Inhoud

Eindtoets

Eindtoets

INTRODUCTIE

Deze eindtoets is bedoeld als voorbereiding op het tentamen van de cursus Objectgeoriënteerd programmeren in Java 1 en is te beschouwen als proeftentamen. Het is belangrijk dat u de eindtoets pas probeert te maken op het moment dat u denkt klaar te zijn met de tentamenvoorbereiding. Hebt u over dat laatste nog twijfels, bekijk dan nog eens de leerdoelen en bestudeer de samenvattingen in de leereenheden om te ontdekken welke onderdelen u nog onvoldoende beheerst.

Toegestane hulpmiddelen

Tijdens het tentamen mag het cursusmateriaal geraadpleegd worden. Dat cursusmateriaal hoeft niet 'schoon' te zijn, maar mag ook aantekeningen en dergelijke bevatten. Het raadplegen van ander materiaal is niet toegestaan.

Tentamenduur

Een tentamen duurt drie uur. We adviseren u dan ook de eindtoets binnen een aaneengesloten periode van drie uur te maken.

Samenstelling

Het aantal opgaven, de moeilijkheidsgraad en de verdeling over de leerstof komen overeen met het tentamen. Het tentamen bestaat uit vier open vragen. Opgave 1 gaat vooral over de theorie. In opgave 2 wordt gevraagd een klasse uit de API te gebruiken die nog niet uit de cursus bekend is (uiteraard wordt het relevante gedeelte van de interface gegeven). Opgave 3 bevat het ontwerp van een volledige klasse. In opgave 4 ligt de nadruk op algoritmiek.

Terugkoppeling

De antwoorden op de opgaven staan in de terugkoppeling. We willen echter benadrukken dat u het meest leert als u eerst de opgaven maakt en pas daarna de antwoorden controleert.

Beoordeling

Per opgave kunnen 25 punten behaald worden; in totaal dus maximaal 100 punten halen. Het aantal punten voor elk onderdeel staat vermeld in de marge. Voor een voldoende voor het tentamen moet u tenminste 55 punten behalen.

Studeeraanwijzingen
De studielast van deze eindtoets bedraagt circa 4 uur, inclusief het nakijken van de opgaven aan de hand van de terugkoppeling.

Opgaven

OPGAVE 1

Gegeven is de klasse Punt, als volgt.

```
public class Punt {
  private int x = 0;
  private int y = 0;

  public Punt(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public int getX() {
    return x;
  }

  public int getY() {
    return y;
  }

  /**
   * Telt dit punt en punt p bij elkaar op door de
   * x- en y-coordinaten bij elkaar op te tellen.
   * @param p  een punt
   * @return som van dit punt en p
   */
  public Punt plus(Punt p) {
    x = x + p.getX();
    y = y + p.getY();
    return this;
  }
}
```

De volgende opdrachten worden verwerkt.

```
Punt p1 = new Punt(2, 5);
Punt p2 = new Punt(2, 5);
Punt p3 = p1;
Punt p4 = p1.plus(p2);
```

4 punten a Geldt na verwerking van deze opdrachten p1 == p2? Licht uw antwoord kort toe.

7 punten b Teken het toestandsdiagram met variabelen p1, p2, p3 en p4 na verwerking van het getoonde fragment.

7 punten c Stel dat bij het intypen van deze klasse het type van het attribuut x per ongeluk als Integer is aangemerkt:

```
  private Integer x = 0;
  private int y = 0;
```

De klasse werkt als voorheen.
Geef precies aan waar in de code van klasse Punt autoboxing plaatsvindt, en waar auto-unboxing (welke waarde betreft het en wat gebeurt daarmee)?

7 punten

d Hoewel de terugkeerwaarde van de methode plus voldoet aan hetgeen in de specificatie beloofd is, zijn er ernstige bezwaren tegen de getoonde implementatie. Noem één bezwaar en toon een betere implementatie.

OPGAVE 2

Bij de uitwerking van deze opgave moet u gebruik maken van de API-klassen Random en Arrays. De klasse Random is bekend uit de cursus; een deel van de interface vindt u op de pagina's 89 en 90 van deel 2 van het cursusmateriaal.

De klasse Arrays bevat een verzameling klassenmethoden om arrays te manipuleren. Een representatief deel van de interface van de klasse Arrays vindt u in de bijlage bij deze eindtoets. (Alle getoonde methoden zijn voor arrays van type int[]; er bestaan echter overloaded versies voor arrays van ander primitieve typen en voor arrays van objecten). U dient zelf te bepalen welke van de getoonde methoden u kunt gebruiken.

Gegeven is een klasse Scores, die wordt gebruikt om bepaalde bewerkingen uit te voeren op een lijst van scores van studenten. Een score is een geheel getal dat tenminste 0 en ten hoogste 100 is. Het skelet van een deel van de klasse ziet er als volgt uit.

```
public class Scores {
  // de array met scores
  private int[] scorelijst = null;

  /**
   * Constructor bedoeld voor testdoeleinden:
   * scorelijst krijgt als waarde een array van
   * n random bepaalde scores.
   * @param n  de afmeting van de array met scores
   */
  public Scores(int n) {
    ...
  }
  ...

  /**
   * Geeft een versie van de scorelijst waarin
   * de scores in oplopende volgorde staan
   * @return een gesorteerde array van alle scores
   */
  public int[] oplopendeScores(){
    ...
  }

  /**
   * Toont in volgorde de scores die voorkomen in
   * scorelijst.
   * @return een gesorteerde array van scores, zonder
   *         dubbelen, die alle scores bevat die voorkomen
   *         in scorelijst
  public int[] uniekeScores() {
    ...
  }
}
```

Andere constructoren en methoden zijn niet getoond.

5 punten

a Toon een implementatie van de methode oplopendeScores.

10 punten

b Toon een implementatie van de constructor .

10 punten

c De methode uniekeScores heeft als waarde een gesorteerde array met alle scores die minimaal een keer voorkomen in de scorelijst. Is de scorelijst bijvoorbeeld [17, 29, 67, 64, 29, 62, 11, 2, 67], dan is de terugkeerwaarde van uniekeScores gelijk aan [2, 11, 17, 29, 62, 64, 67]. Merk op dat 29 en 67, die in de scorelijst beide twee keer voorkomen, nu maar één keer voorkomen.
Toon een implementatie van de methode uniekeScores. Maak waar mogelijk gebruik van methoden van de klasse Arrays.

OPGAVE 3

Alle koopwoningen in een gemeente worden elke twee jaar getaxeerd. De getaxeerde waarde wordt vastgesteld op grond van de prijzen die vergelijkbare woningen in het jaar voor de taxatiedatum hebben opgebracht.

Een woning wordt gekenmerkt door een postcode en een huisnummer. Voor de waardebepaling zijn verder de volgende eigenschappen van belang: het woningtype (vrijstaand, twee-onder-een-kap, hoekhuis, tussenwoning of appartement), het woonoppervlak en de grootte van de kavel. Voor elke woning wordt bovendien bijgehouden wanneer deze van eigenaar is gewisseld en wat de verkoopprijs was.

De gemeente ontwikkelt een informatiesysteem waarmee de taxatie automatisch kan gebeuren. Deze opgave gaat over het *ontwerp* van dit informatiesysteem; hoe de taxatie precies in zijn werk gaat (het algoritme) komt niet aan de orde.

In het informatiesysteem worden onder andere de klassen Woning en Verkoop opgenomen.

De klasse Verkoop representeert één verkoop, waarbij de datum en de prijs van die verkoop (in hele euro's) worden vastgelegd (gegevens over de oude en nieuwe eigenaar blijven in deze opgave buiten beschouwing).

De klasse Woning representeert een woning met de eigenschappen als zojuist geschetst. Bij een woning worden alle geregistreerde verkopen bijgehouden, vanaf de datum dat de woning in het systeem is opgenomen. De klasse moet een nieuwe verkoop aan deze lijst toe kunnen voegen. Een belangrijke verantwoording van de klasse is verder de taxatie van de woning, waarbij een lijst wordt meegegeven van vergelijkbare woningen die in het afgelopen jaar verkocht zijn. Binnen de klasse Woning moet gecontroleerd kunnen worden of de woningen in deze lijst aan de laatstgenoemde eis voldoen.

7 punten

a Eén van de attributen van de klasse Woning is het woningtype. Wat is het meest geschikte type voor dit attribuut in de implementatie van de klasse Woning? Toon de benodigde declaratie(s).

6 punten

b Uit bovenstaande omschrijving volgt dat de klasse Woning in elk geval methoden moet hebben om een Verkoop toe te voegen en om een woning te taxeren. Toon van deze methoden de javadoc en de signatuur.

12 punten

c Ontwerp een gedetailleerd klassendiagram voor de klassen Woning en Verkoop. Neem daarin ook de get-methoden op die op grond van de beschrijving in deze opgave in elk geval nodig zijn. Licht uw ontwerp kort toe.

OPGAVE 4

Html is (zoals u vermoedelijk wel weet) een opmaaktaal die door een webbrowser wordt geïnterpreteerd. Een voorbeeld is onderstaande string:

```
<html><b>vet</b> en <u>onderstreept</u></html>
```

De codes tussen "<" en ">" heten tags en worden door de webbrowser niet weergegeven maar geïnterpreteerd. Bovenstaande voorbeeld wordt door een webbrowser weergegeven als:

vet en onderstreept

Deze opgave betreft een klasse Html die html-code splitst in tag-fragmenten en tekst-fragmenten. De te splitsen html-code wordt gerepresenteerd als string. De klasse ziet er globaal uit als volgt.

```
public class Html {
  private ArrayList<String> fragmenten =
      new ArrayList<String>();

  /**
   * Creeert een nieuwe instantie.
   * Splitst html op in tag-fragmenten en tekst-fragmenten
   * @param html de op te splitsen string
   */
  public Html(String html) {
    //... nog te implementeren, zie opgave 4c
  }

  /**
   * Voegt het fragment toe mits het fragment geen lege
   * string is.
   * @param fragment het toe te voegen fragment
   */
  private void addFragment(String fragment) {
    //... nog te implementeren, zie opgave 4a
  }
```

```
  /**
   * Voegt de tekst-fragmenten samen, in de volgorde
   * waarin ze in de lijst staan.
   * @return de samenvoeging van de tekst-fragmenten
   */
  public String geefTekst() {
        //...nog te implementeren, zie opgave 4b
  }

  // Eventuele andere methodes die voor deze opgave
  // niet relevant zijn, zijn weggelaten.
}
```

We gaan ervan uit dat de klasse alleen correcte html aangeboden krijgt.

We lichten de klasse toe aan de hand van een voorbeeld. Gegeven is de volgende code.

```
String s =
  "<html><b>vet</b> en <u>onderstreept</u></html>";
Html h = new Html(s);
System.out.println(h.geefTekst());
```

De lijst *fragmenten* van *h* bevat de elementen getoond in de volgende tabel.

0	"<html>"
1	""
2	"vet"
3	""
4	" en "
5	"<u>"
6	"onderstreept"
7	"</u>"
8	"</html>"

De afgedrukte waarde is de string “vet en onderstreept”.

Bij de uitwerking van de deelopgaven kunt u, naast de methoden die al uitgebreid beschreven staan in leereenheid 12, ook de volgende methoden van String gebruiken:

`String substring(int beginIndex)`
Levert een nieuwe String die een substring is van deze string. De substring begint met het karakter op de opgegeven index (beginIndex) en loopt tot het einde van de string.

`String substring(int beginIndex, int endIndex)`
Levert een nieuwe String die een substring is van deze string. De substring begint met het karakter op de opgegeven index (beginIndex) en loopt tot het karakter op index endIndex – 1. De lengte van de substring is dus endIndex – beginIndex en als beginIndex gelijk is aan endIndex, wordt de lege string teruggegeven

Voorbeelden:
`"unhappy".substring(2)` levert `"happy"`
`"hamburger".substring(4, 8)` levert `"urge"`

4 punten

a Geef een implementatie van de private hulpmethode addFragment.

8 punten

b Geef een implementatie van de methode geefTekst.

13 punten

c Geef een implementatie van de constructor.
Aanwijzingen
– Maak gebruik van de hulpmethode addFragment.
– Als twee tags meteen op elkaar volgen; is de tekst ertussen leeg. Bij een goed gebruik van substring en addFragment hoeft u daar niet apart op te testen.

BIJLAGE BIJ EINDTOETS

Op deze en de volgende twee pagina's is een representatief deel getoond van de interface van de klasse Arrays.

binarySearch

```
public static int binarySearch(int[] a,
                               int key)
```

Searches the specified array of ints for the specified value using the binary search algorithm. The array must be sorted (as by the `sort(int[])` method) prior to making this call. If it is not sorted, the results are undefined. If the array contains multiple elements with the specified value, there is no guarantee which one will be found.

Parameters:

`a` - the array to be searched
`key` - the value to be searched for

Returns:

index of the search key, if it is contained in the array; otherwise, `(-(`*insertion point*`) - 1)`. The *insertion point* is defined as the point at which the key would be inserted into the array: the index of the first element greater than the key, or `a.length` if all elements in the array are less than the specified key. Note that this guarantees that the return value will be >= 0 if and only if the key is found.

---

copyOf

```
public static int[] copyOf(int[] original,
                           int newLength)
```

Copies the specified array, truncating or padding with zeros (if necessary) so the copy has the specified length. For all indices that are valid in both the original array and the copy, the two arrays will contain identical values. For any indices that are valid in the copy but not the original, the copy will contain `0`. Such indices will exist if and only if the specified length is greater than that of the original array.

Parameters:

`original` - the array to be copied
`newLength` - the length of the copy to be returned

Returns:

a copy of the original array, truncated or padded with zeros to obtain the specified length

Throws:

`NegativeArraySizeException` - if `newLength` is negative
`NullPointerException` - if `original` is null

Since:

1.6

---

copyOfRange

```
public static int[] copyOfRange(int[] original,
                                int from,
                                int to)
```

Copies the specified range of the specified array into a new array. The initial index of the range (`from`) must lie between zero and `original.length`, inclusive. The value at `original[from]` is placed into the initial element of the copy (unless `from == original.length` or `from == to`). Values from subsequent elements in the original array are placed into subsequent elements in the copy. The final index of the range (`to`), which must be greater than or equal to `from`, may be greater than `original.length`, in which case `0` is placed in all elements of the copy whose index is greater than or equal to `original.length - from`. The length of the returned array will be `to - from`.

Parameters:

`original` - the array from which a range is to be copied
`from` - the initial index of the range to be copied, inclusive
`to` - the final index of the range to be copied, exclusive. (This index may lie outside the array.)

Returns:

a new array containing the specified range from the original array, truncated or padded with zeros to obtain the required length

Throws:

`ArrayIndexOutOfBoundsException` - if `from < 0` or `from > original.length()`
`IllegalArgumentException` - if `from > to`
`NullPointerException` - if `original` is null

Since:

1.6

---

fill

```
public static void fill(int[] a,
                        int val)
```

Assigns the specified int value to each element of the specified array of ints.

Parameters:

`a` - the array to be filled
`val` - the value to be stored in all elements of the array

---

sort

```
public static void sort(int[] a)
```

Sorts the specified array of ints into ascending numerical order. The sorting algorithm is a tuned quicksort, adapted from Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function", Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November 1993). This algorithm offers n*log(n) performance on many data sets that cause other quicksorts to degrade to quadratic performance.

Parameters:

`a` - the array to be sorted

---

toString

```
public static String toString(int[] a)
```

Returns a string representation of the contents of the specified array. The string representation consists of a list of the array's elements, enclosed in square brackets (`"[]"`). Adjacent elements are separated by the characters `", "` (a comma followed by a space). Elements are converted to strings as by `String.valueOf(int)`. Returns `"null"` if `a` is `null`.

Parameters:

`a` - the array whose string representation to return

Returns:

a string representation of `a`

Since:

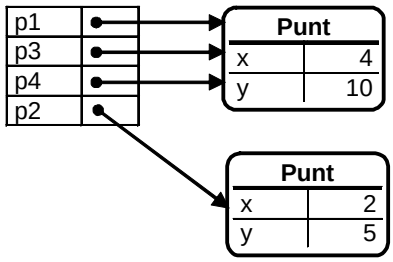
1.5

TERUGKOPPELING

Uitwerking van de opgaven

1 a De Punt-objecten p1 en p2 hebben wel dezelfde waarden van x en y, maar het zijn twee verschillende, apart gecreëerde objecten. Er geldt dus niet dat p1 == p2.

b Zie figuur 1.



FIGUUR 1 Toestandsdiagram van punten p1, p2, p3 en p4

De toekenning p3 = p1 maakt p1 en p3 tot aliassen. Aan p4 wordt vervolgens de waarde van p1.plus(p2) toegekend. De methode plus, aangeroepen op p1, wijzigt allereerst de waarden van de attributen x en y van p1: de waarden van de attributen p2.x respectievelijk p2.y worden er bij opgeteld. Vervolgens geeft de methode als waarde het object terug waar de methode op werd aangeroepen, ofwel het object waar p1 naar verwijst. Ook p1 en p4 worden dus aliassen.

c Op de volgende punten in de code vindt boxing / unboxing plaats:
– Bij toekenning aan x in de declaratie vindt autoboxing plaats van de waarde 0.
– In de constructor wordt bij de toekenningen aan het attribuut x de waarde van de parameter x met behulp van autoboxing omgezet in een waarde van type Integer.
– In de methode getX wordt de waarde van het attribuut x met behulp van auto-unboxing omgezet in een terugkeerwaarde van type int.
– In de toekenning x = x + p.getX() in de methode plus, wordt de waarde van x eerst uitgepakt (auto-unboxing). De resulterende int-waarde wordt opgeteld bij de terugkeerwaarde van p.getX() (die is al van type int). Het resultaat wordt geconverteerd naar Integer (autoboxing) en dan toegekend aan x.

d Deze implementatie heeft een neveneffect dat niet in de specificatie vermeld wordt: hij wijzigt de waarden van de attributen x en y. Bovendien is de terugkeerwaarde het object zelf, waardoor mogelijk onbedoelde aliasing optreedt (zoals bij p4; zie figuur bij onderdeel b). Het is veel beter de waarden van x en y ongemoeid te laten en een nieuw Punt-object te maken, bijvoorbeeld als volgt:

```
public Punt plus(Punt p) {
  return new Punt(x + p.getX(), y + p.getY());
}
```

2 a Een mogelijke implementatie van deze methode is de volgende:

```
public int[] oplopendeScores(){
  int[] oplopend =
    Arrays.copyOf(scorelijst, scorelijst.length);
  Arrays.sort(oplopend);
  return oplopend;
}
```

Het is zeer belangrijk dat u de scorelijst kopieert vóór u gaat sorteren. Schrijft u bijvoorbeeld het volgende op:

```
int[]oplopend = scorelijst;
Arrays.sort(oplopend);
return oplopend;
```

dan zijn oplopend en scoreLijst aliassen en bent u de oorspronkelijke scorelijst na het sorteren kwijt. Als u deze fout heeft gemaakt, krijgt u geen punten voor dit onderdeel.

b Een mogelijke implementatie van deze constructor is de volgende:

```
public Scores(int n) {
  Random random = new Random();
  scorelijst = new int[n];
  for (int i=0; i < n; i++) {
    scorelijst[i] = random.nextInt(101);
  }
}
```

Merk op dat de parameter van nextInt gelijk is aan 101; dat betekent dat er int-waarden van 0 tot en met 100 worden gegenereerd.

c U moet in elk geval uitgaan van de gesorteerde versie van de array. Het vervolg kunt u op verschillende manieren aanpakken. Een manier is, om met behulp van de methode binarySearch van elke score na te gaan of deze voorkomt; zo ja, dan wordt deze toegevoegd aan het resultaat. Een andere manier is, om uit de gesorteerde versie de dubbelen weg te laten. De eerste aanpak leidt tot de minst complexe code; de tweede is efficiënter.

```
public int[] uniekeScores() {
  int[] gesorteerdeLijst = oplopendeScores();
  int[] uniekeScores = new int[101];
  int index = 0;
  for (int score = 0; score <= 100; score++) {
    if (Arrays.binarySearch(gesorteerdeLijst,score) >= 0){
        uniekeScores[index] = score;
        index++;
    }
  }
  return Arrays.copyOfRange(uniekeScores, 0, index);
}
```

Het vooraf sorteren van de scorelijst is nodig om te kunnen zoeken met binarySearch.

Er kunnen maximaal 101 unieke scores zijn (0 tot en met 100); we creëren daarom in eerste instantie een array van die lengte. Alternatieven zijn een array ter lengte van de scorelijst of van het minimum van de twee. Vervolgens zoeken we voor elke score van 0 tot en met 100 of deze voorkomt in de gesorteerde versie van de scorelijst; zo ja, dan voegen we deze toe aan de lijst uniekeScores. De variabele index bevat de index van de eerste lege plaats in uniekeScores.
Als laatste stap maken we met behulp van copyOfRange een array die alleen de scores bevat, zonder overbodige nullen aan het eind.

Een alternatief is, om de array te sorteren en vervolgens een kopie te maken die de dubbel voorkomende scores weglaat. Het afwijkende deel is grijs gemarkeerd.

```
public int[] uniekeScores() {
  int[] gesorteerdeLijst = oplopendeScores();
  int[] uniekeScores = new int[101];
  int index = 0;
  int laatsteUniekeScore = -1;
  for (int score: gesorteerdeLijst) {
    if (score != laatsteUniekeScore) {
      uniekeScores[index] = score;
      index++;
      laatsteUniekeScore = score;
    }
  }
  return Arrays.copyOfRange(uniekeScores, 0, indexUniek);
}
```

3 a Omdat het woningtype een waarde heeft die afkomstig is uit een vaste waardeverzameling, is een enumeratietype het meest geschikt. De benodogde declaraties zijn bijvoorbeeld

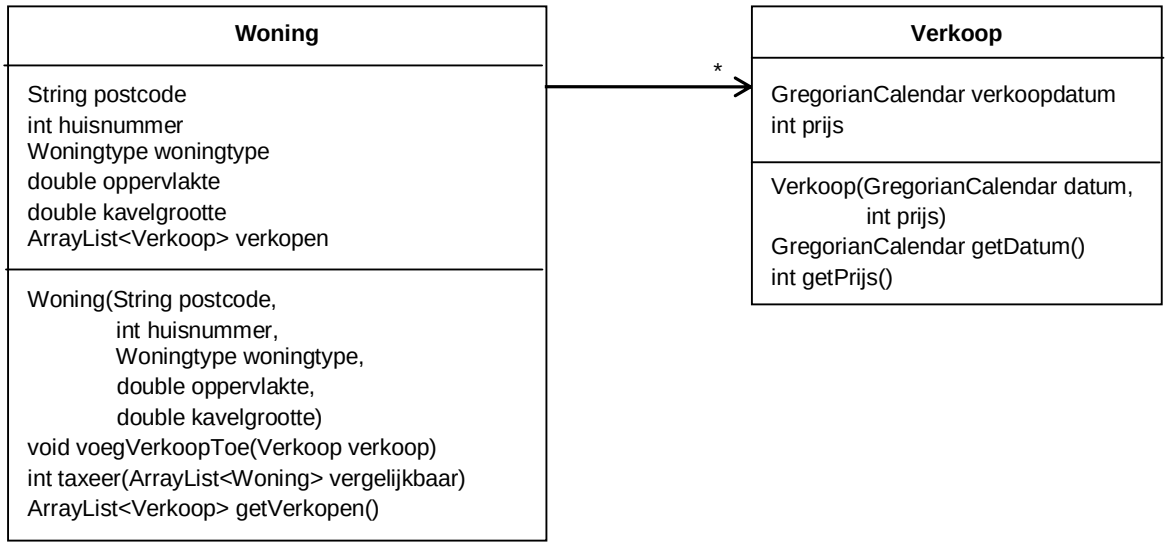
```
public enum Woningtype {VRIJSTAAND, TWEEONDEREENKAP,
                        HOEKHUIS, TUSSENWONING,
                        APPARTEMENT}

private Woningtype woningtype = null;
```

b Een mogelijke uitwerking is de volgende.

```
/**
 * Voegt een nieuwe verkoop toe aan de lijst van verkopen.
 * @param verkoop  de toe te voegen Verkoop-instantie
 */
public void voegVerkoopToe(Verkoop verkoop)

/**
 * Schat de waarde van deze woning op grond van de
 * eigenschappen (ligging, type, oppervlakte,
 * kavelgrootte) en op grond van de verkoopprijzen van
 * een lijst vergelijkbare woningen.
 * @param vergelijkbaar  lijst van vergelijkbare woningen
 *        die in het afgelopen jaar verkocht zijn
 * @return  de geschatte waarde van deze woning
 */
public int taxeer(ArrayList<Woning> vergelijkbaar)
```



FIGUUR 2 Klassendiagram van de klassen Woning en Verkoop

c Een mogelijke uitwerking is getoond in figuur 2.
Als type voor het attribuut postcode kozen wij String. Omdat een postcode uit precies zes karakters bestaat, is het type char[] ook een mogelijk keuze (die echter iets minder voor de hand ligt). Voor het huisnummer gebruikten wij int. De specificatie maakt geen melding van toevoegingen bij huisnummers (34 hs, t/o 78, 121e), maar als u daar zelf aan gedacht heeft, is String ook mogelijk.

De klasse Woning heeft een constructor die alle attributen behalve de ArrayList een waarde geeft. Die laatste blijft null tot de woning voor het eerst verkocht wordt (alleen de verkopen na ingebruikname van het systeem worden opgenomen). De klasse heeft drie methoden:
– een methode om een verkoop toe te voegen (geëist in de omschrijving)
– een methode om de woning te taxeren, met als parameter een lijst (recent verkochte) woningen
– een get-methode voor de lijst verkopen.

In feite heeft de klasse op grond van alleen de opdrachtomschrijving, helemaal geen get-methoden nodig. De methode taxeer heeft toegang nodig tot de lijst van Verkopen van de vergelijkbare woningen, maar omdat dit een attribuut is van een *andere* instantie van *dezelfde* klasse, is die toegang er toch al (in de methode taxeer mag code voorkomen als)

```
for (Woning w: vergelijkbaar) {
  for (Verkoop v: w.verkopen) {
    …
  }
}
```

Heeft u de get-methode weggelaten, dan dient u wel vermeld te hebben dat dit de reden is (in de cursus wordt dit wel vermeld, maar het komt nooit voor).

De klasse Verkoop heeft als attributen de verkoopdatum en de prijs. Omdat de prijs in hele euro's wordt bijgehouden, is het type int en niet double. (De maximale waarde van een variabele van type int is ruim twee miljard; voor de verkoop van een woonhuis is dat meer dan voldoende. Maar het is niet fout als u het zekere voor het onzekere heeft genomen en voor type long heeft gekozen).

De klasse heeft get-methoden voor beide attributen. De eerste is nodig om binnen de methode taxeer te kunnen controleren of de laatste verkoop recent genoeg is; de tweede is nodig voor de bepaling van de taxatiewaarde.

4 a, b,c Een mogelijke uitwerking is

```
private void addFragment(String fragment) {
  if (fragment.length() > 0) {
    fragmenten.add(fragment);
  }
}

public String geefTekst() {
  String tekst = "";
  for (String fragment: fragmenten) {
    if (fragment.charAt(0) != '<')
      tekst = tekst + fragment;
  }
  return tekst;
}

public Html(String html) {
  int beginTag = html.indexOf('<');
  while (beginTag >= 0) {
    addFragment(html.substring(0, beginTag));
    html = html.substring(beginTag);
    int eindTag = html.indexOf('>');
    addFragment(html.substring(0, eindTag + 1));
    html = html.substring(eindTag + 1);
    beginTag = html.indexOf('<');
  }
  addFragment(html);
}
```