Zur Übungsserie 4: Schlaufen

1 Achterbahn fahren in Eiffel

Schlaufen sind Programmblöcke, welche wiederholt ausgeführt werden – von nie, über einmal bis unendlich. Eine solche Schlaufe sieht in Eiffel so aus:

```
-- vorhergehender Teil

from
        -- Initialisierung
until
        -- Endbedingung
loop
        -- zu repetitierender Teil
end

-- nachfolgender Teil
```

} eine Schleife im Programm

Bevor wir die Schlaufe starten, wollen wir noch einige Dinge vornehmen – die Schlaufe wird vorbereitet bzw. initialisiert.

Wir wollen praktisch immer vermeiden, dass unsere Schlaufe unendlich viele Male ausgeführt wird. Deshalb benötigen wir eine Endbedingung. Wenn diese erfüllt ist, wird der Loop nicht mehr ausgeführt, und das Programm fährt mit dem nachfolgenden Teil weiter.

Der zu wiederholende Teil steht zwischen den Schlüsselwörter (*keywords*) **loop** und **end**. Dieser Teil kann Berechnungen, Feature calls oder wiederum weitere Schlaufen enthalten.

Ein möglicher Ablauf wäre bei folgendem Beispiel:

```
-- davor
from
        i := 0
until
        i > 2
loop
        i := i + 1        -- (X)
end
-- Schluss
```

Zuerst wird i auf null gesetzt, danach wird überprüft, ob i > 2 – trifft nicht zu, also den zu repetitierenden Teil ein erstes Mal ausführen – i wird der Wert 0 + 1, also 2 zugewiesen – danach wieder schauen, ob i > 2 – immer noch nicht, also ein zweites Mal ausführen: i wird zu 2. Danach wird wieder getestet, ob i > 2. Nein? Also, nochmals ausführen: i wird zu 3. Ein erneuter Test, ob i > 2 zutrifft; jawohl, also springen wir an den Schluss.

Wir haben also dreimal die (X) Zeile ausgeführt und dadurch am Schluss i den Wert 3 gegeben. Die Schlaufe hat sich also nur endlich viele Male wiederholt, weil i, das zählende Element sich allmählich dahin bewegt hat, dass die Endbedingung wahr wird. Wir brauchen also immer ein solches Element, welches unsere Schlaufe begrenzt.

2 Die Stadt der Liebe

Klassen vererben, oder nur Objekte von Klassen verwenden?

Jede Klasse hat Features, welche wir auf die Objekte einer Klasse anwenden können.

```
class TRANSISTORRADIO inherits RADIO
feature
        eingelegte_Batterie : BATTERIE
end
```

Wir haben also eine Klasse ***TRANSISTORRADIO***, die alle Fähigkeiten eines ***RADIO***s hat und in welchen man zusätzlich noch Batterien einlegen kann.

Das heisst also, wir haben ***TRANSISTORRADIO*** von ***RADIO*** vererbt, weil unser ***TRANSISTORRADIO*** genau das Gleiche und noch mehr kann.

Der Radio benötigt Strom, welcher er von einem Objekt der Klasse ***BATTERIE*** bekommt. Zu diesem Zweck erweitern wir den Radio mit einer Batterie – das heisst mit einem Feature, welches auf ein Objekt der Klasse ***BATTERIE*** zeigt.

Ein schlechtes Beispiel wäre das hier:

```
class KOMISCHES_TRANSISTORRADIO
inherits
        RADIO
        BATTERIE
end
```

Weil ja ein Transistorradio Strom braucht, geben wir ihm die Fähigkeit, selbst Strom zu produzieren. Oder haben wir jetzt eine Batterie, die zusätzlich auch noch Musik machen kann?!

Wie ihr seht, macht das keinen Sinn. Deshalb immer überlegen, wie ihr das macht – nur selten vererbt man! Meistens erstellt man ein Feature, welches auf ein Objekt einer Klasse zeigt.

Eine Besichtigungsfahrt

Sehenswürdigkeiten werden im TRAFFIC als kleine Miniaturbilder dargestellt. Irgendjemand bzw. wir Assistenten wollen nun, dass diese durch simple Kreise erstetzt werden. Wir haben doch keine Zeit – deshalb müsst ihr das jetzt implementieren.

Dazu müsst ihr die Klasse ***LANDMARK*** verändern:

- fügt ein Feature ***icon_on*** vom Typ ***BOOLEAN*** hinzu und
- schreibt ein Feature ***set_icon_on***, welches ***icon_on*** verändert.

Als zweites müsst ihr eine If-Anweisung programmieren:

```
if ein_boolscher_Ausdruck then
        -- etwas
else
        -- etwas anderes
end
```

Dieses If macht etwas, wenn ein bool'scher Ausdruck wahr ist, und etwas anderes, wenn er nicht wahr ist.

Platziert dieses If in der Klasse ***LANDMARK_DISPLAYER_IMP*** im Feature ***initialize***.

```
initialize is
                -- Initialize graphical representation.
        do
                if not element.has_graphical_representation
                then
                        Precursor
                else
                        basic_representation.extend (icon)
                        highlighted_representation.extend (icon)
                        marked_representation.extend(circle
                        (radius * Mark_factor, Spotlight_color,
                        False))
                end
        end
```

Der grau unterlegte Teil soll nun nur dann ausgeführt werden, wenn das Element ***icon_on*** wahr hat.

Als Drittes müsst ihr in einem neuen Cluster eine ganz neue Klasse schreiben. Sie soll:

- vererbt sein von EXERCISE und SHARED_MAP

- ein Feature ***start*** haben; dazu müsst ihr *irgendwo* Folgendes dazuschreiben (wenn ihr bei anderen Klassen schaut, findet ihr raus, wo genau ihr das dazuschreiben müsst):

```
redefine
        start
end
```

Danach füllt ihr das Feature start mit einer Schlaufe füllen, die ganz ähnlich wie in der Aufgabe 1 aussieht. Mit ***map***.*blablabla* könnt ihr auf alles Nötige zugreifen...

Um das Ganze im System irgendwie anzuknüpfen, müsst ihr das der Klasse ***EXERCISE_REGISTRATION*** sagen. Erstellt dazu im einzigen Feature ein neues Objekt des Typs ***SWITCH_REPRESENTATION*** und gebt ihm den Namen "Landmarks". Es hat ja schon eine Zeile, die das mit ***TRY_CONTRACTS*** genau vormacht...

3 Wem nach all diesen Loopings noch immer nicht schwindlig ist...

Wie schon erwähnt, können auch Schleifen in Schleifen in Schleifen in Schleifen geschrieben werden. Wie schon erwähnt, können auch Schleifen in Schleifen in Schleifen in Schleifen geschrieben werden. Wie schon erwähnt, können auch Schleifen in Schleifen in Schleifen in Schleifen geschrieben werden.

Dies sollt ihr jetzt hier tun mit einer neuen Klasse METRO_STATISTICS, welche über alle Metrolinien iteriert und deren Länge und Anzahl Stationen errechnet.

An die Metrolinien kommt man mit ***map.city.metro_lines***. Auf was diese Entity zeigt, müsst ihr selbst noch herausfinden, indem ihr schaut, was ***map*** für ein Typ hat und welchen Typ das Feature ***city*** in der Klasse des Typs von ***map*** hat usw.

Die erste Metrolinie bekommt ihr mit ***map.city.metro_lines.start***. Die jeweils Nächste mit ***map.city.metro_lines.forth***.

An die Elemente der aktuellen Metrolinie kommt ihr über

map.city.metro_lines.item_for_iteration.segment(i)

Die Übung gebt ihr bitte am Mittwoch, dem 26. November 2003 ab.