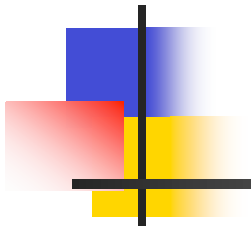
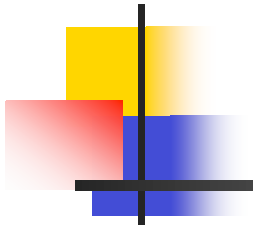


Software Engineering

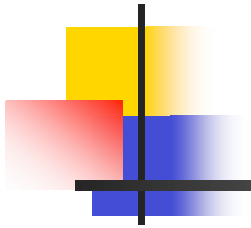
Besprechung zur Uebung 1



Allgemeines

- **Benennungen** der ZIP-Dateien der Abgaben, bitte einheitlich nach folgendem Muster:
  - Mustermann_Hinterseer_Auerberg_Uebung2.zip
  - Abgabe: am Besten ***ein*** Dokument und als PDF.
- Zu jeder Übung bitte ein **Deckblatt** mit folgenden Daten:
  - Namen und Matrikel-Nr. aller Mitglieder
  - Inhaltsverzeichnis der Uebung
- Gruppen
  - Noch nicht alle haben eine Gruppe, bevorzugt für weitere Übungen idealerweise 3er Gruppen.

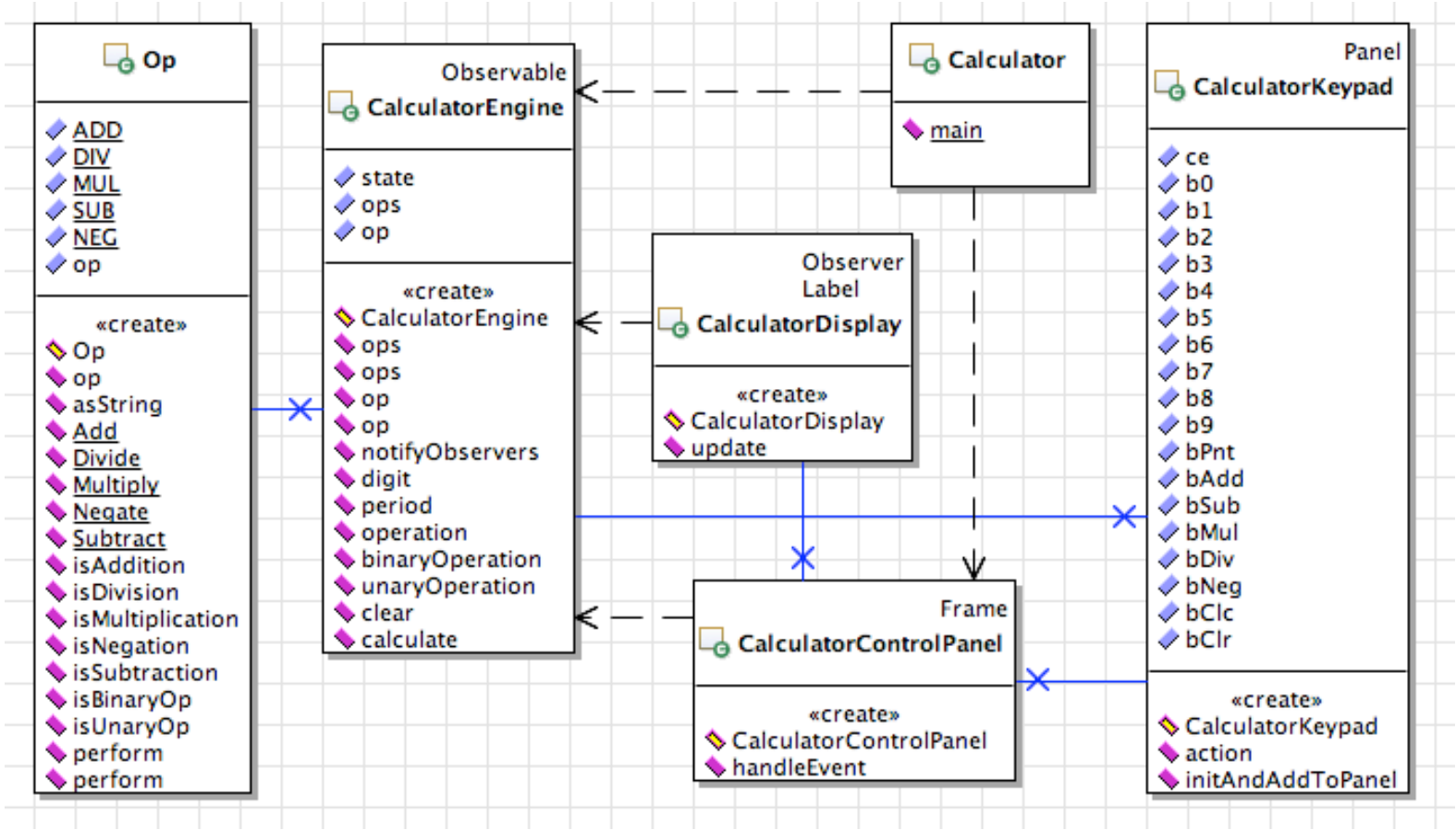
2.1. Verstehen des Quellcodes (1)



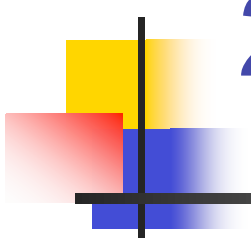
- UML Klassendiagramm
  - Klassen wurden gut modelliert.
  - Beziehungen waren eher ein Problem
    - Abhängigkeiten
      - Änderungen an der Definition können Änderungen in der abhängigen Klasse nach sich ziehen.
      - --> gerichtete Beziehungen!

2.1. Verstehen des Quellcodes (2)

UML Klassen-diagramm des Calculator Codes.



Anmerkung: Bei den Attributen sollten auch die Typen angezeigt werden (bsp. ops und op sonst schwer verständlich)



2.2. Reverse Engineering (1)

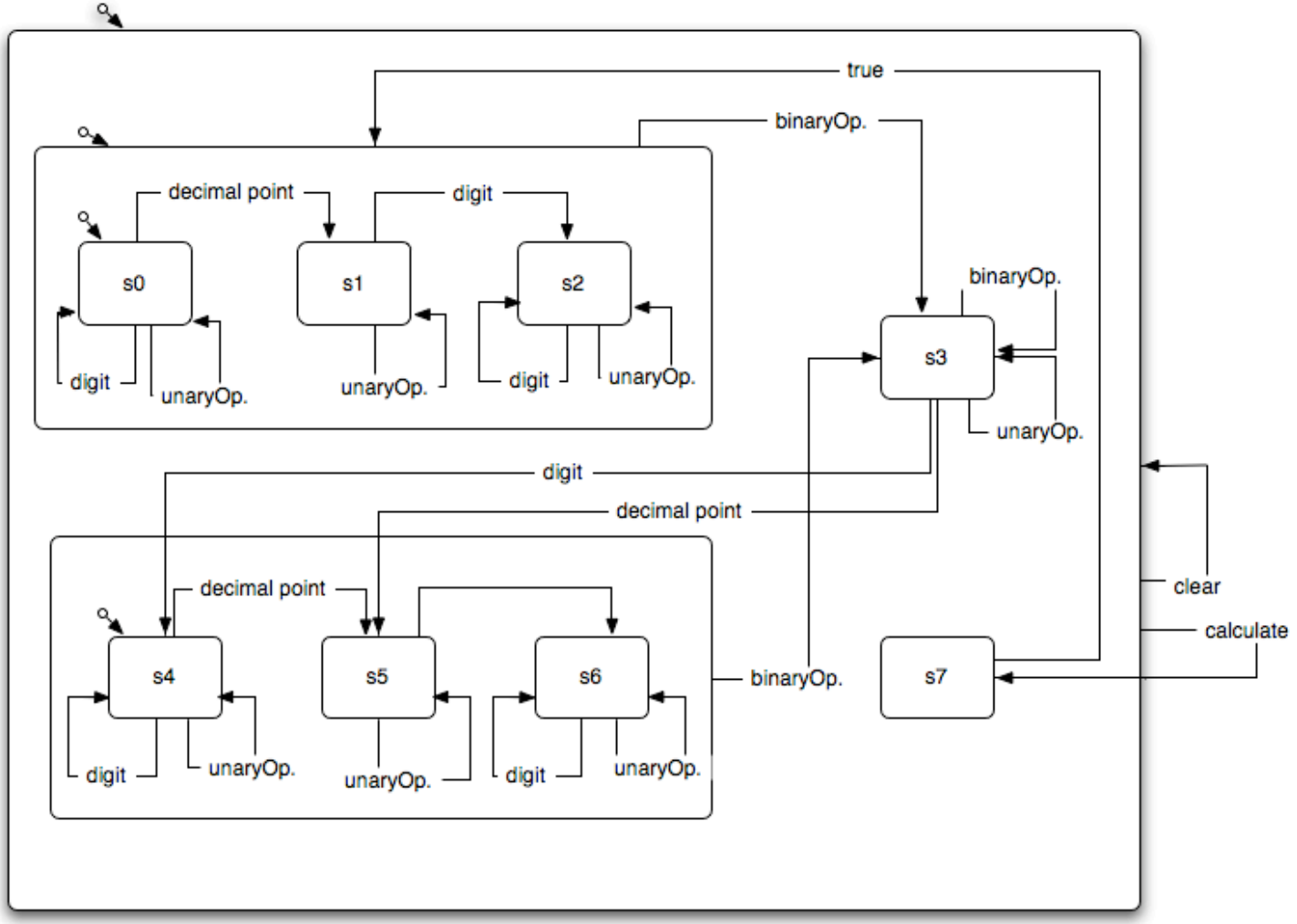
- Gut geeignete Notation: ein Statechart.

- Beschreibung der internen Zustände des Taschenrechners:

```
state = 0 -> Initialzustand, Eingabe Vorkommastellen der ersten Zahl.
state = 1 -> Eingabe Nachkommastellen (erste Zahl) gewünscht.
state = 2 -> Eingabe Nachkommastellen der ersten Zahl.
state = 3 -> Operation angegeben.
state = 4 -> Eingabe Vorkommastellen der zweiten Zahl.
state = 5 -> Eingabe Nachkommastellen (zweite Zahl) gewünscht.
state = 6 -> Eingabe Nachkommastellen der zweiten Zahl.
state = 7 -> Berechnung durchgeführt.
```

2.2. Reverse Engineering (2)

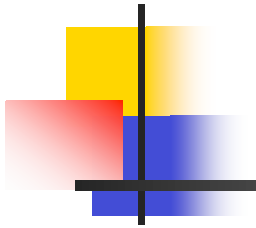
Abstraktes Verhalten der Klasse Calculator-Engine.



2.3. Nachdokumentation von Code (1)

- Aufgabe wurde gut gelöst.
- 2x Klassenkommentar
- 3x Instanzvariable in Engine
- Beschreibung der Zustände (vgl. Aufgabe 2.2)
- 13x Methodenkommentar in Engine
- 2x Methodenkommentar in Display
- Jeweils -1/4 punkt.

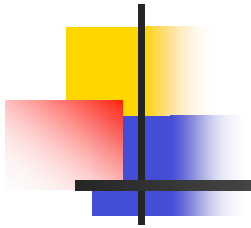
2.3. Nachdokumentation von Code (2)



- Folgendes Muster für Klassenkommentare:

```
/**
 *  Hier kommt eine Zweckbeschreibung der Klasse. Mit {@link <java element>}
 *  kann auf eine Klasse, Methode oder ein Attribut referenziert werden.
 *
 *  @author             Name
 *  @copyright          copyright, fakultativ
 *  @history            yyyy-mm-dd Namenskuerzel kommentar
 *  @version            Versionsnummer, wird CVS eingesetzt
 *                      so wird durch CVS automatisch die entsprechende
 *                      Information in den folgenden String eingesezt:
 *                      $Date: 2003/03/10 14:25:15 $, $Author: smeier $, $Revision: 1.1 $
 *                      Dieser String kann bei Verwendung von CVS beim Version-Tag
 *                      eingesetzt eruebrigt ein manuelles nachfuehren der Versionsnummer.
 *  @responsibilities   Verantworklichkeiten
 *  @invariant          Invariante, fakultativ
 *  @obligation         Verpflichtungen, fakultativ
 *  @since              besteht seit Version, fakultativ
 *  @see                java element, fakultativ
 */
```

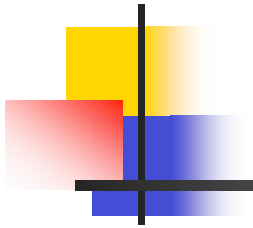
2.3. Nachdokumentation von Code (3)



- Folgendes Muster für Methodenkommentare:

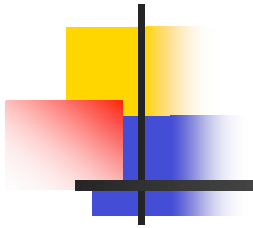
```
/**
 *   @pre Vorbedingung
 *   @post Nachbedingung
 *   @invariant Invariante der Methode, fakultativ
 *   @obligation Verpflichtung der Methode, fakultativ
 *   @param Parametername Beschreibung des Parameters
 *   @return Beschreibung des Rueckgabewertes
 *   @throws Exceptions die durch diese Methode geworfen werden, fakaultativ
 *   @since Versionsnummer, seit die Methode existiert
 *   @see Referenziertes Java-Element, fakultativ
 *   @deprecated angeben, falls Methode veraltet ist, fakultativ
 */
```

2.3. Nachdokumentation von Code (4)



- Umbenennungen:
  - ```
    /**
     * Stack fuer die Operanden. Der oberste Wert im Operandenstack
     * repraesentiert den Akku des Rechners.
     */
    private Stack ops;
                                --> besser: private Stack operands;

    /**
     * Durchzufuehrende Operation.
     */
    private Op op;
                                --> besser: private Op operation; (bzw. auch die Klasse Op umbenennen)
    ```
  - ```
    Stack<String> ops() { ... }                 -->  Stack<String> getOperands() { ... }
    void ops( Stack<String> aStack ) { ... }    -->  void setOperands( Stack<String> operands ) { ... }
    Op op() { ... }                             -->  Op getOperand() { ... }
    void op( Op anOp ) { ... }                  -->  void setOperands( Op anOp ) { ... }
    ```

2.3. Nachdokumentation von Code (5)

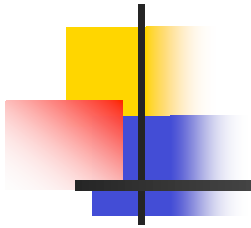


- Gute Loesung der Gruppe Lawniczak, Huber, Zuberbuehler:
  - u. a. Benennung der States mit Konstanten:

```
/**
* Represents the entry state of the state machine
*/
private static final int START = 0;

/**
* Represents state 1 (input 0 and period pressed)
*/
private static final int ZERO_AND_PERIOD = 1;

/**
* State when the input is a zero followed by a period followed by number,
* eg. 0.34
*/
private static final int ZERO_PERIOD_NUMBER = 2;

/**
* State when a binary operator has been pressed
*/
private static final int BINARY_OPERATOR_PRESSED = 3;

...
```

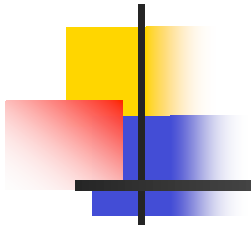
Ein Ausdruck kann eingesehen werden.

2.4. Fehlerbehebung und Erweiterung (1)



- Die Aufgaben meist *gut* gelöst;
  - teilweise leider auch gar nicht gelöst.
- Teilweise schlechtes Layout der Tasten am Calculator.
- Teilweise enthielt der Code noch Fehler --> nicht kompilierbar
  - Bitte vor der Abgabe immer testen, bzw. selbst kompilieren und mit abgeben.
- Korrektur: Black-Box Tests der Funktionalitäten.

2.4. Fehlerbehebung und Erweiterung (2)



- Aufgabe 1: Führende Nullen
  - Häufiger Fehler: Clear, VZ-Wechsel und Zahl -- meist führende Null.
  - Zweite Zahl einer binären Operation.
  - Auch: Clear und Operator -- teilweise Exception.
- Aufgabe 2: Vorzeichenwechsel / Nachkommastelle
  - Häufiger Fehler: binäre Operation z.B.: 2 + 4 ergab oft 6.0
    - Kommastelle sollte auch beim Ergebnis nicht erscheinen (wenn nicht nötig).
- Aufgabe 3: Quadratwurzel und Quadrieren
  - Generell gut gelöst. Häufiger Fehler: keine Realisierung als *unäre Operation*.
    - 2 + 2^2 = 6, nicht: 2 + 2^2 = 16 (entspricht: (2 + 2)^2)
  - Teilw. gab es bufferOverflows oder indexOutOfBounds Exceptions.
- Individuelle Korrekturen können auf Anfrage eingesehen werden.

- Schönes Wochenende!