U7.6 Sortera hundarna

Henrik Bergström Olle Karlsson Patrick Wentzel

Uppgiften

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.

Algoritmen du ska implementera är urvalssortering (selection sort, avsnitt 7.11 i kursboken). Detta är inte en effektiv sorteringsalgoritm, tvärt om, det är en av de minst effektiva, men i gengäld en av de enklaste att förklara.

Antag att du har N element i en lista eller array som ska sorteras. Spara det första elementet (eller dess index) i en variabel. Jämför denna variabel med alla andra element i samlingen. Varje gång du hittar ett ”mindre” element, alltså ett som kommer före i sorteringsordningen, så uppdaterar du variabeln. När du har gått igenom alla elementen innehåller variabeln det ”minsta” elementet i hela samlingen.

Om detta element inte är samma som redan står på första platsen så byter du plats på dem. Det minsta elementet i hela samlingen står nu först.

Upprepa samma sak, men börja nu på det andra elementet i samlingen. När du är klar är de två första elementen sorterade. Fortsätt sedan att göra samma sak för alla övriga element i samlingen, förutom det sista. När du kommer till det så är det redan korrekt sorterat eftersom det föregående elementet jämfördes med det.

Denna uppgift går ut på att implementera ovanstående algoritm i en metod för att sortera hundar i en `ArrayList`. Du kommer att använda denna metod i det slutgiltiga programmet i uppgift 11.1 för att sortera hundarna vid utskrift i kommandot `list dogs` som ska lista alla hundar i sorterad ordning. Det är fullt möjligt att implementera algoritmen i en enda metod, och det skulle antagligen vara det normala eftersom algoritmen är så pass liten. Här ska vi dock passa på att öva på hjälpmetoder genom att dela upp algoritmen.

Samtliga metoder nedan ska ligga i en klass som heter `AssignmentSevenPointSix`, och vara märkta med `@UnderTest(id=''...'')` där ... ersätts med uppgiftsnumret i rubriken för respektive metod.

Förutom metoderna ska klassen `AssignmentSevenPointSix` innehålla en `ArrayList`:a av hundar märkt med `@UnderTest(id=''dogs'')`.

Då det är möjligt att viss funktionalitet hamnar i hundklassen måste du denna gång också lämna in den. Däremot ska du inte lämna in inläsningsklassen. Metoderna i denna uppgift ska inte kommunicera med användaren.

Byta plats på två hundar

Det enklaste steget i algoritmen är att byta plats på två hundar i listan, så låt oss börja med det. Det finns två alternativ vi kan använda oss av här: vi kan implementera funktionen själva i en metod, eller använda oss av en redan existerande metod från klassbiblioteket. Vi ska testa att bägge dessa alternativ. Vilket du sedan använder för att implementera sorteringsalgoritmen är upp till dig.

U7.6.1.1 Egen metod för att byta plats

Implementera en metod som byter plats på två hundar i en `ArrayList`. Metoden ska ta indexen för hundarna som ska byta plats som parametrar. Du får inte använda `Collections.swap` för att lösa denna deluppgift. Detta kontrolleras inte av testprogrammet i ilearn, utan kommer att kontrolleras manuellt i efterhand.

U7.6.1.2 Använda en metod från klassbiblioteket

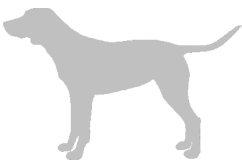
Implementera samma metod en gång till, men nu med hjälp av en existerande metod i klassbiblioteket. Att byta plats på element i en lista är en vanlig operation, och det finns redan en sådan metod implementerad i klassbiblioteket.

Metoden för att byta plats på två element i en lista finns inte som instansmetod i `ArrayList`, utan är en statisk klassmetod i klassen `Collections`. Leta reda på metoden i dokumentationen för `Collections`-klassen[1] och använd den för att implementera din metod.

U7.6.2 Jämföra två hundar

Vårt slutliga mål är att sortera listan av hundar, och för att kunna göra det behöver vi kunna jämföra två hundar och se vilken som ska komma först i den sorterade

1 `https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Collections.html`



listan. Hundarna ska vara sorterad efter svanslängd i stigande ordning, men om två hundar har samma svanslängd ska de istället vara sorterade efter namn i bokstavsordning. Hela namnet ska användas i det andra fallet, och stora och små bokstäver ska inte spela någon roll för jämförelsen[2].

För att kunna göra detta behöver vi en metod som jämför om två hundar ligger i rätt ordning, eller om de behöver byta plats. Denna metod kan se ut på ganska många olika sätt, och testprogrammet gör sitt bästa för att gissa sig till hur den fungerar från placering, parametrar och returvärden. Det är dock möjligt att testprogrammet inte förstår hur du har tänkt att metoden ska fungera. Om så är fallet, skicka ett meddelande till handledningsforumet och motivera din lösning. Om den är korrekt kommer vi att lägga till den till de som testfallen accepterar.

Ett starkt tips om du känner dig osäker är att inte skriva en enda metod för jämförelse, utan tre: en som bara jämför hundarnas svanslängd, en som bara jämför hundarnas namn, och slutligen en som använder de andra två för att jämföra på både svanslängd och namn. Det är bara den sista av dessa som är obligatorisk, men det finns testfall för alla tre i ilearn.

Placering av metoderna för jämförelse

Det finns två möjliga placeringar av metoderna för jämförelse: i klassen där sorteringsmetoden ska ligga, eller i `Dog` eftersom det är hundar som ska jämföras. Den viktigaste skillnaden mellan dessa placeringar är antalet parametrar till metoden. Om jämförelsemetoden placeras i samma klass som sorteringsmetoden så behöver den få bägge hundarna skickade till sig. Om metoden istället placeras i `Dog` har den redan tillgång till en av hundarna via objektet metoden anropas på och behöver bara få reda på vilken annan hund den nuvarande hunden ska jämföras med. Du får själv välja vilket av dessa sätt du vill göra på, men ett tips är att testa på bägge.

U7.6.3 Hitta den minsta hunden

När vi nu kan jämför två hundar är det dags att skriva en metod som hittar den ”minsta” av de kvarvarande hundarna, alltså den som ska stå först i den sorterade listan. Testkoden i ilearn förutsätter att metoden tar en `int` som parameter, och returnerar en `int`. Varför det? Vad representerar dessa heltal? Fundera på detta och varför vi inte, till exempel, kan returnera en hund istället innan du sätter igång med metoden. Känner du dig osäker kan det vara läge att prata med en handledare.

2 Ett tips som underlättar jämförelsen är att alltid normalisera namnen direkt vid inläsningen, se sidan V.

U7.6.4 Implementera sorteringsalgoritmen

Det sista steget är att implementera själva sorteringsmetoden med hjälp av de metoder vi implementerat eller testat ovan. Metoden ska returnera hur många byten som gjordes under sorteringen. Denna information används av testprogrammet för att avgöra om rätt algoritm är implementerad.

Andra sorteringsalgoritmer

Vi sa i inledningen att urvalssortering är en av de minst effektiva sorteringsalgoritmerna som finns. För dem som gått IDSV och är bekanta med ordo-begreppet så är den $O(N^2)$. Andra enkla sorteringsalgortmer som också är $O(N^2)$ är bubbelsortering (bubbel sort) och insättningssortering (insertion sort).

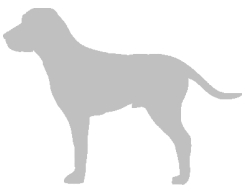
Exempel på mer avancerade, och betydligt mer effektiva, sorteringsalgoritmer är quicksort och mergesort som bägge är $O(NlogN)$. Skillnaden kan vara otydlig för den som inte är van vid logaritmer, men om vi ska sortera tusen element så behöver de enkla sorteringsalgoritmerna göra i storleksordningen en miljon steg, medan de avancerade bara behöver tio tusen. Varje steg tar förvisso längre tid, men skillnaden blir större och större ju fler element som ska sorteras. Ska vi sortera en miljon element behöver de enkla sorteringsalgoritmerna göra runt biljon steg, medan de effektiva klarar sig med något tiotal miljoner. Det är en enorm skillnad, till och med för en snabb dator.

En utmärkt övning inför tentan är att läsa in sig på någon av ovanstående algoritmer och implementera denna. Förutom urvalssortering [2, avsnitt 7.11] tar kursboken upp sortering på flera andra ställen. Indexet för upplaga 12 listar samtliga algoritmer ovan, plus bucket sort och heap sort.

Om du gått IDSV så tar kursboken därifrån ([1]) också upp sortering, och om du behöver mer information så kommer en snabb Google eller YouTube-sökning att ge dig hur många träffar som helst.

Hur man borde gjort

Att implementera sorteringsalgoritmer är vanliga övningar på programmeringskurser, men i verkligheten är det mycket sällan man gör det. Istället används redan implementerade algoritmer från standardbiblioteken. I Java finns implementerade sorteringsalgoritmer på (minst) tre olika ställen: `List` (ett interface som `ArrayList` implementerar), `Collections` (flera varianter) och `Arrays` (flera varianter).



Ingen av dessa metoder får användas för att sortera hundarna i uppgiften. Om du vill sortera något annat får du däremot gärna använda dem.

Obs!

Skillnaderna mellan de olika alternativen är vad de sorterar: i de två första fallen listor, i det sista arrayer, samt hur de jämför elementen som ska sorteras. Det finns två sätt, som motsvarar de två sätt som diskuteras på sidan 3: en jämförelse-metod som är placerad utan för klassen som ska jämföras, och en som är placerad i klassen. I bägge fallen är det frågan om `interface` [2, avsnitt 13.5] som implementeras.

Implementera Comparator

Det första interfacet är `Comparator`[3] som definierar metoden `compare` som tar två parametrar, de bägge objekten som ska jämföras, och returnerar ett tal som beskriver om det första objektet är mindre, samma, eller större än det andra.

Implementera Comparable

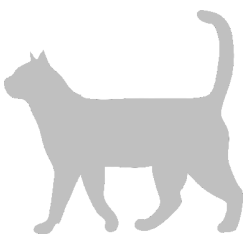
Det andra interfacet är `Comparable`[4] som istället har metoden `compareTo`. Denna metod fungerar exakt som `compare` i `Comparator`, men tar bara en parameter. Den första är ersatt med objektet metoden anropas på.

Comparator eller Comparable

Så, vilken variant ska man använda, `Comparator` eller `Comparable`? Svaret beror på om det som ska sorteras har en naturlig sorteringsordning eller inte. Om den har det använder man `Comparable`, annars använder man `Comparator`. `Comparator` används också om det som ska sorteras är `Comparable`, men man av någon anledning vill sortera på något annat sätt någonstans.

3 `https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Comparator.html`

4 `https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Comparable.html`



Generella krav

Introduktion

Resten av detta dokument ser exakt likadant ut i samtliga beskrivningar av programmets funktionalitet. Det tar upp generella krav som gäller alla delar, och står med i alla dokument bara för att du inte ska behöva leta efter informationen när du behöver den. Läs igenom den en gång ordentligt, och referera sedan till den vid behov.

Inlämning och rättning

Inlämning av uppgift U6.2 till U8.7, samt U11.1, sker i Ilearn och rättas automatiskt. De automaträttade uppgifterna är inställda så att betyget ”Varning(ar)” är tillräckligt för att låsa upp eventuella beroenden. Du ansvarar själv för att avgöra om en varning är ett fel som behöver rättas eller inte.

Det är också viktigt att du har klart för dig att rättningsprogrammet inte har någon intelligens. Det är bara ett program som kör ett antal utvalda scenarier, och det kan komma att uppdateras under kursen om kursledningen upptäcker att det missar något viktigt. I sådana fall kommer alla inlämningar automatiskt att rättas om, och det finns en risk att tidigare godkända uppgifter då underkänns. Testprogrammet är till för att hjälpa dig, men det är du själv som ansvarar för att alla krav är uppfyllda.

Det kan också förekomma fall där automaträttade uppgifter rättas om manuellt för att hantera något fall som de automaträttade testen inte kan täcka.

Övriga uppgifter är antingen självvärderingar, där du själv avgör när du gjort ett tillräckligt bra arbete, eller manuellt rättade.

Paket och moduler får inte användas

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
}
```

Namn och användarnamn i samtliga inlämnade filer

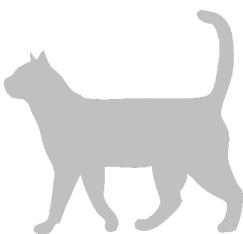
Alla inlämnade filer måste innehålla en kommentar högst upp som innehåller ditt namn och ditt användarnamn. Ett tips är att använda en så kallad JavaDoc-kommentar [3] till klassen i filen. Denna kommer då att hamna nedanför eventuella import-satser. En sådan kommentar skulle kunna se ut ungefär så här:

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. I exemplet ovan är användarnamnet `hebe1234`.



Koden ska vara kompatibel med Java 11

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 kodkonventioner ska följas

De grundläggande kodkonventionerna för Java [4] ska följas. 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`.

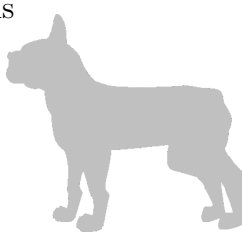
Namn i olika format

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.

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

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: Namn skrivna på olika format

```
Command?> register new user
Name?> peter
Peter added to register
Command?> give dog
Enter the name of the dog?> kArO
Enter the name of the new owner?> pETER
Peter now owns Karo
```

Exempel: Namn skrivet med extra mellanslag

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

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 kravet på att namn ska accepteras i alla versioner 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.

Ledtexter ska alltid följas av ?>

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 kravet?

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.

Felmeddelanden måste markeras

Varje felmeddelande som ditt program skriver ut måste markeras som ett sådant. Detta görs genom att det innehåller ett av orden ”fel” eller ”error” så att testprogrammet kan se att det rör sig om ett felmeddelande. Det är också viktigt att inte använda orden ”fel” eller ”error” i andra sammanhang eftersom testprogrammet då felaktigt skulle tro att det rör sig om ett felmeddelande.

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
```

Privata data och genomtänkta skyddsnivåer på metoder

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

Potentiellt problem: alla metoder publika

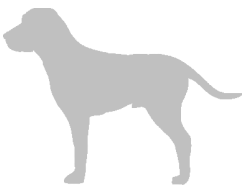
En av de saker vi kommer att titta på i slutinlämningen är vilka metoder som är publika och vilka som inte är det. Du förväntas gå igenom alla metoder och sätta en lämplig skyddsnivå på dem. Det enklaste sättet att uppnå detta är att fundera på skyddsnivån direkt när du skapar metoden.

Potentiellt 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.

Ordentlig metoduppdelning

Det slutgiltiga programmet ska vara ordentligt uppdelat i 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.



Metoddesign kommer att diskuteras på flera föreläsningar och övningar under kursen, och ett syfte med att lämna in varje funktion i programmet för sig är att göra det enklare att uppfylla kravet.

Statiska metoder och variabler

Statiska metoder och variabler (undantaget konstanter) är ovanliga i ett objektorienterat system, och det finns mycket liten anledning att använda dem i denna uppgift. Ett tips är därför att betrakta nyckelordet `static` som nästa förbjudet förutom för `main`-metoden och eventuella konstanter.

Om du använder fler statiska metoder eller variabler i ditt program så kommer du att få motivera deras användning i slutrapporten. Denna motivation måste vara relevant och tydligt visa att den statiska metoden eller variabeln var lämplig att använda i just den situation du använder den i.

Statiska metoder är något vanligare, och lättare att motivera, än statiska varibler. Ofta är det då frågan om metoder som implementerar rena funktioner, alltså inte inte är intresserad av ett objekts tillstånd. För ett exempel se klassen `Math`. Detta är dock sällan en bra objektorienterad lösning.

För den som är intresserad av argumenten mot statiska variabler är denna tråd från Stack overflow bra: `https://stackoverflow.com/questions/7026507/why-are-static-variables-considered-evil`.

Vanlig fråga: behöver jag förstå statiska metoder och variabler?

Ja, absolut. Du kommer att stöta på dem och behöver förstå dem.

Potentiellt problem: jag var tvungen att göra X statisk!

Nej, det var du (nästan säkert) inte! Statiska variabler och metoder är användbara i några situationer, men ganska få, och i hela inlämningsuppgiften finns det bara ett ställe där det är korrekt att använda dem, och ett par till där det går att motivera.

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 method() { variable = 10; }
```


En objektorienterad lösning

Java är ett objektorienterat programmeringsspråk med en imperativ grund. Detta, i kombination med att hela projektet är så pass litet 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 av programmet måste därför vara objektorienterad. Detta är något du bör tänka på redan från början, och arbeta mot kontinuerligt.

Litteraturförteckning

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

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

[3] Sun Microsystems. How to write doc comments for the javadoc tool. `https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html`.

[4] Sun Microsystems. Code conventions for the Java™ programming language. `https://www.oracle.com/java/technologies/javase/codeconventions-contents.html`, 1999.

Hundarna, och katten, i marginalen är från `https://pixabay.com/` och är skapade av Annalise Batista[5] (AnnaliseArt) och gdakaska[6].

5 `https://pixabay.com/users/annaliseart-7089643/`
6 `https://pixabay.com/users/gdakaska-1113303/`