseNumber = licenseNumber;
19     }
20     /**
21      * Ställer in resultatet på en inspektion
22      * @param result
23      */
24     public void setResult(boolean result){
25         this.result = result;
26     }
27     /**
28      * Ställer in boolean attribut isLastInspection till true på en inspektion
29      * @param isLast
30      */
31     public void setLastInspection(boolean isLast){
32         this.isLastInspection = isLast;
33     }
34
35     public String getInspectionPart(){
36         return partToInspect;
37     }
38     public int getCost(){
39         return cost;
40     }
41
42     public String getLicenseNumber(){
43         return licenseNumber;
44     }
45
46     public boolean getIsLast(){
47         return isLastInspection;
48     }
49
50     public boolean getResult(){
51         return result;
52     }
53 }
```

```

55 /**
56 * Jämförelse av två kontroller har samma attribut
57 * @param inspection inspektionen som skall jämföras med
58 * @return boolean sant om lika falskt om inte.
59 */
60 public boolean equals(Object object){
61
62     if(object == null){
63         return false;
64     }
65     Inspection inspection = (Inspection) object;
66
67     if(!(this.cost == inspection.getCost())){
68         return false;
69     }
70     if(!(this.licenseNumber == inspection.getLicenseNumber())){
71         return false;
72     }
73     if(!(this.partToInspect == inspection.getInspectionPart())){
74         return false;
75     }
76     if(!(this.result == inspection.getResult())){
77         return false;
78     }
79     if(!(this.isLastInspection == inspection.getIsLast())){
80         return false;
81     }
82     return true;
83 }
84
85 }
86
2 package vehicleinsp.model;
3 import java.util.ArrayList;
4 import java.util.List;
5 import java.lang.StringBuilder;
6
7 public class ListOfInspections {
8     private int cost = 0;
9     private List<Inspection> list = new ArrayList<Inspection>();
10
11    /**
12     * skapar en ny instans
13     */
14    public ListOfInspections(){
15    }
16
17    public List<Inspection> getList(){
18        return list;
19    }
20
21    /**
22     * Returnerar kostnaden för alla kontroller i listan
23     * @return kostnaden
24     */
25    public int getCost(){
26        return cost;
27    }
28
29    /**
30     * Lagrar en inspektion i en lista och uppdaterar kostnaden för hela listan
31     * @param inspection Kontrollen som ska lagras.
32     */
33    public void storeInspection(Inspection inspection){
34        list.add(inspection);
35        cost += inspection.getCost();
36    }
37
38    /**
39     * tar en ListOfInspections och returnerar en sträng som innehåller delar inspekteras
40     * och resultat
41     * @param lista_en_ListOfInsections
42     * @return En sträng som innehåller resultaten och delar inspekteras
43     */
44    public String resultsToString(ListOfInspections list){
45        StringBuilder sb = new StringBuilder();
46        int len = list.getList().size();
47        for (int i = 0; i < len; i++){
48            sb.append("Part to inspect: ");
49            sb.append(list.getList().get(i).getInspectionPart() + "\n");
50            if(list.getList().get(i).getResult()){
51                sb.append("Result: pass" + " ");
52            } else{
53                sb.append("Result: fail" + " ");
54            }
55        }
56    }
57
58    return sb.toString();
59 }
60
61 }
62

```

```

1 package vehicleinsp.model;
2 import vehicleinsp.integration.Printer;
3 import vehicleinsp.integration.PaymentAuthorizationSys;
4
5
6 /**
7 * Denna klass är ansvarig för hantering av betalningen.
8 */
9 public class PaymentController {
10     private Printer printer;
11     private PaymentAuthorizationSys pas;
12
13     /**
14      * Skapar en ny instans
15      * @param printer
16      */
17     public PaymentController(Printer printer){
18         this.printer = printer;
19         this.pas = new PaymentAuthorizationSys();
20     }
21     /**
22      * En metod för kontant betalning tillbaka den förändring som ska lämnas tillbaka till kunden.
23      * skickar listan inspektioner som skall utföras till skrivare som är ansvarig för att skriva ut.
24      * @param cashAmmount många kontanter som betalas av kunden.
25      * @param inspectionCost kostnaden för hela inspektion
26      * @param list listan över kontroller som kommer att utföras.
27      * @return change, ammount av kontanter för att lämnas tillbaka till kunden.
28      */
29     public int cashPaym(int cashAmmount, int inspectionCost, ListOfInspections list){
30         Receipt receipt = new Receipt(inspectionCost, list);
31         printer.printReceipt(receipt);
32
33         int change = cashAmmount - inspectionCost;
34         System.out.println("Change: " + change);
35         return change;
36     }
37     /**
38      * En metod för kortbetalning gav tillstånd för betalning. Om kortet
39      * fick godkänt metoden kommer att kalla skrivaren för att skriva ut ett kvitto för inspektion.
40      * @param card kundens kort
41      * @param list lista över alla inspektioner göras, skickar den till skrivaren som skriver ut.
42      * @param inspectionCost Kostnaden för hela inspektion
43      * @return a boolean som är sant om kreditkort har godkänts av paymentauthorizationsys.
44      * och falskt om inte.
45      */
46     public boolean cardPaym(CreditCard card, ListOfInspections list, int inspectionCost){
47         Receipt receipt = new Receipt(inspectionCost, list);
48         boolean authorization = pas.getAuthorization(inspectionCost, card);
49         if(authorization){
50             printer.printReceipt(receipt);
51         }
52         return authorization;
53     }
54 }
55
56
57

```

```

2 package vehicleinsp.model;
3 import java.lang.StringBuilder;
4
5 public class Receipt {
6     private StringBuilder sb = new StringBuilder();
7     private int cost;
8
9     /**
10      * Skapar en ny instans
11      * @param inspectionCost kostnaden för inspektion
12      * @param list lista över inspektioner som har utförts
13      */
14     Receipt(int inspectionCost, ListOfInspections list){
15         cost = inspectionCost;
16         int len = list.getList().size();
17         for (int i = 0; i < len; i++){
18             sb.append(list.getList().get(i).getInspectionPart() + " ");
19         }
20     }
21
22     public String getSBToString(){
23         return sb.toString();
24     }
25
26     public int getCost(){
27         return cost;
28     }
29

```

```

2 package vehicleinsp.integration;
3
4 /**
5 * Denna klass är ansvarig för att visa rätt nummer på den fysiska display
6 */
7 public class Display {
8
9     Display(){}
0
1
2
3     void displayNumber(int number){
4         System.out.println("Next number is: " + number);
5     }
6
7 }
8

```

```

package vehicleinsp.integration;

public class Garage {

    private GarageDoor door;
    private Display display;
    private Queue queue;

    public Garage(){
        door = new GarageDoor();
        display = new Display();
        queue = new Queue();
    }

    /**
     * denna metod anropas när en ny inspektörer skall utföras. den hämtar
     * det aktuella könummern och skickar den till den display som visar antalet
     * Det öppnar också upp garageporten .
     */
    public void nextInspection(){
        int number = queue.getQueueNumber();
        display.displayNummber(number);
        door.openDoor();
    }

    public void closeDoor(){
        door.closeDoor();
    }

    /**
     * Hämtar meddelandet om dörren.
     * @return sant om dörren är öppen och falskt om inte.
     */
    public boolean getDoor(){
        return door.isOpen();
    }
}

```

```
1 package vehicleinsp.integration;
2
3
4 public class GarageDoor {
5     private boolean openDoor;
6
7     GarageDoor(){
8         openDoor = false;
9     }
10    /**
11     * Informrar om tillståndet dörren är in
12     * @return returnerar true om dörren är öppen och falskt om inte
13     */
14    boolean isOpen(){
15        return openDoor;
16    }
17    /**
18     * Ställer dörren för att vara öppen
19     */
20    void openDoor(){
21        System.out.println("Door opened");
22        openDoor = true;
23    }
24    /**
25     * Ställer dörren att stängas
26     */
27    void closeDoor(){
28        System.out.println("Door closed");
29        openDoor = false;
30    }
31
32 }
33 }
```

```
1
2 package vehicleinsp.integration;
3 import vehicleinsp.model.CreditCard;
4
5 public class PaymentAuthorizationSys {
6
7     public PaymentAuthorizationSys(){
8
9     }
10    public boolean getAuthorization(int inspectionCost, CreditCard card){
11        return true;
12    }
13
14 }
15 }
```

```
1 package vehicleinsp.integration;
2
3 import vehicleinsp.model.ListOfInspections;
4 import vehicleinsp.model.Receipt;
5 import java.lang.StringBuilder;
6
7
8 public class Printer {
9
10    public Printer(){
11
12    }
13
14    /**
15     * Printa ut kvitto
16     * @param receipt ett kvitto som innehåller kostnad för inspektion och vilka delar som finns att inspektera.
17     */
18    public void printReceipt(Receipt receipt){
19        String stringprint = receipt.getSBToString();
20        int cost = receipt.getCost();
21
22        System.out.println("Parts to inspect: " + stringprint);
23        System.out.println("InspectionCost: " + cost);
24    }
25
26    /**
27     * När inspektionen har utförts skrivaren skriver ut resultatet av varje inspektion.
28     * @param list
29     */
30    public void printListOfInspections(ListOfInspections list){
31        String listPrintout = list.resultsToString(list);
32        System.out.println(listPrintout);
33    }
34}
```

```
1
2 package vehicleinsp.integration;
3
4 public class Queue {
5     private int queueNumber;
6
7     Queue(){
8         queueNumber = 0;
9     }
10
11    /**
12     * Ställer könummer
13     * @param queueNumber
14     */
15    void setQueueNumber(int queueNumber){
16        this.queueNumber = queueNumber;
17    }
18
19    /**
20     * ökar könummer
21     */
22    void incrementQueueNumber(){
23        queueNumber++;
24    }
25
26    /**
27     * @return aktuella könummer
28     */
29    int getQueueNumber(){
30        return queueNumber;
31    }
32}
```

```
2 package vehicleinsp.controller;
3
4 import org.junit.After;
5 import org.junit.AfterClass;
6 import org.junit.Before;
7 import org.junit.BeforeClass;
8 import org.junit.Test;
9 import static org.junit.Assert.*;
10 import vehicleinsp.model.*;
11 import vehicleinsp.integration.*;
12 import vehicleinsp.registry.InspectionCollector;
13 import vehicleinsp.registry.RegistryCreator;
14
15
16
17 public class ControllerTest {
18     private Printer printer = new Printer();
19     private Garage garage = new Garage();
20     private RegistryCreator regCreator = new RegistryCreator(printer);
21     private InspectionCollector inspColl = regCreator.getInspColl();
22     private PaymentController paymContr = new PaymentController(printer);
23     private Controller contr = new Controller(garage, regCreator, paymContr);
24     private String licenseNumber = "abc123";
25
26     public ControllerTest() {
27
28 }
29
30     @Test
31     public void testNextInspection() {
32         contr.nextInspection();
33         boolean result = garage.getDoor();
34         boolean expResult = true;
35         assertEquals("Door isn't open", result, expResult);
36     }
37
38     @Test
39     public void testCloseDoor() {
40         contr.closeDoor();
41         boolean result = garage.getDoor();
42         boolean expResult = false;
43         assertEquals("Door isn't closed", result, expResult);
44     }
45
46     @Test
47     public void testGetCost() {
48         int expResult = 1002;
49         int result = contr.getCost(licenseNumber);
50         assertEquals("Wrong cost", expResult, result);
51     }
52
53     @Test
54     public void testPaymentDone(){
55         Inspection expResult = new Inspection(234, "Hjul", licenseNumber);
56         Inspection result = contr.paymentDone(licenseNumber);
57         assertEquals("Wrong inspection got", result, expResult);
58     }
59
60     @Test
61     public void testStoreInspection(){
62         contr.paymentDone(licenseNumber);
63         Inspection result = contr.storeInspection(1, false);
64         Inspection expResult = new Inspection(334, "Tank", "abc123");
65         assertEquals("Wrong inspection got", result, expResult);
66     }
67
68     @Test
69     public void testCardPayment(){
70         boolean result = contr.cardPaym(1231, licenseNumber, "12332sdfsdf", "Per Hansson", "2019-09-12", 232);
71         boolean expResult = true;
72         assertEquals("Wrong result", result, expResult);
73     }
74
75     @Test
76     public void testCashPayment(){
77         int expResult = 198;
78         int result = contr.payWithCash(1200, licenseNumber);
79         assertEquals("Incorrect change" + result, expResult, result);
80     }
81 }
82
```

```

1 package vehicleinsp.integration;
2
3 import org.junit.After;
4 import org.junit.AfterClass;
5 import org.junit.Before;
6 import org.junit.BeforeClass;
7 import org.junit.Test;
8 import static org.junit.Assert.*;
9
10 public class GarageDoorTest {
11     private GarageDoor door = new GarageDoor();
12
13     @Test
14     public void testOpenDoor() {
15         door.openDoor();
16         boolean openDoor = door.isOpen();
17         boolean expResult = true;
18         assertEquals("The door is not open.", expResult, openDoor);
19     }
20
21     @Test
22     public void testCloseDoor() {
23         door.closeDoor();
24         boolean openDoor = door.isOpen();
25         if(!(openDoor == false)){
26             fail("The test case is a prototype.");
27         }
28     }
29
30 }
31
32 package vehicleinsp.integration;
33
34 import org.junit.After;
35 import org.junit.AfterClass;
36 import org.junit.Before;
37 import org.junit.BeforeClass;
38 import org.junit.Test;
39 import static org.junit.Assert.*;
40
41 public class GarageTest {
42     private Garage garage = new Garage();
43
44     @Test
45     public void testNextInspection() {
46         garage.nextInspection();
47         boolean open = garage.getDoor();
48         if (!(open == true)){
49             fail("The door should be closed");
50         }
51     }
52
53     @Test
54     public void testCloseDoor() {
55         garage.closeDoor();
56         boolean open = garage.getDoor();
57         if(!(open == false)){
58             fail("The door should be closed");
59         }
60     }
61
62 }
63

```

```
1 package vehicleinsp.integration;
2
3 import org.junit.After;
4 import org.junit.AfterClass;
5 import org.junit.Before;
6 import org.junit.BeforeClass;
7 import org.junit.Test;
8 import static org.junit.Assert.*;
9
10 public class GarageDoorTest {
11     private GarageDoor door = new GarageDoor();
12
13     @Test
14     public void testOpenDoor() {
15         door.openDoor();
16         boolean openDoor = door.isOpen();
17         boolean expResult = true;
18         assertEquals("The door is not open.", expResult, openDoor);
19     }
20
21     @Test
22     public void testCloseDoor() {
23         door.closeDoor();
24         boolean openDoor = door.isOpen();
25         if(!(openDoor == false)){
26             fail("The test case is a prototype.");
27         }
28     }
29
30 }
31
```

```
2 package vehicleinsp.model;
3
4 import org.junit.After;
5 import org.junit.AfterClass;
6 import org.junit.Before;
7 import org.junit.BeforeClass;
8 import org.junit.Test;
9 import static org.junit.Assert.*;
10 import vehicleinsp.integration.Printer;
11
12
13 public class PaymentControllerTest {
14     private Printer printer = new Printer();
15     private PaymentController paymCont = new PaymentController(printer);
16     private CreditCard card = new CreditCard(123, "123abc", "Douglas Appelberg", "2019-01-01");
17     private ListOfInspections list = new ListOfInspections();
18
19     @Test
20     public void testCardPaym() {
21         boolean expResult = true;
22         boolean result = paymCont.cardPaym(card, list, 3456);
23         assertEquals("Should be true", expResult, result);
24     }
25
26 }
27
```

```
2 package vehicleinsp.model;
3
4 import org.junit.After;
5 import org.junit.AfterClass;
6 import org.junit.Before;
7 import org.junit.BeforeClass;
8 import org.junit.Test;
9 import static org.junit.Assert.*;
10 import vehicleinsp.model.Inspection;
11 import vehicleinsp.integration.Printer;
12 import vehicleinsp.registry.RegistryCreator;
13 import vehicleinsp.registry.InspectionCollector;
14
15
16 public class ReceiptTest {
17     private Printer printer = new Printer();
18     private PaymentController paymCont = new PaymentController(printer);
19     private RegistryCreator regCreator = new RegistryCreator(printer);
20     private InspectionCollector inspColl = regCreator.getInspColl();
21     private ListOfInspections list = inspColl.getInspections("abc123");
22     private Receipt receipt = new Receipt(1002, list);
23
24     public ReceiptTest() {
25
26     }
27
28     @Test
29     public void testGetSB() {
30         String result = receipt.getSBToString();
31         String expResult = "Hjul, Tank, Motor, ";
32         assertEquals("Wrong String", result, expResult);
33     }
34 }
```

```
1 package vehicleinsp.registry;
2
3 import org.junit.After;
4 import org.junit.AfterClass;
5 import org.junit.Before;
6 import org.junit.BeforeClass;
7
8 import org.junit.Test;
9 import static org.junit.Assert.*;
10 import vehicleinsp.model.Inspection;
11 import vehicleinsp.model.ListOfInspections;
12 import vehicleinsp.integration.Printer;
13
14
15 public class InspectionCollectorTest {
16     private InspectionCollector inspColl = new InspectionCollector();
17     private String licenseNumber = "abc123";
18
19     @Test
20     public void testGetInspections() {
21         ListOfInspections list = inspColl.getInspections(licenseNumber);
22         Inspection expResult = new Inspection(334, "Tank", "abc123");
23         Inspection result = list.getList().get(1);
24         assertEquals("Wrong object has been returned.", result, expResult);
25     }
26
27     @Test
28     public void testStoreResult(){
29         inspColl.getFirstInspection(licenseNumber);
30         Inspection result = inspColl.storeResult(2, true);
31         Inspection expResult = new Inspection (434, "Motor", "abc123");
32         assertEquals("Wrong object has been returned", result, expResult);
33     }
34
35     @Test
36     public void testGetFirstInspection(){
37         Inspection result = inspColl.getFirstInspection(licenseNumber);
38         Inspection expResult = new Inspection(234, "Hjul", "abc123");
39         assertEquals("Wrong object has been returned.", result, expResult);
40     }
41
42     }
43
44 }
```

Discussion

En misstag vi gjorde skriva koden var att göra enhetstester efter att ha skrivit programmet. Det skulle ha varit mer meningsfullt att skriva testerna när vi gjort en ny metod som vi ville testa. Vi har lyckats följa mönstret från seminariet två. koden tycker jag är lätt att förstå då det är hög sammanhållning, så alla klasser är namngivna efter det arbete de genomför och endast det. Vi har utgått mycket från att programmet ska se nästan ut som Design delen då koderna för varje klass ligger i paket. Vi använde både objekt och primitiv data i vår kod där vi använder oss av t.ex. int och boolean. I vår kod så har vi kommentarer för varje metod så att man förstår vad som ska ske så att man kan följa med koden. Testkörningen lyckades vi bra med vi testkörde även all kod för varje metod i klasserna. Det finns inga direkt duplicerade kodrader men när man kollar i efterhand så finns det några få rader som man kunnat extrahera till sina egna metoder istället. Själv tycker jag att det finns inga för långa metoder eller klasser i vår kod. Vi har kommenterat alla publika deklarationer, inga kommentarer inuti metoderna. Jag tycker att vårt program har bra inkapsling, detta då klasserna och metoderna endast innehåller kod kopplade till deras respektive uppgift.