Programmeren 1 21 januari 2011 Prof. T. Schrijvers

Instructies

Schrijf al je antwoorden op deze vragenbladen (op de plaats die daarvoor is voorzien). Geef ook je kladbladen af. Bij heel wat vragen moet je zelf Java-code schrijven. Hou dit *kort en bondig*. Je hoeft geen commentaarregels te schrijven en ook eventuele `import`-opdrachten zijn niet nodig.
Veel succes!

Vraag 1: Schaken (7pt)

Een schaakbord wordt voorgesteld door een 2-dimensionale array van `Schaakstuk`ken. Je mag ervan uitgaan dat een bord vierkant is, maar niet noodzakelijk met afmetingen 8×8.

De definitie van de klasse `Schaakstuk` is hieronder gegeven.

```
public abstract class Schaakstuk {
  private Schaakstuk[][] bord;
  private int x;
  private int y;
  private boolean isWit;

  public Schaakstuk(Schaakstuk[][] bord, int x, int y, boolean isWit) {
    this.bord = bord;
    this.x = x;
    this.y = y;
    this.isWit = isWit;
  }

  public int getX() { return x; }
  public int getY() { return y; }
  public Schaakstuk[][] getBord() { return bord; }

  public boolean isWit() { return isWit; }
  public boolean isZwart() { return !isWit; }

  public abstract List<Schaakstuk> bedreigt();

}
```

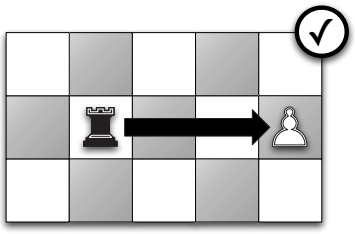
Een schaakstuk houdt het schaakbord bij waarop het staat en ook zijn positie op dit bord, zodat voor elk schaakstuk `stuk` geldt:

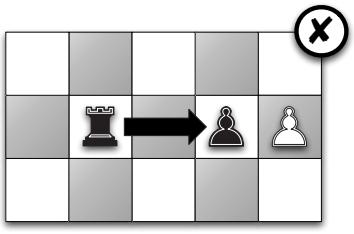
```
stuk.getBord()[stuk.getX()][stuk.getY()] == stuk
```

Verder heeft een schaakstuk ook een kleur (zwart of wit).

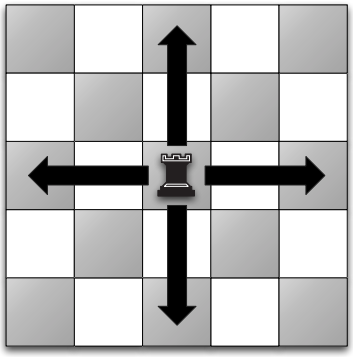
De methode `bedreigt` geeft een lijst terug van alle andere schaakstukkken die *bedreigd* worden door een stuk. De algemene regel is dat een stuk alleen stukken van een andere kleur kan bedreigen.

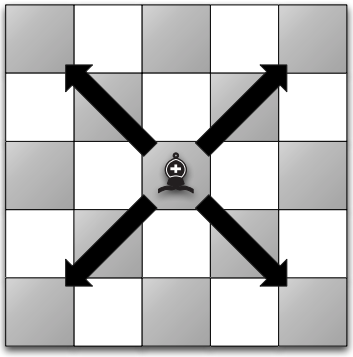
Elk specifiek type schaakstuk heeft zijn eigen bijkomende regels voor bedreiging. `Toren`s en `Loper`s zijn schaakstukken die elk stuk bedreigen dat op een rechte lijn van hen afligt; er mag geen ander stuk in de weg staan. Hier zie je hoe een zwarte toren (links) een witte pion bedreigt en (rechts) geen van beide pionnen bedreigt.





Het verschil tussen torens en lopers is dat we bij torens enkel kijken naar lijnen loodrecht op het schaakbord (links) en bij lopers enkel naar diagonale lijnen (rechts).





Opgave

- Implementeer de klassen `Toren` en `Loper`.
- Maak een gemeenschappelijke superklasse `LijnAanvaller` die de methode `bedreigt` in 1 keer voor beide implementeert. Deze methode doet beroep op de hulpmethode:

  ```
  protected abstract Stapper[] getStappers();
  ```

  met gepaste implementaties in de subklassen, en de methode `zoekSchaakstuk` van de klasse `Stapper`.
- Implementeer de klasse `Stapper` die de methode

  ```
  public Schaakstuk zoekSchaakstuk();
  ```

  aanbiedt. Deze methode vertrekt van een positie op het bord en stapt van daar in een rechte lijn een richting uit tot het een ander schaakstuk tegenkomt of buiten het bord terechtkomt. In het eerste geval, geeft het het gevonden stuk terug. In het tweede geval geeft het geen stuk terug.

  Voorzie ook een gepaste constructor voor `Stapper` die het vertrekpunt op het bord bepaalt en de richting waarin gestapt wordt.

...

Vraag 2: Methode-oproepen **(1,5pt)**

```
public abstract class A {
  public int m() {
    return 2;
  }
  public abstract int n();
}
```

```
public class B extends A {
  public int m() {
    return super.m() * n();
  }
  public int n() {
    return 3;
  }
}
```

```
public class C extends B {
  public int n() {
    return 5;
  }
}
```

Gegeven bovenstaande klassedefinities, telkens met een lege constructor die niet is weergegeven, geef aan wat het resultaat is van onderstaande expressies:

(a)	`new A().m()`	
(b)	`new B().m()`	
(c)	`new C().m()`	

Vraag 3: Waarde **(1pt)**

```
int x = 1;
try {
  x = x + 1;
  throw new ???("something is wrong");
} catch (NullPointerException e) {
  x++;
} catch (Exception e) {
  x += 2;
}
```

Geef aan welke waarde zich in de variable `x` bevindt na het uitvoeren van deze code, als we `???` vervangen door een van onderstaande woorden:

(a)	`NullPointerException`	
(b)	`IOException`	

Vraag 4: Matrix compacteren **(3pt)**

Schrijf een methode met signatuur

```
public void compacteer(Student[][] inschrijvingen)
```

De twee-dimensionale array `inschrijvingen` houdt bij welke studenten ingeschreven zijn voor elk vak. De studenten die ingeschreven zijn voor het vak met nummer `i` staan in `inschrijvingen[i]`. Sommige plaatsen in de array zijn *leeg* omdat bepaalde studenten, die oorspronkelijk ingeschreven waren voor een vak, zich intussen weer hebben uitgeschreven.

Het opzet van de methode `compacteer` is om de lege plaatsen in de array te *verwijderen* zodat ze geen ruimte meer innemen.

Vraag 5: Objecten tellen **(1,5pt)**

Geef telkens aan hoeveel objecten er aangemaakt worden in onderstaande expressies:

(a)	`new int[0]`	
(b)	`new int[2][2]`	
(c)	`new int[2][]`	

Vraag 6: Arrays naar lijsten **(3pt)**

Herschrijf onderstaande methodes zodat ze een lijst in plaats van een array als parameter nemen. Wijzig zo weinig mogelijk aan de structuur van elke methode.

(a)
```
public void m1(String[] l) {
  for (String s : l) {
    System.out.println(s);
  }
}
```

(b)
```
public void m2(int[] l) {
  for (int i = 1; i < l.length; i+=2) {
    l[i] = (l[i] + l[i-1])/2;
  }
}
```

(c)
```
public void m3(int[] l) {
  int i = 0;
  while (i >= 0) {
    i = l[l[i]];
  }
}
```

Vraag 7: Vertaler/tolk **(3pt)**

Schrijf een methode met signatuur

```
public String vertaal(String tekst, String[] van, String[] naar)
```

De string `tekst` bestaat uit een zin van woorden gescheiden door spaties, maar zonder leestekens. Het is de bedoeling dat je deze zin omzet naar een nieuwe zin door elk woord dat voorkomt in `van` om te zetten naar het woord op de overeenkomstige positie in `naar`.

Bijvoorbeeld deze oproep:

```
vertaal( "Juno is de vrouw van Jupiter"
       , new String[]{"Juno","Jupiter","vrouw"}
       , new String[]{"Hera","Zeus"   ,"gemalin"})
```

resulteert in `"Hera is de gemalin van Zeus"`.

Maak gebruik van een map om niet telkens de `van` en `naar` arrays te moeten overlopen.

Einde