Laboration 2.

Syfte

Syftet med denna laboration är att få övning i att använda färdiga klasser samt konstruera egna enkla klasser.

Redovisning

Uppgifterna skall vara demonstrerade för och godkända av en handledare senast fredag 14/2.

Uppgift 1

Datatypen **int** kan lagra heltal i intervallet [-2147483648, 2147483647] (dvs [-2^{31}, 2^{31}-1]). Om man behöver göra exakta beräkningar med heltal som inte ryms i detta intervall kan man använda klassen BigInteger som finns i Javas API.

Skriv ett Javaprogram som läser in två heltal, som är så stora att de inte kan representeras som **int**, och lagrar dessa som två objekt av typen BigInteger samt skriver ut summa och produkt av talen.

Tips: Slå upp BigInteger i API:n och se efter vilka metoder och konstruktorer som finns att tillgå för att lösa uppgiften.

Uppgift 2

SI-systemet, dvs det internationella måttenhetssystemet, antogs 1960. De traditionsbundna engelsmännen har förvisso officiellt övergått till SI-systemet, men i praktiken använder de dock fortfarande sina gamla brittiska måttenheter från imperietiden. Din uppgift är att skriva en klass SIUnits för att konvertera några vanliga måttenheter mellan SI-systemet och det brittiska systemet. Klassen skall åtminstone innehålla följande metoder:

meterToYard	för att omvandla meter till yards
yardToMeter	för att omvandla yards till meter
kiloToPound	för att omvandla kilo till pounds
poundToKilo	för att omvandla pounds till meter
literToPint	för att omvandla liter till pints
pintToLiter	för att omvandla pints till liter

Implementera samtliga metoder som klassmetoder (statiska metoder). Förvissa dej om att du förstår varför det är lämpligare att låta metoderna vara klassmetoder istället för instansmetoder.

Skriv också ett huvudprogram för att testa klassen.

Tips: 1 meter = 1.093613 yards
1 kilogram = 2.204623 pounds
1 liter = 1.759750 pints
Dessa förhållanden skall representeras som konstanter i klassen SIUnits.

Uppgift 3

Kopiera klassen Die, som behandlades på föreläsning 6 från kursens hemsida. Utgå från denna klass och skriv *en ny klass* GenericDie för att handha tärningar med *godtyckligt antal sidor*. Klassen skall, förutom andra nödvändiga metoder, innehålla en metod getSides som returnerar hur många sidor tärningen har.

Tips: Behöver GenericDie ytterligare instansvariabler jämfört med Die (dvs behöver vi hålla reda på något mer än vilken sida av tärningen som ligger upp då vi har en tärning med godtyckligt antal sidor)? Behövs GenericDie någon mer konstruktor? Behövs GenericDie några fler metoder? Vad betyder egentligen *ett godtyckligt antal sidor*?

Uppgift 4

Spelet Hundra tillgår på följande sätt:

> Spelaren kastar en tärning upprepade gånger och summerar poängen på tärningen. Om spelaren hamnar exakt på 100 poäng är spelet slut. Om vid ett kast den dittills uppnådda poängsumman och poängen i detta kast överskrider 100, adderas *inte* kastets poäng till poängsumman. För att spelet skall ta slut skall summan av poängen och den dittills uppnådda poängsumman bli *exakt* 100.

Skriv ett program som genomför 10 omgångar av spelet Hundra med en åttasidig tärning och skriver ut hur många kast som erfordrades i varje omgång. Naturligtvis använder du dej av klassen GenericDie från förra uppgiften.

Tips: Börja med den förenklade uppgiften att spela endast en omgång.

Vad är villkoret för att spelaren skall fortsätta kasta tärningarna?

Uppgift 5

På laboration 1 skrev ni ett program för att växla svenska kronor till euro. Nu vill man ju ibland även växla svenska kronor till andra valutor, såsom norska kronor, amerikanska dollar, japanska yen eller ryska rubel. Det är också emellanåt önskvärt att kunna växla dessa valutor till svenska kronor.

Att skriva ett speciellt växlingsprogram för varje valuta som man vill växla till/från verkar något omständligt. Det är smidigare att skapa en klass CurrencyConverter som har egenskapen att kunna *växla mellan två valutor* (t.ex. från svenska kronor till amerikanska dollar och till svenska kronor från amerikanska dollar). Har vi en sådan klass är det bara att skapa *en instans av denna klass för varje par av valutor* vi önskar växla mellan. Din uppgift är att skriva klassen CurrencyConverter.

Klassen behöver en instansvariabel som anger den aktuella växelkursen mellan två valutor (dvs växelkursen från valuta1 till valuta2):

```
private double exchangeRate;
```

Klassen skall ha en konstruktor som har den aktuella växelkursen som parameter:

```
public CurrencyConverter(double exangeRate)
```

Klassen skall också ha följande tre metoder:

public void setExchangeRate(**double** exchangeRate)	som sätter den aktuella växelkursen mellan valuta1 och valuta2 (växelkursen mellan valutor förändras ju som bekant emellanåt)
public double toValue(**double** domesticMony)	som växla från valuta1 till valuta2
public double fromValue(**double** foreignMony)	som växlar till valuta1 från valuta2

Du skall också skriva en main-metod som testar klassen. Utskriften från main-metoden skall vara enligt:

```
100 Skr är 13.42 US$
100 Skr är 10.88 euro
100 Skr är 89.69 norska kronor
100 Skr är 392.00 ryska rubel
100 US$ är 745.12 svenska kronor
100 euro är 919.23 svenska kronor
100 norska kronor är 111.49 svenska kronor
100 ryska rubel är 25.51 svenska kronor
```

Detta betyder att du skall ha *fyra objekt* av CurrencyConverter i main-metoden, som hanterar växling mellan Skr -US$, mellan Skr - euro, mellan Skr - Nkr respektive mellan Skr - rubel.

Uppgift 6

Det nya hälsoinstitutet Svett&Fett tänker ta marknadsandelar från Viktväktarna genom att ta ny teknologi i bruk. Varje gäst på hälsoinstitutet förses med ett smartcard. I utgången från träningshallen skall en våg vara installerad och alla som passerar utgången identifieras via sitt smartcard och vägs. Uppgifterna skall naturligtvis registreras i företagets dataregister.

Du har blivit anlitad för att hjälpa till med att realisera detta. Din uppgift är att skapa en klass Guest som skall innehålla följande:

Instansvariabler

namn	för att hålla reda på gästens namn
vikt	för att hålla reda på gästens vikt
längd	för att hålla reda på gästens längd

Konstruktorer

en tom (dvs parameterlös) konstruktor
en konstruktor för att sätta samtliga instansvariabler till specificerade värden

Instansmetoder

setName	som ändrar namnet
setWeight	som ändrar vikten
setLength	som ändrar längden
getName	som returnerar namnet
getWeight	som returnerar vikten.
getLength	som returnerar längden
isSlim	som returnerar värdet **true** om gästen inte är överviktig och värdet **false** om gästen är överviktig
toString	som returnerar en sträng som skriver ut objektet på formen Kalle Kula 123.7 kg 161 cm

En personens idealvikt beräknas enligt formeln 0.9*(längd-100) och en person betraktas som överviktig om personens vikt överskrider idealvikten med 8% eller mer.

Skriv också ett enkelt huvudprogram för att testa klassen.