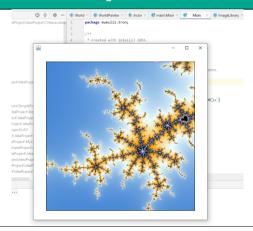
FOP Recap #14



Wiederholung, Fragen und Klaurvorbereitung



Fragerunde 1!

Das steht heute auf dem Plan



Wiederholung Programmablauf Objektorientierte Konzepte

Programmablauf - Befehle



Befehl/Anweisung/Statement/...:

- Anweisung im imperativen Programmieren
- u,tu das, dann das, als letztes das"
- Hat keinen zugehörigen Wert/Wert wird "verworfen"
- Beispiele:

```
- int a = 7;
- foo();
- for (String s : list) System.out.println(s);
```

Programmablauf - Ausdrücke



Ausdruck/Expression/...:

- Wird (eventuell erst zur Laufzeit) ausgewertet und zu Wert evaluiert
- Hat immer einen Wert und einen Typen
- Oft zusammengesetzt aus kleineren Ausdrücken, die kombiniert werden
- Lässt sich über induktiv definierte Menge beschreiben
- Beispiele:
 - "Hi"
 - -6+4
 - Math.max(a, b * 6)

Programmablauf - Java/Objektorientiert



- Programmablauf beginnt mit Aufruf der main-Methode
- Diese enthält nun Befehle, die nacheinander ausgeführt werden, bis das Ende der main-Methode erreicht ist:
 - Objekte erstellen
 - Methoden aufrufen
 - ...
- Programmablauf besteht aus Interaktionen von Objekten
- Objekte kapseln Zustand ein, der während dem Programmablauf potentiell verändert wird

Programmablauf - Racket/Funktional



- In der FOP leider nur oberflächlich behandelt
- Es gibt keinen Zustand
- Programmablauf ist das (potentiell rekursive) evaluieren von Ausdrücken
- Deklarative Programmierung: Beschreibe die Logik, nicht den Kontrollfluss

Objektorientierte Konzepte - Statischer vs. Dynamischer Typ



- Statisch: Alles, was zur Kompilierzeit überpüft werden kann/bekannt ist
- Dynamisch: Alles, was zur Laufzeit überpüft werden kann/bekannt ist
- Dynamischer Typ einer Referenz muss immer gleich dem statischen Typen oder ein Subtyp davon sein -> sonst kompiliert das Programm nicht!
- Generics sind nur statische Typüberprüfung
- Gutes Beispiel: Interfaces, insbesondere List:



Liskov Substitution Principle (LSP):

Sei $\varphi(x)$ eine beweisbare Eigenschaft über Objekte x des Typen T.

Dann soll auch $\varphi(y)$ für alle Objekte y eines Typens S gelten, wenn S ein Subtyp von T ist.

Auf Deutsch:

- Man kann überall, wo man ein Objekt des Typens T erwartet, auch ein Objekt des Typens S nutzen, wenn S ein Subtyp von T ist.
- Somit muss S mindestens so viel Funktionalität wie T bereitstellen und kann die Funktionalität von T nur erweitern, wenn das LSP nicht verletzt werden soll.

Fragerunde 2!

Klausuraufgaben!