**Opgave 1.** (2 pt – 14 ln[1]) Schrijf een Booleaanse functie *minstensDeHelft*(*lijst*) die nagaat of een lijst van personen voor minstens de helft uit volwassenen bestaat. Stop hierbij met het doorlopen van de lijst zodra je zeker bent dat er voldoende volwassenen zijn, of te veel 'onvolwassenen'.

Een persoon wordt voorgesteld als een object van de klasse *Persoon* met daarin o.a. een methode *getLeeftijd* die de leeftijd van die persoon teruggeeft als een geheel getal. Met volwassen bedoelen we hier iemand met een leeftijd van minstens 18.

**Oplossing:**

```
public minstensDeHelft (List<Persoon> list) {
    int pos = 0;
    int oud = 0;
    int jong = 0;
    while (2*oud < list.size() && 2*jong < list.size()) {
        if (list.get(pos).getLeeftijd() >= 18) {
            oud ++;
        } else {
            jong ++;
        }
        pos ++;
    }
    return 2*oud >= list.size();
}
```

Er zijn varianten mogelijk op deze oplossing, maar behalve dat er correcte logica gebruikt wordt (ook voor lijsten met een oneven aantal elementen!) is het hier ook belangrijk dat een while-lus wordt gebruikt om de lus vroegtijdig te stoppen.

---

**Opgave 2.** (2 pt – 12 ln) Schrijf een methode *verwijderVolwassenen*(*lijst*) die een nieuwe lijst retourneert met daarin alle volwassen personen uit de gegeven lijst *lijst* , en tegelijk alle volwassenen uit *lijst* verwijdert. Maak hierbij **verplicht** gebruik van een iterator.

**Oplossing:**

```
public ArrayList<Persoon> verwijderVolwassenen(ArrayList<Persoon> list) {
    Iterator<Persoon> iterator = list.iterator();
    ArrayList<Persoon> resultaat = new ArrayList<> ();
    while (iterator.hasNext()) {
       Persoon persoon = iterator.next();
       if (persoon.leeftijd() >= 18) {
            resultaat.add (persoon);
            iterator.remove();
       }
    }
    return result;
}
```

Vergeet de scheve haken niet bij de declaraties van de lijsten.

[1] We geven telkens het aantal lijnen JAVA-code aan die een standaardoplossing voor deze opgave bevat, blanco lijnen niet meegerekend. Jouw oplossing kan (en mag) ook korter of langer zijn. Dit lijnenaantal dient als indicatie van hoeveel antwoord er van jou wordt verwacht.

**Opgave 3.** (2 pt – 12 ln) Onderstaande formule geeft de waarde van $\cosh x$ voor een reëel getal $x$.

$$\cosh x = 1 + \frac{x^2}{2 \cdot 1} + \frac{x^4}{4 \cdot 3 \cdot 2 \cdot 1} + \frac{x^6}{6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1} + \cdots$$

Schrijf een functie *cosh* met als parameter $x$ waarmee je de waarde van $\cosh x$ berekent op bovenstaande manier. Doe dit door achtereenvolgens termen toe te voegen aan een resultaat. Je mag (moet) stoppen zodra de laatste term die je hebt toegevoegd kleiner is dan 0.0000001.

Je mag voor het berekenen van de machten van $x$ *niet* gebruikmaken van de methode *Math.pow*. Let op dat je de tellers en noemers van de opeenvolgende breuken *niet* telkens helemaal opnieuw uitrekent.

**Oplossing:**

```
public double cosh(double x) {
    double som  = 1.0;
    double term = 1.0;
    double x2 = x * x;
    int n = 1;
    while (term >= 0.0000001) {
        term = term * x2 / n / (n+1);
        n += 2;
        som += term;
    }
    return som;
}
```

Het kan ook zonder de variabele *x2*. Opnieuw is het belangrijk dat je hier een while-lus gebruikt (en geen for-lus).

---

**Opgave 4.** (3 pt) Zij $A$ een *vierkante* matrix. Zoals je weet, bekom je de getransponeerde $A^T$ van die matrix door $A$ om zijn hoofddiagonaal te spiegelen:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \quad \Longrightarrow \quad A^T = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

**A.** (10 ln) Schrijf een methode *getransponeerde*(*matrix*) die de getransponeerde teruggeeft van de gegeven *matrix*, een 2-dimensionale tabel (= array) van kommagetallen. Hierbij moet je een *nieuwe* matrix retourneren en moet de oorspronkelijke *matrix* ongewijzigd blijven.

**B.** (9 ln) Schrijf een methode *transponeer*(*matrix*) die *matrix* vervangt door zijn getransponeerde. Deze methode wijzigt dus de inhoud van *matrix*! Je mag hierbij geen gebruik maken van een bijkomende hulptabel of -lijst en ook niet van de methode *getransponeerde* uit deel A van deze opgave.

In beide gevallen mag je aannemen dat de matrix die je als parameter meekrijgt, een *vierkante* matrix is, m.a.w., evenveel rijen heeft als kolommen. Je hoeft dit niet te controleren.

**Oplossing A:**

```
public double[][] transposed(double[][] matrix) {
    int len = matrix.length;
    double[][] resultaat = new double[len][len];
    for (int i=0; i < len; i++) {
       for (int j=0; j < len; j++) {
          resultaat[i][j] = matrix[j][i];
       }
    }
    return resultaat;
}
```

**Oplossing B:**

```
public void transpose(double[] matrix) {
    for (int i=1; i < matrix.length; i++) {
       for (int j=0; j < i; j++) {
          double tmp = matrix[i][j];
          matrix[i][j] = matrix[j][i];
          matrix[j][i] = tmp;
       }
    }
}
```

De binnenste lus kan ook starten bij $j$=$i$+1 en doorlopen tot *matrix.length*, maar dan moet de buitenste lus starten bij $i$=0.

Merk op dat dit een procedure is, en er dus geen retourwaarde hoeft gegeven te worden. (Maar dit hebben we bij het verbeteren door de vingers gezien.)

---

**Opgave 5.** (3 pt – 26 ln) In een bepaalde toepassing komt het vaak voor dat we tabellen van strings nodig hebben waar heel veel herhaling in zit, zoals in het volgende voorbeeld:

```
String[] tabel = {
    "appel", "appel", "appel", "appel", "appel", "peer", "peer",
    "banaan", "banaan", "banaan", "banaan", "banaan", "peer"
};
```

Om het gemakkelijker te maken om dergelijke tabellen aan te maken, gebruiken we een klasse *Builder*, op de volgende manier:

```
Builder builder = new Builder ();
builder.add ("appel", 3); // string + aantal keer
builder.add ("appel", 2);
builder.add ("peer", 2);
builder.add ("banaan", 5);
builder.add ("peer", 1);
String[] tabel = builder.get ();
```

Het eindresultaat van deze reeks opdrachten is een tabel met exact dezelfde inhoud als in het bovenstaande voorbeeld.

**Opgave** Schrijf de klasse *Builder* zodat ze op bovenstaande manier kan gebruikt worden: opeenvolgende oproepen van *add* geven telkens aan welke string er in de tabel gezet moet worden en hoeveel keer, en de methode *get* maakt dan de tabel uiteindelijk aan en retourneert die.

Hou (**verplicht**) twee lijsten bij: één met alle strings, en één met de overeenkomstige aantallen.

**Oplossing:**

```
public class Builder {

    private ArrayList<String> strings;
    private ArrayList<Integer> aantallen;
    private int totaal;

    public Builder () {
        strings = new ArrayList<> ();
        aantallen = new ArrayList<> ();
        totaal = 0;
    }

    public void add (String string, int aantal) {
        strings.add(string);
        aantallen.add(aantal);
        totaal += aantal;
    }

    public String[] get () {
        String[] resultaat = new String[totaal];
        int pos =  0;
        for (int i = 0; i < strings.size(); i++) {
            for (int j = 0; j < aantallen.get(i); j++) {
                resultaat[pos] = strings.get(i);
                pos ++;
            }
        }
        return resultaat;
    }
}
```

Het is ook mogelijk om het *totaal* pas te berekenen als eerste stap(pen) in de methode *get*.

---

6. Voor deze opgave bekijken we een gezelschapsspel dat lijkt op Monopolie. Pionnen (= spelers) in dit spel verplaatsen zich op *velden* van een spelbord. Verschillende soorten velden spelen een verschillende rol. In een aantal gevallen moet de speler betalen wanneer hij op een veld komt.

- Komt de speler op een *Boete*-veld terecht, dan moet hij een bedrag betalen tussen 100 en 200 EURO (inclusief). Dit bedrag is willekeurig (*random*) en kan telkens anders zijn wanneer een speler op dat veld terechtkomt.

- Komt de speler op een *Straat*-veld terecht, dan hangt het bedrag dat hij moet betalen af van hoeveel huizen er op dit veld zijn geplaatst. Voor een straat zonder huizen wordt een *basisbedrag* aangerekend, en per huis wordt daarbij telkens de helft van dit basisbedrag opgeteld. Verschillende straatvelden hebben verschillende basisbedragen. (In een realistisch spel moet je niet betalen wanneer je eigenaar bent van een straat. Hier houden we daarmee *geen* rekening.)

Het hoofdprogramma van dit spel gebruikt een interface *Veld* en twee klassen *Boete* en *Straat* die aan deze interface voldoen. Het spelbord wordt in het hoofdprogramma voorgesteld als een tabel *velden* van het type '*Veld*[]'. Alle velden worden bij het opstarten van het programma ingevuld. Wanneer een speler op een boete- of straatveld terechtkomt met nummer $i$, zal het hoofdprogramma het bedrag dat de speler moet betalen, bepalen met behulp van de volgende opdracht:

**int** *bedrag* = *velden*[*i*].*teBetalen* ();

**Opgave** (4 pt – 23 ln.) Schrijf de volledige broncode voor deze interface en deze twee klassen. Het hoofdprogramma zelf hoef (mag/kan) je *niet* (te) schrijven.

- Zorg dat de klassen en hun objecten kunnen gebruikt worden zoals hierboven beschreven.
- Definieer, indien nodig, constructoren met voldoende parameters zodat in het begin van het programma de vakjes kunnen worden 'opgevuld' zoals beschreven. Hou er rekening mee dat er verschillende straten zijn met verschillende basisbedragen.
- Tijdens de loop van het spel kan er een huis worden toegevoegd aan een straat. Het exacte mechanisme dat hiervoor gebruikt wordt, is niet belangrijk, maar zorg er wel voor dat de klasse *Straat* dit toelaat. (Bij aanvang staan er geen huizen op de straten.)
- Hou het kort: laat alles weg wat niet strikt noodzakelijk is.
- Bedragen zijn steeds in EURO. Indien nodig wordt er naar beneden afgerond.

**Oplossing:**

```
public interface Veld {
    int teBetalen();
}

public class Boete implements Veld {

    private static final Random RG = new Random();

    public int teBetalen () {
        return 100 + RG.nextInt(101);
    }
}

public class Straat implements Veld {

    private int huizen;
    private int basisBedrag;

    public Straat (int basisBedrag) {
        this.basisBedrag = basisBedrag;
        this.huizen = 0;
    }

    public int teBetalen () {
        return basisBedrag + huizen * basisBedrag / 2;
    }

    public void extraHuis() {
        huizen ++;
    }
}
```

De *random generator* moet een klassenvariabele zijn (static).

Opletten bij de deling door 2! Eerst het aantal *huizen* delen door 2 heeft niet het correcte resultaat, want het is een *gehele* deling.

**Opgave 7.** (1 pt – 10 ln.) Een beginnend programmeur schreef de volgende methode waarmee hij telt hoeveel hoofdletters een bepaalde string bevat. Hij gebruikt hier echter een 'for each' op een plaats waar dit niet mag.

```
public int telHoofdletters (String str) {
    int aantal = 0;
    for (char ch: str) {
        if (Character.isUpperCase(ch)) {
            aantal ++;
        }
    }
    return aantal;
}
```

Herschrijf deze methode zodat ze wel correct werkt (en volg daarbij zoveel mogelijk het origineel.)

**Oplossing:**

```
public int telHoofdletters (String str) {
    int aantal = 0;
    for (int pos=0; pos < str.length(); pos ++) {
       char ch = str.charAt(pos);
       if (Character.isUpperCase(ch)) {
          aantal ++;
       }
    }
    return aantal;
}
```