Detaljerad kravspecifikation

PROG1 HT 2019

Innehåll

Introduktion

Kursens inlämningsuppgift går ut på att skapa ett program som registrera och hanterar hundar. Det ska gå att lägga till och ta bort hundar, och i slutet också sälja dem till högstbjudande. Programmet är förenklat, men innehåller de grundläggande CRUD-operationerna (Create, Read, Update, Delete) som är vanliga i dataorienterade applikationer.

Inlämningsuppgiften motsvarar ungefär två veckors heltidsarbete, och det är nog en ganska god uppskattning av hur mycket tid man får räkna med att den tar om man inte har tidigare erfarenhet av programmering. Detta förutsätter dock att man förberett sig och övat på de koncept som tagits upp under kursen. Om man inte gjort det kan den ta längre tid.

För att göra uppgiften hanterbar för alla är den därför indelad i ett antal delinlämningar som var och en fokuserar på en aspekt av programmet. Det är lämpligt att börja med uppgiften någonstans runt föreläsning 7–9 då vi gått igenom allt som krävs för de första delarna och sen arbeta på den parallellt med egna övningar under resten av kursen.

Hjälp med uppgiften kan man få via handledningen. Kursen har fysisk handledning varje dag under kursperioden, och ett elektroniskt handledningsforum i Ilearn. Mer information om detta finns i Ilearn.

På torsdagarna anordnar dessutom DISK workshop handledning. Denna är inte knuten specifikt till PROG1 utan hjälper till även med andra kurser.

Storleken på dokumentet

Att detta dokument är på 61 sidor beror inte beror inte på att uppgiften är extremt stor utan på hur dokumentet är tänkt att användas. Tanken är att du ska kunna ha dokumentet öppet bredvid dig när du arbetar och (i princip) ha direkt tillgång till all information du behöver för varje steg. Detta betyder att ”pappersformatet” är A5 istället för A4, och att mycket information upprepas gång på gång så att du ska slippa hoppa fram och tillbaka.

Samarbete

Inlämningsuppgiften är individuell. Att lämna in något som någon annan har skrivit är plagiat och naturligtvis inte tillåtet. Kursledningen är skyldig att anmäla misstankar om otillåtet samarbete, liksom andra brott mot de regler som gäller för examination, för vidare utredning. Detta kan i värsta fall leda till avstängning från studierna. Naturligtvis blir man inte heller godkänd på uppgiften.

Med det sagt så är det tillåtet att samarbeta med andra så länge varje person gör sin egen implementation. Vår rekommendation för studenter utan tidigare erfarenhet av programmering är att man arbetar tillsammans med någon annan på varsin version av koden. På så sätt finns det någon att bolla idéer med och diskutera när man behöver det.

Vanlig fråga: får jag använda kod från boken, föreläsningarna, webben, etc.?

Ja, så länge du använder den som en bas för din egen kod, förstår den och inte bara kopierar in den och hoppas på det bästa.

Vanlig fråga: nu blev det rörigt, var går gränsen för vad man får använda?

Detta är mycket svårt att svara på. Det finns ofta standardsätt att lösa vissa problem på, och en viktig del av kursen handlar om att lära sig en delmängd av dessa. Detta gör automatiskt att du inte kommer att kunna skriva helt unik kod oavsett vad du gör. Så länge du känner att koden är din egen och att du förstår den så är det antagligen inte ett problem. Vi kommer att ge dig en möjlighet att visa att koden är din egen om det skulle behövas.

Deadlines

De flesta av uppgiftens delar examineras i form av kod som automaträttas av VPL i Ilearn. Deadline för uppgiftens automaträttade delar samt slutrapporten är på kvällen kursens sista dag. Dessa deadlines kontrolleras av inlämningslådorna, och när dessa är stängda så är de stängda. Det går då inte att lämna in på något annat sätt.

Undantaget från ovanstående är granskningen som görs i seminarieform.

Om man missar den ordinarie inlämningen under kursen, eller om man får rest, så ges det två möjligheter till att lämna in uppgifterna. En i samband med omtentan under vårterminen, och en i samband med uppsamlingstentan i augusti. Datum för dessa deadlines är inte bestämda än eftersom de beror på saker som när vi blir klara med rättningen och när uppsamlingstentan placeras. När deadline bestäms anslås detta i kursforumet.

Vanlig fråga: jag missade deadlinen, vad ska jag göra?

Du får vänta tills nästa inlämningsmöjlighet och lämna in då. Om deadlinen du missade var den sista i augusti får du göra inlämningsuppgiften nästa gång kursen går istället.

Vanlig fråga: men snälla, kan jag inte få lämna in ändå?

Nej.

Vanlig fråga: men, jag har en mycket bra anledning...

Nästan säkert nej. Om du har ett intyg om särskilt pedagogiskt stöd som innefattar extra tid för inlämningsuppgifter kan du eventuellt få en extra möjlighet till inlämning efter augustiinlämningen om du gjort ett seriöst försök vid de tidigare inlämningarna. Vill du utnyttja denna möjlighet så kontakta kursansvarig.

Rättningen av slutrapporten sker *efter* deadline. Det spelar alltså ingen roll om du lämnar in slutrapporten en timme eller en månad innan deadline, den kommer att rättas efter deadline oavsett.

I samband med rättningen av slutrapporten kommer vi också att titta på de uppgifter som rättats automatiskt och passa på att uppdatera de automatiska testerna om vi bedömer det nödvändigt. Det finns alltså en liten risk att uppgifter som tidigare blivit godkända av de automatiska testerna inte längre är det efter den manuella rättningen. De automatiska testerna ger bara en rekommendation, de sätter inte det slutgiltiga betyget.

För att bli godkänd på inlämningsuppgiften måste samtliga deluppgifter vara godkända senast efter den sista restinlämningen i augusti. Om de inte är det blir du inte godkänd på denna examinationsdel utan får börja om med den nästa gång kursen ges. Eftersom uppgifter ofta görs om mellan kursomgångar så sparar vi inga delbetyg, utan du börjar i så fall om från början med den uppgift som gäller då.

Uppgiften

Del 1 Uppstart

Det första steget i inlämningsuppgiften går ut på att visa att du har en fungerande utvecklingsmiljö och har kommit igång ordentligt. Du visar detta genom att göra ett antal mindre test och övningar i Ilearn. Du kommer också att få testa på de miljöer för att automattesta kod som används på kursen, och skriva ett eget mindre program. På de efterföljande sidorna listas ett antal ickefunktionella krav som koden ska uppfylla.

För att klara uppgiften måste du ha en fungerande utvecklingsmiljö, ha gått igenom de första kapitlen i kursboken, och skrivit några program på egen hand.

Inlämning och rättning

All inlämning i detta steg sker i Ilearn och rättas automatiskt. Inlämning av uppgiften är möjlig fram till deadline (se sid. 3). Vid deadline stängs inlämningen automatiskt och förblir stängd medan rättningen av slutrapporterna sker. Du kan räkna med att detta tar två till tre veckor. När rättningen av slutrapporterna är klar öppnar inlämningen igen, och du kan lämna in fram till nästa deadline. Undantaget från detta är den sista deadlinen i augusti då inlämningen inte öppnas igen.

Tips för denna del

Behöver du hjälp med uppgiften så finns det handledning, både elektronisk och fysisk. Tänk dock på att den fysiska handledningen måste bokas i förväg, och att vi inte svara på handledningsfrågor via mejl utan bara via handledningsforumet här i ILearn.

Icke-funktionellt krav: namn och användarnamn i samtliga inlämnade filer

Detta krav gäller för hela inlämningsuppgiften, inte bara denna del.

Alla inlämnade filer måste innehålla en kommentar högst upp som innehåller ditt namn och ditt användarnamn. Ett tips är att använda en så kallad JavaDoc-kommentar [6] till klassen i filen. Denna kommer då att hamna nedanför eventuella import-satser. En sådan kommentar skulle kunna se ut ungefär så här:

Exempel: namn och användarnamn i en javadoc-kommentar

```
/**
 * @author Henrik Bergström hebe1234
 */
public class DemoClass {
// ...
}
```

Vanlig fråga: vad är användarnamn för något?

Det ”namn” du använder när du loggar in på DSVs system. Normalt fyra små bokstäver följt av fyra siffror.

Icke-funktionellt krav: koden ska vara kompatibel med Java 11

Detta krav gäller för hela inlämningsuppgiften, inte bara denna del.

All inlämnad kod måste vara kompatibel med Java 11 då detta är den version av Java som finns installerad på testservern. I praktiken kommer du inte att behöva bekymra dig om detta krav eftersom skillnaderna mellan Javas versioner i de delar som ingår på kursen är mycket liten. Om du har en senare version av Java installerad och absolut vill vara säker på att uppfylla kravet så går det att ställa in vilken version som ska användas i din utvecklingsmiljö. Handledningen kan hjälpa dig med detta.

Javas versioner

För den som är intresserad så beskrivs Javas versionshistorik på `https://en.wikipedia.org/wiki/Java_version_history`
En relativt tillgänglig presentation ges i de första tio minuterna av Trisha Gee i *GOTO 2019 Life After Java 8*: `https://www.youtube.com/watch?v=eBuFzQeiGe0`

Icke-funktionellt krav: paket och moduler får inte användas

Detta krav gäller för hela inlämningsuppgiften, inte bara denna del.

Paket och moduler får inte användas i kod som lämnas i för rättning. En av anledningarna är att det är begrepp som inte tas upp på kursen annat än mycket perifert, en annan är att koden då blir svårare att testa med VPL.

Exempel: paketdeklaration

```
// Denna rad får *INTE* stå med i kod som lämnas in.
package demoproject;

public class DemoClass {
// ...
}
```

Exempel: module-info.java

```
// Denna fil ska *INTE* skickas med när kod lämnas in.
// Om du råkar skapa en av misstag kan du plocka bort den.
module se.su.dsv.demomodule {
// Information om modulen
}
```

Icke-funktionellt krav: Javas kodkonventioner ska följas

Detta krav gäller för hela inlämningsuppgiften, inte bara denna del.

De grundläggande kodkonventionerna för Java ska följas [7]. Du behöver inte läsa igenom det fulla dokumentet, vi kommer att gå igenom allt som är relevant för kursen. En sammanfattning är:

- Koden ska vara korrekt indenterad.
- All namngivning av klasser, metoder, variabler, etc. ska ske på engelska.
- Klassnamn är normalt substantiv: `Dog`, `ArrayList`.
- Klassnamn skrivs med stor begynnelsebokstav i varje ingående ord: `String`, `ArrayList`.
- Metodnamn är normalt skrivna i verbform: `getAge`, `add`.
- Metodnamn skrivs med stor begynnelsebokstav i varje ingående ord utom det första: `size`, `getSize`.

Del 2 Programskelett

Med en fungerande utvecklingsmiljö, och en viss vana vid att programmera i Java, är det dags att börja arbeta på den faktiska inlämningsuppgiften, och det första stegen är att implementera skelettet till applikationen. Strukturen för ett sådant generellt programskelett kommer att tas upp på föreläsning 9.

Det går att bli godkänd på denna del genom att ta koden från föreläsningen och göra några mindre förändringar. Gör inte det, du kommer inte att få ut speciellt mycket av det! Tag dig istället tid att lösa uppgiften på egen hand, med hjälp av det som tagits upp på föreläsningen, och se till att du förstår alla delar i koden ordentligt.

Det du ska göra i denna uppgift är att implementera och testa ett skelett till ett kommandostyrt program. Programmet är tänkt att användas för att hålla reda på hundar, och de kommandona som programmet ska acceptera i första skedet är följande:

- `register new dog`
- `increase age`
- `list dogs`
- `remove dog`
- `exit`

Det enda av dessa kommandon som ska fungera i denna första version är `exit`. En beskrivning av hur det ska fungera finns på sidan 14. Om något

Vanlig fråga: måste programmet vara på engelska?

Nej, programmets dialog kan vara på vilket språk du vill. Men, det *måste* acceptera de engelska kommandona också. Anledningen till detta är att testprogrammen annars inte skulle fungera.

annat kommando än de ovanstående anges så ska ett felmeddelande ges. En beskrivning av hur felmeddelandena ska fungera finns på sidan 19.

De övriga kommandona ska bara skriva ut en kort text som berättar vilket kommando som angavs. Denna text *måste* innehålla det fullständiga kommandonamnet enligt ovan så att testprogrammet kan identifiera dem. Namnen på resten av kommandona får inte finnas med i denna utskrift. Då kan testprogrammet inte skilja mellan de olika kommandona.

Förutom de funktionella kraven ovan så finns det ett antal icke-funktionella krav också. Dessa listas efter beskrivningen av kommandot `exit` med början på sidan 8. Så många som möjligt av dessa icke-funktionella krav kontrolleras av det automatiska testprogrammet, men i vissa fall, till exempel metoduppdelningen, är detta inte möjligt. Detta betyder att du kan bli underkänd på uppgiften i efterhand, även om de automatiska testerna går igenom. Risken för detta är normalt liten om man gjort ett ordentligt försök att uppfylla kraven, men den finns, så är du det tveksam: gå på handledningen och diskutera din lösning med en handledare. Handledarna är inte examinatorer, så de kan inte lova att något blir godkänt, men de har god koll på programmering.

Inlämning och rättning

All inlämning i detta steg sker i Ilearn och rättas automatiskt. Inlämning av uppgiften är möjlig fram till deadline (se sid. 3). Vid deadline stängs inlämningen automatiskt och förblir stängd medan rättningen av slutrapporterna sker. Du kan räkna med att detta tar två till tre veckor. När rättningen av slutrapporterna är klar öppnar inlämningen igen, och du kan lämna in fram till nästa deadline. Undantaget från detta är den sista deadlinen i augusti då inlämningen inte öppnas igen.

Tips för denna del

Du förväntas organisera ditt program ordentligt i denna uppgift. En mainmetod som gör allting kommer att gå igenom testprogrammet, men kommer

Vanlig fråga: varför är kommandona så långa?

Därför att de gör feedbacken från det automatiska testprogrammet mycket enklare att läsa.

inte att bli godkänd i slutändan.

Det kommer snabbt att bli jobbigt att skriva in långa kommandonamn varje gång du vill testa något. Ett tips är att låta ditt program acceptera flera olika alias för samma kommando, varav ett är mycket kort, till exempel en enda siffra eller bokstav. Switch-satser är mycket bra för detta.

Syftet med denna del av uppgiften är *inte* att utveckla hela programmet i ett steg utan bara att få ihop en grund att stå på inför de kommande delarna. Funktionaliteten i denna version av programmet ska alltså vara så extremt begränsad som uppgiftstexten säger. Exemplet nedan visar hur det är tänkt att fungera. Det ska inte vara mer än så förutom att alla de givna kommandona ska fungera.

Exempel: programskelettet

```
Hej och välkommen till hundprogrammet!

Kommando?> register new dog
Du gav kommandot: register new dog
Kommando?> remove dog
Du gav kommandot: remove dog
Kommando?> inte ett kommando
Fel, inte ett kommando!
Kommando?> exit

Hej då!
```

Kommando: ”exit”

Detta kommando skriver ut ett meddelande om att programmet avslutas och sen avslutas det. Detta ska göras genom att kommandoloopen avslutas, och inte med `System.exit` som inte får användas.

Exempel: exit

```
Command?> exit
Goodbye!
```

Icke-funktionellt krav: privata data och genomtänkta skyddsnivåer på metoder ska användas

Detta krav gäller för hela inlämningsuppgiften, inte bara denna del.

Alla instansvariabler ska vara privata. Detsamma gäller eventuella klassvariabler. Eventuella konstanter kan vara publika om det är motiverat.

Alla konstruktorer och metoder ska ha en explicit angiven skyddsnivå, antingen private eller public beroende på vad som är lämpligt. Om ni använder arv (som inte tas upp på kursen) så kan det finnas skäl att använda någon av de andra skyddsnivåerna i något enstaka fall, men detta ska i så fall kunna motiveras

Vanligt problem: alla metoder publika

En av de saker vi kommer att titta på i slutinlämningen är vilka metoder som är publika och vilka som inte är det. Du förväntas gå igenom alla metoder och sätta en lämplig skyddsnivå på dem. Det enklaste sättet att uppnå detta är att fundera på skyddsnivån direkt när du skapar metoden.

Vanligt problem: hundklassens metoder privata

De metoder som finns i klassdiagrammet för hundklassen *ska* ha den skyddsnivå som visas där även i slutversionen. Du ska alltså inte göra de get-metoder som inte används i uppgiften privata. De skulle kunna vara intressanta i något annat program som använder klassen.

Icke-funktionellt krav: inga statiska metoder och variabler

Inga statiska metoder eller variabler får användas någonstans i programskelettet. Statiska konstanter får användas för kommandonamnen om man vill, men i övrigt är nyckelordet `static` helt förbjudet i denna del av uppgiften.

Detta krav gäller enbart för denna del av uppgiften. Under senare delar av uppgiften får du använda statiska variabler och metoder, men *bara* om du kan motivera dem ordentligt. Statiska variabler och metoder är nästa alltid en indikation på att man inte programmerar objektorienterat.

Vanligt problem: jag var tvungen att göra X statisk!

Nej, det var du inte! Statiska variabler och metoder är användbara i några situationer, men ganska få, och i hela den här inlämningsuppgiften finns det bara något enstaka ställe där det *eventuellt* skulle vara korrekt att använda dem.

Exempel: Cannot make a static reference to the non-static field variable

```
// Det korrekta här är nästan säkert inte att göra
// variabeln statisk utan att arbeta med en instans
// av klassen istället.
int variable;

public static void main(String[] args) {
variable = 10;
}
```

Icke-funktionellt krav: ordentlig metoduppdelning

Detta krav gäller för hela inlämningsuppgiften, inte bara denna del.

Programmet ska vara uppdelat i lämpliga metoder. Det programskelett som tas upp på föreläsningen är en bra utgångspunkt. Detta krav kan inte kontrolleras av testprogrammet, men det kommer att försöka varna för om metoder börjar bli väl långa.

En bra utgångspunkt när det gäller metoduppdelning är att varje metod ska göra en enda sak. Detta betyder att de flesta metoder ska vara mycket korta.

Metoddesign kommer att diskuteras på flera föreläsningar och övningar under kursen.

Icke-funktionellt krav: ledtexter ska alltid följas av ?>

Detta krav gäller för hela inlämningsuppgiften, inte bara denna del.

Innan *varje* inläsning måste det komma en ledtext som berättar vad användaren förväntas skriva in följt av `?>`. Ledtexten måste bestå av minst ett ord, den kan alltså inte bara vara `?>`. Det får finnas mellanslag mellan ledtexten och `?>`.

Ditt program får *inte* skriva ut `?>` någon annanstans i dialogen med användaren än precis innan programmet ska vänta på inmatning.

Exempel: korrekt inläsning

```
Vad heter hunden?>
```

Exempel: tre inkorrekta inläsningsförsök

```
Vad heter hunden?
Vad heter hunden>
Vad heter hunden:
```

Vanlig fråga: varför finns detta krav?

Detta krav kan tyckas godtyckligt, och till viss del är det också det. Men, det har en viktig funktion: det möjliggör för testprogrammet att sluta leta efter ett svar som aldrig kommer mycket snabbare. På de stora testerna i slutet av kursen när hela programmet ska gås igenom kan det spara minuter vid testkörningen.

Icke-funktionellt krav: felmeddelanden måste innehålla ”fel” eller ”error”

Detta krav gäller för hela inlämningsuppgiften, inte bara denna del.

Varje felmeddelande som ditt program skriver ut måste innehålla ett av orden ”fel” eller ”error” så att testprogrammet kan identifiera dem.

Exempel: korrekt felmeddelande

```
Command?> list dogs
Error: no dogs in register
```

Exempel: felaktigt felmeddelande som inte kommer hittas av testprogrammet

```
Command?> list dogs
No dogs in register
```

Icke-funktionellt krav: kommandoloopen måste implementeras iterativt

Detta krav gäller för hela inlämningsuppgiften, inte bara denna del.

Kommandoloopen ska vara implementerad iterativt (med en loop) och inte med rekursion. Detta kan inte kontrolleras automatiskt på ett enkelt sätt, utan du får själv se till att så är fallet.

Rekursion kommer att tas upp mot slutet av kursen, men problemet med en rekursiv kommandoloop kommer att tas upp på föreläsningen om programskelettet.

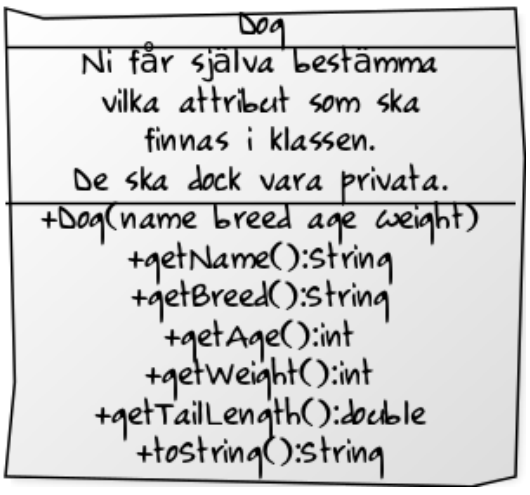
Del 3 Implementera en hundklass

Nu är det dags att börja jobba med hundarna, och den först uppgiften blir att implementera en klass som representerar en hund. Till din hjälp har du klassdiagrammet i figur 1 nedan. Som du kan se är enbart det publika gränssnittet medtaget i diagrammet, och detta publika gränssnitt måste följas exakt. Hur klassen fungerar internt är däremot upp till dig att bestämma. Privata variabler och metoder är alltså tillåtna. Du kan börja med denna uppgift efter den första av föreläsningarna om klasser och objekt.

Svanslängden för en hund kan räknas ut med formeln:

$svanslängd = ålder \cdot \frac{vikt}{10}$

Denna formel gäller för alla hundar utom taxar. En tax har alltid svanslängden 3,7. När du implementera detta är det viktigt att det gäller för taxar på alla språk eftersom kennelvärlden är internationell. I praktiken räcker det med att programmet klarar av att hantera det svenska ordet ”tax” och det engelska ”dachshund”, men fundera gärna på en lösning som också skulle klara ”mäyräkoira”, ”teckel”, etc.



Figur 1: Klassdiagrammet

En sak saknas helt i klassdiagrammet, metoder för att uppdatera attributen. Detta beror på att beställaren av systemet ännu inte bestämt vilka av dessa som ska kunna uppdateras. Vi vet dock att hundar blir äldre med åren. I uppgiften ingår därför också att själv designa och implementera en metod för att uppdatera åldern. När du gör det så tänk på att åldern bara kan öka, aldrig minska.

Inlämning och rättning

All inlämning i detta steg sker i Ilearn och rättas automatiskt. Inlämning av uppgiften är möjlig fram till deadline (se sid. 3). Vid deadline stängs inlämningen automatiskt och förblir stängd medan rättningen av slutrapporterna sker. Du kan räkna med att detta tar två till tre veckor. När rättningen av slutrapporterna är klar öppnar inlämningen igen, och du kan lämna in fram till nästa deadline. Undantaget från detta är den sista deadlinen i augusti då inlämningen inte öppnas igen.

Tips för denna del

Det är viktigt att du viktigt att du följer klassdiagrammet i figur 1 exakt. Minsta avvikelse kommer att göra att din kod inte kompilerar på testservern. Klassdiagram tas upp i kursboken [4, sid. 347–348], men också i kursböckerna från OOS [2], IDSV [1] och OOP [3].

En array kan hjälpa dig om du vill kunna hantera många språks ord för tax.

Vanligt problem: kompileringsfel vid testning

Om du får kompileringsfel vid automattestningen av denna klass så beror detta på att du inte följt klassdiagrammet. Det är *mycket* viktigt att du följer det *exakt*.

Vanlig fråga: var är programmet?

Ett syfte med denna deluppgift är att tydliggöra skillnaden mellan klasser och program. Klassen `Dog` representerar en hund, en hund är inte ett program. Det är något som ett program kan hantera.

Del 4 Sortering av hundar

Ett av kursens mål säger att du efter avklarad kurs ska kunna ”konstruera algoritmer som löser programmeringsproblem, implementera algoritmerna samt rätta eventuella syntaktiska och logiska fel.”

En lämplig startpunkt för att öva på detta är att börja med en given algoritm och implementera den, och det är precis det vi ska göra nu. Uppgiften går ut på att läsa in dig på en sorteringsalgoritm och implementera den i en metod som ska sorterar en `ArrayList` innehållande hund-objekt. Listan ska vara sorterad efter svanslängd. Om två hundar har samma svanslängd ska de vara sorterade efter namn. I Ilearn finns det kod som skapar en lista med hundar och anropar metoden. Använd denna kod som utgångspunkt för din implementation.

Det finns många sorteringsalgoritmer att välja mellan. Några lite enklare är *insertion sort*, *selection sort* och *bubbel sort*. Mer effektiva, men något mer komplicerade, är *quick sort* och *merge sort*.

Kursboken tar upp sortering på många olika ställen. *Selection sort* gås igenom i avsnitt 7.11 [4, sid. 293]. För dem som gick IDSV så tar kursboken därifrån upp *insertion sort* i avsnitt 5.4 [1, sid. 285]. Bägge dessa algoritmer är lämpliga för uppgiften.

Inlämning och rättning

All inlämning i detta steg sker i Ilearn och rättas automatiskt. Inlämning av uppgiften är möjlig fram till deadline (se sid. 3). Vid deadline stängs inlämningen automatiskt och förblir stängd medan rättningen av slutrapporterna sker. Du kan räkna med att detta tar två till tre veckor. När rättningen av slutrapporterna är klar öppnar inlämningen igen, och du kan lämna in fram till nästa deadline. Undantaget från detta är den sista deadlinen i augusti då inlämningen inte öppnas igen.

Tips för denna del

Försök inte göra allt på en gång utan ta det steg för steg. Börja med att läsa på om ett par av de rekommenderade sorteringsalgoritmerna och välj ut den du tycker verkar enklast eller mest intressant.

Testa därefter att implementera algoritmen för någon enkel datatyp som heltal eller strängar så att du kan testa att din implementation av algoritmen fungerar utan att behöva blanda in hundklassen.

Byt därefter ut typen av data som ska sorteras till `Dog` och modifiera implementationen så att hundarna sorteras efter svanslängd. Bry dig inte om namnen än så länge.

När detta fungerar så modifierar du implementationen en gång till så att den istället sorterar på namnen och ignorerar svanslängden.

Slutligen slår du samman dessa bägge versioner så att du sorterar först på både svanslängd och därefter på namn.

Del 5 Hantera hundarna

Nu när programskelettet, hundklassen och sorteringsmetoden är klara är det dags att sätta ihop dessa till ett program. I sin första inkarnation ska detta program bara vara ett enkelt register för hundar. Kommandona är desamma som i programskelettet, men nu ska de fungera enligt kommandobeskrivningarna som på sidan 27 och framåt. Eftersom programmet ska testas automatiskt är det viktigt att instruktionerna följs till punkt och pricka.

De olika hund-objekten som skapas skall lagras i en datasamling. Här skall `ArrayList<Dog>` användas. Hantering av hundobjekt görs då genom att skapa ett objekt av samlingsklassen och i detta objekt anropa olika metoder, exempelvis `add` för att addera en ny hund till samlingen.

Vid lösningen av denna uppgift kan du förutsätta att användaren gör rätt. Om man t.ex. frågar efter ett heltal så kan du förutsätta att ett heltal verkligen matas in (skulle något annat matas in så leder det förmodligen till ett exekveringsavbrott). Det behövs alltså inga indatakontroller för att man skall bli godkänd.

Inlämning och rättning

All inlämning i detta steg sker i Ilearn och rättas automatiskt. Inlämning av uppgiften är möjlig fram till deadline (se sid. 3). Vid deadline stängs inlämningen automatiskt och förblir stängd medan rättningen av slutrapporterna sker. Du kan räkna med att detta tar två till tre veckor. När rättningen av

Vanlig fråga: måste alla test gå igenom?

Den inlämnade koden måste gå igenom samtliga VPL-test för uppgiften innan den är godkänd. Detta gäller samtliga automaträttade delar. Eventuella saker som testprogrammet säger sig vara osäker på måste antingen rättas till eller motiveras ordentligt i slutrapporten. Om du är osäker på om något behöver rättas så kontakta handledningen.

slutrapporterna är klar öppnar inlämningen igen, och du kan lämna in fram till nästa deadline. Undantaget från detta är den sista deadlinen i augusti då inlämningen inte öppnas igen.

Tips för denna del

Rättningsprogrammet för denna del påverkas inte av om det finns mer funktionalitet i programmet än kommandona på de kommande sidorna. Det går alltså att göra klart det fullständiga programmet och sen skicka exakt samma program för rättning tre gånger, en gång för del 5, en för del 6 och en för del 7. Om du känner dig säker på dig själv kan detta spara tid, men annars är det antagligen betydligt bättre att ta uppgiften steg för steg.

En annan sak som är bra att tänka på är att ta kommandona i den ordning de presenteras och att göra klart ett kommando innan du ger dig på nästa.

När användaren vill avsluta ska programmet avsluta som tidigare. En tråkig effekt är att när programmet avslutas så ”försvinner” allt lagrat data (objekt som skapas ligger i primärminnet och finns bara så länge programmet körs). Alla registrerade hundar försvinner alltså varje gång vi avslutar programmet. Ett tips för att underlätta testning är därför att låta programmet lägga in ett par hundar automatiskt när det startar eller lägga till ett extra kommando som automatiskt lägger in några hundar.

Om du inte redan gjort det bör du också överväga att lägga in alternativa korta kommandon så att du slipper skriva in de fullständiga kommandona varje gång du testkör.

Kommando: ”register new dog”

Detta kommando registrerar en ny hund. Användaren ska få frågor om namn, ras, ålder och vikt för hunden i denna ordning. Ett hundobjekt skapas sen och läggs in i registret.

Du kan anta att alla hundnamn är unika och att användaren inte gör något fel med siffrorna.

Exempel: register new dog

```
Command?> register new dog
Name?> Lassie
Breed?> Collie
Age?> 78
Weight?> 25
Lassie added to the register
```

Kommando: ”list dogs”

Detta kommando listar hundarna i registret. Användaren ska få en fråga om minsta svanslängd och programmet ska skriva ut en lista på alla hundar hos kenneln som har samma eller längre svanslängd än denna minsta angivna. Om man anger 0 så kommer alltså alla hundar att skrivas ut. Anger man istället 10 så skrivs bara de hundar vars svanslängd är större eller lika med 10 ut.

Vid utskriften skall alla hundens attribut samt svanslängden skrivas ut. Listan ska vara sorterad med hjälp av den sorteringsmetod du implementerade i det föregående steget.

Exempel: list dogs

```
Command?> list dogs
Smallest tail length to display?> 1,2
The following dogs has such a large tail:
* Fido (Dachshund, 1 years, 2 kilo, 3,70 cm tail)
* Karo (Bulldog, 8 years, 7 kilo, 5,60 cm tail)
* Milou (Terrier, 7 years, 8 kilo, 5,60 cm tail)
```

Om det inte finns några hundar registrerade ska det inte komma någon fråga om minsta svanslängd. Istället ska det skrivas ut ett felmeddelande. Var noga med att felmeddelandet innehåller något av orden *fel* eller *error*, annars kan testprogrammet inte känna igen att det rör sig om ett felmeddelande.

Exempel: list dogs

```
Command?> list dogs
Error: no dogs in register
```

Kommando: ”increase age”

Detta kommando ökar åldern på en hund med ett. Användaren ska få en fråga efter namnet på hunden som ska åldras. Hunden med det angivna namnet ska därefter bli ett år äldre.

Denna funktion påverkas av kravet att acceptera namn i alla möjliga olika format. Tänk därför på att till exempel *Tim*, *tim* och *TIM* alla ska ge samma resultat eftersom det är samma namn. För mer information om detta se sidan 31.

Exempel: increase age för hund som finns

```
Command?> increase age
Enter the name of the dog?> Karo
Karo is now one year older
```

Om det inte finns någon hund med det angivna namnet så skall programmet skriva ut ett felmeddelande. Var noga med att felmeddelandet innehåller något av orden *fel* eller *error*, annars kan testprogrammet inte känna igen att det rör sig om ett felmeddelande.

Exempel: increase age för hund som inte finns

```
Command?> increase age
Enter the name of the dog?> Ratata
Error: no such dog
```

Kommando: ”remove dog”

Detta kommando tar bort en hund ur registret. Användaren ska få en fråga om namnet på den hund som skall tas bort varefter hunden med det angivna namnet tas bort.

Du behöver inte hantera komplikationen att det kan finnas flera hundar med samma namn.

Denna funktion påverkas av kravet att acceptera namn i alla möjliga olika format. Tänk därför på att till exempel *Tim*, *tim* och *TIM* alla ska ge samma resultat eftersom det är samma namn. För mer information om detta se sidan 31.

Exempel: remove dog när hunden finns

```
Command?> remove dog
Enter the name of the dog?> Karo
Karo is removed from the register
```

Om det inte finns någon hund med det angivna namnet så skall programmet skriva ut ett felmeddelande. Var noga med att felmeddelandet innehåller något av orden *fel* eller *error*, annars kan testprogrammet inte känna igen att det rör sig om ett felmeddelande.

Exempel: remove dog när hunden inte finns

```
Command?> remove dog
Enter the name of the dog?> Ratata
Error: no such dog
```

Övergripande krav: inläsning av namn

Detta krav gäller för hela inlämningsuppgiften, inte bara denna del.

Programmet som ska skrivas hanterar namn på hundar, raser och användare. Tyvärr är det lätt hänt att användare skriver namn på olika sätt; Henrik en gång, henrik nästa, och med Caps Lock intryckt hENRIK.

Programmet ska acceptera samtliga former på namn och klara av att känna igen det även om det skrivs in på något annat sätt.

Exempel: Namn skrivna på olika format

```
Command?> register new user
Name?> peter
Peter added to register
Command?> make bid
Enter the name of the user?> pETER
Enter the name of the dog?> rEx
Amount to bid (min 1)?> 100
Bid made
```

Programmet ska även ta bort extra mellanslag i börja och slutet av namnen.

Exempel: Namn skrivet med extra mellanslag

```
Command?>␣register␣new␣user
Name?>␣␣␣␣␣␣␣␣␣␣␣␣␣Lotta
Lotta␣added␣to␣register
```

Om namnet är tomt, eller bara består av mellanslag ska användaren få en ny chans att skriva in namnet. Detta ska upprepas tills användaren skriver något.

Exempel: Inget namn angivet

```
Command?> register new dog
Name?>
Error: the name can't be empty
Name?>
Error: the name can't be empty
Name?> Ratata
Breed?>
Error: the name can't be empty
Breed?> No idea
Age?> 5
Weight?> 7
Ratata added to the register
```

Tips!

Det enklaste sättet att uppfylla detta krav är att skriva en metod för inläsning av namn som alltid normaliserar namnet till det format du vill ha det på. Detta kan vara bara små bokstäver, bara stora, eller med lite mer jobb det normala skrivsättet med stor begynnelsebokstav.

Icke-funktionellt krav: en objektorienterad lösning

Detta krav gäller för hela inlämningsuppgiften, inte bara denna del.

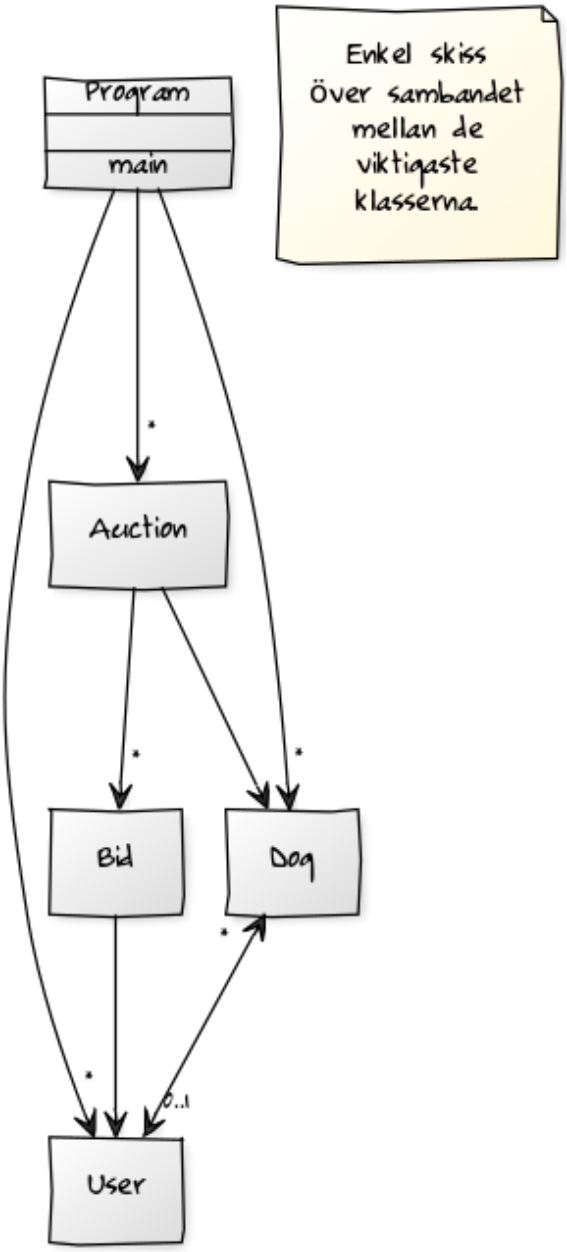
Java är ett objektorienterat programmeringsspråk med en imperativ grund. Detta, i kombination med att uppgiften är så pass liten att det inte finns något egentliget behov av modularisering, gör det lätt att lösningen blir mer imperativ än objektorienterad. Det går att uppfylla alla funktionella krav i ett program som består av en enda klass med enbart statiska metoder. En sådan lösning kommer *inte* att godkännas.

Den programmeringsparadigm som kursen behandlar är objektorienterad programmering, och implementationen måste därför vara objektorienterad.

Figur 2 på nästa sida visar en skiss över relationerna mellan de viktigaste klasserna i uppgiften. Denna skiss utgör inte på något sätt något facit, men kan fungera som utgångspunkt för arbetet. Det är alltså tillåtet att introducera fler klasser, döpa om klasser (förutom `Dog`), etc.

I del 5 så kommer du bara att arbeta med de klasser som i diagrammet heter `Program` och `Dog`. Utgångspunkten för `Program` är det programskelett du tog fram i del 2, och som nu bör döpas om eftersom den inte längre är ett programskelett. `Dog`-klassen är den du skrev i del 3.

Designskissen i figur 2 kommer att tas upp när vi diskuterar inlämningsuppgiften på någon av föreläsningarna. Där kommer vi också att titta på var funktionalitet bör ligga.



Figur 2: Designskiss

Del 6 Hantera ägare

Repetition sägs ju vara kunskapens moder, så nu ska vi göra nästan samma sak som i del 5 en gång till. Den enda skillnaden är att vi introducerar ytterligare en klass, den som i skissen för systemet (figur 2 sid. 34) kallas för `User`.

Funktionaliteten som ska läggas till är kommandon för att lägga till en ägare, lista ägare och ta bort ägare som alla beskrivs i detalj på de kommande sidorna. Det ska också finnas ett kommando för att lägga till en hund till en ägare (sid. 38).

Inlämning och rättning

All inlämning i detta steg sker i Ilearn och rättas automatiskt. Inlämning av uppgiften är möjlig fram till deadline (se sid. 3). Vid deadline stängs inlämningen automatiskt och förblir stängd medan rättningen av slutrapporterna sker. Du kan räkna med att detta tar två till tre veckor. När rättningen av slutrapporterna är klar öppnar inlämningen igen, och du kan lämna in fram till nästa deadline. Undantaget från detta är den sista deadlinen i augusti då inlämningen inte öppnas igen.

Tips för denna del

Rättningsprogrammet för denna del påverkas inte av om det finns mer funktionalitet i programmet än kommandona på de kommande sidorna. Det går alltså att göra klart det fullständiga programmet och sen skicka exakt samma program för rättning tre gånger, en gång för del 5, en för del 6 och en för del 7. Om du känner dig säker på dig själv kan detta spara tid, men annars är det antagligen betydligt bättre att ta uppgiften steg för steg.

En annan sak som är bra att tänka på är att ta kommandona i den ordning de presenteras och att göra klart ett kommando innan du ger dig på nästa.

Kommando: ”register new user”

Detta kommando registrerar en ny användare. Användare är ett nytt koncept i denna version av programmet. För tillfäller representerar användarna kunder, men det finns tankar på att i framtiden utveckla systemet så att det stödjer andra typer av verksamheter så som sociala media för hundar, hundtävlingar och hunddagis. Man har därför valt att använda det mer generella ordet användare (user) istället för kund.

När användaren (av programmet) vill registrera en ny användare (kund) så ska denne få en fråga om namnet på den nya användaren som ska skapas. Därefter skapas en användare och läggs in i systemet.

Precis som för hundar kan du anta att alla användare har unika namn.

Exempel: register new user

```
Command?> register new user
Name?> Henrik
Henrik added to the register
```

Kommando: ”list users”

Detta kommando listar samtliga användare i registret. Vid utskriften ska namnet på användaren finnas med tillsammans med namnet på alla hundar denne äger. Det sistnämnda kravet betyder att du kommer att behöva arbeta på kommandot ”give dog” (sid. 38) parallellt med ”list users”.

Exempel: list users

```
Command?> list users
Henrik [Karo, Rex]
Olle []
```

Om det inte finns några användare registrerade ska det istället skrivas ut ett felmeddelande. Var noga med att felmeddelandet innehåller något av orden *fel* eller *error*, annars kan testprogrammet inte känna igen att det rör sig om ett felmeddelande.

Exempel: list users när det inte finns några användare

```
Command?> list users
Error: no users in register
```

Kommando: give dog

Användaren får frågan om vilken hund som ska ges bort och vilken användare som ska få den.

Exempel: give dog

```
Command?> give dog
Enter the name of the dog?> Karo
Enter the name of the new owner?> Henrik
Henrik now owns Karo
```

När man lägger till en hund till en ägare så måste information om detta sparas någonstans. Figur 2 (sid. 34) berättar var: hos hunden respektive ägaren. Den dubbelriktade pilen mellan klasserna `User` och `Dog` betyder att de har en direkt relation. Varje ägare ska alltså hålla reda på vilka hundar denna äger, och varje hund ska hålla reda på vem som äger den. I bägge fallen ska de hålla referenser till instanser av den andra den andra klassen, de får inte spara enbart namnen.

Ägaren ska spara sina hundar i en array, *inte* i en `ArrayList`. Det får inte finnas någon restiktion på hur många hundar en ägare kan ha, så om arrayen blir full måste den växa. Kursboken tar upp detta i avsnitt 7.5 [4, sid. 280–281].

Om hunden eller användaren inte finns, eller om hunden redan har en ägare ska ett felmeddelande ges. Var noga med att felmeddelandet innehåller något av orden *fel* eller *error*, annars kan testprogrammet inte känna igen att det rör sig om ett felmeddelande.

Exempel: give dog

```
Command?> give dog
Enter the name of the dog?> Karo
Error: no dog with that name
```

Exempel: give dog

```
Command?> give dog
Enter the name of the dog?> Karo
Enter the name of the new owner?> Henrik
Error: no such user
```

Exempel: give dog

```
Command?> give dog
Enter the name of the dog?> Karo
Error: the dog already has an owner
```

Kommando: ”remove user”

Detta kommando tar bort en användare ur registret. Användaren (av programmet) ska få en fråga om namnet på användaren som skall tas bort varefter användaren med det angivna namnet tas bort.

När en användare tas bort ska även användarens hundar tas bort.

Du behöver inte tänka på komplikationen att det kan finnas flera användare med samma namn.

Exempel: remove user när användaren finns

```
Command?> remove user
Enter the name of the user> Henrik
Henrik is removed from the register
```

Om det inte finns någon användare med det angivna namnet så skall programmet skriva ut ett felmeddelande. Var noga med att felmeddelandet innehåller något av orden *fel* eller *error*, annars kan testprogrammet inte känna igen att det rör sig om ett felmeddelande.

Exempel: remove user när användaren inte finns

```
Command?> remove user
Enter the name of the user?> Henrik
Error: no such user
```

Förändrat kommando: ”list dogs”

Detta kommando fanns med i programmet du lämnade in i del 5, men har något förändrade krav. Det som har förändrats är att ägaren ska finnas med vid utskriften.

Detta kommando listar hundarna i registret. Användaren ska få en fråga om minsta svanslängd och programmet ska skriva ut en lista på alla hundar hos kenneln som har samma eller längre svanslängd än denna minsta angivna. Om man anger 0 så kommer alltså alla hundar att skrivas ut. Anger man istället 10 så skrivs bara de hundar vars svanslängd är större eller lika med 10 ut.

Vid utskriften skall alla hundens attribut samt svanslängden skrivas ut. Detta inkluderar i denna version av programmet en eventuell ägare. Listan ska vara sorterad efter svanslängd. Om två hundar har samma svanslängd ska de vara sorterade efter namn.

Exempel: list dogs

```
Command?> list dogs
Smallest tail length to display?> 1
The following dogs has such a large tail:
* Fido (Dachshund, 1 years, 2 kilo, 3,70 cm tail)
* Rex (Bulldog, 8 years, 7 kilo, 5,60 cm tail, owned by Bo)
* Milou (Terrier, 7 years, 8 kilo, 5,60 cm tail)
```

Om det inte finns några hundar registrerade ska det inte komma någon fråga om minsta svanslängd. Istället ska det skrivas ut ett felmeddelande. Var noga med att felmeddelandet innehåller något av orden *fel* eller *error*, annars kan testprogrammet inte känna igen att det rör sig om ett felmeddelande.

Exempel: list dogs

```
Command?> list dogs
Error: no dogs in register
```

Förändrat kommando: ”remove dog”

Detta kommando fanns med i programmet du lämnade in i del 5, men har något förändrade krav. Det som har förändrats är att det inte räcker med att bara ta bort hunden ur registret. Den måste också tas bort från sin ägare.

Detta kommando tar bort en hund ur registret. Användaren ska få en fråga om namnet på den hund som skall tas bort varefter hunden med det angivna namnet tas bort. I den tidigare versionen av programmet räckte det med att ta bort hunden ur `ArrayList`en, men nu måste hunden också tas bort från en eventuell ägare, och från auktionerna om den är till salu.

Du behöver inte hantera komplikationen att det kan finnas flera hundar med samma namn.

Exempel: remove dog när hunden finns

```
Command?> remove dog
Enter the name of the dog?> Karo
Karo is removed from the register
```

Om det inte finns någon hund med det angivna namnet så skall programmet skriva ut ett felmeddelande. Var noga med att felmeddelandet innehåller något av orden *fel* eller *error*, annars kan testprogrammet inte känna igen att det rör sig om ett felmeddelande.

Exempel: remove dog när hunden inte finns

```
Command?> remove dog
Enter the name of the dog?> Ratata
Error: no such dog
```

Del 7 Hantera auktioner

Den sista delen av programmet handlar om att sälja hundar på auktion och involverar de två sista klasserna i figur 2 (sid. 34): `Auction` och `Bid`.

Inlämning och rättning

All inlämning i detta steg sker i Ilearn och rättas automatiskt. Inlämning av uppgiften är möjlig fram till deadline (se sid. 3). Vid deadline stängs inlämningen automatiskt och förblir stängd medan rättningen av slutrapporterna sker. Du kan räkna med att detta tar två till tre veckor. När rättningen av slutrapporterna är klar öppnar inlämningen igen, och du kan lämna in fram till nästa deadline. Undantaget från detta är den sista deadlinen i augusti då inlämningen inte öppnas igen.

Tips för denna del

De bägge ”list”-kommandona kräver att buden ska vara sorterade. Du behöver inte nödvändigtvis implementera en sorteringsalgoritm för detta om du ser till att buden alltid sparas i rätt ordning.

Rättningsprogrammet för denna del påverkas inte av om det finns mer funktionalitet i programmet än kommandona på de kommande sidorna. Det går alltså att göra klart det fullständiga programmet och sen skicka exakt samma program för rättning tre gånger, en gång för del 5, en för del 6 och en för del 7. Om du känner dig säker på dig själv kan detta spara tid, men annars är det antagligen betydligt bättre att ta uppgiften steg för steg.

En annan sak som är bra att tänka på är att ta kommandona i den ordning de presenteras och att göra klart ett kommando innan du ger dig på nästa.

Kommando: ”start auction”

Detta kommando startar en ny auktion för att sälja en hund. Användaren ska få frågan om vilken hund som ska säljas varefter auktionen skapas. När en auktion skapas ska den automatiskt få ett löpnummer. Den första auktionen som skapas ska få nummer 1, nästa nummer 2, osv.

Exempel: start auction

```
Command?> start auction
Enter the name of the dog?> Fido
Fido has been put up for auction in auction #5
```

Om hunden som ska säljas inte finns ska ett felmeddelande ges och ingen auktion skapas. Var noga med att felmeddelandet innehåller något av orden *fel* eller *error*, annars kan testprogrammet inte känna igen att det rör sig om ett felmeddelande.

Exempel: start auction när hunden inte finns

```
Command?> start auction
Enter the name of the dog?> Karo
Error: no such dog
```

Om hunden som ska säljas redan finns utlagd för försäljning ska ett felmeddelande ges och ingen auktion skapas. Var noga med att felmeddelandet innehåller något av orden *fel* eller *error*, annars kan testprogrammet inte känna igen att det rör sig om ett felmeddelande.

Exempel: start auction när hunden redan är utlagd till försäljning

```
Command?> start auction
Enter the name of the dog?> Fido
Error: this dog is already up for auction
```

Om hunden som ska säljas redan har en ägare ska ett felmeddelande ges och ingen auktion skapas. Var noga med att felmeddelandet innehåller något av orden *fel* eller *error*, annars kan testprogrammet inte känna igen att det rör sig om ett felmeddelande.

Exempel: start auction när hunden redan har en ägare

```
Command?> start auction
Enter the name of the dog?> Fido
Error: this dog already has an owner
```

I exemplen ovan ges tre olika felmeddelanden för de olika feltillstånden. Detta är bra ur användarsynvinkel, men inte ett krav. Det kan vara samma felmeddelande i alla tre fallen.

Kommando: ”make bid”

Detta kommando registrerar ett nytt bud på en hund. Användaren ska få frågor om vem som vill lägga budet, vilken hund som ska bjudas på, och hur mycket som bjuds i denna ordning.

Exempel: make bid

```
Command?> make bid
Enter the name of the user?> Henrik
Enter the name of the dog?> Milou
Amount to bid (min 201)?> 300
Bid made
```

Om användaren redan avgett ett bud ersätter det nya budet det gamla. En användare kan alltså bara ha ett bud per auktion.

Exempel: make bid uppdaterat bud

```
Command?> make bid
Enter the name of the user?> Henrik
Enter the name of the dog?> Rex
Amount to bid (min 1)?> 100
Bid made
Command?> make bid
Enter the name of the user?> Henrik
Enter the name of the dog?> Rex
Amount to bid (min 101)?> 200
Bid made
Command?> list bids
Enter the name of the dog?> Rex
Here are the bids for this auction
Henrik 200 kr
```

Om personen som avger budet inte existerar ska ett felmeddelande ges. Var noga med att felmeddelandet innehåller något av orden *fel* eller *error*,

annars kan testprogrammet inte känna igen att det rör sig om ett felmeddelande.

Exempel: make bid när användaren inte finns

```
Command?> make bid
Enter the name of the user?> Rolf
Error: no such user
```

Om hunden inte är till salu ska ett felmeddelande ges. Var noga med att felmeddelandet innehåller något av orden *fel* eller *error*, annars kan testprogrammet inte känna igen att det rör sig om ett felmeddelande.

Exempel: make bid hunden inte är till salu

```
Enter the name of the user?> Henrik
Enter the name of the dog?> Ratata
Error: this dog is not up for auction
```

Om budet är för lågt, alltså lägre eller lika med det nuvarande högsta budet, ska ett felmeddelande ges. Var noga med att felmeddelandet innehåller något av orden *fel* eller *error*, annars kan testprogrammet inte känna igen att det rör sig om ett felmeddelande. Till skillnad från de flesta övriga felmeddelandena ska detta inte avbryta inmatningen utan användaren ska få en ny chans att lägga ett bud. Detta ska fortsätta tills användaren lägger ett godkänt bud.

Det lägsta möjliga budet om det inte tidigare finns något bud är en krona.

Exempel: make bid när budet är för lågt

```
Command?> make bid
Enter the name of the user?> Henrik
Enter the name of the dog?> Milou
Amount to bid (min 201)?> 100
Error: too low bid!
Amount to bid (min 201)?> 200
Error: too low bid!
Amount to bid (min 201)?> 300
Bid made
```

Kommando: ”list auctions”

Detta kommando listar samtliga pågående auktioner. Vid utskriften ska löpnumret på auktionen finnas med, namnet på hunden som ska säljas, samt de tre högsta buden. Buden ska vara sorterade med det högsta budet först.

Tänk på att det inte behöver finnas tre bud registrerade än.

Exempel: list auctions

```
Command?> list auctions
Auction #1: Milou. Top bids: [Anna 200 kr, Henrik 100 kr]
Auction #2: Rex. Top bids: []
```

Om det inte pågår några auktioner ska istället ett felmeddelande ges. Var noga med att felmeddelandet innehåller något av orden *fel* eller *error*, annars kan testprogrammet inte känna igen att det rör sig om ett felmeddelande.

Exempel: list auctions när det inte finns några pågående auktioner

```
Command?> list auctions
Error: no auctions in progress
```

Kommando: ”list bids”

Detta kommando ska lista samtliga bud i en viss auktion. Vid utskriften ska buden vara sorterade med det högsta budet först. Listan ska innehålla namnet på den som gav budet och beloppet.

Exempel: list bids

```
Command?> list bids
Enter the name of the dog?> Milou
Here are the bids for this auction
Anna 200 kr
Henrik 100 kr
```

Om inga bud angetts än ska detta skrivas ut. Observera att detta inte är ett fel, och alltså inte ska betraktas som ett felmeddelande.

Exempel: list bids när inga bud finns

```
Command?> list bids
Enter the name of the dog?> Rex
No bids registred yet for this auction
```

Om hunden som anges inte ligger ute till försäljning ska däremot ett felmeddelande ges. Var noga med att felmeddelandet innehåller något av orden *fel* eller *error*, annars kan testprogrammet inte känna igen att det rör sig om ett felmeddelande.

Exempel: list bids när hunden inte är till försäljning

```
Command?> list bids
Enter the name of the dog?> Karo
Error: this dog is not up for auction
```

Kommando: ”close auction”

Detta kommando stänger en pågående auktion. Användaren ska få frågan om vilken hunds auktion som ska stängas och därefter ska hundens ägare sättas till den som gav det högsta budet. Utskriften från programmet ska innehålla namnet på den som vann auktionen och beloppet.

Exempel: close auction

```
Command?> close auction
Enter the name of the dog?> Milou
The auction is closed. The winning bid was 200kr and was made
by Anna
```

Om inget bud avgetts ska auktionen ändå stängas, men hunden får ingen ny ägare. Detta ska meddelas användaren, men räknas inte som ett fel, så meddelandet ska *inte* innehålla något av orden ”fel” eller ”error”.

Exempel: close auction när inget bud lagst

```
Command?> close auction
Enter the name of the dog?> Rex
The auction is closed. No bids where made for Rex
```

Om hunden inte är till salu eller inte finns ska ett felmeddelande ges. Var noga med att felmeddelandet innehåller något av orden *fel* eller *error*, annars kan testprogrammet inte känna igen att det rör sig om ett felmeddelande.

Exempel: close auction när hunden inte är till salu eller inte finns

```
Command?> close auction
Enter the name of the dog?> Fido
Error: this dog is not up for auction
```

Ändrat kommando: ”remove dog”

Detta kommando fanns med i programmet du lämnade in i del 5, men har något förändrade krav. Det som har förändrats är att det inte räcker med att bara ta bort hunden ur registret. Den måste också tas bort från sin ägare om den har någon eller från auktionen om den är till försäljning.

Detta kommando tar bort en hund ur registret. Användaren ska få en fråga om namnet på den hund som skall tas bort varefter hunden med det angivna namnet tas bort. I den tidigare versionen av programmet räckte det med att ta bort hunden ur `ArrayList`en, men nu måste hunden också tas bort från en eventuell ägare, och från auktionerna om den är till salu.

Du behöver inte hantera komplikationen att det kan finnas flera hundar med samma namn.

Exempel: remove dog när hunden finns

```
Command?> remove dog
Enter the name of the dog?> Karo
Karo is removed from the register
```

Om det inte finns någon hund med det angivna namnet så skall programmet skriva ut ett felmeddelande. Var noga med att felmeddelandet innehåller något av orden *fel* eller *error*, annars kan testprogrammet inte känna igen att det rör sig om ett felmeddelande.

Exempel: remove dog när hunden inte finns

```
Command?> remove dog
Enter the name of the dog?> Ratata
Error: no such dog
```

Ändrat kommando: ”remove user”

Detta kommando fanns med i programmet du lämnade in i del 5, men har något förändrade krav. Det som har förändrats är att det inte räcker med att bara ta bort användaren och dennes hundar ur registret. Man måste också ta bort eventuella bud användaren lagt i auktionerna.

Detta kommando tar bort en användare ur registret. Användaren (av programmet) ska få en fråga om namnet på användaren som skall tas bort varefter användaren med det angivna namnet tas bort.

När en användare tas bort ska även användarens hundar tas bort samt eventuella bud användaren har lagt i alla auktioner.

Du behöver inte tänka på komplikationen att det kan finnas flera användare med samma namn.

Exempel: remove user när användaren finns

```
Command?> remove user
Enter the name of the user> Henrik
Henrik is removed from the register
```

Om det inte finns någon användare med det angivna namnet så skall programmet skriva ut ett felmeddelande. Var noga med att felmeddelandet innehåller något av orden *fel* eller *error*, annars kan testprogrammet inte känna igen att det rör sig om ett felmeddelande.

Exempel: remove user när användaren inte finns

```
Command?> remove user
Enter the name of the user?> Henrik
Error: no such user
```

Del 8 Granskning av klassdiagram

Ett av de överlägset bästa sätten att förbättra kods kvalitet är att granska den. Detta innebär att man låter någon eller några andra personer gå igenom koden noga, ofta med stöd i form av verktyg eller checklistor.

Granskningar kan också göras på andra saker än kod, till exempeldesigndokument, och det är det vi ska göra i detta steg. Vi ska gemensamt granska din kods design och försöka förbättra den. Det är nödvändigt att du har kommit en ganska god bit på ditt program innan granskning är meningsfullt, men det behöver inte vara helt klart. Det är antagligen lämpligt att genomföra en granskning när du är i början av auktionsdelen.

Granskning sker i seminarieform. Datum för dessa granskningsseminarier finns inte i schemat eftersom de olika studentgrupperna går i olika tempon, och det därför finns tillfällen utspridda vid flera tillfällen under kursen. Att lägga in alla dessa i schemat skulle vara förvirrande för dem som inte kommit lika långt.

Inlämning och rättning

Du bokar tid för granskning i Ilearn.

Vanlig fråga: det är fullt i alla seminariegrupperna, jag kan inte boka någon tid! Vad ska jag göra?

Vara ute i bättre tid nästa gång. Det finnas *mycket* gott om tider, flera gånger fler än det går studenter på kursen, så enda möjligheten till att det kan vara ett problem är om man väntar tills sista möjliga tillfället att boka in sig.

Vanlig fråga: måste ändringar som görs efter granskningen testas?

Ja, självklart. Alla ändringar du gör ska testas innan du lämnar in.

Till granskningen ska du ta med en penna och ett klassdiagram för din kod i *två* pappers-exemplar med ditt namn och användarnamn på och plats för anteckningar. Klassdiagrammet ska:

1. vara automatgenererat från din kod,
2. rymmas på en A4-sida (behövs det A3 av någon anledning så kan det antagligen ordnas också, men det bör inte behövas)
3. vara detaljerat, och slutligen
4. vara läsbart utan förstoringsglas.

Om du vill ha hjälp med att skriva ut ditt klassdiagram så kan du skicka in det till en inlämningslåda i Ilearn senast dagen innan. Se till att diagrammet har ditt namn och användarnamn.

Det finns flera verktyg som kan rita klassdiagram direkt från kod. För Eclipse finns till exempel ”ObjectAid UML explorer”. Du hittar det på `http://www.objectaid.com/` och installationsanvisningar på `http://www.objectaid.com/installation`. Observera dock att det bara är klassdiagramsverktyget som är gratis.

IntelliJ har en plugin som heter ”UML Support”. Denna är dock bara tillgänglig i Ultimate Edition. Instruktioner för denna finns på `https://www.jetbrains.com/help/idea/class-diagram.html`.

Tips för denna del

Det finns gott om granskningstillfällen, och det är helt okej att gå på flera om det finns plats och du vill ha feedback på de ändringar som du gjort.

Se till att ha koll på hur klassdiagram ser ut, men tänk också på att det finns många varianter.

Vanlig fråga: vad menas med detaljerat?

Klassdiagrammet ska innehålla all relevant information, i princip vara så detaljerat som verktygen kan göra det. Det ska visa alla klasser, metoder, variabler på klassnivå, skyddsnivåer, relationer mellan klasserna, etc.

Syftet med granskningen är att förbättra designen, inte att mobba någon. Tveka inte att poängtera saker som du undrar över eller tycker är fel, men gör det på ett trevligt sätt, och var öppen för att det finns andra lösningar än dem du har använt.

Anteckna allt som sägs om din design så att du kan gå igenom det senare och se vad som eventuellt bör ändras. Allt som sägs kommer inte att vara relevant, den som granskar kan till exempel ha missförstått någonting i din lösning, eller du har en annan, lika bra idé om hur något ska lösas.

Ta gärna en bild på eller scanna klassdiagrammen om det gjorts anteckningar på dem.

Del 9 Slutrapport

När hela systemet fungerar, och du genomfört en granskning av designen, återstår bara att göra en slutrapport som beskriver hur du byggt upp ditt program och skicka in den.

En mall för rapporten finns i Ilearn på samma ställe som filerna till sorteringsuppgiften fanns. Om du av någon anledning inte kan eller vill använda mallen så finns kraven på rapporten i de kommande två avsnitten.

Format-krav

Följande gäller för rapportens format.

- Stående A4 med normala marginaler och fonter.
- Till vänster i sidhuvudet på varje sida ska ditt namn namn och användarnamn stå.
- Till höger i sidhuvudet ska kurskoden (PROG1) och terminen (HT19) stå.
- Rapporten ska innehålla de sju avsnitt som finns listade nedan *i den angivna ordningen.*
- Varje avsnitt ska börja på en ny sida, och inledas med en numrerad rubrik med samma text som i listan nedan.
- Rapporten ska *inte* ha någon separat framsida, innehållsförteckning, etc.

Vanlig fråga: var ska jag skicka in koden?

Det har du redan gjort i steg 7. Det enda du ska skicka in i detta steg är den efterfrågade slutrapporten.

Innehåll i rapporten

1: Noteringar

Detta avsnitt kan du använda för att ge den som rättar information som är relevant för rättningen. Det kan till exempel vara information om vilka du samarbetat med, eller hur du har tolkat något krav.

Om du får rest och lämnar in rapporten igen så *ska* du här också beskriva vilka ändringar som skett sen den förra versionen av rapporten.

De flesta kan antagligen lämna detta avsnitt tomt om det inte är en restinlämning.

2: Klassdiagram

Ett klassdiagram för den slutliga versionen av din kod, alltså den som du blivit godkänd av VPL i steg 7. Detta ska alltså *inte* vara samma klassdiagram som du tog med till granskningen utan en ny version för ditt kompletta system.

Tänk på att klassdiagrammet måste vara läsbart av den som rättar. Det spelar ingen roll om det blir väldigt litet, bara det i så fall går att zooma in.

3: Skyddsnivåer

En *kort* sammanfattning om varför du har satt de skyddsnivåer du gjort på dina metoder. Du ska inte gå igenom varenda metod utan fokusera på dem som inte är uppenbara. det du ska visa är att du har haft en tanke med de skyddsnivåer du satt och inte bara slentrianmässigt satt dem till `public` för att det fungerar.

4: Korta metoder och lite koduppprepning

Ett par exempel på ställen i ditt program där du har använt dig av metoder för att strukturera ditt program och/eller undvika koduppprepning. Det finns inget fast format för detta avsnitt, men utdrag ur koden tillsammans med en kort förklaring till exemplet är lämpligt. Tänk på att koden ska vara läsbar. Använd inte skärmdumpar utan klipp in koden, och se till att indenteringen bevaras.

Om du har ställen i programmet där du inte är nöjd med metoduppdelningen så kan du också ta upp det och förklara hur det borde ändras.

5: Sortering

Information om vilken sorteringsalgoritm du implementerat för hundarna, var du hittade information om den när du läste på om den, samt den fullständiga koden för sorteringen av hundar. Tänk på att koden ska vara läsbar. Använd inte skärmdumpar utan klipp in koden, och se till att indenteringen bevaras.

6: Arrayanvändning

De metoder i `User`-klassen (eller motsvarande klass om du inte kallat den för `User`) som används för att lägga till och ta bort hundar, samt för de hjälpmetoder som dessa eventuellt anropar. Syftet är att visa att du kunde använda en array för att hålla reda på användarens hundar, så det är viktigt att du har med de metoder som till exempel utökar arrayens storlek när det behövs.

Om klassen har en metod som returnerar arrayen av hundar ska denna metod också finnas med.

Som vanligt: tänk på att koden ska vara läsbar. Använd inte skärmdumpar utan klipp in koden, och se till att indenteringen bevaras.

7: Kommentarer till testprogrammets feedback

Testprogrammets feedback för det fulla programmet (steg 7, samma som för klassdiagrammet). Feedbacken från testprogrammet är indelad i numrerade steg och du ska ta med all feedback från steg 7 (flera scanners) till steg 20 (wrappertyper).

För varje steg där testprogrammet gett någon annan feedback än att kontrollen lyckades ska du kommentera denna feedback. Kommentaren ska *tydligt* visa varför du gjort rätt som ignorerat testprogrammets feedback.

Vanlig fråga: Ska jag plocka bort de steg kontrollen lyckades?

Nej, allting testprogrammet skrev för steg 7 till 20 ska finnas med, även om det bara ser ut så här:

```
Leta efter flera Scanner-objekt (steg 7 av 74, tid 0 sek.)
Lyckad kontroll: inga problem.
```

Tänk på att feedbacken ska vara läsbar och att det tydligt ska framgå vad testprogrammet sagt och vad som är dina kommentarer. Använd inte skärmdumpar utan klipp in feedbacken som text, och se till att rubrikerna bevaras.

Det enklaste stället att kopiera feedback-texten från är antagligen edit-fliken i VPL.

Inlämning och rättning

Det som ska lämnas in här är slutrapporten. Denna ska lämnas in i Ilearn i pdf-format. Rapporten rättas efter deadline. Det spelar alltså ingen roll om du skickar in den långt i förväg eller i sista sekund, den kommer att rättas vid samma tillfälle.

Tips för denna del

Se till att allt som efterfrågas i mallen finns med och är läsbart. De är de vanligaste orsakerna till rest. Om det är något du känner dig osäker på så skicka en fråga till handledningsforumet i Ilearn eller prata med en handledare.

Bilagor

Litteraturförteckning

[1] J. Glenn Brookshear and Dennis Brylow. *Computer Science – An Overview*. Pearson, 13 edition, 2019.

[2] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Prentice Hall, 3 edition, 2004.

[3] John Lewis and William Loftus. *Java Software Solutions*. Pearson, 2017.

[4] Y Daniel Liang. *Introduction to Java Programming and Data Structures, Comprehensive Version, Global Edition*. Pearson, 2018.

[5] Jan Skansholm. *Java direkt med Swing*. Studentlitteratur, 2014.

[6] Sun Microsystems. How to write doc comments for the javadoc tool. `https://www.oracle.com/technetwork/articles/java/index-137868.html`.

[7] Sun Microsystems. Code conventions for the Java™ programming language. `https://www.oracle.com/technetwork/java/codeconvtoc-136057.html`, 1999.

Sakregister