Informatiker sind kreativ!

Die Qual der Wahl

Ihr habt nun zwei Möglichkeiten:

- Option A: Ihr erweitert das FLAT_HUNT-Programm mit fünf Funktionen. Die Gehirne sollen besser werden. Zudem gibt es neu auch Taxis. Strassen werden neu aufgebaut und sind zeitweilig wegen Umbau gesperrt. Und ihr sollt die Bedienerfreundlichkeit verbessern.
- Option B: Was ihr schon immer tun wolltet... Hier könnt ihr komplett neue Programme schreiben, ein bestehendes Programm erweitern, oder irgendein tolles Ding dem FLAT_HUNT-Programm hinzufügen. Das Ganze muss natürlich in Eiffel geschrieben sein. Ihr müsst die Aufgabe einfach von mir genehmigen lassen.

Die Option A eignet sich für diejenigen von euch, die nicht sehr erfahren sind im Programmieren, oder einfach, wenn ihr keine wirklich gute Idee habt.

In diesem Paper nehmen wir an, wir erstellen einen Kulturkalender, in welchem man unterhaltsame Dinge wie Filmvorführungen, Theaterstücke etc. eingeben kann und diese dann angezeigt werden, wenn man danach sucht.

Schritt für Schritt – anhand eines Beispiels

Bevor ihr anfangt mit dem Programmieren müsst ihr euch einmal Gedanken über die Eigenschaften von Unterhaltung machen:

a. An welchen Orten gibt es Unterhaltung (Kino, Theater, Sportanlässe, Festivals, Disco)?

b. Wie tauchen die verschiedenen Ereignisse auf (einmalige Vorstellung, täglich, wöchtenlich wiederholt)?

c. Welche Beziehungen gibt es zwischen den einzelnen Ereignissen (gleiche Kinofilme werden in mehreren Kinos gezeigt)?

Mathieu Macier: Cube noir sur socle blanc, 2002. Schwarzer Kubus, weißer Sockel aus MDF-Platten, 100 x 90 cm

Wenn ihr eine grobe Übersicht habt, dann könnt ihr euch mit Stift und Papier überlegen, wie ihr das in ein objekt-orientiertes System übertragt:

d. Falls ihr ein anderes Programm/Projekt erweitert, studiert den Developer Guide und seht, welche Ausgangslage ihr bezüglich Klassen und Klassenrelationen habt.

e. Welche Klassen braucht ihr (Schauplatz, Anlass, Thema, Zeit, Film, Theaterstück)?

f. Welche Eigenschaften haben die Klassen (Titel, Zeit)? Welche Operationen können auf die Objekte ausgeführt werden (löschen, Titel ändern)?

g. Wie sind sie miteinander verbunden (Ort – Kino – Film – Zeit, Stadion – Fussballmannschaft – Zeit)?

Versucht dabei möglichst nahe die Realität abzubilden. Das heisst, erfindet genau so viele neue Klassen, die aber sehr logisch miteinander verknüpft werden können: nicht zu viel und nicht zu wenig.

Stellt sicher, dass

h. das Ganze immer noch einfach, übersichtlich und logisch ist,

i. ihr damit sämtliche Anforderungen aus Punkt 2 der Aufgabenstellung erfüllen und

j. die Aufgaben von Punkt 3.3 erfüllen könnt.

Erst wenn ihr alles fertig modelliert habt, beginnt ihr mit dem Computer zu arbeiten:

k. schreibt zuerst sämtliche neuen Klassen, mit all den Features, aber noch ganz ohne den do-Teil.

l. Danach könnt Ihr euch überlegen, wie ihr die einzelnen Features gestalten wollt, damit sie das Gewünschte tun.

Nach zahlreichen Kompilierversuchen und Tests könnt ihr dann mal versuchen, die Tests von Punkt 3.3 auszuprobieren.

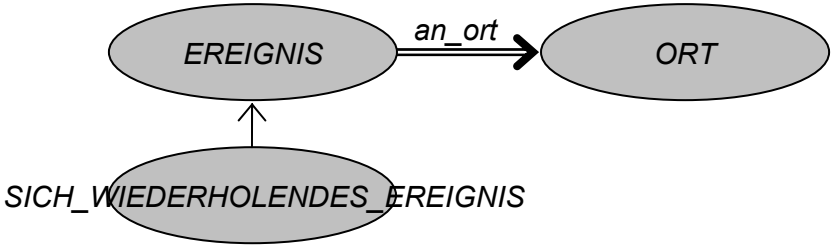
Das Schöpferische an der Informatik

Die Informatik ist eine äusserst kreative Disziplin: in der ersten Phase der Arbeit eines Informatikers kommt immer der Entwurf: man nimmt Papier und Stift zur Hand und überlegt sich, was man zu tun hat, und wie man das mit den Konzepten der Informatik umsetzen könnte.

Und gerade dieser Schritt macht den guten Informatiker aus. Es geht nicht (nur) darum, Nächte lang hinter dem Bildschirm enorme Mengen an Code zu generieren. Denn wenn man im ersten Schritt der Konzeption und des Modellieren gekonnt vorgeht, kann man sich wahrscheinlich die meisten Nächte des Programmierens sparen.

Aber wie macht man das: man muss, nachdem die Antworten zu (a) bis (c) gefunden wurden, überlegen, wie man das Konzept der Klassen gekonnt ausnützt.

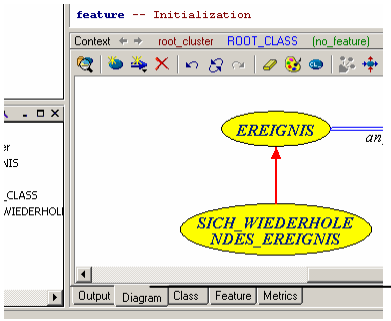
- Sinnigerweise erstellt man zu jeder Art von Dingen eine Klasse. Beispielsweise würde es sich anbieten, eine Klasse *EREIGNIS* zu erstellen. Eine Weitere wäre vielleicht *ORT*.
- Nun versucht man die Objekte dieser Klassen in Relation zu bringen. Beispielsweise ereignet sich ein Ereignis meistens an einem bestimmten Ort. Es muss also in der Klasse *EREIGNIS* ein Feld *an_ort : ORT* angeboten werden.
- Nun gibt es auch noch spezielle Ereignisse; zum Beispiel jene, welche sich mehrmals ereignen: *SICH_WIEDERHOLENDES_EREIGNIS*.
- In welcher Relation steht diese Klasse? Ja, es ist quasi eine Spezialisierung oder eine Untergruppe von Ereignissen. Es erweitert also die Fähigkeiten der Klasse EREIGNIS, womit wir nun wissen, dass *SICH_WIEDERHOLENDES_EREIGNIS inherits EREIGNIS*.
- Daraus würde sich also folgendes, **noch unvollständiges** BON-Diagramm ergeben:



In den Ellipsen stellen Klassen dar, durchgezogene Pfeile sind Vererbungsrelationen (*inherits*) und doppelte Pfeile zeigen Client-Server-Beziehungen an. Die Worte neben den Linien sind die dazugehörigen Felder. In der obigen Darstellung hat jedes Objekt des Typs *EREIGNIS* die Felder *an_ort*.

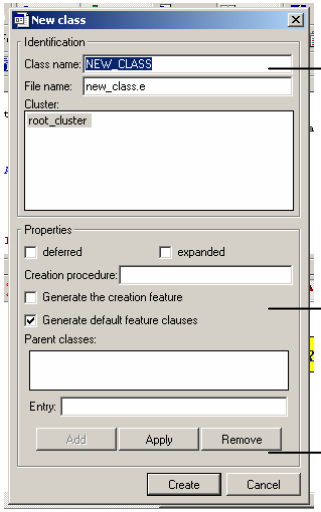
Tu dois peindre ~~bien~~ BON!

Im Eiffel Studio kann man die Klassen und die Relationen dazu einfach in der BON-Notation zeichnen (Wichtig: nur in der Enterprise Edition, welche im Computerraum installiert ist!). Um in diesen Zeichenbereich zu gelangen, braucht ihr bloss nachdem ihr ein neues Projekt erstellt habt, im Kontextfenster auf Diagram zu klicken und dann drauflos zu zeichnen. Eine Anleitung dazu findet ihr in der Eiffelhilfe (siehe Abschnitt *Referenzen*).



Reiter 'Diagram' im Fenster 'Context'

Klassen erstellen

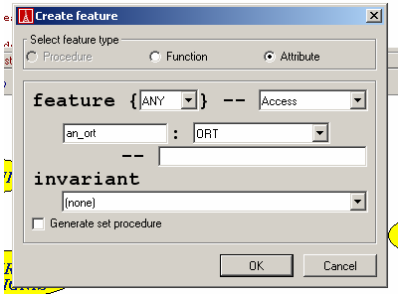


wie soll die Klasse heissen

von welchen Klassen wird vererbt

Klassen erstellt man mit der Ellipsenschaltfläche, worauf dann ein Fenster erscheint, wo man den Namen und die Klassen, von denen vererbt wird, eingegeben werden können.

Klassen verknüpfen oder Features erstellen



Klassen können in einer Client-Server-Beziehung stehen. Diese können mit der Pfeilsymbol-Schaltfläche erstellt werden. Danach erscheint das oben stehende Fenster, wo man dann das Feature genauer definieren kann.

Kleine Werkzeugkiste

Wie der Turm liest

Falls ihr in eurem Programm auf Dateien zugreifen wollt, dann müsst ihr die PLAINT_TEXT_FILE Klasse benützen:

```
local
        f : PLAIN_TEXT_FILE
do
        from
                create f.make_open_read( "c:\datei.txt" )
        until
                not f.is_readable or else f.after
        loop
                f.read_line
                io.put_string( f.last_string )
                io.put_new_line
        end
end
```

Ihr öffnet also eine Datei mit dem Konstruktor *make_open_read* für den Lesevorgang. Danach lest ihr jeweils eine Zeile ein mit dem Feature *read_line* und das Resultat erhält ihr in *last_string*. Wenn man am Ende der Datei angelangt ist, wird *is_readable* falsch oder *after* wahr.

Schaut euch zusätzlich noch den Klassentext von *FILE* und *PLAIN_TEXT_FILE* an, falls ihr zum Beispiel zeichenweise oder wortweise einlesen wollt.

Durch die Datei hindurch

Die Daten könnt ihr beispielsweise in folgendem Format bereitstellen, um ganz einfach einlesen zu können:

```
#
#Movie Events
#
RecurringEvent|Movie|Finding Nemo|Kino
ABC|14:00|16:00|2004/01/01|2004/01/14|daily
RecurringEvent|Movie|Finding Nemo|Kino
ABC|16:15|18:15|2004/01/01|2004/01/14|daily
RecurringEvent|Movie|Finding Nemo|Kino
ABC|18:45|20:45|2004/01/01|2004/01/14|daily
```

Dann könnt ihr jeweils jede Zeile mit einem *TOKENIZER* in Stücke schneiden und dann die entsprechenden Ereignisse kreieren in der Art *create {MOVIE_EVENT}.make(Was, Wann, Wo)*.

Referenzen

PLAIN_TEXT_FILE Flat contracts – `http://www.gehr.org/technical/source/Documentation/base/kernel/plain_text_file_flatshort.html`

Skulptur Biennale Münsterland 2003 – `http://www.skulptur-biennale-muensterland.de/d_menu.htm`

Schritt für Schritt Anleitung wie ein BON-Diagramm erstellt wird – *Menu Help > How to's > Designing a project > ...*