Analyse

Alexander Shipunov

04.10.15

Forord

Forbehold: Jeg valgte å gjennomføre målinger uten JIT kompileringen og slo av den i Eclipse. For å måle tid brukte jeg System.nanoTime, siden StopWatchCPU classen fra bokens bibliotekk ga merkelige resultater, da resultatene utvikles trinnvis og i noen tilfeller registreres det 0 sec kjøretid.

Jeg gjennomførte målinger for push og pop operasjoner separat både for generic og primitive type av implementasjon. Dette betyr at jeg fikk fire forskjellige resultater (2 for generic og 2 for primitive inmpementasjoner) som kunne sammenlignes på flere måter. De viktigste sammenligningsmetoder jeg valgte var:

1. å sammenligne push operasjoner for generic og primitive implementasjonene
2. å sammenligne pop operasjoner for og primitive implementasjonene
3. å sammenligne praktiske resultatene med teoretiske beregninger

Teotretisk analyse

For å vurdere resultater av praktiske målinger gjennomførte jeg teoretisk analyse av kjøretid som implementasjonene av IntegerStack, jeg lagde, skulle ha (se kommentarer i kildekoden). Generic implementasjon benytter nyttige metoder av ArrayList klassen, bl.a. add og remove som er aktuelle for mitt analyse. Begge to metodene benytter en del andre metoder (ensureCapacityInternal, grow, Arrays.CopyOf, rangeCheck) som skulle ha konstant kjøretid og System.arraycopy metode som er naitive og trenger ytterligere kunnskaper for å bli analysert. Derfor gikk jeg ut fra en et forbehold at System.arraycopy metode skulle ha en konstant kjøretid og derfor både push og pop (som benytter add og remove metodene) skulle ha konstant kjøretid som skulle stige når størrelsen på stacken øker[1]. Dette betyr at kjøretid for både pop og push metodene i generic implementering er følgende:

$$F(n) = a * n,$$

hvor n er antall elementer som pushes/popes (problemstørrelse) og a er egen argument for konret datamaskin.

Push og pop metodene i primitive implementasjon av IntegerStack klassen benytter seg av resize metode som dobler størrelse på array hvis arrayen blir fyll og halverer størrelsen hvis det er ¼ del av arrayen som er utfylt av elementene. Jeg går ut fra at hvis man bare pusher elementer så kalles det resize metode ved 2., 3., 5., 9., 17., 33. osv. push.

Dette betyr at for 1 + (2+1) +(2^2 + 1) + (2^3 + 1) + ... + (2^k + 1) antall pusher kjøres det resize k. ganger.

[1] Ifølge dokumentasjonen for ArrayList klassen har add operasjon er ”amortized constant time”. Se - http://hg.openjdk.java.net/jdk7/jdk7/jdk/file/9b8c96f96a0f/src/share/classes/java/util/ArrayList.java

1 + (2+1) +(2^2 + 1) + (2^3 + 1) + ... + (2^k + 1) = 2^0 + 2^1 + 2^2 + 2^3 + ... +2^k + k - 1).

2^0 + 2^1 + 2^2 + 2^3 + ... +2^k = 2^(k+1) – 1

Resize metoden kjøres k ganger for 2^(k+1) + k – 2 pusher. Bruker vi O-notasjon for vi konstant kjøretid for resize metoden ved store antall pusher.

Detter er også aktuelt for pop metoden.

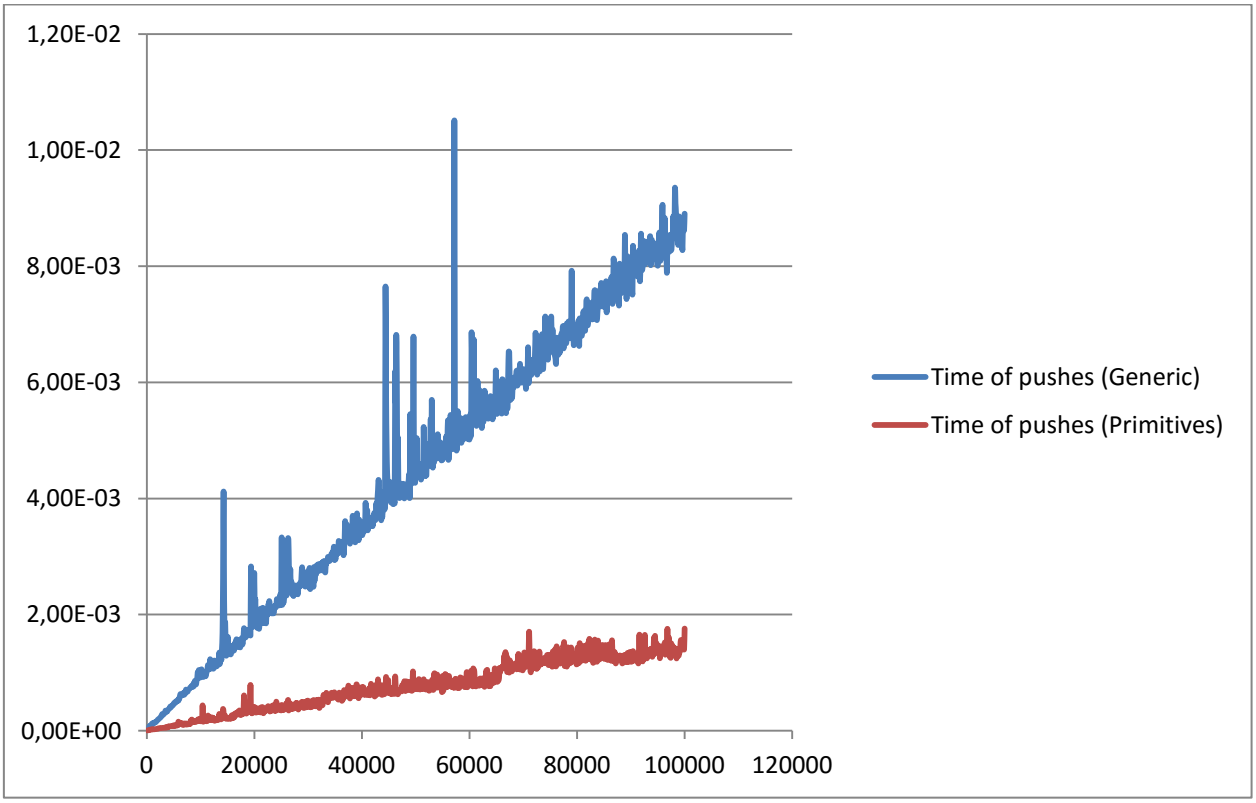
Selv om resize metoden kan bli tidskrevene siden den kopierer den gamle arrayen til en ny og kjøretiden for denne metoden i O-notasjone er antall elementer som skal kopieres, har resize metoden en konstant kjøretid i forhold til push og pop metodene. Derfor kan vi konkludere at push og pop metodene i primitive implementasjon har også konstant kjøretid:

$$F(n) = a * n,$$

hvor n er antall elementer som pushes/popes (problemstørrelse) og a er egen argument for konret datamaskin.

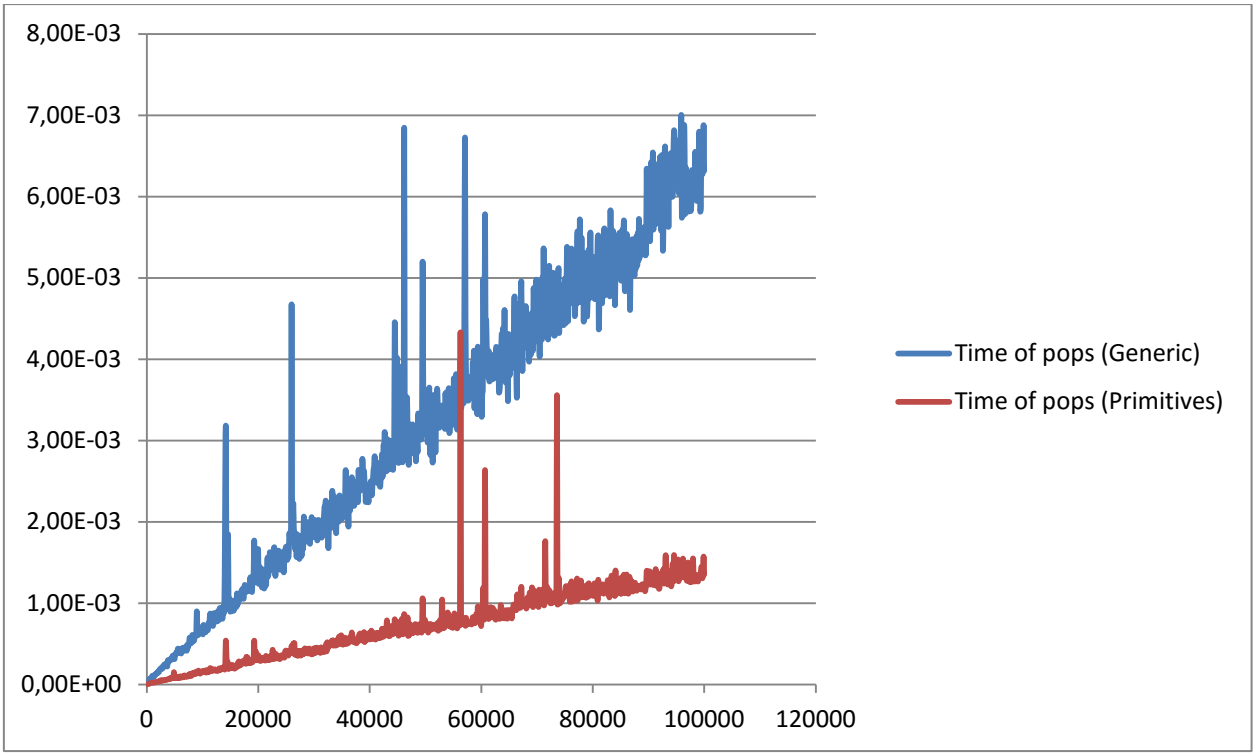
Sammenligning push operasjoner for generic og primitive implementasjonene

Alle målingene ble gjennomført for IntegerStack objektene som har størrelse f.o.m. 100 t.o.m. 100000 elementer. Dete ble gjennomført 1000 målinger per operasjon. Følgende figur viser forrskjell i kjøretid for push operasjon for primitive og generic implementasjoner.



Figuren bekrefter at primitive impimentasjonen har bedre kjøretid.

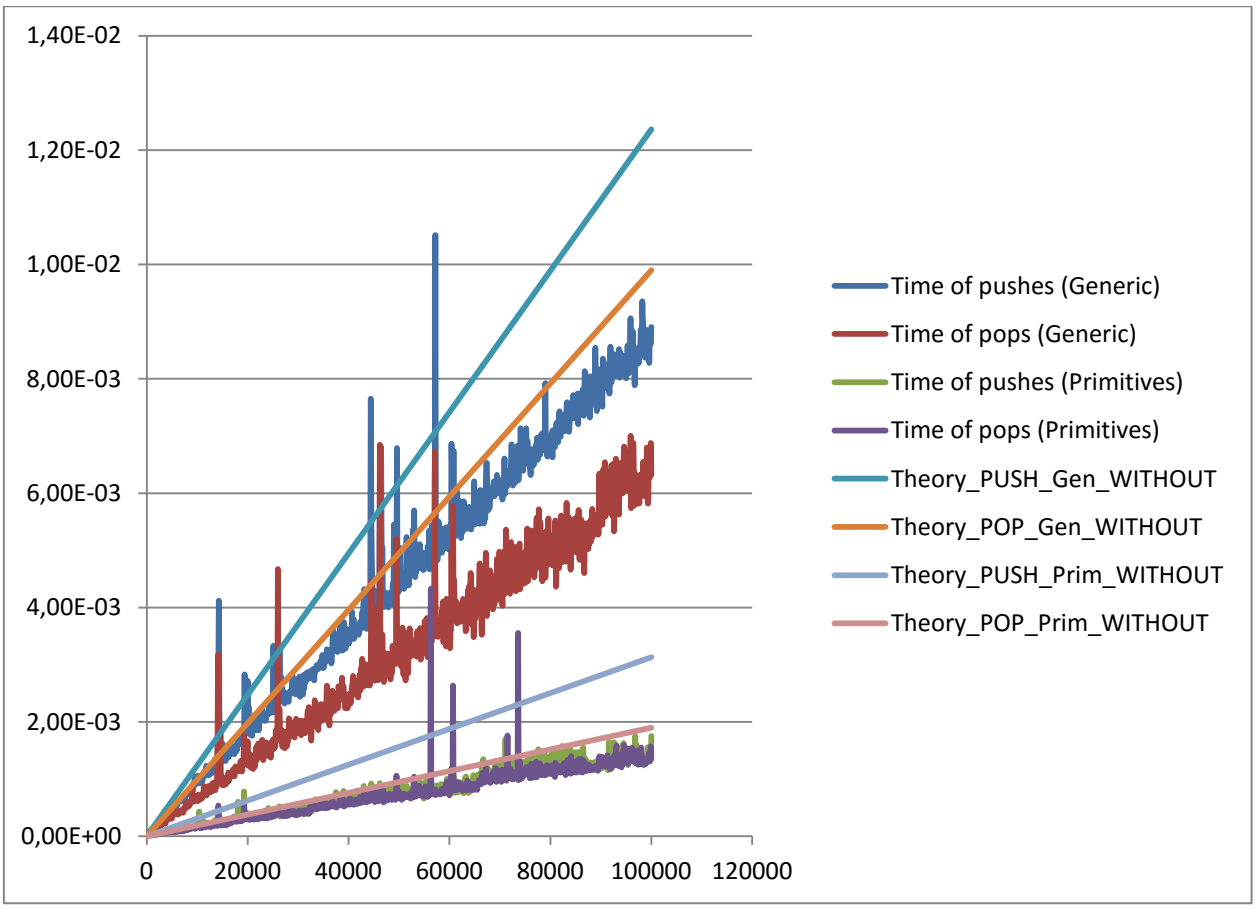
Sammenligning pop operasjoner for generic og primitive implementasjonene



Kjøretider for pop operasjonene har samme tendens som push operasjonene for forskjellige implementeringer.

Sammenligne praktiske resultatene med teoretiske beregninger

Teoretisk vurdering og praktiske resultater for begge to implementasjoner.



Konklusjon

1. Generic implementasjon av StackInteger har en lengre kjøre tid for push og pop metoder. Mest sansynlig knyttes det til autoboxing og unboxing operasjoner som skal gjennomføres hver gang man pusher/poper primitiv type (integer) inn-/ut fra stack. Det kan også forårsakes metoden System.arraycopy som i noen tilfeller kan ha lengre kjøre tid enn vanlig kopiering med hjelp av for-loop. Dette er særlig aktuelt for array med primitive datatyper vs. objekter.
2. Teoretiske vurderinger av kjøretid ga truverdige resultater med merkelige variasjoner. Teoretiske resulteter svarer worst-case scenario siden praktiske målinger ligger under. Mer presis teoretisk vurdering trenger ekstra kunnskap og dypere analyse, særlig for System.arraycopy metode og push metode for primitiv implementasjon av IntegerStack.