1DV506, Problemlösning och programmering, ht 2017

Laboration 4: Exceptions, IO och interface

Problem? Tveka inte att fråga er labbhandledare (eller Jonas på lektionerna) om ni har problem med någon av uppgifterna. Ni kan också använda forumet i Moodle.

Eclipse förberedelser: Skapa ett nytt paket med namnet `DittLnuAnvändarNamn_lab4` inne i Java-projektet `1DV506` och spara alla filer relaterade till denna lab i det paketet.

Lecture 9 - IO and Exceptions

Alla exceptions relaterade till uppgifterna 1 och 2 skall hanteras inom programmet. Vidare, för att lösa uppgifterna 1 och 2 behövs det bara en klass per uppgift (`Histogram` och `RaknaTecken`) som, var och en, innehåller en main-metod. Det kan dock kanske vara lämpligt att dela upp dessa klasser i ett antal metoder.

- **Uppgift 1**
  Skriv ett program `Histogram.java` som läser ett godtyckligt antal heltal från en fil och sedan ritar upp ett histogram (stapeldiagram) för de av talen som är mellan 1 och 100. Notera: Alla talen i filen behöver ej vara inom intervallet [1,100]. En körning kan se ut enligt följande:

```
Läser heltal från filen: C:\Temp\heltal.dat
Antal i intervallet [1,100]: 46
Övriga: 16
Histogram
  1  - 10  | ******
  11 - 20  | ****
  21 - 30  | **
  31 - 40  | ***
  41 - 50  | *******
  51 - 60  | ****
  61 - 70  | ***
  71 - 80  | *********
  81 - 90  | *****
  91 - 100 | ***
```

  Notera 1: Du måste skapa din egen datafil. Vi förväntar oss en vanlig textfil med ett heltal per rad.
  Notera 2: Datafilens sökväg (tex `C:\Temp\heltal.dat`) skall vara indata till programmet (via `String[] args`).

- **Uppgift 2**
  Skriv ett program `RaknaTecken.java` som räknar tecken av olika typer i en text som läses från en fil. Vi vill att ni skall ange hur många tecken som kan hittas i texten av följande kategorier.
  - Stora bokstäver
  - Små bokstäver
  - Siffror
  - "Whitespace" (Dvs, mellanslag, tab och return)
  - Övriga tecken

  Sökvägen till den textfil som ni läser ifrån får vara hårdkodad i programmet. (Vi kommer att testa programmet på en egen textfil.) Ett exempel på en textfil är `HistoryOfProgramming`. Det är en artikel

om programmeringens historia som är hämtad från Wikipedia. Vi rekommenderar dock att ni börjar med en lite mindre text där ni själv kan kontrollera resultatet.

En körning med filen `HistoryOfProgramming` som indata bör ge följande resultat:

```
Antal stora bokstäver: 86
Antal små bokstäver: 3715
Antal siffror: 41
Antal "whitespaces": 728
Antal övriga: 111
```

Om ditt resultat inte överensstämmer exakt med ovanstående ska en skriftlig förklaring till skillnaderna inlämnas tillsammans med labben.

Lecture 10 - Interface och statiska medlemmar

- **Uppgift 3**
  Börja med att skapa ett under-paket med namnet `stack` i paketet `DittLnuAnvändarNamn_lab4`. Spara sedan alla .java-filer relaterade till denna uppgift i det paketet.
  En stack är en LiFo (Last-in, first-out) datastruktur med tre operationer: push, pop och peek. Push lägger in ett element överst på stacken, pop tar bort och returnerar det översta elementet, och peek returnerar (utan att ta bort) det översta elementet. Namnet "stack" kommer från engelskan och betyder "hög" eller "stapel". Stacken kan liknas med en tallriksstapel som kan påträffas i en skolbespisning eller lunchrestaurang. På stapeln kan man endast lägga en tallrik eller ta bort den översta – det går inte att ta bort en tallrik från mitten av stapeln. Er uppgift är att implementera följande Stack interface:

```
public interface Stack {
        int size();                             // Current stack size
        boolean isEmpty();                      // True if stack is empty
        void push(Object element);              // Add element at top of stack
        Object pop();                           // Return and remove top element
                                                // exception if stack is empty.
        Object peek();                          // Return (without removing) top element,
                                                // exception if stack is empty.
        Iterator<Object> iterator();            // Element iterator
}
```

  Iteratorn itererar över alla stackens element. Skriv också ett testprogram StackMain.java som visar hur alla metoder fungerar. Notera: Ni får inte använda några av de fördefinerade datastrukturerna i Javas klassbibliotek. Ni får dock använda arrayer.

- **Uppgift 4**
  Börja med att skapa ett under-paket med namnet `sorter_orter` i paketet `DittLnuAnvändarNamn_lab4`. Spara sedan alla .java-filer relaterade till denna uppgift i det paketet.
  Skriv ett program SorteraOrter som läser ett godtyckligt antal ortsnamn med dess postkod från en textfil. Ni kan anta att varje ort (sträng) och postkod (heltal) finns på en separat rad i textfilen och att de är separerade av ett semikolon(;). Skapa en klass Ort som representerar en inläst ort. Klassen Ort skall implementera interfacet Comparable. Efter inläsningen skall alla orter skrivas ut på skärmen sorterade efter deras postkoder. En körning kan se ut enligt följande:

```
Läser orter från filen: C:\Temp\orter.dat
Antal funna orter: 7
23642 Höllviken
35243 Växjö
51000 Jönköping
72211 Västerås
75242 Uppsala
```

```
90325 Umeå
96133 Boden
```

Java generellt

I följande uppgifter skall ni skapa ett antal egna klasser för att lösa olika problem.

- **Uppgift 5** (The Drunken Walker)
  Skapa en klass `RandomWalk.java` som simulerar en slumpvandring. En slumpvandring utförs på ett begränsat plan och varje steg består av en förflyttning i en slumpmässig riktning. Vandringen upphör när ett maximalt antal steg har tagits eller när ett steg tas ut från det givna planet.

  Antag att planet är givet av ett rutnät, där punkt (0, 0) ligger i mitten. Storleken på planet bestäms av ett heltal; om värdet på heltalet är k så kan x- och y-koordinaterna för punkter på planet variera från $–k$ till k. I varje steg sker en förflyttning ett steg upp, ett steg ner, ett steg till höger eller ett steg till vänster (inga diagonala steg).

  Klassen `RandomWalk` behöver följande fält:

  - X-koordinaten för nuvarande position
  - Y-koordinaten för nuvarande position
  - Det maximala antalet steg en vandring får ha
  - Antalet steg tagna hittills i vandringen
  - Storleken på planet (enligt ovan)

  Övriga medlemmar

  - `RandomWalk(int max, int size)`: Det maximala antalet steg skall vara `max` och `size` är storleken på planet. Startpositionen sätts till (0, 0).
  - `String toString()`: Returnerar en sträng bestående av antalet steg tagna hittills och den nuvarande positionen.
  - `void takeStep()`: Simulerar ett steg i vandringen. Slumpa ett heltal som kan anta 4 olika värden och låt dessa representera att steget tas uppåt, nedåt, till höger eller till vänster. Metoden ska också öka antalet tagna steg.
  - `boolean moreSteps()`: Returnerar `true` om antalet tagna steg är mindre än maxantalet, annars returneras `false`.
  - `boolean inBounds()`: Returnerar `true` om nuvarande position är innanför planets område, annars returneras `false`.
  - `void walk()`: Simulerar en slumpvandring, dvs ett antal steg utförs tills det maximala antalet har uppnåtts eller tills vi vandrar ut från planets område.

  Simulering
  Skapa en klass `DrunkenWalk` som simulerar onyktra personers vandring på en plattform i en sjö. Programmet ska fråga användaren efter storleken på plattformen (planet), det maximala antalet steg och antalet onyktra personer som sätts på plattformen. En person i taget ska sättas på plattformen och utföra sin vandring. Ditt program ska räkna hur många av dem som ramlar i sjön. Testa ditt program med några olika värden på storleken och antalet steg. Exempel på körning:

```
Ange storlek: 10
Ange max antal steg: 50
Ange antal slumpvandringar: 150
Av 150 onyktra personer, föll 14 (9.34%) i vattnet.
```

- **Uppgift 6 (VG-uppgift)**
  Denna uppgift är medvetet ganska vagt formulerad. Den skissar upp ett scenario. Er uppgift är att skapa de klasser som behövs för att simulera ett sådant scenario. Gör de antaganden ni tycker är nödvändiga för att konkretisera uppgiften. Alla klasser skall vara utförligt kommenterade och följa principer så som inkapsling.

  Börja med att skapa ett under-paket med namnet `nyhetsbyra` i paketet `DittLnuAnvändarNamn_lab4`. Spara sedan alla .java-filer relaterade till denna uppgift i det paketet.
  Tidningar utbyter nyheter via nyhetsbyråer (t ex Reuters och TT). En tidning registrerar sig hos en nyhetsbyrå och skickar alla sina egna nyheter till denna. Nyhetsbyrån tar emot nyheter och skickar dem vidare till alla registrerade tidningar. Dock ej till den som "skapade" nyheten. Skapa de klasser som behövs för att simulera detta scenario. Bifoga också en Main-klass som visar en simulering av detta system.

Redovisning

Alla uppgifter skall lämnas in och vi är bara intresserade av era .java- filer (och VG-uppgiften 6 är inte obligatoriska). Zippa därför ihop .java-filerna i katalogen `DittLnuAnvändarNamn_lab4` (som finns inuti katalogen `src`) och lämna in dem mha Moodles redovisningssystem.