Debugger

Problemstellung

Für Fehlersuche notwendig: Nachvollziehen, was Code genau macht; Fehler können beim Codelesen übersehen werden und es dauert sehr lange

Lang bekannte Möglichkeit: Mit print-Ausgaben im Programmlauf Positionen und Variablenwerte ausgeben

Nachteile:

- Programm wird dadurch unübersichtlicher
- Ausgabe unübersichtlich, wenn verschiedene Stellen gleichzeitig etwas ausgeben
- Änderungen im Programm, obwohl sich an der Ausführlogik nichts geändert hat
- Für zusätzliche Informationen muss jedesmal das Programm komplett neu gestartet werden

Debugger

Debugger ermöglichen effiziente Programmuntersuchung ohne Codeänderung:

- Variablen können beobachtet werden
- Es können Unterbrechungspunkte (Breakpoints) zum Beobachten des Programms gesetzt werden
- Programm kann pausiert und wieder fortgesetzt werden
- Werte im Programm können für Tests geändert werden
- Exceptions (auch gefangene) können beobachtet werden

Debugger für verschiedene Programmiersprachen arbeiten ähnlich, wir behandeln hier den Debugger von IntelliJ für Java

Breakpoints setzen

Breakpoint kann in IntelliJ durch Mausklick (neben Zeilennummer) gesetzt werden

```
48         });
49
50     EventHandler<Action
51         GraphicsContext
52         gc.clearRect(0,
53         gc.setFill(Colo
54         for(ColCircle c
55             gc.fillOval
56             cc.size *=
57         }
58     };
```

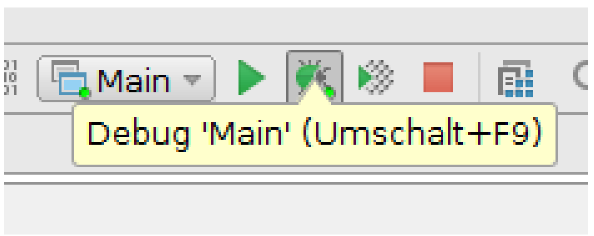
Breakpoints setzen

Breakpoint kann in IntelliJ durch Mausklick (neben Zeilennummer) gesetzt werden

```
49
50      EventHandler<Action
51          GraphicsContext
52          gc.clearRect(0,
53          gc.setFill(Colo
54          for(ColCircle c
55              gc.fillOval
56              cc.size *=
57          }
58      };
```

Debugger starten

Programm im Debugger starten funktioniert ähnlich, wie normal Programm starten



Auswirkungen:

- Programmlauf hält an Breakpoints an
- Es steht das Debugmenü zur Verfügung (unter dem Menüpunkt Run): Programm anhalten, schrittweise fortsetzen, ...

Debugmenü

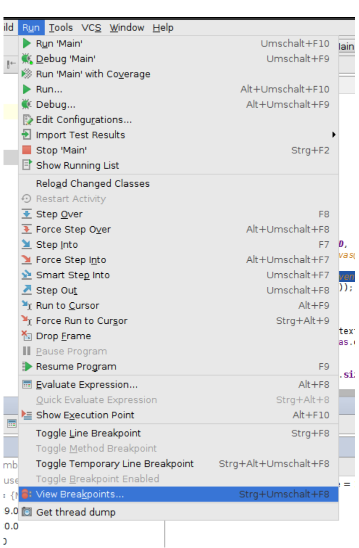
Aktionen im Debugmenü beinhalten unter anderem

- Step Over: Nächstes Statement in gleicher Datei, Methodenaufrufe in einem Schritt ausführen
- Step Into: Bei Methodenaufrufen in den Rumpf der aufgerufenen Methode gehen
- Step Out: so lange weitergehen, bis die Abarbeitung des aktuellen Methodenrumpfes beendet ist.
- Resume Program: Programm fortsetzen, bis zum nächsten Breakpoint, der ausgelöst wird

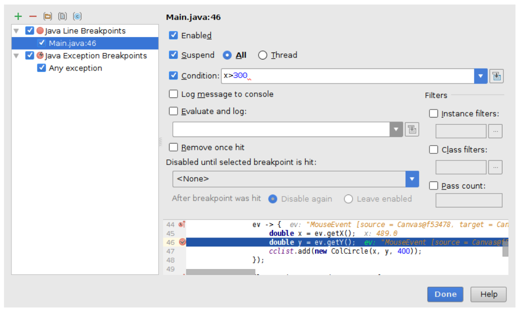
Breakpointliste

Im Debugmenü kann eine Liste der Breakpoints aufgerufen werden.

Beinhaltet gesetzte Breakpoints, sowie Breakpoints für Exceptions (selbst, wenn sie gefangen werden)



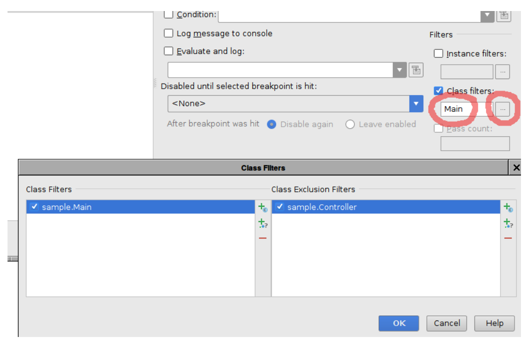
Breakpoints konfigurieren



In der Breakpointliste können Bedingungen gesetzt werden, wann der Breakpoint aktiviert wird (Bsp. $x > 300$: nur, wenn $x > 300$ und die Ausführung an dieser Stelle ist)

Breakpointliste – Class filters

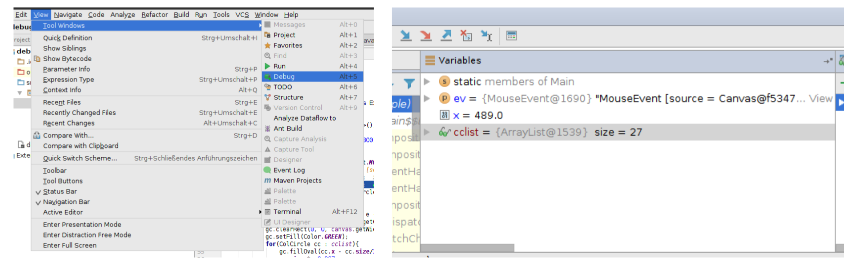
Class filters sorgen dafür, dass der Breakpoint nur in der angegebenen Klasse verwendet wird (bei Exception Breakpoints wichtig, da JavaFX beim Start viele interne Exceptions verwendet)



Eingabe einer Klasse oder erweiterter Class Filters-Editor

Variablen beobachten und setzen

Wenn das Programm pausiert, etwa weil ein Breakpoint erreicht ist: Im Debugfenster können Variablen beobachtet und gesetzt werden



Zusammenfassung

- Kennt man den Wert von Variablen versteht man genauer, warum ein Programm ein gewisses Verhalten zeigt
- Debugger zu verwenden hat viele Vorteile gegenüber einer reinen Textausgabe
  - Es muss kein Quellcode geändert werden (starker Vorteil bei Zusammenarbeit in Gruppe)
  - Programm kann übersichtlicher nachvollzogen werden
  - Man kann im Programmlauf entscheiden, welche Werte interessant sind
  - Man kann Werte auch testweise neu setzen