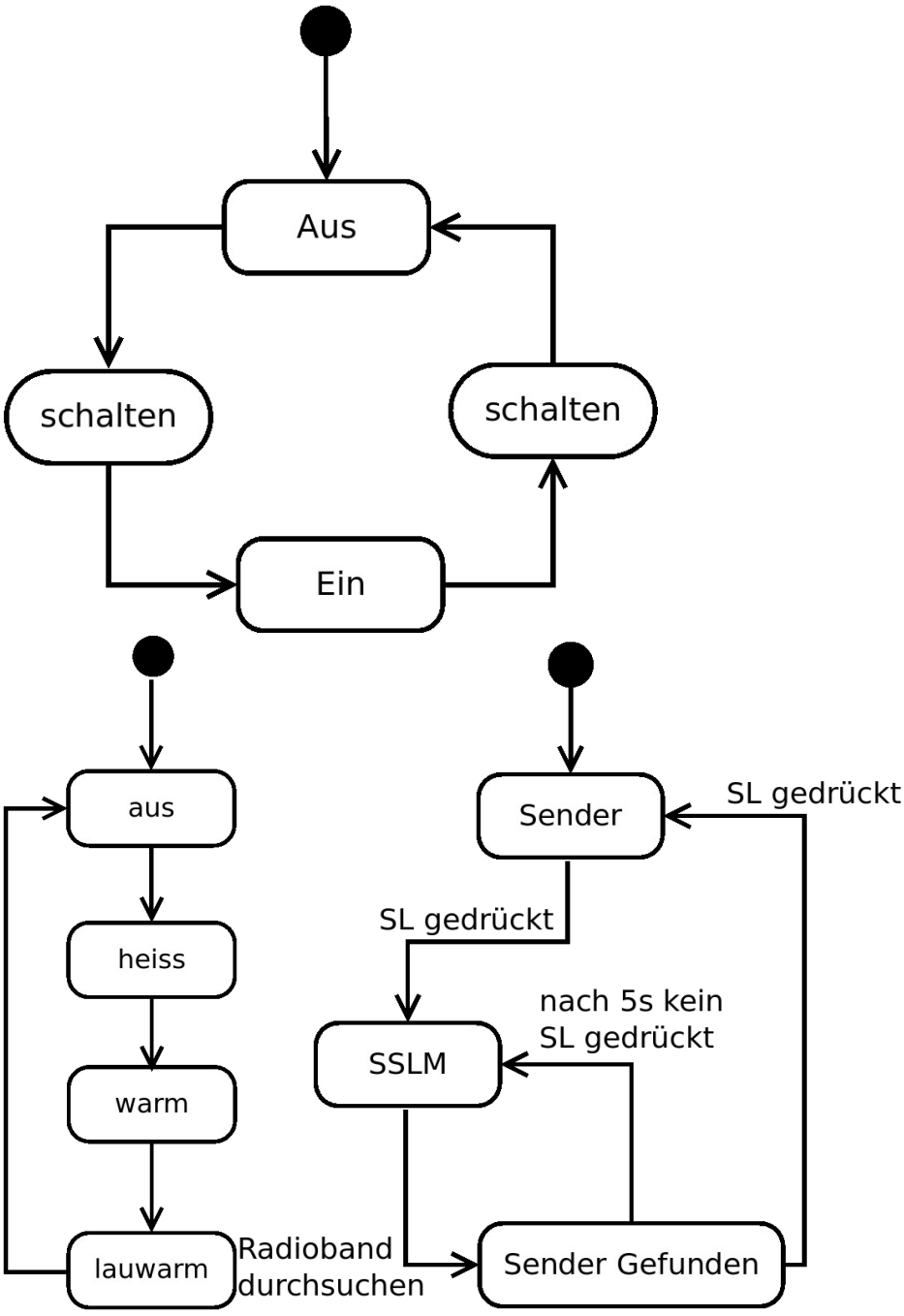
14.7 zeig einen Zustandsautomaten mit 2 Sub-Zuständen, dem Startzustand in dem der Klingelton abgespielt und angehalten wird und dem Wahlzustand, in dem Ziffer für Ziffer eine Nummer eingegeben wird. Der Endzustand wird dann erreicht, wenn die Nummer valide ist.

14.14 zeigt das Zustandsdiagramm eines Geldautomaten mit einem Sub-Zustand, in dem die Karte verifiziert wird, der Betrag ausgelesen wird und die Karte entweder nach Abbruch oder nach erfolgreicher Transaktion wieder ausgegeben wird.

14.13 zeigt den Sub-Zustand des Erfassens des auszuzahlenden Betrages aus 14.14, in welchem der Betrag entweder selbst gewählt oder aus einer vorgefertigten Liste gewählt wird, was Zum Ende des Sub-Zustandes führt. Alternativ kann man mit Abbruch das Ende des Zustandes herbeiführen.

14.32 zeigt ein Zustandsdiagramm mit Verzweigung, je nach Größe der ID wird entweder der linke oder der Rechte Zweig des Diagramms abgelaufen



Der Pseudozustand ist ein Steuerungselement, das den Ablauf eines Zustandsautomaten beeinflusst. In einem Pseudozustand existieren im Unterschied zu einem echten Zustand keine Wertebelegungen.

Startzustand:
hat keine eingehenden Transitionen, gibt an in welchem Zustand begonnen wird

Endzustand:
hat keine ausgehenden Transitionen, das modellierte Objekt hört auf zu existieren

Vereinigung/Gabelung:
Vereinigung von mehreren parallelen Zuständen bzw Gabelung in mehrere parallele Zustände

Kreuzung/Entscheidung:
mehrere Abgehende Transitionen

Eintrittspunkt:
Zugangspunkt zu einem Sub-Zustand

Austrittspunkt:
Ausgangspunkt von einem Sub-Zustand

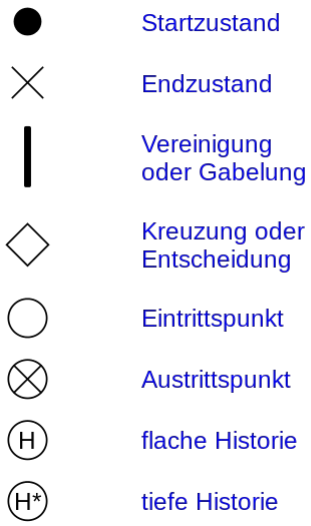


Schaubild 1: https://de.wikipedia.org/wiki/Zustandsdiagramm_(UML)#/media/Datei:Uml-Zustandsdiagramm-4.svg

flache Historie:
zuletzt eingenommener Zustand des Subzustandes wird gemerkt und beim erneuten Aufrufen eingenommen (nicht rekursiv auf eventuelle Sub-Sub-Zustände)

tiefe Historie:
wie flache Historie, nur incl. „erinnern“ an Sub-Sub-Zustände

```java
import java.util.Scanner;

enum States{
  TIME, HRS, MIN;
}

public class Uhr_States {
  States state = States.HRS;

  public void do_State(){
    switch (state){
      case HRS:
        System.out.println(hrs);
        break;
      case MIN:
        System.out.println(min);
        break;
      case TIME:
        getTime();
        break;
      default:
        System.out.println("something went wrong...");
        break;
    }
  }

  public void inc(){
    switch (state){
      case HRS:
        hrs++;
        hrs = hrs%24;
        break;
      case MIN:
        min++;
        min = min%60;
        break;
      case TIME:
        System.out.println("cannot increment right now (State: TIME)");
        break;
      default:
        System.out.println("something went wrong...");
        break;
    }
  }

  public void set(){
    switch (state){
      case TIME:
        state = States.HRS;

        break;
      case HRS:
        state = States.MIN;
        break;
      case MIN:
        state = States.TIME;
        break;
      default:
        System.out.println("State Error, RESETTING...");
        hrs = 0;
        min = 0;
```

```
        state = States.HRS;
        break;
      }
    }

    public void getTime(){
      System.out.println("current time is "+hrs+" hrs and "+min+" minutes");
    }

    private int min = 0;
    private int hrs = 0;

    public static void main(String[] args) {
      Uhr_States uhr = new Uhr_States();
      Scanner myObj = new Scanner(System.in);  // Create a Scanner object
      while(true) {
        System.out.println(uhr.state);
        System.out.println("Enter action");

        String action = myObj.nextLine();  // Read user input

        switch (action) {
          case "+":
            uhr.inc();
            uhr.do_State();
            break;
          case "s":
            uhr.set();
            uhr.do_State();
            break;
          default:
            System.out.println("enter valid action [+(increment), s(set)]");
        }
      }
    }

}
```