NACKADEMIN

Utbildning *JAVA23 och JAVAD23*

Objektorientering och Java - Övningsuppgifter Sprint 1

Innehåll

Instuderingsfrågor

När Sprint 1 är slut ska du kunna svara på följande frågor:

1. Vad är en klass?
2. Vad är ett objekt?
3. Vad är relationen mellan klass och objekt?
4. Hur fungerar en metod i en klass?
5. Vad är inkapsling? Varför används det?
6. Hur gör man kodmässigt för att kapsla in något i en klass?

7. Vad är en konstruktor och vad används den till?
8. Vad är en default-konstruktor?
9. Hur deklareras en konstruktor?
10. Hur fungerar instansvariabler?
11. Hur används nyckelordet ”this”?
12. Vad skiljer en statisk variabel från en icke-statisk?
13. Vad innebär en ”static ” metod? När används det speciellt?
14. Hur används nyckelordet ”private”?
15. Hur används nyckelordet ”protected”?
16. Hur används nyckelordet ”super”?
17. Hur används annotationen @Override?
18. Vad är skillnaden på överskuggade och överlagrade metoder?
19. Hur deklareras arv?
20. Vad är polymorfism?
21. Vad är dynamisk bindning?
22. Vad skiljer en abstrakt klass från en icke-abstrakt klass?
23. Hur deklareras en abstrakt klass?
24. Vad har man abstrakta klasser till?
25. Vad är ett interface?
26. Vad innebär att en klass implementerar ett interface?
27. Vad är skillnaden på ett interface och en abstrakt klass? Vilka likheter finns?
28. Hur deklareras att en klass implementerar ett interface?
29. Vad skiljer en List från en array?
30. Nämn två olika javaklasser som implementerar interfacet List.
31. Hur deklarerar man en List av objekt?
32. Vad är unikt med klassen ”Object”?
33. Nämn några användbara metoder som finns i klassen ”Object”.

Övningsuppgifter

Tänk på att det inte finns någon enskild ”bästa lösning” på uppgifterna nedan, alla uppgifter kan lösas på olika sätt. Diskutera gärna lösningsförslag med kamraterna i klassen, men skriv dina lösningar själv.

Notera att all kod skall vara prydligt skriven, kommenterad och indenterad.

Uppgift 1a – Arv

Du har fått i uppdrag att konstruera ett bilregeister! I ditt register ska du hålla koll på människor, bilar och vilka av dess människor som äger vilka bilar.

Konstruera en klass Person. En person ska ha ett namn, adress och ålder. Utforma en konstruktor och några lämpliga metoder för Person (getters och setters).
Konstruera en klass Bilägare som ärver Person.

Konstruera en klass Bil med registreringsnummer, modell och märke. En bil ska ha en Bilägare. Konstruera metoder som anropas när man köper eller säljer en bil (alltså byter bilägare).

Uppgift 1b – Arv - accessors

Gå igenom din lösning och fundera på om den följer Best Practices för inkapsling.
Lägg till paket och public/private/protected för klasser/instansvariabler/instansmetoder för bästa möjliga inkapsling.

Testa att ha variablerna både privat och protected och notera vilka skillnader det innebär i övrig kod.

Uppgift 1c – Arv - Huvudprogram

Lägg till ett huvudprogram som skapar upp några bilar och bilägare. Låt bilägarna köpa och sälja ett par bilar i programmet. Skriv, till sist, ut alla bilägarna samt vilka bilar de (eventuellt) äger.

I ditt huvudprogram, se till att du inte använder static mer än i main.

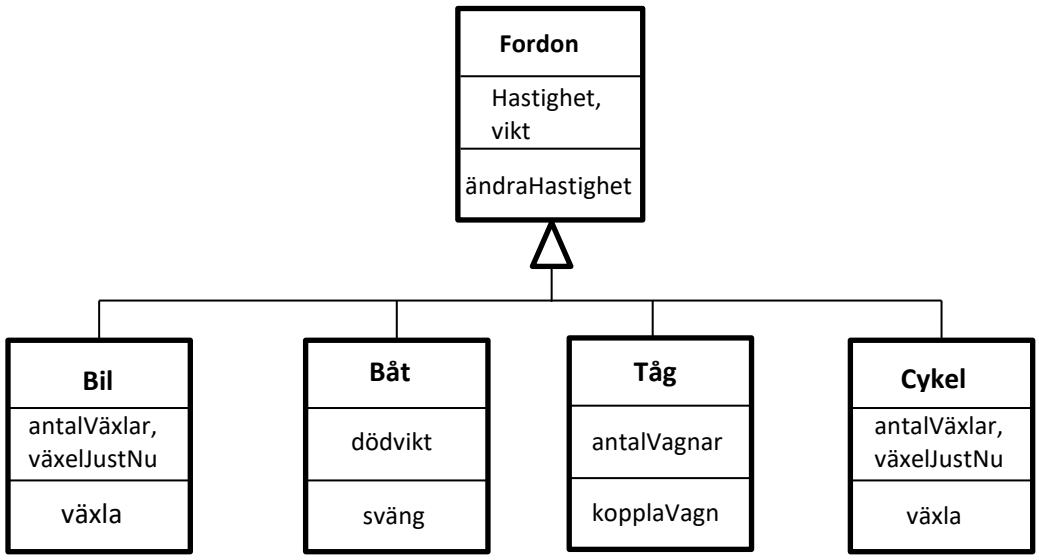
Testa att bryta ut huvudprograms-koden och lägga in den huvudprograms-klassens konstruktor.

Uppgift 2a - Arv

Implementera följande klassdiagram. Detta sätt att rita kallas UML och är vanligt förekommande för att illustrera klasshierarkier. Överst, i varje ruta, står klassnamnet på de klasser ni ska skriva. I mitten finns instansvariablerna. Nederst instansmetoderna i varje klass. Triangeln betyder att nedanstående klasser ärver ovanstående.

Ni ska alltså skriva nedanstående 5 klasser, med de instansvariabler och instansmetoder som anges (och också arvet). Ni behöver inte lägga tid på vettig kod för ”sväng”-metoden.

Testa att använda superkonstruktorn.



Uppgift 2b – Abstrakt superklass, polymorfism

Låt Fordon bli abstrakt
Lägg till den abstrakta metoden printMe() i Fordon och implementera den i de olika subklasserna. printMe() ska skriva ut vikt, hastighet och övriga data för varje subklass.
Lägg sedan till ett huvudprogram som skapar upp ett fordon av varje sort.

Lägg till en metod som skriver ut data om de fordon du skapat.
Anropa utskriftsmetoden från ditt huvudprogram.

Uppgift 2c - Interface, polymorfism

Skriv ett interface Printable och låt metoden printMe vara deklarerad där. Låt sedan dina fordonsklasser implementera Printable.
Gör en ny utskriftsmetod som skriver ut fordonens data genom att anropa printMe via Printable-interfacet, istället för att direkt anropa fordonsklassernas printMe-metod. Lägg till ett anrop till denna nya metod från ditt huvudprogram.

Uppgift 2d - Interface, polymorfism

Lägg till ett ytterligare interface "Hjulburen" som har metoden getAntalHjul()
Låt relevanta klasser implementera interfacet (lägg till instansvariabler vid behov)
Skapa upp några objekt av typen "Hjulburen" med olika implementerande klasser och skriv ut antalet hjul för varje fordon.

Uppgift 3a – Arv och arrayer

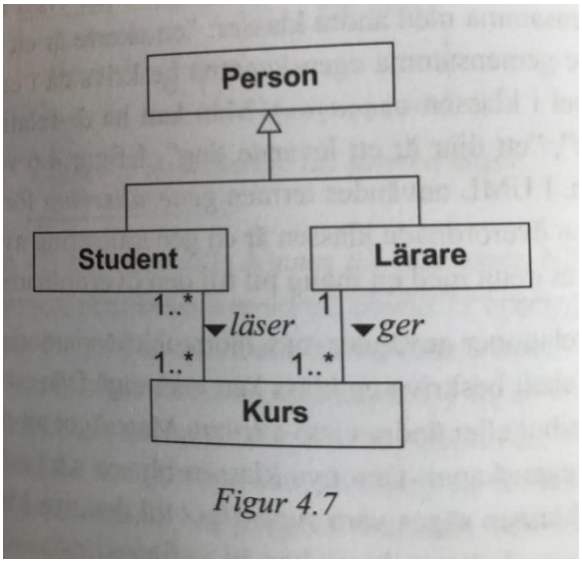
Implementera klassdiagrammet på bilden nedan, dvs skapa klasser som motsvarar varje ruta. Använd dig av arrayer där detta behövs.

Tänk på att:

- en Student ska kunna gå flera kurser.
- en lärare ska kunna ge flera kurser.
- en kurs har flera studenter.
- en kurs har en lärare.

För att din design ska få komma till användning, skapa följande:

- Gör ett huvudprogram där du skapar upp några studenter, en lärare och en kurs.
- Gör en funktion som skriver ut en klasslista för din kurs med kursnamnet, läraren och studenterna.
- I mån av tid: Gör en större utskriftsfunktion där du skriver ut alla studenter och vilka kurser de går. (Skriv först ut namnet på en student och direkt efter kurserna. Fortsätt med nästa student osv.)



Figur 4.7

Uppgift 3b - Listor

Kopiera din lösning från 3a. Byt ut alla arrayer mot listor. Se till att dina klasslistor skrivs ut på ett fint sätt.

Uppgift 3c - Diskussion

Betrakta uppgift 3b ovan. Vi har en situation där en student kan gå flera kurser och en kurs kan ha flera studenter. Om du har byggt din lösning exakt efter hur klassdiagrammet ser ut och inte lagt till andra, egna, klasser innehåller nu klassen student en lista på Kurser och klassen Kurs innehåller en lista på Studenter.

Med andra ord har vi två klasser som refererar till varandra. Är detta en bra eller dålig sak? Varför? Fundera och diskutera.

Uppgift 3d – Refaktorering - Deltagande

Nu ska vi bygga om koden från 3b på ett bättre sätt, där man slipper den trista dubbellagringen av data!

- Ta bort listan av Kurs hos Student och listan av Student hos Kurs
- Skapa istället klassen Deltagande. Ett objekt av typen Deltagande ska innehålla en Student och en Kurs, och markerar därmed att en student går en kurs. Notera hur vi flyttar ut dubbellagringen till en egen klass, och vips så lagras det bara på ett ställe. WIN!!!
- Skapa upp objekt av Kurser och Studenter. Skapa sen objekt av Deltagande där du kopplar ihop vilka studenter som går vilka kurser. Lägg alla Deltaganden i en lista så att man lätt kan iterera över alla Deltaganden.
- Skapa en printningsfunktion som tar en Student och sedan skriver ut alla kurser som studenten deltar i.
- Skapa en printningsfunktion som tar en Kurs och sedan skriver ut kursens lärare och en lista av alla Studenter som går kursen.
- Skapa en ”jätteutskriftsfunktion” som skriver ut samtliga studenter och efter varje student listas alla kurser som studenten går på.

Uppgift 4a - Kurstillfälle

Antag att vi på Nackademin har ett system som i uppgift 3. Vi vet vilka lärare som undervisar på en kurs och vilka elever som är registrerade på kursen.

Men nu behöver vi lägga till stöd för att registrera närvaro varje gång ett kurstillfälle äger rum. Kopiera koden till uppgift 3 och modifiera den till att även innehålla kurstillfälle.

Börja med att fundera över hur kurstillfällen skulle kunna uttryckas i kod och var de behöver stoppas in i den befintliga designen. Rita sedan ett klassdiagram med den nya designen.

Ett tips är att använda sig av Javas klass LocalDate, för att hantera datum.

Uppgift 4b - Refaktorering, Separation of Concernes

Implementera designen från 4a.

Uppgift 4c - Huvudprogram

Lägg till kod där du skapar upp ett kurstillfälle, några studenter, en lärare och en kurs. Lägg till en funktion som skriver ut en klasslista för ditt kurstillfälle med kursnamnet och lärarens namn först, följt av alla studenters, som ska vara med på kurstillfället, namn.

Uppgift 5 – Git

- Ladda ner git och installera på din dator
- Skaffa konto på GitHub,
- I IntelliJ, skapa ett projekt. Skapa en klass i projektet.
- Gör projektet versionshanterat med git.
- Adda din klassfil. (sker automatiskt)
- Committa klassfilen.
- Ändra i filen och committa igen.
- Pusha nu ditt projekt till gitHub. Kolla att filen pushas upp
- Ändra i filen igen. Committa, pusha, kolla att ändringarna skickas upp

Uppgift 6 – Polymorfism

Olika sorters djur kan beskrivas med hjälp av klasser och arv. Definiera några olika djur. Utgå från en abstrakt klass Djur och härled de olika djuren från denna. Använd gärna mellanliggande klasser som t.ex Däggdjur, Kräldjur, Fågel, Fisk etc.
Låt klassen Djur ha en abstrakt metod ”läte” som beskriver hur ett djur låter. Implementera denna metod i subklasserna. Deklarera sedan en array eller lista med djur och skriv satser för att löpa igenom arrayen och låta alla djuren ge ifrån sig ett läte.

Uppgift 7a – Listor, arv, generics

Definiera en klass Punkt som beskriver en punkt (x,y) i ett tvådimensionellt koordinatsystem. Deklarera sedan en abstrakt klass ”Figur” som innehåller en startpunkt. Låt klassen Figur ha en abstrakt metod, ”Area”.

Definiera ett antal subklasser till Figur, t.ex. Cirkel, Liksidig Triangel, och Rektangel. Ge dem lämpliga instansvariabler. Låt varje subklass ha en egen implementation av metoden Area.

I ditt huvudprogram, skapa några olika figurer. Deklarera slutligen en lista av ”Figur” enligt följande:

```
List<Figur> figurlista = new LinkedList<>();
```

Iterera över listan och skriv ut vilken sort varje figur är av och figurens area.

Uppgift 7b – Listor, arv, klassen Object, cast

Skapa en ytterligare lista, men för denna lista ska du INTE ange någon typ. Deklarationen kan t.ex se ut enligt följande:

```
List figurlista = new LinkedList();
```

Ändra din utskrift så att den skriver ut samma sak som innan. Vad är skillnaden?

Uppgift 8 – Factory pattern

Kopiera Uppgift 7 till ett egen projekt.

Implementera ett factory pattern i din kod.

Tips: Detta kan göras genom att du lägger till en klass ”Figurfabrik” med metoden getFigur(String form). Beroende om du skickar in strängarna ”rund”, ”trekantig” eller ”fyrkantig” ska du få ut antingen en cirkel, triangel eller rektangel från getFigur().

I din kod, låt fabriken skapa några figurer. Lägg de figurer du fått från fabriken i en lista och skriv ut deras areor som i Uppgift 7.

Uppgift 9 – Enums

De romerska siffrorna anges med bokstäverna I, V, X, L, C, D och M som står för 1, 5, 10, 50, 100, 500 resp. 1000.
Deklarera en uppräkningstyp Romersksiffra som innehåller uppräkningsvärdena I, V, X, L, C, D och M. För varje uppräkningsvärde ska det finnas en instansvariabel som innehåller motsvarande siffervärde.
I ditt huvudprogram, iterera över konstanterna i uppräkningen och skriv ut vad varje romersk siffra har för värde.

Uppgift 10 – Bankomaten (finns inte i facit)

Ni har fått i uppgift att designa ett banksystem.
Systemet behöver hålla reda på följande

- Vilka kunder banken har
- Bankens personal, med personuppgifter och lön
- Vilka konton och lån varje kund har (en kund kan ha flera konton och lån)
- Räntesats och saldo på varje konto
- Räntesats och belopp på varje lån
- Det ska gå att se vilken anställd som har beviljat varje lån
- Det ska finnas historik över ränteändringar och vilken anställd som beviljade ändringen

Gör diagram över vilka klasser och interface som behöver finnas
I varje klass, skriv ut vilka instansvariabler och instansmetoder som ska finnas. Markera om de är private/public eller protected

Implementera objektmodellen till ditt system, alltså, skriv alla klasser och interface i kod.

Vi kommer att bygga vidare på denna uppgift i varje sprint!

Uppgift 11a – Test

Fixa i ordning ditt projekt för Sprint 1 så att du kan ha enhetester.

- Skapa en testkatalog med relevant paketstruktur
- Se till att testkatalogen är grönmarkerad

Uppgift 11b – Test

I uppgift 1 skrev du ett Bilregister-system där bilar kunde byta ägare.
Skriv tester för de metoder du har i ditt system.

Tänkbart upplägg:

- Skapa paket och klasser i testkatalogen som motsvarar Övningsuppgift 1
- I din testklass, skapa några Bilar och Bilägare
- Verifiera att det går att ge en bil en ägare, **genom test**, t.ex. kan bil A ägas av en Bilägare B.
- Låt bli A byta ägare, till ägare C
- Verifiera, **genom test**, att bil A nu ägs av Bilägare C.
- Se till att alla tester går gröna

Beroende hur ni har skrivit ert system kan testerna se lite olika ut.

Uppgift 11c – Test

Skriv tester som verifierar att uppgift 9 (enums) funkar.
Gör tester för samtliga romerska siffror som finns i din enum och verifiera, genom test, att rätt siffervärde är kopplat till rätt romersk bokstav.

Uppgift 11d – Test

I uppgift 3a skapade vi upp massa Personer, Kurser, Studenter, Lärare m.fl och skrev ut dessa.
Titta på utskrivts-funktionen i Sigruns facit. Varför är denna problematisk att testa automatiskt?

Hur skulle man, på ett enkelt sätt, kunna skriva om den för att göra den mer testbar?

Uppdatera er egen uppg 3 så att funktionaliteten blir testbar (om den inte redan är det). Skriv test för detta.

Extrauppgift: Hederlige Harrys Bilar (utan interfaces)

- Vi ska skapa en grund för att hålla reda på vilka fordon som ska listas på Hederlige Harrys Bilar AB:s webbsajt

- Vi börjar med FordonsAnnons. Alla fordon bör ha ett pris, en rubrik, en beskrivning, årsmodell, antalmil etc etc Dessutom ska vi ha en funktion string GetAnnonsText() som ska generera själva annonstexten som ska visas, tex en grym Volvo 240 Årsmodell 1981, Röd, 49000 mil 249000 kr En perfekt bil för den händige till ett fantastiskt pris.
- Skapa nu en klass BilAnnons som ärver från FordonsAnnons. Den ska ha lite mer information, tex Färg, Sommardäck (J/N), Vinterdäck (J/N)
- Skapa nu en klass HusvagnsAnnons som ärver från FordonsAnnons. Den ska ha lite mer information, tex Dusch J/N, Antal bäddar etc
- Och sist en klass MotorCykelAnnons som också ärver från FordonsAnnons. Den kan ha Motorvolym (ex 800 cm3), Drivtyp (kardan/kedja)
- Gör override på GetAnnonsText() och se till att det blir ”snygga” annonser för respektive typ.
- Skapa upp några annonser och skriv ut!

Extrauppgift: Hederlige Harrys Bilar (med interfaces)

- Istället för att GetAnnonsText ärvs från super-annons-klassen, skapa ett interface, Publishable, som annonsklasserna får implementera
- Publishable innehåller 2 metoder:
    - PrintHeader (ska printa rubrik och pris, typ: ”Fantastisk husbil, 25000 kr”)
    - PrintCompleteAd (ska printa all info för en annons)
    - (vi tänker oss att PrintHeader är den metod som anropas när vi vill lista alla fordon på Hederlige Harrys hemsida, PrintCompleteAd är den som anropas när en användare har klickat på en annons-header)
- Lägg till ytterligare ett interface, Revenuable, som innehåller en metod calcuateRevenue som räknar ut hur mycket Harry kan tjäna på en annons (förutsatt att nån köper). Vi måste dra av 25% av alla priser pga moms och sen är det dessutom 10% rea på motorcyklar.

För dig som vill ha mer

Googla på fler ”design patterns” och läs. Fundera på om något mönster skulle kunna användas för uppgifterna ovan. Testa att implementera.

Facit

Sigruns lösningar: https://github.com/sigrunolafsdottir/OOPJavaSprint1_IntelliJ
Jan Skansholms lösningar: http://skansholm.com/java_dir/losn8/