Labb 4

Objektorienterad Design, IV1350

Evan Saboo
saboo@kth.se
2015-05-20

Innehåll

1 Introduktion

Syftet med fjärde och sista seminariet i kursen är att man ska lära sig att designa och koda undantagshantering, designmönster och polymorfism. Första uppgiften gick ut på att skapa ett undantag (exception) för hantering av licensnummer. I andra uppgiften skulle man använda observatör ”Observer” mönster för att visa resultaten av alla utförda inspektioner för ett visst fordon. För högre betyg skulle man använda en till GoF mönster i inspektionsprogrammet och designen.

2 Metod

I första uppgiften skapades undantagsklassen InvalidLicenseNumberException som används när användaren matar in något annat än inmatningen ”abc123” vilket är en hårdkodad lisensnummer. InvalidLicenseNumberException används i View för att meddela användaren om den angivna lisensnummer är felaktig. Undantagsnamnet beskriver ganska tydligt vad för slags undantag det är. InvalidLicenseNumberException är en så kallad ”checked” undantag, vilket använder ”throw” deklarationer.

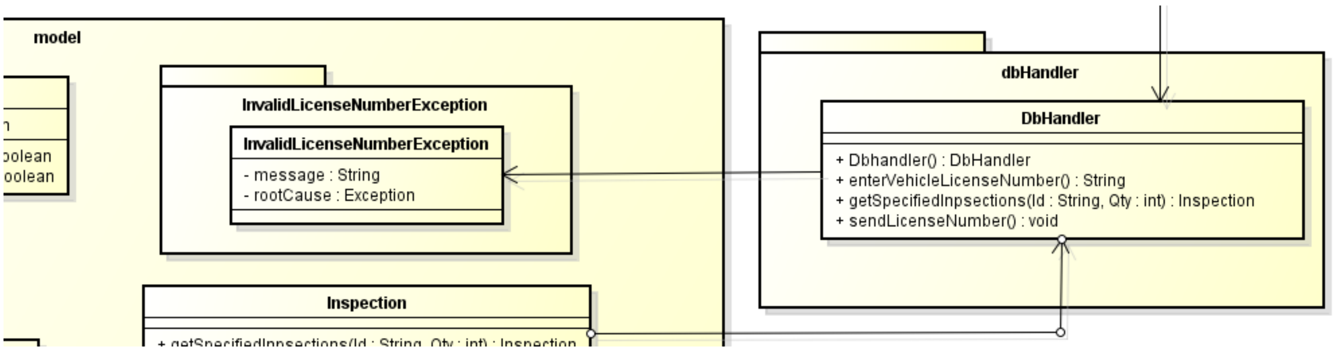
I första deluppgiften i uppgift 2 skapades observatör ”observer” mönstret för att skriva ut resultaten för inspektionen. Detta utfördes genom att skapa en observer interface och implementera det till klassen View. I andra deluppgiften skapades bara ett exemplar av garage door med hjälp av singleton mönstret. Detta utfördes genom att skapa en ny GarageDoor objekt som kallas för myGarageDoor. Metoden getGarageDoor returnerar myGarageDoor som en referens för kunna sedan användas i klassen Controller.

Alla uppgifter utfördes både som design och javaprogram för att ge en mycket bättre bild på hur de nya metoderna fungerar och hur undantagsklassen används i olika metoder och klasser.

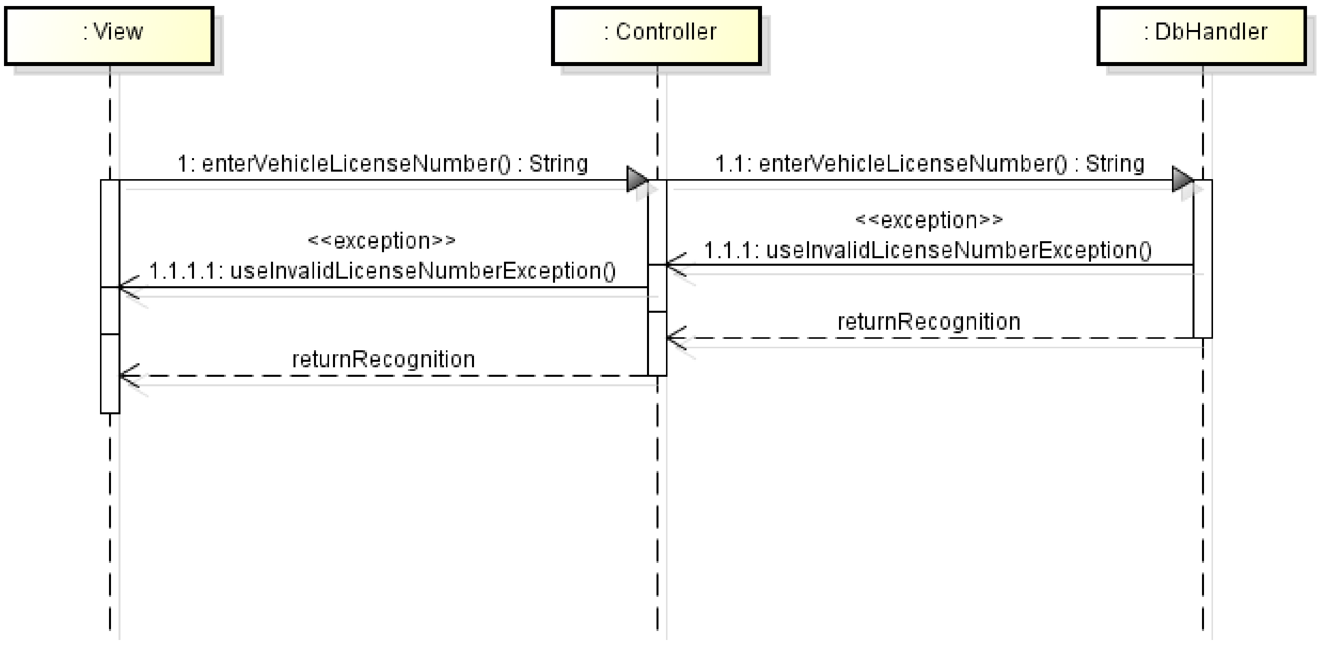
3 Resultat

All design och programkod utfördes med Emil Nordin.

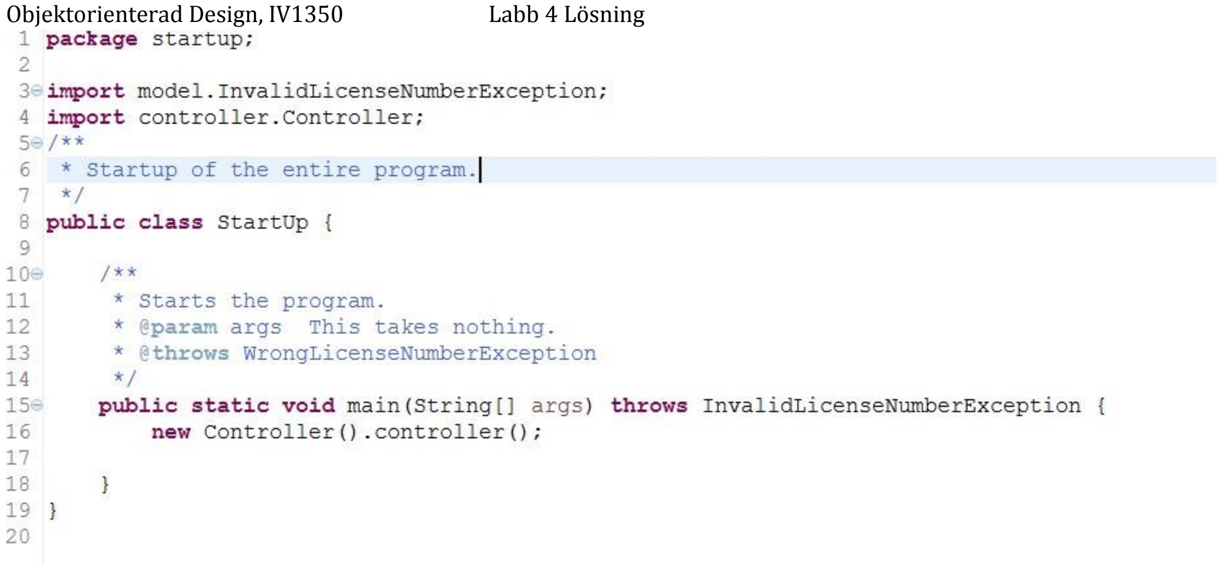
Uppgift 1:


model
InvalidLicenseNumberException
InvalidLicenseNumberException
- message : String
- rootCause : Exception
Inspection
+ getSpecifiedInpsections(Id : String, Qty : int) : Inspection
dbHandler
DbHandler
+ Dbhandler() : DbHandler
+ enterVehicleLicenseNumber() : String
+ getSpecifiedInpsections(Id : String, Qty : int) : Inspection
+ sendLicenseNumber() : void

Figur 1.0: En bild på klassen InvalidLicenseNumberException taget från klass diagrammet.

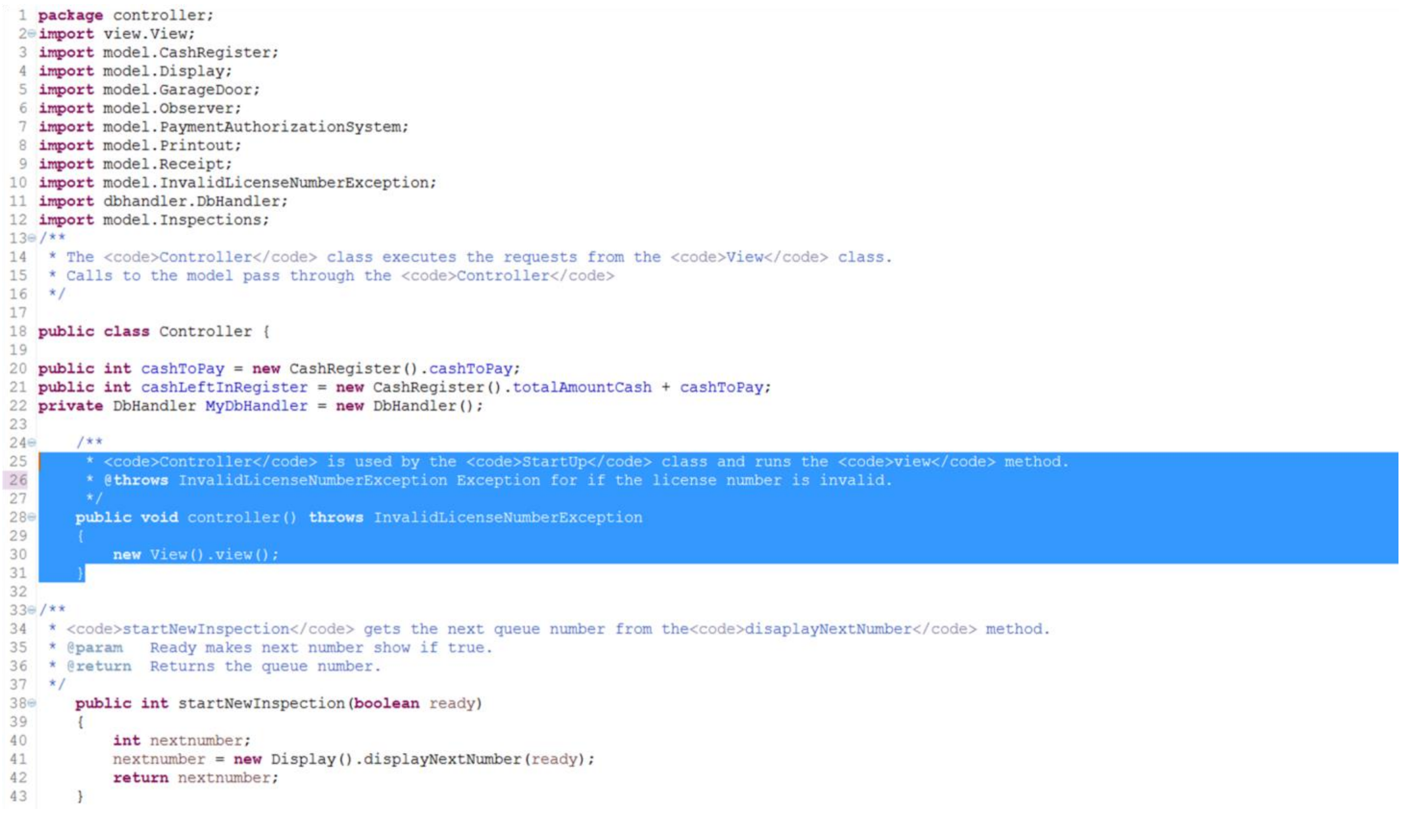

: View
: Controller
: DbHandler
1: enterVehicleLicenseNumber() : String
1.1: enterVehicleLicenseNumber() : String
<<exception>>
1.1.1: useInvalidLicenseNumberException()
<<exception>>
1.1.1.1: useInvalidLicenseNumberException()
returnRecognition
returnRecognition

Figur 1.1: En bild på lisensnummer validering med undantag taget från sekvensdiagrammet.



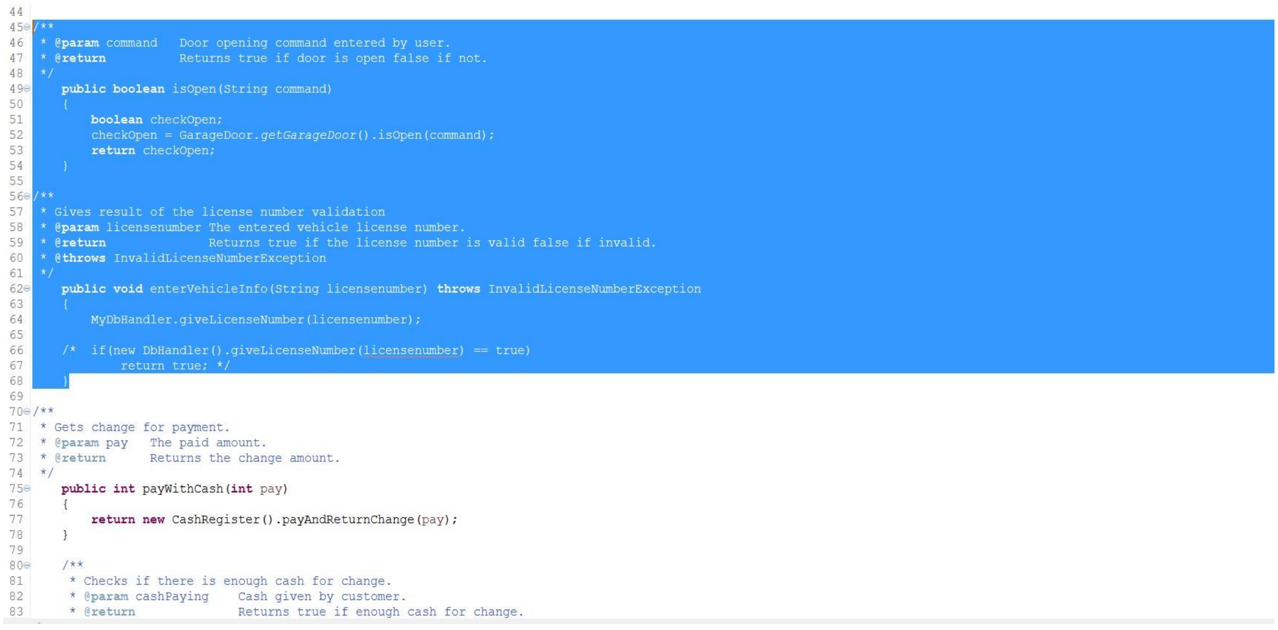
```java
package startup;

import model.InvalidLicenseNumberException;
import controller.Controller;
/**
 * Startup of the entire program.
 */
public class StartUp {

    /**
     * Starts the program.
     * @param args  This takes nothing.
     * @throws WrongLicenseNumberException
     */
    public static void main(String[] args) throws InvalidLicenseNumberException {
        new Controller().controller();

    }
}

```

Figur 1.2: En bild på klassen Startup som använder ”throws InvalidLicenseNumberException”.



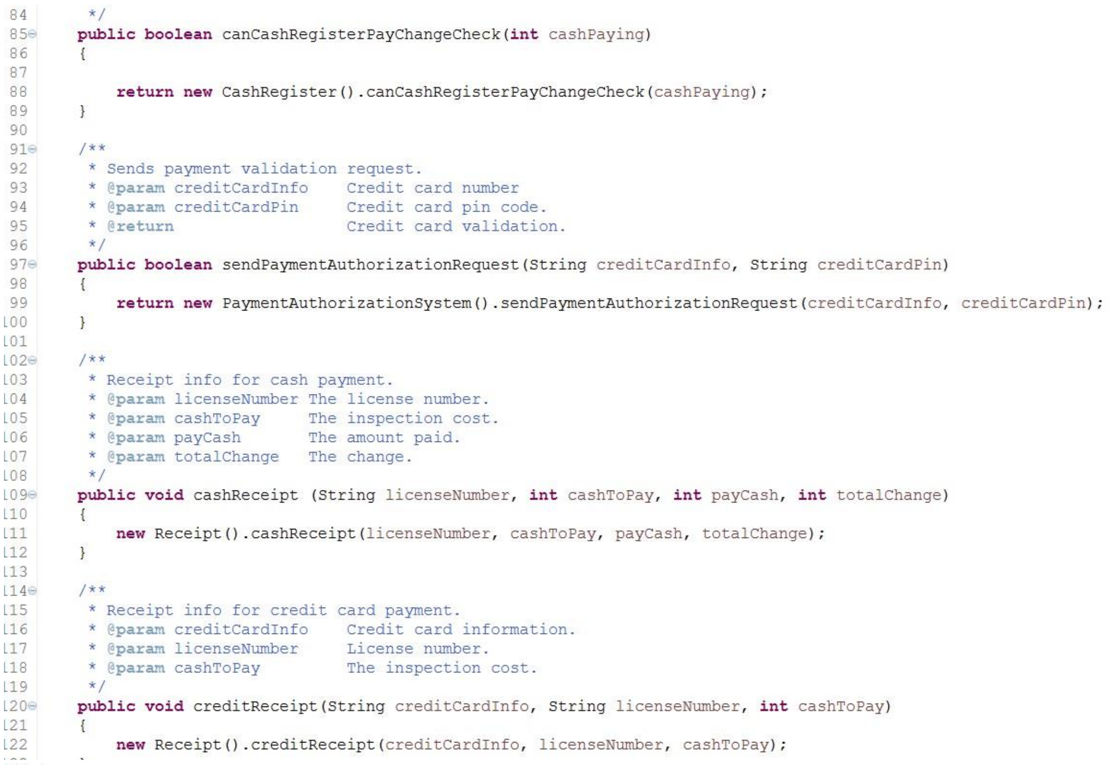
```java
package controller;
import view.View;
import model.CashRegister;
import model.Display;
import model.GarageDoor;
import model.Observer;
import model.PaymentAuthorizationSystem;
import model.Printout;
import model.Receipt;
import model.InvalidLicenseNumberException;
import dbhandler.DbHandler;
import model.Inspections;
/**
 * The <code>Controller</code> class executes the requests from the <code>View</code> class.
 * Calls to the model pass through the <code>Controller</code>
 */

public class Controller {

public int cashToPay = new CashRegister().cashToPay;
public int cashLeftInRegister = new CashRegister().totalAmountCash + cashToPay;
private DbHandler MyDbHandler = new DbHandler();

    /**
     * <code>Controller</code> is used by the <code>StartUp</code> class and runs the <code>view</code> method.
     * @throws InvalidLicenseNumberException Exception for if the license number is invalid.
     */
    public void controller() throws InvalidLicenseNumberException
    {
        new View().view();
    }

/**
 * <code>startNewInspection</code> gets the next queue number from the<code>disaplayNextNumber</code> method.
 * @param   Ready makes next number show if true.
 * @return  Returns the queue number.
 */
    public int startNewInspection(boolean ready)
    {
        int nextnumber;
        nextnumber = new Display().displayNextNumber(ready);
        return nextnumber;
    }
```

Figur 1.3: En bild på klassen Controller som använder ”throws InvalidLicenseNumberException”.

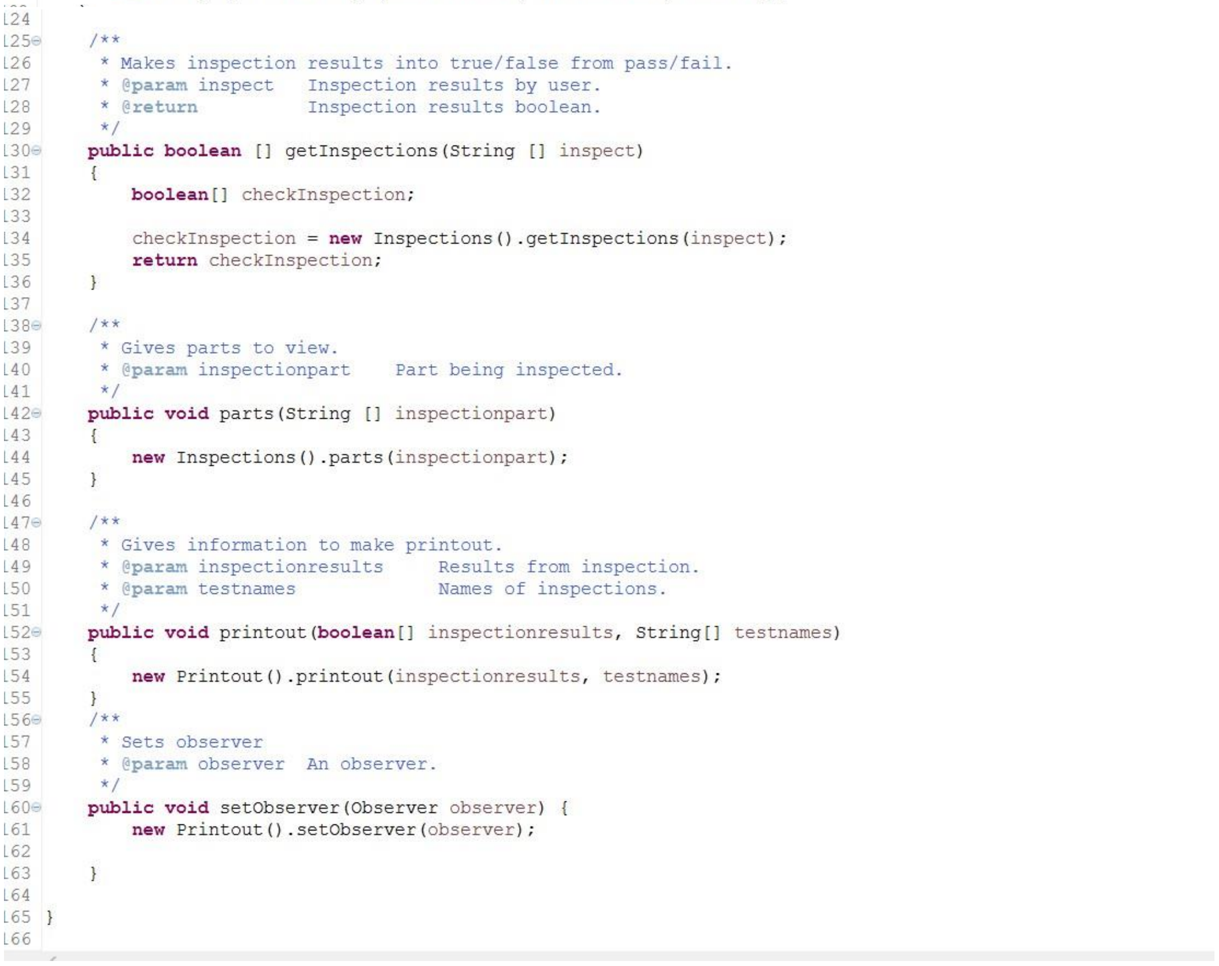


```java

/**
 * @param command   Door opening command entered by user.
 * @return          Returns true if door is open false if not.
 */
    public boolean isOpen(String command)
    {
        boolean checkOpen;
        checkOpen = GarageDoor.getGarageDoor().isOpen(command);
        return checkOpen;
    }

/**
 * Gives result of the license number validation
 * @param licensenumber The entered vehicle license number.
 * @return              Returns true if the license number is valid false if invalid.
 * @throws InvalidLicenseNumberException
 */
    public void enterVehicleInfo(String licensenumber) throws InvalidLicenseNumberException
    {
        MyDbHandler.giveLicenseNumber(licensenumber);

    /*  if(new DbHandler().giveLicenseNumber(licensenumber) == true)
            return true; */
    }

/**
 * Gets change for payment.
 * @param pay   The paid amount.
 * @return      Returns the change amount.
 */
    public int payWithCash(int pay)
    {
        return new CashRegister().payAndReturnChange(pay);
    }

    /**
     * Checks if there is enough cash for change.
     * @param cashPaying    Cash given by customer.
     * @return              Returns true if enough cash for change.
```

Figur 1.4: Ändrat kod på enterVehicleInfo som nu har ”throws InvalidLicenseNumbereException” i klassen Controller.

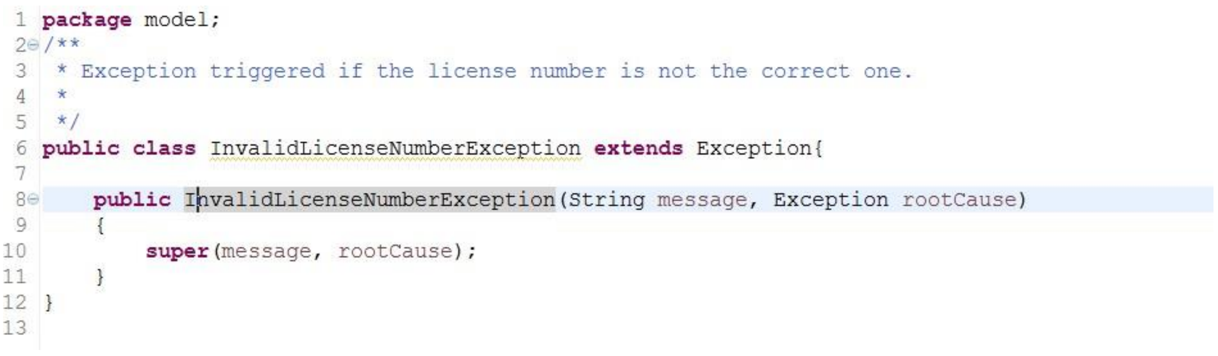


```java
     */
    public boolean canCashRegisterPayChangeCheck(int cashPaying)
    {

        return new CashRegister().canCashRegisterPayChangeCheck(cashPaying);
    }

    /**
     * Sends payment validation request.
     * @param creditCardInfo    Credit card number
     * @param creditCardPin     Credit card pin code.
     * @return                  Credit card validation.
     */
    public boolean sendPaymentAuthorizationRequest(String creditCardInfo, String creditCardPin)
    {
        return new PaymentAuthorizationSystem().sendPaymentAuthorizationRequest(creditCardInfo, creditCardPin);
    }

    /**
     * Receipt info for cash payment.
     * @param licenseNumber The license number.
     * @param cashToPay     The inspection cost.
     * @param payCash       The amount paid.
     * @param totalChange   The change.
     */
    public void cashReceipt (String licenseNumber, int cashToPay, int payCash, int totalChange)
    {
        new Receipt().cashReceipt(licenseNumber, cashToPay, payCash, totalChange);
    }

    /**
     * Receipt info for credit card payment.
     * @param creditCardInfo    Credit card information.
     * @param licenseNumber     License number.
     * @param cashToPay         The inspection cost.
     */
    public void creditReceipt(String creditCardInfo, String licenseNumber, int cashToPay)
    {
        new Receipt().creditReceipt(creditCardInfo, licenseNumber, cashToPay);
```



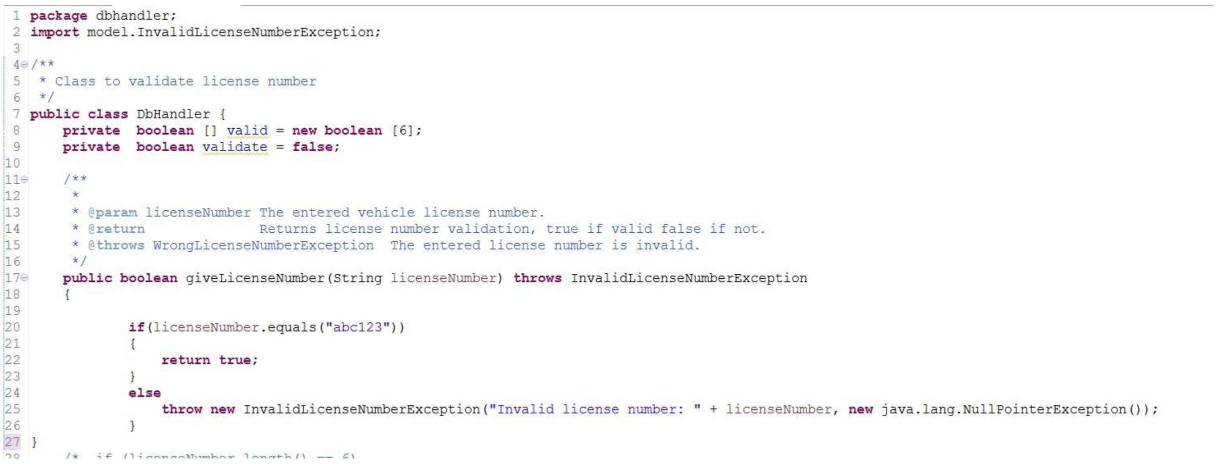
```java

    /**
     * Makes inspection results into true/false from pass/fail.
     * @param inspect   Inspection results by user.
     * @return          Inspection results boolean.
     */
    public boolean [] getInspections(String [] inspect)
    {
        boolean[] checkInspection;

        checkInspection = new Inspections().getInspections(inspect);
        return checkInspection;
    }

    /**
     * Gives parts to view.
     * @param inspectionpart    Part being inspected.
     */
    public void parts(String [] inspectionpart)
    {
        new Inspections().parts(inspectionpart);
    }

    /**
     * Gives information to make printout.
     * @param inspectionresults     Results from inspection.
     * @param testnames             Names of inspections.
     */
    public void printout(boolean[] inspectionresults, String[] testnames)
    {
        new Printout().printout(inspectionresults, testnames);
    }
    /**
     * Sets observer
     * @param observer  An observer.
     */
    public void setObserver(Observer observer) {
        new Printout().setObserver(observer);

    }

}

```

Figur 1.5: Bilder på oförändrad kod i klassen Controller.



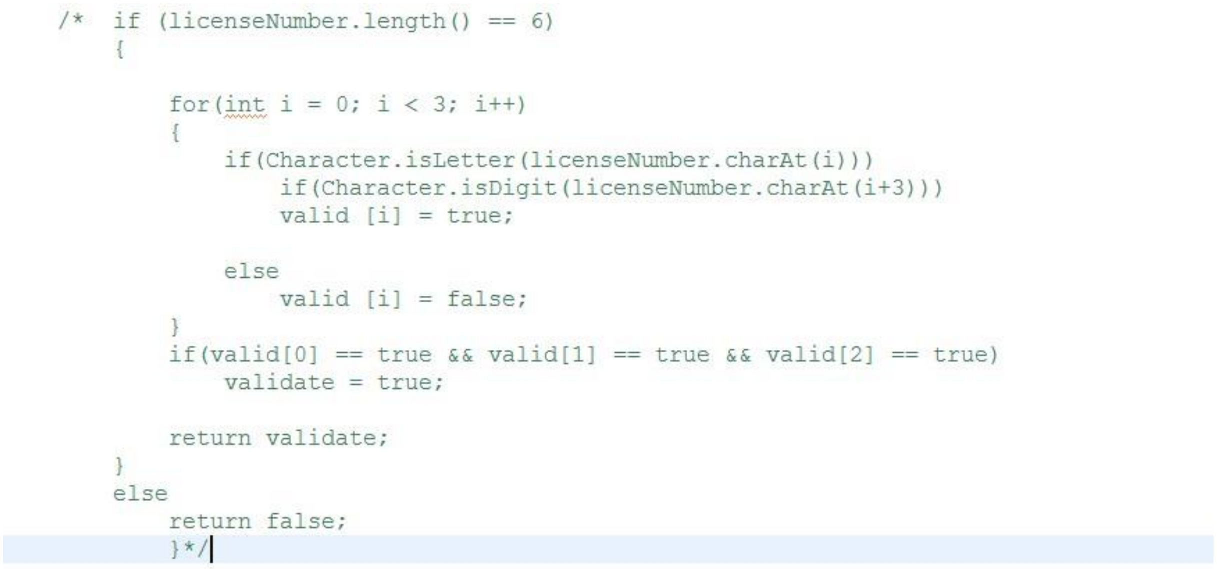
```java
package model;
/**
 * Exception triggered if the license number is not the correct one.
 *
 */
public class InvalidLicenseNumberException extends Exception{

    public InvalidLicenseNumberException(String message, Exception rootCause)
    {
        super(message, rootCause);
    }
}

```

Figur 1.6: En bild på klassen InvalidLicenseNumberException.



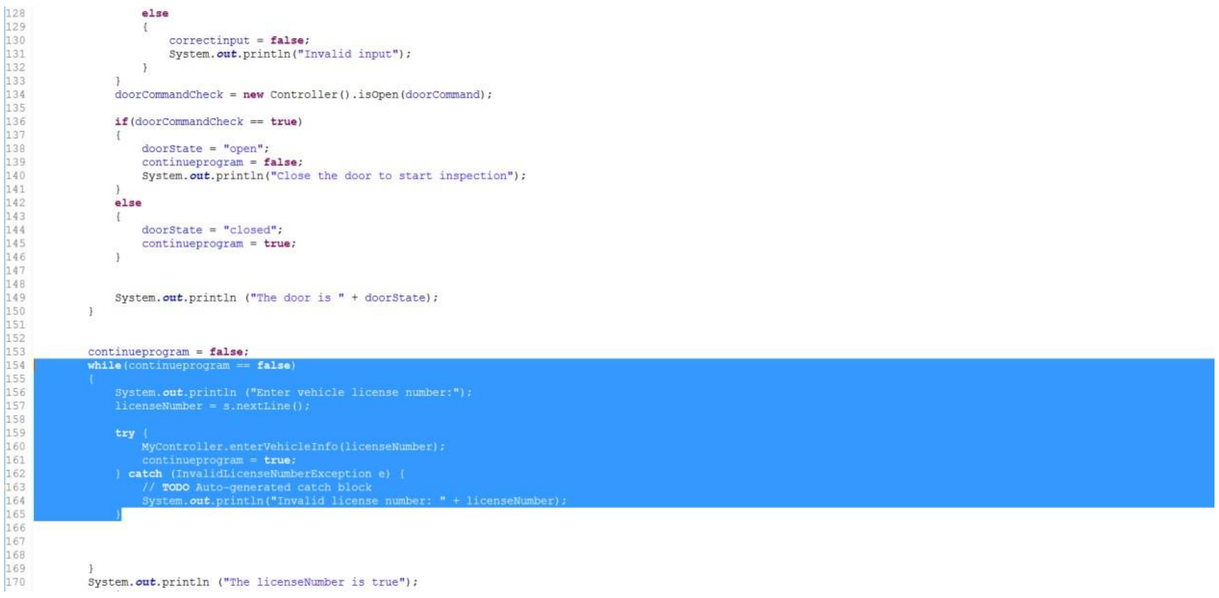
```java
package dbhandler;
import model.InvalidLicenseNumberException;

/**
 * Class to validate license number
 */
public class DbHandler {
    private  boolean [] valid = new boolean [6];
    private  boolean validate = false;

    /**
     *
     * @param licenseNumber The entered vehicle license number.
     * @return              Returns license number validation, true if valid false if not.
     * @throws WrongLicenseNumberException  The entered license number is invalid.
     */
    public boolean giveLicenseNumber(String licenseNumber) throws InvalidLicenseNumberException
    {

            if(licenseNumber.equals("abc123"))
            {
                return true;
            }
            else
                throw new InvalidLicenseNumberException("Invalid license number: " + licenseNumber, new java.lang.NullPointerException());
            }
}
    /*  if (licenseNumber.length() == 6)
```

Figur 1.7: En bild på ny kod i klassen DbHandler som använder InvalidLicenseNumberException.



```java
/*  if (licenseNumber.length() == 6)
    {

        for(int i = 0; i < 3; i++)
        {
            if(Character.isLetter(licenseNumber.charAt(i)))
                if(Character.isDigit(licenseNumber.charAt(i+3)))
                valid [i] = true;

            else
                valid [i] = false;
        }
        if(valid[0] == true && valid[1] == true && valid[2] == true)
            validate = true;

        return validate;
    }
    else
        return false;
        }*/
```

Figur 1.8: Gammal kod i klassen Dhandler som inte används längre.

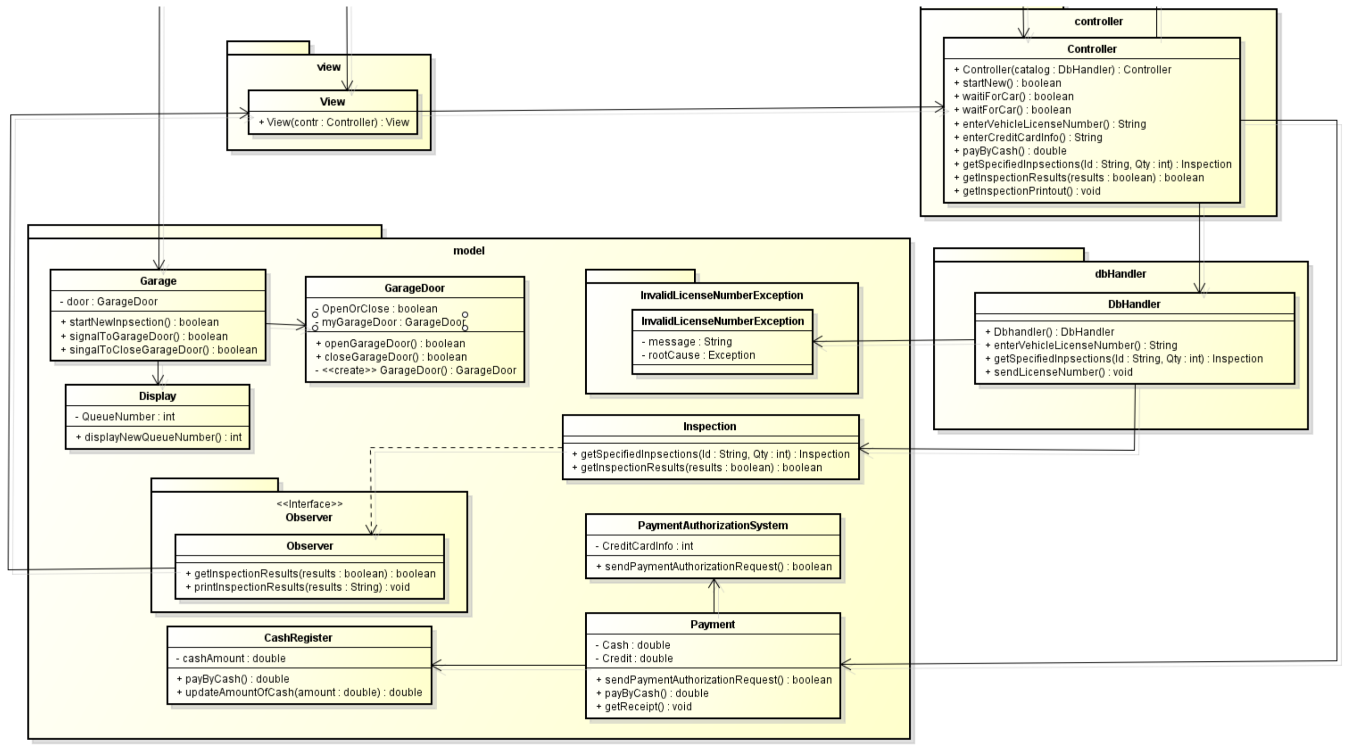


```
            else
            {
                correctinput = false;
                System.out.println("Invalid input");
            }
        }
        doorCommandCheck = new Controller().isOpen(doorCommand);

        if(doorCommandCheck == true)
        {
            doorState = "open";
            continueprogram = false;
            System.out.println("Close the door to start inspection");
        }
        else
        {
            doorState = "closed";
            continueprogram = true;
        }


        System.out.println ("The door is " + doorState);
    }


    continueprogram = false;
    while(continueprogram == false)
    {
        System.out.println ("Enter vehicle license number:");
        licenseNumber = s.nextLine();

        try {
            MyController.enterVehicleInfo(licenseNumber);
            continueprogram = true;
        } catch (InvalidLicenseNumberException e) {
            // TODO Auto-generated catch block
            System.out.println("Invalid license number: " + licenseNumber);
        }



    }
    System.out.println ("The licenseNumber is true");
```

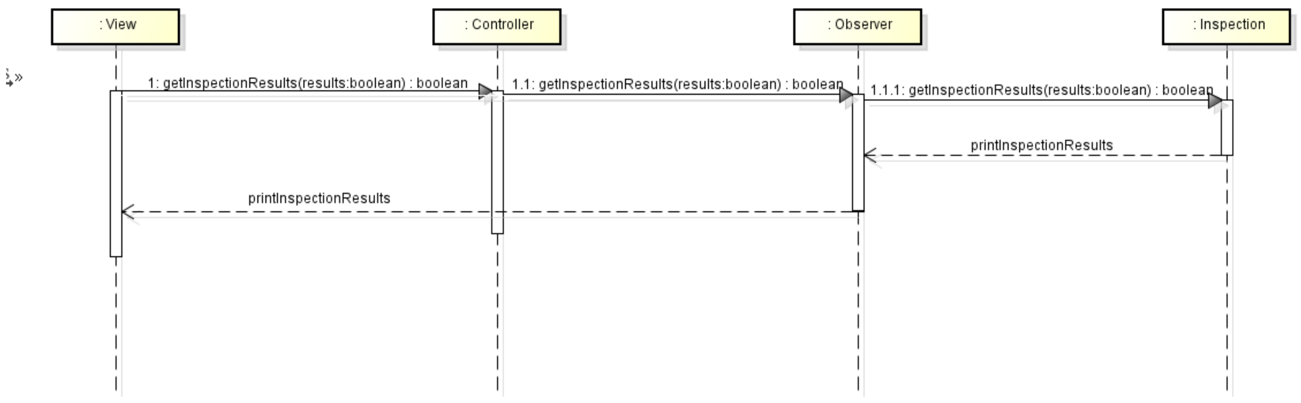
Figur 1.9: Ny markerad kod i klassen View.

Uppgift 2:

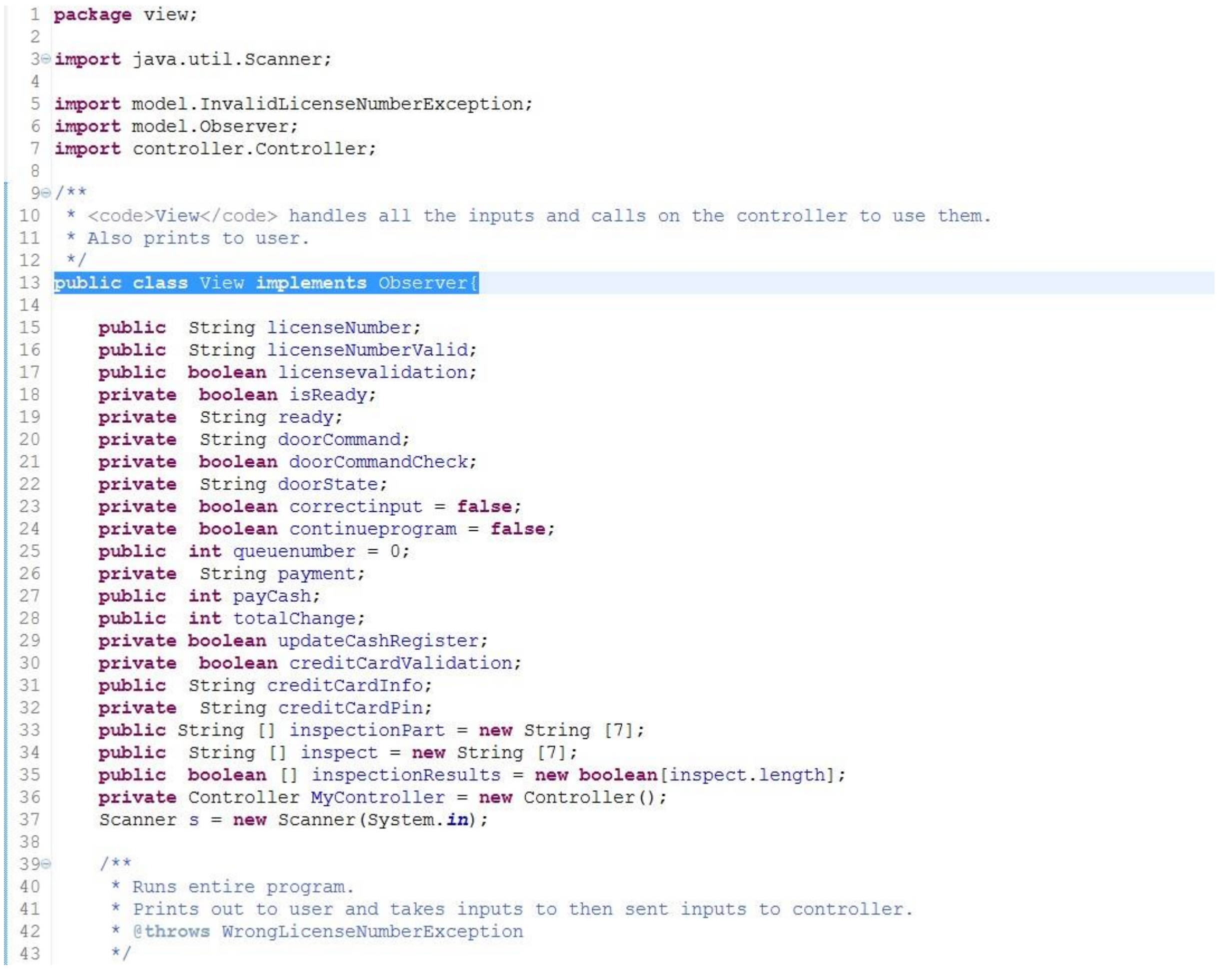
Deluppgift a:


view
View
+ View(contr : Controller) : View
controller
Controller
+ Controller(catalog : DbHandler) : Controller
+ startNew() : boolean
+ waitiForCar() : boolean
+ waitForCar() : boolean
+ enterVehicleLicenseNumber() : String
+ enterCreditCardInfo() : String
+ payByCash() : double
+ getSpecifiedInpsections(Id : String, Qty : int) : Inspection
+ getInspectionResults(results : boolean) : boolean
+ getInspectionPrintout() : void
model
Garage
- door : GarageDoor
+ startNewInpsection() : boolean
+ signalToGarageDoor() : boolean
+ singalToCloseGarageDoor() : boolean
GarageDoor
- OpenOrClose : boolean
- myGarageDoor : GarageDoor
+ openGarageDoor() : boolean
+ closeGarageDoor() : boolean
- <<create>> GarageDoor() : GarageDoor
InvalidLicenseNumberException
InvalidLicenseNumberException
- message : String
- rootCause : Exception
dbHandler
DbHandler
+ Dbhandler() : DbHandler
+ enterVehicleLicenseNumber() : String
+ getSpecifiedInpsections(Id : String, Qty : int) : Inspection
+ sendLicenseNumber() : void
Display
- QueueNumber : int
+ displayNewQueueNumber() : int
Inspection
+ getSpecifiedInpsections(Id : String, Qty : int) : Inspection
+ getInspectionResults(results : boolean) : boolean
<<Interface>>
Observer
Observer
+ getInspectionResults(results : boolean) : boolean
+ printInspectionResults(results : String) : void
PaymentAuthorizationSystem
- CreditCardInfo : int
+ sendPaymentAuthorizationRequest() : boolean
CashRegister
- cashAmount : double
+ payByCash() : double
+ updateAmountOfCash(amount : double) : double
Payment
- Cash : double
- Credit : double
+ sendPaymentAuthorizationRequest() : boolean
+ payByCash() : double
+ getReceipt() : void

Figur 2.0: En bild på klass diagram med Observer interface.

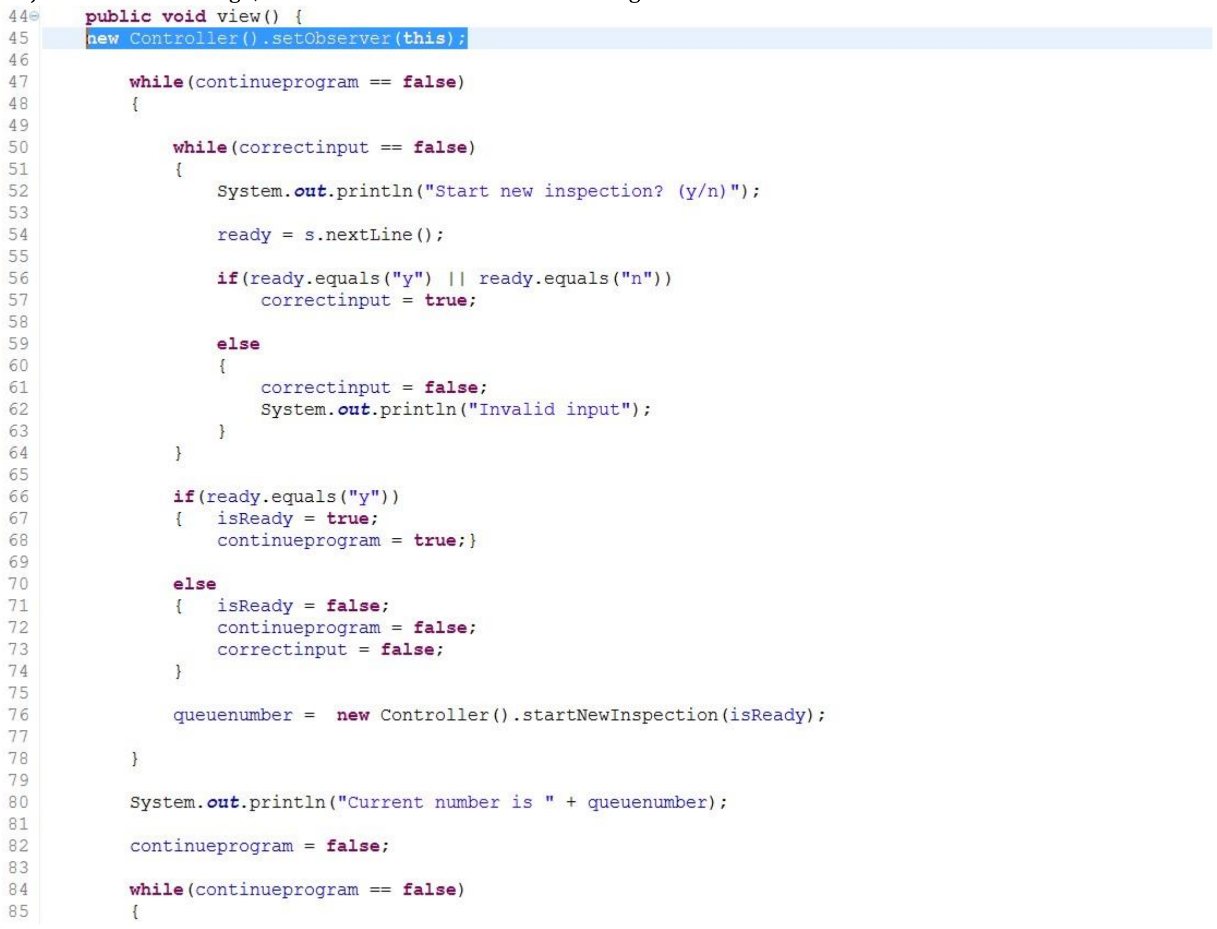

: View
: Controller
: Observer
: Inspection
1: getInspectionResults(results:boolean) : boolean
1.1: getInspectionResults(results:boolean) : boolean
1.1.1: getInspectionResults(results:boolean) : boolean
printInspectionResults
printInspectionResults

Figur 2.1: Ny sekvensdiagram för Inspections med Observer.



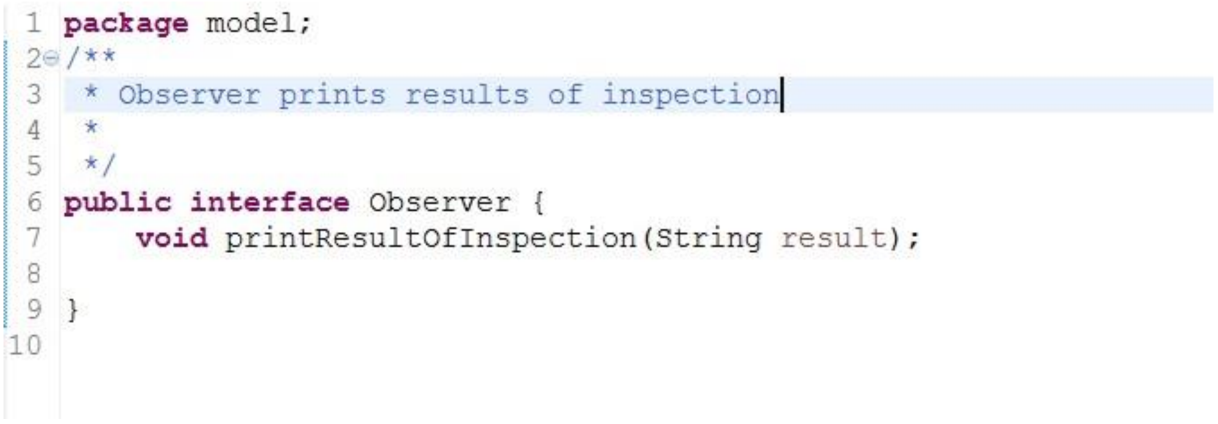
```java
package view;

import java.util.Scanner;

import model.InvalidLicenseNumberException;
import model.Observer;
import controller.Controller;

/**
 * <code>View</code> handles all the inputs and calls on the controller to use them.
 * Also prints to user.
 */
public class View implements Observer{

    public  String licenseNumber;
    public  String licenseNumberValid;
    public  boolean licensevalidation;
    private  boolean isReady;
    private  String ready;
    private  String doorCommand;
    private  boolean doorCommandCheck;
    private  String doorState;
    private  boolean correctinput = false;
    private  boolean continueprogram = false;
    public  int queuenumber = 0;
    private  String payment;
    public  int payCash;
    public  int totalChange;
    private boolean updateCashRegister;
    private  boolean creditCardValidation;
    public  String creditCardInfo;
    private  String creditCardPin;
    public String [] inspectionPart = new String [7];
    public  String [] inspect = new String [7];
    public  boolean [] inspectionResults = new boolean[inspect.length];
    private Controller MyController = new Controller();
    Scanner s = new Scanner(System.in);

    /**
     * Runs entire program.
     * Prints out to user and takes inputs to then sent inputs to controller.
     * @throws WrongLicenseNumberException
     */
```

Figur 2.2: En bild på klassen View som implementerar Observer.



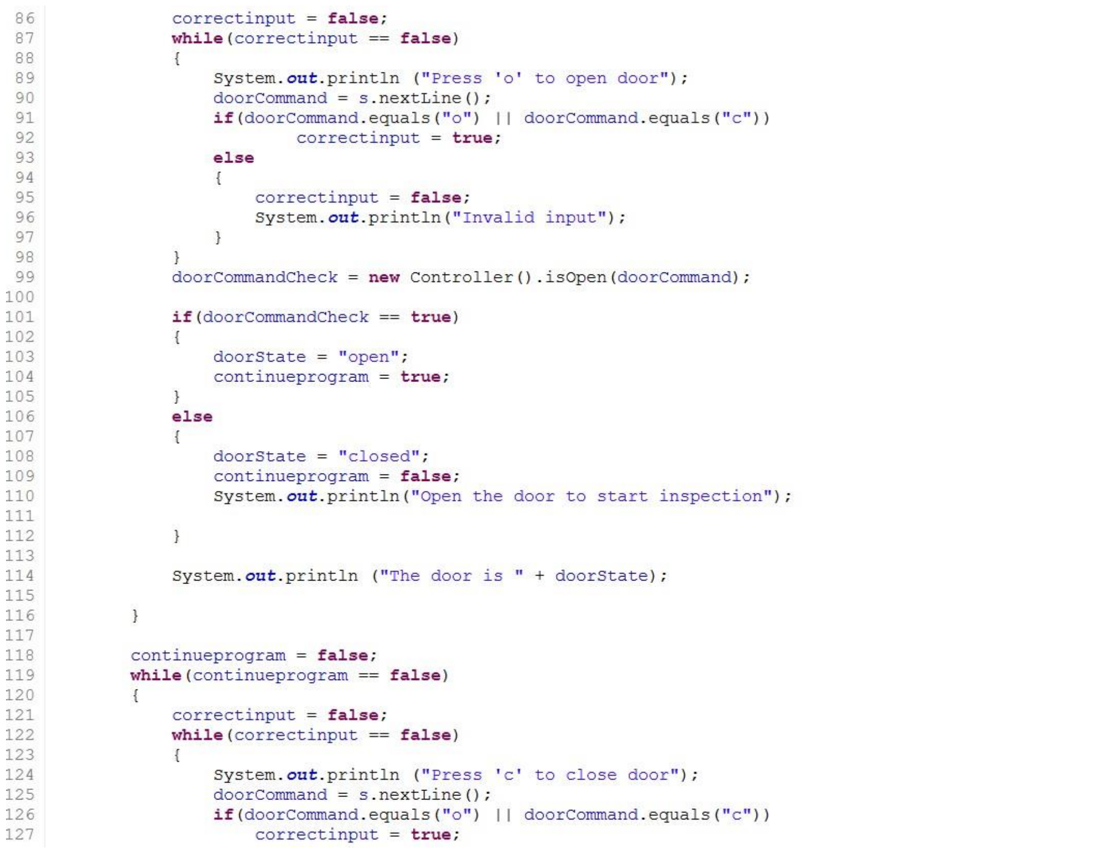
```
    public void view() {
    new Controller().setObserver(this);

        while(continueprogram == false)
        {

            while(correctinput == false)
            {
                System.out.println("Start new inspection? (y/n)");

                ready = s.nextLine();

                if(ready.equals("y") || ready.equals("n"))
                    correctinput = true;

                else
                {
                    correctinput = false;
                    System.out.println("Invalid input");
                }
            }

            if(ready.equals("y"))
            {   isReady = true;
                continueprogram = true;}

            else
            {   isReady = false;
                continueprogram = false;
                correctinput = false;
            }

            queuenumber =  new Controller().startNewInspection(isReady);

        }

        System.out.println("Current number is " + queuenumber);

        continueprogram = false;

        while(continueprogram == false)
        {
```

Figur 2.3: Ny markerad kod i klassen View.

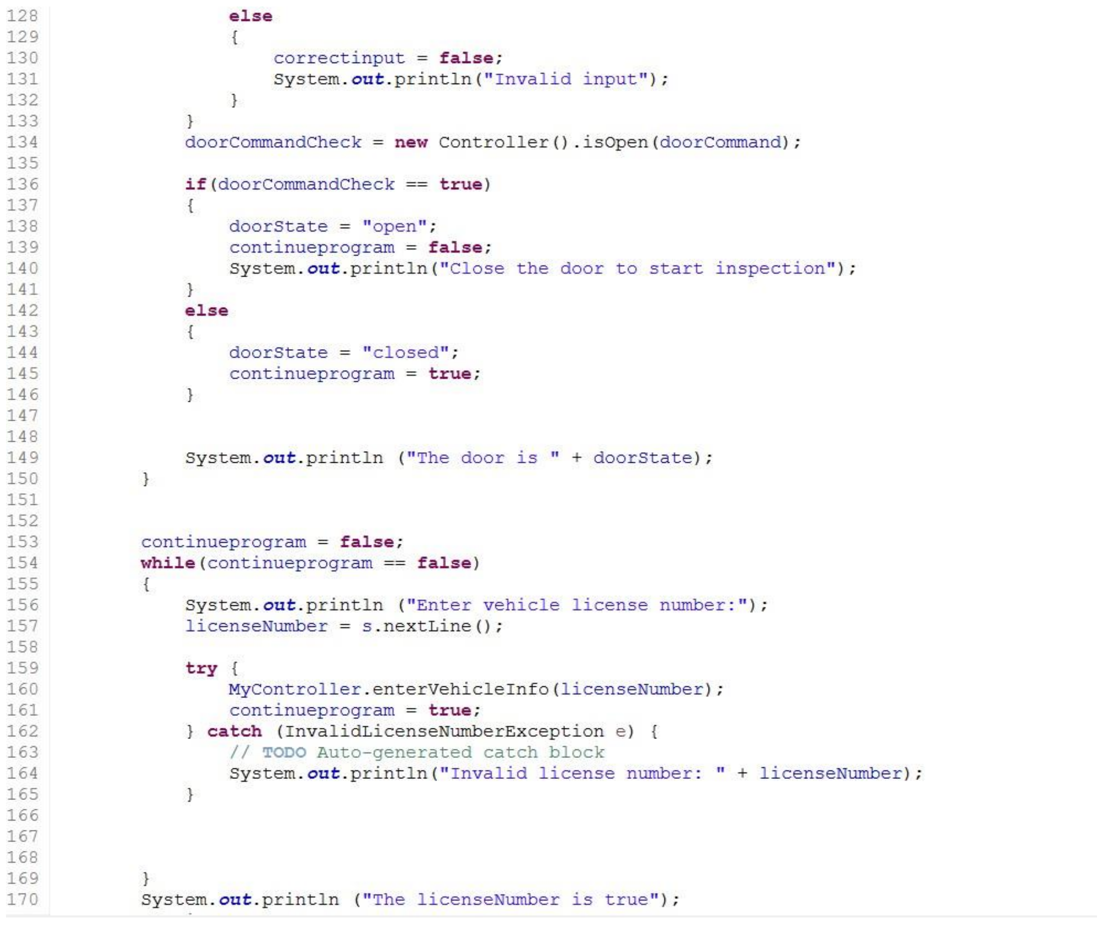


```
package model;
/**
 * Observer prints results of inspection
 *
 */
public interface Observer {
    void printResultOfInspection(String result);

}
```

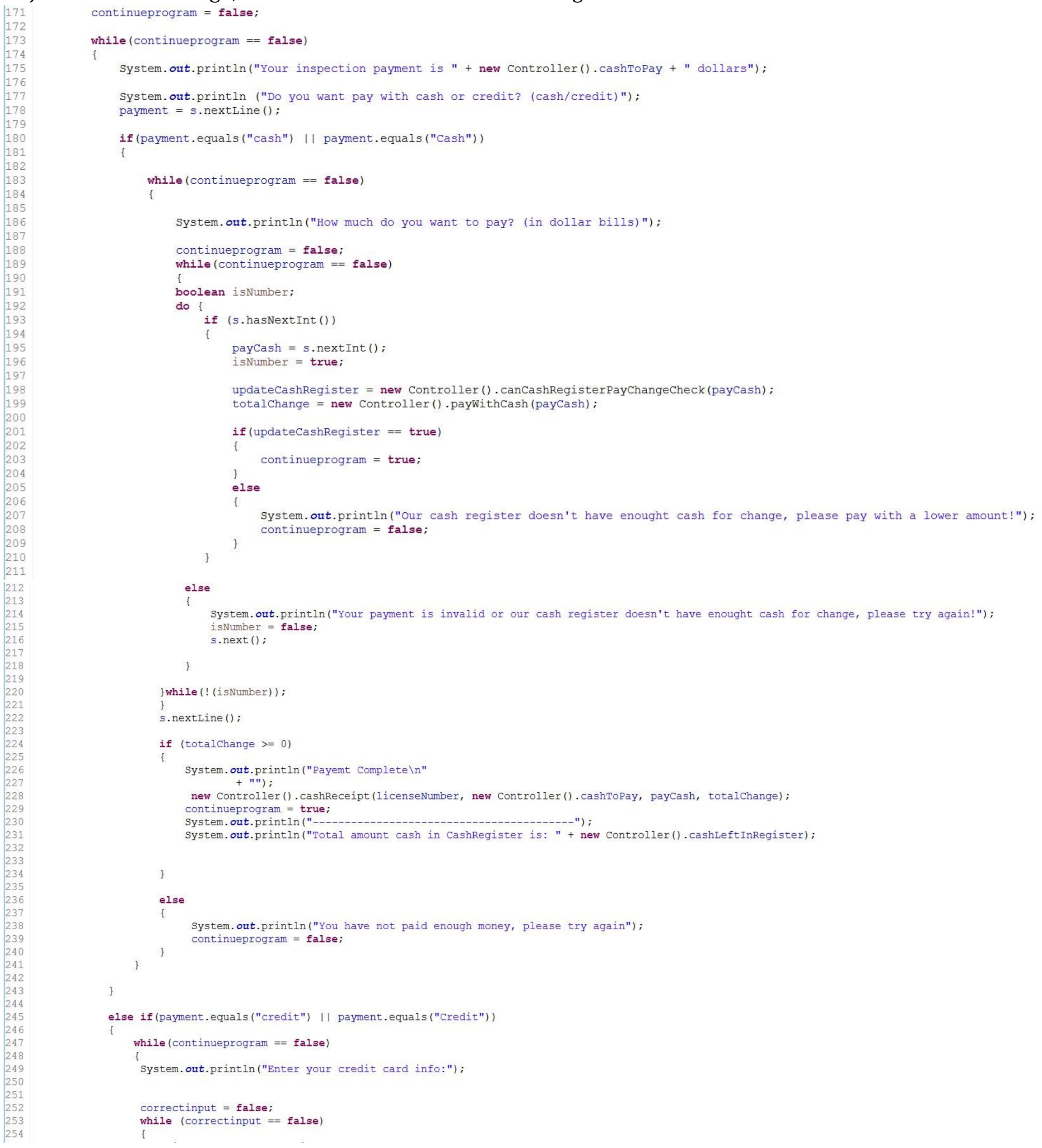
Figur 2.4: En bild på interface Observer.



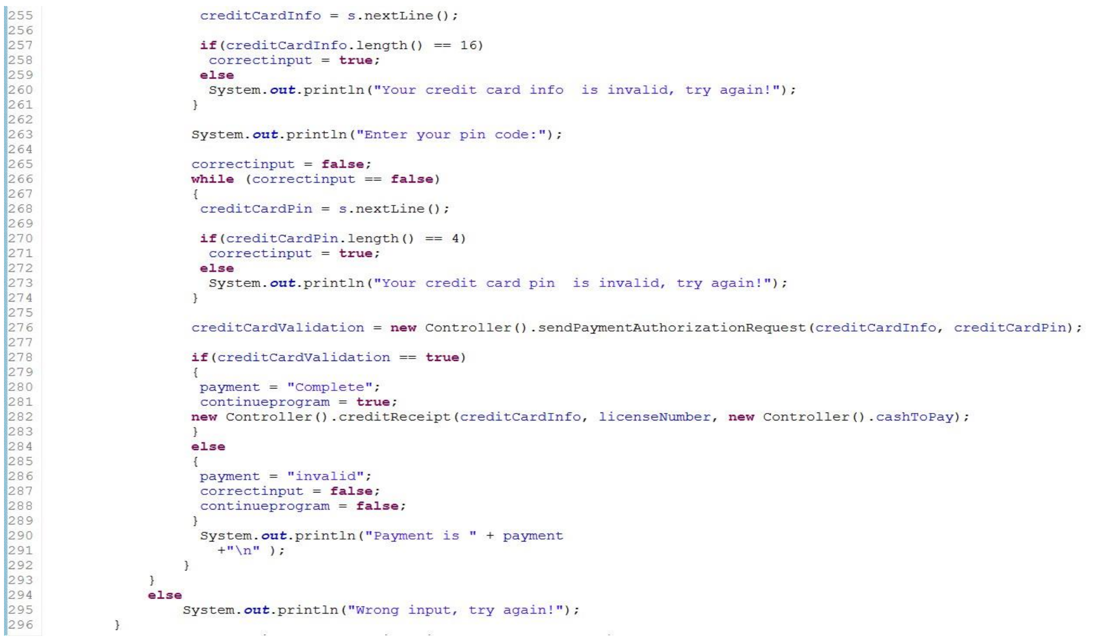
```
            correctinput = false;
            while(correctinput == false)
            {
                System.out.println ("Press 'o' to open door");
                doorCommand = s.nextLine();
                if(doorCommand.equals("o") || doorCommand.equals("c"))
                        correctinput = true;
                else
                {
                    correctinput = false;
                    System.out.println("Invalid input");
                }
            }
            doorCommandCheck = new Controller().isOpen(doorCommand);

            if(doorCommandCheck == true)
            {
                doorState = "open";
                continueprogram = true;
            }
            else
            {
                doorState = "closed";
                continueprogram = false;
                System.out.println("Open the door to start inspection");

            }

            System.out.println ("The door is " + doorState);

        }

        continueprogram = false;
        while(continueprogram == false)
        {
            correctinput = false;
            while(correctinput == false)
            {
                System.out.println ("Press 'c' to close door");
                doorCommand = s.nextLine();
                if(doorCommand.equals("o") || doorCommand.equals("c"))
                    correctinput = true;
```



```
                else
                {
                    correctinput = false;
                    System.out.println("Invalid input");
                }
            }
            doorCommandCheck = new Controller().isOpen(doorCommand);

            if(doorCommandCheck == true)
            {
                doorState = "open";
                continueprogram = false;
                System.out.println("Close the door to start inspection");
            }
            else
            {
                doorState = "closed";
                continueprogram = true;
            }


            System.out.println ("The door is " + doorState);
        }


        continueprogram = false;
        while(continueprogram == false)
        {
            System.out.println ("Enter vehicle license number:");
            licenseNumber = s.nextLine();

            try {
                MyController.enterVehicleInfo(licenseNumber);
                continueprogram = true;
            } catch (InvalidLicenseNumberException e) {
                // TODO Auto-generated catch block
                System.out.println("Invalid license number: " + licenseNumber);
            }



        }
        System.out.println ("The licenseNumber is true");
```

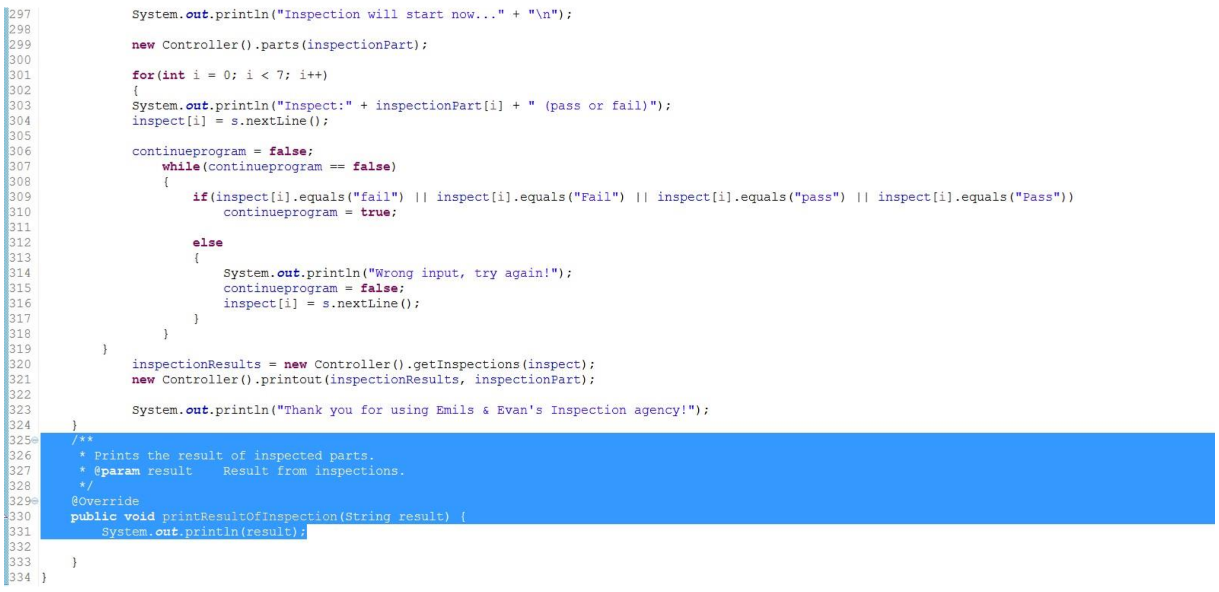


```
        continueprogram = false;

        while(continueprogram == false)
        {
            System.out.println("Your inspection payment is " + new Controller().cashToPay + " dollars");

            System.out.println ("Do you want pay with cash or credit? (cash/credit)");
            payment = s.nextLine();

            if(payment.equals("cash") || payment.equals("Cash"))
            {

                while(continueprogram == false)
                {

                    System.out.println("How much do you want to pay? (in dollar bills)");

                    continueprogram = false;
                    while(continueprogram == false)
                    {
                    boolean isNumber;
                    do {
                        if (s.hasNextInt())
                        {
                            payCash = s.nextInt();
                            isNumber = true;

                            updateCashRegister = new Controller().canCashRegisterPayChangeCheck(payCash);
                            totalChange = new Controller().payWithCash(payCash);

                            if(updateCashRegister == true)
                            {
                                continueprogram = true;
                            }
                            else
                            {
                                System.out.println("Our cash register doesn't have enought cash for change, please pay with a lower amount!");
                                continueprogram = false;
                            }
                        }

                        else
                        {
                            System.out.println("Your payment is invalid or our cash register doesn't have enought cash for change, please try again!");
                            isNumber = false;
                            s.next();

                        }

                    }while(!(isNumber));
                    }
                    s.nextLine();

                    if (totalChange >= 0)
                    {
                        System.out.println("Payemt Complete\n"
                                + "");
                         new Controller().cashReceipt(licenseNumber, new Controller().cashToPay, payCash, totalChange);
                        continueprogram = true;
                        System.out.println("----------------------------------------");
                        System.out.println("Total amount cash in CashRegister is: " + new Controller().cashLeftInRegister);


                    }

                    else
                    {
                         System.out.println("You have not paid enough money, please try again");
                         continueprogram = false;
                    }
                }

            }

            else if(payment.equals("credit") || payment.equals("Credit"))
            {
                while(continueprogram == false)
                {
                 System.out.println("Enter your credit card info:");


                 correctinput = false;
                 while (correctinput == false)
                 {
```



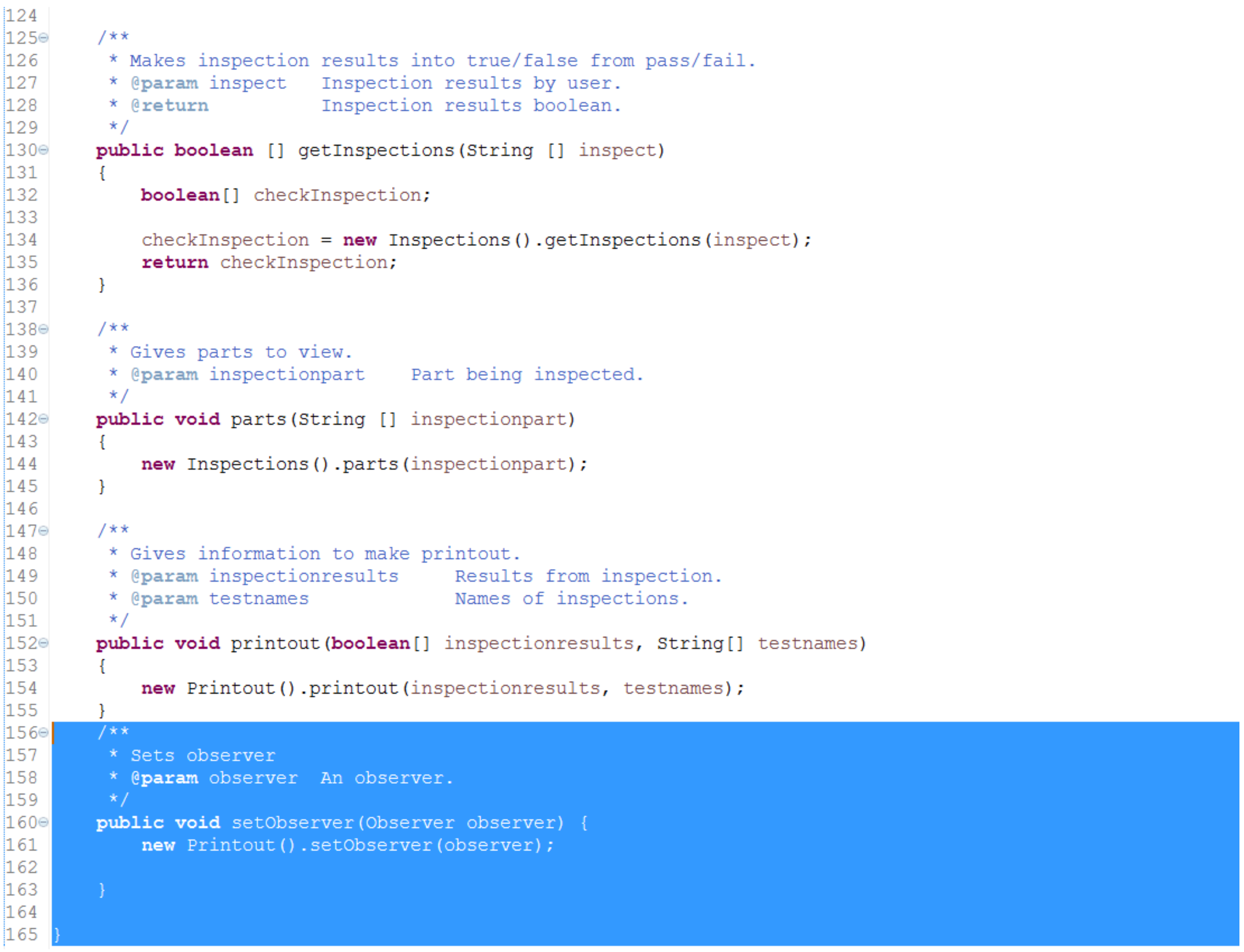
```
                  creditCardInfo = s.nextLine();

                  if(creditCardInfo.length() == 16)
                   correctinput = true;
                  else
                   System.out.println("Your credit card info  is invalid, try again!");
                 }

                 System.out.println("Enter your pin code:");

                 correctinput = false;
                 while (correctinput == false)
                 {
                  creditCardPin = s.nextLine();

                  if(creditCardPin.length() == 4)
                   correctinput = true;
                  else
                   System.out.println("Your credit card pin  is invalid, try again!");
                 }

                 creditCardValidation = new Controller().sendPaymentAuthorizationRequest(creditCardInfo, creditCardPin);

                 if(creditCardValidation == true)
                 {
                  payment = "Complete";
                  continueprogram = true;
                 new Controller().creditReceipt(creditCardInfo, licenseNumber, new Controller().cashToPay);
                 }
                 else
                 {
                  payment = "invalid";
                  correctinput = false;
                  continueprogram = false;
                 }
                  System.out.println("Payment is " + payment
                    +"\n" );
                 }
             }
             else
                 System.out.println("Wrong input, try again!");
         }
```

Figur 2.5: Bilder på oförändrade koden i klassen View.



```java
            System.out.println("Inspection will start now..." + "\n");

            new Controller().parts(inspectionPart);

            for(int i = 0; i < 7; i++)
            {
            System.out.println("Inspect:" + inspectionPart[i] + " (pass or fail)");
            inspect[i] = s.nextLine();

            continueprogram = false;
                while(continueprogram == false)
                {
                    if(inspect[i].equals("fail") || inspect[i].equals("Fail") || inspect[i].equals("pass") || inspect[i].equals("Pass"))
                        continueprogram = true;

                    else
                    {
                        System.out.println("Wrong input, try again!");
                        continueprogram = false;
                        inspect[i] = s.nextLine();
                    }
                }
        }
            inspectionResults = new Controller().getInspections(inspect);
            new Controller().printout(inspectionResults, inspectionPart);

            System.out.println("Thank you for using Emils & Evan's Inspection agency!");
    }
    /**
     * Prints the result of inspected parts.
     * @param result    Result from inspections.
     */
    @Override
    public void printResultOfInspection(String result) {
        System.out.println(result);

    }
}
```

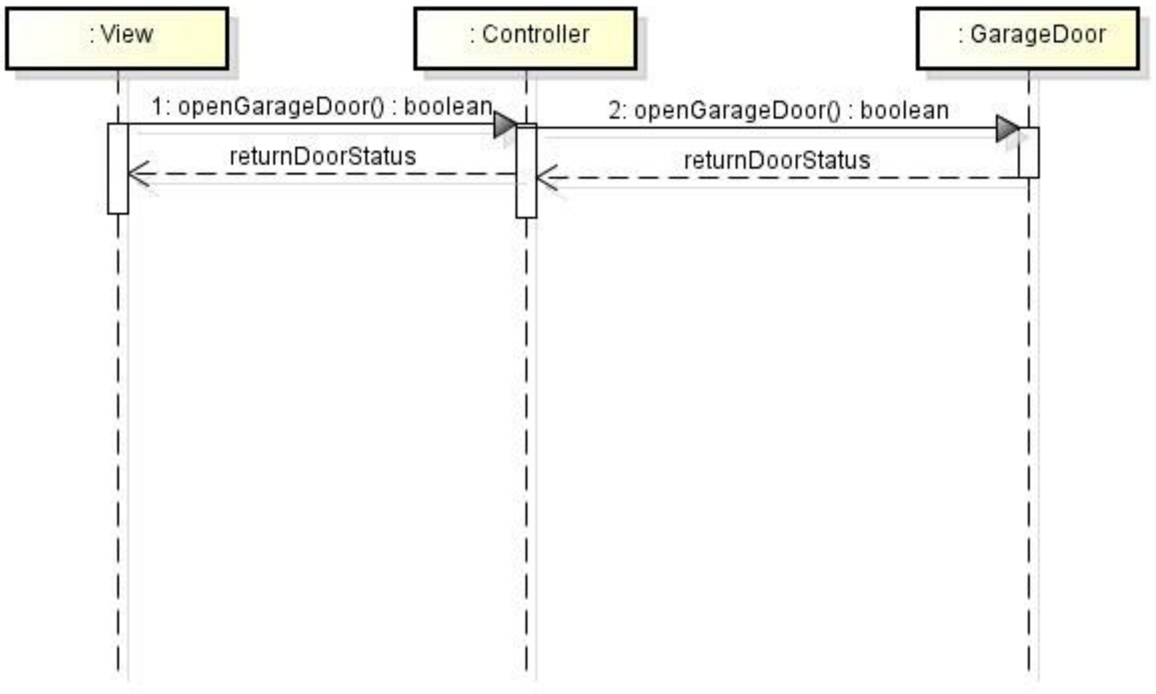
Figur 2.6: Ny Observer relaterad kod i klassen View för utskrivning av inspektionsresultat.



```java

    /**
     * Makes inspection results into true/false from pass/fail.
     * @param inspect   Inspection results by user.
     * @return          Inspection results boolean.
     */
    public boolean [] getInspections(String [] inspect)
    {
        boolean[] checkInspection;

        checkInspection = new Inspections().getInspections(inspect);
        return checkInspection;
    }

    /**
     * Gives parts to view.
     * @param inspectionpart    Part being inspected.
     */
    public void parts(String [] inspectionpart)
    {
        new Inspections().parts(inspectionpart);
    }

    /**
     * Gives information to make printout.
     * @param inspectionresults     Results from inspection.
     * @param testnames             Names of inspections.
     */
    public void printout(boolean[] inspectionresults, String[] testnames)
    {
        new Printout().printout(inspectionresults, testnames);
    }
    /**
     * Sets observer
     * @param observer   An observer.
     */
    public void setObserver(Observer observer) {
        new Printout().setObserver(observer);


    }

}
```

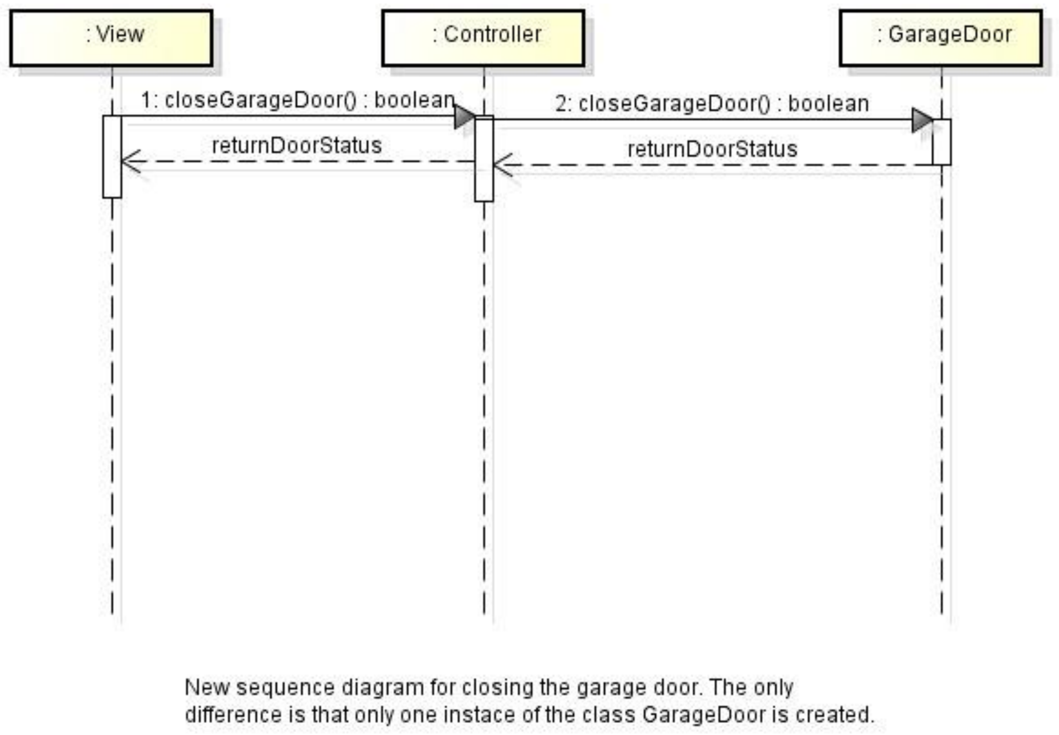
Figur 2.7: Ny kod i klassen Controller för setObserver.

Deluppgift b:

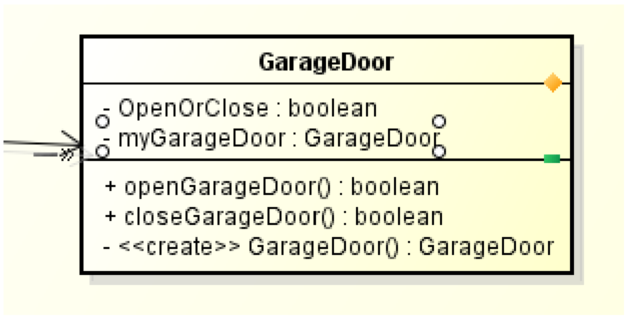


New sequence diagram for opening the garage door. The only difference is that only one instace of the class GarageDoor is created.

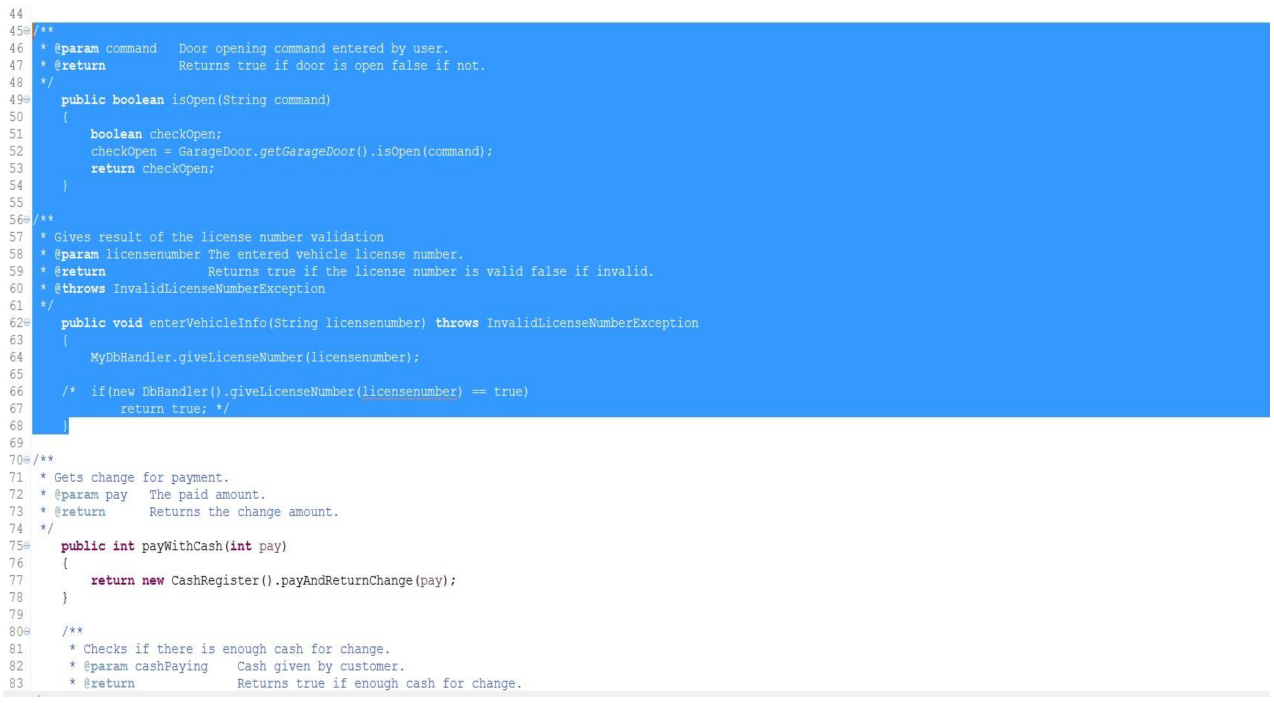
Figur 3.0: Ny sekvensdiagram på openGarageDoor.


: View
: Controller
: GarageDoor
1: closeGarageDoor() : boolean
2: closeGarageDoor() : boolean
returnDoorStatus
returnDoorStatus
New sequence diagram for closing the garage door. The only difference is that only one instace of the class GarageDoor is created.

Figur 3.1: Ny sekvensdiagram på closeGarageDoor.

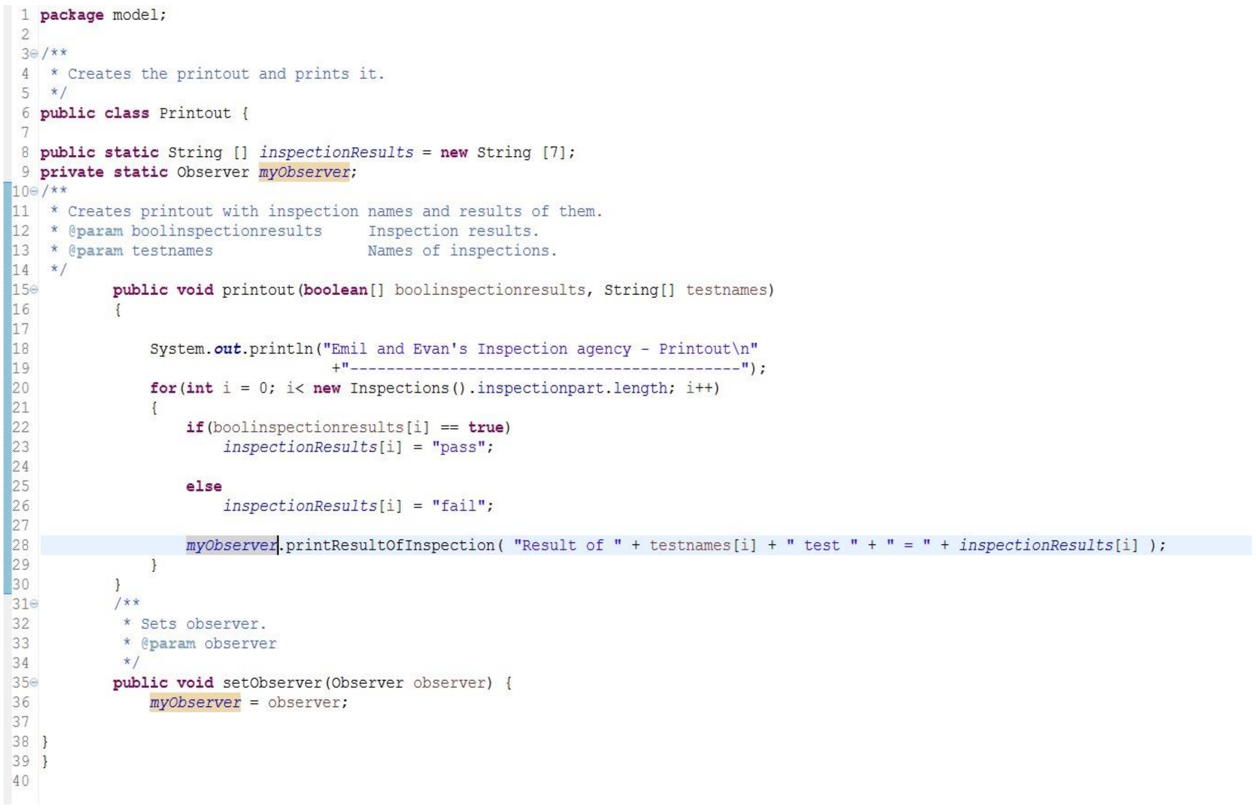

GarageDoor
- OpenOrClose : boolean
- myGarageDoor : GarageDoor
+ openGarageDoor() : boolean
+ closeGarageDoor() : boolean
- <<create>> GarageDoor() : GarageDoor

Figur 3.2: En bild på den Uppdaterade klassen GarageDoor taget från klassdiagrammet.



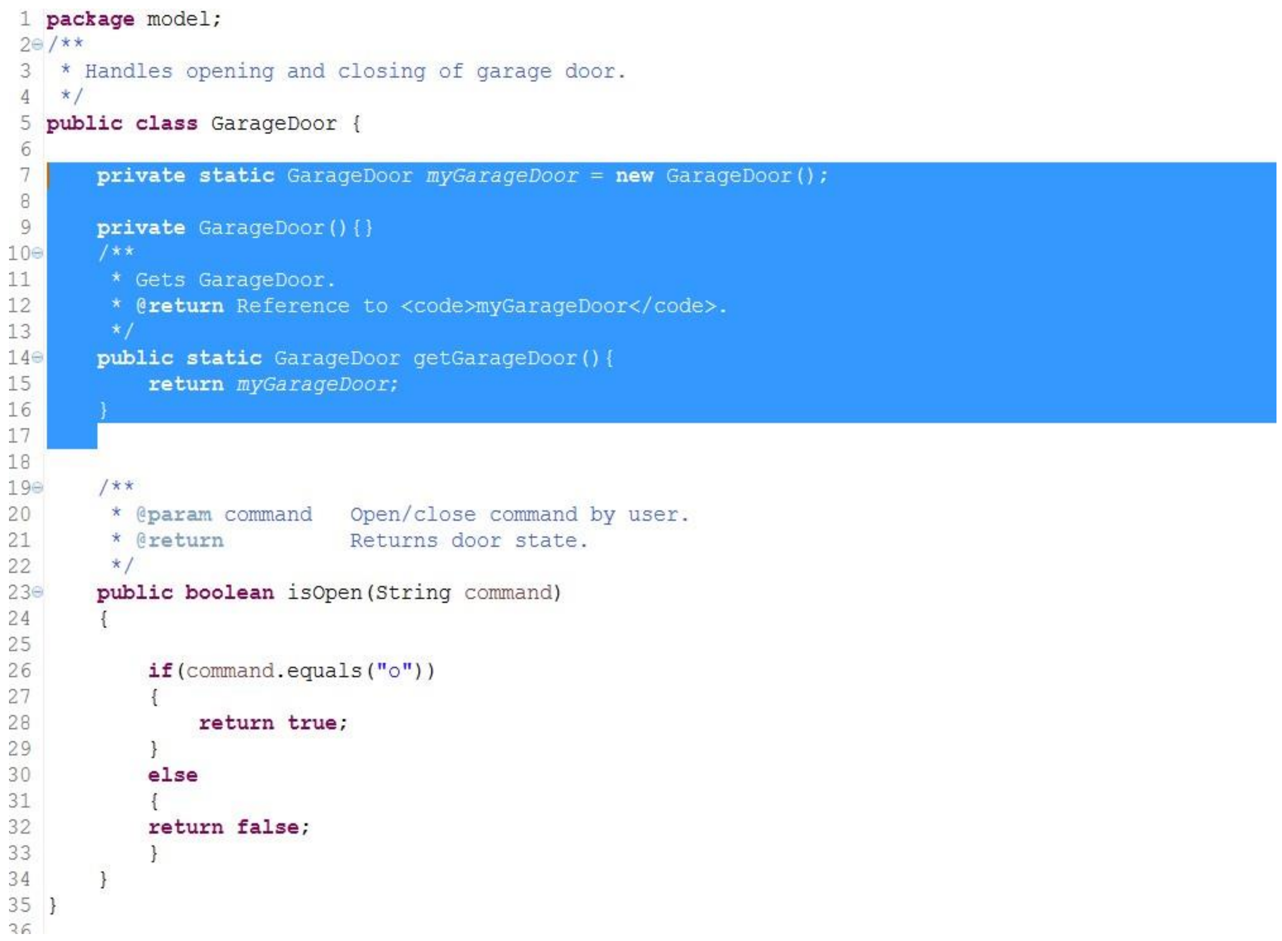
```java

/**
 * @param command   Door opening command entered by user.
 * @return          Returns true if door is open false if not.
 */
    public boolean isOpen(String command)
    {
        boolean checkOpen;
        checkOpen = GarageDoor.getGarageDoor().isOpen(command);
        return checkOpen;
    }

/**
 * Gives result of the license number validation
 * @param licensenumber The entered vehicle license number.
 * @return              Returns true if the license number is valid false if invalid.
 * @throws InvalidLicenseNumberException
 */
    public void enterVehicleInfo(String licensenumber) throws InvalidLicenseNumberException
    {
        MyDbHandler.giveLicenseNumber(licensenumber);

    /*  if(new DbHandler().giveLicenseNumber(licensenumber) == true)
            return true; */
    }

/**
 * Gets change for payment.
 * @param pay   The paid amount.
 * @return      Returns the change amount.
 */
    public int payWithCash(int pay)
    {
        return new CashRegister().payAndReturnChange(pay);
    }

    /**
     * Checks if there is enough cash for change.
     * @param cashPaying    Cash given by customer.
     * @return              Returns true if enough cash for change.
```

Figur 3.3: Ny kod i klassen Controller som använder ”GarageDoor.getGarageDoor().isOpen()”.



```java
package model;

/**
 * Creates the printout and prints it.
 */
public class Printout {

public static String [] inspectionResults = new String [7];
private static Observer myObserver;
/**
 * Creates printout with inspection names and results of them.
 * @param boolinspectionresults     Inspection results.
 * @param testnames                 Names of inspections.
 */
        public void printout(boolean[] boolinspectionresults, String[] testnames)
        {

            System.out.println("Emil and Evan's Inspection agency - Printout\n"
                        +"-------------------------------------------");
            for(int i = 0; i< new Inspections().inspectionpart.length; i++)
            {
                if(boolinspectionresults[i] == true)
                    inspectionResults[i] = "pass";

                else
                    inspectionResults[i] = "fail";

                myObserver.printResultOfInspection( "Result of " + testnames[i] + " test " + " = " + inspectionResults[i] );
            }
        }
        /**
         * Sets observer.
         * @param observer
         */
        public void setObserver(Observer observer) {
            myObserver = observer;

}
}

```

Figur 3.4: En bild av klassen Printout som nu använder observer.



```java
package model;
/**
 * Handles opening and closing of garage door.
 */
public class GarageDoor {

    private static GarageDoor myGarageDoor = new GarageDoor();

    private GarageDoor(){}
    /**
     * Gets GarageDoor.
     * @return Reference to <code>myGarageDoor</code>.
     */
    public static GarageDoor getGarageDoor(){
        return myGarageDoor;
    }


    /**
     * @param command   Open/close command by user.
     * @return          Returns door state.
     */
    public boolean isOpen(String command)
    {

        if(command.equals("o"))
        {
            return true;
        }
        else
        {
        return false;
        }
    }
}
```

Figur 3.5: Uppdaterad kod i klassen GarageDoor som följer Singelton mönster.