Løsningsforslag, gruppetime 9 - Designmønstre

Oppgave 2: Iterator-mønsteret og tilskuerliste - Løsningsforslag

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

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

        @Override
        public boolean hasNext() {
            return denne != null && denneIndeks <= stoppIndeks;
        }

        @Override
        public Tilskuer next() {
            Tilskuer data = denne.data;
            denne = denne.neste;
            denneIndeks++;
            return data;
        }
    }
```

Oppgave 3: Observer-mønsteret og aksjeoppdateringer - Løsningsforslag

```
import java.util.ArrayList;
import java.util.List;

interface Investor {
    void oppdater(Aksje aksje, double nyPris);
}
class KonkretInvestor implements Investor {
    private String navn;
```

```
    public KonkretInvestor(String navn) {
        this.navn = navn;
    }

    @Override
    public void oppdater(Aksje aksje, double nyPris) {
        System.out.println(
            "Investor " + navn + " fikk beskjed om at prisen på " +
            aksje.hentNavn() + " er nå " + nyPris);
    }
}

class Aksje {
    private String navn; private double pris;
    private List<Investor> investorer = new ArrayList<>();

    public Aksje(String navn, double pris) {
        this.navn = navn; this.pris = pris;
    }

    public String hentNavn() { return navn; }

    public void leggTilInvestor(Investor investor) {
        investorer.add(investor);
    }

    public void settNyPris(double nyPris) {
        pris = nyPris;
        for (Investor investor : investorer) {
            investor.oppdater(this, nyPris);
        }
    }
}

class AksjemarkedMonitor {
    public static void main(String[] args) {
        // Opprett aksjer
        Aksje apple = new Aksje("Apple", 150.7);
        Aksje microsoft = new Aksje("Microsoft", 200.2);

        // Opprett investorer
        Investor ola = new KonkretInvestor("Ola");
        Investor kari = new KonkretInvestor("Kari");
```

```
        // Knytt investorer til aksjer
        apple.leggTilInvestor(ola);
        apple.leggTilInvestor(kari);
        microsoft.leggTilInvestor(ola);

        // Endre kurs på aksjer
        apple.settNyPris(160.3);
        microsoft.settNyPris(180.5);
    }
}
```

Oppgave 4: Decorator-mønsteret og hamburgerbar - Løsningsforslag

```
// Definerer interfacet for basis hamburger
interface Hamburger {
    String hentBeskrivelse();
    double hentPris();
}

// Lager grunnleggende hamburger
class HamburgerBasis implements Hamburger {
    @Override
    public String hentBeskrivelse() {
        return "Hamburger";
    }
    @Override
    public double hentPris() {
        return 50.0;
    }
}

abstract class HamburgerDekorator implements Hamburger {
    private final Hamburger dekorertHamburger;
    public HamburgerDekorator(Hamburger hamburger) {
        this.dekorertHamburger = hamburger;
    }

    @Override
    public String hentBeskrivelse() {
        return dekorertHamburger.hentBeskrivelse();
    }
```

```
    @Override
    public double hentPris() {
        return dekorertHamburger.hentPris();
    }
}

class MedOst extends HamburgerDekorator {
    public MedOst(Hamburger hamburger) {
        super(hamburger);
    }

    @Override
    public String hentBeskrivelse() {
        return super.hentBeskrivelse() + ", med ost";
    }

    @Override
    public double hentPris() {
        return super.hentPris() + 10.0;
    }
}

class UtenSylteagurk extends HamburgerDekorator {
    public UtenSylteagurk(Hamburger hamburger) {
        super(hamburger);
    }

    @Override
    public String hentBeskrivelse() {
        return super.hentBeskrivelse() + ", uten sylteagurk";
    }

    @Override
    public double hentPris() {
        return super.hentPris() - 5.0;
    }
}

class Hamburgerbar {
    public static void main(String[] args) {
        // Oppretter en hamburger, legger til ost, fjerner sylteagurk
        Hamburger hamburger = new HamburgerBasis();
        hamburger = new MedOst(hamburger);
```

```
        hamburger = new UtenSylteagurk(hamburger);

        // Skriver ut den endelige beskrivelsen og prisen på hamburgeren
        System.out.println(
            hamburger.hentBeskrivelse() + ", koster: " + hamburger.hentPris()
        );
    }
}
```

Oppgave 5: Singleton-mønsteret og mangler - Løsningsforslag

```
// Definerer en konstant klasse, fordi subklasser ikke skal kunne arve fra den
public final class Singleton {
    // Statisk instans. Vi skal kunne aksessere den i en statisk kontekst.
    private static Singleton instance;

    // Privat konstruktør. Ingen utenfor klassen skal kunne opprette instanser.
    private Singleton() {}

    // Metode som returnerer Singleton-instans. Opprettes hvis den ikke finnes.
    public static Singleton getInstance() {
        if (instance == null) instance = new Singleton();
        return instance;
    }
}
```

Økt robusthet: Vi kan gjøre Singleton-klassen trådsikker. Mer om tråder seinere i IN1010.