Løsningsforslag, gruppetime 10 - Invarianter

Oppgave 1: Løsningsforslag - Handlekurv

a) Før hver iterasjon er verdien av total alltid lik summen av prisene på produktene fra og med prisen på indeks 0 til og med prisen på indeks i-1. *Alternativt:* Etter hver iterasjon er verdien av total alltid lik summen av prisene på produktene fra og med prisen på indeks 0 til og med prisen på indeks i.

b) **Førbetingelser**
Arrayet eksisterer: produktpriser må peke på et array med double-verdier. Arrayet har elementer: produktpriser må peke på et array med minst ett element. Hvis arrayet er tomt, er det ingen priser å summere, og totalen er meningsløs (selv om en tom handlekurv teknisk sett har en total på 0). Ikke-negative verdier: Alle elementene i produktpriser må representere gyldige priser og bør derfor være ikke-negative.

Etterbetingelser
Total er summen av produktpriser: Etter at totalPris-metoden har kjørt, skal verdien av total være lik summen av alle prisene i produktpriser-arrayet. Total er ikke-negativ: Den totale handlekurv-summen kan ikke være negativ. Arrayet er uendret: Arrayet produktpriser peker på skal ikke endres som en bieffekt av å kjøre totalPris(). Den forventede adferden til en slik beregningsmetode er at den ikke endrer inputdataene.

Oppgave 2: Løsningsforslag - Bankkonto

Klasseinvarianter for Bankkonto-klassen:

- saldo bør alltid være større enn eller lik 0 (ingen negativ saldo er tillatt).
- kontonummer bør aldri være null eller tomt etter initialisering.

Før- og etterbetingelser for Bankkonto-klassen:

- For innskudd(double belop): førbetingelsen er at belop > 0. Etterbetingelsen er at saldoen har økt med belop.
- For uttak(double belop): førbetingelsene er at belop > 0 og saldo >= belop. Etterbetingelsen er at saldoen blir redusert med belop.

Oppgave 3: Løsningsforslag - Fruktbutikk

```
import java.util.List; import java.util.ArrayList;
import java.util.Map;import java.util.HashMap;
public class Fruktbutikk {
    private Map<String, Integer> lager, behov;
    public Fruktbutikk(Map<String, Integer> lager, Map<String, Integer> behov) {
        this.lager = lager; this.behov = behov;
    }
```

```
    // Rekursiv hjelpemetode
    private boolean sjekkLagerRekursiv(List<String> fruktListe, int indeks) {
        if (indeks == fruktListe.size()) {
            return true;
        }

        String fruktType = fruktListe.get(indeks);
        if(!lager.containsKey(fruktType) || lager.get(fruktType)<behov.get(fruktType)) {
            return false;
        }
        // Rekursivt kall med neste indeks
        return sjekkLagerRekursiv(fruktListe, indeks + 1);
    }

    public boolean sjekkLager() {
        // Lager en listekopi av nokkelmengden for aa muliggjore rekursjon
        List<String> fruktListe = new ArrayList<>(behov.keySet());
        return sjekkLagerRekursiv(fruktListe, 0);
    }

    public static void main(String[] args) {
        Map<String, Integer> lager = new HashMap<>();
        lager.put("Epler", 50);
        lager.put("Bananer", 30);
        lager.put("Paerer", 20);

        Map<String, Integer> behov = new HashMap<>();
        behov.put("Epler", 20);
        behov.put("Bananer", 10);
        behov.put("Paerer", 25);

        Fruktbutikk butikk = new Fruktbutikk(lager, behov);
        if(butikk.sjekkLager()) {
            System.out.println("Vi har nok av alle frukttyper paa lager!");
        } else {
            System.out.println("Vi har ikke nok av alle frukttyper paa lager.");
        }
    }
}
```

Oppgave 4: Løsningsforslag - Leilighetskompleks

```
import java.util.HashMap;
class Leilighet {
    // Leilighetsattributter som leilighetsnummer, kvadratmeter, osv.
}
```

```java
class Leilighetskompleks {
    private final HashMap<String, Leilighet[]> etasjer;

    public Leilighetskompleks() {
        etasjer = new HashMap<>();
    }

    // Metode for aa legge til en etasje i komplekset.
    public void leggTilEtasje(String etasjenummer) {
        validerEtasjenummer(etasjenummer);

        if (etasjer.containsKey(etasjenummer)) {
            throw new IllegalArgumentException("Etasjenummeret finnes allerede.");
        }

        Leilighet[] leiligheter = new Leilighet[15];
        etasjer.put(etasjenummer, leiligheter);
    }

    // Hjelpemetode for aa validere formatet til etasjenummeret.
    private void validerEtasjenummer(String etasjenummer) {
        if (etasjenummer == null || !etasjenummer.startsWith("etasje")) {
            throw new IllegalArgumentException("Etasjenummer maa starte med 'etasje'");
        }
        String nummerdel = etasjenummer.substring(6);
        try {
            Integer.parseInt(nummerdel);
        } catch (NumberFormatException e) {
            throw new IllegalArgumentException("Etasjenummeret maa inneholde et gyldig
tall etter 'etasje'.");
        }
    }

    // Metode for aa hente leilighetene i en gitt etasje.
    public Leilighet[] hentLeilighetene IEtasje(String etasjenummer) {
        validerEtasjenummer(etasjenummer);
        return etasjer.get(etasjenummer);
    }

    // Metode for aa plassere en leilighet i en bestemt etasje paa en gyldig indeks.
    public void settInnLeilighet(String etasjenummer, int indeks, Leilighet leilighet) {
        validerEtasjenummer(etasjenummer);

        if (leilighet == null) {
            throw new IllegalArgumentException("Leiligheten kan ikke vaere null.");
        }
```

```
        Leilighet[] leiligheter = etasjer.get(etasjenummer);
        if (leiligheter == null) {
            throw new IllegalArgumentException("Etasjen finnes ikke.");
        }

        if (indeks < 0 || indeks >= leiligheter.length) {
            throw new IndexOutOfBoundsException("Ugyldig indeks for leilighet.");
        }

        leiligheter[indeks] = leilighet;
    }
}
```

Oppgave 5: Løsningsforslag - Produktkatalog

Klasseinvarianter:

- Produkt[] inneholder maksimalt 20 produkter.
- Produkt[] kan inneholde null, men kun på indekser >= antall. Det vil si at alle produkter frem til og med indeks antall-1 er gyldige Produkt-objekter.
- antall skal alltid tilsvare en gyldig indeks i Produkt[]. antall skal aldri være negativ eller større enn lengden på Produkt[].
- antall skal alltid reflektere antall Produkt-objekter som finnes i Produkt[].

Løkkeinvarianter:

- For den ytre løkken i fjern-metoden: Ved begynnelsen av hver iterasjon, er ingen av Produkt-objektene i Produkt[] fra indeks 0 til i-1 lik produktID. Dette sikrer at om produktID finnes i Produkt[], vil det bli funnet i løpet av løkkekjøringen.
- For den indre løkken i fjern-metoden: Etter hver iterasjon har alle produkter fra indeks i og utover blitt skiftet en posisjon til venstre, og produkter[j] inneholder nå den neste produktreferansen fra produkter[j+1].

```
class Produkt {
    private int produktID; private String produktnavn;
    private double produktpris; private String produktbeskrivelse;
    // Konstruktor og hent-/sett-metoder defineres her etter behov
}

class Produktkatalog {
    private Produkt[] produkter = new Produkt[20];
    private int antall = 0;

    public void leggTil(Produkt produkt) {
        if (antall < produkter.length) {
            produkter[antall] = produkt; antall++;
        } else {
            // F.eks. utskrift av melding eller kaste et unntak om katalogen er full.
        }
    }
```

```
    public void fjern(int produktID) {
        for (int i = 0; i < antall; i++) {
            if (produkter[i].hentProduktID() == produktID) {
                for (int j = i; j < antall - 1; j++) {
                    produkter[j] = produkter[j + 1];
                }
                produkter[antall - 1] = null;
                antall--;
                break;
            }
        }
    }
}
```