Inlämningsuppgift 4

Uppgiften har två syften. Du ska lära Dig att använda och implementera datastrukturen Stack och Queue för att kunna lösa olika typer av programmeringsproblem.

Uppgift 1

1. Utgå ifrån klassen **ArrayStack** från kursens hemsida ladda ner den i din mapp. Komplettera klassen med metoden *public ArrayStack copy ()* som skapar returerar en kopia av stacken (obs, inklusive innehållet i stacken om det finns).

Skriv en Junit test som bevisar att kopia du får är lika med originalet. Dvs. innehåller samma objekt i precis samma ordning.

2. Komplettera klassen ArrayQueue med metoden *printQueue()*. Den skriver ut innehållet av en kö. Obs! Glöm inte att kön är cirkulär. Alla element som finns i arrayen finns inte i kö.

Skriv ett Junit test för alla metoder som finns definierade i ArrayQueue.

2. En **DataBuffert** är en cirkulär queue (wrap-around) med fixt storlek som kan användas för att överföra data mellan t.ex. asynkrona processer, lagra loggfiler, mm.

Vanligtvis, när en buffert är tom väntar programmet på att bufferten skall fyllas. När bufferten är full överförs data (eller skrivs till fil mm). En sådan buffert-objekt har vanligtvis följande metoder.

-isEmpty()

- isFull()

-enqueue()

-dequeue()

-size()

Inspirera dig från min klass ArrayQueue som finns på kursens hemsida. Implementera en ny klass, kalla den DataBuffert. Implementera de angivna metoderna. (obs. det är mycket likt ArrayQueue). DataBuffertens storlek skall skickas som input via klassens konstruerade.

Skriv ett enkelt program som simulerar det tidiga beskrivna scenariot.
För att kunna simulera asynkrona processer skall du låta enqueue och dequeue exekveras i separata trådar. Klasserna Producer och Consumer som finns bifogade visar hur du kan skapa en sådan kö. I mitt fall har jag använd klassen ArrayQueue. Anpassa dessa klasser så de använder din DataBuffer klass, dessutom skall dequeue börja exekveras när bufferten är full. Då skall dequeue exekveras så många gånger som storleken på kön. Därefter skall dequeue vänta tills bufferten är full igen.
Samtidigt som bufferten töms skall nya värde kunna läggas till bufferten.

I programmet skall kön fyllas med slumpade numeriska värden tills den är full. När kön är full skall börja göra dequeue() och skriva de värden till fil. Om kön blir tom skall inget göras. Använd klassen DataOutputStream från java.io för att skriva data till fil. Se exempel i blackboard.

Uppgift 2

(Stack och beräkning av aritmetiska uttryck)

1. Läs kapitlen 4.3 i första kursboken Programming. Det handlar om Stack och kö och dess tillämpningar.

En av tillämpningarna för stackar är beräkningsmaskiner. Lada ner filen **Infix.java** (från boken) som omvandlar en aritmetisk utryck från infix notation till postfixnotation. Exekvera och analyser koden.

Då märker du att den resulterade postfix utrycket blir korrekt om infix utrycket är ”full parenteriserad”, dvs. parenteserna visar i vilken ordning operationerna skall utföras.

Du skall ändra i programmet så att den klarar av även utryck utan parenteser dvs. **3+4*2-1** omvandlas till **243*+1**-, men

3+4*(3-1) omvandlas till **2431-*+**

I princip skall du ändra koden så att operatorerna med högre och lika prioritet skall alltid ”popas” från stacken innan den nya operator ”pushas” i stacken.

För att göra programmet bättre antar vi att programmet läser in från tangent bordet en String innehållande en infix notation och producerar en länkad lista innehållande den postfix notationen .

Jag har byggt början på programmet i filen **InfixtoPostfixProgram.java** som du hittar länkad. Spara filen och komplettera koden där.

När programmet fungerar korrekt, flytta den relevanta koden till metoden

public static LinkedList<String> infixToPostfix(String exp){} returnerar listan med postfix notationen. Metoden finns i klassen Calculator.java som också finns länkad. Ladda ner filen och exekvera den.

1. Undersök algoritmen **EvaluatePostfix.java** från kurslitteraturen. Exekvera och förstå. Med utgångspunkt i algoritmen skriv en metoden

   public static double evaluate(LinkedList<String> exp) {} som tar som argument en länkad lista med en aritmetisk utryck i postfix notation och returnerar resultatet. Metoden finns tom i klassen Calculator. Här en grov beskrivning av algoritmen.

   Skapa Stack- Object

   x= läs ett tecken från listan

   sålänge (listan inte är tom) { om (x är operand)

   Push (x i Stack)

   om (x is operator){

   gör pop från stacken två gånger

   Operand1= stack.pop() Operand2=

   stack.pop()

   resultat= CalculateExpresion (Operand1, Operand2, Operator)

   Stack push (resultat)

   2. När du har testat att metoderna fungerar korrekt ta bort kommentaren i klassen Calculator *//expressionField.setText(evaluate(infixToPostfix(exp)))*och låt metoden anropas. Förhoppningsvis fungerar din miniräknare…

   Extra (Frivilligt): Miniräknaren hanterar just nu inte decimaltal. Hinner du då kan du ändra i koden så att det klarar av även beräkning med decimalt värde.

Frågor att besvara:

1) Ge exempel på två tillämpningar där du skulle använda en stack implementerad med array och en med länkad lista.

2) Om du inte vill implementera en Stack klass. Skulle du kunna använda ArrayList eller LänkadList i ett program där du behöver Stack push() och pop(). Hur? Vilka metoder använder du i så fall.

3) Finns det i Java biblioteket en klass Queue? Vilka metoder finns där?

4)Visa med ett exempel (inte från boken) hur stacken används för att omvandla ett aritmetiskt uttryck i infix notation till aritmetiskt uttryck i postfix notation samt hur stacken används för att beräkna uttrycket. Uttrycket du väljer skall innehålla alla operatorer samt minst 2 parenteser.