What's more, you must attribute each of your domain object classes with DBStorageGroup If you don't attribute with DBStorageGroup, re-store will assume a slightly different schema and thus be incompatible with your existing schema. This effect of DBStorageGroup, and the choice of "either put your classes into separate storage groups or re-generate your database" might not look quite logical and more a bug than a feature. An explanation can be found in the section *DBStorageGroup- and BindableDomainObject effects* below.

Your PhoneBook.Domain project should still build and run after these modifications. If you attributed each class with DBStorageGroup, it should still be compatible with your database schema, as it was generated by dbschema.exe.

# Layer supertype

In larger applications it makes sense to have all domain objects share common properties and methods. For example, all of our PhoneBook domain object classes could support an extra propery "notes", so that you can annotate a domain objects in arbitrary ways. For example

* driving directives for Location objects
* if it is a restaurant, that you did not like the food there
* snyde comments about Person objects
* "never call there after 8 pm (kids)" for a phone-number

Such an extra Notes-property can be put into a common abstract base-class named NoteTakingDomainObject, derived from BindableDomainObject. Location, Person and PhoneNumber can inherit the Notes-property from that NoteTakingDomainObject  
base-class.

Introducing a common base-class for a certain layer (the domain layer, for example) is a pattern named *layer supertype*. Martin Fowler summarizes this pattern as "a type that acts as the supertype for all types in its layers" (in "Patterns of Enterprise Application Architecture").

We will not really introduce this feature into the application, but such a *layer supertype* has implications for schema generation. That's why you must attribute your domain object classes  
with DBStorageGroup as soon as you derive them from BindableDomainObject.

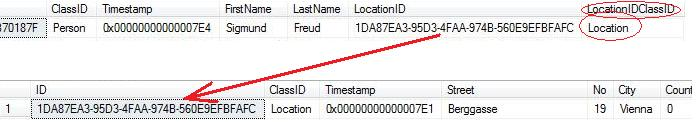
In a way, BindableDomainObject *is* such a layer supertype. After all, it gives all the domain objects in the *binding layer* its ability to have their controls rendered on a screen.

# DBStorageGroup- and BindableDomainObject effects

The introduction of an extra step of inheritance between DomainObject and your derived domain object classes also introduces a new twist into your data model. Per default the  
framework assumes that all classes derived from a common base-class will share at least some data in a single database row. Until now, with each of the three domain object classes   
(Location, Person, PhoneNumber) directly derived from DomainObject, the framework had no reason to think about how to share data among the three classes, because none of it had a common ancestor between DomainObject and itself.

Enter the new base-class BindableDomainObject. No matter whether you opt for single table inheritance or concrete table inheritance (see previous section), in the case of   
BindableDomainObject, both dbschema.exe and re-store will assume that domain object classes will share data, and that database records will be shared among classes.

For single table inheritance it should be clear that a row in a table contains the data for more than one class. So when a domain object (like Person) has a reference to another domain object (like-) it is not enough to know that the database record 1001 holds the record for that PhoneNumber. Since we use records in our single table for Locations, Persons and PhoneNumbers, re-store must always store type information (i.e. the class name) together with the foreign key. This type information is obsolete for a schema where each class gets its own table. The following pictures illustrate the concept. Here is the Sigmund Freud record, "pointing" to his home Location record. The extra column LocationIDClassID specifies that the record behind the foreign key must be interpreted as a Location:



The problem here is that

* both dbschema.exe and re-store assume that type-specifying extra columns are necessary even if they are not
* you can fix this by attributing classes with DBStorageGroup although it is not quite plausible why this is so

In other words: this is a wart in the re-store API. re-store is not as smart as you and me and can't always judge competently where columns like LocationIDClassID are really required and sometimes plays it safe. You should be aware of this when making experiments with domain object classes, layer supertypes and table inheritance.