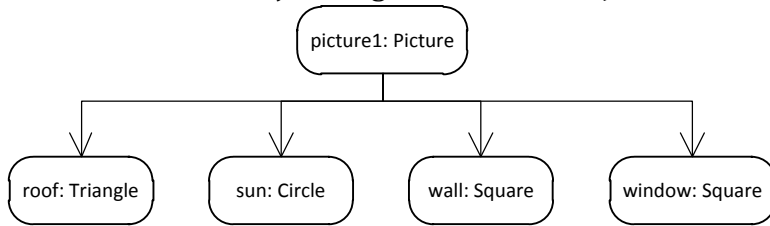


Mit Kapitel 3.6

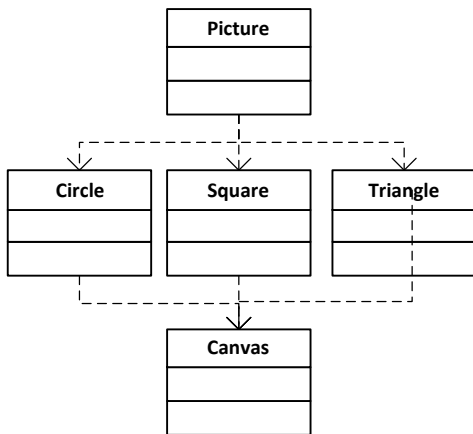
1. Wie deklarieren Sie eine Referenzvariable?

Mit „new“ z.B. `Student studentA = new Student("Meier");`

2. Zeichnen Sie das Objektdiagramm zum BlueJ Picture Projekt aus Kapitel 1 (Übung 1.10).



3. Erstellen Sie nun auch noch das Klassendiagramm zum BlueJ Picture Projekt aus

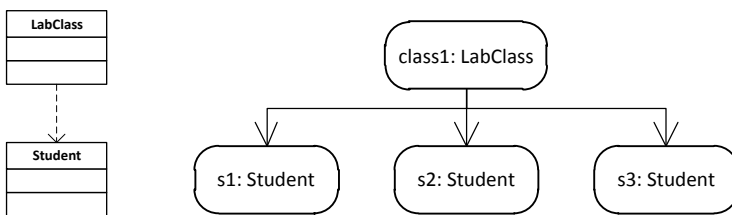


Kapitel 1 (Übung 1.10).

4. zu bearbeitende Aufgaben: 3.1 bis 3.4

3.1:

Das Klassendiagramm (links) zeigt die zwei Klassen. Im Objektdiagramm (rechts) sind die einzelnen Objekte sichtbar, z.B. das es drei Studenten-Objekte gibt.



3.2:

Wenn sich der Source Code ändert. Wenn z.B. eine neue Klasse dazukommt.

3.3:

Zur Laufzeit des Codes. Wenn z.B. eine neue Instanz erstellt wird.

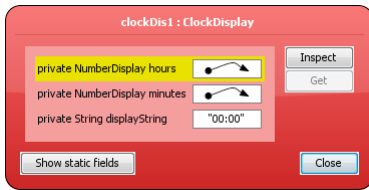
3.4:

`private Instructor tutor;`

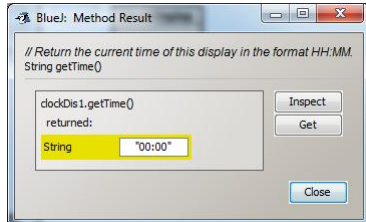
Kapitel 3.8

5. zu bearbeitende Aufgabe: 3.5

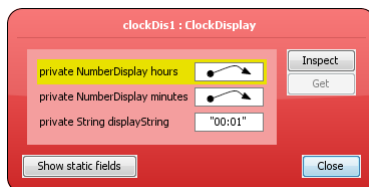
3.5:



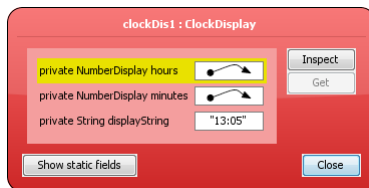
getTime();



timeTick();



setTime(13, 5);



6. Welche Werte liefern die folgenden Ausdrücke:

$(3 > 2) \wedge (4 > 5)$	True
$(3 < 2) \wedge (4 > 5)$	False
$(3 < 2) \&\& (4 > 5)$	False
$(3 > 2) \parallel (4 > 5)$	True
$!(3 > 2)$	False

7. zu bearbeitende Aufgaben: 3.6 bis 3.8

3.6:

Der Wert wird nicht gesetzt, es wird aber auch kein Error o.ä. ausgegeben.

3.7:

Es kann nicht mehr der Wert 0 gesetzt werden.

3.8:

Es muss nur eine der beiden Bedingungen erfüllt sein. In diesem Fall würde z.B. auch 287 einen gültigen Wert sein.

8. zu bearbeitende Aufgaben: 3.15 bis 3.17, 3.19

3.15:

*Modulo gibt jeweils den Rest einer Integer Division zurück.**Bsp: $27 \% 4 \Rightarrow 27 / 4 = 6 \text{ Rest } 3 \Rightarrow \text{Modulo} = 3$*

3.16:

 $8 \% 3 = 2$

3.17:

 $8 \% 3$
2 (int) $-4 \% 3$
-1 (int) $4 \% -3$
1 (int) $-4 \% -3$
-1 (int)

Für das Vorzeichen vom Ergebnis wird nur das Vorzeichen des ersten Operanden beachtet.

3.19:

Integer

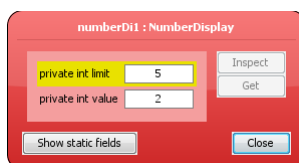
9. zu bearbeitende Aufgaben: 3.21 und 3.22

3.21:

```
/**
 * Increment the display value by one, rolling over to zero if the
 * limit is reached.
 */
public void increment()
{
    if ((value + 1) >= limit) {
        value = 0;
    }
    else {
        value += 1;
    }
}
```

Mit Modulo wird der Code viel kürzer, ggf. jedoch schwieriger zum lesen.

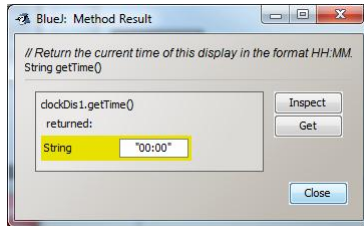
3.22:



Kapitel 3.9

10. zu bearbeitende Aufgaben: 3.23 bis 3.27

3.23:



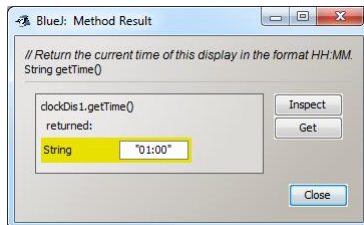
```
public ClockDisplay()
{
    hours = new NumberDisplay(24);
    minutes = new NumberDisplay(60);
    updateDisplay();
}

public NumberDisplay(int rolloverLimit)
{
    limit = rolloverLimit;
    value = 0;
}
```

Im Konstruktor wird value mit 0 initialisiert.

3.24:

60 mal, alternativ könnte man setTime(1,0) aufrufen.



3.25:

```
NumberDisplay nd = new NumberDisplay(80)
Error: ';' expected
```

```
NumberDisplay nd = new NumberDisplay(80);
```

```
nd.getValue()
0 (int)
```

```
nd.setValue(79)
Error: ';' expected
```

```
nd.setValue(79);
```

```
nd.getValue()
79 (int)
```

```
nd.increment();
nd.increment();
nd.getValue()
1 (int)
```

3.26:

Editor(String fileName, int newInt)

3.27:

private Rectangle window;

window = new Rectangle(5, 15);

Kapitel 3.10

11. Erstellen Sie die Signaturen aller möglichen Konstruktoren, welche mit der folgenden

Objektkreierung übereinstimmen:

```
new Student("Peter", 34);
public Student(String myName, int myAge)
{
    [...]
}

public Student(String myName, short myAge)
{
    [...]
}

public Student(String myName, byte myAge)
{
    [...]
}
```

12. zu bearbeitende Aufgaben: 3.28 und 3.29

3.28:

Es werden zwei Instanzen vom Typ `NumberDisplay` erstellt, eine für die Minuten mit dem Limit-Wert 60 und einer für die Stunden mit dem Limit-Wert von 24. Danach wird noch die Methode `setTime()` aufgerufen und die dem Konstruktor übergebenen Zeit gesetzt.

3.29:

Der zweite Konstruktor erwartet zwei Parameter (Stunden, Minuten). Beim Konstruktor ohne Parameter werden nur die zwei `NumberDisplay` instanziiert, somit muss explizit `updateDisplay()` aufgerufen werden. Beim zweiten Konstruktor wird dies innerhalb der Methode `setTime()` ausgeführt.

Kapitel 3.11

13. zu bearbeitende Aufgaben: 3.30

3.30:

```
p1.print("test.txt", False);
p1.print("test2.txt", True);
p1.getStatus(50);
p1.getStatus(0);
```

Kapitel 3.12

14. zu bearbeitende Aufgaben: 3.33 und 3.34

3.33:

done.

3.34:

object diagram

Kapitel 3.13

15. zu bearbeitende Aufgaben: 3.35 bis 3.42

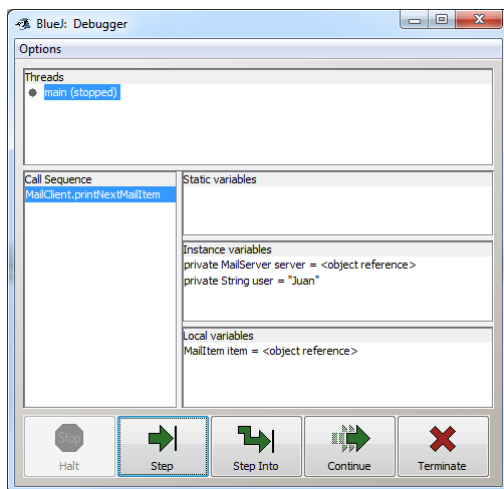
3.35:



3.36:

```
public void printNextMailItem()
{
    MailItem item = server.getNextMailItem(user);
}
```

3.37:



3.38:

Es wird die jeweils nächste Linie ausgewählt welche ausgeführt wird. Z.b: IF-Statement, und nicht einfach die nächste Zeile im Source-Code.

3.39:

if(item == null) wird ausgeführt, es gibt kein weiteres Mail welches abgerufen werden könnte.

3.40:

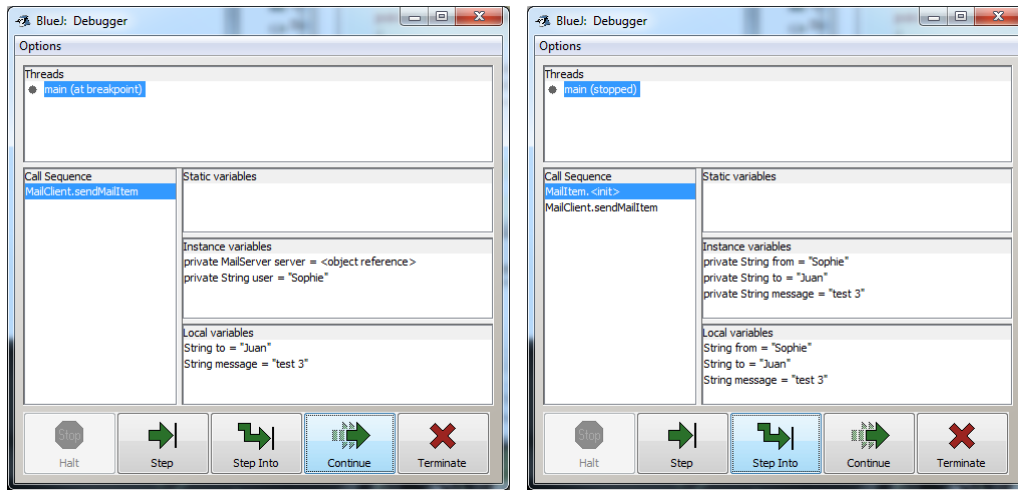
```
public void print()
{
    System.out.println("From: " + from);
    System.out.println("To: " + to);
    System.out.println("Message: " + message);
}
```

From: Sophie

To: Juan

Es wird jeder einzelne Schritt angezeigt, inkl. deren Sub-Routinen.

3.41:



3.42: