NACKADEMIN

Utbildning *JAVA23 och JAVAD23*

Objektorientering och Java - Övningsuppgifter Sprint 2

Innehåll

Kurslitteratur

Skansholm: I denna Sprint1 ingår kapitel 2.12, 5, 9.6, 11 och 16.

Övningsuppgifter

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

Uppgift 1a – Geometriska figurer, TDD

Skriv ett program som representerar olika geometriska figurer. Börja med att göra ett interface Figure som har två funktioner, getArea och getCircumference (omkrets). Alla dina klasser ska senare implementera detta interface.

Uppgift 1b – Geometriska figurer, square

- Skapa testklassen **SquareTest** i din testkatalog
- Skapa klassen **Square** som implementerar Figure
- Låt Square ha length och width (integers)
- Skriv tester getAreaTest() och getCircumferenceTest()
- Skriv sedan funktionerna som räknar ut Area och omkrets
- Verifiera att testerna går gröna

Uppgift 1c – geometriska figurer, Triangle

Gör tester för klassen Triangle som representerar en likbent triangel. Skriv sen klassen och ge den en höjd och en bredd. Låt den implementera Figure. Följ i övrigt punkterna i 1b, skriv tester först, implementera sedan funktionalitet i Triangle-klassen.

Uppgift 1d – geometriska figurer, Cirlce

Gör samma sak för klassen circle.

Uppgift 1e – att fundera på

Går det att testa interface?

Uppgift 2 – Bensinförbrukning, TDD

Skriv ett program som beräknar och skriver ut hur många mil en bil har gått under det senaste året och dessutom beräknar bilens genomsnittliga bensinförbrukning. Indata till programmet ska vara dagens mätarställning, mätarställningen för ett år sedan samt antal liter bensin som förbrukats under året. Använd klassen JOptionPane för att läsa in data.

Utskriften ska ske i kommandofönstret och ha format enligt följande:

Antal körda mil: 1487.0
Antal liter bensin: 1235.4
Förbrukning per mil: 0.83

Skriv testerna innan du skriver koden. **Du ska minst ha gjort tester för de beräkningar som görs i programmet och för att utskrifterna följer specifikationen ovan.**

Uppgift 3 – Härmapa

Gör ett program där du läser in det du själv skriver på kommandoraden, och skriver ut det igen
Varje gång du trycker “enter” ska inläsning av hela den rad du skrivit läsas in med en Scanner.
Skriv ut “Du skrev: ” följt av det du faktiskt skrev. Du behöver inte jobba testdrivet i denna uppgift, den är bara till för att du ska få scannern att funka.

Uppgift 4 – Bensinförbrukning med Scanner

Kopiera uppgift 2 (inklusive tester). Du ska nu refaktorera ditt program så att du använder Scanner, som läser från System.in, istället för JOptionPane.
Utskriften ska ske i kommandofönstret och ha formatet:

Antal körda mil: 1487
Antal liter bensin: 1235,4
Förbrukning per mil: 0,83

Använd dig av testerna för att kontrollera att du inte förstört något i programmet under refaktoreringen.

Uppgift 5 – Exceptions

Kopiera koden Bilhyra2 från:
http://skansholm.com/java_dir/exempel6/BilHyra2.java
Lägg till den felhantering som behövs.
Fel som behöver fångas upp är

- Om användaren skriver in något annat än siffror där programmet läser in int eller double
- Om användaren skriver in för få parametrar och trycker ”OK”.

Använd exceptions.

Uppgift 6a – Växel tillbaka med TDD

Skriv ett program som beräknar hur mycket växel som ska ges tillbaka.

Indata till programmet är det pris man ska betala samt det belopp man betalar med. Läs och skriv till/från kommandoraden.

Beräkna hur många 1000-lappar, 500-lappar, 200-lappar, 100-lappar, 50-lappar, 20-lappar, 10-kronor, 5-kronor, 2-kronor och enkronor man ska få tillbaka utifrån, av användaren, givna belopp.

Växeln ska alltid innehålla så mycket som möjligt av högsta möjliga valörer

Skriv helst testerna först, koden sen**.** Om det känns **väldigt** svårt är det bättre att fuska med detta än att inte skriva nån kod alls

Ledtrådar:
Innan du sätter igång, fundera över vad som måste hända:

- Du måste räkna ut växeln (utifrån pris och lämnade pengar)
- Du måste hålla reda på vilka valörer som finns
- Rekommendation är att lägga dessa värden i en lista som du itererar över när du räknar ut hur mycket av varje valör som ska ges tillbaka
- Du måste räkna ut hur många enheter av en valör som ska ges i växeln
- När du har räknat ut hur många antal (y) av valör x som ska ges i växel måste du subtrahera dessa från växeln inför nästa iteration Ex: antag att växeln är 654, då ska du ge 1st 500-lappar, sen måste du dra av 500 innan du räknar ut hur många 50-lappar som ska ges

Alla dessa punkter lämpar sig väl för att skriva enhetstester på!!!

Uppgift 6b – Växel tillbaka, TDD, formattering

När räkneverket funkar är det dags att fixa snygga utskrifter**.** Fixa din lösning så att varje rad ser ut enligt följande:

Antal 20-lappar: 3
Antal 10-kronor: 1

- Du ska skriva ut ”kronor” när valörerna är 1,2,3,5,10.
- Du ska skriva ut ”lappar” när valörerna är 20, 50, 100, 200, 500, 1000**.**
- Om användaren betalar med jämna pengar ska en utskrift som säger ”Det blev ingen växel” skrivas ut
- Om användaren betalar för lite ska en utskrift som säger ”Du lämnade för lite pengar” visas
- Skriv tester för ovanstående krav och implementera sen koden

Uppgift 6c – StringBuilder

I din lösning till uppgift 6a, ändra så att svaret skrivs ut mha en StringBuilder. Testerna ska inte behöva ändras iom. denna ändring.

Uppgift 6d – StringBuilder

Dags att lägga till tester för inläsnignen**, implementera följande:**

- Lägg till en publik test-parameter så att ni kan skilja på om ni befinner er i test-läge eller inte.
- Skriv tester som testar er inläsning
- Skriv tester som testar att de exception som ni slänger verkligen slängs.
- Formatera om koden så att inläsningen görs i en egen metod
- Se till att alla tester går gröna.

Uppgift 7 – Ränta på ränta

Skriv ett program som visar hur kapital förräntar sig över ett antal år givet startkapital, räntesats och antal år. Uppgiften behöver inte lösas med hjälp av rekursion. Skriv ut resultatet i en tabell med högercentrerade kolumner och där summorna anges med 2 decimaler. Exempel på utskrift för 100 kr insatta med 10% ränta över 10 år:

```
 År         Summa
--------------------
  0        100,00 kr
  1        110,00 kr
  2        121,00 kr
  3        133,10 kr
  4        146,41 kr
  5        161,05 kr
  6        177,16 kr
  7        194,87 kr
  8        214,36 kr
  9        235,79 kr
 10        259,37 kr
```

Uppgift 8a – Inläsning från fil, formattering av text

Ladda ner filen temp.txt från Nackademins portal, den innehåller temperaturer som mätts upp kl 13 på en plats under en månad. Lägg filen i den katalog där du skriver ditt program.
Skriv ett program som läser filen och skriver ut högsta och lägsta värden, samt beräknar medeltemperaturen för månaden.
Använd dig av klassen FileReader wrappad i en BufferedReader för att läsa från filen.

Uppgift 8b – Inläsning från fil, formattering av text

Kopiera uppgift 8a. Modifiera koden så att du använder dig av Scanner för att läsa från filen.

Uppgift 9a – Inläsning/skrivning till fil

Ladda ner filen personuppgifter.txt från Nackademins portal.
Filen innehåller personuppgifter. För varje person står personens namn, adress och på nästa rad personens ålder, längd och vikt. Du ska läsa in filen i ditt program och hitta alla personer som är längre än 2 meter.
Skapa sedan en ny textfil som bara innehåller uppgifterna för de långa personerna.
Både infilens och utfilens namn ska läsas in av programmet.
Använd try-with-resources
Exempel på personpost i infilen:

Kalle Nilsson, Xvägen 1, 12345 Ystad
25, 80, 175

Uppgift 9b – Inläsning, felkontroller

För att jobba på bästa objektorienterade sätt:

- Skapa upp en klass ”Person”
- Tillverka en List<Person> där du skapar upp en instans av Personuppgift för varje person i filen, och lägger in i listan
- Sök igenom listan för att hitta de långa personerna.
- Skriv testerna först och koden sen.

Uppgift 10 – Tid

Skriv ett program som tar en stad och skriver ut vad klockan är i den staden just nu på formatet HH:mm:ss. Implementera detta för åtminstone Toronto, Stockholm och Shanghai (fler städer om du vill). Googla för att hitta namnen på tidszonerna eller deras id:n.

Programmet ska funka både då användaren skriver stadsnamnet med stora, små, eller blandade bokstäver. Det ska funka även om användaren lägger in blankslag efter stadens namn.
Om en användare skriver in en stad som ditt program inte kan visa tid för, visa upp ett lämpligt felmeddelande.

Uppgift 11 – Videobandspelare

Förr när man fortfarande spelade in film på videoband var det ibland svårt att veta om ett kommande tv-program skulle få plats på bandet.
Skriv ett program som hjälper oss räkna ut om ett band kommer att räcka eller inte.
Programmet ska fråga efter videobandets längd (i minuter), hittills använd tid av bandet, den tidpunkt då tv-programmet börjar och den tidpunkt då det slutar (anges som klockslag, tt:mm).

Använd klasserna LocalTime och Duration för att räkna ut tiden. Jobba testdrivet och skriv testerna innan du skriver koden. Om användaren skriver in något konstigt, visa ett felmeddelande.

Uppgift 12– Serialisering

Kopiera koden från uppgift 2 i Sprint 1 (Fordonen).
Gör först en funktion där du skapar en lista av olika Fordonsobjekt, som du sparar ner till fil via Serialisering.
Sedan, i huvudprogrammet, radera de Fordonsobjekt du skapar upp.
Gör sedan en deserialiseringsfunktion där du läser upp listan av Fordonsobjekt från filen och skriver ut.

Uppgift 13a – Kalkylator, Inläsning data

Användaren skriver in ett uttryck på formen ”a ? b” där a och b är flyttal och ? är antingen +, -, * eller /. Resultatet av uttrycket skrivs sedan ut.
Börja med inläsningen, mha Scanner, läs in
a till en int-variabel
? till en String- eller char-variabel
b till en int-variabel
Ni väljer om ni läser från System.in eller från en JOptionPane
Kolla att ? är ett tillåtet tecken (alltså något av +, -, *, /)
Utför uträkningen och skriv ut resultatet

Uppgift 13b – Kalkylator, Inläsning data

Vi vill gärna skilja på det rent matematiska och inläsning/utskrift
Bryt ut själva uträknandet till en egen klass (Calculator.java).
Låt uträkningen ske här
Låt även felkontrollen av operator ske här
Skapa en egen exception-klass OperatorNotSupportedException som slängs när användaren skriver in en operator som inte stöds. (Exceptions är helt vanliga klasser, vad de har gemensamt är att de ärver javas klass Exception, vilket gör att de kan kastas då Exception ärver klassen Throwable)

Uppgift 14 – Bankomaten

Utöka datamodellen till ditt banksystem. Om det inte redan finns, lägg till datum för när varje konto skapades och när varje lån utfärdades. Alla ändringar behöver också ha datum.

Skapa filer som har datat för dina kunder, personal, konton och lån.

Gör ett console-program som vänder sig till bankens kunder. De ska få se en meny som de sedan kan få välja ifrån. De ska kunna se kontoöversikt, eller ta ut pengar. Väljer de kontoöversikt ska de få se saldo på alla sina konton. Väljer de att ta ut pengar ska de få en fråga om hur mycket de ska ta ut. Sedan ska detta belopp dras av på saldot.

Om du får tid över, gör ett alternativ ”utförlig kontoöversikt” där kunderna kan se ränta på varje konto samt även sina lån med ränta och nästa förfallodatum.

Gör även ett admin-program för bankens personal. Där ska det gå att lista alla kunder, ändra kunduppgifter och ändra räntesats på konton och lån.

För er som vill ha mer

Googla på ”Clean Code” och lär dig mer. Applicera gärna på övningsuppgifterna ovan.

Utforska ”JUnit” och lär dig mer om testdriven utveckling. Applicera gärna på övningsuppgifterna ovan.

För mer allmän kodträning:
Gå in på https://codesignal.com och fighta loss.

Facit

Sigruns lösningar: https://github.com/sigrunolafsdottir/OOPJavaSprint2_IntelliJ