Hanzehogeschool Groningen
University of Applied Sciences

Instituut voor Communicatie, Media & IT

Thema 1.2 I/TI: De geprogrammeerde applicatie

Leertaak 1: De Kantine

Inleiding

De komende zes weken wordt in tweetallen gewerkt aan het maken van een kantine-simulatie. De uiteindelijke simulatie is een text-based applicatie waarbij klanten artikelen kunnen kiezen, in de rij gaat staan en afrekenen. Allerlei zaken zoals het betalen met contant geld of pin en het berekenen van omzetgegevens komen gaandeweg aan bod. Elke week bestaat uit een aantal opgaven die aansluiten bij de behandelde stof in het theoriecollege en het practicum: de opgaven vormen een indicatie van de mate waarin de stof van die week beheerst wordt.

In deze opgaven wordt een aantal skeletcodes gegeven. Deze skeletcode kun je als zip downloaden van Blackboard.

We raden je aan om voor de weekopdrachten subversion (svn) te gebruiken. Dit is een versiebeheersysteem voor je broncode en is er erg handig wanneer je met meerdere personen aan een opdracht of project werkt. Je kunt code in– en uitchecken van en naar de server zodat iedereen lokaal, op je eigen PC of laptop, de meest recente versie heeft van de software. Online zijn er diverse sites waar je een gratis account aan kunt maken. Zoek bijvoorbeeld op “free svn repository”. Naast een account heb je een svn-client nodig. Overigens is het gebruik van SVN voor Leertaak 2, Vossen en Konijnen, verplicht.

Voor de weekopgaven gelden de volgende regels:

- ✓ De deadline voor het inleveren van de uitwerking is steeds 18:00 op de eerste maandag volgend op de week waar de opgaven bij horen. Je levert dus bijvoorbeeld je uitwerking van week 1 uiterlijk maandag in week 2 in. De docent zal vragen om een korte demonstratie en uitleg over je uitwerking.
- ✓ Het is niet erg als je de opgaven verdeelt, maar je wordt wel geacht alle uitwerkingen te kunnen toelichten. Een excuus als “Dat heb ik niet gemaakt, dus dat kan ik niet uitleggen” wordt dan ook gezien als het niet hebben gemaakt van de opgave. Ook vormt het maken van de opgaven een goede voorbereiding op de twee practicumtoetsen.
- ✓ Niet aanwezig zijn bij de nabespreking is hetzelfde als het niet hebben gemaakt van de opgave.
- ✓ In principe wordt één cijfer aan een groep gegeven, maar de docent zal hier van afwijken als er sprake is van onevenredige werkverdeling of het niet kunnen uitleggen van een uitwerking.
- ✓ De tutor geeft aan hoe hij of zij de uitwerkingen wil ontvangen (zip, mail of via Blackboard).

– Weekopdracht 1–

In deze week staat een tweetal basisklassen die in de kantinesimulatie een belangrijke rol spelen centraal: Artikel en Persoon. Volgende week bouw je de Kantine-klasse.

Opgave 1 – De klasse Artikel

a. Maak een klasse Artikel waarin de volgende gegevens kunnen worden opgeslagen:

   - ✓ Naam;
   - ✓ Prijs.

   Bedenk zelf wat goede datatypen zijn voor deze gegevens.

b. Je hebt hierboven twee instantievariabelen gedeclareerd. Voordat je ze zinnig zou kunnen gebruiken moet je ze wel initialiseren. Leg uit wat de begrippen declaratie en initialisatie betekenen. Let op: in het BlueJ boek worden declaratie en initialisatie vaak in één keer gedaan.

c. Eén manier om de instantievariabelen een waarde te geven is via de constructor. Maak een constructor die dezelfde gegevens uit vraag a) als parameters heeft en de instantievariabelen de meegegeven waarden geeft, dat wil zeggen een constructor van de vorm

```
public Artikel(... naamArtikel, ... prijsArtikel) { ... }
```

d. Mutator en accessor methoden worden ook wel setters en getters genoemd vanwege de eerste drie letters die deze methoden hebben. Maak getters en setters voor de twee instantievariabelen.

Opgave 2 – De klasse Persoon

a. Maak een klasse Persoon waarin de volgende gegevens kunnen worden opgeslagen:

   - ✓ BSN (BurgerServiceNummer);
   - ✓ Voornaam;
   - ✓ Achternaam;
   - ✓ Geboortedatum (dag, maand en jaar als gehele getallen). Voorbeeld: Als iemand op 28 augustus 1989 is geboren dan is dag=28, maand=8 en jaar=1989;
   - ✓ Geslacht (M/V). Gebruik het datatype `char` om dit gegeven op te slaan.

b. Maak vijf setters voor deze instantievariabelen. Let hierbij op de volgende eisen:

   - ✓ De setter van geboortedatum heeft drie parameters. In deze speciale setter moet je de volgende drie controles uitvoeren:
     - ✓ Dagnummers moeten altijd groter dan of gelijk zijn aan 1;
     - ✓ De maanden liggen tussen 1 en 12;
     - ✓ De jaren liggen tussen 1900 en 2100;

- ✓ De dag/maand combinatie moet bestaan. Zoals je wellicht weet hebben de maanden 1, 3, 5, 7, 8, 10 en 12 31 dagen, maand 2 28 dagen (met uitzondering van schrikkeljaren, zie opgave 5) en resterende maanden 30 dagen. In gewone mensentaal: de geboortedata 34 januari 1987 en 31 april 1992 zijn niet mogelijk, terwijl 31 maart 1990 wel een geldige datum is. (Hint: Je kunt (dus niet verplicht) voor deze controle een `switch-statement` gebruiken, zie bijlage D BlueJ boek. Zie ook bijvoorbeeld http://www.faqs.org/docs/javap/c3/s6.html waarin uitgelegd wordt dat je meerdere case waarden in de switch kan combineren tot één resultaat, iets wat handig is in dit geval).
- ✓ Als minstens één van de bovenstaande controles faalt, maak dan de drie instantievariabelen dag, maand en jaar gelijk aan 0.

- ✓ De setter van geslacht heeft ook een controle nodig. Bedenk zelf welke controle dat is en bouw deze ook in. Doe iets met de waarde van geslacht als de controle mislukt, zodat duidelijk is dat de waarde niet goed is gezet door de setter.

c. Maak een constructor die de gegevens uit vraag a) als parameters heeft en de instantievariabelen de meegegeven waarden geeft. Let op: de controles uit de vorige vraag moet je ook hier uitvoeren! Hoe doe je dat? Er zijn meerdere mogelijkheden, maar voorkom in ieder geval dubbele code!

d. De getter voor geboortedatum wordt gegeven:

```
    /**
     * Getter voor geboortedatum
     * @return Geboortedatum
     */
    public String getGeboorteDatum() {
        String temp;
        if (dag==0 && maand==0 && jaar==0) {
            temp="Onbekend";
        } else {
            temp=dag+"/"+maand+"/"+jaar;
        }
        return temp;
    }
```

Maak de vier overige getters. Let hierbij op het volgende:

- ✓ De getter van geslacht geeft een `String` terug: “Man” of “Vrouw”. Als de waarde van geslacht geen correcte waarde heeft (zie ook vraag b)) retourneer dan “Onbekend”.

Opgave 3 – Afdrukken van artikel- en persoonsgegevens

Voeg aan de klassen Persoon en Artikel een public void drukAf() methode toe waarmee je de waarden van de instantievariabelen laat zien. Hint: gebruik de getters, System.out.print(...) en System.out.println(...).

Opgave 4 – Belangrijk verschil tussen primitieve typen en objecten

Primitieve typen (zie 3.7 en bijlage B BlueJ boek) en objecten zijn twee fundamenteel verschillende soorten typen. Het belangrijkste verschil in gedrag zie hieronder geïllustreerd aan de hand van twee code fragmenten:

```
// primitieve typen
int i=1;
int j=2;
i=j;
j=3;
// drukt ”i=2, j=3” af
System.out.println(“i =”+i+”, j=”+j);

// objecten
// a en b zijn objecten van de klasse Artikel
a.setPrijs(1);
b.setPrijs(2);
a=b;
b.setPrice(3);
// drukt “prijs a=3, prijs b=3” af
System.out.println(“prijs a =”+a.getPrijs()+”, prijs
b=”+b.getPrijs());
```

Verklaar dit verschil in gedrag. Gebruik in je uitleg de uitdrukkingen “directe opslag” en “referentie (verwijzing)”. In het college is hier ook aandacht aan besteed.

Opgave 5 – Schrikkeljaar

In opgave 2b) heb je een controle ingebouwd of de dag/maand combinatie wel kan, maar er is hier nog geen rekening gehouden met schrikkeljaren. In een schrikkeljaar heeft de maand februari 29 in plaats van 28 dagen. Een jaar is een schrikkeljaar als het jaartal deelbaar is door 4, maar als het jaar deelbaar is door 100 is het geen schrikkeljaar, tenzij het jaar deelbaar is door 400. Het jaar 1900 is dus geen schrikkeljaar, de jaren 1996, 2000, 2004 en 2008 zijn dat wel. Bouw deze controle in opgave 2b) in. Gebruik hierbij de modulo(%)-operator.

– Weekopdracht 2 –

In deze week maak je een eerste versie van de kantinesimulatie op basis van de Artikel en Persoon klasse. Voorzie al je code van zinvol commentaar.

De ervaring leert dat je op het verkeerde pad zit wanneer je de gegeven code gaat aanpassen. Het is heel verleidelijk als je ergens vastloopt om wat te gaan “rommelen” met code waardoor je een onbedoeld een domino effect creëert: een oplossing die ergens schijnt te werken zorgt ervoor dat je in allerlei andere code ook aanpassingen moet maken, met als waarschijnlijk resultaat dat je er bij de laatste wijziging achterkomt dat de oorspronkelijke wijziging toch niet handig was. De gegeven code is zo opgezet dat als je de concepten van de week begrijpt, je gaandeweg ook ziet wat de aanvullingen logischerwijs moeten zijn. Kortom, gegeven code niet aanpassen, maar alleen aanvullen.

Opgave 1 – Aanpassen Artikel en Persoon klasse

Voeg aan de klassen Artikel en Persoon, die je vorige week gemaakt hebt. een parameterloze constructor toe. Zorg ervoor dat in de klasse Persoon de instantievariabelen voor geboortedatum en geslacht een waarde krijgen in de parameterloze constructor zodanig dat de getters “Onbekend” teruggeven (zie ook opgave 2d) van vorige week). Maak de namen van de parameters van de constructors met parameters (zie vorige week) gelijk aan de namen van de instantievariabelen. Verander ook de namen van de parameter van de setters in de overeenkomstige namen van de instantievariabelen die worden gezet door de setters. Gebruik hiervoor het sleutelwoord `this`.

Opgave 2 – De klasse Dienblad

Als een persoon de kantine binnenloopt, pakt deze een dienblad, een aantal artikelen en plaatst deze op het dienblad. Hieronder staat de skeletcode staan voor de klasse Dienblad.

```
public class Dienblad {
    private ArrayList<Artikel> artikelen;

    /**
     * Constructor
     */
    public Dienblad() {
        // method body omitted
    }

    /**
     * Methode om artikel aan dienblad toe te voegen
```

```
         * @param artikel
         */
        public void voegToe(Artikel artikel) {
            // method body omitted
        }

        /**
         * Methode om aantal artikelen op dienblad te tellen
         * @return Het aantal artikelen
         */
        public int getAantalArtikelen() {
            // method body omitted
        }

        /**
         * Methode om de totaalprijs van de artikelen
         * op dienblad uit te rekenen
         * @return De totaalprijs
         */
        public double getTotaalPrijs() {
            // method body omitted
        }
    }
```

a. Vul de bovenstaande klasse aan, zodat de methodes die gegeven zijn doen wat ze moeten doen.

b. Implementeer de onderstaande skeletcode in de klasse Persoon; ga er daarbij van uit dat een Persoon maar één dienblad kan dragen.

```
    /**
     * Methode om dienblad te koppelen aan een persoon
     * @param dienblad
     */
    public void pakDienblad(Dienblad dienblad) {
        //method body omitted
    }
```

c. Implementeer de volgende drie methoden in de klasse Persoon. Hou daarbij rekening met de mogelijkheid dat een persoon nog geen dienblad heeft gepakt.

```
    /**
     * Methode om artikel te pakken en te plaatsen op het dienblad
     * @param artikel
     */
    public void pakArtikel(Artikel artikel) {
        //method body omitted
    }

    /**
     * Methode om de totaalprijs van de artikelen
     * op dienblad dat bij de persoon hoort uit te rekenen
     * @return De totaalprijs
     */
    public double getTotaalPrijs() {
           //method body omitted
    }

    /**
     * Methode om het aantal artikelen op dienblad dat bij de
     * persoon hoort te tellen
     * @return Het aantal artikelen
     */
    public int getAantalArtikelen() {
           //method body omitted
    }
```

Opgave 3 – De klasse Kassarij

Nadat een persoon alle gewenste artikelen op het dienblad heeft geplaatst, sluit deze zich achteraan in de rij voor de kassa. De kassarij wordt volgens het First In First Out (FIFO) principe afgewerkt. Hieronder zie je de skeletcode voor de klasse Kassarij:

```
    public class KassaRij {
        /**
         * Constructor
         */
        public KassaRij() {
           //method body omitted
        }

        /**
```

```
        * Persoon sluit achter in de rij aan
        * @param persoon
        */
       public void sluitAchteraan(Persoon persoon) {
          //method body omitted
       }

       /**
        * Indien er een rij bestaat, de eerste Persoon uit
        * de rij verwijderen en retourneren.
        * Als er niemand in de rij staat geeft deze null terug.
        * @return Eerste persoon in de rij of null
        */
       public Persoon eerstePersoonInRij() {
           //method body omitted
       }

       /**
        * Methode kijkt of er personen in de rij staan.
        * @return Of er wel of geen rij bestaat
        */
       public boolean erIsEenRij() {
           //method body omitted
       }
   }
```

Implementeer deze klasse. Gebruik hierbij een `ArrayList<Persoon>` om de personen in op te slaan. Een andere manier van opslaan komt terug in de laatste opgaven van deze week.

Opgave 4 – De klasse Kassa

Hieronder zie je de skeletcode voor de klasse Kassa. Implementeer deze klasse.

```
   public class Kassa {
       /**
        * Constructor
        */
       public Kassa(KassaRij kassarij) {
           //method body omitted
       }

       /**
```

```
        * vraag het aantal artikelen en de totaalprijs op.
        * De implementatie wordt later vervangen
        * door een echte betaling door de persoon.
        * @param persoon die moet afrekenen
        */
       public void rekenAf(Persoon persoon) {
          //method body omitted
       }

       /**
        * Geeft het aantal artikelen dat de kassa
        * heeft gepasseerd,
        * vanaf het moment dat de methode resetWaarden
        * is aangeroepen.
        * @return aantal artikelen
        */
       public int aantalArtikelen() {
          //method body omitted
       }

       /**
        * Geeft het totaalbedrag van alle artikelen die
        * de kassa zijn gepasseerd, vanaf het moment dat de methode
        *resetKassa
        * is aangeroepen.
        * @return hoeveelheid geld in de kassa
        */
       public double hoeveelheidGeldInKassa() {
           //method body omitted
       }

       /**
        * reset de waarden van het aantal gepasseerde artikelen en
        * de totale hoeveelheid geld in de kassa.
        */
       public void resetKassa() {
           //method body omitted
       }
   }
```

Opgave 5 – De klasse Kantine

Hieronder zie je de skeletcode voor de Kantine klasse.

```
public class Kantine {
    private Kassa kassa;
    private KassaRij kassarij;

    /**
     * Constructor
     */
    public Kantine() {
        kassarij=new KassaRij();
        kassa=new Kassa(kassarij);
    }

    /**
     * In deze methode wordt een Persoon en Dienblad
     * gemaakt en aan elkaar
     * gekoppeld. Maak twee Artikelen aan en plaats
     * deze op het dienblad.
     * Tenslotte sluit de Persoon zich aan bij de rij
     * voor de kassa.
     */
    public void loopPakSluitAan() {
        //omitted
    }

    /**
     * Deze methode handelt de rij voor de kassa af.
     */
    public void verwerkRijVoorKassa() {
        while() {
            //omitted
        }
    }

    /**
     * Deze methode telt het geld uit de kassa
     * @return hoeveelheid geld in kassa
     */
    public double hoeveelheidGeldInKassa() {
       //omitted
    }
```

```
    /**
     * Deze methode geeft het aantal gepasseerde artikelen.
     * @return het aantal gepasseerde artikelen
     */
    public int aantalArtikelen(){
       //omitted
    }

    /**
     * Deze methode reset de bijgehouden telling van
     * het aantal artikelen
     * en "leegt" de inhoud van de kassa.
     */
    public void resetKassa() {
       // omitted
    }
}
```

a. Leg uit waarom het gebruik van een `while` lus in de methode `verwerkRijVoorKassa()` handiger is dan een `for` lus.

b. Implementeer de ontbrekende methoden.

Opgave 6 – De klasse KantineSimulatie

In deze opgave ga je de kantine-simulatie starten aan de hand van de volgende code.

```
public class KantineSimulatie {
    private Kantine kantine;

    /**
     * Constructor
     */
    public KantineSimulatie() {
        kantine=new Kantine();
    }

    /**
     * Deze methode simuleert een aantal dagen in het
     * verloop van de kantine
     * @param dagen
  */
```

```
        public void simuleer(int dagen) {
            // for lus voor dagen
            for(...) {
                // per dag nu even vast 10+i personen naar binnen
                // laten gaan, wordt volgende week veranderd...
                // for lus voor personen
                for(int j=0;j<10+i;j++){
                    // kantine.(…);
                }

                // verwerk rij voor de kassa
                // toon dagtotalen (artikelen en geld in kassa)
                // reset de kassa voor de volgende dag
            }
        }
    }
```

a. Vul de ontbrekende delen in. Voer de methode simuleer uit in BlueJ.

b. Teken een objectdiagram.

Opgave 7 - Alternatieve opslagstructuren

De klassen Dienblad en Kassarij gebruiken allebei intern een ArrayList om Artikelen of Personen op te slaan. Je zou kunnen zeggen dat een dienblad een “stapelstructuur” heeft; dat wil zeggen dat het eerste artikel dat er op wordt geplaatst als laatste wordt afgehaald. Dit wordt ook wel een LIFO systeem genoemd, dit betekent Last In First Out. Zoals al eerder opgemerkt in opgave 2) heeft een kassarij juist de omgekeerde eigenschap, namelijk FIFO.

Als je meer ervaring krijgt met programmeren ontdek je dat deze twee structuren veel vaker voorkomen. In de Java bibliotheek in de java.util.* package (zie de documentatie op oracle.com) kun je een `Stack` (stapel, het LIFO systeem) en een `Queue` (rij, het FIFO systeem) terugvinden.

a. Vervang de ArrayList in Dienblad door een Stack<Artikel>. Je kunt in Java niet direct een Queue<Persoon> aanmaken omdat dat een interface is (later tijdens dit thema leer wat dat precies betekent). Gebruik een LinkedList<Persoon> in Kassarij om de ArrayList<Persoon> te vervangen.

– Weekopdracht 3 –

Deze week staat in het teken van het aanpassen (refactoren) en deels uitbreiden van de code van vorige week. Voorzie al je code van zinvol commentaar en Javadoc.

Opgave 1 – Refactoren: dubbele methodes en het gebruik van een iterator

Als je goed kijkt naar de code van de eerste versie van de kantine simulatie valt je misschien op dat er soms methodes zijn die twee keer voorkomen. Eén van die twee methodes is slechts een soort doorgeefluik. Dit kun je efficiënter oplossen.

a. Bij welke methodes in Kassa en Kantine komt dit voor?

b. Verwijder deze methodes in Kantine. Maak een getter voor de private instantie variabele kassa in de klasse Kantine.

c. Als je je project nu compileert krijg je een foutmelding in de klasse KantineSimulatie. Los dit op door de getter uit de vorige vraag te gebruiken.

Hetzelfde probleem komt in Dienblad en Persoon voor. In principe kun je dezelfde manier van oplossen kiezen maar nu gaan we wat rigoureuzer te werk. Je kunt namelijk terecht opmerken dat de klasse Dienblad, laat staan Persoon, de methoden `double getTotaalPrijs()` en `int getAantalArtikelen()` helemaal niet zou moeten bevatten. Immers, Dienblad is niks anders dan een soort container voor artikelen. Het is beter dat de klasse Kassa via een methode `Dienblad getDienblad()` in Persoon het dienblad opvraagt. Vervolgens zou de klasse Dienblad een methode moeten bevatten die een Iterator<Artikel> retourneert waarmee door de artikelen op het dienblad heen gelopen kan worden. Zo kan de klasse Kassa zelf de totaalprijs en het aantal artikelen berekenen.

d. Pas de code van de klassen Dienblad, Persoon en Kassa aan.

e. Ook `pakArtikel(Artikel artikel)` in de klasse Persoon is overbodig. Verwijder deze methode uit Persoon en pas je code ergens anders in de applicatie aan zodat het geheel nog compileert en correct werkt.

Opgave 2 – De klasse KantineAanbod

Je krijgt van ons de klasse KantineAanbod cadeau; deze kun je van Blackboard halen. Voeg deze klasse aan je BlueJ project toe. Zorg dat je de code goed begrijpt.

a. Leg uit waarom het goed is om de methodes `ArrayList<Artikel> getArrayList(String productnaam)` en `Artikel getArtikel(ArrayList<Artikel>)` private te maken.

b. In welke situatie gebruik je een HashMap en wanneer een HashSet?

c. Voeg een instantievariabele kantineaanbod van het type KantineAanbod toe aan de klasse Kantine. Voeg ook een getter en setter voor deze variabele toe.

In de eerste versie van de kantinesimulatie maakt de methode `loopPakSluitAan()` in Kantine zelf een Persoon en een Dienblad aan, om vervolgens twee Artikelen te pakken. In de nieuwe versie van de kantinesimulatie willen we dat de klasse KantineSimulatie zelf een persoon met een dienblad aanlevert, samen met een lijst van artikelnamen die uit het kantineaanbod moeten worden gehaald. Kortom, de signatuur van loopPakSluitAan() verandert in

```
/**
  * In deze methode kiest een Persoon met een dienblad
  * de artikelen in artikelnamen.
  * @param persoon
  * @artikelnamen
  */
public void loopPakSluitAan(Persoon persoon, String[] artikelnamen) {
    //omitted
}
```

d. Implementeer bovenstaande methode.

Opgave 3 – Refactoren van Kantine en KantineSimulatie

Hieronder staat de skeletcode voor een nieuwe versie van de KantineSimulatie. We gaan ervan uit dat er vier verschillende artikelen zijn waarbij de hoeveelheid via de klasse Random wordt bepaald.

```
import java.util.*;

public class KantineSimulatie {
    // kantine
    private Kantine kantine;

    // kantineaanbod
    private KantineAanbod kantineaanbod;

    // random generator
    private Random random;

    // aantal artikelen
    private static final int AANTAL_ARTIKELEN=4;
```

```
    // artikelen
    private static final String[] artikelnamen=
        new String[] {"Koffie","Broodje pindakaas", "Broodje kaas",
"Appelsap"};

    // prijzen
    private static double[] artikelprijzen=
        new double[]{1.50, 2.10, 1.65, 1.65};

    // minimum en maximum aantal artikelen per soort
    private static final int MIN_ARTIKELEN_PER_SOORT=10000;
    private static final int MAX_ARTIKELEN_PER_SOORT=20000;

    // minimum en maximum aantal personen per dag
    private static final int MIN_PERSONEN_PER_DAG=50;
    private static final int MAX_PERSONEN_PER_DAG=100;

    // minimum en maximum artikelen per persoon
    private static final int MIN_ARTIKELEN_PER_PERSOON=1;
    private static final int MAX_ARTIKELEN_PER_PERSOON=4;

    /**
     * Constructor
     */
    public KantineSimulatie(){
        kantine=new Kantine();
        random=new Random();
        int[] hoeveelheden=getRandomArray(
AANTAL_ARTIKELEN,MIN_ARTIKELEN_PER_SOORT, MAX_ARTIKELEN_PER_SOORT);
       kantineaanbod=new KantineAanbod(artikelnamen, artikelprijzen,
       hoeveelheden);
        kantine.setKantineAanbod(kantineaanbod);
    }

    /**
     * Methode om een array van random getallen liggend tussen min en
max
     * van de gegeven lengte te genereren
     * @param lengte
     * @param min
     * @param max
     * @return De array met random getallen
     */
```

```
    private int[] getRandomArray(int lengte, int min, int max) {
        int[] temp=new int[lengte];
        for(int i=0;i<lengte;i++) {
            temp[i]=getRandomValue(min, max);
        }
        return temp;
    }

    /**
     * Methode om een random getal tussen min(incl) en
     * max(incl) te genereren.
     * @param min
     * @param max
     * @return Een random getal
     */
    private int getRandomValue(int min, int max) {
        return random.nextInt(max-min+1)+min;
    }

    /**
     * Methode om op basis van een array van indexen voor de array
     * artikelnamen de bijhorende array van artikelnamen te maken
     * @param indexen
     * @return De array met artikelnamen
     */
    private String[] geefArtikelNamen(int[] indexen) {
        String[] artikelen=new String[indexen.length];
        for(int i=0;i<indexen.length;i++) {
            artikelen[i]=artikelnamen[indexen[i]];
        }
        return artikelen;
    }

    /**
     * Deze methode simuleert een aantal dagen in het
     * verloop van de kantine
     * @param dagen
     */
    public void simuleer(int dagen) {
        // for lus voor dagen
        for(int i=0;i<dagen;i++) {
            // bedenk hoeveel personen vandaag binnen lopen
            int aantalpersonen=...
```

```
                // laat de personen maar komen...
                for(int j=0;j<aantalpersonen;j++) {
                    // maak persoon en dienblad aan, koppel ze
                    // bedenk hoeveel artikelen worden gepakt
                    int aantalartikelen=...

                    // genereer de “artikelnummers”, dit zijn indexen
                    // van de artikelnamen array
      int[] tepakken=getRandomArray(aantalartikelen, 0,
    AANTAL_ARTIKELEN-1);

                    // vind de artikelnamen op basis van
                    // de indexen hierboven
                    String[] artikelen=geefArtikelNamen(tepakken);

                    // loop de kantine binnen, pak de gewenste
                    // artikelen, sluit aan
                }

                // verwerk rij voor de kassa
                // druk de dagtotalen af en hoeveel personen binnen
                // zijn gekomen
                // reset de kassa voor de volgende dag
            }
        }
    }
```

Je ziet in de bovenstaande code het veelvuldige gebruik van static final variabelen. In de eerste opgave van volgende week komen we terug op de precieze werking van de keywords static en final.

a. Leg de werking van de constructor uit.

b. Leg de implementatie van int `getRandomValue(int min, int max)` uit en met name waarom er +1 in voorkomt. Gebruik de Java API. Hint: denk aan de betekenis van inclusief en exclusief.

c. Implementeer de ontbrekende delen van de code van de tweede versie van de kantine-simulator. Roep de methode `simuleer(int dagen)` aan.

Opgave 4 – Aanvullen van de voorraden

In de huidige simulatie is er geen mogelijkheid om de voorraden aan te vullen. Pas je code aan zodat als je onder een bepaald minimum voorraad komt de voorraad

weer tot het beginniveau wordt aangevuld, zonder dat je steeds een nieuwe instantie maakt van KantineAanbod.

– Weekopdracht 4 –

Opgave 1 – De klasse Administratie

Hieronder zie je de skeletcode voor de klasse Administratie. Deze klasse wordt later gebruikt om kassagegevens uit te lezen en een paar statistische berekeningen uit te voeren. De arrays die als parameter worden gebruikt in de methoden worden later aangeleverd door een KantineSimulatie klasse die de kantine over een periode van bijvoorbeeld dertig dagen simuleert. Elke dag levert twee metingen op: het aantal gepasseerde artikelen en de omzet.

a. Implementeer deze klasse. Maak je implementatie wel flexibel; ga dus niet uit van arrays met een omvang van dertig elementen.

```
public class Administratie {
     /**
      * Deze methode berekent van de int array aantal de
      * gemiddelde waarde
      * @param aantal
      * @return het gemiddelde
      */
     public double berekenGemiddeldAantal(int[] aantal) {
        //omitted
     }

     /**
      * Deze methode berekent van de double array omzet de
      * gemiddelde waarde
      * @param omzet
      * @return Het gemiddelde
      */
     public double berekenGemiddeldeOmzet(double[] omzet) {
         //omitted
     }
}
```

Het gemiddelde van een rij getallen is de som van de rij getallen gedeeld door het aantal getallen. Het gemiddelde van een lege rij getallen is 0.

b. Test de methodes met onderstaand verwacht resultaat:

Methode	Input	Verwacht resultaat
berekenGemiddeldAantal	{45, 56, 34, 39, 40, 31}	40.8333

berekenGemiddeldeOmzet	{567.70, 498.25, 458.90}	508.2833

Hint bij berekenGemiddeldAantal: wat gebeurt er als je twee ints deelt? Los het probleem op door te casten naar een double.

c. Er is geen constructor gedefinieerd voor Administratie terwijl je gewoon new `Administratie()` kan aanroepen. Leg uit waarom dat kan.

d. Leg uit waarom de twee al bestaande methoden van Administratie static kunnen zijn. Verander ze in static.

e. We hebben door het static maken van de twee methodes geen instantie meer nodig van Administratie. Het is echter wel mogelijk om een instantie van Administratie aan te maken en daar de static methoden op aan te roepen. Als je dat wil voorkomen kun je een private constructor voor Administratie maken. Doe dat en leg uit waarom je je doel nu bereikt.

De mensen op de administratie willen naast de gemiddelden van het aantal verkochte artikelen en de omzet over een periode een nieuw overzicht zien. Ze zijn geïnteresseerd in de dagtotalen over een periode van de omzet, dat wil zeggen naast het gemiddelde over de hele periode willen ze zeven totalen over de periode, voor elke dag één. Een voorbeeld: stel dat de omzet per dag volgens de onderstaande array verloopt:

```
{321.35, 450.50, 210.45, 190.85, 193.25, 159.90, 214.25, 220.90,
201.90, 242.70, 260.35}
```

We tellen gemakshalve vanaf nul, omdat in Java dat ook gebeurt. Je mag er van uit gaan dat het 'nulde' element van de array de omzet op maandag is, het eerste dinsdag, enzovoort. Na zeven dagen, dus vanaf het zesde element, begin je weer van voor af aan: omzet op maandag, de volgende is dinsdag. We gaan er gemakshalve vanuit dat de kantine zeven dagen per week open is. De totaalomzet op maandag is 321.35+220.90, die van dinsdag=450.50+201.90. Merk op dat in dit voorbeeld vrijdag, zaterdag en zondag maar één keer voorkomen en daarmee de totaalomzet op die dagen gelijk is aan de behaalde omzet op die dagen.

f. Voeg een static methode toe aan Administratie met bovenstaande functionaliteit met onderstaande signatuur. Vul de code aan.

```
/**
 * Methode om dagomzet uit te rekenen
 * @param omzet
 * @return array (7 elementen) met dagomzetten
 */
public static double[] berekenDagOmzet(double[] omzet) {
    double[] temp=new double[7];
    for(int i=0;i<7;i++) {
```

```
                int j=0;
                while(...) {
                    temp[i]+=omzet[i+7*j];
                    //omitted
                }
            }
            return temp;
        }
```

g. Maak een activiteitendiagram van je implementatie van de vorige vraag.

h. In plaats van dat je de "magic constant" 7 gebruikt in de implementatie van `berekenDagomzet(double[] omzet)` kun je ook een final int days_in_week als een private instantievariabele toevoegen. Pas je code aan. Leg uit wat final doet.

i. Als het goed is klaagt de compiler over zoiets als "Cannot make a static reference to the non-static field ...". Leg uit waarom de compiler hierover klaagt.

j. Een manier om het probleem te verhelpen is om het woord final te vervangen door static. Waarschijnlijk compileert het werk nu wel weer, maar is het niet meer goed. Welk "probleem" heb je nu geïntroduceerd? Hint: wat was nou ook alweer de oorspronkelijke aanleiding om days_in_week te introduceren?

k. Voeg nu alsnog final toe en vervang days_in_week door DAYS_IN_WEEK. Het is een conventie in Java om de naam van static final variabelen met hoofdletters te schrijven.

Opgave 2 – Overerving met Student, Docent, KantineMedewerker

In deze opgave maken we drie subklassen van Persoon: Student, Docent en KantineMedewerker. De verschillen staan hier onder opgesomd:

- ✓ Een student heeft een studentnummer en volgt een studierichting;
- ✓ Een docent heeft een vierletterige afkorting en werkt bij een afdeling;
- ✓ Een Kantinemedewerker heeft een medewerkersnummer en een boolean waarde die aangeeft of hij/zij achter de kassa mag staan.

a. Maak de bovenstaande drie subklassen van Persoon, met constructors en getters en setters. Zorg dat in alle drie constructors alle gegevens staan en gebruik een super aanroep in de constructor.

b. Waarom moet een super aanroep in de constructor altijd bovenaan staan?

c. Override de methode `drukAf()` in Persoon in alledrie subklassen waarbij je alleen de extra gegevens van de nieuwe klassen afdrukt.

Opgave 3 – Kantine Simulatie

d. Pas de kantine simulatie van vorige week zodanig aan dat in plaats van een random aantal objecten van het type Persoon de volgende drie type objecten met de genoemde hoeveelheden worden aangemaakt (totaal dus 100 objecten):

   ✓ Een student: 89 instanties;
   ✓ Een docent: 10 instanties;
   ✓ Een kantinemedewerker: 1 instantie.

   Let op: alledrie typen klassen spelen de rol van klant, dat wil zeggen dat de kantinemedewerker niet achter de kassa staat maar zelf artikelen koopt.

e. Gebruik de methode drukAf() in je simulatie zodat je ziet wat voor type persoon de kantine binnenkomt.

f. Roep de drie methodes van Administratie aan en druk het resultaat af.

Opgave 4 – Random soorten bezoekers

In plaats van dat je in opgave 3a) een vast aantal bezoekers van de kantine maakt is het realistischer om het aantal bezoekers nog steeds random te genereren – zoals in week 3 – en de opgegeven aantallen te interpreteren als kansen:

   ✓ Een student wordt aangemaakt met een kans 89 op 100;
   ✓ Een docent wordt aangemaakt met een kans 10 op 100;
   ✓ Een kantinemedewerker wordt aangemaakt met een kans 1 op 100.

– Weekopdracht 5 –

Opgave 1 – Overriding en equals

a. We willen de `drukAf()` methode vervangen door het overriden van de methode `String toString()`. Doe dit en gebruik super daar waar nodig in de opbouw van je uitkomst van de te retourneren string.

b. Implementeer de `equals(Object object)` methode in Persoon.

c. Maak een klasse PersoonsVergelijker en gebruik een `public static void main(String[] args)` methode waarin je twee identieke Persoon objecten maakt via new; identiek wil zeggen dat de waarden van alle eigenschappen van de objecten hetzelfde is. Test op == en equals. Wat concludeer je over het verschil tussen het van deze twee manieren?

d. Als je twee string inhoudelijk met elkaar wil vergelijken, moet je dan == of equals(Object object) gebruiken. Licht je antwoord toe. Geef aan wat er gebeurt als je de andere mogelijkheid zou kiezen.

e. Run je klasse PersoonsVergelijker vanaf de commandline.

Opgave 2 – Betalen doe je zo

In deze opgave ga je de daadwerkelijke betaling van een persoon implementeren. Een betaling kan op twee manieren gebeuren: contant of met een pinpas. In principe kun je een interface Betaalwijze introduceren, maar bij beide betaalwijzes zou je wel het tegoed kunnen opslaan. We gaan ervan uit dat een betaling met een pinpas gebeurt vanaf een rekening waar een kredietlimiet op zit. Kredietlimieten bestaan niet voor een contante betaling, tenminste de bodem van je portemonnee is de kredietlimiet. Kortom, we introduceren een abstracte klasse Betaalwijze als volgt:

```
public abstract class Betaalwijze {
    protected double saldo;

    /**
     * Methode om krediet te initialiseren
     * @param krediet
     */
    public void setSaldo(double saldo){
       this.saldo=saldo;
    }

    /**
     * Methode om betaling af te handelen
     * @param tebetalen
```

```
     * @return Boolean om te kijken of er voldoende saldo is
     */
     public abstract boolean betaal(double tebetalen);
}
```

Daarnaast zijn er twee concrete subklassen Contant en Pinpas die deze abstracte superklasse extenden.

```
public class Contant extends Betaalwijze {
    /**
     * Methode om betaling af te handelen
     */
     public boolean betaal(double tebetalen) {
       // omitted
     }
}
```

en

```
public class Pinpas extends Betaalwijze {
    private double kredietlimiet;

    /**
     * Methode om kredietlimiet te zetten
     * @param kredietlimiet
     */
    public void setKredietLimiet(double kredietlimiet) {
        //omitted
    }

    /**
     * Methode om betaling af te handelen
     */
    public boolean betaal(double tebetalen) {
       //omitted
    }
}
```

Merk trouwens op dat een Persoon een referentie heeft naar een abstracte klasse Betaalwijze. Dit is een ‘heeft-een’ relatie, dus er is bestaat geen overervingsstruc-

tuur tussen Persoon en Betaalwijze. Je zou inderdaad niet kunnen beweren dat een persoon een betaalwijze is, maar wel dat een persoon een betaalwijze heeft.

a. Implementeer deze drie klassen.

b. Stel dat bij de pinpasbetaling zou worden gecommuniceerd met een Bank object. Teken een sequentie-diagram waaruit blijkt hoe je de methode `boolean betaal(double tebetalen)` in Pinpas implementeert. Hint: vergeet niet hoe de klasse Pinpas een referentie naar een Bank object heeft gekregen.

c. Waarom is de instantie variabele saldo protected gemaakt? Waarom is dat handig?

d. Maak een private instantie variabele betaalwijze in Persoon. Maak een getter en een setter. Aanpassen van de constructor is niet nodig.

e. Pas de code in Kassa aan zodat bij de betaling naar de betaalwijze van de persoon wordt gevraagd en de methode `betaal(double tebetalen)` wordt aangeroepen. Indien de betaling faalt laat dan een melding zien op het scherm.

Opgave 3 – Kortingskaarthouder

De docenten en kantinemedewerkers zijn in de gelukkige omstandigheid dat ze korting krijgen op de aangeschafte artikelen, voor docenten is dat 25%, met een maximale korting van 1.00 Euro per kantinebezoek en kantinemedewerkers krijgen zelf 35% korting, zonder een maximumbedrag. Dit is te realiseren door de onderstaande interface te definiëren:

```
public interface KortingskaartHouder {
    // methode om kortingspercentage op te vragen
    public double geefKortingsPercentage();
    // methode om op te vragen of er maximum per keer aan de korting
zit
    public boolean heeftMaximum();
    // methode om het maximum kortingsbedrag op te vragen
    public double geefMaximum();
}
```

Doordat je zou kunnen zeggen dat docenten en kantinemedewerkers kortingskaarthouders zijn is het verstandig om de Docent en KantineMedewerker klasse deze interface te laten implementeren.

a. Pas de Docent en KantineMedewerker klasse aan zodanig dat ze de interface KortingskaartHouder implementeren.

b. Pas de code in de klasse Kassa aan zodat er gecheckt wordt of een Persoon ook een kortingskaart heeft (in objecttermen kortingskaarthouder is) en daar re-

kening mee wordt gehouden bij de betaling. Hint: gebruik de instanceof operator en casting.

Opgave 4 – Uitgebreid klassediagram

BlueJ tekent automatisch een klassediagram, maar er zijn details weggelaten. Maak zelf een klassediagram van de applicatie tot nu toe op basis van het klassediagram dat door BlueJ is gegenereerd. De onderstaande uitbreidingen dien je toe te voegen:

- ✓ instantievariabelen en methoden;
- ✓ scope met +, - en #;
- ✓ onderscheid tussen statische en niet statische instantievariabelen en methoden;
- ✓ het gebruik van stereotypes daar waar nodig.

Opgave 5 – Theorie over (abstracte) klassen en interfaces

a. Kun je een instantie maken van een interface via new? Leg uit waarom het logisch is dat het wel of niet kan.

b. Herhaal de vorige vraag met abstracte klassen.

c. Kan een klasse meerdere klassen overerven?

d. Kan een klasse meerdere interfaces implementeren?

e. Kan een klasse tegelijk een klasse overerven en interfaces implementeren?

f. Klopt de stelling dat elke methode in een interface abstract is? Licht je antwoord toe.

g. Moet een klasse abstract zijn als minstens één methode abstract is? Licht je antwoord toe.

h. Leg het begrip polymorfisme van klassen uit en geef twee voorbeelden (één met abstracte klassen en één met interfaces).

Opgave 6 – Een paar doordenkers

a. Kan een klasse abstract zijn als geen enkele methode abstract is in die klasse? Probeer het eens uit. Leg waarom het logisch is dat dit wel of niet kan.

b. Moet een subklasse van een abstracte klasse altijd alle abstracte methodes implementeren? Leg uit waarom het logisch is dat dit wel of niet kan.

c. Als een klasse niet alle methoden van een interface implementeert kun je iets doen om een (compiler)fout te voorkomen. Wat? Waarom is de oplossing logisch?

d. Leg uit waarom het logisch is dat een instantie variabele niet abstract kan zijn.

e. (Uitdaging) Zoek uit wat een final methode is. Leg daarna uit waarom het logisch is dat een methode niet tegelijkertijd abstract en final kan zijn.

– Weekopdracht 6 –

Opgave 1 – TeWeinigGeldException maken

In opgave 2 van vorige week heb je een abstracte klasse met twee concrete subklassen gemaakt. De methode `betaal(double tebetalen)` geeft een boolean terug als indicatie dat de betaling wel of niet is gelukt.

a. Maak een eigen TeWeinigGeldException, met drie constructors:

- ✓ TeWeinigGeldException()
- ✓ TeWeinigGeldException(Exception e)
- ✓ TeWeinigGeldException(String message)

b. Verander het return type van de methode `betaal(double tebetalen)` in void en voeg een throws declaratie toe waarbij een `TeWeinigGeldException(String message)` wordt gethrowd als er onvoldoende saldo is. Als de betaalmethode een TeWeinigGeldException gooit wordt deze gevangen in KantineSimulatie. Zorg dat in het catch-blok de naam van diegene die te weinig geld heeft wordt afgedrukt.

c. Pas de code in Kassa aan zodat de TeWeinigGeldException wordt gevangen.

Opgave 2 – Artikelen aanmaken

a. Tot nu toe heb heeft de klasse KantineSimulatie de artikelen aangemaakt. Pas de code zo aan dat je een eigen artikel kan toevoegen aan de applicatie. Verwijder de voorgedefinieerde artikelen in de kantinesimulatie en static variabelen daar waar nodig.