27.10.2004 13:41:48

0th dimension: "Field"
- is the SUI (Smallest Unit of Interest)
- is virtually accessible by a fieldIdentifier
- no axis

1th dimension: "Record"
- is a Set of Fields
- provides access to fieldValues by means of fieldIdentifier
- notifies about changes of Fields, the event carrying the fieldIdenifier
- analogy to 0th - is virtually accessible by recordIdentifier
- one dimensional axis spanned by fieldIdentifiers

2th dimension: "Table"
- is a Set of Records
- provides 2-level depth access, first level: to records by means of recordIdentifier, second level: to fields by means of a pair of fieldIdentifier/recordIdentifier
- provides 2-level depth notification, first level: about Record changes, the event carrying the recordIdentifier; second level: about Field changes, the event carrying a pair of fieldIdentifier/recordIdentifier
- one dimensional axis is spanned by fieldIdentifiers, the other dimensional axis is spanned by recordIdentifiers

The characterization as "Set" is intentional: there is no inherent sequence in the underlying objects to be modelled, neither the properties of a bean, nor the attributes of a relation, nor the records returned by a query are "naturally" ordered. Any indices involved are implementation artefacts.

A common pattern across the levels is that both access and change notification is handled by the next higher dimension. In these "looking down" contexts the center of interest is the content, not the controlling structure of the content.

The Table has no notion of "current" - that's in analogy with the core Swing which always keeps the selection decoupled from the data.

Literally translating Record and Table into a minimal pseudo-code:

Record

```
        Set getFieldIdentifiers();
        Object getFieldValue(fieldIdentifier);
        void setFieldValue(fieldIdentifier, fieldValue)

        add/removeFieldChangeListener(l)

FieldChangeEvent
        getRecord();
        getFieldName();

a Listener will react to notifications:
        Record record = event.getRecord();
        changedField = record.getFieldValue(event.getFieldName());
```

Table

one level down:

```
        Set getRecordIdentifiers();
        Object getRecordValue(recordIdentifier);
```

```
        void setRecordValue(recordIdentifier, recordValue)
        add/removeRecordChangeListener(l)

RecordChangeEvent
        getTable()
        getRecordIdentifier()

a Listener will react to notifications:
        Table table = event.getTable();
        changedRecord = table.getRecordValue(event.getRecordIdentifier())

two levels down:
        Set getFieldNames();
        Object getFieldValue(fieldName, recordIdentifier)
        void setFieldValue(fieldName, recordIdentifier, fieldValue)
        add/removeTableFieldValueListener(l)

TableFieldChangeEvent
        getTable();
        getFieldIdentifier();
        getRecordIdentifier();

a Listener will react to notifications:
        Table table = event .getTable();
        changedFieldValue = table.getFieldValue(
          event.getFieldName(), event.getRecordIdentifier)
```

A "field value change" fired by a Record is different from a "field value changed" by a record because the type of sender and the type of information in the event is different, so the listeners should be different as well:

```
RecordListener
        void fieldValueChanged(FieldChangeEvent)

TableListener
        void recordValueChanged(RecordChangeEvent)
        void fieldValueChanged(TableFieldChangeEvent)
```

The points of contact to the current jdnc codebase:

Binding is interested in Field: it wants to handle the synch between a fieldValue and a Component. To do so it needs to know the controlling DataModel.

A Field is an Object as accessed by a DataModel getValue(fieldName)

A DataModel is roughly on the level of a Record with a slight mix-in of a Table. The mix-in consists in the recordIndex/count related methods. The 1th dimensional axis is spanned by an ordered set of fieldNames, for the sake of the view-related clients. At this level the introduction of indices does no harm because the value access is by identifier and the convenience indices dont interfere.

In the core jdnc there is no equivalent to a Table. The mix-in of the recordIndex in DataModel is suggestive to be used as an extension into the second dimension - but is not strong enough, because it has no identifying notion.