

Implementation Techniques

- Design versus construction
- Creating class definitions
- Container/collection classes
- Exceptions and error handling
- Order of implementation



Design versus Construction

- **Boundary somewhat blurred**
 - Prototype/experiment during design
 - Revise design during coding
- **But be careful ...**
 - Can be a slippery slope
 - Try to maintain distinction
 - Complete design before coding
- **Iterative process (A/D/C/T cycles)**

Creating Class Definitions

- **Code from design class diagram**
 - Automatic with CASE tool
- **Design features to map to code**
 - Simple attributes
 - Operations
 - Associations

Simple Attributes

Map to class data member

- **Attribute name**
 - Data member name
- **Attribute type**
 - Data member type
- **Initial value**
 - Constructor initializer list entry
- **Accessor/mutator methods**
 - GetAttrib(), SetAttrib(val)

Operations

Map to class member function

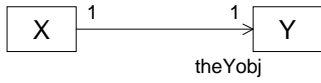
- **Operation name**
 - Member function name
- **Return value and parameters**
 - Member function arguments, return
- **Canonical operations (automatic?)**
 - No-argument & copy constructors
 - Destructor
 - Assignment operator

Associations

Map to class data member(s)

- **Role name (if navigable)**
 - Data member name
- **What about data member type?**
 - Containment (value/reference)
 - Multiplicity (one, fixed, unbounded)
 - Order (sorted or not)
 - Qualified (accessed by "key")

Simple Association



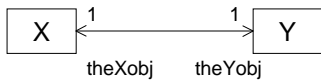
```

// value
class X
{
  ...
  Y theYobj;
}
  
```

```

// reference
class X
{
  ...
  Y* theYobj;
}
  
```

Bi-directional Navigation



```

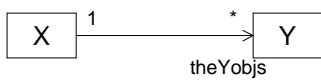
#include "y.h"
class X
{
  ...
  Y* theYobj;
}
  
```

```

#include "x.h"
class Y
{
  ...
  X* theXobj;
}
  
```

See any problem here?

Container/Collection Classes



```

// value
class X
{
  ...
  list<Y> theYobjs;
}
  
```

```

// reference
class X
{
  ...
  list<Y*> theYobjs;
}
  
```

STL Containers for Associations

- **Unordered**
 - STL vector, list, deque
- **Ordered**
 - STL set, multiset
- **Qualified** (see page 365)
 - STL map, multimap
- **Containment by reference**
 - Use smart pointers, clone operation?

Exceptions and Error Handling

- **Operation status return**
 - Caller checks success/failure
- **Precondition checking**
 - Assertions (#include <cassert>)
 - C++ exception mechanism
 - Model in collaboration diagram??
- **Postcondition checking**
 - Same mechanisms



Order of Implementation

- **Least coupled first?**
 - Basic building blocks
 - Test with scaffold (driver) code?
- **Subsystems in parallel?**
 - Maximize team benefit
 - Implement stub methods for early use?
