Chap 6. CORBA-based Architecture

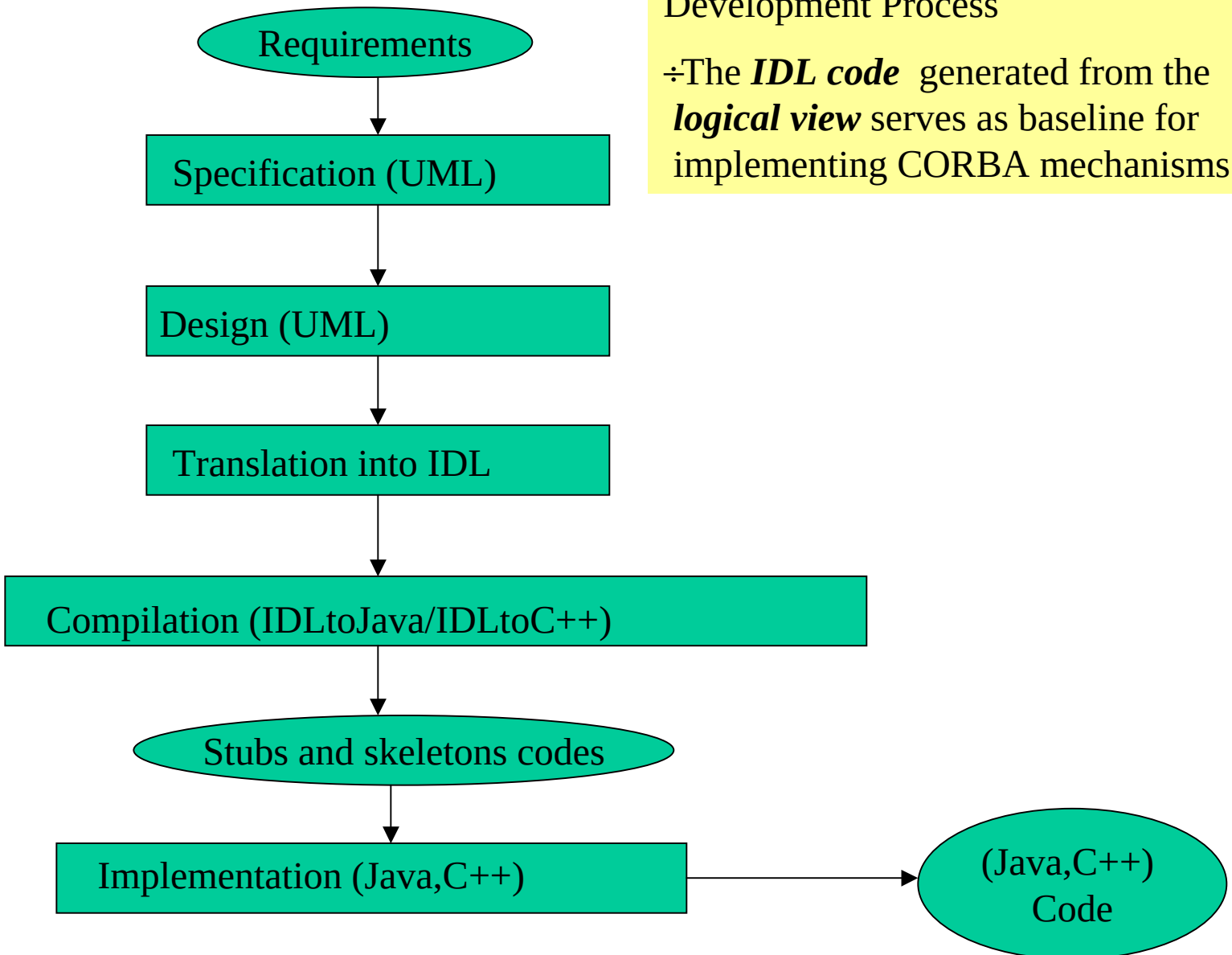
Part 6.3 Designing CORBA Systems

1. General Approach
2. From OO Design to IDL
3. UML Profile for CORBA
4. Refinement of UML models-IDL Generation

1. General Approach

÷The design of a CORBA system fits in regular Software Development Process

÷The ***IDL code*** generated from the ***logical view*** serves as baseline for implementing CORBA mechanisms



2. From OO Design to IDL

☛IDL model is a refinement of OO design product:

•doesn't capture all the semantics of an OO design, however it expands the details of features such as attributes and operations in the static model.

•captures only the interface information included in OO models

•is not intended to represent implementation characteristics such as dynamic behavior, object interactions etc.

☛IDL is useful to capture a key subset of OO static models:

•public attributes and public operations

•inheritance relationships

•associations may be represented indirectly as attributes

•expands the attributes and operations definitions by providing detailed types definitions, strongly typed operation signatures, and exception definitions.

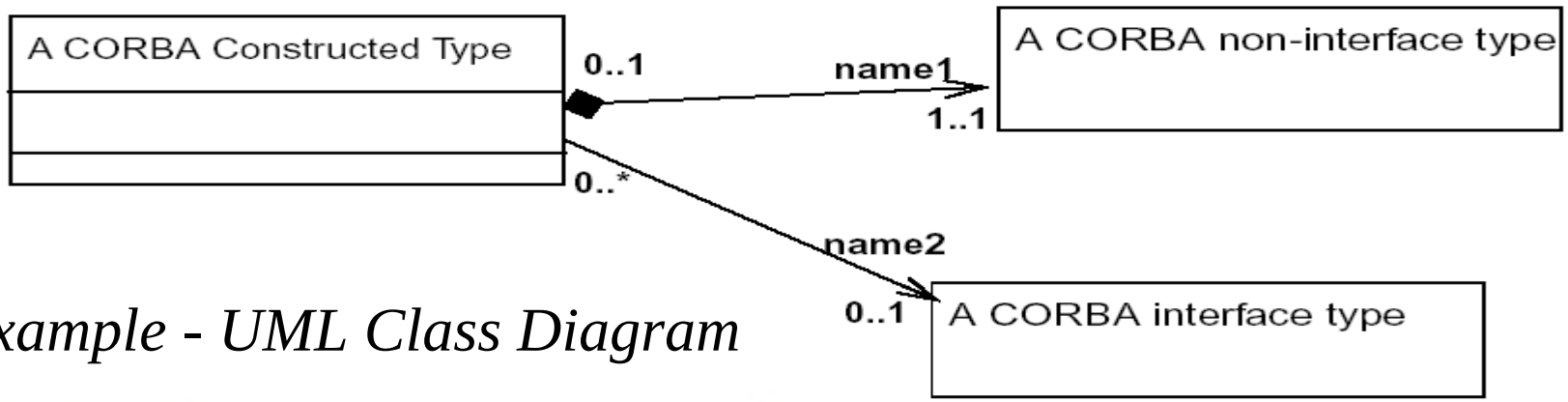
☛Refinement from OO to IDL may be conducted systematically using ***UML profile for CORBA***:

•allows partial automation of the IDL generation process.

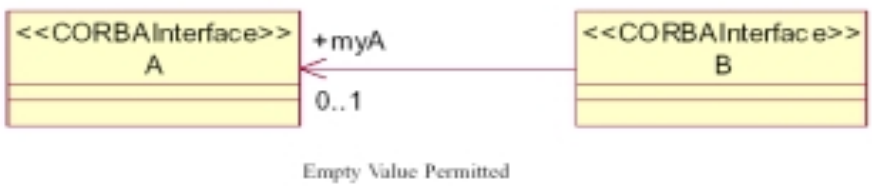
3. UML Profile for CORBA

•The UML Profile for CORBA specification was designed to provide a standard means for expressing the semantics of CORBA IDL using UML notation and thus to support generation of these semantics using UML tools.

Using Associations to Represent User-Defined Types



Example - UML Class Diagram



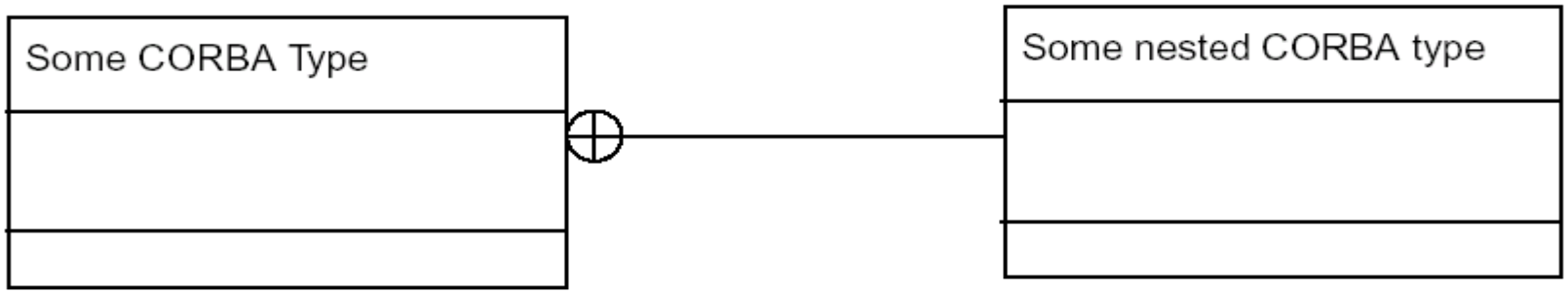
Empty Value Permitted

Empty Value Not Permitted

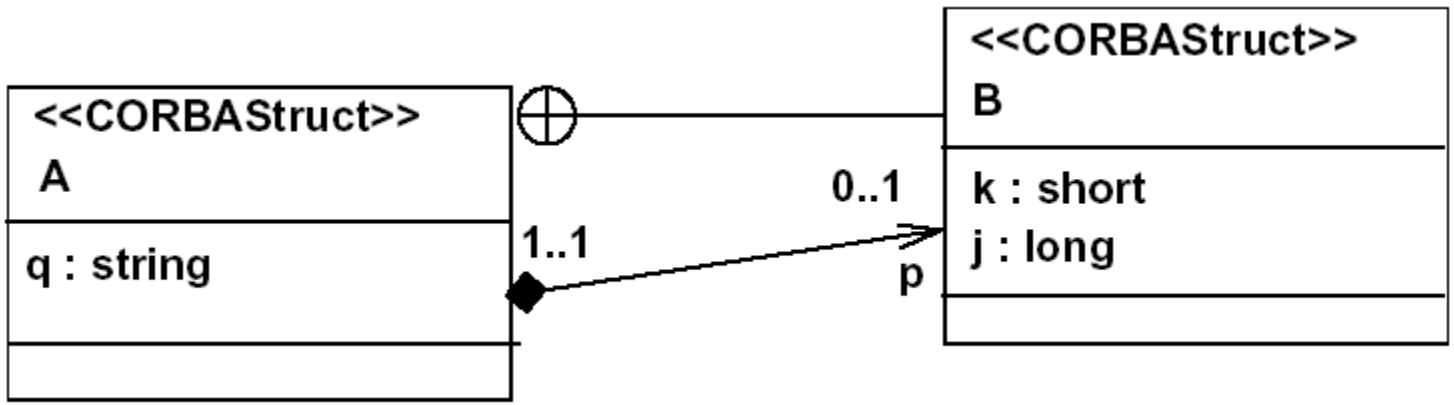
Example IDL

```
interface A {};
interface B {
        attribute A myA;
};
```

UML Namespace Containment Notation for Nested CORBA Constructs

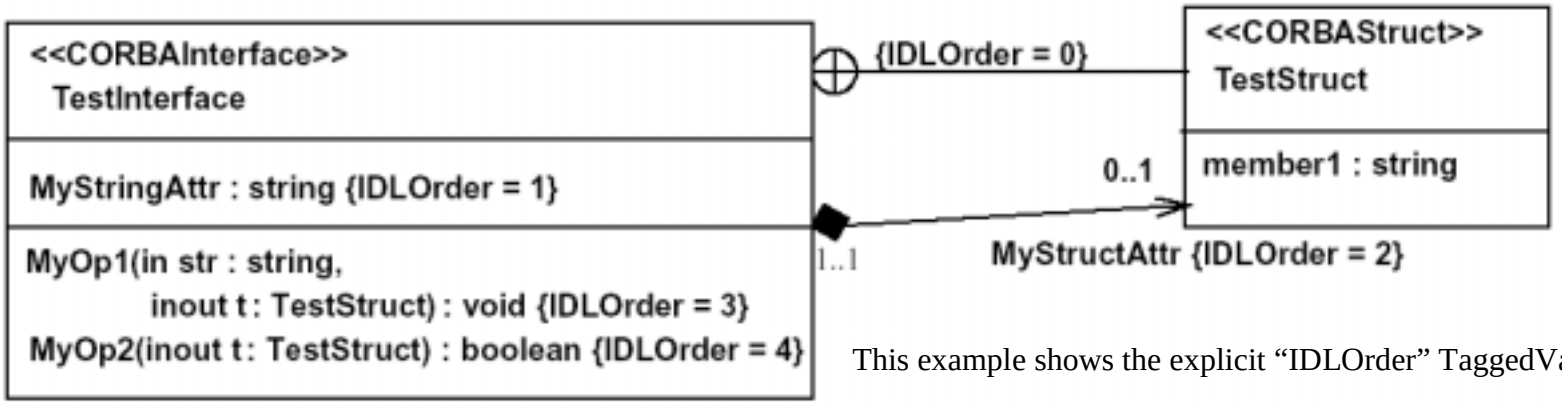


Example - Nested Struct




```
struct A {
        struct B {
                short k;
                long j;
                        } p;
                string q;
};
```

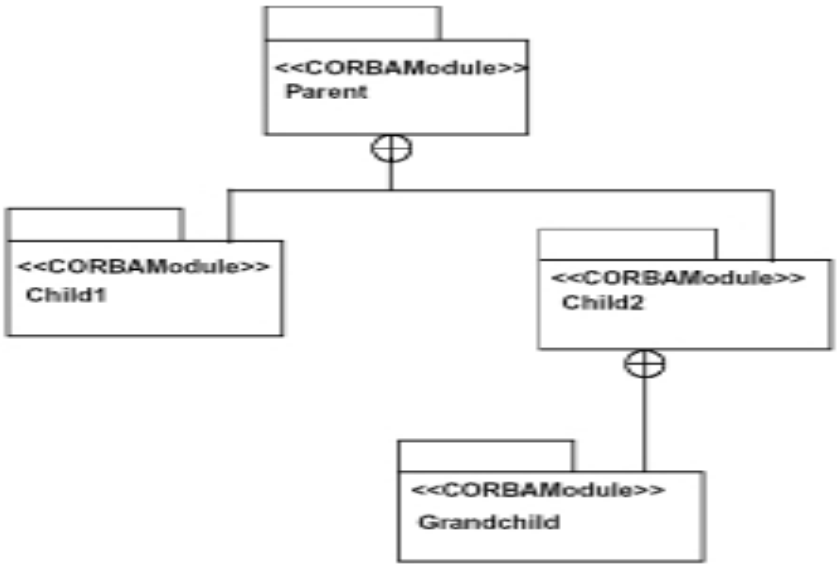
Example - Interface Containing a Struct



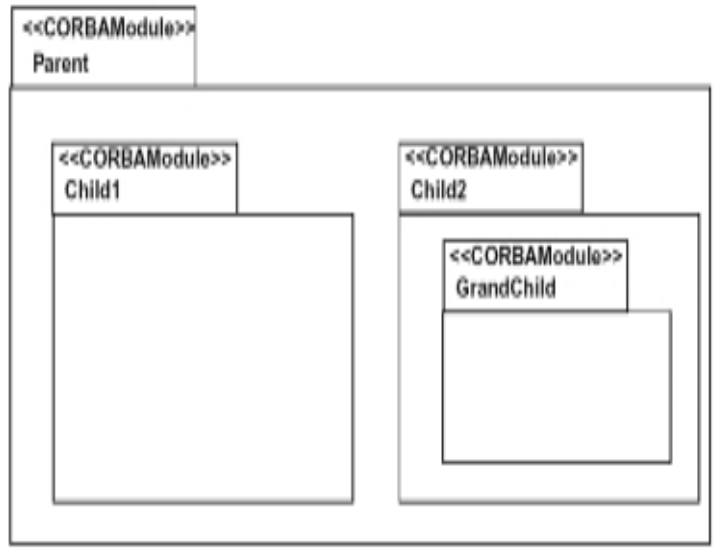
This example shows the explicit "IDLOrder" TaggedValues on each of the Attributes, Associations, and Namespace containments for preserving the ordering given in the IDL.

```
interface TestInterface {
        struct TestStruct {
                string Member1;
        };
        attribute string MyStringAttr;
        attribute TestStruct MyStructAttr;
        void MyOp1( in string str, inout TestStruct t);
        boolean MyOp2( inout TestStruct t);
};
```

Example - Module Namespace Containment



Module Package Notation



Module Declaration

```
module Parent {
        module Child1 {};
        module Child2 {
                module Grandchild {};
        };
};
```

Type Definition Using UML Profile for CORBA

-You can map Corba constructs to UML model elements in two ways:

÷by defining the elements within an interface (as nested element), in which case the element is referenced in terms of the interface.

÷by defining an independent class that represent the types; in that case other elements can reference the type directly.

Features	**Sterotype**	**Properties in Rational Rose**
Interface	Interface	
Exception	CORBAException	Set **Raises** property for relevant operations.
Constant	CORBAConstant	**implementation type** and **value**
Enumeration	CORBAEnum	
Structure	CORBAStruct	
Type renaming	CORBATypedef	**Implementation Type**
Union	CORBAUnion	**Implementation Type** for the **switch**
Sequence	(Association to supplier)	**Bounded Role Type** set to **Sequence** and **Cardinality** set to dimension of sequence.
Array	(Association to supplier)	**Bounded Role Type** of supplier set to **Array; cardinality** set to array dimension.

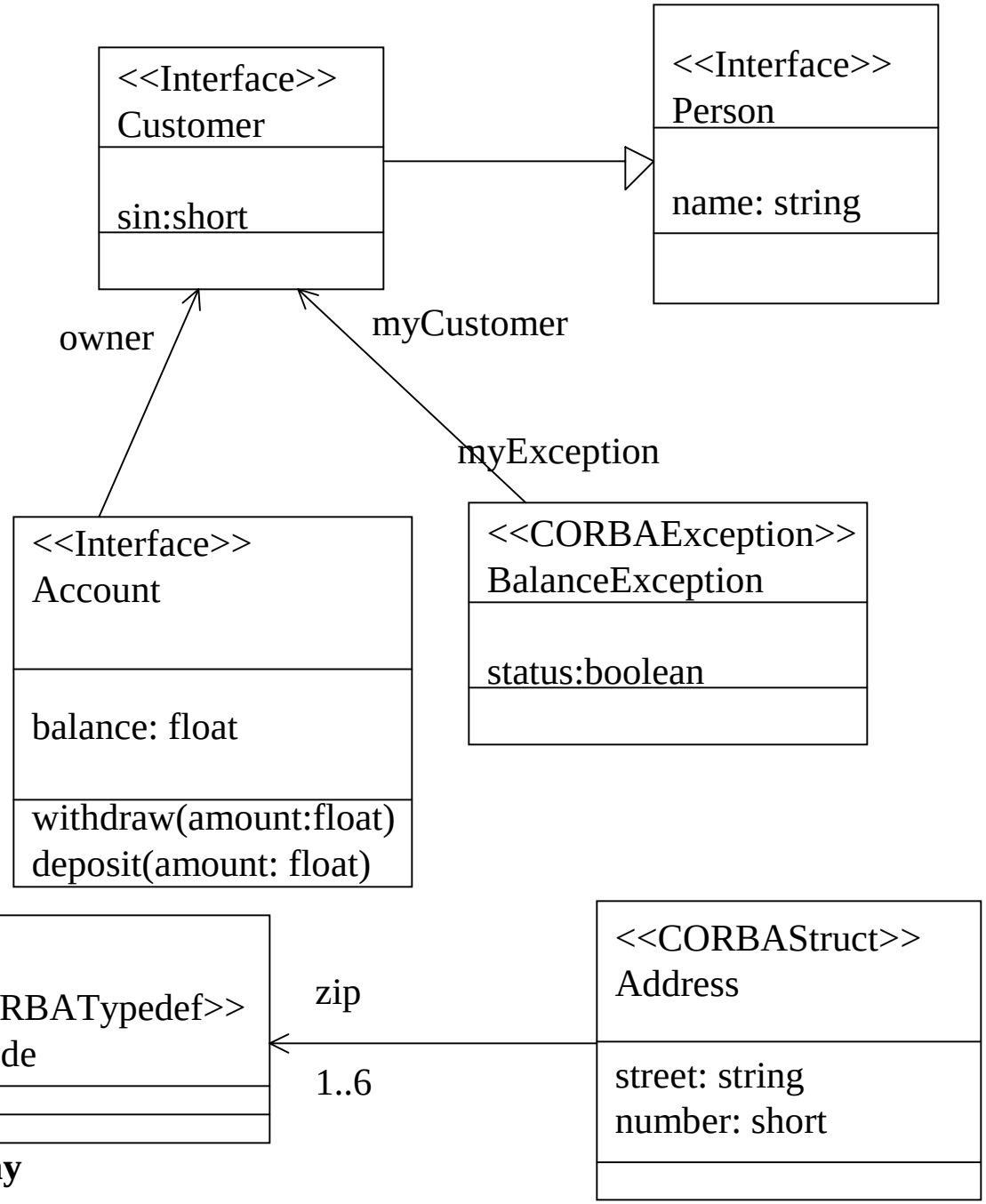
•*Examples*

```
module myModule {
 interface Person {
   attribute string name;
 };

 interface Customer::Person {
    attribute short sin;
  };

 exception BalanceException {
   boolean status;
   Customer  myCustomer;
 };

interface Account {
 attribute float balance;
 attribute Customer owner;
 void deposit(float amount);
 void withdraw(float amount)
        raises BalanceException;
};
typedef short ZipCode;
struct Address {
   string street;
   short number;
   ZipCode zip[6]; //single-dimension array
 };
```

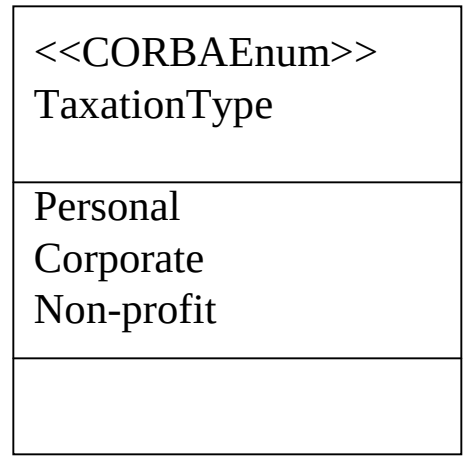


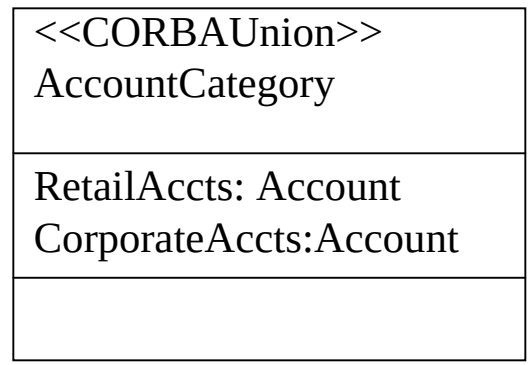
•*Examples (ctd)*

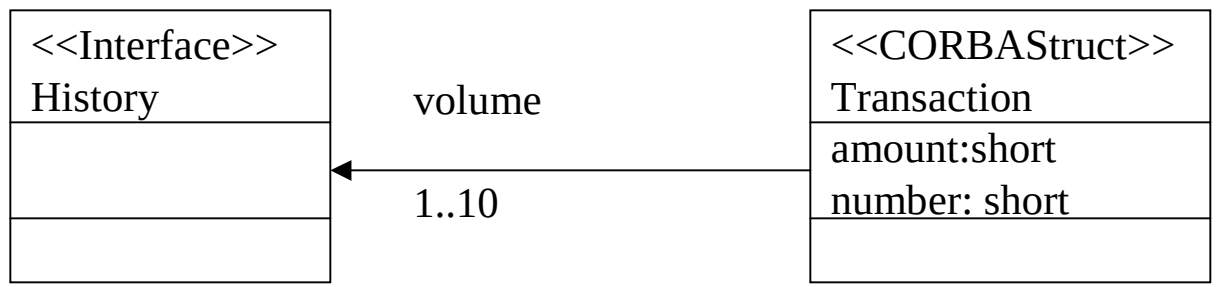
```
enum TaxationType {
    Personal,
    Corporate,
    Non-profit
  };


 union AccountCategory  switch(short) {
      case 0: Account RetailAccts;
      case 1: Account CorporateAccts;
   };


interface History {};
struct Transaction {
  long amount;
  short number;
  sequence <History,10> volume; // sequence definition
 };
}
```







4. Refinement of UML models-IDL Generation

-Necessary step for systematic IDL generation consists of ***refining*** the UML design model based on UML profile for CORBA

÷Refinement involves deriving a concrete UML model by adding type information (for attributes and operations), and exceptions, forward references and include.

÷Based on the refined UML model, IDL code can be generated automatically using tools such as Rational Rose.

Refinement Rules

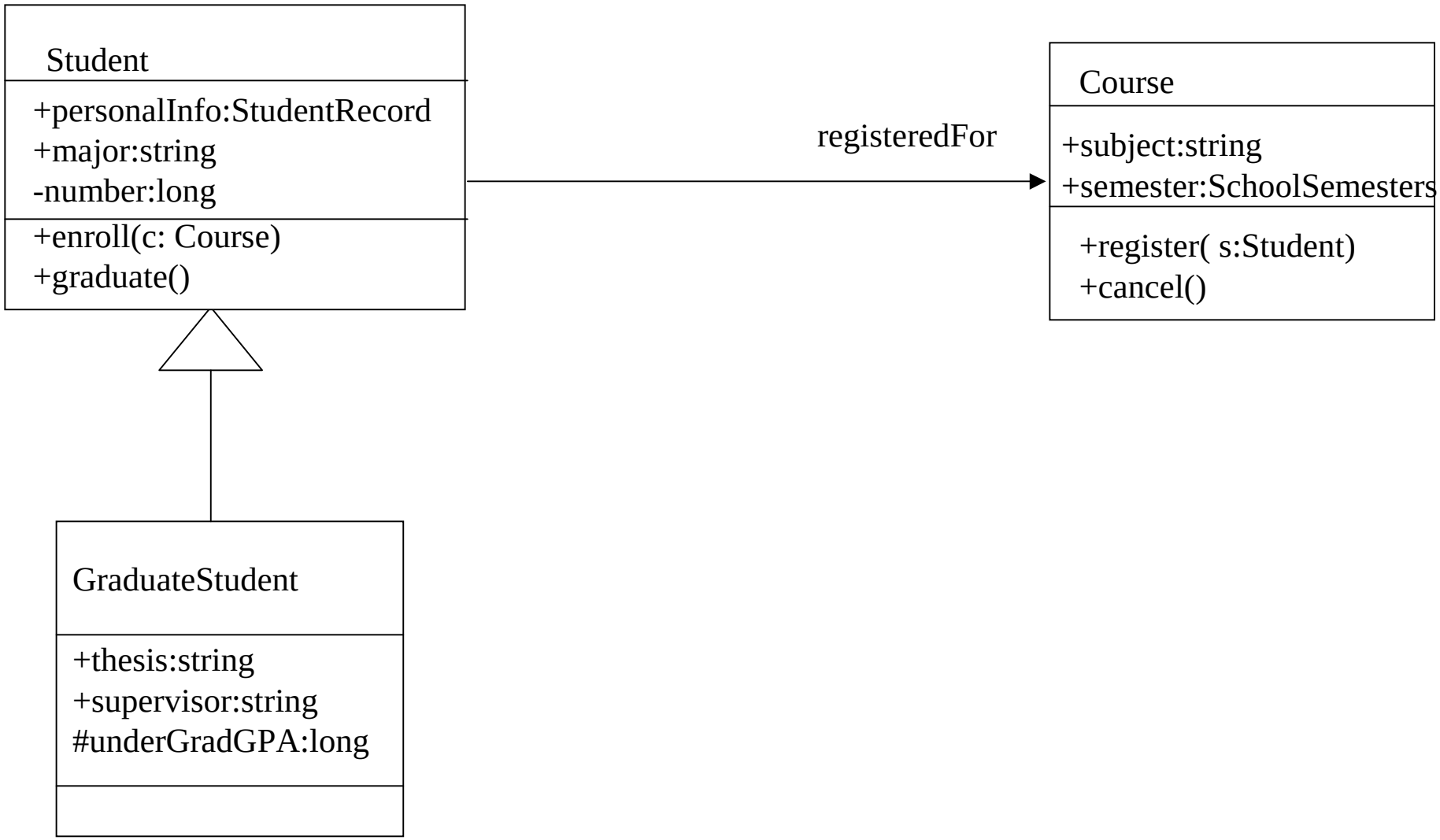
1. *Define for each class one or several interfaces that support the public features (e.g. attributes, operations) of the class. Protected and private features remain in the class.*

2. *Convert the relationships between classes into equivalent relationships between interfaces. Optionally eliminate the implementation classes for clarity.*

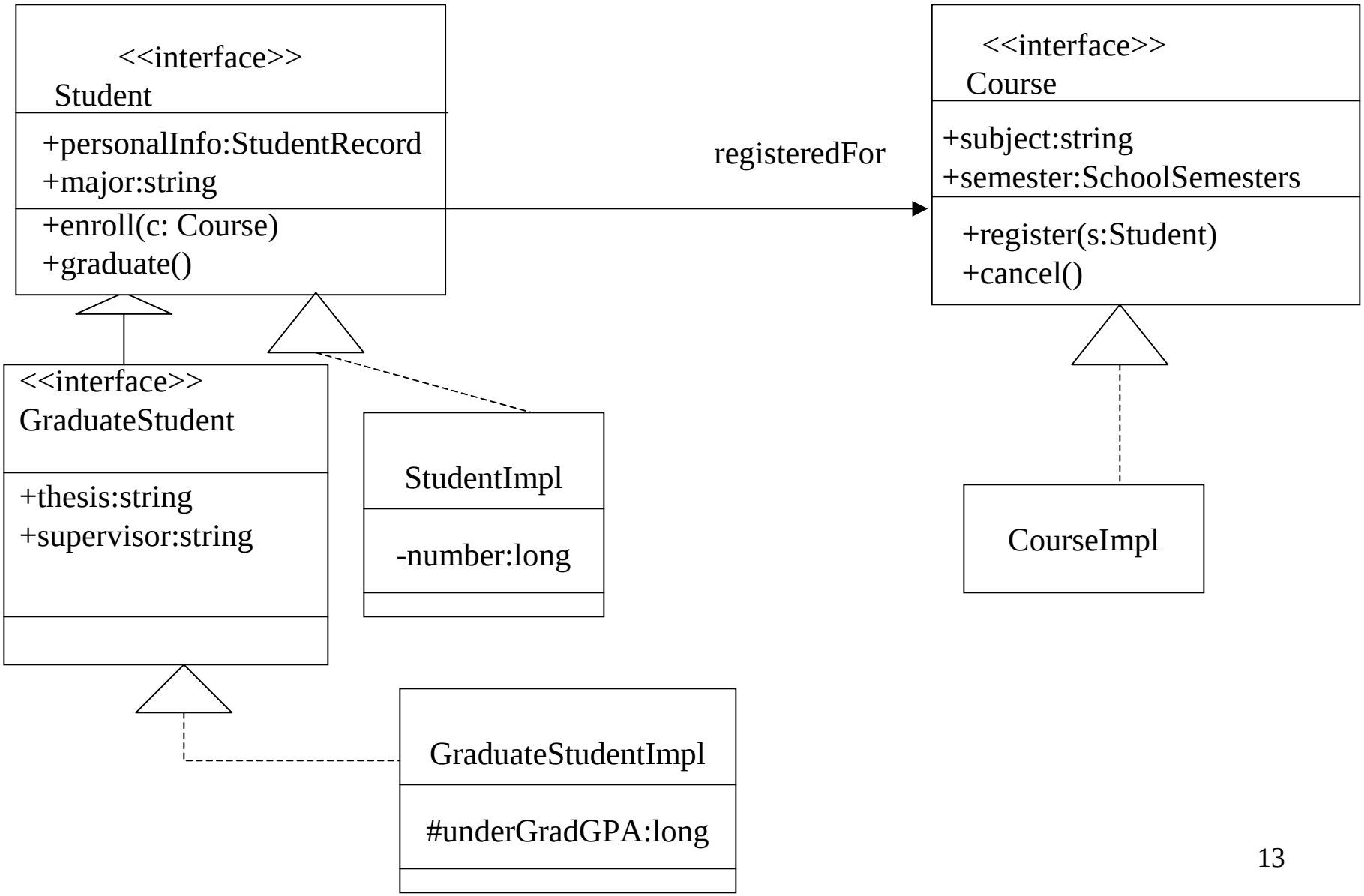
3. *Define precisely operations and attributes by specifying their types and signatures. Define precisely corresponding types and exceptions using CORBA stereotypes whenever applicable.*

Example

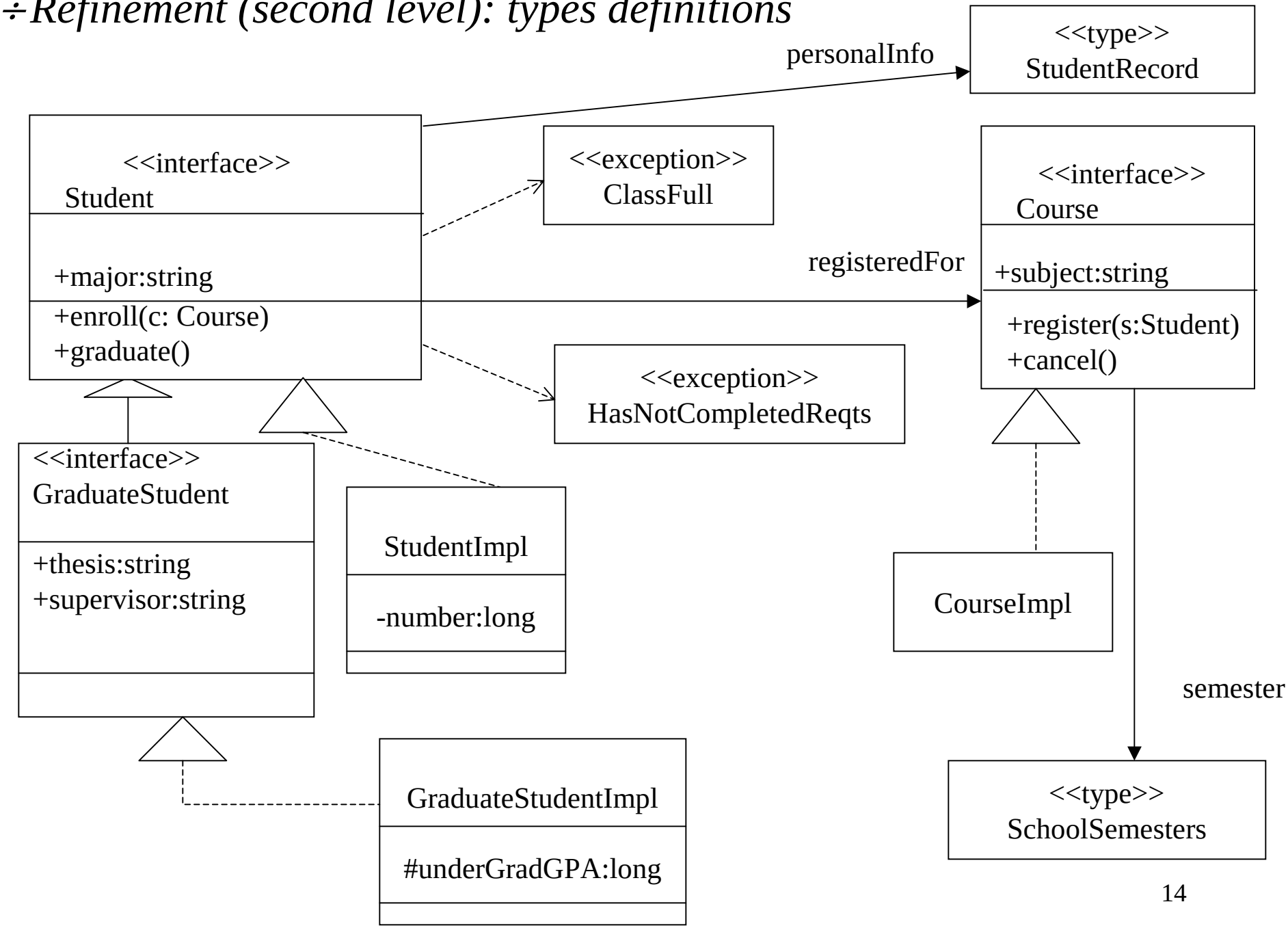
÷Design Diagram



÷*Refinement (first level): separation of interfaces and implementations*



÷*Refinement (second level): types definitions*



÷*Refinement (third level): CORBA profile/model simplification*