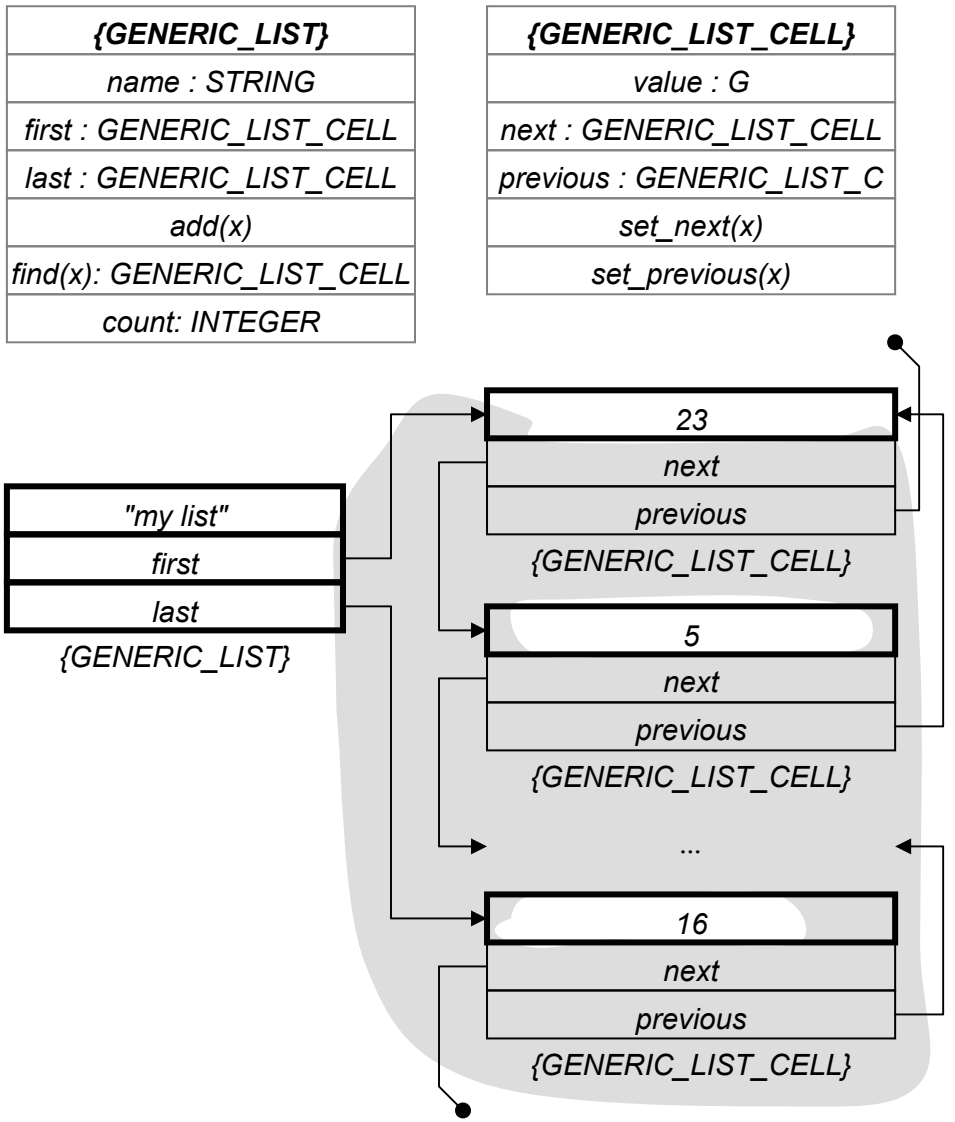
Information verstecken

Interface

Die Klasse definiert die *Interface* zwischen dem *Client* und dem Objekt. Der Klient kann über die *Features* auf das Objekt zugreifen. Ein Klient kann also nur auf die deklarierten Kommandos und Abfragen zugreifen und muss dabei immer die geforderten Argumente mitliefern. Die Definition der Schnittstelle wird zudem mit den *Contracts* ergänzt, welche noch klarere Verhältnisse schaffen soll. So soll zum Beispiel verhindert werden, dass ein *Feature* mit unsinnigen Argumenten aufgerufen wird. Zudem hält die Klasseninvariante Bedingungen fest, welche für das Objekt "immer" gelten sollen.

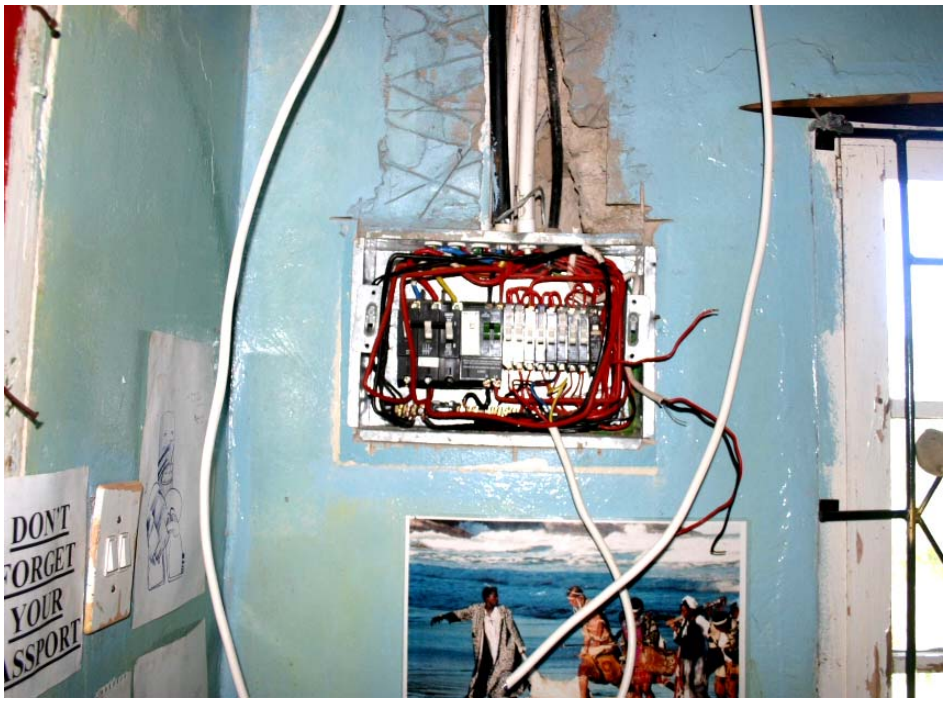
Soweit, so gut- aber man stelle sich nun folgenden Fall vor: eine Klasse *GENERIC_LIST* repräsentiert eine Liste mit Elementen. Diese Elemente sind als Attribut der Klasse *GENERIC_LIST_CELL* im Listenobjekt festgehalten. Der Klient erhält zum Beispiel über das *Feature count* Auskunft über die Anzahl Elemente in der Liste. Solche Features geben Information zum Zustand des Objektes.

Im Beispiel auf der rechten Seite sieht man oben die Klassen dargestellt und darunter Objekte dieser Klassen mit den Referenzen aufeinander, nachdem die Zahlen 23, 5, ..., 16 hinzugefügt wurden. Die Klasseninvarianten sind erfüllt, *first* zeigt auf das erste Element, *last* auf das letzte und die einzelnen *GENERIC_LIST_CELL* Objekte sind auch sinnvoll miteinander verknüpft. Soweit hat die *GENERIC_LIST* Klasse ihre Aufgabe korrekt erfüllt: die Objekte halten sich an die Invariante.



Eine Liste mit Zahlen (23, 5, ..., 16)

Nun schreiben wir ein Programm, das in *my_list* die Nummer 5 sucht: mit dem Abfrage *e := find(5)* erhalten wir das entsprechende Element als Objekt der Klasse *GENERIC_LIST_CELL*. Wir können nun die Referenzen von *next* und *previous* ändern. Das heisst, der Zustand der Liste kann von aussen beliebig verändert werden. Das ist schlecht, weil jetzt die Klasseninvariante von *GENERIC_LIST* nicht mehr stimmt.



Port St. Johns, Süd Afrika: Bei diesem Sicherungskasten wurde kein Information Hiding angewendet

Interner und von aussen sichtbarer Zustand

Abhilfe schafft die genaue Kontrolle, was von aussen (das heisst, vom *Client*) sichtbar ist: von aussen sichtbarer Zustand. Und was nicht nach aussen dringen soll: interner Zustand. Im oberen Beispiel sind die internen Details grau hinterlegt und die von aussen sichtbaren Features mit einem fetten Kasten umrahmt.

In Eiffel kann jedes Feature für spezifische Klassen freigegeben bzw. sichtbar gemacht werden. Dazu gibt man einfach an:

```
feature { ANY }
    mein_von_aussen_sichtbares_feature is ...
    ein_anderes_sichtbares_feature is ...
feature { NONE }
    mein_nur_vom_eigenen_objekt_erreichbares_feature is ...
feature { KLASSEA, KLASSEB }
    mein_feature_fuer_mich_und_a_und_b is ...
```

Die Sichtbarkeit wird also in geschweiften Klammern nach dem Keyword *feature* angegeben. Mehrere Klassen werden mit einem Kommas getrennt hintereinander aufgezählt. Die Vererbungshierarchie gilt hier auch: alle von einer Klasse vererbten Klassen sehen die ihr sichtbaren *Features* auch. Da alle von *ANY* vererben, sehen diese *Features* alle. Und weil niemand von *NONE* vererbt, sind jene *Features* immer unsichtbar.

Geheimnisvolle Zelle

Zurück zu unserem Beispiel: die *GENERIC_LIST_CELL* sollte also lediglich sich und der Liste die Möglichkeit geben, die Attribute *next* und *previous* zu ändern. Von aussen soll nur der Wert gelesen und geändert werden können. Deshalb implementieren wir:

```
feature { GENERIC_LIST }
    set_next( x : G ) is ...
    set_previous( x : G ) is ...
feature { ANY }
    set_value( x : G ) is ...
```