Invarianter

IN1010 – Objektorientert programmering

Lars Tveito

Vår 2024

Introduksjon

Plan for timen

- Vi skal lære litt om *tilstander* og *tilstandsoverganger*
- Det lar oss snakke om *egenskaper* ved tilstander og invarianter
- Vi skal lære om *før- og etterbetingelser*
- Vi skal lære om to former for *invarianter*:
  - Klasseinvarianter
  - Løkkeinvarianter

Tilstander

Tilstander

- En tilstand gir et *øyeblikksbilde* av et kjørende program
- En tilstand består av:
  - verdiene til alle variabler i hele programmet, samt
  - hvor i programutførelsen man er
- Hvor i programutførelsen man er gis av en *programteller*
  - Vi vil tenke på programtelleren neste linjenummer som skal eksekveres
  - Dette er en forenkling, siden den vil egentlig være mer finkornet
  - Vi kaller den `PC`, kort for Program Counter

Tilstander (eksempel)[1]

Innholdet i `main`:

```
1  int n = 5;
2  int i = 1;
3  int f = 1;
4
5  while (i <= n) {
6      f *= i;
7      i++;
8  }
9
10 System.out.println(f);
```

Ved første linje har vi kun en programteller

PC
1

[1] Se eksempelet i Python tutor (for Java)

Tilstander (eksempel)[1]

Innholdet i `main`:

```
1  int n = 5;
2  int i = 1;
3  int f = 1;
4
5  while (i <= n) {
6      f *= i;
7      i++;
8  }
9
10 System.out.println(f);
```

Ved linje fire har vi initialisert tre variabler

PC	n	i	f
4	5	1	1

[1] Se eksempelet i Python tutor (for Java)

Tilstander (eksempel)[1]

Innholdet i `main`:

```
1  int n = 5;
2  int i = 1;
3  int f = 1;
4
5  while (i <= n) {
6      f *= i;
7      i++;
8  }
9
10 System.out.println(f);
```

I løkken endrer vi variablene `f` og `i`

PC	n	i	f
6	5	1	1
7	5	1	1
5	5	2	1

[1] Se eksempelet i Python tutor (for Java)

Tilstander (eksempel)[1]

Innholdet i `main`:

```
1  int n = 5;
2  int i = 1;
3  int f = 1;
4
5  while (i <= n) {
6      f *= i;
7      i++;
8  }
9
10 System.out.println(f);
```

I løkken endrer vi variablene `f` og `i`

PC	n	i	f
6	5	2	1
7	5	2	2
5	5	3	2

[1] Se eksempelet i Python tutor (for Java)

Tilstander (eksempel)[1]

Innholdet i `main`:

```
1  int n = 5;
2  int i = 1;
3  int f = 1;
4
5  while (i <= n) {
6      f *= i;
7      i++;
8  }
9
10 System.out.println(f);
```

I løkken endrer vi variablene `f` og `i`

PC	n	i	f
6	5	3	2
7	5	3	6
5	5	4	6

[1] Se eksempelet i Python tutor (for Java)

Tilstander (eksempel)[1]

Innholdet i `main`:

```
1  int n = 5;
2  int i = 1;
3  int f = 1;
4
5  while (i <= n) {
6      f *= i;
7      i++;
8  }
9
10 System.out.println(f);
```

I løkken endrer vi variablene `f` og `i`

PC	n	i	f
6	5	4	6
7	5	4	24
5	5	5	24

[1] Se eksempelet i Python tutor (for Java)

Tilstander (eksempel)[1]

Innholdet i `main`:

```
1  int n = 5;
2  int i = 1;
3  int f = 1;
4
5  while (i <= n) {
6      f *= i;
7      i++;
8  }
9
10 System.out.println(f);
```

I løkken endrer vi variablene `f` og `i`

PC	n	i	f
6	5	5	24
7	5	5	120
5	5	6	120

[1] Se eksempelet i Python tutor (for Java)

Tilstander (eksempel)[1]

Innholdet i `main`:

```
1  int n = 5;
2  int i = 1;
3  int f = 1;
4
5  while (i <= n) {
6      f *= i;
7      i++;
8  }
9
10 System.out.println(f);
```

Ved siste linje skrives verdien av `f` ut

PC	n	i	f
10	5	6	120

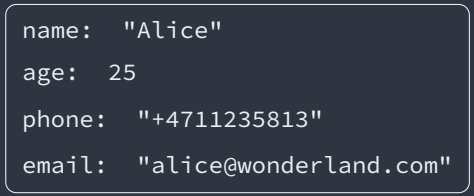
[1] Se eksempelet i Python tutor (for Java)

Litt mer om tilstander

- En kjøring av et program kan ses på som en sekvens av tilstander
- Hver programsetning tar programmet fra en tilstand til en annen
  - Disse kalles tilstandsoverganger
- Programtelleren endres implisitt
  - Semikolon kan ses på en inkrementering av programtelleren
  - Metodekall, `if`, `while`, og lignende har egne regler for programtelleren
- Variabler endres eksplisitt med tilordninger (=)
- En *debugger* er et verktøy som lar deg inspisere tilstanden til et program
- Tilstanden til *hele* programmet blir veldig fort vanskelig å tenke på
  - Vi ønsker å kunne resonnere over deler av tilstanden

Tilstand i objekter

- Tilstanden i et *objekt* er gitt av verdiene til alle instansvariablene
- Objektdiagrammer er en måte å illustrere tilstanden i et objekt



- Objekter gir en meningsfull måte å dele opp tilstanden på

Invarianter

Invarianter

- En invariant er en påstand om programmets tilstand som alltid er sann
  - En påstand eller betingelse er noe som kan sies å være sant eller usant
- Eksempler:
  - `size` er et ikke-negativt heltall
  - I et person-objekt med peker til en forelder er `this.age` < `this.parent.age`
  - Etter `i` iterasjoner inneholder `f` det `i`-te fakultetstallet
  - Etter `i` iterasjoner er de `i` siste tallene i arrayet sortert
  - Hastigheten til en partikkel er mindre enn lysets hastighet
- Ofte vil de uttrykke ting som virker helt åpenbare
  - For eksempel uttrykker typen til en variabel en invariant
- De er nyttige fordi de gir noen holdepunkter når vi resonnerer om programmer
- Brudd på invarianter er et tydelig symptom på at noe er galt i programmet

`assert`[2]

- `assert` tar et boolsk uttrykk, og kaster et unntak hvis det er `false`
- Et `assert` kommer i to former:
  - ```
    assert booleanExpression;
    ```

    ```
    Exception in thread "main" java.lang.AssertionError
        at Main.main(Main.java:4)
    ```
  - ```
    assert booleanExpression : "uffda";
    ```

    ```
    Exception in thread "main" java.lang.AssertionError: uffda
        at Main.main(Main.java:4)
    ```
- Assert må aktiveres med et kommandolinjeflagg:

```
$ java -ea Main
```

[2] Se Java sin dokumentasjon

Når `assert` bør brukes

- Du kan legge inn `assert` for å sørge for at invariantene ikke brytes
- Det skal *ikke* brukes for å behandle feil
  - Det er en måte å oppdage feil *under utvikling og testing*
- Ofte tar det tid å sjekke om invarianten holder

```
assert myList.isSorted()
```

  - Derfor er de avskrudd ved vanlig kjøring (i produksjon)
- Tommelfingerregel:
  - De bør *ikke* brukes til å sjekke input til `public` metoder
  - De bør brukes til å sjekke input til `private` metoder

Før- og etterbetingelser

- Metoder kan ha før- og etterbetingelser (eng: pre- and post conditions)
- Førbetingelser må være oppfylt før et metodekall
  - `collection != null && !collection.isEmpty()`
  - `file.exists()`
  - `0 <= age && age <= 120`
- Etterbetingelser må være oppfylt når metoden returnerer
  - `result != null`
  - `collection.isEmpty()`
  - `0 < age`
- Å programmere med før- og etterbetingelser kalles «Design by contract»

Klasseinvarianter

- En klasseinvariant er en invariant som holder for *alle* instanser av klassen
  - Konstruktøren må gjøre invarianten sann
  - Hver metode i klassen har klasseinvarianten som både før- og etterbetingelse
- Ofte er det tilstrekkelig å legge dem inn som en kommentar i klassen
  - De kan eksplisitt sjekkes ved å legge til en `assert` i slutten av hver metode
- Hold felter `private`
  - Da er det tilstrekkelig å sørge for at alle metoder opprettholder invarianten
  - Hvis et felt er `public` har du ingen garantier utover typen
  - (Det samme gjelder ubegrensede `public` settere)
- Sjekk input til `public` metoder
  - Vi har ingen garantier for hvordan `public` metoder blir kalt
  - Sjekk inputverdier og kast fornuftige unntak
- Bruk `assert` for `private` metoder
  - Private metoder kan kun kalles fra andre metoder i samme klasse
  - Innad i klassen kan du anta at metoden kalles med fornuftige argumenter

Løkkeinvarianter

- En løkkeinvariant er en invariant som holder for alle iterasjoner av løkken
- Oftest vil løkkeinvarinten si noe om *målet* med løkken
  - `sum` inneholder summen av de `i` første tallene i `arr`
  - De `i` første elementene i `arr` er sortert
- Du kan bruke løkkeinvarianten til å vise at algoritmen er korrekt

Løkkeinvarianter (eksempel)

```
static int min(int arr[]) {
    int min = arr[0];
    // min inneholder det minste tallet mellom arr[0] og arr[0]
    for (int i = 1; i < arr.length; i++) {
        // min inneholder det minste mellom arr[0] og arr[i-1]
        if (arr[i] < min) { min = arr[i]; }
        // min inneholder det minste mellom arr[0] og arr[i]
    }
    // min inneholder det minste mellom arr[0] og arr[arr.length - 1]
    return min;
}

public static void main(String[] args) {
    int arr[] = { 86, 91, 81, 14, 37, 5, 97, 2, 75, 72, 93, 35, 48 };
    System.out.println(min(arr));
}
```

Et større eksempel med klasseinvarianter

SortedSet[3]

- Vi skal lage en enkel datastruktur som implementerer en mengde som en ordnet syklisk dobbeltlenket liste
- Den inneholder klasseinvarianter som:
  - Elementene er ordnet (fra minst til størst)
  - Det er ingen duplikater
  - At en node sin neste sin forrige er noden selv (og andre veien)
  - Den lagrede størrelsen svarer til antall noder i strukturen

[3] https://gist.github.uio.no/larstvei/4948335fb4bfbec8eab1673d0c4e2f42

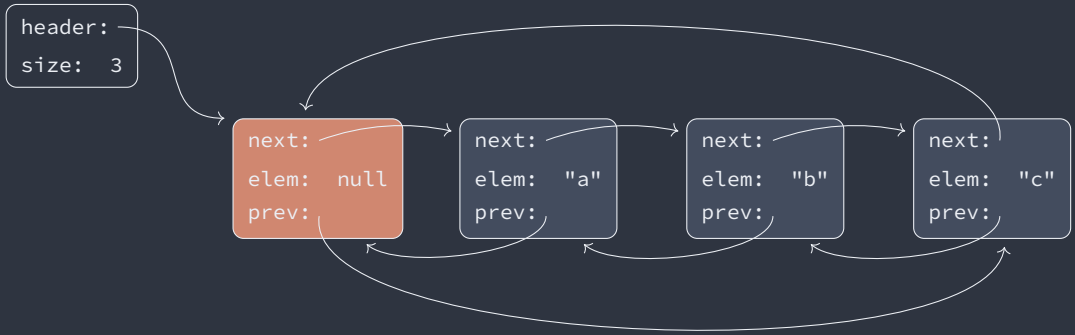
SortedSet (struktur)

```
public class SortedSet<T extends Comparable<T>> {
    private DLLNode header;
    private int size = 0;

    private class DLLNode {
        DLLNode prev, next;
        T elem;
        void add(T elem) { ... } // Adds an element directly after this
        void remove() { ... }   // Removes this node
    }

    public SortedSet() { ... }
    public boolean isEmpty() { ... }
    public int size() { ... }
    public SortedSet<T> add(T elem) { ... }
    public SortedSet<T> remove(T elem) { ... }
}
```

`SortedSet` (tilstand)



SortedSet (invarianter)

```
private void checkInvariants() {
    assert isSorted() : "The SortedSet isn't sorted!";
    assert isProperlyLinked() : "The elements in the SortedSet aren't
    ↪  properly linked!";
    assert noDuplicates() : "The SortedSet contains duplicate
    ↪  elements!";
    assert checkSize() : "The size of the SortedSet doesn't match the
    ↪  calculated size!";
}
```

SortedSet (add)

```
public SortedSet<T> add(T elem) {
    DLLNode current = header.next;
    while (current != header && current.elem.compareTo(elem) < 0) {
        current = current.next;
    }
    if (!elem.equals(current.elem)) {
        size++;
        current.prev.add(elem);
    }
    checkInvariants();
    return this;
}
```

SortedSet (remove)

```
public SortedSet<T> remove(T elem) {
    DLLNode current = header.next;
    while (current != header && !elem.equals(current.elem)) {
        current = current.next;
    }
    if (elem.equals(current.elem)) {
        size--;
        current.remove();
    }
    checkInvariants();
    return this;
}
```

SortedSet(isSorted)

```
private boolean isSorted() {
    DLLNode current = header.next;
    while (current.next != header) {
        if (current.elem.compareTo(current.next.elem) > 0) {
            return false;
        }
        current = current.next;
    }
    return true;
}
```

SortedSet (noDuplicates)

```
private boolean noDuplicates() {
    DLLNode current = header.next;
    while (current.next != header) {
        if (current.elem.equals(current.next.elem)) {
            return false;
        }
        current = current.next;
    }
    return true;
}
```

SortedSet (isProperlyLinked)

```
private boolean isProperlyLinked() {
    DLLNode current = header.next;
    while (current.next != header) {
        if (current.next.prev != current || current.prev.next !=
        ↪  current) {
            return false;
        }
        current = current.next;
    }
    return true;
}
```

SortedSet (checkSize)

```
private boolean checkSize() {
    DLLNode current = header.next;
    int calculatedSize = 0;
    while (current != header) {
        current = current.next;
        calculatedSize++;
    }
    return calculatedSize == size;
}
```

SortedSet (Testprogram)

```
public static void main(String[] args) throws Exception {
    String dictPath = "random-order-wordle-words.txt";
    List<String> words = Files.readAllLines(Paths.get(dictPath));
    System.out.println("Number of words in the file: " + words.size());

    SortedSet<String> mySet = new SortedSet<>();

    for (String word : words) { mySet.add(word); }
    System.out.println("After adding all words: " + mySet.size());

    // No duplicates get in?
    for (String word : words) { mySet.add(word); }
    System.out.println("After adding them all again: " + mySet.size());

    // Removing is fine?
    for (String word : words) { mySet.remove(word); }
    System.out.println("And removing them all: " + mySet.size());
}
```

SortedSet (Kjør test)

```
$ time java SortedSet
Number of words in the file: 14855
After adding all words: 14855
And after adding them all again: 14855
And removing them all: 0

________________________________________________________
Executed in    1.64 secs    fish           external
   usr time    1.63 secs    0.10 millis    1.63 secs
   sys time    0.04 secs    3.19 millis    0.03 secs
```

SortedSet (Kjør test med -ea)

```
$ time java -ea SortedSet
Number of words in the file: 14855
After adding all words: 14855
And after adding them all again: 14855
And removing them all: 0

________________________________________________________
Executed in   14.43 secs    fish           external
   usr time   14.46 secs    0.10 millis   14.46 secs
   sys time    0.06 secs    2.08 millis    0.06 secs
```