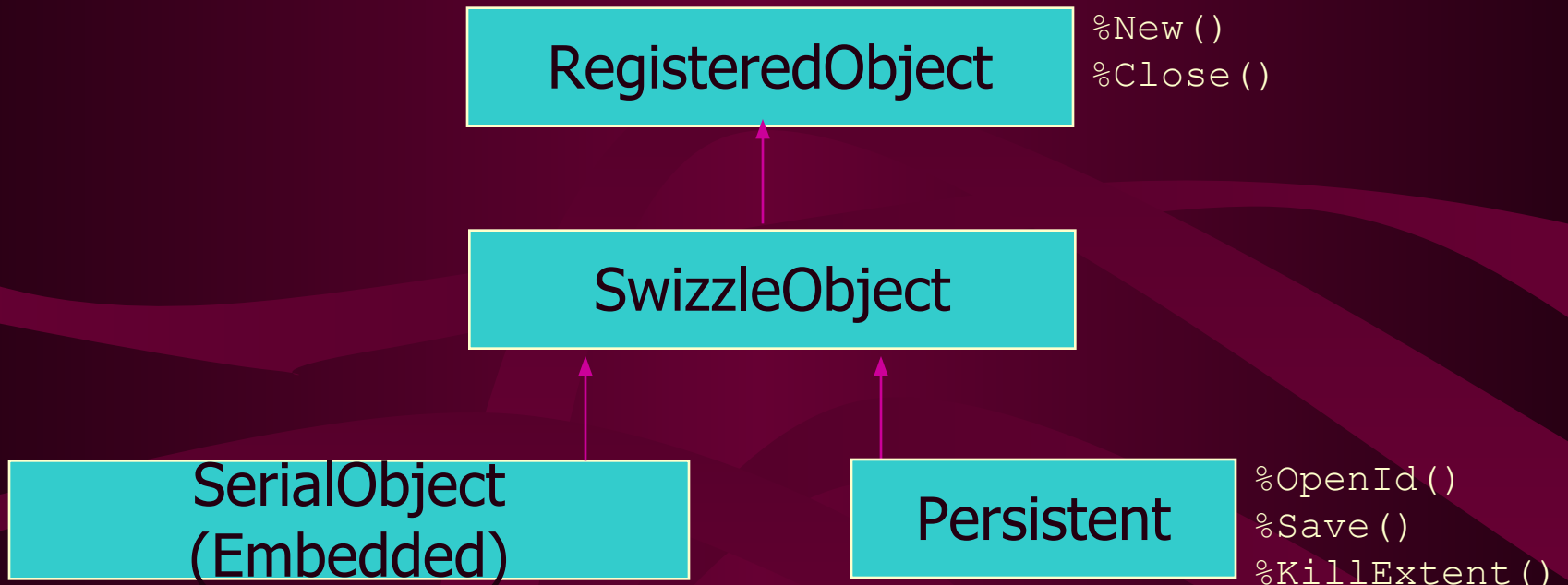
Модуль: Свойства

Иерархия: классы %Library

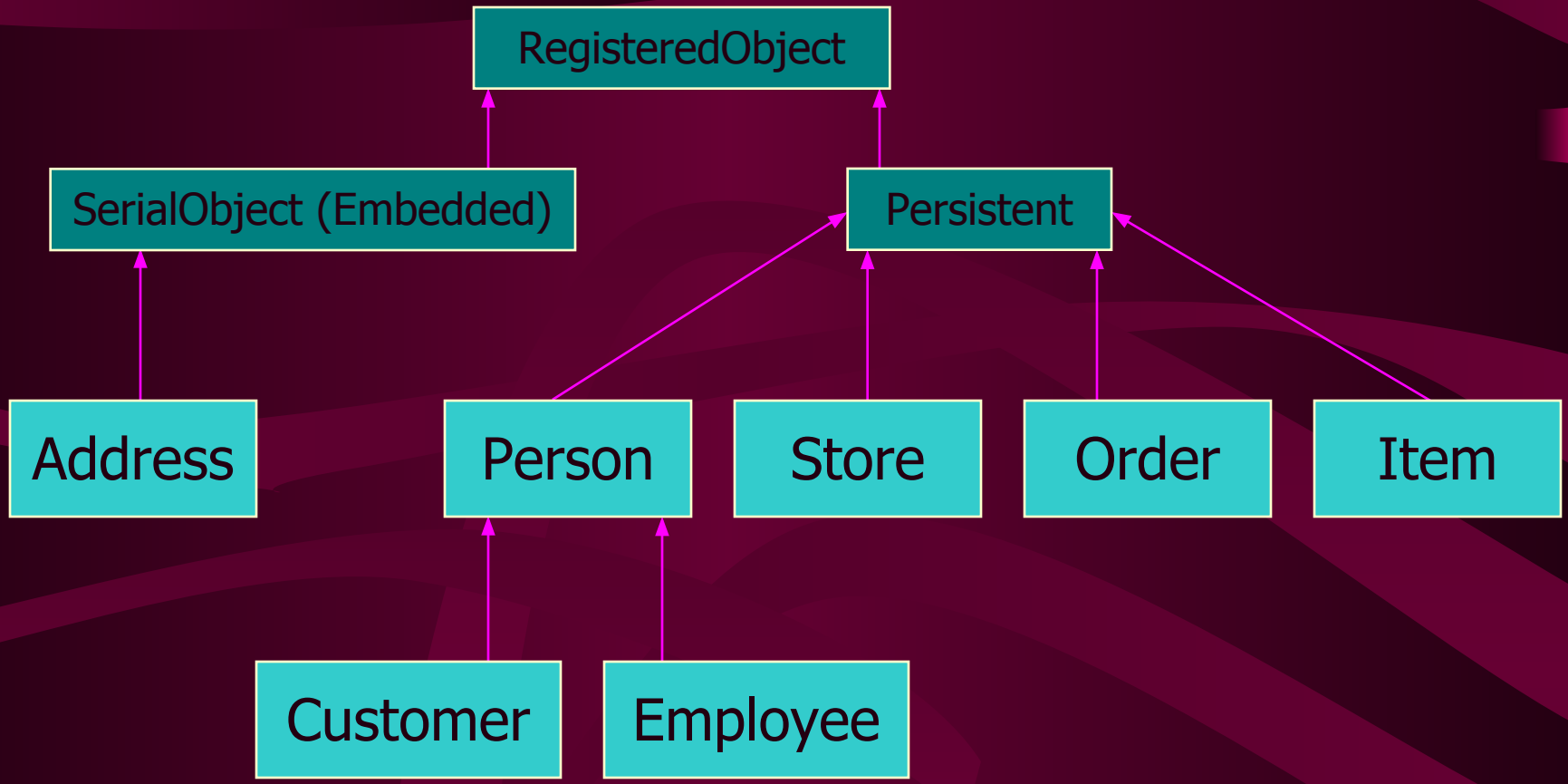
- Классы объектов



- Классы типов данных

String | Integer | Name | Date

Иерархия: Классы Lipa



Свойства

- Свойства определяют состояние объекта. Они могут представлять:
  - Литералы
  - Потоки символьных или двоичных данных
  - Коллекции литералов
  - Ссылки на встраиваемые или хранимые объекты
  - Отношения между классами
- Свойство, которое представляет литеральное значение имеет набор методов для форматирования и валидации.

Свизлинг

- Свойства, которые являются ссылками на встраиваемые или хранимые объекты автоматически подкачиваются в память при первом обращении. Это называется свизлинг или подкачка.
- Следующая строка кода «подкачивает» Customer в память, через открытый объект Order.

```
write ord.Customer.Name
```

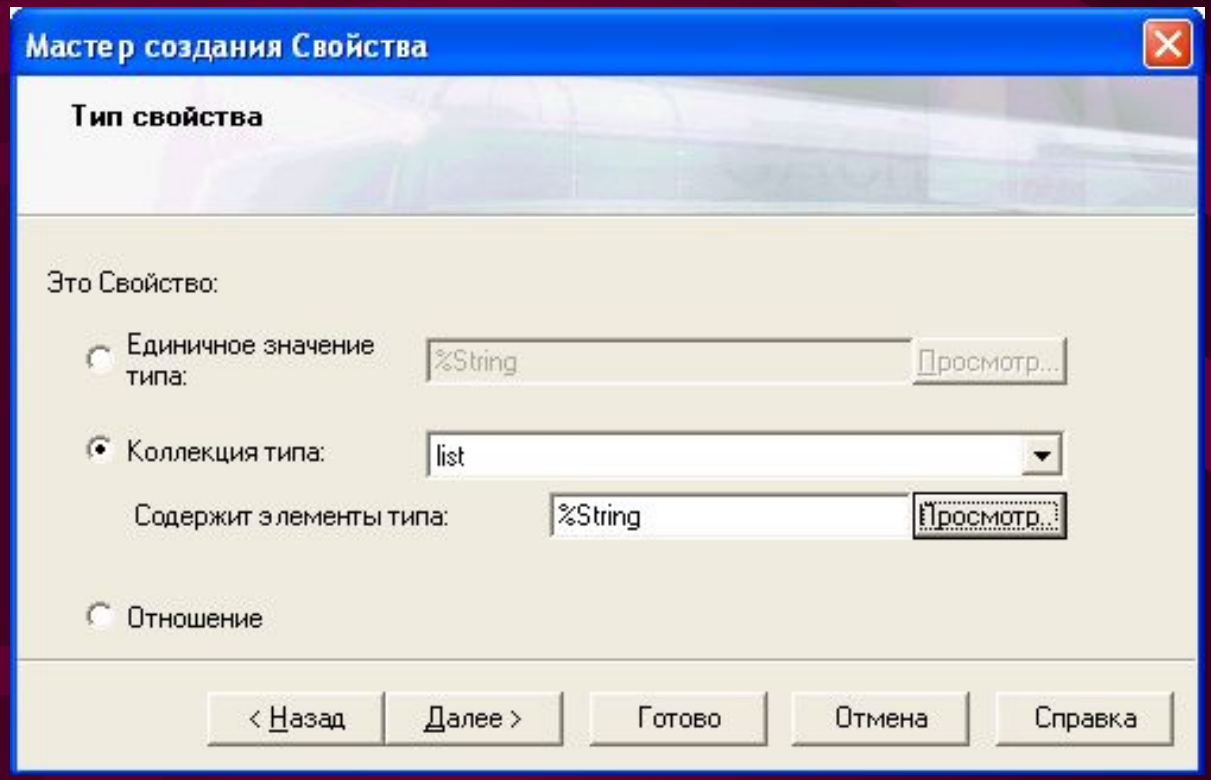
- Таким образом, объект Customer извлечен без использования метода %OpenId().

Характеристики

- Свойство типа Private может быть использовано только из методов данного класса.
- Свойство типа Calculated не хранится перманентно; его значение вычисляется в режиме работы из значений других свойств.
- Свойство типа Required обязательно должно иметь значение во время сохранения объекта.
- Свойство типа Final не может быть переопределено в классах-наследниках.
- Свойство типа Transient не хранится в БД; оно доступно только в режиме работы.
- Многомерное свойство – это многомерный массив.
- По свойству типа Indexed создается индексная таблица.
- Свойство типа Unique должно иметь уникальное значение в классе.

Коллекции

- Коллекции могут быть следующими:
  - Список (%Collection.AbstractList)
  - Массив (%Collection.AbstractArray)
- Информацию об особенностях использования коллекций можно найти в документации вышеуказанных классов.



InterSystems CACHE

Списки и массивы

- Списки и массивы используются как коллекции значений.
- Каждый элемент коллекции имеет индекс.
- В списке элементы характеризуются порядковым номером.
  - Например, если мы добавляем три элемента в список, им будут присвоены номера 1-3. Если второй элемент удаляется, оставшиеся элементы будут пронумерованы как 1 и 2.
- В массиве элементы определяются по указанному индексу.
  - Например, три элемента в массиве могут иметь индексы"A", "X" и "4". Если элемент "X" удаляется, индексы других элементов остаются неизменными.
- Свойства типа список могут содержать до 32 Кб данных.

Пример использования списка

- Создать список детей:

```
USER>set list = ##class(%Library.ListOfDataTypes).%New()
USER>do list.Insert("Aric")
USER>do list.Insert("Lon")
USER>do list.Insert("Emily")
```

- Вывести второй элемент:

```
USER>write list.GetAt(2)
Lon
```

- Подсчитать количество подарков на новый год

```
USER>write list.Count()
3
```

Пример использования массива

- Создать массив детей:

```
USER>set array = ##class(%Library.ArrayOfDataTypes).%New()
USER>do array.SetAt("Aric",$zdh("12/16/64"))
USER>do array.SetAt("Lon",$zdh("6/23/67"))
USER>do array.SetAt("Emily",$zdh("4/21/70"))
```

- Вывести элемент:

```
USER>write $zdh("6/23/67")
46194
USER>write array.GetAt(2)

USER>write array.GetAt(46194)
Lon
USER>write array.Count()
3
```

Массивы и списки в проекции SQL

- Список проецируется в виде отдельного столбца, со всеми элементами в одной строке.
- Массив проецируется как отдельная таблица с ссылкой на основную таблицу.
  - Для большого количества элементов коллекции лучше проецировать в отдельную таблицу.

Потоки

- Потоки используются для хранения больших объемов данных, превышающих предел обыкновенных свойств в 32Кб.
- Существуют 2 типа потоков – символьные и бинарные.
  - Символьные потоки используются для хранения большого объема текста, например, глав книги.
  - Двоичные потоки хранят большие двоичные объекты, например, картинки.

Параметры потоков

- Для создания свойства как потока выбрать тип из:
  - %GlobalCharacterStream
  - %FileCharacterStream
  - %GlobalBinaryStream
  - %FileBinaryStream

- Для указания места хранения используйте параметр LOCATION
  - LOCATION: <имя глобали> или <имя директории>
  - Вначале Caché пишет поток во временное хранилище.
  - Когда объект, содержащий поток сохраняется Caché копирует поток в постоянное хранилище.

Пример использования потоков

- Потоки – это встраиваемые объекты. В модуле 4 мы видели, что создать новый встраиваемый объект можно либо явно, либо неявно (используя точечный синтаксис).
- В отличии от встраиваемых объектов, потоки необходимо создавать неявно. Иначе, Caché будет хранить данные потока только во временном хранилище.
- Создайте клиента и определите некоторые заметки о нем.

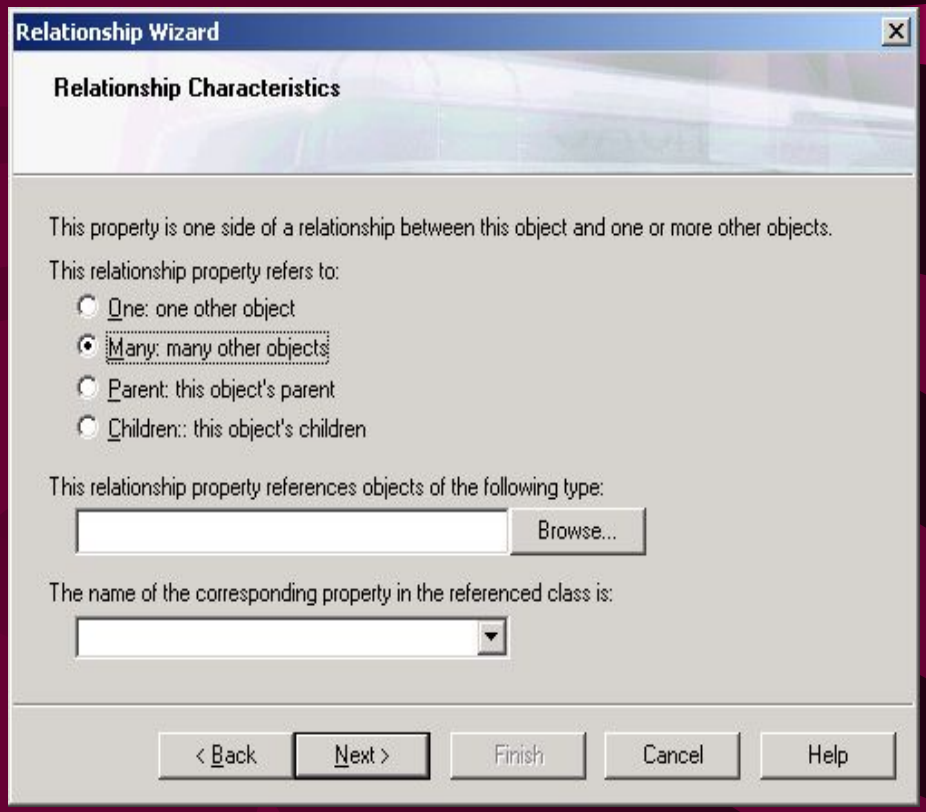
```
USER>set cust = ##class(Nothing.Customer).%New()
USER>do cust.Notes.Write("This is my first stored stream.")
USER>set cust.Name = "Doe,Jane"
USER>set st = cust.%Save()
```

Отношения

- Отношения определяются между двумя хранимыми классами.
  - Свойство в одном классе имеет указатель на другой класс.
- Класс может также иметь указатель на себя.
- Существует два типа отношений:
  - Один-ко-многим (независимое)
  - Parent-to-children (зависимое)
- Отношения обеспечивают ссылочную целостность при операциях удаления на стороне один или parent.

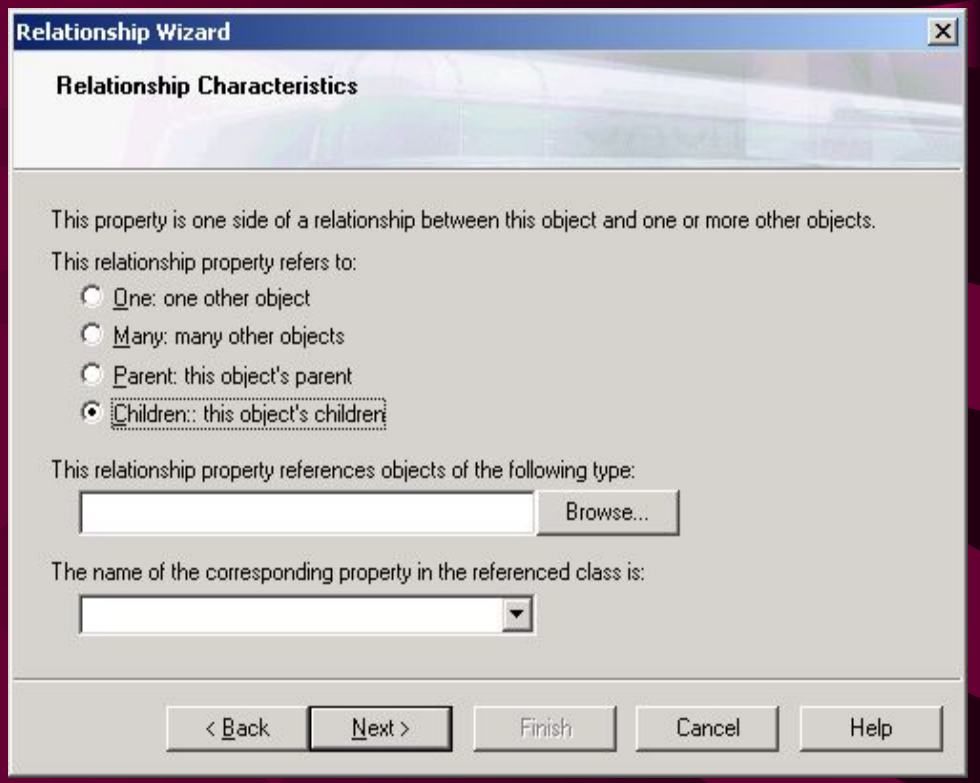
Один-ко-многим

- Используйте этот тип отношений, когда:
  - Более чем один объект класса B (сторона много) может ссылаться на объект класса A (сторона один),
  - Объекты класса B могут также существовать независимо от объектов класса A.
- Попытка удаления объекта класса A будет неуспешна, если имеется ссылка хотя бы на один объект класса B.
  - Для удаления объекта класса A, необходимо удалить ссылки на него из всех объектов класса B.



Parent-To-Children

- Используйте этот тип отношений, когда:
  - Более чем один объект класса B (children) может ссылаться на объект класса A (parent),
  - Объекты класса B не могут существовать независимо от объектов класса A.
- Удаление объекта класса A **удалит все связанные с ним объекты класса B**.
- Объект класса B не может изменить связанным с ним объект класса A на другой объект.

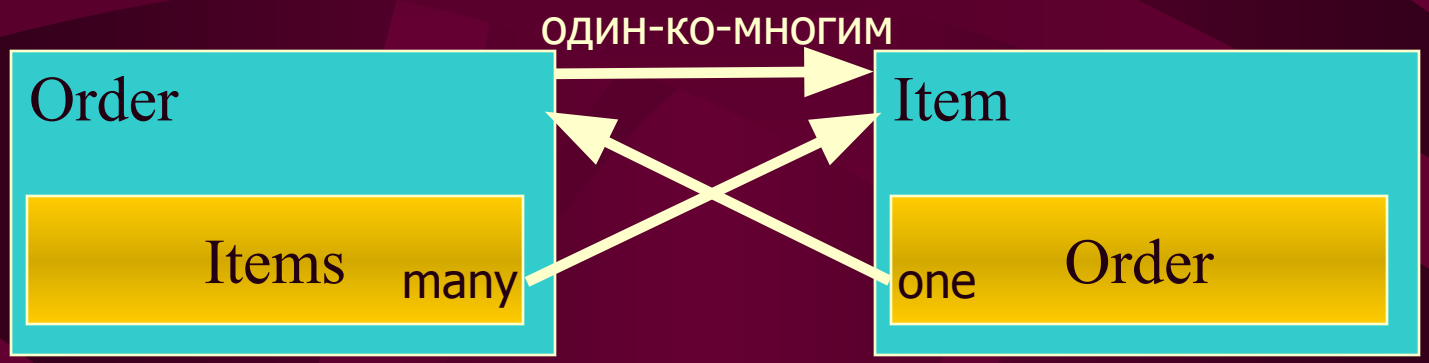


Создание отношений

- Создать отношение можно либо со стороны Один/Parent либо со стороны Много/Children.
- Добавьте свойство типа отношение в хранимый класс и определите его мощность. Caché добавит связующее свойство в другой класс.
- Классы Parent/Children компилируется совместно.
- Отношения используются вместо списков или массивов объектов, когда необходимо обеспечить ссылочную целостность.

Мощность отношения

- Когда между двумя классами определено отношение, опишите мощность в терминах классов.
  - “Order имеет один-ко-многим отношение с Item: Один Заказ содержит несколько товаров.”
- При создании отношения, опишите мощность в терминах свойств.
  - “Свойство Items класса Order имеет мощность Many. Свойство Order класса Item имеет мощность One.”



Пример использования отношений

- Информация об отношениях доступна через документацию класса %Library.RelationshipObject.
  - RelationshipObject имеет те же имена методов, что и класс AbstractList.
- Использовать отношения можно с любой стороны. Например, есть объект класса Item-`it` и объект класса Order `ord`, необходимо добавить Item в Order:

```
USER>set it.Order = ord
USER>do ord.Items.Insert(it)
```

- Создавайте каждый объект явно. Определяйте между ними отношение, как показано выше.

Использование свойств

- Для использования свойств связанных объектов необходимо использовать каскадный точечный синтаксис.
- Например:

```
USER>write it.Order.Date
58700
USER>write ord.Items.Count()
2
USER>write ord.Items.GetAt(1).Price
5.99
```

Parent->Children ID

- Идентификаторы объектов Children в отношении Parent->Children имеют формат “a||b”:
  - a = ID стороны parent и
  - b = ID стороны child внутри parent.
- Например:

```
USER>write ord.Items.GetAt(1).%Id()
3||1
```

- ID заключается в двойные кавычки , если используется в качестве аргумента к методу %OpenId.

```
set it1 = ##class(User.Item).%OpenId("3||4")
```

Использование стороны Children/Много

- Создать список объектов:

```
USER>set ord = ##class(Nothing.Order).%New()
USER>set item1 = ##class(Nothing.Item).%New()
USER>set item2 = ##class(Nothing.Item).%New()
USER>do ord.Items.Insert(item1)…
```

- При выводе объекта, выводится его oref.

```
USER>write ord.Items.GetAt(2)
9 ; всего лишь oref
```

- child не имеет ID, пока объект не сохранен.

```
USER>write ord.Items.GetAt(2).%Id()

USER>set st = ord.%Save()

USER>write ord.Items.GetAt(2)
9 ; всего лишь oref
USER>write ord.Items.GetAt(2).%Id()
1||2 ; агрегация ID
```

Многие-ко-многим

- Отношения типа Многие-ко-многим не поддерживаются, однако могут быть построены при помощи двух отношений типа Один-ко-многим.
- На примере поставщики и рестораны могут быть рассмотрены 2 варианта отношения многие-ко-многим.
- Группа
  - Существует несколько групп с представителя в каждом классе
  - Например: вертикальная монополия где несколько крупных корпораций владеют группой поставщикой и группой ресторанов.
- Транзакция
  - Каждый представитель каждой группы-независимый агент.
  - Пример: ситуация свободного рынка, где любой поставщик может обслуживать любой ресторан.

Пример группы

- Между 2 классами (Поставщик и Ресторан), связанных отношением многие-ко-многим, создайте третий класс-группу (MegaCorp).
- MegaCorp имеет отношение один-ко-многим с классом Поставщик и отношение один-ко-многим с классом Ресторан.
- Объект MegaCorp не может быть удален до тех пор, пока имеется указатель на любой объект класса Поставщик или Ресторан.

Пример Транзакции

- Для отношения многие-ко-многим между 2 классами (Поставщик и Ресторан) создайте третий класс (Transaction).
- Каждая Transaction-это новое отношение.
- Поставщик и Ресторан относятся по схеме один-ко-многим классу Transaction.
  - Каждая Transaction представляет ссылку между одним Поставщиком и одним Рестораном.
- Поставщик и Ресторан не могут быть удалены до тех пор, пока существует объект Transaction.