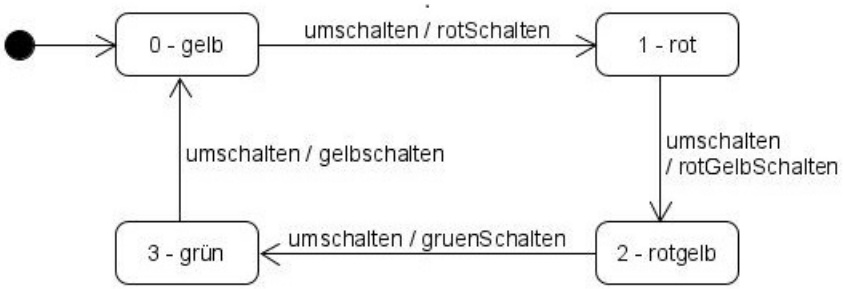
7 Zustandsdiagramm – Umsetzung in Java

Das Zustandsdiagramm der Ampel soll nun in Java realisiert werden:



Für die Umsetzung der Ampel in Java müssen alle Zustände in geeigneter Weise abgespeichert werden. Für eine Sammlung von konstanten Werten eignet sich in Java eine so genannte **enum**. Die Zustände für eine Ampel könnten damit wie folgt erfasst werden:

```
public enum Ampelzustand{



}
```

Die Klasse `Ampel` wird nun wie folgt angepasst:

```
public class Ampel{
  private Rechteck gehaeuse;
  private Kreis gruenerKreis, gelberKreis, roterKreis;
  //Erweitern um eine Variable für den Zustand


  //Startzustand herstellen
  public Ampel(int positionX, int positionY, int breite){

   int hoehe = 3*breite;
   int kreisePosX = positionX + breite/2;
   int radius = breite/2;
   gehaeuse = new Rechteck(positionX, positionY, breite, hoehe, "schwarz");
   roterKreis = new Kreis(kreisePosX, positionY + hoehe/6, radius,          );
   gelberKreis = new Kreis(kreisePosX, positionY + hoehe/2, radius,          );
   gruenerKreis = new Kreis(kreisePosX, positionY + hoehe*5/6, radius,
                  );}
```

```
  //Realisierung mit einem switch-case-Konstrukt
  public void umschalten(){









  }
```

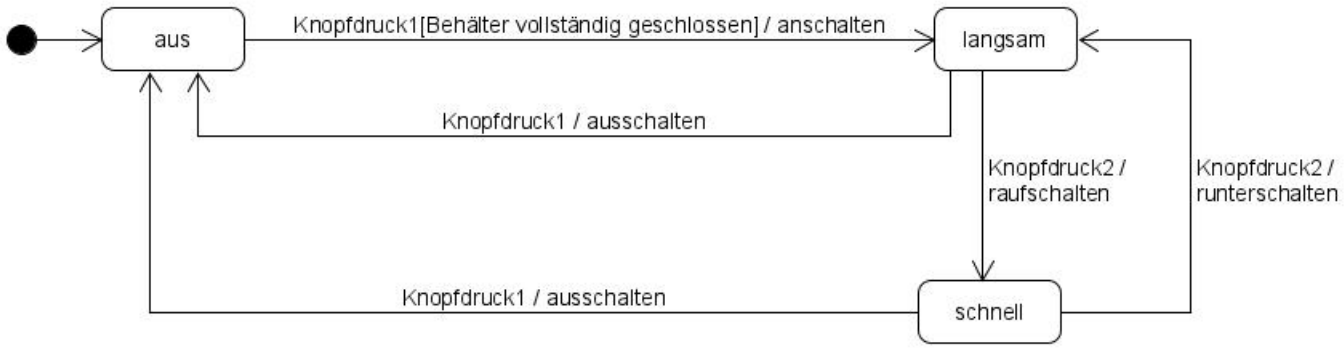
```
}
```

Diese Implementierung kann beispielsweise mit der Klasse `Leinwand` mit folgendem Programm getestet werden:

```
public class AmpelTest{
   public static void main(String[] args){
      Ampel a = new Ampel();
      Leinwand l = new Leinwand();
      a.zeichne(l);
      //Die Ampel sollte gelb leuchten
      l.warte(1000);
      for(int i = 0; i<4; i++){
         a.umschalten();
         a.zeichne(l);
         l.warte(1000);
         //Man sollte einen Wechsel von gelb zu rot zu rot-gelb zu grün
         //und anschließend wieder zu gelb beobachten
      }
   }
}
```

Aufgaben:

1. Gegeben ist folgendes Zustandsdiagramm eines Mixers:



Dieses komplexere Zustandsdiagramm soll nun schrittweise implementiert werden.

(a) Erstellen Sie ein neues Java-Projekt und dort eine Enum `MixerZustand` mit entsprechenden Einträgen für die möglichen Zustände des Mixers.

(b) Erstellen Sie nun eine Klasse `Mixer` mit einem passenden Attribut für den Zustand des Mixers und einer `boolean`-Variable `behaelterGeschlossen`. Legen Sie ebenfalls einen Konstruktor an, der den Startzustand des Mixers herstellt und die Variable `behaelterGeschlossen` mit `true` initialisiert.

(c) Erstellen Sie für die sechs Aktionen jeweils eine entsprechende `void`-Methode in der Klasse `Mixer`, die folgende Nachrichten auf der Konsole ausgibt:

- `knopfdruck1()`: „Knopf 1 wurde gedrückt“
- `knopfdruck2()`: „Knopf 2 wurde gedrückt“
- `ausschalten()`: „Mixer wird ausgeschaltet“
- `einschalten()`: „Mixer wird eingeschaltet“
- `raufschalten()`: „Geschwindigkeit wird erhöht“
- `runterschalten()`: „Geschwindigkeit wird verringert“

Überlegen Sie sich für jede Methode, welche Sichtbarkeit jeweils passend ist.

(d) Erweitern Sie nun jeweils die Methoden `knopfdruck1()` und `knopfdruck2()` um `switch-case`-Konstrukte, so dass in diesen Methoden der passende Zustandswechsel realisiert wird und die entsprechenden Methoden für die ausgelösten Aktionen aufgerufen werden.
Falls eine auslösende Aktion keine Zustandsänderung hervorruft, so soll auf der Konsole „**Mit dieser Aktion keine Zustandsänderung**“ ausgegeben werden.

(e) Erstellen Sie eine Klasse `MixerMain` mit folgender `main`-Methode und überprüfen Sie, ob Ihre Konsolenausgabe mit der abgebildeten übereinstimmt:

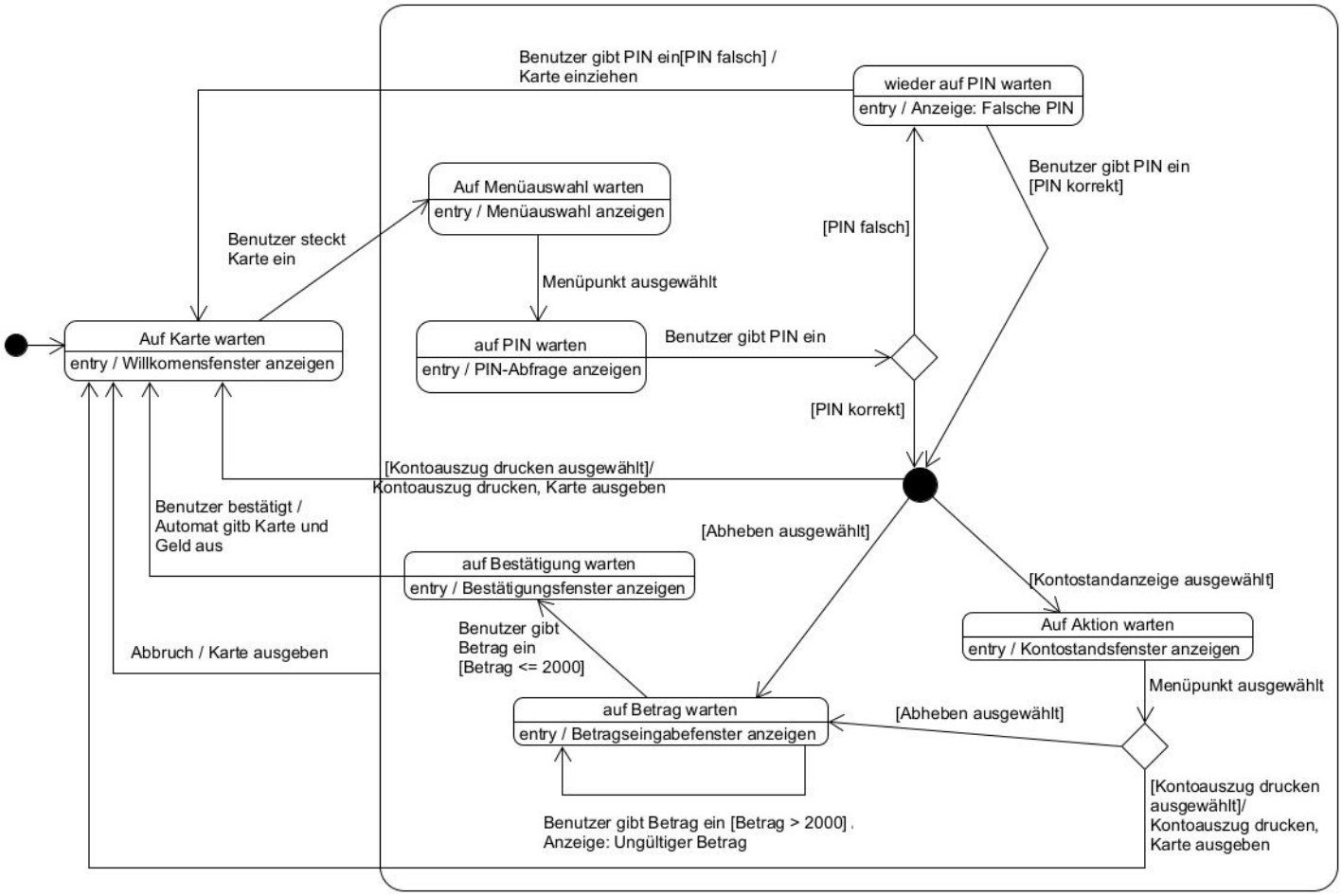
```
public class MixerMain {

   public static void main(String[] args) {
      Mixer m = new Mixer();
      m.knopfdruck1();
      m.knopfdruck2();
      m.knopfdruck2();
      m.knopfdruck1();
      m.knopfdruck2();
      m.knopfdruck1();
   }
}
```

Konsolenausgabe:

```
Knopf 1 wurde gedrückt
Mixer wird eingeschaltet
Knopf 2 wurde gedrückt
Geschwindigkeit wird erhöht
Knopf 2 wurde gedrückt
Geschwindigkeit wird verringert
Knopf 1 wurde gedrückt
Mixer wird ausgeschaltet
Knopf 2 wurde gedrückt
Mit dieser Aktion keine Zustandsänderung
Knopf 1 wurde gedrückt
Mixer wird eingeschaltet
```

2. Gegeben ist folgendes Zustandsdiagramm eines Geldautomaten:



Dieses Zustandsdiagramm soll nun schrittweise implementiert werden, dazu finden Sie im Klassenlaufwerk die entsprechende Eclipse-Projekt-Vorlage `AufgabeGeldautomat`.

(a) Vorbereitungen:

i. Importieren Sie dieses Projekt in Ihre Entwicklungsumgebung. In diesem Projekt finden Sie die beiden Ordner `src` und `test`. Für die folgenden Aufgaben ist erstmal nur der Ordner `src` relevant, der Ordner `test` ist dabei erst ab Teilaufgabe (c) relevant.
Importanleitung für

- Eclipse
- IntelliJ (Eventuell müssen Sie selbstständig JUnit 5 selbstständig zum `classpath` hinzufügen)
- VS Code: Ordner entpacken und mit VS Code öffnen.

ii. Bei den Attributen `ABHEBELIMIT` und `pin` in der Klasse `Geldautomat` finden Sie die Schlüsselwörter **`static`** und **`final`**. Informieren Sie sich über diese Schlüsselwörter und erklären Sie diese kurz in eigenen Worten. Wie kann mit Hilfe des Schlüsselwortes `static` gezählt werden, wie viele Objekte einer Klasse erzeugt wurden?

iii. Um die Implementierung des komplexeren Zustandsdiagramms am Ende zu testen, ist der Vergleich einer Konsolenausgabe nicht mehr passend. Die Implementierung soll dafür mit **JUnit-Tests** getestet werden. Informieren Sie sich daher zuerst über allgemeine Begriffe zum Thema Testen:

- Schreibtisch-Test
- Black-Box-Test
- White-Box-Test
- Testautomatisierung

In welcher Verbindung zu diesen Begriffen stehen **JUnit-Tests**?

iv. Bei genauerer Betrachtung der Datei `GeldautomatTest` im Package `test` finden Sie einige Annotationen. Informieren Sie sich über diese. Grenzen Sie zudem die Annotationen `BeforeEach` und `AfterEach` von `BeforeAll` und `AfterAll` ab.

- `TestMethodOrder`
- `BeforeEach`

- `AfterEach`
- `DisplayName`
- `Test`

v. Die Methode `assertEquals` spielt bei JUnit-Tets eine zentrale Rolle. Informieren Sie sich über diese Methode und erklären Sie diese kurz.

(b) Implementierung:
Hinweise:

- Achten Sie darauf, dass Sie bei Zustandswechseln auch die Methoden für die ausgelösten Aktionen aufrufen bzw. die Methode `ungueltigeAktion()`, falls die aufgerufene Methode im aktuellen Zustand nicht vorgesehen ist.
- Nachdem Sie einen Aufgabenabschnitt bearbeitet, können Sie Ihre Implementierung testen, indem Sie die Datei `GeldautomattTest` als JUnit-Test ausführen (in Eclipse: Rechtsklick auf die Datei → `Run As` → `JUnit Test`).

i. Stellen Sie im Konstruktor den Startzustand her.

ii. Implementieren Sie die Methoden `karteEinstecken()` und `abbruch()`.

iii. Implementieren Sie die Methode `menuePunktAuswaehlen(Menuepunkt m)`. Achten Sie darauf, dass die Auswahl eines Menüpunktes an zwei Stellen im Zustandsdiagramm möglich ist und der ausgewählte Menüpunkt nach der korrekten Eingabe der PIN wieder benötigt wird.
Hinweis: Sollte ein ungültiger Menüpunkt übergeben werden, soll die Methode `ungueltigeAktion()` aufgerufen werden.

iv. Bevor nun die Methode `pinEingabe(String pin)` realisiert werden kann, benötigt man die Methode `pinPruefen(String pin)`.

   α) Implementieren Sie die Methode `pinPruefen(String pin)` so, dass `true` zurückgegeben wird, wenn der Inhalt des übergebenen Parameter mit dem Inhalt der Klassenvariable `pin` übereinstimmt. Andernfalls soll `false` zurückgegeben werden.
   Wieso sollte für diesen Vergleich nicht == verwendet werden?

   β) Implementieren Sie nun mithilfe der Methode `pinPruefen` die Methode `pinEingabe(String pin)`.

v. Implementieren Sie abschließend die Methoden `betragEingabe(int betrag)` und `betragBestaetigen()`. Achten Sie dabei auf die Variablen `kontostand` und `auszuzahlenderBetrag`.

vi. Es gibt noch einige Abläufe in Ihrer Implementierung, die noch nicht durch JUnit-Tests abgedeckt sind. Erstellen Sie daher für folgende Szenarien in der Klasse `GeldautomatTest` jeweils einen weiteren JUnit-Test:
   - Nachdem eine Karte eingesteckt wurde, „`Kontostand anzeigen`“ ausgewählt wurde, die PIN zunächst falsch eingegeben wurde, anschließend die PIN jedoch korrekt eingegeben wurde, befindet sich der Automat im Zustand „`Auf Aktion warten`“.

   - Nachdem eine Karte eingesteckt wurde, „`Kontoauszug drucken`“ ausgewählt wurde, die PIN zweimal falsch eingegeben wurde, wird die Karte eingezogen und der Geldautomat befindet sich im Zustand „`Auf Karte warten`“.

(c) **Für Experten:**

i. Generell können bei Bankautomaten keine Münzen ausgegeben werden. Passen Sie daher die Methode `betragPruefen(int betrag)` so an, dass nur noch Beträge gültig sind, die sich unter 2000€ befinden und auch in Scheinen ausbezahlt werden können.

ii. Der Text der Konsolenausgaben bei Eintritt eines neuen Zustandes sollte bei einem sauberen Design zu den entsprechenden Enum-Einträgen gehören. Ergänzen Sie die Enum `GeldautomatZustand` um ein `String`-Attribut `text` und ein Konstruktor `GeldautomatZustand (String text)`, der das Attribut auf den übergebenen Parameter setzt. Implementieren Sie dort zudem eine Methode `public String toString()`, die den Attributwert von `text` zurückgibt. Verwenden Sie diese Methode in der Klasse `Geldautomat` für die Konsolenausgabe bei einem Eintritt in einen neuen Zustand. Löschen Sie anschließend nicht mehr verwendete Attribute und passen Sie ebenfalls die JUnit-Tests entsprechend an.

iii. Erweitern bzw. verändern Sie die Implementierung, so dass nach einer zweiten fehlerhaften PIN-Eingabe, die PIN noch ein drittes Mal eingegeben werden kann und erst nach der dritten fehlerhaften PIN die Karte eingezogen wird. Passen Sie dazu ebenfalls ihre selbst geschriebenen JUnit-Test aus Aufgabe (b) an.