Aufgabe 1 – until_1

$s \models A \ \mathit{until}\ B$ **gdw.**

es gibt $s' > s$ mit $t \models A$ für alle $s < t < s'$ und $s' \models B$

$s \models A \ \mathit{until}_1\ B$ **gdw.**

(1) es gibt $s' > s$ mit $t \models A$ für alle $s < t < s'$ und $s' \models B$ oder
(2) für alle $s' > s$ gilt $s' \models A$

Unterschied

Bei until_1 muß B nicht wirklich irgendwann wahr werden, sondern es genügt auch, wenn A für immer wahr bleibt.

Aufgabe 1 – until_1

Ausdrückbarkeit

$$\begin{aligned}
\Diamond A &\equiv \mathit{true}\ \mathit{until}\ A \\
\Box A &\equiv \neg\Diamond\neg A \\
&\equiv \neg(\mathit{true}\ \mathit{until}\ \neg A) \\
A\ \mathit{until}_1\ B &\equiv (A\ \mathit{until}\ B) \lor \Box A \\
&\equiv (A\ \mathit{until}\ B) \lor \neg(\mathit{true}\ \mathit{until}\ \neg A)
\end{aligned}$$

$$\begin{aligned}
\Box A &\equiv A\ \mathit{until}_1\ \mathit{false} \\
A\ \mathit{until}\ B &\equiv (A\ \mathit{until}_1\ B) \land \neg\Box A \\
&\equiv (A\ \mathit{until}_1\ B) \land \neg(A\ \mathit{until}_1\ \mathit{false})
\end{aligned}$$

Aufgabe 1 – until_2

$s \models A \ \mathit{until}\ B$ **gdw.**

es gibt $s' > s$ mit $t \models A$ für alle $s < t < s'$ und $s' \models B$

$s \models A \ \mathit{until}_2 B$ **gdw.**

es gibt $s' \geq s$ mit $t \models A$ für alle $s \leq t < s'$ und $s' \models B$

Unterschied

Bei until_2 muß der Zustand in dem B wahr wird kein echter Folgezustand sein, sondern er kann auch mit dem aktuellen Zustand identisch sein.
Auch muß A im aktuellen Zustand gelten (es sei denn, B gilt sofort).

Aufgabe 1 – until_2

$s \models \mathit{next}\ A$ **gdw.**

$s' \models A$ für s' mit: $s < s'$ und es gibt kein t mit $s < t < s'$

Ausdrückbarkeit

$$A \ \mathit{until}_2 B \equiv B \lor (A \land (A \ \mathit{until}\ B))$$

$$\mathit{next}\ A \equiv \mathit{false}\ \mathit{until}\ A$$

$$A \ \mathit{until}\ B \equiv \mathit{next}(A \ \mathit{until}_2 B)$$

Aufgabe 2

Gegeben

Programm α_{square}

$$
\begin{array}{l}
\underline{\textbf{while}}\ i < x\ \underline{\textbf{do}} \\
\quad y := s(y + s(s(0)) * i); \\
\quad i := s(i) \\
\underline{\textbf{end}}
\end{array}
$$

Nat: **eine DL-Formel, die die natürlichen Zahlen charakterisiert.**

$$\phi \quad = \quad (Nat \land i \doteq 0 \land y \doteq 0) \to [\alpha_{\text{square}}]\, y \doteq x * x$$

Aufgabe 2 a

$$\phi \quad = \quad (Nat \land i \doteq 0 \land y \doteq 0) \to [\alpha_{\text{square}}]\, y \doteq x * x$$

Bedeutung der Formel ϕ

Haben die Symbole $<, \leq, 0, s, +$ die in der Arithmetik übliche Bedeutung und terminiert das Programm α_{square} nachdem es in einem Zustand gestartet wurde, in dem die Variablen i und y mit 0 belegt sind, dann ist im Endzustand die Variable y mit dem Quadrat der Zahl belegt, mit der x belegt ist.

Aufgabe 2 b

Regel:

$$\frac{\begin{array}{l}(Nat \land \gamma) \to inv \\ (Nat \land inv \land \epsilon) \to [\alpha]\, inv \\ (Nat \land inv \land \neg\epsilon) \to \phi\end{array}}{Nat \land \gamma \to [\underline{\textsf{while}}\ \epsilon\ \underline{\textsf{do}}\ \alpha\ \underline{\textsf{end}}]\, \phi}$$

Programm $\alpha_{\textsf{square}}$

$\underline{\textsf{while}}\ i < x\ \underline{\textsf{do}}$
$\quad y := s(y + s(s(0)) * i);$
$\quad i := s(i)$
$\underline{\textsf{end}}$

Geeignete Invariante

$$inv \quad = \quad i \leq x \land y \doteq i * i$$

Aufgabe 2 c

Zu zeigen

$$(Nat \land inv \land \epsilon) \to [\alpha]\, inv$$

$$(Nat \land \color{red}{i \leq x} \land y \doteq i * i \land \color{red}{i < x}) \to [\alpha]\, inv$$

$$(Nat \land i < x \land y \doteq i * i) \to [\alpha]\, inv$$

$$(Nat \land i < x \land y \doteq i * i) \to$$
$$\quad [y := s(y + s(s(0)) * i);\ i := s(i)]\, inv$$

$$(Nat \land i < x \land y \doteq i * i) \to$$
$$\quad [y := s(y + s(s(0)) * i)][i := s(i)]\, inv$$

Aufgabe 2 c

Regel für Zuweisung

$$\frac{(\psi(x_{alt}) \land x \doteq t(x_{alt})) \to \rho(x)}{\psi(x) \to [x := t(x)]\rho(x)}$$

Zu zeigen

$$(Nat \land i < x \land y \doteq i * i) \to \\ [y := s(y + s(s(0)) * i)][i := s(i)]\, inv$$

$$(Nat \land i < x \land y_{alt} \doteq i * i \land y \doteq s(y_{alt} + s(s(0)) * i)) \to \\ [i := s(i)]\, inv$$

$$(Nat \land i < x \land y \doteq s(i * i + s(s(0)) * i)) \to \\ [i := s(i)]\, inv$$

Aufgabe 2 c

Regel für Zuweisung

$$\frac{(\psi(x_{alt}) \land x \doteq t(x_{alt})) \to \rho(x)}{\psi(x) \to [x := t(x)]\rho(x)}$$

Zu zeigen

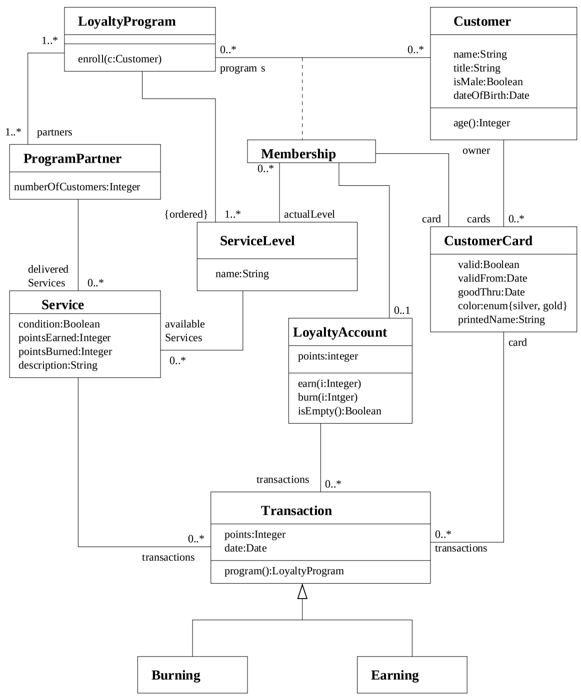
$$(Nat \land i < x \land y \doteq s(i * i + s(s(0)) * i)) \to [i := s(i)]\, inv$$

$$(Nat \land i_{alt} < x \land y \doteq s(i_{alt} * i_{alt} + s(s(0)) * i_{alt}) \land \\ i \doteq s(i_{alt})) \to inv$$

$$(Nat \land s(i_{alt}) \leq x \land y \doteq s(i_{alt}) * s(i_{alt}) \land i \doteq s(i_{alt})) \to inv$$

$$(Nat \land \underbrace{i \leq x \land y \doteq i * i}_{= inv} \land i \doteq s(i_{alt})) \to inv$$

Aufgabe 3



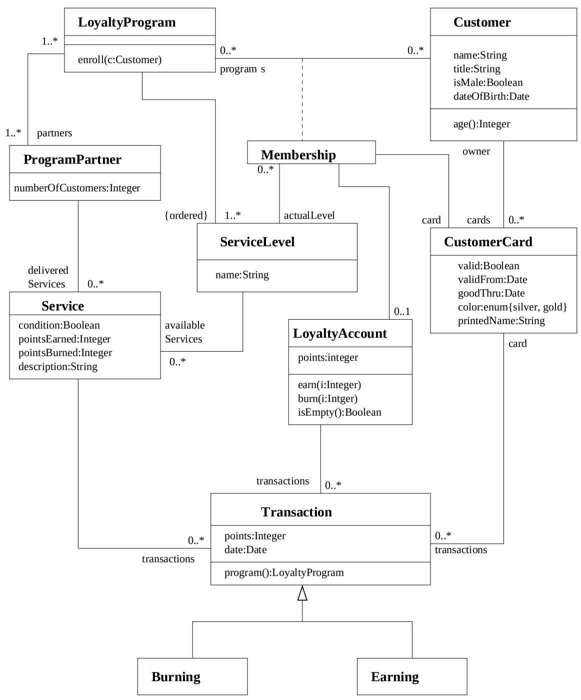
Teilaufgabe a (i)

`LoyaltyProgram`

`self.customer->size < 10000`

Die Zahl der an einem bestimmten Loyalty-Programm beteiligten Kunden ist kleiner als 10000.

Aufgabe 3

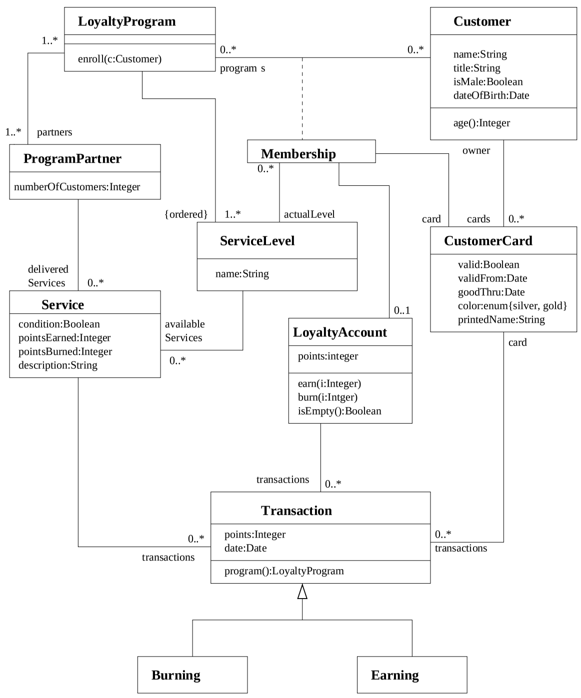


Teilaufgabe a (ii)

```
LoyaltyAccount
transaction
  ->collect( points )
  ->exists( p:Integer |
            p=500 )
```

Mit jedem Loyalty-Account ist eine Transaktion assoziiert, die einen Punktewert von (genau) 500 hat.

Aufgabe 3

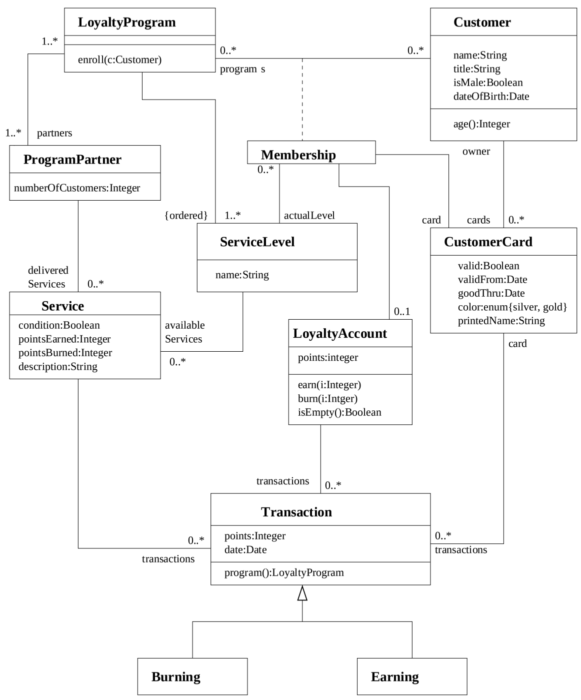


Teilaufgabe a (iii)

```
LoyaltyProgram::
    cancel(c : Customer)
post : customer =
       customer@pre
       ->excluding(c)
```

Nachdem ein Kunde aus dem Programm gelöscht wurde, ist die Menge der Kunden gleich der Menge der Kunden vor dem Löschen ohne den gelöschten Kunden.

Aufgabe 3

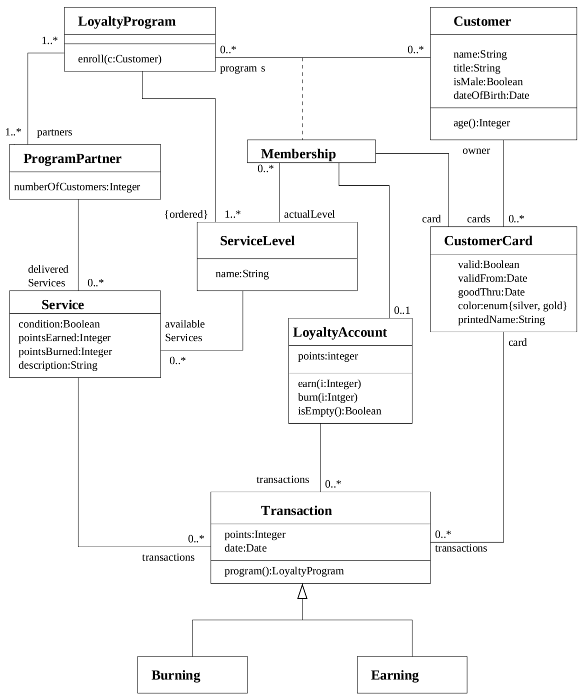


Teilaufgabe b (i)

Der mit einer Karte assoziierte Kunde ist mit dem Kunden identisch, der mit derjenigen Mitgliedschaft assoziiert ist, welche mit der Karte assoziiert ist.

```
CustomerCard
membership.customer = owner
```

Aufgabe 3

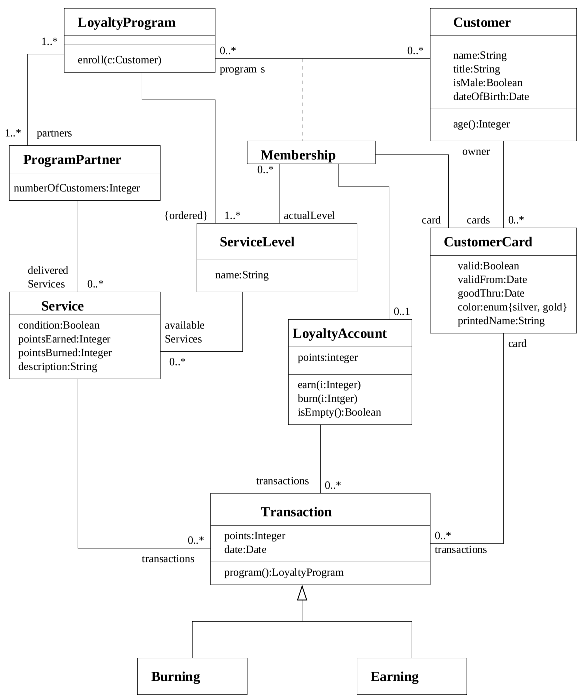


Teilaufgabe b (ii)

Verschiedene Kunden eines Loyalty-Programms haben verschiedene Karten.

```
LoyaltyProgram
customer->
  forAll(c1, c2 | c1 <> c2
      implies
    c1.cards->forAll(crd1|
                        c2.cards-
>forAll(crd2|
            crd1 <> crd2)))
```


Aufgabe 3

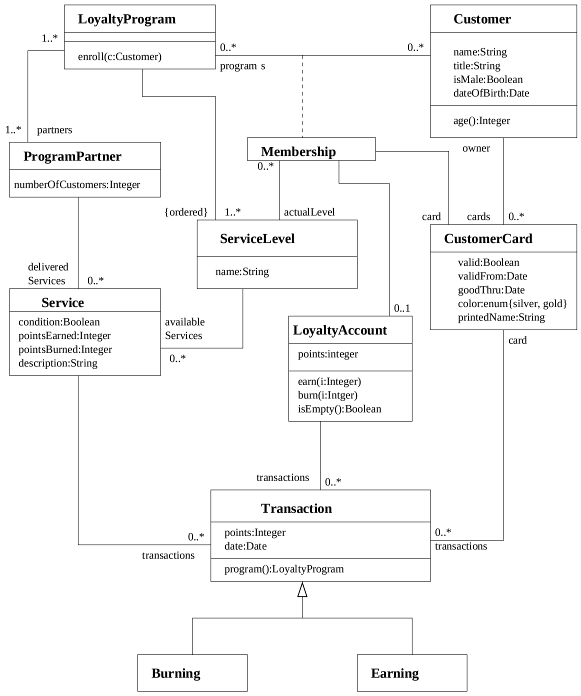


Teilaufgabe b (iii)

Wenn es mehr als 0 Punkte in einem Loyalty-Account gibt, dann muß eine Transaktion mit dem Account assoziiert sein, deren Punktwert größer als 0 ist.

```
LoyaltyAccount
points > 0 implies
   transactions->
   exists(t | t.points > 0)
```

Aufgabe 3



Teilaufgabe b (iv)

Die Summe der Punktwerte derjenigen mit einem Loyalty-Account assoziierten Transaktionen, die einen Punktwert kleiner als 10 haben, ist kleiner als 500.

```
LoyaltyAccount
transactions
  ->select(points < 10)
  ->iterate(
     t :  Transaction;
     result : Integer = 0 |
     result + points)
  < 500
```