Bitte tragen Sie individuell bis 08:00 Uhr am Vortag des Seminars über den Checklistenbaustein in Opal ein, welche Aufgaben Sie präsentieren können. Für die Präsentation sollten Sie ein Dokument im Querformat vorbereitet haben, das Sie dann entsprechend im Seminar in BigBlueButton hochladen können. Auch müssen Sie den virtuellen Klassenraum mit Mikrofon betreten.

Aufgabe 1: Zyklomatische Komplexität

Betrachten Sie die folgende C-Funktion.

```
1  void output( int selection, int y) {
2     while (selection < y) {
3       if (selection == 1)
4         printf("\n 1");
5       else
6         if (selection == 2)
7           printf("\n 2");
8         else
9           if (selection == 3)
10            printf("\n 3");
11          else
12            if (selection == 4)
13              if (y == 0)
14                printf("\n 4");
15              else
16                selection++;
17            else
18              y=1;
19      y--;
20    }
21    printf("\n 5");
22 }
```

a) Stellen Sie den Kontrollflussgraphen auf.

b) Berechnen Sie die zyklomatische Komplexität.

Aufgabe 2: Testen

Betrachten Sie die nebenstehende Java-Methode.
Gesucht ist jeweils eine Menge an Testfällen für vollständige Anweisungs-, Zweig- und Termüberdeckung.

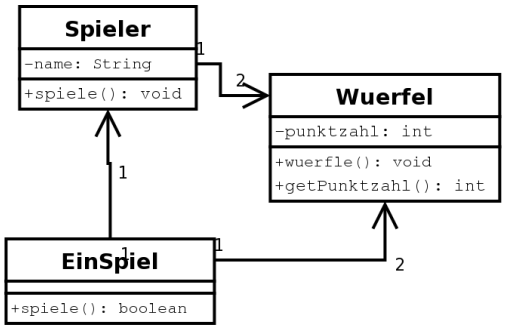
```
   public int compute (int x, int y) {
2:   int result = 100;
3:   if ( (x < 30) || (x > y) )
4:     result = result * 2;
     else
6:     result = result + x;
7:   if ( (result > 140) && (y >= 100) )
8:     result = 0;
9:   return result;
   }
```

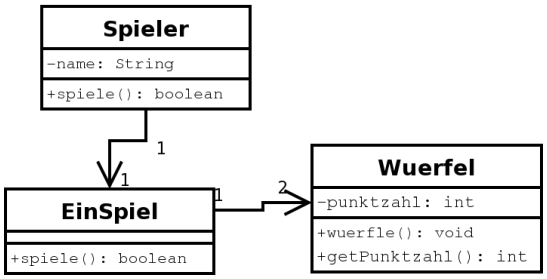
Aufgabe 3: Implementation nach zwei Entwürfen

In einem einfachen „Spiel“ würfelt ein Spieler mit zwei Würfeln. Ist deren Summe 7 hat er gewonnen, sonst verloren. Es liegen zwei unterschiedliche Entwürfe als Klassendiagramm vor, woraus sich der Ablauf des Programms ergibt.

a) Geben Sie jeweils den Ablauf als Sequenzdiagramm an.

b) Geben Sie jeweils die Klasse `EinSpiel` in Java an.





Aufgabe 4: Klassendiagramm und Quelltext

Demonstrieren Sie mit einem kleinen Beispiel, wie eine Assoziation $^{*} \longrightarrow^{1}$ beim Programmieren in Java umgesetzt wird.

Aufgabe 5: Testen II

Betrachten Sie die nebenstehende Methode.

a) Stellen Sie den Kontrollflussgraphen auf.

b) Bestimmen sie für den Aufruf `toTest(7,8)` die Anweisungs-, Zweig und Termüberdeckung.

c) Gesucht ist jeweils eine (möglichst kleine) Menge an Testfällen für vollständige Anweisungs-, Zweig- und Termüberdeckung.

```
   public int toTest (int x, int y){
2:   while (x > 2 && x <= 7) {
3:     if (y > x)
4:       x = x+1;
       else {
5:       x= x/2;
6:       y = y-1;
       }
     }
9:   return x;
   }
```