TENTAMEN

Softwaresystemen

vakcode: 201500111
datum: 18 januari 2016
tijd: 8:45 - 11:45

Algemeen

- Bij dit tentamen mag gebruik gemaakt worden van:
  - de handleiding,
  - de slides van de colleges,
  - de boeken die voorgeschreven zijn als studiemateriaal voor de module (of, als je deze niet hebt, een afdruk/kopie van de betreffende pagina's) en
  - een woordenboek.

  Je mag *geen* gebruik maken van:
  - uitwerkingen van opgaven die op Blackboard gepubliceerd zijn (recommended exercises en oude tentamens) en
  - eigen materiaal (afdrukken van je eigen practicumopgaven, aantekeningen in welke vorm dan ook).
- Het aantal punten voor het tentamen wordt meegenomen in de berekening van het eindcijfer van de module, op de manier zoals aangegeven in de handleiding.
- Dit tentamen bestaat uit 6 opgaven, waarvoor in het totaal 100 punten behaald kunnen worden. Het minimale aantal punten per opgave bedraagt 0 punten. Het eindcijfer van het tentamen wordt bepaald door de optelsom van de punten per opgave.

In dit tentamen gaan we een aantal interfaces, klassen en methoden ontwikkelen die gebruikt kunnnen worden voor een online bestelsysteem van een supermarkt.

Opgave 1 (25 punten)

a. *(3 punten)* Definieer een interface `Product` om een product dat in de supermarkt besteld kan worden te definiëren. De interface bevat methoden om de volgende kenmerken van een item op te vragen:

   - standaardprijs,
   - kortingspercentage op de standaardprijs, en
   - aantal op voorraad.

   De interface bevat daarnaast ook methodes om:

   - het kortingspercentage op de standaardprijs te veranderen en
   - om de voorraad te verlagen als er een aantal producten verkocht zijn.

   De korting wordt gegeven in de vorm van een kortingspercentage, d.w.z. een getal tussen 0 en 100.

b. *(6 punten)* Geef JML specificaties die het gewenste gedrag van de methoden in `Product` beschrijven.
   N.B. er worden niet hele uitgebreide specificaties verwacht, maar wel meer dan alleen `ensures true;`

c. *(3 punten)* Definieer een enum voor verschillende soorten frisdrank: cola, sinas, spa, cassis.

d. *(6 punten)* Definieer een klasse `Frisdrank` die `Product` implementeert. Voor elke instantie wordt bijgehouden welke type frisdank het is, en wat de prijs per fles is. Denk aan de constructor, deze krijgt het type frisdrank, de prijs per fles, en de initiële voorraad mee als argument. De methode die de prijs teruggeeft, houdt *geen* rekening met het kortingspercentage.

e. *(7 punten)* Flessen frisdrank worden vaak in grotere hoeveelheden bij elkaar verpakt als eenheid verkocht. Definieer een klasse `GrootverpakkingFris` die `Frisdrank` uitbreidt met het aantal eenheden dat in de verpakking zit en met een query methode om dit aantal eenheden op te vragen. Hergebruik methode-implementaties zoveel mogelijk. De constructor krijgt het type frisdrank, de prijs per fles, de initiële voorraad, en ook het aantal eenheden waaruit de grootverpakking bestaat mee.
   Voor de eenvoud mag er van uit gegaan worden dat elke fles op voorraad, ook als onderdeel van een grootverpakking verkocht kan worden. Als de voorraad grootverpakkingen van 6 flessen uit 4 bestaat, zijn er dus 24 flessen op voorraad. De prijs van de grootverpakking zal in dat geval 6 $\times$ de prijs van een enkele fles zijn.

Opgave 2 (20 punten)

Stel dat we een klasse `BestellingsItem` hebben.

```
public class BestellingsItem {

        private Product product;
        private int aantal;

        public BestellingsItem(Product p, int a){
                product = p;
                aantal = a;
        }

        public Product getProduct() {
                return product;
        }

        public int getAantal() {
                return aantal;
        }
}
```

De klasse `Bestelling` houdt een bestelling bij als een lijst van `BestellingsItem`s. In deze opgaven gaan we een aantal methoden in de klasse `Bestelling` implementeren.

```
public class Bestelling {

        private List<BestellingsItem> bestelling;

        public Bestelling(List<BestellingsItem> b) {
                bestelling = b;
        }

        public int totaalPrijs() {
                // Te implementeren
        }
        public void voegReceptToeAanBestelling(Recept recept) {
                // Te implementeren
        }

        public Map<Product, Set<Recept>> maakSuggestieMap(List<Recept> recepten) {
                // Te implementeren
        }
}
```

a. (*5 punten*) Implementeer de methode `totaalPrijs`, die de totaalprijs van de bestelling berekent. Deze methode houdt rekening met korting.

De online supermarkt heeft ook een voorraad recepten beschikbaar voor klanten. Vanuit het oogpunt van de bestelling kan een recept gezien worden als een lijst `BestellingsItem`s, d.w.z. een product, en het aantal van deze producten dat nodig is. Hiervoor is een klasse `Recept` gedefinieerd.

```
public class Recept {

        private List<BestellingsItem> recept;

        public List<BestellingsItem> getRecept() {
                return recept;
        }

}
```

b. (*6 punten*) Definieer een methode `voegReceptToeAanBestelling` in de klasse `Bestelling` die alle ingrediënten die nog niet in de boodschappenlijst staan toevoegt.

c. (*3 punten*) Geef een JML specificatie voor de methode `voegReceptToeAanBestelling`.

d. (*6 punten*) Gegeven een lijst van recepten, definieer een methode `maakSuggestieMap` die een map aanmaakt die voor elk product een verzameling oplevert met alle recepten die dit product als ingredient gebruiken.

Opgave 3 (10 punten)

De supermarkt houdt voor alle klanten hun eerdere aankopen bij, zodat klanten bij een volgende aankoop makkelijk terug kunnen vinden wat ze eerder gekocht hebben. Hiervoor wordt van elke klant een profiel aangemaakt, dat de volgende interface implementeert.

```
public interface Profiel {

        public String getNaam();

        public String getAdres();

        public List<BestellingsItem> getEerderBesteld();

}
```

Vervolgens worden verschillende strategieën uitgeprobeerd om klanten suggesties te geven over producten die ze mogelijk aan hun bestelling toe zouden willen voegen. Deze suggesties worden gebaseerd op het profiel van de klant, en op het totale aanbod van de supermarkt. Alle strategieën zijn implementaties van de interface `Suggestie`:

```
public interface Suggestie {

        public Set<Product> geefSuggesties(Profiel klant, List<Product> aanbod);

}
```

a. *(2 punten)* Implementeer een klasse `BekendeSuggestie` die alle eerder gekochte producten met korting suggereert.

b. *(3 punten)* Aan de interface `Product` wordt een methode `public boolean nieuwInHetAssortiment()` toegevoegd. Implementeer een klasse `SpannendeSuggestie` die alle nieuwe producten die nog niet eerder gekocht zijn suggereert.

c. *(5 punten)* Definieer tenslotte een klasse `SlimmeSuggestie` die alle bekende en spannende suggesties retourneert, zonder dat er code gedupliceerd wordt.

Opgave 4 (15 punten)

Omdat de supermarkt goed loopt, komt het steeds vaker voor dat producten niet meer leverbaar zijn. Het bestelsysteem moet aangepast worden om hier op een goede manier mee om te gaan.

a. *(4 punten)* Definieer een exception die gebruikt kan worden als er producten uit een bestelling niet meer leverbaar zijn. De exceptie moet een bericht kunnen geven met daarin alle producten die niet meer leverbaar zijn.

b. *(5 punten)* Voeg een methode `totaalPrijsAlsLeverbaar` toe in de klasse `Bestelling`. Deze methode controleert eerst of alle artikelen op voorraad zijn. Als dat het geval is, dan wordt de totaalprijs berekend, anders wordt een exception met niet leverbare producten teruggegeven.

Als alle producten leverbaar zijn, kan de bestelling gedaan en de voorraad aangepast worden. De methode `haalUitVoorraad` zorgt dat de voorraad aangepast wordt:

```
/*@ ensures (\forall BestellingsItem bi; bestelling.contains(bi);
                        bi.getProduct().getVoorraad() ==
                          \old(bi.getProduct().getVoorraad()) - bi.getAantal());
*/
public void haalUitVoorraad() {
    // correcte implementatie
}
```

c. *(6 punten)* Implementeer methode `plaatsBestelling` die de methode `totaalPrijsAlsLeverbaar` gebruikt om te kijken of alle producten voldoende op voorraad zijn. Als dat het geval is, wordt de voorraad aangepast en de totaalprijs teruggegeven. Als niet alle producten aanwezig zijn, wordt een melding op het scherm gegeven en de methode levert een (zelf te definiëren) speciale waarde op die aangeeft dat de bestelling niet gedaan is.

Opgave 5 (15 punten)

Bij een dergelijk online supermarkt zullen gebruikers zich wel eerst moeten authenticeren; het is niet de bedoeling dat gebruikers elkaars koopgeschiedenis en andere persoonlijke informatie kunnen inzien. Een van de manier om gebruikers zich te laten authenticeren is met een wachtwoord.

a. *(3 punten)* Het is tegenwoordig gemeengoed om in softwaresystemen wachtwoorden niet zomaar op te slaan, maar om daarbij een (cryptografische) hash-functie te gebruiken. Leg uit waarom.

b. *(3 punten)* Stel dat je moet kiezen tussen SHA-256 en scrypt voor het opslaan van het wachtwoorden. Welke zou je kiezen en waarom?

c. *(3 punten)* Stel Alice logt via een laptop met een wachtwoord in op de website van de online supermarkt. Ze doet dit echter buiten op een terras en Eve (die schuin achter haar zit) kijkt stiekem mee terwijl Alice haar wachtwoord intypt (dit wordt ook wel "shoulder surfing" genoemd). Eve kan het niet helemaal goed zien, maar ze kan er wel het volgende uit opmaken:

   - De eerste 3 karakters van het wachtwoord bestaan enkel uit de karakters 'a', 's', 'd', 'f' of 'g'.
   - Het 4e karakter is een '[' of een ']'.
   - De laatste 4 karakters bestaan enkel uit cijfers.

   Hoeveel verschillende wachtwoorden zijn er mogelijk?

Stel nu dat zo'n online supermarkt wordt geïmplementeerd en er dat gedurende enkele jaren door vele verschillende personen aan geknutseld is. Er lijken wat problemen te zijn rondom betalingen: de balansen kloppen niet. Jij wordt gevraagd het op te lossen. Je duikt in de code en merkt al gauw dat de broncode wat rommelig is en dat de programmeerstijl niet overeenkomt met hoe jij het geleerd hebt.

De twee klassen die verantwoordelijk zijn voor de afhandeling van de betaling zijn `BetaalModule` en `BestellingVerwerker`. Voor beide klassen zijn de relevante onderdelen op de volgende bladzijden afgedrukt. Zie voor `Bestelling` en `Profiel` de eerdere opgaven.

d. *(6 punten)* Leg uit hoe een kwaadwillende klant ervoor kan zorgen dat z'n boodschappen gratis bezorgd worden. Je mag er vanuit gaan dat een klant zelf zijn naam en adres kan wijzigen.

```
public class BetaalModule {
    /**
     * Initieer een betaling, Retourneer een betalingsnummer voor verdere verwerking.
     * @param bedrag
     * @param klantNaam
     * @return
     */
    public int initieerIDEALBetaling(int bedrag, String klantNaam) {
        // ... [inhoud niet relevant voor opgave]
    }

    /**
     * Retourneert true wanneer de betaling is afgerond (betaald of geannuleerd).
     * @return
     */
    public boolean betalingAfgerond(int betalingsnummer) {
        // ... [inhoud niet relevant voor opgave]
    }

    /**
     * Indien de betaling (met gegeven betalingsnummer) is afgerond, geef informatie terug.
     * Dit is een String met de volgende inhoud:
     * "[bedrag]:[klantNaam]:[status]:[referentienummer]"
     * Hierbij is "[status]" een van:
     *  - "OK" (indien betaling succesvol) of,
     *  - "ANN" (indien de betaling geannuleerd is.
     *
     *  Voorbeeld: "432:Jan Jansen:OK:r2424232"
     *
     *  Het is een String omdat dat doorgegeven wordt door het (externe) betaal-
     *  systeem.
     *
     * @param betalingsNummer
     * @return
     */
    public String getBetaalInfo(int betalingsNummer) {
        // ... [inhoud niet relevant voor opgave]
    }
}

public class BestellingVerwerker {

   private BetaalModule bm = new BetaalModule();

   public void verwerkBestelling(Bestelling bestelling, Profiel klant) {
        if (startBetaling(bestelling, klant)) {
            // supermarkt.Bestelling is betaald, kan verstuurd worden
            startLevering(bestelling, klant);
        } else {
            // Betaling is geannuleerd of niet op tijd afgerond, annuleer bestelling
            annuleerBestelling(bestelling);
        }
    }

    protected boolean startBetaling(Bestelling bestelling, Profiel klant) {
        int betaalVolgNummer =
                bm.initieerIDEALBetaling(bestelling.totaalPrijs(), klant.getNaam());
        while (!bm.betalingAfgerond(betaalVolgNummer)) {
```

```
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                // ignore...
            }
        }

        String betaalInfo = bm.getBetaalInfo(betaalVolgNummer);

        // Dit zou moeten checken of betaald is..., toch?
        for (int i=0; i < betaalInfo.length();i++) {
            if (betaalInfo.charAt(i) == ':') {
                if (betaalInfo.substring(i+1,i+3).equals("OK")) {
                    return true;
                } else if (betaalInfo.substring(i+1,i+4).equals("ANN")) {
                    return false;
                }
            }
        }
        return false;
    }


    public void startLevering(Bestelling bestelling, Profiel klant) {
        // ... [inhoud niet relevant voor opgave]
    }

    public void annuleerBestelling(Bestelling bestelling) {
        // ... [inhoud niet relevant voor opgave]
    }
}
```

Opgave 6 (15 punten)

Het is mogelijk dat verschillende klanten op hetzelfde moment een bestelling willen plaatsen. Om dit mogelijk te maken, wordt het plaatsen van een bestelling door een klant in een eigen thread gedaan.

```
public class Klant extends Thread {

        Bestelling mijn_bestelling;

        Klant(Bestelling b) {
                mijn_bestelling = b;
        }

        public void run() {
                mijn_bestelling.plaatsBestelling();
        }
}
```

Stel dat we op een gegeven moment 7 flessen cola in voorraad hebben, en dat er 2 klanten zijn, Alice en Bob, die allebei tegelijkertijd een bestelling plaatsen van 3 flessen cola. Na afloop hiervan wordt geprint hoeveel flessen cola er nog in voorraad zijn.

```
public static void main (String [] args) {
        // initialisatiestappen
        // 7 cola in voorraad
        Product c = new Frisdrank(FrisdrankType.cola, 1, 7);
        BestellingsItem bi = new BestellingsItem(c, 3);
        // bestelling b bestaat uit 3 flessen cola
        List<BestellingsItem> l = new ArrayList<BestellingsItem>();
        l.add(bi);
        Bestelling b = new Bestelling(l);
        // alice en bob willen alle twee 3 flessen cola bestellen
        Klant alice = new Klant(b);
        Klant bob = new Klant(b);
        // alice en bob kunnen tegelijkertijd hun bestelling plaatsen
        alice.start();
        bob.start();
        try {
                alice.join();
                bob.join();
        }
        catch (InterruptedException e) { }
        // hoeveel flessen cola zijn er nog in voorraad
        System.out.println(c.getVoorraad());
}
```

a. (*3 punten*) Wat zijn de mogelijke waarden die op het scherm geprint zullen worden? Leg uit hoe deze mogelijke waarden tot stand zijn gekomen.

b. (*3 punten*) Stel dat we in de `main` methode, het try-block

```
try {
        alice.join();
        bob.join();
}
catch (InterruptedException e) { }
```

weglaten. Verandert dit iets aan de mogelijke waarden die op het scherm geprint worden? Zo ja, wat verandert er dan en waarom? Zo nee, waarom niet?

c. (*4 punten*) Stel dat we in de klasse `Klant` de `run` methode als volgt veranderen:

```
public void run() {
        synchronized(this) {
                mijn_bestelling.plaatsBestelling();
        }
}
```

Wat heeft dit voor gevolgen voor de mogelijke waarden die geprint kunnen worden. Waarom?

d. (*5 punten*) Stel dat we in de klasse `Klant` de `run` methode als volgt veranderen:

```
public void run() {
        synchronized(mijn_bestelling) {
                mijn_bestelling.plaatsBestelling();
        }
}
```

Wat heeft dit voor gevolgen voor de mogelijke waarden die geprint kunnen worden. Waarom? En wat gebeurt er als Alice naast 3 flessen cola ook nog iets anders besteld.