Notater: INF1010

Veronika Heimsbakk
veronahe@student.matnat.uio.no

5. juni 2013

1 Tilgangsnivåer

Modifier	**Class**	**Package**	**Subclass**	**World**
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

2 CompareTo

I interfacet Compareable ligger metoden `compareTo` som er fin å bruke i sammenheng med bl.a. generiske typer ved sortering.

Mindre enn	negativ verdi
Like	0
Større enn	positiv verdi

3 Invariant

Et predikat (boolsk verdi) er kalt en *invariant* til en sekvens av operasjoner når: predikatet er `true` *før* sekvensen startes, da er det `true` på slutten av sekvensen også.

4 for-each

```
1 public metodeNavn (Integer[] i) {
2   for (Integer x : i) { // cruiser gjennom arrayen
3     whatToDo;
4   }
5 }
```

5 Subklasser

```java
class A {
  int a;

  A (int a) {
    this.a = a;
  }
}

class B extends A {
  int b;

  B (int a, int b) {
    super(a);
    this.b = b;
  }
}

class C extends B {
  int c;

  C (int a, int b, int c) {

    // Vil finne konstruktoren i B, og saa titte paa A sin.
    // Siden B har to parametere, saa velger den B sin konstruktor,
    // men vi faar med begge variablene vi vil ha.

    super(a, b);
    this.c = c;
  }
}
```

code/subklasse.java

6 Lenkede lister

Eksempel på lister er: filer, arrayer, lister, FIFO og LIFO.

6.1 Innsetting

Tenke på om lista skal være FIFO (First In First Out) eller LIFO (Last In First Out).

6.1.1 FIFO

```java
void settInnSist (E e) {
  if (liste == null)
    liste = e;
  else {
    E tmp = liste;
    while (tmp.neste != null) {
      tmp = tmp.neste;
      tmp.neste = e;
    }
```

```
10   }
11 }
```

code/settinnFIFO.java

6.1.2 LIFO

```
1 void settInnForst (E e) {
2   if (liste == null)
3     liste = e;
4   else {
5     e.neste = liste;
6     liste = e;
7   }
8 }
```

code/settinnLIFO.java

6.1.3 Legge til et objekt med attributter

```
1 public boolean leggTil (String navn, String nummer) {
2   if (hentObjekt(navn) == null) {
3     E tmp = new E(navn, nummer);
4     tmp.neste = root;
5     root = tmp;
6     return true;
7   }
8   return false;
9 }
```

code/leggetil.java

6.2 Fjerne et objekt

```
1 public boolean fjern (String navn) {
2   E tmp = root;
3
4   if (tmp == null) return false;
5
6   if ( root.navn.equals(navn) ) {
7     root = root.neste;
8     return true;
9   }
10
11  while (tmp.neste != null) {
12    if ( tmp.neste.navn.equals(navn) ) {
13      tmp.neste = tmp.neste.neste;
14      return true;
15    }
16    tmp = tmp.neste;
17  }
18  return false;
19 }
```

code/fjern.java

7 Interface, generiske typer og abstract

7.1 Interface

Man bruker interface på en klasse om man ønsker egenskaper som flere forskjellige klasser skal arve. Metodene som interfacet inneholder på implementeres på nytt hver gang det interfacet implementeres.

```
interface Name {
  method();
  type varName = value;
}
```

Dette implementeres ved å si:

```
class className implements Name {. . .}
```

Variabler i et interface vil være public, static og final.

7.2 Abstract

Når en klasse er abstract, så kan man bare lage objekter av subklassene til denne klassen.

7.3 Generiske typer

Dette er objektholdere som kan inneholde hva som helst. Men hvis den generiske typen extender en klasse, så kan man kun legge inn objekter av den klassen eller klasser som extender den respektive klassen.

7.3.1 Generiske metoder

En generisk metode må ha <E> (eller hvilken som helst annen bokstav) foran typen sin:

```
public <E> void printNoe (E[] x) {
  for (E e : x) {
    System.out.println(e); // printer hele arrayen
  }
}
```

7.3.2 Generiske returneringstyper

I dette eksempelet skal man finne største verdi av tre generiske saker.

```
public < E extens Compareable<E> > E maximum (E a, E b, E c) {
  E max = a; // antar at max verdien er a.

  if (b.compareTo(a) > 0)
    max = b;
```

```
6   if (c.compareTo(max) > 0)
7     max = c;
8   return max;
9 }
```

code/generic.java

7.4 Eksempel med noder, interface og generiske typer

```
1 interface FifoCollection <E> { // interface for FIFO-liste
2   public void insertLast(E e);
3   public E get();
4 }
5
6 class Main {
7   public static void main(String[] args) {
8     // lager en liste med PS3-spill.
9     GameContainer <PS3Game> ps3List = new GameContainer <
          PS3Game>();
10  }
11 }
12
13 // extender Game, for man skal lage en liste med KUN objekter
     av Game.
14 // implementerer FifoCollection fordi vi vil ha de metodene.
15
16 class GameContainer <E extends Game> implements FifoCollection
     <E> {
17   Node head;
18
19   public void insertLast (E e) {
20     if (head == null) {
21       head = new Node(e);
22     } else {
23       Node tmp = head;
24       // saa lenge head sin neste ikke er null, gaar man videre
             .
25       while (tmp.next != null) {
26         tmp = tmp.next;
27       }
28       // naar man har funnet den som har null som sin neste,
29       // saa setter man inn en ny node.
30       tmp.next = new Node(e);
31     }
32   }
33
34   public E get (E e) {
35     return head.element;
36   }
37
38   // siden Node er en intern klasse, saa trenger vi ikke sette
        <E>.
39   class Node {
```

```
40      E element; // objektet noden vil peke paa.
41      Node next;
42
43      Node (E element) {
44        this.element = element;
45      }
46    }
47  }
48
49  class Game {
50    private String name;
51
52    public Game (String name) {
53      this.name = name;
54    }
55
56    public String getName() {
57      return name;
58    }
59  }
60
61  class PS3Game extends Game {
62    public PS3Game (String name) {
63      super(name); // faar fatt i konstruktoren i superklassen
            Game.
64    }
65  }
```

code/node.java

8 Iterator

```
1  import java.util.Iterator;
2
3  class It implements Iterator<E> {
4    Node current;
5    Node prev;
6    boolean removeNext = false;
7
8    It() {
9      curr = new Node();
10     curr.next = head;
11     prev = curr;
12   }
13
14   public boolean hasNext() {
15     return curr.next != null;
16   }
17
18   public E next() {
19     if( hasNext() ) {
20       removeNext = true;
21       prev = curr;
22       curr = curr.next;
```

```
23        return curr.element;
24      }
25      throw new NoSuchElementException();
26    }
27
28    public void remove() {
29      if(removeNext) {
30        if(curr.element.compareTo(head.element) == 0) {
31          head = head.next;
32        } else {
33          curr = prev;
34          curr.next = curr.next.next;
35        }
36        removeNext = false;
37      } else {
38        throw new IllegalStateException();
39      }
40    }
41  }
```

code/iterator.java

9 Tråder

```
1  import java.util.Random;
2
3  class Traader {
4    public static void main(String[] args) {
5      // thread-klassen mekkes av nytt objekt (Random-objekt) med
           parameter navn = one.
6      Thread t1 = new Thread ( new Random("one") );
7      Thread t2 = new Thread ( new Random("two") );
8      Thread t3 = new Thread ( new Random("three") );
9
10     // naar man kaller paa start(), vil den hoppe direkte til
           run().
11     t1.start();
12     t2.start();
13     t3.start();
14   }
15 }
16
17 class Random implements Runnable {
18   String navn;
19   int tid;
20   Random r = new Random();
21
22   Traader(String s) {
23     name = s;
24     // tiden vil vaere mellom 0 og 999 milisekunder.
25     tid = r.nextInt(999);
26   }
27
28   public void run() {
```

```
29      try {
30        System.out.println(name + " is sleeping for " + time);
31        Thread.sleep(time);
32        System.out.println(name + " is done " + name);
33      } catch (Exception e) {. . .}
34    }
35  }
```

code/traad.java

Nøkkelordet `synchronized` gjør at kun en tråd kan jobbe på metoden av gangen.

10 GUI

```
1  import javax.swing.*;
2  import java.awt.*;
3
4  class Main {
5    public static void main(String[] args) {
6      new Eksempel();
7    }
8  }
9
10 class Eksempel {
11   Eksempel() {
12     // Lager vinduet
13     JFrame frame;
14     frame = new JFrame("Testvindu");
15
16     // Avslutt ved kryss i hjornet
17     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18
19     // Setter px x px str
20     frame.setSize(300, 200);
21     frame.setVisible(true);
22   }
23 }
```

code/gui.java

10.1 Komponenter

```
1  JButton button = new JButton("Trykk her");
2  JLabel label = new JLabel("Skriv inn noe");
3
4  // bruker skriver tekst, getText() returnerer dette.
5  JTextField text = new JTextField(30);
6  JTextArea window = new JTextArea(10, 30);
7
8  // Lager rullevindu ut av window (TextArea).
9  JScrollPane scroll = new JScrollPane(window);
10
11 // Et panel inneholder andre komponenter.
```

```
12 JPanel panel = new JPanel();
```

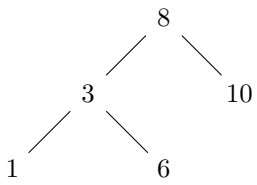
code/komp.java

10.2 Action Listener

```
1  // Lager en knapp
2  JButton btn = new JButton("Hi");
3
4  // Legger til denne i vinduet
5  getContentPane().add(btn);
6
7  // Sier hva som skal lytte pa knappen etter trykk.
8  btn.addActionListener(btnListen);
9
10 /* Action Listener */
11 btnListen = new Listen();
12 btn.addActionListener(btnListen);
13
14 class Listen implements ActionListener {
15   public void actionPerformed(ActionEvent e) {
16     System.out.println("Someone pushed me!");
17   }
18 }
```

code/listen.java

11 Binære trær



Binære trær har på det meste to child-noder. Et sortert binært tre har alltid større verdi enn foreldrenoden til høyre og den mindre verdien i ventre node. Det lille treet over er sortert.

11.1 Iterering

- Pre-order: root, venstre barn, høyre barn
- In-order: venstre barn, root, høyre barn
- Post-order: venstre barn, høyre barn, root

11.2 Eksempel på binær tre

11.2.1 Innsetting

```
class BinaryTree {
  public static void main(String[] args) {
    BinaryTree tree = new BinaryTree();
  }

  Node root;

  public Node addNode (int key, String name) {
    Node newNode = new Node(key, name);

    if (root == null) {
      root = newNode;
    } else {
      Node focusNode = root;
      Node parent;

      while (true) {
        parent = focusNode;

        if (key < focusNode.key) {
          focusNode = focusNode.left;

          if (focusNode == null) {
            parent.left = newNode;
            return;
          }
        } else {
          focusNode = focusNode.right;

          if (focusNode == null) {
            parent.right = newNode;
            return;
          }
        }
      }
    }
  }
}
```

code/tre_legginn.java

11.2.2 Fjerne

```
public boolean remove (int key) {
  Node focusNode = root;
  Node parent = root;

  boolean isItALeft = true;

  while (focusNode.key != key) {
    parent = focusNode;

    if (key < focusNode.key) {
      isItALeft = true;
      focusNode = focusNode.left;
```

```
13      } else {
14        isItALeft = false;
15        focusNode = focusNode.right;
16      }
17
18      if (focusNode == null)
19        return false;
20    }
21
22    if (focusNode.left == null && focusNode.right == null) {
23      if (focusNode == root)
24        root = null;
25      else if (isItALeft)
26        parent.left = null;
27      else
28        parent.right = null;
29
30    } else if (focusNode.left == null) {
31      if (focusNode == root)
32        root = focusNode.right;
33      else if (isItALeft)
34        parent.left = focusNode.right;
35      else
36        parent.right = focusNode.left;
37    } else {
38      Node replacement = replace(focusNode);
39
40      if (focusNode == root)
41        root = replacement;
42      else if (isItALeft)
43        parent.left = replacement;
44      else
45        parent.right = replacement;
46
47      replacement.left = focusNode.left;
48    }
49    return true;
50  }
51
52  public Node replace (Node replaceNode) {
53    Node replaceParent = replaceNode;
54    Node replacement = replaceNode;
55    Node focusNode = replaceNode.right;
56
57    while (focus != null) {
58      replaceParent = replacement;
59      replacement = focusnode;
60      focusNode = focusNode.left;
61    }
62
63    if (replacement != replaceNode.right) {
64      replaceParent.left = replaceNode.right;
65      replaceNode.right = replacement.right;
66    }
```

```
67   return replaceNode;
68 }
```

code/tre_fjerne.java

11.2.3 Traversering

```
1  /* IN-ORDER TRAVERSING */
2
3  public void inOrder (Node focusNode) {
4    if (focusNode != null) {
5      inOrder(focus.left);
6
7      System.out.println(focusNode);
8
9      inOrder(focusNode.right);
10   }
11 }
12
13 /* PRE-ORDER TRAVERSING */
14
15 public void preorder (Node focusNode) {
16   if (focus != null) {
17     System.out.println(focusNode);
18
19     preorder(focusNode.left);
20     preorder(focusNode.right);
21   }
22 }
23
24 /* POST-ORDER TRAVERSING */
25
26 public void postorder (Node focusNode) {
27   if (focusNode != null) {
28     postorder(focusNode.left);
29     postorder(focusNode.right);
30
31     System.out.println(focusNode);
32   }
33 }
```

code/traversering.java