Entity vs. Value, Modules, Hidden Implementation, Interface Specification

CS 247

University of Waterloo

cs247@uwaterloo.ca

May 19, 2017
Overview

1. Move Constructors, Assignment Operators

2. Entity vs. Value Objects

3. Modules
   - Header Guard
   - Forward Declaration
   - Makefiles

4. Hidden Implementation

5. Interface Specification
   - Queue
lvalue and rvalue

lvalue: a variable that has a permanent address in memory
rvalue: temporary values or literals, which are not lvalue
Consider the following code:

```c
int a = 1;
int b = 2;
int *pa = &a;
```

Classify the following as either rvalue or lvalue:

- a
- a+b
- 1
- *pa
- pa
A move constructor constructs a new object whose value is equal to an existing object, but does not preserve the value of the existing object.
struct Node {
    int data;
    Node * next;
};
Move Constructors

Node::Node(Node && other):
data{other.data}, next{other.next} {
    other.next = nullptr;
}

Move assignment is similar to the move constructor, except that the destination of the move already exists.
Node & Node::operator=(Node && other) {
    swap(other); // implemented in last tutorial
    return *this;
}
Entity vs. Value Objects

Entity:
1. Each object has a distinct identity
2. Objects with the same attributes are not equal
3. Example: UW students

Value:
1. Simply represents a value of an ADT
2. Objects with the same attribute values are considered to be identical
3. Example: EmailAddress
Design

- Operations on copying
- Computations
- Virtual Functions and Inheritance
- Mutability
A Header File typically has all of the declarations associated within a specific class or library. By looking at the header file, you can see all of the provided methods without having to look at the code to see how it works.

class MyClass{
public:
    MyClass(); // constructor
    int myFunction(); // method
    ...
private:
    int myInt; // private data members
    ...
    int myHelperFunction(); // private functions
    ...
}
File A.h
#include "C.h"
class A: public C {
    ...
};

File B.h
#include "C.h"
class B: public C {
    ...
};

File D.h
#include "A.h"
#include "B.h" // redefinition of class C
class D {
    A a;
    B b;
};
A Header Guard prevents a header from being imported multiple times. This is necessary when a header is imported by more than one file in a compilation unit (in this example - toy.h is imported by both main.cpp and by kid.cpp).

```c
#ifndef MYCLASS_H
#define MYCLASS_H

class MyClass {
    ...
};

#endif
```
Include Dependencies

File A.h
ifndef A_H
#define A_H
#include "B.h"
class A {
    B * pb;
};
#endif

File B.h
ifndef B_H
#define B_H
#include "A.h"
class B {
    A * pa;
};
#endif
A forward declaration notifies the compiler that data or function will be declared/defined in the future.
Forward Declaration vs. include

When to include and when to forward declare?
1. in B.h
   class B: public A {...};
2. in C.h
   class C: { A myA; };
3. in D.h
   class D: { A * myA; };
4. in D.cc
   void D::foo() {myA->method();};
5. in E.h
   class E { A foo(A x); };
6. in E.cc
   A E::foo() {
       A a;
       return a;
   };
Makefiles ensure that you don’t re-compile any code you don’t need to - a proper Makefile will only compile:

- Code that has changed (i.e. the source code is newer than the compiled code)
- Code that depends on code that has changed

Using the -MMD flag, the compiler will automatically figure out dependencies for you.
CXX = g++-5
CXXFLAGS = -std=c++14 -Wall -MMD
SOURCES = $(wildcard *.cc)
OBJECTS = ${SOURCES:.cc=.o}
DEPENDS = ${OBJECTS:.o=.d}
EXEC=main

${EXEC}: ${OBJECTS}
    ${CXX} ${CXXFLAGS} ${OBJECTS} -o ${EXEC}
    -include ${DEPENDS}
clean:
    rm *.o main
.PHONY: clean
Hidden Implementation

The PImpl (Pointer to Implementation) idiom is a way of hiding the structure of your class so that someone who reads your header file won’t be able to tell how it stores its data members. This is done by putting all of your members into an “Impl” struct, and pointing to an “Impl” instance within your class. The Impl itself is coded within your .cpp file, which is typically not shown to the client programmer.
The PImpl Idiom

Goal: hide implementation from readers

- put all members into “Impl” struct as nested private struct
- put a pointer to “Impl” struct as a private field

Example: secretNumber.h secretNumber.cc
Interface Specification

Interface Specification is a standard and concise way to describe what a piece of code (function, class, module, etc.) does. Ideally, a spec should describe all the necessary details without being too verbose.

- **requires**: preconditions that must be true before a function is called
- **modifies**: set of things that this function modifies
- **ensures**: *how* this function modifies things
- **returns**: what this function returns
class IntQueue {
   // Specification Fields:
   // back: most recently inserted item
   // front: least recently inserted item
public:
   IntQueue();
   ~IntQueue();
   bool empty();
   const int size();
   void enqueue(int elem);
   int front();
   void dequeue();
The End