Seminaropplegg uke 6

Tema:

- Lister og generiske klasser
- Generiske klasser

Lenkelister:
Introdusere lenkelister

Helt enkel Node-klasse, *liveprogrammer/skriv på tavle*:

```
class Node {
	Node neste; //= null hvis ingen neste

	public void settNeste(Node n){
		neste = n;
	}

	public Node hentNeste(){
		return neste;
	}
}
```

Lite hovedprogram:

```
class Hovedprogram {
	public static void main(String [] args){
		Node start = new Node();
		start.settNeste(new Node());
		Node nrTo = start.hentNeste();
		nrTo.settNeste(new Node());
	}
}
```

- Tegn opp strukturen til denne listen på tavla, gjerne underveis(!) - hvert objekt har listen har en neste-referanse

- Forklare at det er vanlig å benytte seg av “Node” i en lenket liste, men man kan like gjerne lage lenkede lister med andre klasser.

- Nodeklassen over i seg selv er ganske “ubrukelig” siden den ikke holder på noe data.
Utvid Node-klassen, holde på en String-verdi:

```
class Node {
	Node neste; //= null hvis ingen neste
	String verdi;

	//Konstruktør:
	public Node(String verdi){
		this.verdi = verdi;
```

```
        }

        public void settNeste(Node n){
                neste = n;
        }

        public Node hentNeste(){
                return neste;
        }

        public String hentData(){
                return verdi;
        }
}
```

Generiske klasser, enkelt eksempel:

```
public class Boks<T> {
   private T t;

   public void fyll(T t) {
      this.t = t;
   }

   public T hent() {
      return t;
   }
}
```

T - er da en *plassholder* for en type! Når vi så oppretter en instans av Boks, må vi spesifisere hvilken type vi ønsker at T skal representere.

Spm til klassen: Er dette noe de kjenner igjen? Har noen vært borti dette før?

> ArrayLists, HashMap!

Feks.

```
Boks<String> b = new Boks<String>();
String tekst = "hallo!";
b.fyll(tekst);
System.out.println(b.hent());
```

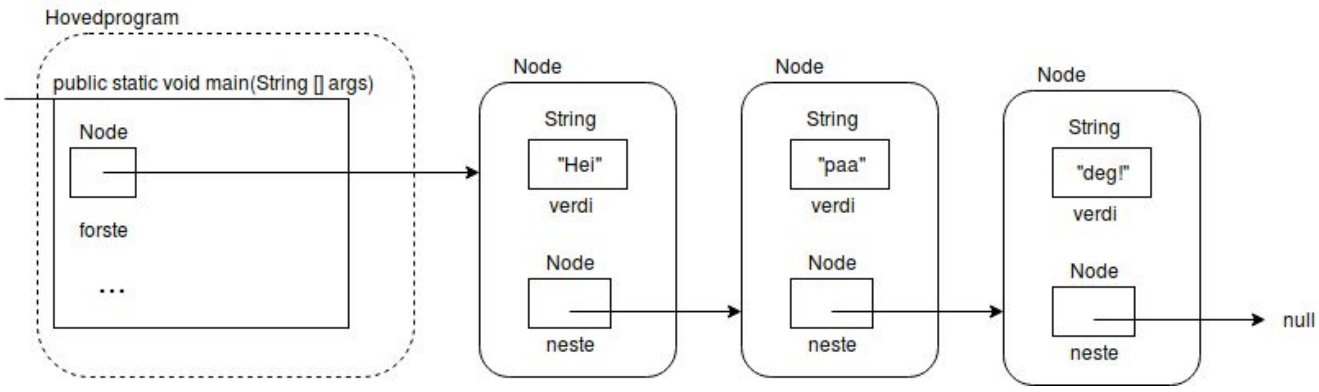
Det vanlige er å bruke “T” (for Type), men det finnes også en rekke andre som er vanlige å bruke, litt avhengig av hvordan man bruker dem:

- E - Element (used extensively by the Java Collections Framework)
- K - Key
- N - Number
- T - Type
- V - Value

Oppgave 1:
Basert på Noden vi programmerte sammen:

```
class Node {
	Node neste; //= null hvis ingen neste
	String verdi;

	public Node(String verdi){
		this.verdi = verdi;
	}

	public void settNeste(Node n){
		neste = n;
	}

	public Node hentNeste(){
		return neste;
	}

	public String hentData(){
		return verdi;
	}
}
```

1.a) Skriv et hovedprogram, hvor du oppretter Noder og lenker dem sammen slik at du får følgende datastruktur:



1.b) Skriv deretter en while-løkke som skriver ut alle verdiene til Nodene.

Oppgave 2:
Du skal nå endre “Node-klassen”, bytt ut det som trengs for å gjøre den generisk. Skriv også om hovedprogrammet slik at den passer til den nye “Node-klassen”.

Oppgave 3:

Du har fått interfacet:

```
public interface Par<K, V> {
    public abstract K hentNokkel();
    public abstract V hentVerdi();
}
```

3.a) Skriv klassen “OrdnetPar”, denne skal implementerer interfacet “Par”.

3.b) Et ordnet par har to instansvariabler, en nøkkel og en verdi, begge er generiske typer. Skriv konstruktøren for OrdnetPar, her settes instansvariablene.

3.c) Implementer deretter metodene som kreves av interfacet.

Løsning:

```
public class OrdnetPar<K, V> implements Par<K, V> {

    private K nokkel;
    private V verdi;

    public OrdnetPar(K nokkel, V verdi) {
       this.nokkel = nokkel;
       this.verdi = verdi;
    }

    public K hentNokkel()    { return nokkel; }
    public V hentVerdi() { return verdi; }
}
```