

# Informatica: Java reeks 3

debuggen & grafische interfaces

Ruben De Smet

2<sup>de</sup> semester 2021

**Syntax errors** fouten tegen de taalregels van Java; code **compileert niet**, Eclipse geeft locatie van fout.

**Runtime errors** code is **syntactisch** correct, maar **niet functioneel** correct.

**Syntax errors** fouten tegen de taalregels van Java; code **compileert niet**, Eclipse geeft locatie van fout.

**Runtime errors** code is **syntactisch** correct, maar **niet functioneel** correct.

Voorbeelden van runtime errors:


**Onmogelijke operatie** deling door nul, toegang tot onbestaande geheugenlocatie, ...;

**Oneindige lus** `while` lus die nooit eindigt;

**Foute resultaten** foute implementatie van een formule.


Oplossing: een tool om de bugs te detecteren: de **debugger**.

# DEBUGGEN

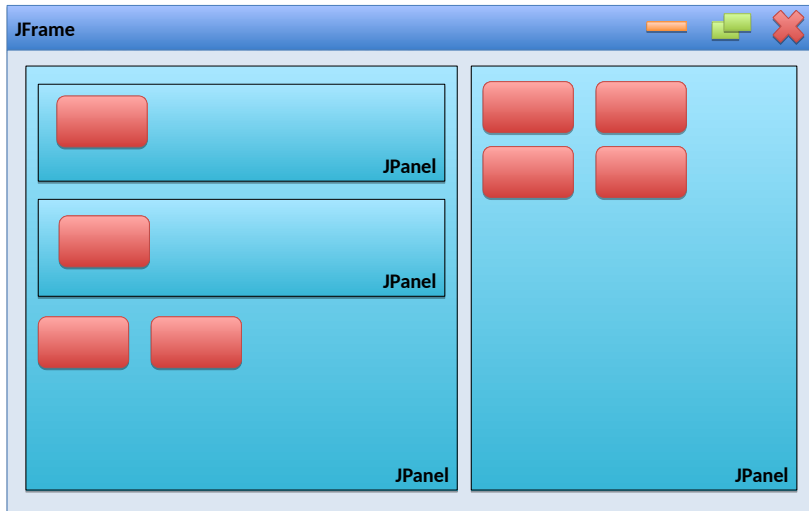
Eclipse heeft complexe ingebouwde **debugger** (gebruik debug-perspective 

**Step In** (F5, 

**Step Over** (F6, 

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

# GRAFISCHE INTERFACES



## EERSTE GRAFISCHE INTERFACE

Download `LinePanel.java`.

```
8 public class LinePanel extends JPanel implements
   ↳ ActionListener
9 {
10     private JTextField afstandVeld;
11     private JButton tekenKnop;
12
14     public LinePanel()
15     {
16         afstandVeld = new JTextField(4);
21         this.add(afstandVeld);
24     }
63 }
```

## JFRAME STARTEN

Starten van JFrame in de `public void main(...)`:

```
52 public static void main(String[] args)
53 {
54     JFrame f = new JFrame();
55     f.setSize(500,500);
56     f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
57     f.setTitle("Lijnen tekenen");
58     f.setLocation(100, 100); //standaard in de hoek
        ↪ van het scherm
59     JPanel hoofdpaneel = new JPanel();
60     f.add(hoofdpaneel);
61     f.setVisible(true);
62 }
```

## EVENTS

```
27 public void actionPerformed(ActionEvent e) {  
46 public void paintComponent(Graphics g) {
```

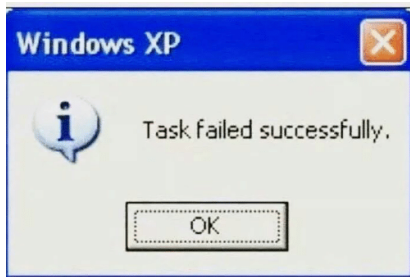


## FOUTEN AFHANDELEN

Fouten komen voor in **alle** programma's:

- ▶ Foute gebruikersinput;
- ▶ Bestand verdwenen;
- ▶ Geheugen is op (out-of-memory).

Elk ietwat groter programma moet fouten afhandelen:



Me\_irl

# FOUTEN AFHANDELEN

## TRY-CATCH

Vorige les: `throw new IllegalArgumentException();`

Vandaag:

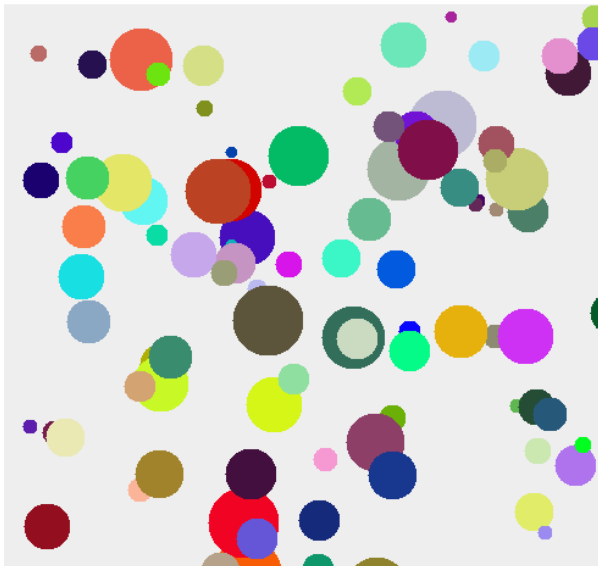
```
1 try {  
2     fraction.setDenominator(0);  
3 } catch (IllegalArgumentException ex) {  
4     // Wat doe ik nu? Verwittig de gebruiker!  
5 } finally {  
6     // Dit wordt *altijd* uitgevoerd, zelf na return!  
7 }
```

Gebruik `Graphics.drawLine(...)` methode in je `void paintComponent()`. Andere nuttige methodes:

- ▶ `Graphics.setColor(...)` om een nieuwe kleur te kiezen.
- ▶ `Graphics.fillOval(...)` om een gevulde ovaal te tekenen.

## BALLPANEL

In oefening 14 maken we volgend JPanel.



## BALLPANEL

### DRAW METHODE

Bij het tekenen van de ballen gebruik je voor elke bal `Graphics.setColor(Color c)` en `Graphics.fillOval(int x, int y, int w, int h)`.

Roep deze op in een nieuwe methode `void draw(Graphics g)` in de klasse `class Ball`. Nu kunnen alle velden van `class Ball` op `private` staan.

De update methode zorgt dat de huidige ball voor 20 ms verplaatst.

```
4 public class Ball {  
...  
24     public void update() {  
25         this.x += vx * 20;  
26         this.y += vy * 20;  
27     }  
28 }
```

Deze manier van `update()` en `draw(Graphics g)` kan je gebruiken voor het project!

# BEWEGING

## UPDATETIMERTASK

```
10 public class BallPanel extends JPanel {
11     ArrayList<Ball> balls;
12
13     class UpdateTimerTask extends TimerTask {
14         @Override
15         public void run() {
16             for (Ball ball: balls) {
17                 ball.update();
18             }
19             repaint();
20         }
21     }
54 }
```

# BEWEGING

## UPDATETIMERTASK (CONT.)

Je kan nu de `UpdateTimerTask` aanmaken, en toevoegen aan de `Timer`.

```
10 public class BallPanel extends JPanel {  
...  
23     public BallPanel() {  
...  
29         Timer t = new Timer();  
30         t.scheduleAtFixedRate(new UpdateTimerTask(),  
           ↪ 0, 20);  
31     }  
54 }
```

De `Timer` is verantwoordelijk voor het oproepen van de taak.



# INTERACTIVITEIT

## KEYLISTENER

```
1  public class BallPanel extends JPanel implements  
   ↪  KeyListener {  
2      BallPanel() {  
3          ...  
4          this.addKeyListener(this);  
5      }  
6      @Override  
7      public void addNotify() {  
8          super.addNotify();  
9          this.requestFocusInWindow();  
10     }  
11     @Override  
12     void keyPressed(KeyEvent e) { ... }  
13 }
```

# INTERACTIVITEIT

## KEYLISTENER: KEYEVENT

```
1 public class BallPanel extends JPanel implements  
   ↳ KeyListener {  
2     void keyPressed(KeyEvent e) {  
3         if(e.getKeyCode() == KeyEvent.VK_PLUS) {  
4             balls.add(new Ball());  
5         }  
6     }  
7 }
```

Bekijk zeker de documentatie over de KeyEvent:

<https://docs.oracle.com/javase/8/docs/api/java/awt/event/KeyEvent.html>

Implementeer op dezelfde manier `MouseListener`. Meerdere **interfaces** scheid je met een komma:

```
public class BallPanel extends JPanel implements  
↳ KeyListener, MouseListener { ... }
```

Bij het toevoegen een **interface** stelt Eclipse voor om de vereiste methodes toe te voegen.