TDA 545: Objektorienterad programmering

Föreläsning 7:
Att skriva egna klasser

Magnus Myréen

Chalmers, läsperiod 1, 2015-2016

Idag

Läsanvisning: kap 2; för nästa gång: kap 10 och 15

- att konstruera en klass
- Javadoc
- en bilklass

Påminnelse om klasser och objekt

new skapar nya objekt, dvs nya instanser av en klass.

Hur hittar man lämpliga abstraktioner?

1. *Vad ska objekten vara?*

   Identiteten. Objekt är vanligen "saker", dvs substantiv.

2. *Vad vill du kunna göra med objektet?*

   Beteende. Instansmetoderna, dvs vad som kan göras med objektet (verb!)

3. *Vilka egenskaper skall/måste lagras?*

   Tillstånd. I instansvariabler lagras egenskaper hos objektet (adjektiv!).

En klass för en bil

1. *Vad ska objekten vara?*

   I detta fall en ***bil***.

2. *Vad vill du kunna göra med objektet?*

   - tilldela bilnummer (en gång)
   - skriva/rita ut det
   - bestäm färg?
   - sätt hastighet? öka/minsta hastighet...
   - sätt startposition?
   - flytta absolut och relativt?

3. *Vilka egenskaper/tillstånd måste lagras?*

   - bilnummer
   - färg
   - hastighet

Vilka egenskaper skall kunna ändras?

Hur ser då gränssnittet ut?

En klass för en bil

Klassens namn:	`Car`
Tillstånd, instansvariabler:	
bilnummer	`carNumber`
märke och modell	`brand`
hastighet, körda km	`speed, distance`
Beteende, instansmetoder:	
kör framåt/bakåt	`drive()`
sätt hastighet till x	`setSpeed(…)`
sätt färg	`setColour(…)`
tanka	`refuel(…)`

Skiss

```
import java.awt.Color;
public class Car {
    // constructors
    public Car () {
    }
    public Car (String carNumber) {
    }
    // mutators ("setters")
    public void setSpeed (int x) {
    }
    public void setColor (Color newColor) {
    }
    // frågor ("getters")
    public Color getColor() {
    }
    public int getSpeed() {
    }
    public int getX () {
    }
    // other
    public void refuel (double liters) {
    }
}
```

Specifikation och Implementation

black-box tänkande…

En specifikation (metodsignaturer + javadoc) talar om
vad som kan göras med ett objekt

Implementationen förverkligar löftena i specifikationen.
En specifikation kan ha flera olika implementationer.

Specifikation: **Javadoc**

Java använder programmet javadoc för att skapa en beskrivning av specifikationen. Javadoc läser Java-koden och extraherar ut metodernas signatur och speciella kommentarer och "taggar".

Javadoc kommentar:

```
/**
* Text som beskriver metoden.
* olika taggar som
* @author Magnus Myreen
*/
metodens deklaration här
```

Exempel på taggar: `@author, @param, @return, @throws`

metodsignatur + javadoc = specifikation

```
metodsignatur + javadoc = specifikation
import java.awt.Color;
/**
* A class for representing a car.
* <p>(Det är skillnad på första raden
* och de övriga här
* <br>Första raden skall börja på stor
* bokstav och avslutas med punkt.)
* <br>The state includes car number,
* speed and color.
* @author Magnus Myreen
* @version 1.15126
*///HTML kod funkar i javadoc kommentarer.
public class Car {
  public Car () {
    ; // tomt ännu så länge
  }
  public Car (String carNumber) {
    ; // tomt ännu så länge
  }
  // mutators ("setters")
  /**
  * Set the speed of the car.
  * Checks the speed for reasonable
  * values i.e. -40<= speed <= 200
  * @param x the new speed
  */
  public void setSpeed (int x) { ; }
  /**
  * Set the color of the car.
  * @param newColor the new color
  */
  public void setColor (Color newColor) {
    ; // tomt ännu så länge
  }
  // accessors ("getters")
  /**
  * Kommentarerna för getters är vanligen
  * rätt torftiga....
  */
  public Color getColor() { ; }
  public double getSpeed() { ; }
  // annat
  public void refuel(double liters){
    // tomt ännu så länge
  }
  /**
  * toString är en standard metod som
  * man alltid skall skriva
  */
  public String toString() {
     // tomt ännu så länge
  }
}
```

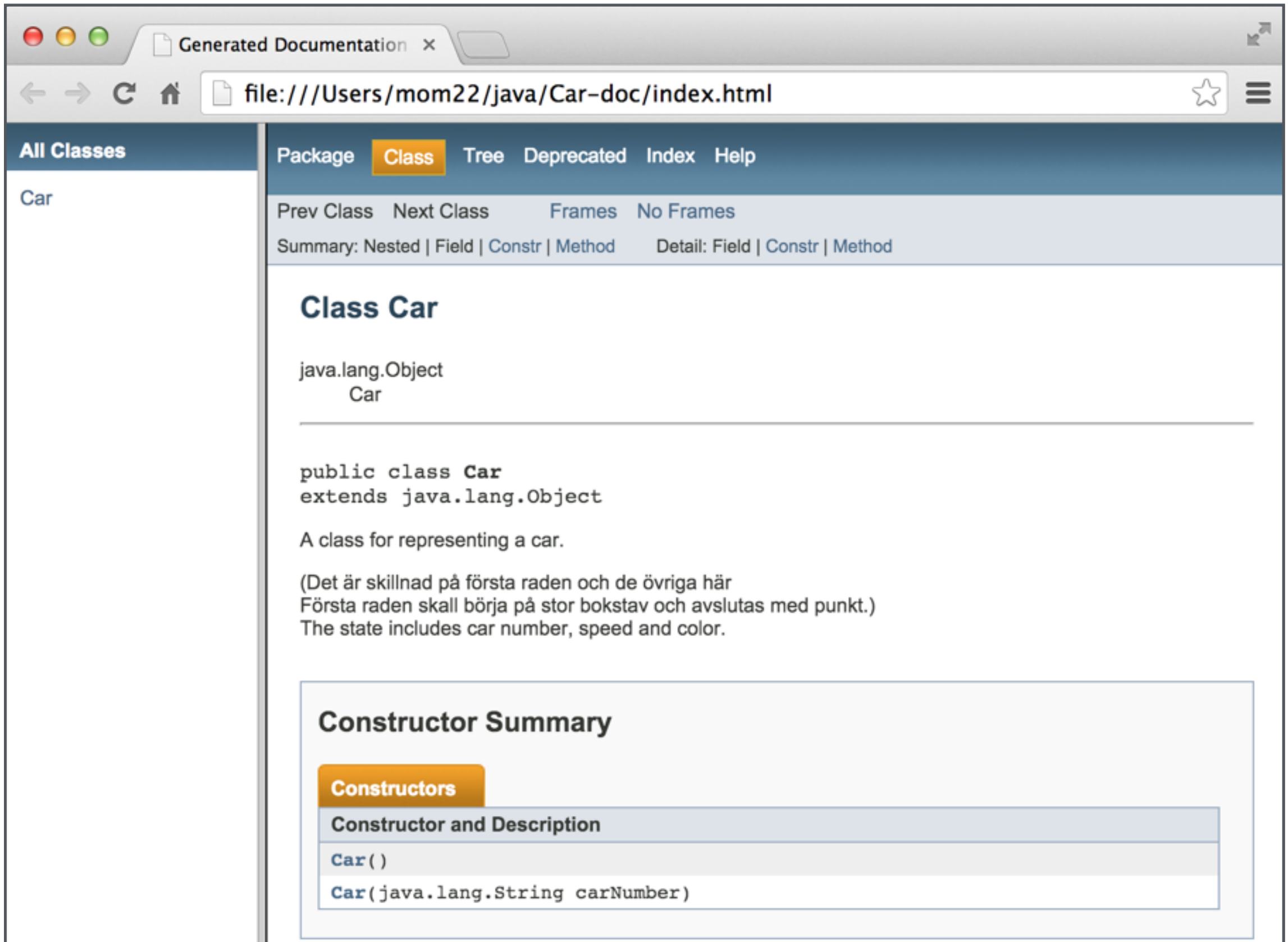
javadoc

```
$ mkdir Car-doc
$ cd Car-doc/
$ javadoc ../Car.java
Loading source file ../Car.java...
Constructing Javadoc information...
Standard Doclet version 1.7.0_67
Building tree for all the packages and classes...
Generating /Car.html...
Generating /package-frame.html...
Generating /package-summary.html...
Generating /package-tree.html...
Generating /constant-values.html...
Building index for all the packages and classes...
Generating /overview-tree.html...
Generating /index-all.html...
Generating /deprecated-list.html...
Building index for all classes...
Generating /allclasses-frame.html...
Generating /allclasses-noframe.html...
Generating /index.html...
Generating /help-doc.html...
```

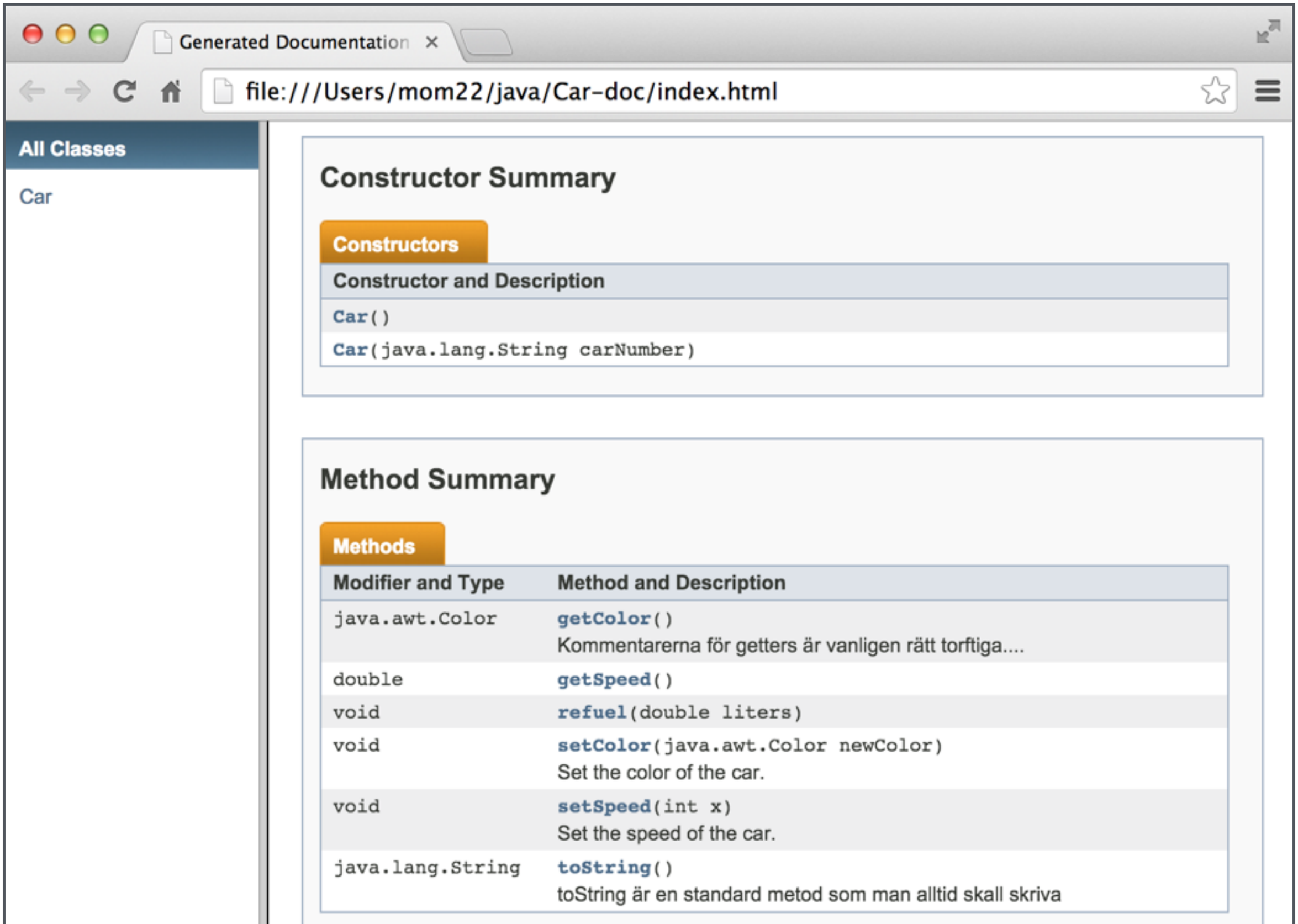
vad gjorde javadoc?

```
$ ls -la
total 160
drwxr-xr-x  17 mom22  staff    578 17 Sep 10:05 .
drwxr-xr-x  29 mom22  staff    986 17 Sep 10:05 ..
-rw-r--r--   1 mom22  staff  10297 17 Sep 10:05 Car.html
-rw-r--r--   1 mom22  staff    569 17 Sep 10:05 allclasses-frame.html
-rw-r--r--   1 mom22  staff    549 17 Sep 10:05 allclasses-noframe.html
-rw-r--r--   1 mom22  staff   3287 17 Sep 10:05 constant-values.html
-rw-r--r--   1 mom22  staff   3237 17 Sep 10:05 deprecated-list.html
-rw-r--r--   1 mom22  staff   7643 17 Sep 10:05 help-doc.html
-rw-r--r--   1 mom22  staff   5709 17 Sep 10:05 index-all.html
-rw-r--r--   1 mom22  staff   2664 17 Sep 10:05 index.html
-rw-r--r--   1 mom22  staff   3448 17 Sep 10:05 overview-tree.html
-rw-r--r--   1 mom22  staff    689 17 Sep 10:05 package-frame.html
-rw-r--r--   1 mom22  staff      1 17 Sep 10:05 package-list
-rw-r--r--   1 mom22  staff   3711 17 Sep 10:05 package-summary.html
-rw-r--r--   1 mom22  staff   3457 17 Sep 10:05 package-tree.html
drwxr-xr-x   6 mom22  staff    204 17 Sep 10:05 resources
-rw-r--r--   1 mom22  staff  11139 17 Sep 10:05 stylesheet.css
$ open index.html
```

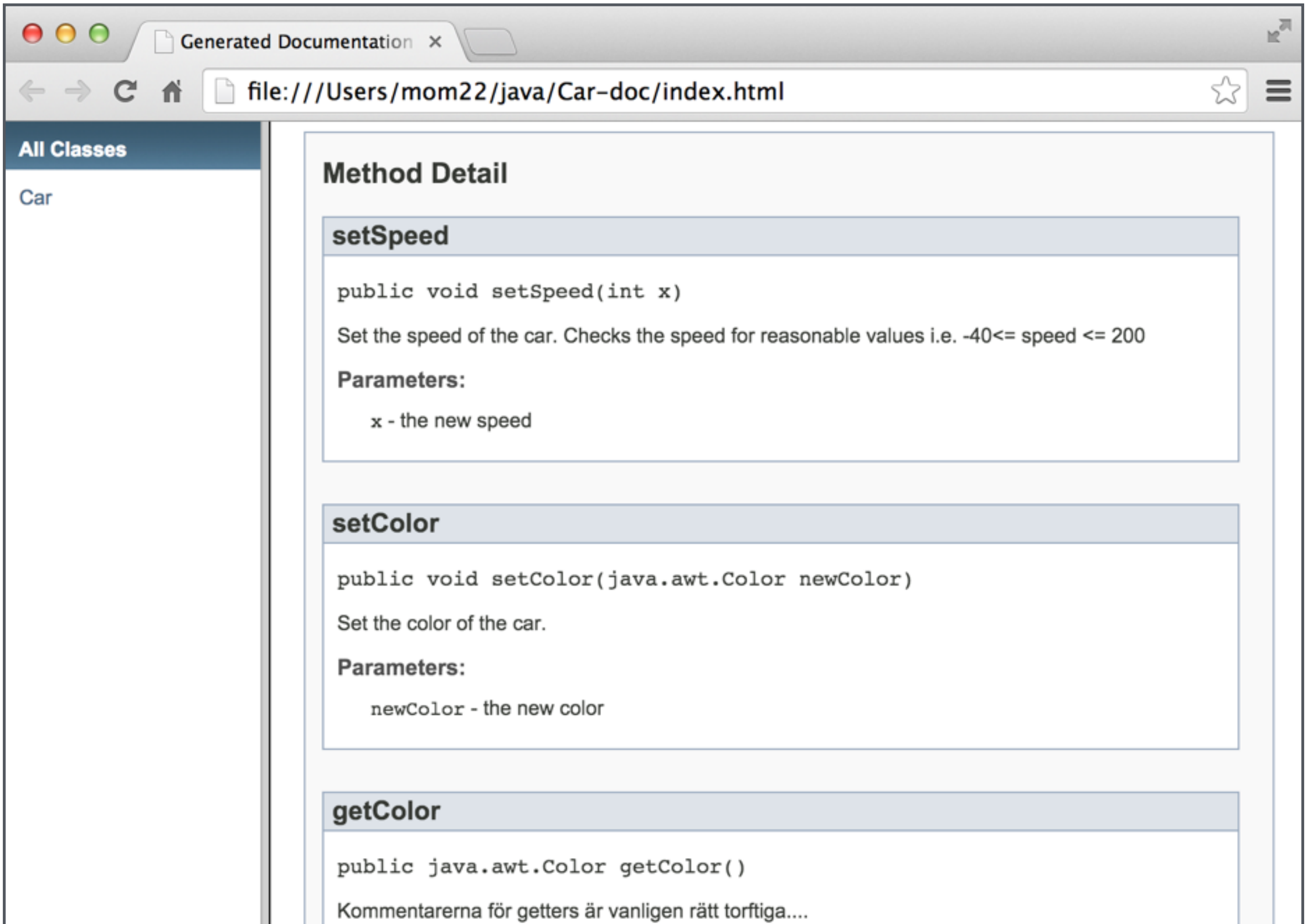
Resultatet


Generated Documentation
file:///Users/mom22/java/Car-doc/index.html
All Classes
Car
Package
Class
Tree
Deprecated
Index
Help
Prev Class
Next Class
Frames
No Frames
Summary: Nested | Field | Constr | Method
Detail: Field | Constr | Method
Class Car
java.lang.Object
Car
public class Car
extends java.lang.Object
A class for representing a car.
(Det är skillnad på första raden och de övriga här
Första raden skall börja på stor bokstav och avslutas med punkt.)
The state includes car number, speed and color.
Constructor Summary
Constructors
Constructor and Description
Car()
Car(java.lang.String carNumber)

Resultatet


Generated Documentation
file:///Users/mom22/java/Car-doc/index.html
All Classes
Car
Constructor Summary
Constructors
Constructor and Description
Car()
Car(java.lang.String carNumber)
Method Summary
Methods
Modifier and Type
Method and Description
java.awt.Color
getColor()
Kommentarerna för getters är vanligen rätt torftiga....
double
getSpeed()
void
refuel(double liters)
void
setColor(java.awt.Color newColor)
Set the color of the car.
void
setSpeed(int x)
Set the speed of the car.
java.lang.String
toString()
toString är en standard metod som man alltid skall skriva

Resultatet


Generated Documentation
file:///Users/mom22/java/Car-doc/index.html
All Classes
Car
Method Detail
setSpeed
public void setSpeed(int x)
Set the speed of the car. Checks the speed for reasonable values i.e. -40<= speed <= 200
Parameters:
x - the new speed
setColor
public void setColor(java.awt.Color newColor)
Set the color of the car.
Parameters:
newColor - the new color
getColor
public java.awt.Color getColor()
Kommentarerna för getters är vanligen rätt torftiga....

Olika javadoc taggar

@author Författare. Flera är ok.

@version Nuvarande version

@since När denna ”feature” infördes

@param Parametrarnas accepterbara värden och deras betydelse. Flera är ok.

@return Betydelse och möjliga värden på resultatet

@see Länk till annan dokumentation

@throws Ev. exception som kastas av en metod

För mer om javadoc gör kommandot “`man javadoc`”

Implementera…

black-box tänkande…

En specifikation (metodsignaturer + javadoc) talar om
vad som kan göras med ett objekt

Implementationen förverkligar löftena i specifikationen.
En specifikation kan ha flera olika implementationer.

Fyller i med kod

Koden implementerar specifikationen.

```
public class Car {
  private String carNumber;
  private Color col;
  private double speed = 0.0;
  public Car (String carNumber) {
      this.carNumber = carNumber;
  }
// mutators ("setters")
/**
* Set the speed of the car.
* Checks the speed for reasonable
* values i.e. -40<= speed <= 200
* @param x the new speed
*/
public void setSpeed (int x) {
    // TODO: check speed
    speed = x;
}
```

this hänvisar till detta objekt

Man bör kompilera och provköra när man skriver implementationen.

En testklass

Koden implementerar specifikationen.

skapar två nya bilar

```
import java.awt.Color;
public class CarTest {
    public static void main (String[] args) {
        Car car1 = new Car ("ABC 123");
        Car car2 = new Car ("YXZ 890");
        car1.setColor(Color.red);
        car1.setSpeed(14);
        car2.setColor(Color.black);
        car2.setSpeed(29);
        System.out.println("car 1 = " + car1.toString());
        System.out.println("car 2 = " + car2);
    }
}
```

Vissa saker är gemensamma för alla objekt

Klassens namn:	`Car`
Tillstånd, klassvariabler:	
adressen av trafiksäkerhetsv.	`authAddress`
Tillstånd, instansvariabler:	
bilnummer	`carNumber`
märke och modell	`brand`
hastighet, körda km	`speed, distance`
Beteende, klassmetoder:	
sätta/hämta adressen	`getAuthAddress()`
konvertera miles till km	`milesToKm(…)`
Beteende, instansmetoder:	
kör framåt/bakåt	`drive()`
sätt hastighet till x	`setSpeed(…)`
sätt färg	`setColour(…)`
tanka	`refuel(…)`

Klasser - en typ för objekten

En klass innehåller en beskrivning av (en mall för, en definition av) tillståndet (dvs variabler) och beteendet (dvs metoder) hos de objekt som skapas av klassen.

Klassen är en beskrivning av tex hur en bil ser ut och av vad man kan göra med den.

Klassvariabler: Alla bilar har samma adress till trafiksäkerhetsverket. Det är alltså en egenskap som finns hos klassen snarare än hos objektet.

Klassmetoder: anropas utan objekt (eftersom inget objekt behövs). Man anropar klassen för att få reda på TSV adressen.

Instans[variabler/metoder]: variabeln eller metoden är bunden till ett objekt (tex ändra hastighet). Man anropar ett objekt.

Varje objekt har sina egna instansvariabler / instansmetoder. Dessa beskriver ju objektets tillstånd.

static

Använda **static** för att deklarera klassvariabler/metoder.

```
public class BilClass1 {
  public static String authAddress;
  public static int numberOfCars;
  public String brand;
  public String carNumber;
  public double speed;
  public int totalKm;
  ...
} // end BilClass1
```

public…

Var försiktig med **public**!

```
public class BilClass1 {
  public static String authAddress;
  public static int numberOfCars;
  public String brand;
  public String carNumber;
  public double speed;
  public int totalKm;
  ...
} // end BilClass1
```

Alla variabler är här publika dvs kan ändras utifrån klassen…

public…

Var försiktig med **public**!

```
public class BilClass1 {
  public static String authAddress;
  public static int numberOfCars;
  public String brand;
  public String carNumber;
  public double speed;
  public int totalKm;
  ...
} // end BilClass1
```

Alla variabler är här publika dvs kan ändras utifrån klassen…

```
public class BilTest1 {
   public static void main (String[] args) {
     // skapa ett bil objekt
     BilClass1 enBil = new BilClass1();
     // ge värden
     enBil.carNumber = "ABC123";
     enBil.totalKm = 250;
     enBil.speed = 90;
     System.out.println(enBil.carNumber);
     enBil.totalKm = enBil.totalKm - 500;
     BilClass1.numberOfCars += 1;
  }
}
```

public…

Var försiktig med **public**!

```
public class BilClass1 {
  public static String authAddress;
  public static int numberOfCars;
  public String brand;
  public String carNumber;
  public double speed;
  public int totalKm;
  ...
} // end BilClass1
```

Alla variabler är här publika dvs kan ändras utifrån klassen…

```
public class BilTest1 {
   public static void main (String[] args) {
     // skapa ett bil objekt
     BilClass1 enBil = new BilClass1();
     // ge värden
     enBil.carNumber = "ABC123";
     enBil.totalKm = 250;
     enBil.speed = 90;
     System.out.println(enBil.carNumber);
     enBil.totalKm = enBil.totalKm - 500;
     BilClass1.numberOfCars += 1;
  }
}
```

public…

Var försiktig med **public**!

```
public class BilClass1 {
  public static String authAddress;
  public static int numberOfCars;
  public String brand;
  public String carNumber;
  public double speed;
  public int totalKm;
  ...
} // end BilClass1
```

… allt detta utanför bil klassen!

Alla variabler är här publika dvs

här ändrar vi bilens nummer…

här ändrar vi på totala körda km

justerar vi antalet bilar som finns…

```
public class BilTest1 {
   public static void main (String[] args) {
     // skapa ett bil objekt
     BilClass1 enBil = new BilClass1();
     // ge värden
     enBil.carNumber = "ABC123";
     enBil.totalKm = 250;
     enBil.speed = 90;
     System.out.println(enBil.carNumber);
     enBil.totalKm = enBil.totalKm - 500;
     BilClass1.numberOfCars += 1;
   }
}
```

private

Normalt ska tillståndet vara private.

```
public class BilClass1 {
  private static String authAddress;
  private static int numberOfCars;
  private String brand;
  private String carNumber;
  private double speed;
  private int totalKm;
  ...
} // end BilClass1
```

fungerar ej längre!

```
public class BilTest1 {
   public static void main (String[] args) {
     // skapa ett bil objekt
     BilClass1 enBil = new BilClass1();
     // ge värden
     enBil.carNumber = "ABC123";
     enBil.totalKm = 250;
     enBil.speed = 90;
     System.out.println(enBil.carNumber);
     enBil.totalKm = enBil.totalKm - 500;
     BilClass1.numberOfCars += 1;
   }
}
```

Tillståndet ska ändras genom metoder

… metoderna kan kolla att tillståndet blir vettigt.

```
public void setSpeed(double newSp){
  if ((newSp >= 0) && (newSp <= 200)) {
    speed = newSp;
  } else {
    // generera felmeddelande här
  }
} // end setSpeed
```

Nu kan vi ändra på farten via en metod:

```
Car enBil = new Car();
enBil.setSpeed(50);
```

Metoden kan också göra andra saker, t.ex. hålla log på vilka hastigheter som bilen kört.

Hur "läsa" privata variabler?

Man skriver en accessor i klassen Car:

```
public double getSpeed() {
    return speed;
}
```

Det viktiga är att det är klassen som bestämmer hur ett värde sätts och vad som returneras av en get-metod.

Hur hanterar man statiska variabler?

Man skriver en statisk metod:

```
private static String address = "";
...
public static void changeTSV_Address(String newAddress) {
   address = newAddress;
}
```

Hur gör man med bilnumret då?

Det skall ju inte gå att ändra när det väl är satt men man måste kunna ge det ett värde.

Lösning: skapa en konstruerare i vilken man sätter bilnumret:

```
public Car(String carNumber) {
    // spara parametrar
    // gör ev. rimlighets kontroll
    this.carNumber = carNumber;
    numberOfCars = numberOfCars + 1;
    // nollställ och gör vad som behöver
    // göras för att skapa en bil
    totalKm = 0;
    this.speed = 0;
    ...
}
```

Obs. ingen returtyp

metodens namn är klassnamnet

Om **carNumber** är private och vi inte skriver en **set-**metod, så går det ej att ändra på instansvariabeln **carNumber**.

Constructor

- en konstruktor:
  - har samma namn som sin klass
  - konstruerar och initialiserar
  - kan ej anropas direkt bara automatiskt av new
  - är vanligen publik
  - har ingen returtyp
  - kan finnas 0 eller flera
- `this` hänvisar till det implicita objektet dvs “mig själv”

Att komma åt objektets data och metoder

Inuti objektet, dvs när man skriver kod för själva objektet, använder man bara namnet på det man vill referera.

```
<variabel>
this.<variabel>
<metodNamn>(...)
```

Utanför objektet, dvs när man skriver ett program som använder sig av objektet så använder man olika notation beroende på om det man vill referera är statisk eller inte.

För instansvariabler/metoder:

```
<objektNamn>.<variabel>
<objektNamn>.<metodNamn>(...)
```

För statiska variabler/metoder:

```
<Klassnamn>.<variabel>
<Klassnamn>.<metodNamn>(...)
```

Bra/dåliga kommentarer

Skall tillföra något, förklara svåra saker

underkommentera inte

överkommentera inte

Dåliga kommentarer tillför ingen information:

```
int i = 0; // deklarera en räknare
i = i+1; // öka i med ett
System.out.println(); // Tomrad
```

Man kan däremot tala om vad man skall använda räknaren ”i” till (men ännu bättre är kanske att hitta ett bättre namn) och man kan tala om varför man ökar i.

Bra kommentar:

```
// vi ökar med 2 för att bara få med udda tal
i = i+2;
```

Begrepp

klass, objekt/instans

klassmetoder, instansmetoder

get och **set** metoder

public och **private**

static

constructor

javadoc

Programmeringsuppgift

Skriv ett program som simulerar bilar.

```
import java.util.Random;
public class FastCar {
    private String name;
    private int speed;
    private int pos;
    private Random t;
    public FastCar(String name, int speed) {
        this.name = name;
        this.speed = speed;
        this.pos = 0;
        this.t = new Random();
    }
    public int getPos() {
        return pos;
    }
    public void go() {
        if (t.nextInt(5) == 0) {
            System.out.println(name +
               " got a boost from the wind!");
            pos = pos + 10;
        }
        pos = pos + speed;
    }
    public String toString() {
        return name + " is at " + pos + " from origin.";
    }
}
```

```
public class CarChase {
    public static void main (String[] args) {
        FastCar car1 = new FastCar("Souped-up Lada", 50);
        // crime happens
        car1.go();
        car1.go();
        car1.go();
        car1.go();
        // police starts going
        FastCar car2 = new FastCar("Police Volvo", 70);
        while (car1.getPos() > car2.getPos()) {
            car1.go();
            car2.go();
            System.out.println(car1.toString());
            System.out.println(car2.toString());
            System.out.println();
        }
    }
}
```