# Seminar 4
# Object-Oriented Design, IV1350

**Maher Jabbar**
**maherj@kth.se**

**2016-05-23**

# Contents

# 1 Introduction

Det finns två uppgifter för labbet. Uppgift 1: Vi kommer att använda undantag för att hantera ogiltiga indata i vårt program. Undantag kommer att användas om ett licensnummer som inte finns i databasen kommer att användas. Uppgift 2: är att genomföra en eller två GoF mönster i vårt program. Det första mönstret som vi måste genomföra är observatören mönster.

# 2 Method

För detta labb Jag har använt Netbeans som min programmering plattform och Astah för min klass och interaktion-diagram. Jag har skapat en klass som heter LicenseNumberRegistry som innehåller de giltiga licensnummer, Så om ett licensnummer inte finns med då kommer metoden som jag har skapat så kommer den kasta en undantag till kontrollen vilket kastar undantag i sin tur till vyn. För Observer pattern delen så har jag gjort ett interface som heter Observer i förpackningen Registry. Den andra patterna som genomfördes var Singelton mönstret.

# 3 Result

Figur 3.1 visar koden för den nya undantag som kommer att kastas om licens numret inte finns i registret

figure 3.1

```java
package vehicleinsp.registry;

public class InvalidLicenseNumberException extends Exception{
   /**
    * Creates a new exception
    * @param invalidLicNum the license number entered
    */
    public InvalidLicenseNumberException(String invalidLicNum){
        super("This license number cannot be found: " + invalidLicNum);
    }

}
```

Figur 3.2 visar klassen LicenseNumberRegistry, metoden matchLicenseNumber tar ett licensnummer som argument och kontrollerar om licensenumber är i registret.

Figure 3.2

```java
package vehicleinsp.registry;
import java.util.ArrayList;
import java.util.List;

public class LicenseNumberRegistry {
    private List<String> licenseNumbers = new ArrayList<String>();

    public LicenseNumberRegistry(){
        licenseNumbers.add("abc123");
        licenseNumbers.add("abb123");
        licenseNumbers.add("acc123");
        licenseNumbers.add("aac123");
        licenseNumbers.add("aaa123");

    }

    private boolean match(String licenseNumber){
        int len = licenseNumbers.size();
        for (int i = 0; i < len; i++){
            if(licenseNumbers.get(i).equals(licenseNumber)){
                return true;
            }
        }
        return false;
    /**
     * Takes a license number and checks if its in the registry
     * @param licenseNumber
     * @throws InvalidLicenseNumberException
     */
    public void matchLicenseNumber(String licenseNumber)throws InvalidLicenseNumberException{
        boolean match = match(licenseNumber);
        if(!(match)){
            throw new InvalidLicenseNumberException(licenseNumber);
        }
    }
}
```

Figur 3.3 visar metoden getCost i Kontrollen. Den här metoden anropar metoden matchLicenseNumber i LicenseNumberRegistry om matchLicenseNumber kommer att kasta ett undantag, kommer denna metod vidarebefordra undantag där denna metod anropas från

```
48      /**
49       * Returns the cost of whole inspecction
50       * @param licenseNumber license number on vehicle that should be inspected
51       * @return an int the cost of the whole inspection to be done
52       * @throws InvalidLicenseNumberException if given license number is not found
53       * in the license number registry
54       */
55      public int getCost(String licenseNumber)throws InvalidLicenseNumberException{
56              licNumReg.matchLicenseNumber(licenseNumber);
57              int cost = inspColl.getCost(licenseNumber);
58              return cost;
59      }
60
```

Figure 3.3

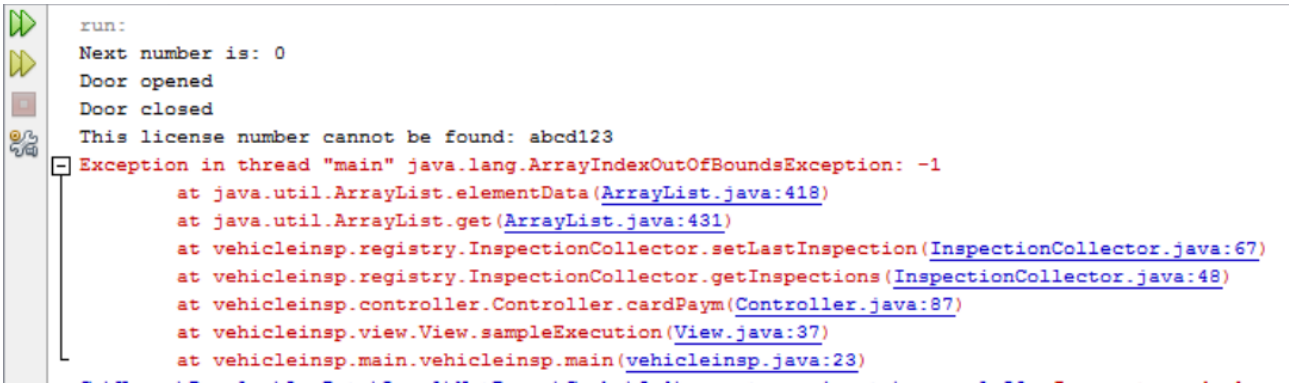
Figur 3.4
visar när vyn går in i licensnumret för att få kostnaden. Om licensenumber är i registret kommer det att finnas en utskrift för kostnaden. Om licensnumret inte var i registret undantaget kommer att nås här och meddelandet fäst vid den kommer att skrivas ut

Figure 3.4

```
22      public void sampleExecution(){
23          contr.nextInspection();
24          contr.closeDoor();
25
26          //Printout for cost of inspection
27
28          try{
29              int inspectionCost = contr.getCost(licenseNumber);
30              System.out.println("The inspection cost is: " + inspectionCost);
31          }
32          catch(InvalidLicenseNumberException invLicNumEx){
33              System.out.println(invLicNumEx.getMessage());
34          }
35
```

Figure 3.5 Visar ett Testrun när en felaktig license nummer kommer upp i View, meddelandet från undantaget visas på linje 5

figure 3.5




Figur 3.6 och 3.7 är en klass och sekvensdiagram

Figure 3.6

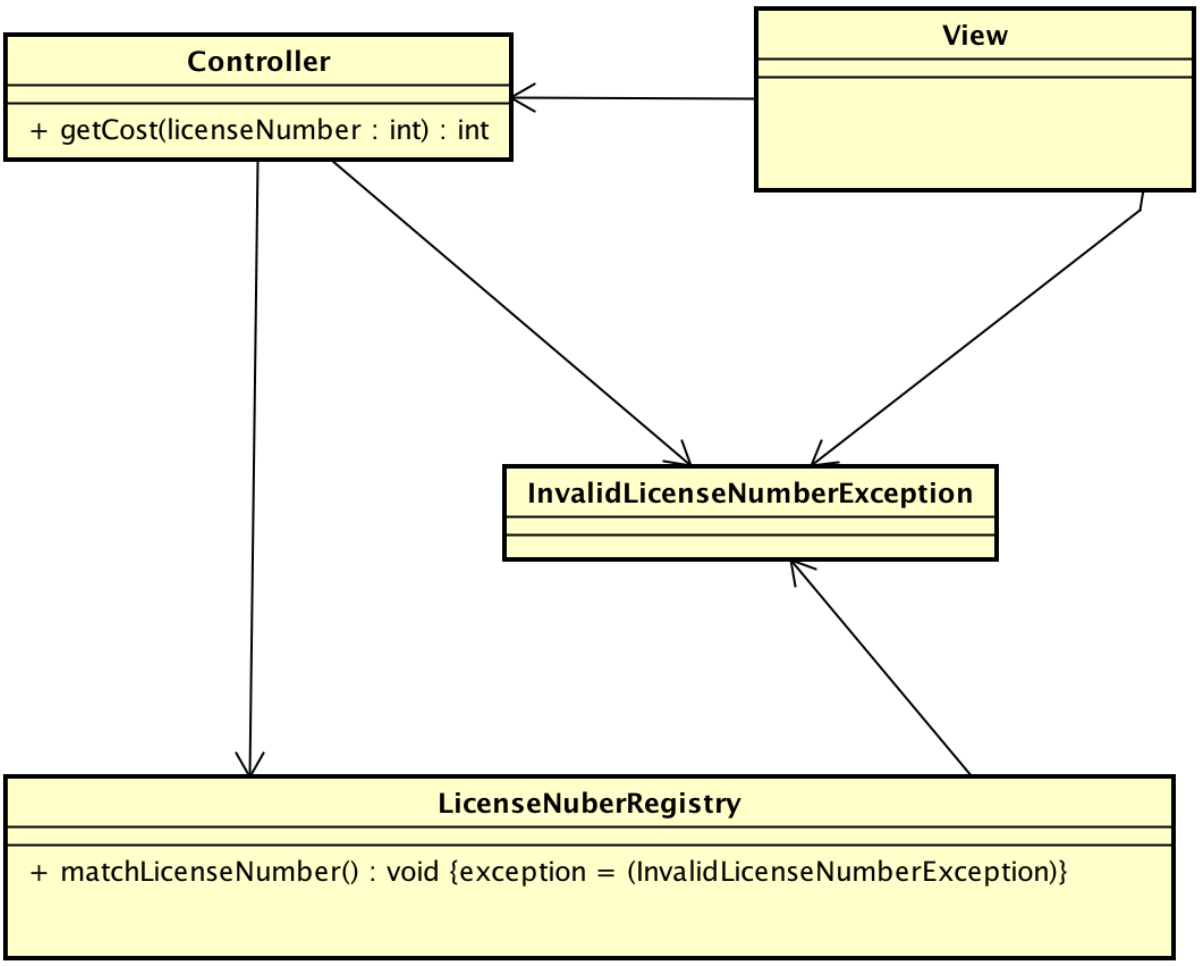
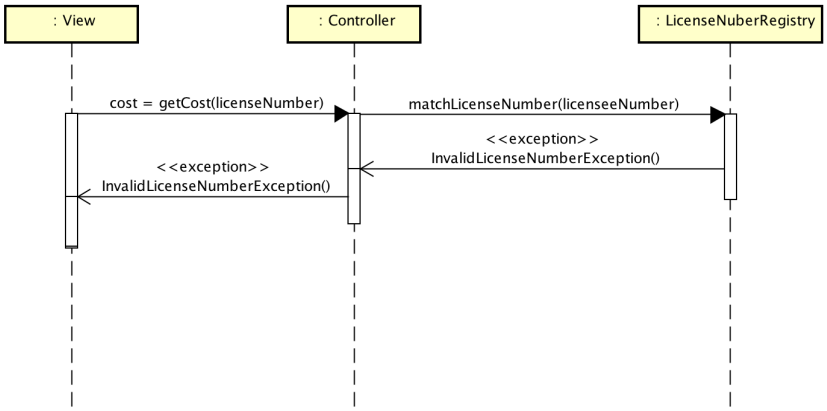


Figure 3.7




Figur 3.8 Visar Interface som genomförs av klassen InspectionStatsView visas i figur 3.10 När ett objekt i denna klass kallas genom metoden notify det kommer uppdatera sina två privata variabler misslyckats och passerade inspektioner och skriva ut dem varje gång en del av en inspektion har utförts.

Figure 3.8

```
1   package vehicleinsp.registry;
2   import vehicleinsp.model.Inspection;
3
    public interface Observer {
5       /**
6        * Called when an inspection has been performed
7        * @param inspection the inspection that has been performed
8        */
        void notify(Inspection inspection);
10  }
```

Figur 3.9 visar koden från klass InspectionCollector som ändrats sedan genomföra Observer mönstret. När en del av en inspektion har utfört resultatet på den del kommer att lagras, vilken görs i metoden storeResult.

Figure 3.9

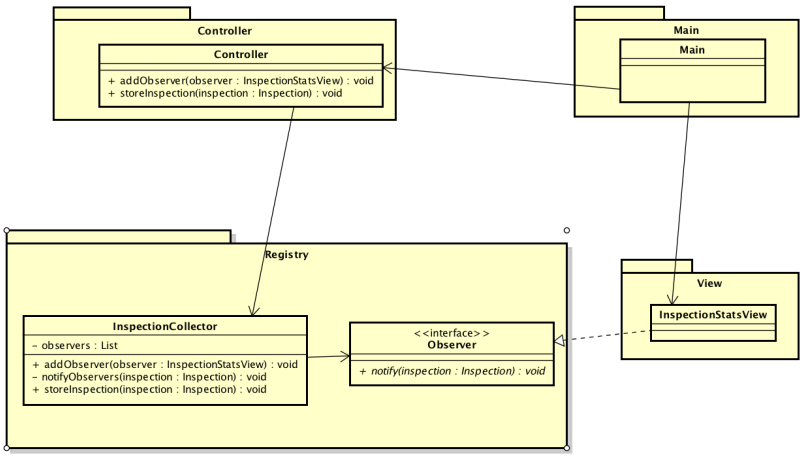
```java
/**
 * This class is responsible for handling inspections
 *
 */
public class InspectionCollector {
    private Printer printer = new Printer();
    private InspectionRegistry inspReg = new InspectionRegistry();
    private ListOfInspections inspList = new ListOfInspections();
    private List<Observer> observers = new ArrayList<Observer>();

    /**
     * creates a new instance
     */
    InspectionCollector(){

    }
    /**
     * Adds an observer to the list of observers
     * @param observer the observer that should be added
     */
    public void addObserver(Observer observer){
        observers.add(observer);
    }
   /**
    * Stores the reult of an inspection and returns the next inspection that
    * will be performed
    * @param index, index - 1 in the inspList is the inspection that the result
    * should be stored in and the Inspection with the index given is the inspection
    * that should be returned.
    * @param result the reult of performed inspection, pass/fail.
    * @return the next inspection that should be performed.
    */
   public Inspection storeResult(int index, boolean result){
       int size = inspList.getList().size() - 1;
       inspList.getList().get(index - 1).setResult(result);
       notifyObservers(inspList.getList().get(index - 1));
       if(size == index - 1){
           printResults();
           return null;
       }
       return getNextInspection(index);
   }
```

Figure 3.10

```java
package vehicleinsp.view;
import vehicleinsp.registry.Observer;
import vehicleinsp.model.Inspection;

public class InspectionStatsView implements Observer{
    private int passedInspections = 0;
    private int failedInspections = 0;

    @Override
    public void notify(Inspection inspection){
        boolean result = inspection.getResult();
        if(result){
            passedInspections++;
        }
        else{
            failedInspections++;
        }
        System.out.println("Number of passed inspections = " + passedInspections);
        System.out.println("Number of failed inspections = " + failedInspections);
    }

}
```

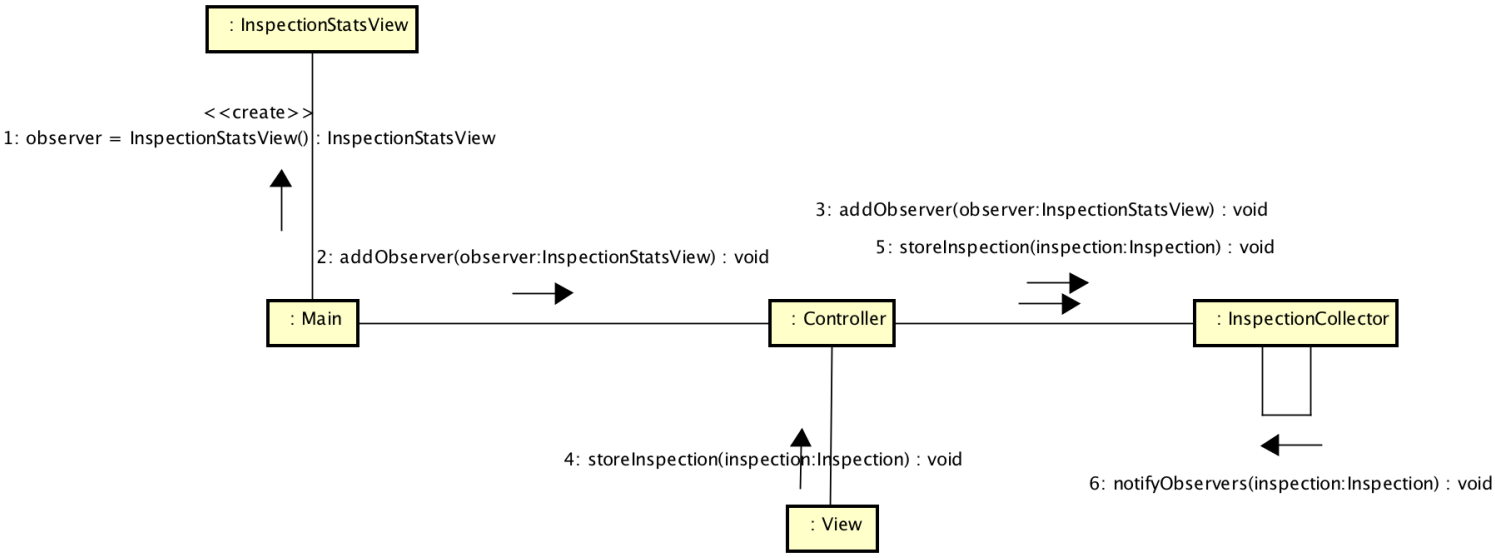
Figure 3.10

Figur 3.11 Visar klassdiagram



figur 3.12 visar ett kommunikationsschema för Observer mönstret

Figure 3.12



Figur 3.13 visar klassen Garage Vilket implementerar Singleton. Klassen har en privat konstruktör vilken den använder för att skapa en statisk variabel av ett objekt Garage.

Figure 3.13

```java
package vehicleinsp.integration;
public class Garage {
    private static Garage instance = new Garage();
    private GarageDoor door;
    private Display display;
    private Queue queue;

    private Garage(){
    door = new GarageDoor();
    display = new Display();
    queue = new Queue();
    }
    public static Garage getGarage(){
        return instance;
    }
    /**
     * This method is called when a new inspections is to be performed. It retrieves
     * the current queue number and sends it to the display wich displays the number.
     * It also opens up the garage door.
     *
     */
    public void nextInspection(){
        int number = queue.getQueueNumber();
        display.displayNummber(number);
        door.openDoor();
    }
    public void closeDoor(){
        door.closeDoor();
    }
    /**
     * Retrieves the statement of the door.
     * @return true if door is open and false if not.
     */
    public boolean getDoor(){
        return door.isOpen();
    }
}
```

Figur 3.14 och 3.15 visar klass och interaktion diagram för genomförandet av Singleton.

Figure 3.14

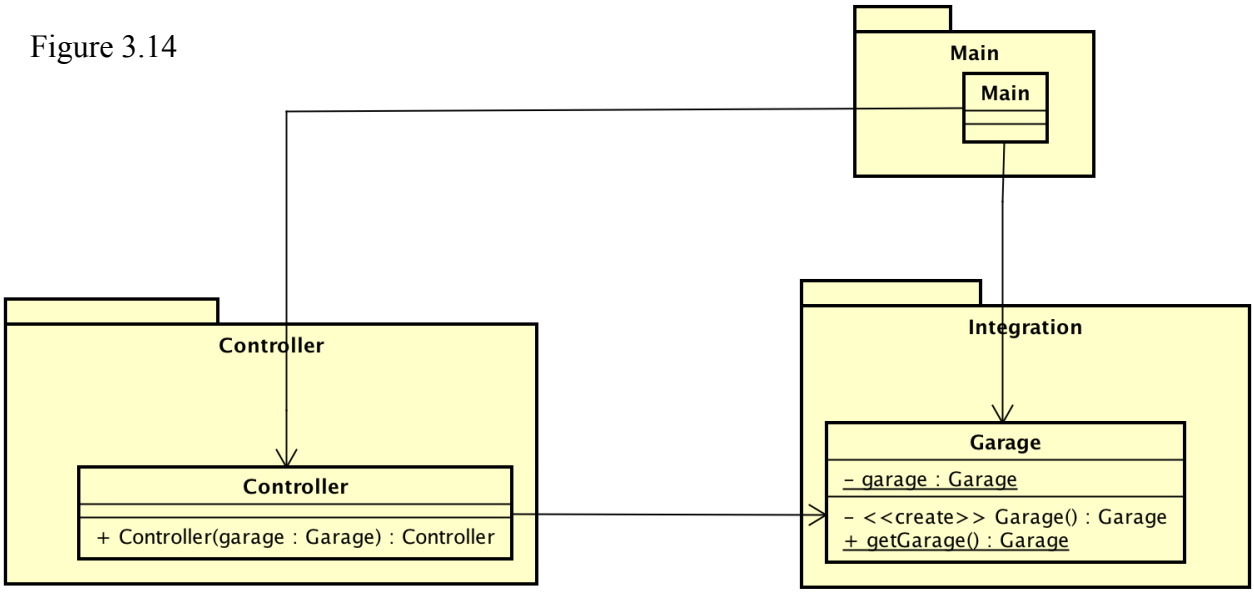
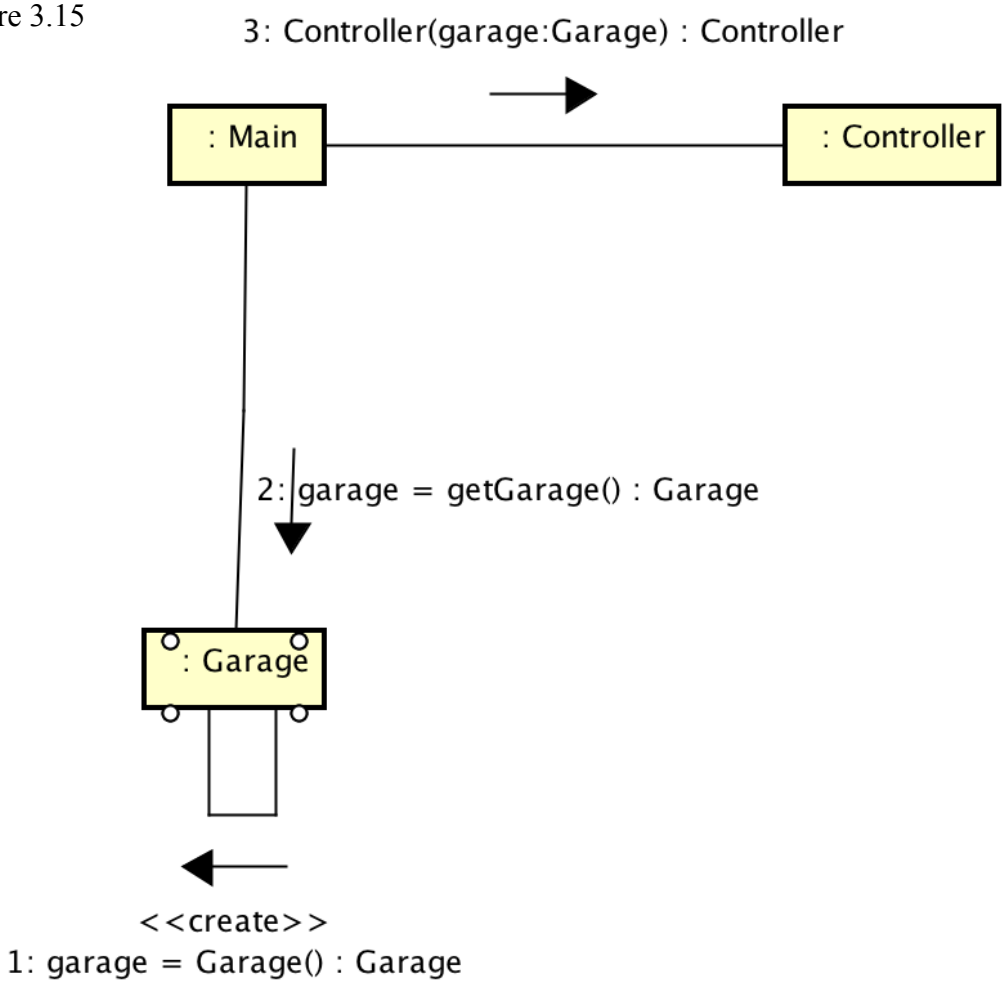



Figure 3.15



# 4 Discussion

Jag valde att göra ett kontrollerat undantaget som skapat av LicenseNumberRegistry så när vyn skriver ett licens nummer som inte finns med i listan alltså i registret då direkt Undantaget kastas till controller vilken i sin tur kastar upp till vyn där den fångas. Genomförande av Observer mönstret görs genom att lägga till en förteckning över de observatörer till klassen InspectionCollector. När en del av inspektion gjort InspectionCollector kommer att kallas, när detta är gjort det finns en privat metod i inspectionCollector som meddelar alla dess observatörer, vilket implementerar ett gränssnitt som finnas i samma paket som InspectionCollector. Genomförande av Singleton gjordes på klass Garage eftersom det bara ska vara ett objekt av klassen. Klassen Garage efter genomförandet innehåller en statisk hänvisning till ett Garage objekt och dess konstruktör är en privat metod. Eftersom inga andra klasser kan komma till konstruktören det endast att finnas en instans av objektet.