Ukesoppgaver, gruppetime 9 - Designmønstre

Oppgave 1: Designmønstre

Eyvind snakket i forelesningen om tre kategorier av designmønstre: *Behavioral*, *Creational* og *Structural*. Diskuter hvorfor designmønstre er nyttige, hva som kjennetegner de tre nevnte typene og i hvilke tilfeller dere ville brukt et designmønster fremfor et annet. Kom frem til konkrete eksempler på brukstilfeller.

Oppgave 2: Iterator-mønsteret og tilskuerliste

Klassen Tilskuerliste under skal bruke iterator-mønsteret. Tilskuerliste er en enkeltlenket liste som tillater objekter fra Tilskuer-klassen. Anta at VIP-tilskuere er i første halvdel av lista og at andre tilskuere er i andre halvdel av lista. Det skal kun være mulig å iterere gjennom halve lista, fra og med første til og med midterste element, siden vi er interesserte i VIP-tilskuerne. Gjør nødvendige implementasjoner i koden under (se kommentarer), slik at vi kun håndterer VIP-tilskuerne når vi itererer over tilskuerlista med en forenklet (enhanced) for-løkke.

```
import java.util.Iterator;

class Tilskuer {
    private String navn;

    public Tilskuer(String navn) {
        this.navn = navn;
    }

    @Override
    public String toString() {
        return "* Tilskuer " + navn;
    }
}

// Enkeltlenket liste: Tilskuerliste
class Tilskuerliste implements Iterable<Tilskuer> {
    class Node {
        Tilskuer data; Node neste;
        Node(Tilskuer data) {
            this.data = data;
            this.neste = null;
        }}

    private Node forste, siste; private int storrelse;
```

```java
    // Legger til et element i slutten av listen
    public void add(Tilskuer tilskuer) {
        Node nyNode = new Node(tilskuer);
        if(forste == null) {
            forste = nyNode; siste = nyNode;
        } else {
            siste.neste = nyNode; siste = nyNode;
        }
        storrelse++;
    }

    // Returnerer en ny iterator som starter fra starten av listen
    @Override
    public Iterator<Tilskuer> iterator() {
        return new ForsteTilMidtenIterator(this.forste, this.storrelse);
    }

    // Iterator-klasse for å iterere fra starten av listen til midten
    class ForsteTilMidtenIterator implements Iterator<Tilskuer> {
        private Node denne; private int stoppIndeks, denneIndeks;

        ForsteTilMidtenIterator(Node forste, int storrelse) {
            this.denne = forste;
            this.stoppIndeks = storrelse / 2; //Stopper iterasjonen etter midten
            this.denneIndeks = 0;
        }

        @Override
        public boolean hasNext() {
            // Implementer nødvendig kode her
        }

        @Override
        public Tilskuer next() {
            // Implementer nødvendig kode her
        }
    }
}
```

Oppgave 3: Observer-mønsteret og aksjeoppdateringer

Du jobber for et softwareselskap som spesialiserer seg på finansiell programvare. Teamet ditt har fått i oppgave å utvikle et system som skal la brukere overvåke og få varsler om endringer i aksjepriser i sanntid. Du har bestemt deg for å bruke Observer-mønsteret for å implementere denne funksjonaliteten. Implementer et enkelt *AksjemarkedMonitor*-program som bruker Observer-mønsteret for å oppdatere investorer når en aksjes pris endrer seg. Du trenger disse klassene og grensesnittene i programmet ditt:

- *Aksje-klassen* (Subject): Skal inneholde aksjens navn og pris. Den skal være ansvarlig for å holde en liste over observatører (investorer) og varsle dem når prisen endrer seg.
- *Investor*-grensesnittet (Observer): Skal deklarere en oppdater-metode som blir kalt når en Aksje de observerer endrer pris.
- *KonkretInvestor*-klasse: Skal implementere Investor-grensesnittet og hjelpe til å demonstrere funksjonaliteten av systemet.
- *AksjemarkedMonitor*: hovedprogram med main-metode

Test programmet ditt ved å opprette et par aksjer, knytte noen investorer til dem, endre pris på aksjene og se at systemet varsler investorene.

Oppgave 4: Decorator-mønsteret og hamburgerbar

Målet ditt er å implementere et hovedprogram som simulerer en Hamburgerbar. Det skal være mulig å dekorere en HamburgerBasis med ost samt fjerne sylteagurk. HamburgerBasis koster 50 kroner. Ost koster 10 kroner ekstra og fjerning av sylteagurk reduserer prisen med 5 kroner. Benytt decorator-mønsteret, og test at hamburgerbaren fungerer etter hensikten ved å “lage” en hamburger med ost og uten sylteagurk.

Oppgave 5: Singleton-mønsteret og svakheter

Ta utgangspunkt i Singleton-mønsteret. Hvilke(n) svakhet(er) oppdager du i koden under? Hvorfor er det viktig at svakheten(e) blir utbedret? Utbedre svakhetene. Hva kunne du gjort for å gjøre programmet enda mer robust (*Hint:* kommer seinere i IN1010).

```
public final class Singleton {
    private static Singleton instance;

    public static Singleton getInstance() {
        instance = new Singleton();
        return instance;
    }
}
```