
Informatik I

Blatt 12 („Testatblatt“, Fassung vom 5.2.2001, 9:30)

Abgabe: 8.2.2001

Die Lösung für dieses Testatblatt wird in der Woche nach der Abgabe persönlich der Tutorin bzw. dem Tutor in der Woche nach dem Abgabetermin vorgeführt. Vereinbare dazu einen Termin mit Deiner Tutorin bzw. Deinem Tutor. Für den Erhalt des Scheines ist es notwendig, das Blatt zur Zufriedenheit der Tutorin bzw. des Tutors zu bearbeiten.

Bei Schwierigkeiten bei der Lösung der Aufgaben stehen Euch die üblichen Sprechstunden zur Verfügung:

- Montags 15–17, Mittwochs 13–15 in C2S1 bei den Tutoren
- Mittwochs 8–10 bei Mike Sperber in Raum 113, Sand 13

Bearbeite das Blatt *rechtzeitig*, damit wir Dir gegebenenfalls noch vor dem Abgabetermin helfen können!

Zu diesem Übungsblatt gehört eine Datei mit Code — `uni.scm`, die auf der Homepage der Vorlesung zu finden ist.

1 Simulationsspiele

In Simulationsspielen kontrolliert ein Spieler imaginäre Personen, die sich in einer imaginären Welt zusammen mit weiteren imaginären Personen befinden. Der Spieler kann seine Person zwischen Orten seiner Welt bewegen und dort Tätigkeiten verrichten lassen. Der Computer simuliert zulässige Spielzüge und weist unzulässige zurück. So ist es beispielsweise (zumindest ohne spezielle Zauberkräfte) nicht zulässig, sich zwischen Orten zu bewegen, die nicht durch einen Weg miteinander verbunden sind.

Die Objekte der Spielwelt — zunächst Dinge, Orte und Personen — werden durch Objekte im Sinne der objektorientierten Programmierung repräsentiert. Benutzt werden die Mittel aus der entsprechenden Vorlesung.

Es gibt drei Konstruktoren für Objekte:

```
(make-place name)
(make-thing name birthplace)
(make-person name birthplace laziness)
```

Jedes Objekt hat einen Namen, repräsentiert durch ein Symbol. Dinge und Personen bekommen zusätzlich noch einen Ort *birthplace* — einen *Geburtsort* gewissermaßen — an dem sie sich zu Anfang des Spiels befinden. Personen haben außerdem einen *Faulheitsfaktor laziness* der bestimmt, wie oft sie sich bewegen. (Dazu mehr später.)

Bei den Objekten der Spielwelt hat ein `baz`-Objekt (also ein Objekt der Klasse `baz`, zurückgegeben von `make-baz`) immer eine Methode `baz?`, welche `#t` liefert. Die Prozedur `is-a` kann dann testen, ob ein Objekt einer bestimmten Klasse zugehörig ist, und zwar via

```
(is-a object 'baz?)
```

Die Konstruktoren erzeugen lediglich Objekte; nach der Erzeugung sind sie noch losgelöst von der Spielwelt. Deswegen gibt es zusätzliche Prozeduren, die nicht nur Objekte erzeugen und zurückgeben sondern auch noch zusätzliche Aktionen vornehmen, um diese in der Spielwelt installieren:

```
(make&install-thing name birthplace)
(make&install-person name birthplace laziness)
```

Die Prozedur `make&install-person` kann benutzt werden, um zwei imaginäre Personen, `mike` und `herb`, in der Spielwelt zu plazieren. Hier zum Beispiel der Code aus der vordefinierten Spielwelt:

```
(define herb-office (make-place 'herb-office))
(define s1 (make-place 's1))

(define herb (make&install-person 'herb herb-office 3))
(define joe-student (make&install-person 'joe-student s1 2))
```

Die Symbole `herb-office`, `s1`, `herb` und `joe-student`, die als Parameter *name* übergeben werden, werden vom System in Meldungen benutzt.

`Herb-office`, `s1`, `herb` und `joe-student` sind allesamt Objekte im Sinne des objektorientierten Programmiersystems aus der Vorlesung.

2 Die Unicolorsität Tübingen

Tatsächlich ist für diese Übung schon eine Spielwelt vordefiniert, in der schon einige Objekte vorinstalliert sind — u.a. `herb`, `joe-student` und Herbs Büro.

Die Spielwelt repräsentiert einige kleine, aber wichtige Teile Tübingens, die zusammen die *Unicolorsität Tübingen* bilden. So gehören Teile des *Wilhelm-Schickeria-Instituts* am Sand sowie der *Gesternstelle* zur Spielwelt. Die Struktur läßt sich durch Spielen des Spiels ergründen.

An der Unicolorsität sind z.B. folgende Spielzüge möglich:

```
> (send herb 'look-around)
At herb-office : herb says -- I see nothing
()
> (send (send herb 'place) 'exits)
(west)
> (send herb 'go 'west)
herb moves from herb-office to wsi-first-floor-hallway
#t
> (send herb 'go 'down)
herb moves from wsi-first-floor-hallway to wsi-ground-floor-hallway
#t
> (send herb 'go 'north)
herb moves from wsi-ground-floor-hallway to sand-bus-stop
#t
> (send herb 'go 'west)
herb moves from sand-bus-stop to bus-sand->gesternstelle
#t
> (send herb 'go 'east)
herb moves from bus-sand->gesternstelle to gesternstelle-bus-stop
#t
> (send herb 'look-around)
At gesternstelle-bus-stop : herb says -- I see nothing
```

```

()
> (send (send herb 'place) 'exits)
(north west)
> (send herb 'go 'north)
herb moves from gesternstelle-bus-stop to mensa
#t
> (send herb 'go 'north)
herb moves from mensa to n3
#t
> (send joe-student 'go 'south)
joe-student moves from s1 to gesternstelle-basement
#t
> (send joe-student 'go 'south)
You cannot go south from gesternstelle-basement
#f
> (send joe-student 'go 'up)
joe-student moves from gesternstelle-basement to n3
At n3 : joe-student says -- Hi herb
#t

```

Die Unicolorsität soll noch etwas erweitert werden. Jemand habe einen Laptop in N3 liegengelassen:

```

(define laptop (make&install-thing 'laptop n3))
> (send herb 'look-around)
At n3 : herb says -- I see laptop joe-student
(laptop joe-student)

```

Herb will den Laptop ins Fundbüro bringen:

```

> (send herb 'take laptop)
At n3 : herb says -- I take laptop
#t
> (send herb 'go 'down)
herb moves from n3 to gesternstelle-basement
#t

```

Es ist allerdings joe-students Laptop, und er hält herb für den Dieb:

```

> (send joe-student 'go 'down)
joe-student moves from n3 to gesternstelle-basement
At gesternstelle-basement : joe-student says -- Hi herb
#t
> (send joe-student 'take laptop)
At gesternstelle-basement : herb says -- I lose laptop
At gesternstelle-basement : herb says -- Yaaaah! I am upset!
At gesternstelle-basement : joe-student says -- I take laptop
#t

```

Joe-student will zu mike, um ihm eine Frage zum aktuellen Übungsblatt zu stellen.

```

> (send joe-student 'go 'up)
joe-student moves from gesternstelle-basement to n3
#t
> (send joe-student 'go 'south)
joe-student moves from n3 to mensa
#t

```

```

> (send joe-student 'go 'south)
joe-student moves from mensa to gesternstelle-bus-stop
#t
> (send joe-student 'go 'west)
joe-student moves from gesternstelle-bus-stop to bus-gesternstelle->sand
#t
> (send joe-student 'go 'east)
joe-student moves from bus-gesternstelle->sand to sand-bus-stop
#t
> (send joe-student 'go 'south)
joe-student moves from sand-bus-stop to wsi-ground-floor-hallway
#t

```

Joe-student trifft mike tatsächlich im Erdgeschoß des WSI:

```

> (send mike 'go 'north)
mike moves from mike-office to wsi-first-floor-hallway
#t
> (send mike 'go 'down)
mike moves from wsi-first-floor-hallway to wsi-ground-floor-hallway
At wsi-ground-floor-hallway : mike says -- Hi joe-student
#t

```

Leider ist mike ein *Assi*, ein Wesen, das beim Empfangen einer *move*-Nachricht garstig wird:

```

> (send mike 'move)
At wsi-ground-floor-hallway : mike says -- Growl... I'm going to eat you, joe-student
At wsi-ground-floor-hallway : joe-student says -- I lose laptop
At wsi-ground-floor-hallway : joe-student says --
    Dulce et decorum est
    pro computatore mori!
joe-student moves from wsi-ground-floor-hallway to heaven
At wsi-ground-floor-hallway : mike says -- Chomp chomp. joe-student tastes yummy!
#t

```

Die *move*-Methode einer Figur veranlaßt diese zu „eigenständigem“ Handeln, also einem Spielzug, der vom Computer ausgewählt wird. Damit dies in geordneten Bahnen automatisch geschehen kann, legt das Spiel eine Liste aller Figuren an, die vom Computer bewegt werden können. Der Verlauf der Zeit wird durch Aufruf einer Prozedur *clock* simuliert, die *move*-Nachrichten an alle Personen in der Liste schickt. In einem realistischen Spiel wird also nach jedem Spielzug (*clock*) ausgewertet. Für den Aufruf von *clock* ist jedoch derzeit der Benutzer selbst verantwortlich, es geschieht nicht automatisch.

Die *move*-Methode veranlaßt dabei nicht immer die zugehörige Figur zu einer Handlung: Meist ist die Figur zu faul, um etwas zu tun. Erst nachdem etwas Zeit vergangen ist, macht sich genügend Langeweile breit, um die Person zur Handlung zu treiben. Darum ist *laziness*, der dritte Parameter für *make-person* das Uhrenintervall, den eine Person abwartet, bis sie etwas tut.

3 Aufwärmübungen

Lies den Code in *uni.scm*! Details sind dabei nicht so wichtig wie ein Überblick über die Struktur des Systems und der Unicolorsität. Als Hilfestellung zeigt Abbildung 1 das Umgebungsdiagramm für den Ausschnitt der Spielewelt mit *herb* und *herb-office*.

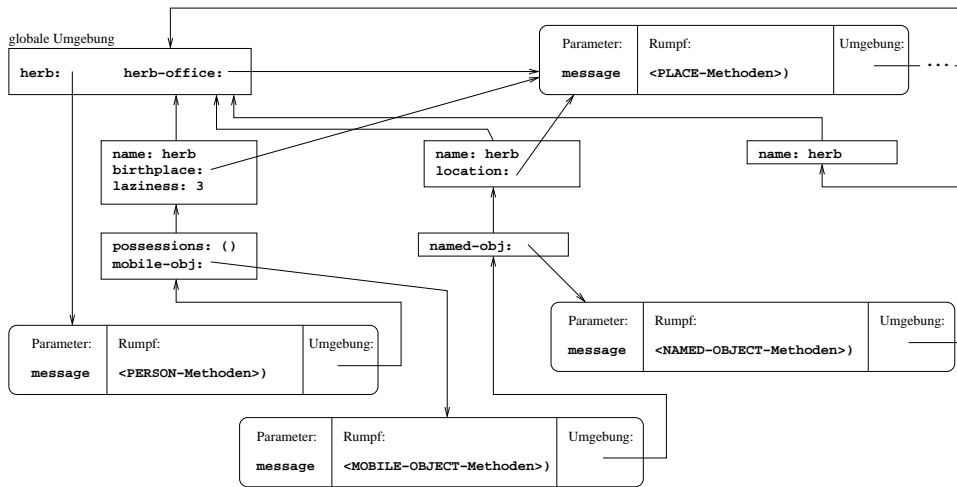


Abbildung 1: Herb in seinem Büro

Übung 1 Zeichne ein Diagramm, in dem die Vererbungsbeziehungen zwischen allen Klassen des Spiels zu sehen sind. Welche Methoden gibt es?

Abzugeben: Vererbungsdiagramm, Methodenlisten für alle Klassen

Übung 2 Zeichne eine Landkarte der Orte, die in der Unicolorsität definiert sind, zusammen mit den Verbindungen, die zwischen den Orten existieren.

Abzugeben: Landkarte

Übung 3 Angenommen, die folgenden Ausdrücke werden ausgewertet:

```
(define maultaschen (make-thing 'maultaschen mensa))
(send maultaschen 'set-owner joe-student)
```

Irgendwann während der Auswertung des zweiten Ausdrucks wird ein Ausdruck

```
(set! owner new-owner)
```

ausgewertet. Finde diesen Ausdruck im Code, verfolge die Auswertung bis zum Aufruf und zeichne ein Umgebungsdiagramm, in dem die Struktur von `maultaschen` zum Zeitpunkt der Auswertung des Ausdrucks zu sehen ist. Die internen Details von `joe-student` und `mensa` sind dabei nicht wichtig.

Abzugeben: Umgebungsdiagramm

Übung 4 Angenommen, zu den `maultaschen` kommt noch:

```
(define kaesespaetzle (make-thing 'maultaschen mensa))
```

hinzu. Sind `maultaschen` und `kaesespaetzle` im Sinne von `eq?` dasselbe Objekt? Warum bzw. warum nicht? Angenommen, die `maultaschen` und `kaesespaetzle` sind mit `make&install-thing` installiert und `joe-student` bewegt sich in die Mensa — welche Meldung erscheint, wenn sich `joe-student` mit `look-around` umsieht?

Übung 5 Finde die Code-Stellen, an denen die `install`-Methode von `mobile-object` und `person` steht. Die Fassung in `person` stellt die Figur auf die Liste der zu animierenden Objekte `*clock-list*` und wendet dann die Fassung von `install` für `mobile-object` auf `self` an, um `birthplace` davon in Kenntnis zu setzen, daß `self` glaubt, an diesem Ort zu sein:

```
((get-method mobile-obj 'install) self)
```

Damit sind `self` und `birthplace` synchronisiert, was ihr Wissen über den Aufenthaltsort von `self` betrifft.

Rinus Öchel meint, daß es einfacher wäre, wenn im Code von `make-person` stattdessen einfach

```
(send mobile-obj 'install)
```

stünde. Crescentia Onyx meint, daß dies ein Fehler wäre. Crescentia sagt folgendes:

„Wenn Du das tust und später `make&install-person` für `herb` aufrufst, und `herb` an einen anderen Ort geht, wird er an zwei Orten auf einmal sein!“

Was meint Crescentia? Was genau geht schief? Zeichne ein Umgebungsdiagramm, welches die Antwort auf die Frage illustriert.

Abzugeben: Umgebungsdiagramm

4 Experimente mit dem System

Übung 6 Lade das System und rufe wiederholt die `move`-Methode von `herb` und `joe-student` auf. Welche der beiden Figuren ist unruhiger? Warum? Wie oft bewegen sich beide gleichzeitig?

Übung 7 Erzeuge und installiere eine neue Person (Du selbst sozusagen), die einen hohen Faulheitsfaktor hat, um unfreiwillige Bewegungen durch `clock` zu vermeiden. Platziere Dich zunächst in die Mensa. Erzeuge und installiere außerdem ein Objekt `late-homework`, das sich auch in der Mensa befindet. Hebe es auf, finde `herb` und bring ihn dazu, `late-homework` aufzuheben. Läßt sich das Ziel erreichen, ohne daß Du am Ende sauer („upset“) bist? Die Lösung der Aufgabe sollte eine Folge von Definitionen und Handlungen sein. Füge nach jedem Spielzug einen Aufruf von `clock` ein, damit die Sache interessanter wird. Paß auf `mike` auf!

Hinweis: Es könnte sinnvoll sein, den Code für die Spielzüge ans Ende von `uni.scm` zu hängen, damit er wiederholt ausgeführt und korrigiert werden kann.

Abzugeben: Code für die neue Person

5 Studierendungsberechtigungsausweise

Das Studentensekretariat der Unicolorsität Tübingen hat das Wilhelm-Schickleria-Institut gebeten, das Studierendungsberechtigungsausweis-System der Unicolorsität zu erweitern.

Ein Studierendungsberechtigungsausweis wird durch eine neue Art Objekt, eine `sb-card` repräsentiert. Es ist ein Ding, hat aber noch eine lokale Identifikationsnummer `id`, welche die Inhaberin bzw. den Inhaber identifiziert. Studierendungsberechtigungsausweise haben eine Methode `id`, welche diese Identifikationsnummer zurückgibt.

Hier ist der entsprechende Code:

```

(define make&install-sb-card
  (lambda (name birthplace id)
    (let ((card (make-sb-card name birthplace id)))
      (send card 'install)
      card)))

(define make-sb-card
  (lambda (name birthplace idnumber)
    (let ((id idnumber)
          (thing (make-thing name birthplace)))
      (lambda (message)
        (cond ((eq? message 'sb-card?) (lambda (self) #t))
              ((eq? message 'id) (lambda (self) id))
              (else (get-method thing message)))))))

```

Übung 8 Eine Person kann einen Ort nur dann betreten, wenn der Ort auf die Nachricht `accept-person?` mit `#t` antwortet:

```

> (send mike-office 'accept-person? mike)
#t

```

Alle Orte im vordefinierten System nehmen alle Personen auf. Erzeuge eine neue Sorte Ort, die eine Person nur dann hereinläßt, wenn sie eine `sb-card` dabei hat. Vervollständige dazu das folgende Gerüst:

```

(define make-card-locked-place
  (lambda (name)
    (let ((place (make-place name)))
      (lambda (message)
        (cond ((eq? message 'accept-person?)
              ...)
              (else (get-method place message)))))))

```

Ändere einige Orte in der Spielwelt, so daß sie nur Ausweisträger hereinlassen. Erzeuge einige Ausweise. Zeige, daß eine Person nur dann einen so gesicherten Ort betreten kann, wenn sie einen Ausweis trägt.

Abzugeben: Code für `make-card-locked-place`

Übung 9 Der unicolorsitar Oberhausmeister Oleg Golem hat verfügt, daß die Computerpools, Bibliotheken und WCs nur noch von Studenten mit speziellen Berechtigungen für die jeweiligen Räume betreten werden dürfen. Deshalb werden diese Räume mit Kartenschlössern versehen, die sich nur für diejenigen Personen öffnen, deren Studierberechtigungsausweise eine registrierte Identifikationsnummer besitzen.

Programmiere eine neue Klasse für Orte, sogenannte `smart-rooms`, welche das Vorhaben von Herrn Golem verwirklichen. Ein `smart-room` sollte dabei eine Liste von Identifikationsnummern verwalten (nicht die Karten selbst — ein Student könnte seine Karte versehentlich im WC herunterspülen und Ersatz beantragen). Zusätzlich zur Methode `accept-person?` wird noch eine weitere Methode `register-card` benötigt, welche die Identifikationsnummer einer Karte zur Liste hinzufügt. Als zusätzliche Sicherheitsmaßnahme soll `register-card` nur dann die Registration vornehmen, wenn die Karte sich bereits im Raum befindet.

Zur Demonstration, daß Deine `smart-rooms` funktionieren, erzeuge drei neue Pools, Bibliotheken und/oder WCs und einen Studierberechtigungsausweis für

Dich. Platziere Dich in einem dieser Räume, registriere Deinen Ausweis, versuche, in einen anderen geschützten Raum einzudringen, und begib Dich wieder zum Ursprungsort.

Abzugeben: Code für `smart-room`

Übung 10 Heimtückische Kriminelle haben in den letzten Wochen wiederholt Studierungsberechtigungsausweise gestohlen. Die Verbrecher müssen gefasst werden. Erzeuge eine neue Sorte Figur — einen `sheriff` — der sich wie ein Assi verhält, aber nur Personen frißt, die eine als gestohlen gemeldete Karte bei sich tragen. Benutze `mike` als Beispiel. Damit `sheriffs` effektiv arbeiten können, benötigen sie einen niedrigen Faulheitsfaktor.

Schreibe eine Prozedur (`report-stolen-id id`), welche einen neuen Sheriff erzeugt, der den Verbrecher jagt, welcher die Karte gestohlen hat. Sheriffs haben ihren Ausgangspunkt in Mikes Büro.

Abzugeben: Code für `report-stolen-id`

Übung 11 Rinus Öchel und Torsten Kulness halten die neuen Kartenschlösser für ein unnötiges Ärgerniss: Seit der Einführung der Schlösser besuchen ihre Freundinnen sie gar nicht mehr im Computerpool. Zum Glück lassen sich Studierungsberechtigungsausweise einfach kopieren.

Oleg Golem weiß um das Problem und will Buch über die Verwendung von Karten führen. Wenn eine Karte an zwei Orten zur selben Zeit auftaucht, dann muß eine der beiden eine Kopie sein.

Implementiere eine neues Objekt `big-brother`, welches eine Nachricht `inform` annimmt, deren Methode als Parameter eine Identifikationsnummer und einen Ort hat. `Big-brother` soll Situationen erkennen, in denen Fälschungen offenbar werden. Dazu könnte `big-brother` zum Beispiel eine Assoziationsliste unterhalten, die eine Person auf den Ort abbildet, an dem sie sich gerade befinden müßte. Diese Assoziationsliste darf allerdings jeweils nur über einen Zeittakt hinweg erhalten bleiben. Zu diesem Zweck wird die Systemzeit von einer Prozedur `current-time` (ohne Parameter) zurückgegeben. Du mußt den Code für gesicherte Räume (beide Sorten) verändern, so daß sie bei `big-brother` petzen, wenn jemand eine Karte benutzt, um Zutritt zu erlangen. `Big-brother` wiederum beauftragt bei der Feststellung eines Diebstahls einen Sheriff.

Abzugeben: Code für `big-brother`, Änderungen am Code für gesicherte Räume

Übung 12 Spiele ein Spiel: Verlasse ein WC, erlange Zutritt zu allen anderen gesicherten Räumen und begib Dich zurück ins WC.

Mache das Spiel interessant, indem Du einige weitere Studenten erzeugst. Programmier `act`-Methoden für diese Studenten, so daß sie sich innerhalb der Unicolorsität bewegen, Studierungsberechtigungsausweise und andere Dinge aufsammeln und auch ab und zu etwas liegenlassen. Erzeuge auf jeden Fall einige gefälschte Ausweise!

Übung 13 Entwirf einige Charaktere mit eigenem Verhalten. Dokumentiere und realisiere Deine Ideen und platziere die Figuren in ein neues Szenario. (Dies ist nicht als großes Projekt gedacht.)

Abzugeben: Code für die Erweiterungen

Übung 14 (Bonus) Wenn Figuren in den Himmel kommen, halten sie noch eine Abschiedsrede. Woher kommt der Text, wer sagte den richtigen Text ursprünglich, und was bedeutet er?