Laborationsrapport

Algoritmer och datastrukturer I , 7,5 hp
HT 2015

Laboration nr: 2

Implementation av stack, kö och lista

av

Viktor Hanstorp (1994-04-13)

Institutionen för matematik, natur- och datavetenskap
Högskolan i Gävle

S-801 76 Gävle, Sweden

Email:
ndi14vhp@student.hig.se

Innehåll

1 Inledning

Laborationen går ut på att implementera en länkad lista samt array-baserad kö och stack. Även skall den ge erfarenhet att använda rekursion och JUnit tester.

2 Förutsättningar och krav

En länkad lista skall implementeras enligt interfacet ExtendedList. Array-baserad kö och stack skall implementeras enligt interfacen Queue och Stack. Samtliga klasser skall använda runtime exceptions vid fel, och det skall finnas Javadoc och JUnit tester för alla klasser.

3 Implementering och test

3.1 Systembeskrivning

3.1.1 LinkedList

Då det finns metoder för bearbetning av listan i båda ändar ”insertFirst() , insertLast() , getFirst() , getLast() , removeFirst() , removeLast()” så valdes det att ha en dubbellänkad lista. Detta gör det lättare att bearbeta båda ändarna.

Räkning av element sker inte iterativt/rekursivt vid förfrågan, istället så finns det en räknare som räknar upp och ner när man lägger in och tar bort element. Detta gör att vi slipper traversera hela listan varje gång man vill veta hur lång den är.

”printList” och ”printListR” traverserar listan och bygger ihop en sträng med en ”StringBuilder” vilken sedan skrivs ut. För omkastad ordning så läggs texten till efter det rekursiva anropet returnerar, då den sista noden returnerar från funktionen först.

Sökning i listan sker med metoden ”search”. Metoden tar emot ett objekt av den typ som lagras i listan, och returnerar alla instanser som anses lika med ”equals”.

Resultatet returneras i form av instanser av klassen ”SearchResult”. ”SearchResult” är en hållare för funna element som också erbjuder extra funktionalitet. Alla element som hittas wrappas av klassen ”Element” vilket exponerar data för det funna värden samt har koll på vilken nod som den ligger i. ”Element” har också en metod ”remove” som tar bort den knutna noden ur listan, vilket man kan utnyttja om man enkelt vill ta bort element.

”SearchResult” har också funktionen ”remove” som tar bort alla funna noder ur listan. Se exempel:

```
LinkedList<Integer> testList = new LinkedList<Integer>();
testList.insertLast(1000);
testList.insertLast(2000);
testList.insertLast(1000);

testList.search(1000).remove();
```

Det är även möjligt at iterera de funna elementen:

```
LinkedList<Integer> testList = new LinkedList<Integer>();
testList.insertLast(1);
testList.insertLast(2);
testList.insertLast(1);
testList.insertLast(3);
for (LinkedList<Integer>.Element element : testList.search(1))
{
    System.out.println(element.data);
}
```

3.1.2 ArrayQueue och ArrayStack

Att nämna är att undantagen är inre klasser för respektive klass, då de bara förväntas användas av objektet i fråga. Samt att de kastar de ovan nämnda ”runtime exceptions” vid full lista, tom lista och vid insättning av null.

Null tillåts inte då det inte är en instans av den generiska typen.

Värdena sparas i en array av typen ”Object”.

Dock så skulle det gå att skapa arrayen första gången man stoppar in ett värde, till exempel:

```
// for array stack

// in class ”head”
T[] array = null;



// in method ”push”
if(array == null)
{
    array = (T[]) Array.newInstance(v.getClass(), maxSize);
}
```

Lösningen krävs då man inte kan initiera en generisk stack utan att veta vilken klass som den generiska klassen är.

Dock så betyder detta att man får fel för minnesallokering vid första insättning av elementet istället för vid skapandet av stacken (dock så är det inte så farligt).

Det kräver även att vid varje insättning så kollar vi om arrayen är null.

För enkelhetens skull så valdes det att använda en array av typen ”object” istället, då indata för metoden ”push” är styrt av den generiska typen (och det inte går att kasta inkompatibla typer till den typen) och null värden inte tillåts. Detta garanterar (då arrayen är privat) att vi alltid stoppar in instanser av korrekt typ.

3.2 Tester

För att testa att det ges exceptions på rätt ställen så används den statiska metoden ”assertThrows” i min klass ”T”. Metoden tar emot vilken typ av exception som förväntas, samt ett lambda-uttyck som den skall köra.

Den kör sedan lambda-uttrycket och om den kastar en exception så kollar den om det var det exception som vi förväntar oss. Är det inte det så ”fail”:ar vi testet. Sker ingen exception så ”fail”:ar vi också testet.

3.2.1 LinkedList

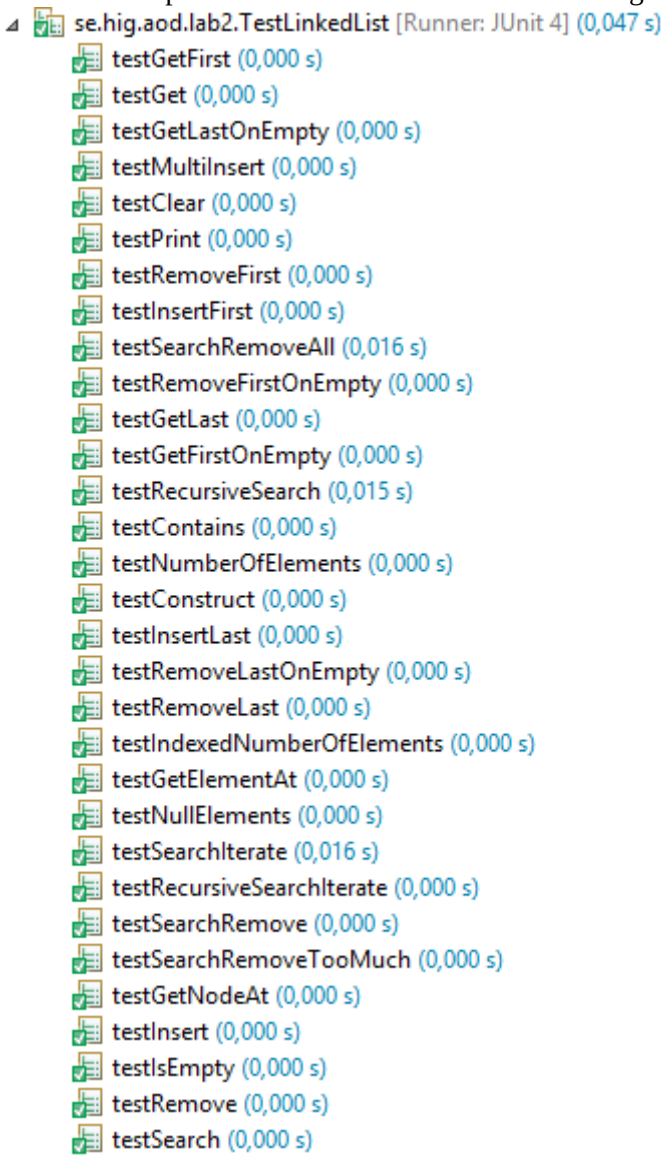
Testning av klassen ”LinkedList” sker i JUnit klassen ”TestLinkedList”. Metoder och extremvärden testas.

För att testa om de iterativa och rekursiva ”print” metoderna ger samma värde så har den statiska metoden ”checkSystemOut” i min klass vid namn ”T” använts.

”checkSystemOut” omdirigerar ”System.out” till en temporär buffer, kör ett givet lambda-uttryck, återställer ”System.out” och returnerar vad som skrivits.

Detta görs då de iterativa och rekursiva ”print” metoderna skall skriva ut samma text, och för att testa att de verkligen gör det så måste vi lyssna på ”System.out”. Detta gör också så att vi inte får massa onödig utskrift i konsolen vid testning av dessa metoder.

En skärmdump av testet för ”LinkedList” kan ses i Figur 1.



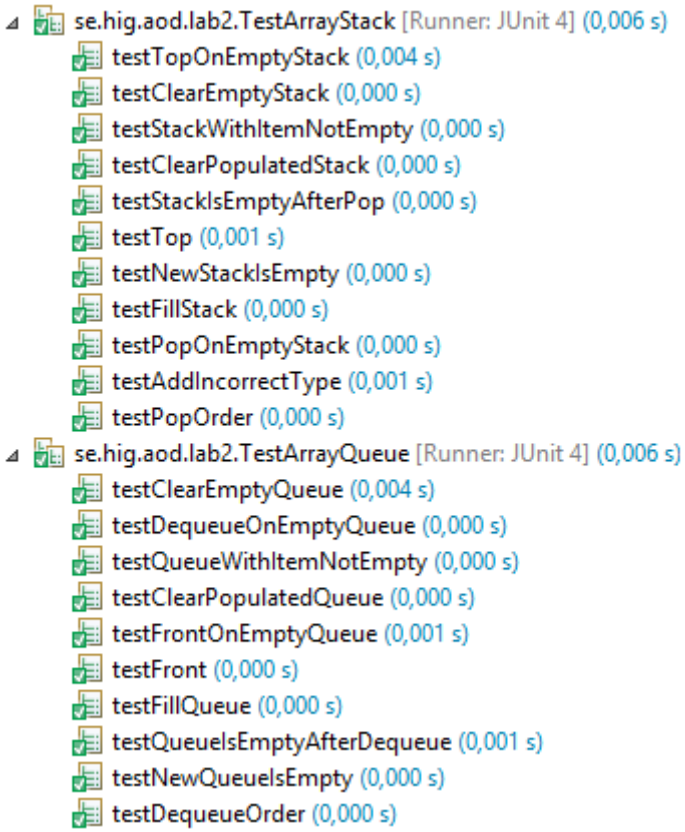
Figur 1 - Skärmdump av test för "LinkedList"

3.2.2 ArrayQueue och ArrayStack

Testning av klassen ”ArrayQueue” sker i JUnit klassen ”Test ArrayQueue” och testning av klassen ”ArrayStack” sker i JUnit klassen ”Test ArrayStack”.

Metoder och extremvärden testas.

Se Figur 2:



Figur 2 - Test av ArrayStack och ArrayQueue

4 Diskussion

4.1 Diskussion av resultatet

Resultatet av labben är tre klasser: ”LinkedList”, ”ArrayStack” och ”ArrayQueue” samt JUnit test och javadoc för samtliga.

Detta är det som efterfrågades i kravspecifikationen.

En förbättring för ”LinkedList” skulle vara att bygga mot ”collection” samt göra listan ”thread-safe”.

4.2 Diskussion av genomförandet

Det kan anses ha lagts till lite för mycket funktionalitet som inte efterfrågas i uppgiften. Till exempel är ”search” och ”for-each” iterering. Nästa gång så kanske det bör göras så lite som möjligt istället.

Att jobba hemifrån skulle utnyttjat tiden bättre.

5 Referenser

Klasser av Magnus Hjelmblom, Anders Jackson och Peter Jenke har använts, se @author i källkoden.

6 Bilagor

Källkoden finns tillgänglig i bifogad folder.