DA339A Laboration L18

Syfte

Laborationens syfte är att studenten ska öva på att implementera kod som innebär att använda en MVC-struktur och att göra program där flera klasser måste användas för att lösa ett problem. De lösningar som studenten kommer fram till ska också dokumenteras i form av klassdiagram och sekvensdiagram. Studenterna ska också öva på att skriva kommentarer till metoder, klasser med mera och därefter skapa Javadocs vilket är ett verktyg som genererar java koddokumentation i HTML format från javakällkod.

Redovisning

Laborationen ska inte redovisas. Vissa av systembeskrivningarna kan återkomma i senare laborationer som ska redovisas.

Förberedelser

Den här laborationen är direkt kopplad till det innehåll som specifikt finns i föreläsning F12-F16. För att kunna genomföra laboration behöver studenten ha bekantat dig med begrepp från F12-F16. Man bör även ha gjort L17 då L18 bygger vidare på vissa av lösningarna från L17. Lösningar från L13 (DogDayCare) kommer att användas som exempel i uppgift 4.

Inför uppgift 1b nedan så titta på java-klassen JOptionPane i webb-kapitel 26.3 från Deitel. Detta finns även som pdf-fil på kursplatsen med labben (*jhtp_LO_26_GUI1.pdf*).

Titta på koden och kör demo-programmet med JOptionPane i filen *DemoJOptionPane.java*.

Förbered lösningen från L13 till Javadoc uppgiften (uppgift 4), eller använd ett eget projekt som du själv har skrivit.

Mer info om Javadoc i länken nedan:
https://www.jetbrains.com/help/idea/working-with-code-documentation.html

Om lösningar

Det finns ibland vissa direkta fel eller olämpliga lösningar men det finns alltid flera olika ”rätta” lösningar. Olika lösningar har olika fördelar och nackdelar. Om någon annans lösning ser annorlunda ut än din lösning behöver det inte betyda att någon av er har fel. Det kan vara så att ni bara har två olika lösningar med olika fördelar och nackdelar och inget är direkt fel – det är bara olika.

Uppgifter

Uppgift 1a: Gissa nummer med text-gränssnitt

Börja med att skriva ett enkelt ”gissa talet” program. Programmet skall ta emot ett värde på svårighetsgrad som multipliceras med tio. Utskriften nedan visar hur det bör se ut när programmet körs.

```
What level do you what to play 4
Try to guess a number between 0-40 20
You guessed to low, guess again 30
You guessed to high, guess again 25
You guessed to high, guess again 22
You guessed to low, guess again 24
It took you 6 moves to guess 24
Process finished with exit code 0
```

Nu är det ju ett ganska lätt program att skriva och vi hade enkelt kommit under 30 rader om vi bara skrivit det i ett enda “main”-program. Nu skall vi designa vårt program så att vi har brutit ut de olika delar som logik och UI (User Interface - användargränssnitt) d.v.s. att implementera kod enligt en MVC-struktur.

Börja med att rita ett klassdiagram så du har något att utgå från innan du börjar programmera. Du ska skapa minst fyra klasser och lägga så lite kod som möjligt i “main”-program. Så förutom “main”-program så skall du ha en controller, spellogik och UI.

Så med andra ord:

- en boundary-klass som hanterar alla utskrifter och inläsningar via konsolen.
- en controller-klass som styr vad som sker beroende på input från bourdary-klassen.
- en eller flera entity-klasser som representerar data och objekt i modellen, exempelvis ett spel som håller reda på svårighetsgraden, en tärning som slumpar fram nummer och en spelare som håller reda på hur många försök man haft.

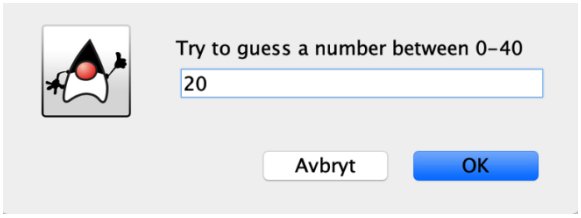
Skriv koden för detta och skapa programmet för att spela spelet ovan. Tänk på att separera ansvarsområdena enligt en strikt MVC-struktur. I nästa uppgift ska du återanvända control- och entity-klasser men byta ut boundary-klassen - helst utan att behöva ändra kod i controller- och entity-klasser.

Uppgift 1b: Gissa nummer med grafiskt gränssnitt

Utgå från din lösning i uppgift 1a och skapa ett grafiskt UI – ett GUI (Graphical User Interface - grafiskt användargränssnitt).

Du behöver inte skapa ett avancerat UI med frames utan använd enkla popup-fönster som

```
JOptionPane.showInputDialog(text)
```



och

```
JOptionPane.showMessageDialog(null, utText).
```

Det första `null` i anropet betyder att fönstret inte är kopplat till en frame.

Har du designat allt rätt så är det enda du skall behöva byta är vilken UI-klass du anropar i control-klassen. Exempelvis

```
GameText outPut = new GameText();
```

byts mot

```
GameUI outPut = new GameUI();
```

Skriv en ny boundary-klass som använder JOptionPane för att kommunicera med användaren. Försök göra detta utan att ändra något i din contol-klass eller entity-klasser. Om du behöver göra ändringar gör då dessa i alla klasser så att både det textbaserade och grafiska UIt kan fungera.

På motsvarade sätt skulle vi nu byta boundary-klasserna med swing och konsolen mot andra UI som fungerar på andra sätt utan att ändra i vår spelkod.

Uppgift 2: Forum testprogram (fortsättning från L17)

Utgå från din lösning till Uppgift 1 i L17 som handlade om ett forum. Du ska från L17 ha en design för ditt forum. I den här uppgiften så ska du skriva kod för ditt forum och använda ett färdigt testprogram eller skriva ditt eget testprogram efter samma mall. I filen *ForumTest.java* (kan laddas ner på Canvas) hittar du ett testprogram för forumet. Du kan se det som att det här programmet skulle simulera saker som sker i boundary-klasser och programmet testar att din control-klass och entity-klasser fungerar som avsett.

Att skriva testprogram är något som upptar en stor del av mjukvaruutveckling. Det är inget konstigt om det finns fler rader kod för att testa en programvara än rader kod i programvaran i sig själv. Ofta är då mycket av den koden genererad via verktyg men att förstå hur man kan bygga upp test av sin kod är viktigt.

Förenkling: I den här uppgiften så kan du helt ignorera moderatorn och vem som har rättighet att ta bort eller ändra ett inlägg. Du kan också bortse från all felkontroll som rimligt borde finnas. Så du behöver inte ta hänsyn till om arrayer blir fulla just nu.

Id på diskussioner, användare och inlägg: Du behöver använda något sätt att identifiera vilken diskussion som du ska göra något med, vilken användare som skapar ett inlägg, vilket inlägg i en diskussion som ska tas bort. Du kan hantera detta på två sätt:

- Koppla detta till platser i arrayer som lagrar de olika objekten
- Förse objekten med ett id som används för att leta upp rätt objekt i arrayen

Att använda platsen i en array är lite enklare just nu men fungerar inte bra på sikt om flera olika saker sker samtidigt. Exempelvis om vi skulle göra detta som en server med klienter som samverkar. Det är dock okej att lösa saker på det här sättet just nu.

Att förse objekten med id och sedan gå igenom arrayen och identifiera rätt objekt utifrån dess id är en mer hållbar lösning i längden om man ska utöka programmet. Detta kräver lite mer kod och går också att göra om du vill ha en lite större utmaning. Vill du göra detta så kan du använda klassvariabler med static för att skapa nya id. Se exempel på detta nedan

```
public class Post {
    private static int nextPostID = 0;

    private User owner;
    private String text;
    private int postID;

    public Post(User owner, String text){
   this.owner = owner;
   this.text = text;
   this.postID = nextPostID;
   nextPostID++;
    }
...
```

Steg 1 - Förstå testprogrammet

Läs koden i filen ForumTest.java och utläs ur denna vilka metoder som behöver finnas i din control-klass. Koden i klassen ForumTest ska endast anropa metoder i control-klassen i din lösning eftersom ForumTest simulerar en boundary-klass. Inga anrop ska ske till entity-klasser i din lösning för forumet.

Metoderna i din control-klass har förmodligen inte samma namn som de i ForumTest så döp om antingen anropen i ForumTest eller metoderna i din control-klass så att de matchar varandra och fungerar att köra.

Vill du skriva ditt eget testprogram så går det bra så länge testprogrammet gör samma saker:

Testsuite 1:

1. Skapa två nya användare i forumet.
   1.1. Lista alla användare i forumet
2. Skapa 3 nya diskussioner med första inlägget i diskussionen, de nya användarna från steg 1 ska användas för detta, en ny användare skapa 2 diskussioner och den andra en diskussion
   2.1. Alla diskussioner i forumet listas med titel och namn på den användare som skapade
3. Lägg till ytterligare 3 inlägg i en av diskussionerna som skapades i steg 2. Växla mellan användarna som skapades i steg 1 för vem som skriver inläggen.
   3.1. Lista alla inlägg i diskussionen med information om vem som skrev inlägget.

Testsuite 2:

1. Skapa 2 nya användare i forumet.
   1.1. Lista alla användare i forumet.
2. Skapa en ny diskussion med ett första inlägg som skrivs av en av användarna som skapades i steg 1.
3. Lägg till ytterligare 3 inlägg i diskussionen som skapades i steg 2. Växla mellan användarna som skapades i steg 1 för vem som skriver inläggen.
   3.1. Lista alla inlägg i diskussionen med information om vem som skrev inlägget.
4. Ta bort det andra inlägget i diskussionen
   4.1. Lista alla inlägg i diskussionen med information om vem som skrev inlägget.
5. Försök ta bort det första inlägget i diskussionen- detta ska inte gå och bör ge ett felmeddelande.
   5.1. Lista alla inlägg i diskussionen med information om vem som skrev inlägget.
6. Ta bort det tredje inlägget i diskussionen
   6.1. Lista alla inlägg i diskussionen med information om vem som skrev inlägget.

Du ska inte köra alla testen med en gång utan vi utökar vilka test som körs efter hand. Se fler instruktioner om detta i kommentarer i ForumTest.java om vad som ska kommenteras bort när.

Steg 2 – Skapa användare
Se till att din kod implementerar så mycket att du kan skapa användare i forumet. Testa detta genom att köra den första delen av Testsuite 1 (kommentera bort kod som kör Testsuite 2 och de senare delarna av Testsuite 1). Om denna fungerar så borde du få en utskrift liknande:

```
Startar Testsuite 1: Lägga till användare, diskussioner och inlägg
alla användare
0: Ada
1: Beata
```

Exakt utseende på utskriften kan variera lite beroende på hur du hanterar id och hur du formaterar strängar som skickas till testprogrammet.

Steg 3 - Lägg till diskussioner
Se till att din kod är så långt implementerad att det går att skapa diskussioner med det första inlägget i forumet. Kör både den första delen och den andra delen av Testsuite1. Du borde nu får en utskrift liknande:

```
Startar Testsuite 1: Lägga till användare, diskussioner och inlägg
alla användare
0: Ada
1: Beata

alla diskussioner
Nr: 0, Titel: Opinions nbr1! by: Ada
Nr: 1, Titel: The second by: Beata
Nr: 2, Titel: Third thoughts! by: Ada
alla inlägg i diskussion 0
Post nr: 0: Ada: First Discussion!
alla inlägg i diskussion 1
Post nr: 0: Beata: Another Discussion
alla inlägg i diskussion 2
Post nr: 0: Ada: Rethinking things
```

Steg 4 - Lägg till fler inlägg i en diskussion

Se till att din kod är så långt implementerad att det går att lägga till fler inlägg i en given diskussion. Kör både den första, andra och tredje delen av Testsuite1. Du borde nu får en utskrift liknande:

```
Startar Testsuite 1: Lägga till användare, diskussioner och inlägg
alla användare
0: Ada
1: Beata

alla diskussioner
Nr: 0, Titel: Opinions nbr1! by: Ada
Nr: 1, Titel: The second by: Beata
Nr: 2, Titel: Third thoughts! by: Ada
alla inlägg i diskussion 0
Post nr: 0: Ada: First Discussion!
alla inlägg i diskussion 1
Post nr: 0: Beata: Another Discussion
alla inlägg i diskussion 2
Post nr: 0: Ada: Rethinking things

Skapar fler inlägg i diskussion 0
alla inlägg i diskussion 0
Post nr: 0: Ada: First Discussion!
Post nr: 1: Beata: Thoughts on first post
Post nr: 2: Ada: Answer to your thoughts
Post nr: 3: Beata: I still do not like it
```

Steg 5 - Kör Testsuite 2
Se till att din kod är så långt implementerad att det går att ta bort ett givet inlägg i en given diskussion Kommentera bort koden som kör Testsuite 1 och kör istället Testsuite 2. Du borde nu får en utskrift liknande:

```
Startar Testsuite 2: Lägga till och ta bort inlägg
alla användare
0: Ada
1: Beata

Skapar fler inlägg i diskussion 0
alla inlägg i diskussion 0
Post nr: 0: Ada: First Discussion!
Post nr: 1: Beata: Thoughts on first post
Post nr: 2: Ada: Answer to your thoughts
Post nr: 3: Beata: I still do not like it

Tar bort post nr 1
Post nr: 0: Ada: First Discussion!
Post nr: 1: Ada: Answer to your thoughts
Post nr: 2: Beata: I still do not like it

Försöker bort post nr 0
Cannot delete first post in discussion.
Post nr: 0: Ada: First Discussion!
Post nr: 1: Ada: Answer to your thoughts
Post nr: 2: Beata: I still do not like it

Tar bort post nr 2
Post nr: 0: Ada: First Discussion!
Post nr: 1: Ada: Answer to your thoughts
```

Steg 6 - Kör bägge testsuiterna
Kör nu bägge testsuiterna med Testsuite1 direkt följd av Testsuite 2. Hur fungerade detta i din kod? Det borde fungera och du får flera användare och diskussioner med samma namn och titlar. Men det ska fortfarande fungera men hur det ser ut kan variera lite mer.

Uppgift 3: Uppdatera diagram
Skapa klassdiagram för Uppgift 1a och 1b samt sekvensdiagram för hur programmet fungerar.

För forumet så uppdaterar du ditt klassdiagram och sekvensdiagram från L16 om du har gjort ändringar som påverkat hur dessa bör se ut. Kod och diagram bör stämma överens.

Gör detta för att öva dig i att se koden i diagrammen och tvärt om.

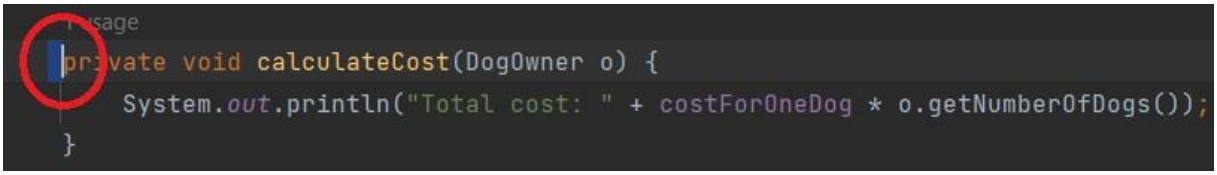
Uppgift 4: Javadoc

I denna uppgift ska du skriva kodkommentarer till metoder, klasser med mera. Vi börjar med att gå igenom lite exempel på hur man skriver en kommentar först och därefter börjar uppgiften.

Uppgift 4.1

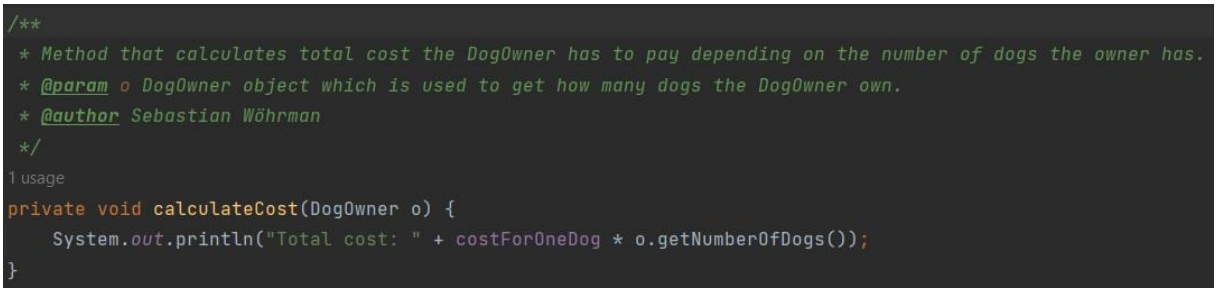
Det finns mer info om Javadoc i länken nedan (det finns även information om Javadoc i F16 och länkar till andra sidor i slutet av föreläsningen):
https://www.jetbrains.com/help/idea/working-with-code-documentation.html

För att skriva en Javadoc kommentar placerar du vart du vill skriva ovanför/bredvid till exempel en metod, se bild nedan:



Därefter skriver du /** och trycker på enter så genereras en kommentar (utan någon info, det måste du själv skriva), se bild nedan:

Det kommer automatiskt att genereras taggar för de olika typer av information som finns i metoden. I bilden ovan ser vi att det har genererats en tagg, @param, för inparametern DogOwner o. Nu när vi har basen/skelettet till Javadoc kommentaren ska vi skriva klart den med informationen som behövs, se bild nedan:



I bilden ovan har vi skrivit en beskrivning till vad metoden gör, vad inparametrarna representerar samt vem som har skrivit metoden/koden.

Nedan kommer ett till exempel på en metod som har ett returvärde:

```
/**
 * Method returns a String when called.
 * @return A String containing the name of the DogOwner.
 * @author Sebastian Wöhrman.
 */
public String getName(){
    return name;
}
```

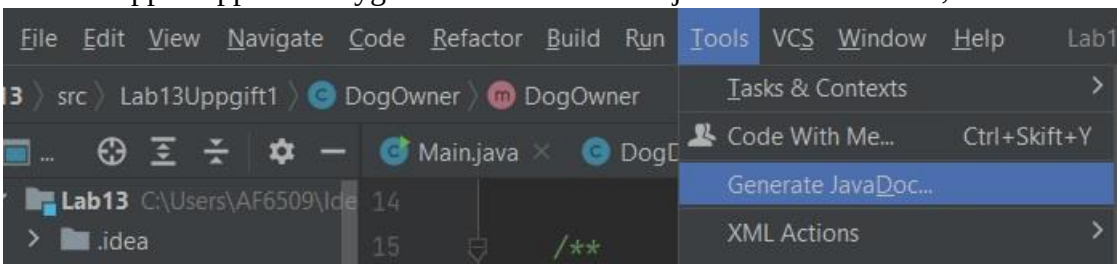
I bilden ovan har vi skrivit en beskrivning till vad metoden gör, vad variabels som returneras representerar samt vem som har skrivit metoden/koden.

Nedan är ett exempel på en kommentar för en klass:

```
/**
 * This class hold information for a dog with a name, gender and age.
 * @author Sebastian Wöhrman
 */
public class Dog {
    2 usages
    private String name;
    2 usages
    private String gender;
    2 usages
    private int age;
```

I bilden ovan har vi skrivit en beskrivning till klassen.

Nu när kommentarer är skrivna på några platser så ska vi generera dokumentation i form av en webbsida. Detta gör vi med hjälp av Javadoc verktyget, vilket du hittar genom att klicka på Tools knappen uppe i verktygsfältet och därefter välj **Generate JavaDoc**, se bild nedan:

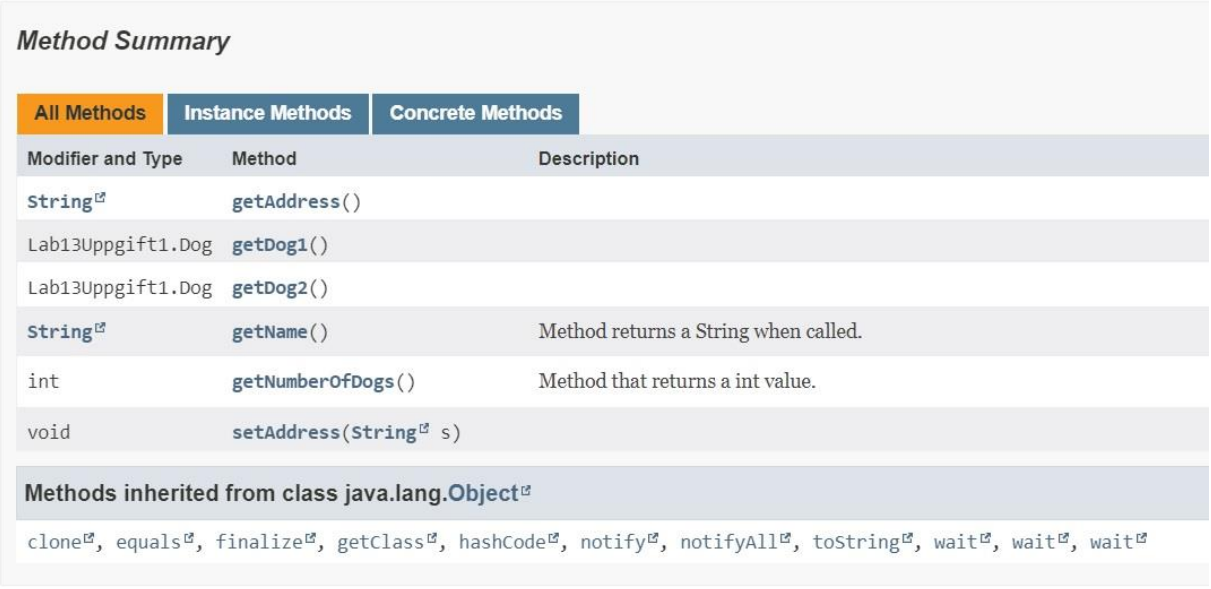


Det finns många alternativ när vi genererar detta men jag valde att endast generera ett Javadoc för klassen DogOwner.

Constructor Summary

Constructors

Constructor	Description
DogOwner(String n, String a, int nbrOfDogs)	Constructor that creates one or two dogs depending on the nbrOfDogs value



Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
String	getAddress()	
Lab13Uppgift1.Dog	getDog1()	
Lab13Uppgift1.Dog	getDog2()	
String	getName()	Method returns a String when called.
int	getNumberOfDogs()	Method that returns a int value.
void	setAddress(String s)	

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Detta som visas på bilden ovan är bara en liten del av det som genereras. Testa att generera ett Javadoc själv och se hur resultatet blir.

Uppgift 4.2

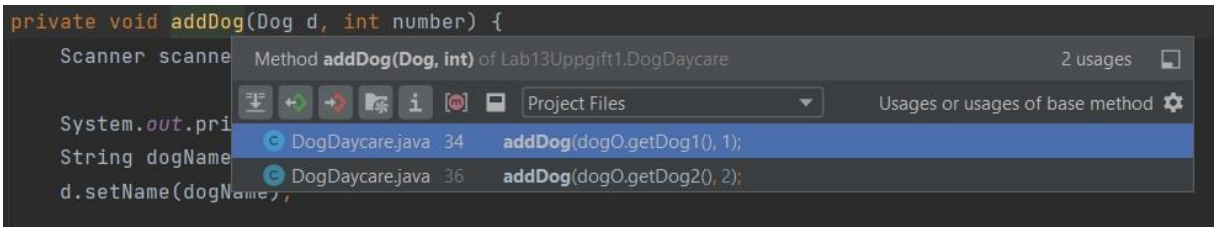
I denna uppgift ska du själv skriva kommentarer samt generera dokumentation. Börja med att öppna ett projekt som redan är skrivet (ett projekt du själv har skrivit eller om det finns ett färdigt skrivet projekt), projektet behöver inte vara färdigt (så länge den innehåller någon klass och att varje klass har någon metod som funkar).

Använd det du lärt dig från uppgiften ovan samt all information som finns i länken given i början av laborationen samt föreläsningarna. Börja med att skriva kommentarer för alla konstruktorerna som finns i projektet. Därefter gör det för alla metoder och generera dokumentationen och se vad resultatet blir.

Uppgift 5: Tips & Hints – ”Ctrl + click”

Ett tips som kan användas när du använder IntelliJ och har problem att hitta alla platser där du anropar olika metoder så kan du hålla mer ctrl-knappen och därefter klicka på en metod. När du gör det kommer det visas hur många gånger som du använt (anropar) metoden samt vart du gjort det.

Exempel nedan visar hur många gånger som metoden addDog har använts samt vart den används:



I bilden ovan ser vi att metoden addDog anropas två gånger, vilka inparametrar den har och vilka kodrader den anropas på.

Om du sedan klickar på en ut av metodanropen så kommer du att tas till den platsen i koden där metoden anropas.