

# Informatica: Java reeks 3: debuggen & grafische interfaces

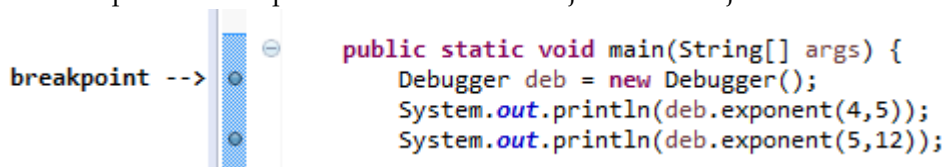
Ruben De Smet

2<sup>de</sup> semester 2021


**Oefening 12.** Download *Debugger.java* (pas de naam niet aan) en importeer in Eclipse. Deze klasse bevat enkele fouten. Spoor hen op met de debugger van Eclipse.


Om in Eclipse een Java-bestand te importeren, klik rechts op je src directory, en kies **Import**. Kies vervolgens **General** > **File System** en druk op **Next**. Kies de map met het bestand d.m.v. de **Browse**-knop. Zet een vinkje voor het bestand en klik op **Finish**.

Om in Eclipse een breakpoint te zetten *dubbelklik* je in de kantlijn.



Klik vervolgens op het Debug icoontje (🐛). Je kan door de code lopen met volgende commando's:

**Step In** (F5, ): voer één lijn code uit. Indien een method call, ga in de methode.

**Step Over** (F6, ): voer één lijn code uit. Indien een method call, voer heel de methode uit. Gebruik dit als je zeker bent dat de lijn geen bugs bevat.

**Step to line** (Ctrl+R,  in Run menu) Run de debugger tot aan je cursorpositie.

**Oefening 13.** Maak een programma dat schuine lijnen plaatst op de onderste helft van het venster. De afstand tussen de lijnen is variabel en door de gebruiker in te stellen via een tekstvak aan de bovenkant van het venster. De eerste lijn loopt van het midden aan de linkerkant naar onderaan rechts, elke volgende lijn is aan de linkerkant de ingestelde afstand lager en aan de rechterkant de ingestelde afstand hoger.

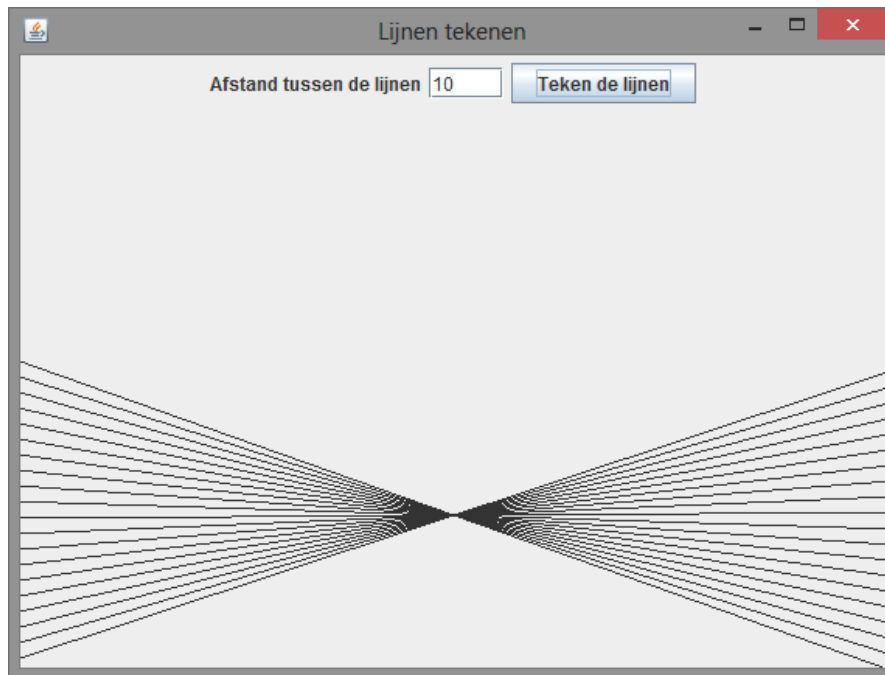
(a) Download *LinePanel.java* en importeer in Eclipse.

(b) Verander *LinePanel.java* als nodig, zie fig. 1 als voorbeeld.

**Oefening 14.** We gaan een animatie maken met verschillende botsende ballen. Hiervoor gaan we enkele klassen maken, die we samen in de **package bouncing** opslaan.

(a) Maak eerst de **class Ball**. Een bal heeft:

- een kleur (van type `Color`),



Figuur 1 – Voorbeeld LinePanel.java

- een  $x$  en  $y$  coördinaat,
  - een straal  $r$  en
  - snelheidscomponenten van de snelheidsvector:  $v_x$  en  $v_y$ .
- (b) Initialiseer de waarden van `class Ball` met random waarden (in de constructor):
- Geef de bal een willekeurige  $x$  en  $y$  positie in het veld. Zorg dat de bal niet buiten de randen van het veld terechtkomt!
  - Kies voor  $r$  een willekeurige waarde tussen 5 en 30.
  - Geef de snelheidsvector een willekeurige lengte tussen 0 en 5, en een willekeurige hoek tussen 0 en  $2\pi$ .
  - Geef de RGB kleurcomponenten (van `Color`) elk een random waarde tussen 0 en 255.
- (c) Maak een nieuwe `class BallPanel`, als een extensie van `JPanel`. Kies een schermgrootte van  $750 \times 750$  pixels. Voorzie een lijst van `Balls` met 100 elementen, en teken deze met de juiste dimensies, kleur en positie met het meegegeven `Graphics`-object `g`.
- (d) Maak vervolgens de methode `public void draw(Graphics g)` in je `Ball`. Verplaats de code van opgave 14.(c) hierheen; nu heb je de logica van de bal opgeslagen waar die thuishoort: bij de `Ball` zelf.
- (e) Om een animatie te maken, voeg je een methode `void update()` toe aan je `Ball`, die periodiek (periode  $\Delta t$ ) zal worden opgeroepen. De methode past de `Ball` aan, volgens  $\Delta \vec{r} = \vec{v} \Delta t$ .

Zorg dat de periode geen magische constante is: gebruik bijvoorbeeld **private static final int** PERIODE = 20;.

- (f) Maak een *interne* klasse UpdateTimerTask toe aan BallPanel, als extensie van TimerTask. Overschrijf de **void run()** methode, en roep daarin update() op op alle Ball objecten.
- (g) Maak een Timer in de constructor van BallPanel, waarmee je de TimerTask elke 20 ms oproept.
- (h) Als de bal buiten de rand van het veld zou gaan, simuleer je een botsing: verander het teken van  $v_x, v_y$  wanneer de bal respectievelijk de verticale dan wel horizontale wand raakt.

**Oefening 15.** We gaan nu een beetje interactiviteit toevoegen. Wanneer je op + drukt op je toetsenbord, komt er een bal bij. Wanneer je op - drukt op je toetsenbord, verdwijnt er een bal. **Hint:** Op <http://parallel.vub.ac.be/education/java/documentatie/index.html> staat er redelijk wat uitleg over toetsenbord lezen.

- (a) Implementeer de KeyListener interface op je BallPanel. Voeg **this** toe als KeyListener m.b.v. de addKeyListener methode van het JPanel. **Hint:** Je kan ook Eclipse de nodige methodes laten genereren, wanneer je **implements** KeyListener hebt toegevoegd.

```
1 public class BallPanel extends JPanel implements KeyListener {
2     BallPanel() {
3         this.addKeyListener(this);
4     }
5     @Override
6     public void addNotify() {
7         super.addNotify();
8         this.requestFocusInWindow();
9     }
10    @Override
11    void keyPressed(KeyEvent e) { ... }
12 }
```

- (b) Voeg een random Ball toe wanneer op + gedrukt wordt. Je kan de + KeyCode bijvoorbeeld vinden door de code (e.getKeyCode()) te printen en op + te drukken.
- (c) Verwijder een random Ball wanneer op - gedrukt wordt.

**Extra oefening 16.** Als je met de muis klikt, verdwijnt er één bal onder de cursor.

- (a) Implementeer de MouseListener interface op je BallPanel, analoog aan opgave 15.(a).
- (b) Wanneer er geklikt wordt, zoek je in de lijst van ballen naar een bal onder de muis pointer:

$$r^2 \geq \Delta x^2 + \Delta y^2$$

en verwijder je er één. Zorg dat er maar één bal verdwijnt, ook al zijn er meerdere onder de aanwijzer.

**Extra oefening 17.** Zorg dat je met de muisaanwijzer op een lege plaats kan klikken om een nieuwe bal toe te voegen. Hoe langer je de pointer ingedrukt houdt, hoe groter de bal wordt; zorg ervoor dat je die animatie kan waarnemen.

- (a) Maak een extra `Ball` veld in de `BallPanel` klasse, waarin je de Ball-in-de-maak kan opslaan.
- (b) Wanneer er geklikt wordt en er geen `Ball` verwijderd werd, steek je in dat veld een nieuwe `Ball`.
- (c) Teken de `Ball` in de `draw` methode.
- (d) Wanneer de muisknop losgelaten wordt, verplaats je een niet-`null` `Ball` van dat veld naar de lijst: voeg het toe aan de lijst, en zet het veld op `null`.
- (e) Wijzig de `TimerTask` zodanig dat de bal 1 px per tick groeit.