Brief Notes on Object-Oriented Software Design and Programming with C++

OpenSees Developer Workshop
Pacific Earthquake Engineering Research Center
NEESit

August 15, 2006

Gregory L. Fenves
Objectives of Notes

• Introduction to topics in software engineering
• Describe data abstraction and modularity
• Demonstrate how objects represent data and operations on data
• Provide two examples (not related to OpenSees)
Problem of Software Design

• Complex problems
• Requirements change
• Unconstrained vs. constrained programs
• Collaborative development process
Data Abstraction

• Abstract data types describe the behavior of data.

• Specification of behavior is distinct from implementation of behavior

• Abstract data type defines:
  – Set of objects of the data type
  – Operations that are specified for objects in set
Example of ADT: Vector

- Vector is mathematical quantity
- Specification includes operations such as:
  - Define vector
  - Magnitude of vector
  - Addition of two vectors
  - Multiplication by scalar
  - Dot product of two vectors
#include"vector.h"

void main ( void )
{
    Vector v1(4,1.0), v2(4,2.0); // v1 initialized to 1; v2 to 2
    Vector s1, s2, v3; // vectors of undefined size.
    float d;

    s1 = v1.vAdd(v2); // addition with vAdd operator
    s2 = v1 + v2; // addition with overloaded + operator

    Vector v4(4,10); // create vector, initial to 10.
    d = v4*v1; // inner product with overloaded *

    v2[0]=v2[0]+v2[1]; // example of index operation
    v2+=v1; // compound assignment, v2=v2+v1
}
Specification of Vector Class

// ADT for Vector in vector.h
class Vector {
public:
    Vector ( int sz=3, float val=0.0); // default to 3D
    Vector ( const Vector& ); // copy constructor
    ~Vector ( void ); // destructor
    Vector& operator= ( const Vector& w ); // assignment
    Vector& operator= ( float s ); // assign vector constant
    float vMag ( void ) const;
    Vector vAdd ( const Vector& w ) const;
    Vector vMult ( float s ) const;
    float vDot ( const Vector &w ) const;
    int vGetSize ( void ) const;

    // Overloaded operators
    Vector operator+ ( const Vector& w ) const; // add
    Vector operator- ( const Vector& w ) const; // subtract
    Vector operator* ( float s ) const; // multiply
    Vector operator/ ( float s ) const; // divide by scalar
    float operator* ( const Vector& w ) const; // dot product

    // Subscript operators
    float& operator[] ( int i ); // LH side
    const float& operator[] ( int i ) const; // RHS

    // Compound assignment operators
    Vector& operator+= ( const Vector& w ); // add to object
    Vector& operator-= ( const Vector& w ); // subtract from object
    Vector& operator*= ( const float s ); // multiply by scalar
    Vector& operator/= ( const float s ); // divide by scalar

    // Equality operations
    int operator== ( const Vector& w ) const;
    int operator!= ( const Vector& w ) const;

private:
    float *vec;
    int size;
};
Object-Oriented Software Design

• Abstraction
• Hierarchy
• Encapsulation
• Concurrency
• Persistence
Example of Software Design

• Represent structural beams with different types of materials
• Illustrate abstraction principles
• Show benefit of dynamic binding: associating functions with objects depending on class of object
• Not the same classes as in OpenSees even though the class names are similar.
Material is a class that represents properties of materials used in structural beam. Objects of class Material have at least one operation, which is to determine the modulus of elasticity.

Object modeling notation for a class:

```
<table>
<thead>
<tr>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public operations</td>
</tr>
<tr>
<td>Private data/operations</td>
</tr>
</tbody>
</table>
```
Details of specifics of materials can be provided by providing *subclasses*.

*Material* is a base class (superclass). *SteelMaterial* and *ConcreteMaterial* are derived classes (subclasses) in the inheritance hierarchy. This is an example of single inheritance. Inheritance is also called an “is-a” relationship.
Decide on specific representation for concrete classes (i.e. not abstract classes):

- Material
  - getE()

  - ConcreteMaterial
    - getE()
    - getFc()
      - Fc

  - SteelMaterial
    - getE()
    - getFy()
      - E
      - Fy

Material is an abstract class since no instances will be created from it.
C++ class declaration for material classes:

class Material  
{  
    public:  
        virtual double getE ( void ) const = 0;
        virtual ~Material (void);  
};

class SteelMaterial : public Material  
{  
    public:
        SteelMaterial ( double f, double e=29000 );
        virtual double getE ( void ) const;
        virtual double getFy ( void ) const;

    private:
        double E; // Modulus of elasticity
        double Fy; // Nominal yield stress
};

class ConcreteMaterial : public Material  
{  
    public:
        ConcreteMaterial ( double f );
        virtual double getE ( void ) const;
        virtual double getFc ( void ) const;

    private:
        double Fc; // Compressive strength
};
Material class implementation:

```cpp
// Default destructor
Material::~Material ( void ) { }

// SteelMaterial methods
SteelMaterial::SteelMaterial ( double f, double e) {
    if ( e > 0 )
        E = e;
    else
        errorExit("SteelMaterial", "Invalid modulus of elasticity.");

    if ( f > 0 )
        Fy = f;
    else
        errorExit("SteelMaterial", "Invalid yield strength.");
}
double SteelMaterial::getE ( void ) const { return E; }
double SteelMaterial::getFy ( void ) const { return Fy; }

// ConcreteMaterial methods
ConcreteMaterial::ConcreteMaterial ( double f ) {
    if ( f > 0 )
        Fc = f;
    else
        errorExit("ConcreteMaterial","Invalid compressive strength");
}
double ConcreteMaterial::getE ( void ) const {
    return 57.0*sqrt(Fc*1000); // per ACI, normal weight concrete
}
double ConcreteMaterial::getFc ( void ) const { return Fc; }
```
Decide on representation of beam sections:

![Class diagram]

RectSingleRCBeamSection is a specialized class.
Beam class specifications:

class BeamSection
{
    public:
        virtual double rigidity ( void ) const = 0;
        virtual double flexCap ( void ) const = 0;
        virtual ~BeamSection ( void );
    
    protected:
        virtual double getIcr ( void ) const = 0;
    
    private:
        SteelSection* aSteelSec;
        const SteelMaterial* aSteelMat;
    
};

class SteelBeamSection : public BeamSection
{
    public:
        SteelBeamSection ( void );
        SteelBeamSection ( const SteelMaterial& b );
        SteelBeamSection ( const SteelMaterial& b, const SteelSection& a
            virtual double rigidity ( void ) const;
            virtual double flexCap ( void ) const;
            virtual void setSteelSection ( const SteelSection& a );
                SteelMaterial* aSteelMat;
    
    private:
        SteelSection* aSteelSec;
        const SteelMaterial* aSteelMat;
    
};

class RConcreteBeamSection : public BeamSection
{
    public:
        virtual double rigidity ( void ) const = 0;
        virtual double flexCap ( void ) const = 0;

    protected:
        const SteelMaterial* aSteelMat;
        const ConcreteMaterial* aConcreteMat;
}
class RectRConcreteBeamSection : public RConcreteBeamSection
{
public:
  virtual double rigidity ( void ) const = 0;
  virtual double flexCap ( void ) const = 0;
  virtual void setWidth ( double w );
  virtual void setDepth ( double h );
  virtual void setEffectiveDepth ( double d );
  virtual double getWidth ( void ) const;
  virtual double getDepth ( void ) const;
  virtual double getEffectiveDepth ( void ) const;

protected:
  RectRConcreteBeamSection ( const SteelMaterial& a,
                           const ConcreteMaterial& b,
                           double w = 0, double h = 0, double d = 0 );
  virtual double getIcr ( void ) const;

private:
  double width;
  double depth_h;
  double depth_d;
};

class RectSingleRConcreteBeamSection : public RectRConcreteBeamSection
{
public:
  RectSingleRConcreteBeamSection ( const SteelMaterial& a,
                                   const ConcreteMaterial& b,
                                   double w = 0, double h = 0, double d = 0, double A = 0 );
  virtual double rigidity ( void ) const;
  virtual double flexCap ( void ) const;
  virtual void setAs ( double A );
  virtual double getAs ( void ) const;

private:
  double As;
};
For different steel section shapes, develop class hierarchy to provide specific representations:

```
SteelSection
getI()
getZ()
I,Z

SteelSection
getI(), getZ(),
getDepth(), getWidth()
h, b
```
Steel section class specifications:

class SteelSection {
    public:
        virtual double getZ ( void ) const;
        virtual double getI ( void ) const;
        virtual ~SteelSection ( void );

    protected:
        SteelSection ( double zxx, double ixx );

    private:
        double Z;
        double I;
};

class WFSteelSection : public SteelSection {
    public:
        WFSteelSection ( double zxx, double ixx, double dep th, double width );
        double getDepth ( void ) const;
        double getWidth ( void ) const;

    private:
        double h;
        double b;
};
int main ( void )
{
    // Create material objects
    SteelMaterial a36 = SteelMaterial(36);
    SteelMaterial a60 = SteelMaterial(60);
    ConcreteMaterial f4 = ConcreteMaterial(4);

    // Create a steel WF section
    SteelSection sec1 = WFSteelSection(400,300,12,8);

    // Create beam sections with material only
    SteelBeamSection beam1 = SteelBeamSection (a36);
    RectSingleRConcreteBeamSection beam2 =
        RectSingleRConcreteBeamSection(a60,f4);

    // Set section for steel beam
    beam1.setSteelSection(sec1);

    // Define a singly reinforced concrete beam
    double h = 24;
    beam2setWidth(h/2);
    beam2.setDepth(h);
    beam2.setEffectiveDepth(h-3);
    beam2.setAs(6);

    // Cast upward to test dynamic binding of member functions
    BeamSection *beam3 = dynamic_cast<BeamSection*>(&beam2);

    // Get flexural properties of two beams
    double EI = beam1.rigidity();
    double Mp = beam1.flexCap();
    double EI2=beam3->rigidity();
    double Mn =beam3->flexCap();

    cout << "Steel Beam:  EI=" << EI  << "  Mp=" << Mp << endl;
    cout << "Concrete Beam: EI=" << EI2 << "  Mn=" << Mn << endl;
}
Questions?