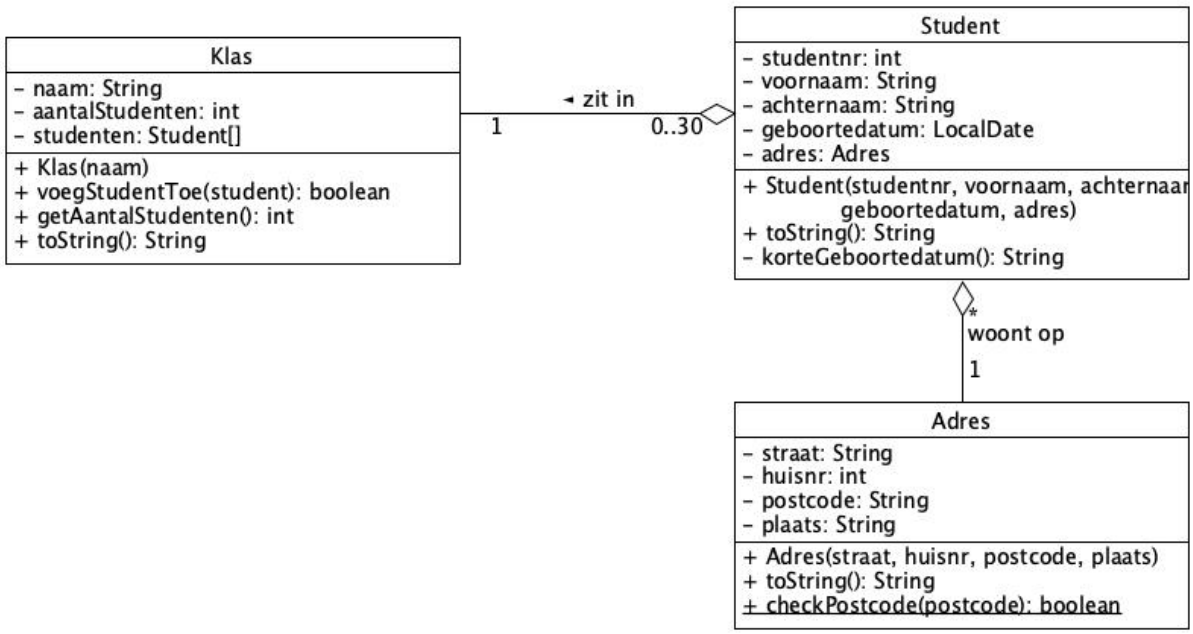
OOP1 – Practicumopdracht 3

Student

Inleiding

Bij deze opdracht maak je een klassenstructuur voor studenten van HBO-ICT. Het UML class diagram ziet er als volgt uit:



In deze opdracht maak je drie classes die als volgt met elkaar te maken hebben: een **`Klas`** bestaat uit een aantal Studenten, maximaal 30. Een **`Student`** heeft onder andere een **`Adres`**.

Opdracht a: class Adres

- Maak een class **`Adres`** aan met de attributen zoals in het klassendiagram weergegeven.
- Deze class heeft de volgende methoden:
  - Constructor: heeft vier argumenten en initialiseert hiermee de attributen.
  - Methode **`toString()`**: geeft een String terug met daarin het adres met de volgende opmaak (voorbeeld):
    `Wibautstraat 2, 1091GM Amsterdam`
  - Static methode **`checkPostcode(postcode)`**: checkt of een postcode een geldige Nederlandse postcode is. Geeft true terug als de postcode geldig is, anders false. Een postcode is geldig als het 1e karakter van de postcode tussen 1 en 9 zit, het 2e t/m 4e karakter van de postcode tussen 0 – 9 zitten, en het 5e en 6e karakter tussen de 'A' en de 'Z' zitten.
    TIP: test als eerste de lengte van de postcode. Een postcode kan alleen geldig zijn als hij uit 6 karakters bestaat.
- Test deze class door in de class **`Main`** een nieuw adres aan te maken en deze vervolgens af te drukken met behulp van de methode **`toString()`**.

- Test in de class **Main** vervolgens ook of de methode **checkPostcode(postcode)** werkt, door deze methode aan te roepen met verschillende geldige en niet geldige postcodes en de uitkomst van de methode af te drukken, bijvoorbeeld met:

```
String postcode = "1234AB"; // geldige postcode
System.out.println(postcode + ": " +
      Adres.checkPostcode(postcode));
postcode = "0123AB";        // ongeldige postcode
System.out.println(postcode + ": " +
      Adres.checkPostcode(postcode));
postcode = "123AB";         // ongeldige postcode
System.out.println(postcode + ": " +
      Adres.checkPostcode(postcode));
postcode = "1234A0";         // ongeldige postcode
System.out.println(postcode + ": " +
      Adres.checkPostcode(postcode));
```

Opdracht b: class Student

- Maak een class **Student** aan met de attributen zoals in het klassendiagram weergegeven.
- Deze class heeft de volgende methoden:
  - Constructor: heeft vijf argumenten en initialiseert hiermee de attributen.
  - Methode **String korteGeboortedatum()**: geeft een String terug met daarin de datum als dag-maand-jaar, bijvoorbeeld **1-12-1998**. Dit is een zogenaamde "helper-method" die handig is om te gebruiken in de methode **toString()**.
  - Methode **String toString()**: geeft een String representatie terug van een **Student** met de volgende opmaak (voorbeeld):
    500812345 Jan Janssen (1-12-1998)
    Adres: Wibautstraat 2, 1091GM Amsterdam
    TIP: maak hierbij handig gebruik van de **toString()** methode van **Adres**. Schrijf zo min mogelijk dubbele code!
- Test deze class door in de class **Main** een nieuwe student aan te maken en deze vervolgens af te drukken met behulp van de methode **toString()**. Gebruik als adres voor de student het adres dat je bij opdracht a hebt aangemaakt.

Opdracht c: class Klas

- Maak een class **Klas** aan met de attributen zoals in het klassendiagram weergegeven.
- Deze class heeft de volgende methoden:
  - Constructor : heeft één argument (naam) en initialiseert hiermee dat attribuut. Een groep begint altijd leeg, dus zorg dat de attributen **aantalStudenten** en **studenten** de juiste waarde krijgen.
    TIP: maak de array wel vast aan, maar zet er nog niets in, dat komt later. Dus een array van 30 lege studenten.
  - Methode **boolean voegStudentToe(Student student)**: voegt de student die als argument wordt meegegeven toe aan de groep. De student wordt dus op de juiste plek in de array gestopt, en de methode geeft dan **true** terug,

TIP: zorg dat je **aantalStudenten** ook aanpast.
TIP: test of de groep vol is. Bij een volle groep mag geen student worden toegevoegd. De methode geeft dan **false** terug.
TIP: gebruik voor het maximale aantal studenten in een klas een constante **MAX_AANTAL_STUDENTEN**, zodat je bij het testen van de class dit aantal gemakkelijk kunt wijzigen.

- Methode **getAantalStudenten()**: heeft de standaardfunctionaliteit van een getter.
- Methode **String toString()**: geeft een String representatie terug van een **Klas** met de volgende opmaak (voorbeeld):

```
Klas IS101 (2 studenten)
500812345 Jan Janssen (1-12-1998)
Adres: Wibautstraat 2, 1091GM Amsterdam
500839104 Piet Pietersen (22-10-1999)
Adres: Wilhelminalaan 14, 1441EL Purmerend
```

Dus ook alle studenten uit de klas moeten door **toString()** worden teruggegeven.
Maak gebruik van **StringBuilder** voor het opbouwen van de return-value.
TIP: maak hierbij handig gebruik van de **toString()** methode van **Student**.
Schrijf zo min mogelijk dubbele code!

- Test deze class door in de class **Main** een nieuwe klas aan te maken en hier de student die je bij opdracht b hebt aangemaakt aan toe te voegen. Druk vervolgens de klas af met behulp van de methode **toString()**.
- Zet vervolgens MAX_AANTAL_STUDENTEN in de class **Klas** op 1. Probeer vanuit de class Main een tweede student toe te voegen en druk daarna de klas weer af met behulp van toString(). Gebruik de debugger van IntellIJ om te controleren of je methode voegStudentToe() correct werkt. Als het goed is wordt de 2e student niet toegevoegd aan de klas.
- Zet vervolgens MAX_AANTAL_STUDENTEN in de class **Klas** terug op 30 en run je code opnieuw. Als het goed is wordt de 2e student nu wel toegevoegd aan de klas.

Opdracht d: class Main

Nu alle klassen getest zijn, kun je in de class **Main** een applicatie te schrijven om de klassenstructuur te gebruiken:

- Vraag de gebruiker om de naam van de klas en sla deze op als een object **Klas**.
- Vraag de gebruiker om de studenten in de klas. Gebruik een loop om de gegevens van alle studenten te vragen. Vraag per student studentnummer, voornaam, achternaam, geboortedatum en zijn adresgegevens. Sla elke student op als een object **Student** en een daaraan gekoppeld object **Adres**. Voeg de **Student** toe aan de **Klas**.
  Tip: De ingegeven geboortedatum (**String**), moet je opdelen in dag, maand en jaar om er een **LocalDate** van te kunnen maken. Je kunt hiervoor bijvoorbeeld de methode **split()** uit de **String** class gebruiken. De Strings dag, maand en jaar vervolgens omzetten naar een **int** kan met de methode **Integer.parseInt()**.
  Tip: vergeet niet de buffer te legen voordat je de methode **nextLine()** van **Scanner** gebruikt.

- Controleer, nadat de gebruiker een postcode heeft ingevoerd, of deze postcode correct is. Gebruik hiervoor de static methode **`checkPostcode()`** in **`Adres`**.
- Zorg dat de loop stopt als de gebruikers als studentnummer het getal 0 invoert, of als het maximale aantal studenten is bereikt.
  Tip: Gebruik hiervoor de constante **`MAX_AANTAL_STUDENTEN`** en de methode **`getAantalStudenten()`** van de class **`Klas`**.
- Print als alle studenten zijn ingevoerd de klas, met alle studenten. Gebruik hiervoor de methode **`toString()`** van **`Klas`**.

Richtlijnen bij coderen (zie ook HBO-ICT code conventions [ICC])

- Zorg dat je naam en het doel van het programma bovenin staan (ICC #1).
- Gebruik de juiste inspringing (*indentation*) bij de lay-out (ICC #2) .
- Let op juist gebruik hoofdletters en kleine letters (ICC #3).
- Gebruik goede namen (ICC #4).
- Vermijd *magic number*s (ICC#5).
- Gebruik javadoc tags: **`@author`**, **`@param`** en **`@return`** (ICC #6).
- Voeg waar nodig commentaar toe die inzicht geven in je code (ICC#7).
- Vermijd dode code (ICC #8).
- Denk aan *encapsulation* (ICC #9).