Oppgave 1

uke 2

Dette er essensielt en repetisjon fra i110.

1 Peker-kjeder

En peker-kjede er en liste med operasjoner som tillater å innsette nye objekter, observere objekter, samt traversere listen. Vi implementerer en peker-kjede som en liste av *noder* der hver node har en peker til *neste* node i listen samt et data objekt *elem*. Noder i listen skal tilhøre følgende klasse:

```
public class Node {
 private Node next;
 private Object elem;
```

med følgende grensesnitt operasjoner:

- `public Node(Object o, Node n);` – ny Node med `n` som `next` og `o` som `elem`
- `public Node();` – ny Node med både `elem` og `next` lik `null`
- `Object getElem()` – returner `elem` ved noden
- `Object getNext()` – returner nodens `next`
- `void setElem(Object o)` – set nodens `elem` til `o`
- `void setNext(Node n)` – set nodens `next` til `n`

```
}
```

1.1

Implementer klassen `Node`.

1.2

Vi vil nå skrive ut objektene samlet i en peker-kjede. Med dette mener vi at for hver node `s` i listen, vil vi skrive ut `toString()` attributtet av tilhørende `elem`-objektet.

Lag en subklasse `NodeS` som utvider klassen `Node` med to metoder

- `skriv()` – et kall `s.skriv()`, der `s` peker på en `NodeS` skal skrive ut hele listen f.o.m. `s` t.o.m. slutten av listen
- `skrivBaklengs()` – et kall `s.skrivBaklengs()`, der `s` peker på en `NodeS` skal skrive ut hele listen baklengs, dvs. f.o.m. slutten av listen t.o.m. `s`.

2 Stabel

En Stabel (LIFO) er en (abstrakt) data type som kan tenkes som en liste av objekter med følgende operasjoner:

- `public void push(Object o)` – setter inn `o` i starten av listen

- `public Object peek()` – returnerer objektet som står fremst i listen (som ble satt inn sist)
- `public Object pop()` – som `peek` men i tillegg fjerner fremste objektet fra listen (etterfølgende kallet til `peek` vil returnere objektet som opprinnelig sto som nr.2)
- `public boolean isEmpty()` – returnerer `true` hviss listen er tom

2.1

Lag `interface Stack` med dette grensesnittet, dokumenter den og plasser den i din katalog der du skal holde alle dine `package`.

Lag en spesiell katalog, la oss si `pakker`, for dette formålet. For at `javac` skal kunne finne den uten din hjelp, må du legge til din `.cshrc` filen (i hjemmekatalogen din) følgende linje:

```
setenv CLASSPATH $HOME/pakker :  /usr/java/lib
```

Katalogen `pakker` kan inneholde underkataloger med forskjellige pakker.

2.2

Du skal nå bruke `Node`-klassen fra oppgave 1.1 til å implementere `Stack`.

Lag en klasse `Stab` med en peker-kjede av `Node` objekter, samt ev. andre data-attributter. Klassen ska ha en metode (med samme argumenter og type) for hver metode fra `interface Stack` (og, selvfølgelig, med den spesifiserte virkning!). Disse metodene skal være *de enseste* som en bruker av `Stab` har adgang til. (Spesielt, skal han *ikke* ha noen adgang til `Node` objekter – "objekt" betyr for ham objekt som brukes i `Stack` (og som i peker-kjede "henger" som `elem` attributtene ved noder)).

I tillegg til metodene fra `Stack`, vil `Stab` måte inneholde konstruktor(er). Deklarasjon av `Stab` skal starte med:

```
public class Stab implements Stack {...
```

3 Kø

En Kø (FIFO) er en (abstrakt) data type som kan tenkes som en liste av objekter med følgende operasjoner:

- `public void enqueue(Object o)` – setter inn `o` på slutten av listen
- `public Object front()` – returnerer objektet som står fremst i listen (som har vært der lengst)
- `public Object dequeue()` – som `front` men i tillegg fjerner fremste objektet fra listen
- `public boolean isEmpty()` – returnerer `true` hviss listen er tom

3.1

Lag `interface Queue` med dette grensesnittet, dokumenter den og plasser den i din katalog `pakker`.

3.2

Du skal nå gjenta oppgave 2.2 for `Queue`. Lag en klasse `Koe` med en peker-kjede av `Node` objekter, samt ev. andre data-attributter, og som implementerer hver metode fra `interface Queue`. Antakelser og restriksjoner her er tilsvarende til de i oppgave 2.2.