Tips för Laboration 3 TDA143

Den här filen innehåller tips för hur ni kan tänka och komma igång med varje uppgift, som är lite för långt för själva uppgiftstexten.

Ett bra tips för alla uppgifter är att börja med att få till ett körbart program som gör *någonting*, även om det inte är ens i närheten av att göra *allting*. Sedan arbetar ni inkrementellt och ändrar stegvis vad programmet gör tills uppgiften är löst.

Att börja med ett program som skriver ut lite saker man kommer behöva är oftast bra.

Uppgifter

Uppgift 1: Stora tal

Börja med att skapa en Scanner och läs in första talet och skriv ut det. Utöka sedan till att läsa in operatorn och andra talet och skriva ut dem också (på var sin rad). Dessa utskrifter ska inte vara kvar i det slutgiltiga programmet, men är användbara för att se att vi förstår vad vi håller på med.

Avgör sedan om operatorn är "+" eller "*" och skriv ut olika saker i de olika fallen, testa att det fungerar.

Slutligen utför ni skälva summan/multiplikationen.

Tips: java.math.BigInteger måste importeras för att använda klassen BigInteger.

Tips: Java-operatorerna + och * fungerar inte på typen BigInteger, istället måste ni leta i API:n efter rätt metoder för att addera/multiplicera medlemmar i klassen.

Tips: Hur gör man för att jämföra strängar? (== fungerar inte)

Tips: Vill ni få snabbare testomgångar kan ni börja med att ha till exempel:
String input = "1000 * 1000";
Sedan ändrar ni så input läses in från en dialogruta. På så vis behöver ni inte göra några inmatningar när ni bara testar koden.

Uppgift 2: Matcha parenteser.

Tipsen här är uppdelade i två delar, en för att hjälpa er förstå algoritmen, den andra för att hjälpa er komma igång att koda och att dela upp kodningen i hanterbara delar. Det är inte nödvändigtvis så att ni måste börja med att förstå algoritmen, man kan börja koda först om man vill.

Förstå algoritmen: Utför algoritmen på papper, tänk er att vi läser in ett tecken i taget. Vad händer vid varje tecken som läses in? Prova algoritmen på några olika strängar. Här är ett exempel på hur algoritmen körs för strängen "(<!>]" (som inte matchar):

Börja med en tom lista, läs in tecken för tecken.

Nytt tecken: '('
Handling: Tecknet är en startparentes, lägg dess slutparentes i listan!
Ny lista: ')' - den tillåtna slutparentesen är nu ')'

Nytt tecken '<'
Handling: Tecknet är en startparentes, lägg dess slutparentes i listan!
Ny lista: ')' '>' - den tillåtna slutparentesen är nu '>'

Nytt tecken '!'
Handling: Tecknet är inte en parentes, ignorera det och gå vidare.

Nytt tecken: '>'
Handling: Tecknet är den tillåtna slutparentesen, ta bort den från listan
Ny lista: ')' - den tillåtna slutparentesen är återigen ')'

Nytt tecken: ']'
Handling: Tecknet är en otillåten slutparentes, returnera false ☹

Gör likadana exempel för ett par andra exempel (det är lättare att göra på papper och penna där man kan lägga till och stryka från listan allteftersom).

Kodning: Börja med att lägg in metoden matches som tar en String. Lägg till return false i metoden så den kompilerar. Lägg till en main-metod som anropar matches, något i stil med:

```
System.out.println(matches("<a>"));
```

Lös alla kompileringsfel som uppstår och kör programmet. Därefter kan ni gradvis förbättra programmet, och köra det för att se att det ni har fungerar som det ska. Nedan är ett antal steg för att komma igång:

1. Gör en loop som går igenom alla tecken i strängen. Börja med att bara skriva ut varje tecken på en egen rad och kontrollera att det fungerar. Gör så att det nuvarande tecknet finns i en variabel kallad c (för character) av typen char.

2. Skriv en metod:

```
private static boolean isStartParenthesis(char c)
```

Som returnerar true enbart om dess parameter är en av de tillåtna startparenteserna.

Ett tips är att definiera en konstant som är en sträng med startparenteser:

```
private static String START = "(<[{";
```

Då kan du skriva isStartParenthesis genom att kontrollera om c finns i strängen.

3. Förfina loopen i `matches` så att den bara skriver ut startparenteser. Så för strängen "<(!)>{}" skriver den ut <, (, {.

4. Skriv en metod:

```
private static char getEndParenthesis(char c)
```

Som tar ett tecken som måste vara en startparentes, och ger dess motsvarande slutparentes som resultat. Ett tips: Det finns en metod i String som heter indexOf, så START.indexOf(c) ger indexet för tecknet c i strängen med startparenteser. Genom att ha en sträng med slutparenteser i samma

ordning kan ni lätt hitta rätt slutparentes (den är på samma index!):

```
private static String END   = ")>]}";
```

5. Ändra loopen i **`matches`** så den skriver ut den slutparenteser varje startparentes förväntar sig.

Exempelvis för strängen "<(!{}" ska den skriva ut >,), } och inget annat.

6. Lägg till en metod som avgör om ett tecken är en slutparentes (som **`isStartParenthesis`**). Använd den för att ändra loopen så den skriver ut alla start och slutparenteser den hittar, så för input "<(!)>" ska den skriva ut något i stil med:

```
start <
start (
end )
end >
```

och inget annat.

7. Skapa en lista i metoden (ArrayList). Ändra loopen så att den lägger till slutparentesen för varje startparentes den hittar i slutet av listan, och skriver ut listan i slutet av varje varv av loopen. För strängen "<(!{}<>" borde den skriva ut något sånt här (notera att listor skrivs ut inneslutna i [...], så ignorera dem):

```
[>]
[> )]
[> )]
[> ) }]
[> ) }]
[> ) } >]
[> ) } >]
```

8. Slutligen ska ni ändra så att programmet tar bort startparenteser när motsvarande slutparentes hittats, och upptäcker när något går fel (fel slutparentes, osv).

Att ni skriver ut listan i varje varv gör att ni enkelt kan se om programmet stämmer med det ni gjort med papper och penna, prova på samma exempel och se att listan uppdateras på rätt sätt i varje steg. När den inte gör det, fundera på vad som behöver ändras i er loop.

Glöm inte städa bort alla utskrifter innan ni redovisar! Att kommentera bort debug-utskrifter som inte behövs är en bra vana, då kan man kommentera in dem igen ifall de skulle behövas.

Uppgift 3: Avståndsenheter

a) Fundera på: Vad är typen av FOOT och METER? Det är inte tal. Kolla på följande anrop från testklassen:

```
DistanceUnit.METER.toMetric(100)
```

METER har alltså en metod toMetric, så typen på konstanten METER måste vara klassen som innehåller den metoden, alltså DistanceUnit!

Tips: Tal som $3.1 * 10^{-8}$ skrivs i java som: 3.1E-8.

Börja med att få programmet att kompilera, med metoder som bara returnerar 0 eller 1 till exempel. Sedan måste ni fixa konstruktorn så den lagrar undan konverteringsfaktorn i en instansvariabel, som ni använder i konverteringsmetoderna.

b) För att göra konverteringen månste ni först göra om värdet till meter, sedan till den andra enheten (så ni måste använda först toMetric, sedan fromMetric (på 'from' och 'to')

För att implementera `reverseDirection` måste ni returnera en ny `DistanceConverter` (använd `new`). 'from' och 'to' ska vara omvända i den nya konverteraren.

Uppgift 4: Persondatabas

a) Börja med en statisk metod som tar ett personnummer (typen long är lämplig) och returnerar en boolean om det är giltigt eller inte. Som i tidigare uppgifter: Börja med att få den att loopa igenom alla siffror i personnumret och skriva ut dem. Använd heltalsdivision och rest (%) ungefär som ni gjode i Labb 1, fast i en loop (x%10 plockar ut den sista siffran från x. x/10 tar bort den sista siffran).

Ändra stegvis loopen så den skriver ut mer användbara saker och räknar ut kontrollsiffran efter given algoritm. Jämför era utskrifter med beräkningar med papper och penna.

Testa metoden på lite olika personnummer, korrekta och felaktiga.

Tips: Skriv en metod som tar ett tal och returnerar summan av dess siffror. Det räcker om metoden fungerar för tal med två värdesiffor (alltså summerar tiotalet och entalet).

b) Ni kan använda en instansvariabel av typen Person för att hålla reda på vem varje person är gift med. Är man inte gift kan variabeln vara null. Ett bra tips är att lägga till en metod isMarried() så x.isMarried() svarar på om x är gift.

Tänk på att för att gifta två personer, så måste båda personernas instansvariabler ändras så att deras partner-variabel pekar den andra personen. Säg att variabeln heter spouse: Ifall x är gift så fall ska det gälla att x.spouse.spouse == x (alltså att partnern till x partner måste vara x).

Tänk på att ni kan använda nyckelordet `this` för att hänvisa till personen en metod anropas på. Till exempel när metodanropet x.marry(y) körs så kommer `this` hänvisa till samma objekt som x, och otherPerson kommer hänvisa till samma objekt som y. Du kan till exempel köra kod som `otherPerson.spouse = this`, för att sätta instansvariabeln spouse i otherPerson till this.

Tips: Så här kastar man ett exception:

```
throw new RuntimeException("Already married!");
```

Tips: Glöm inte att testa er toString-metod! För att skriva ut en person p räcker det med att skriva System.out.println(p);