OpenSees UniaxialMaterial 
Class Interface 

OpenSees Developer’s Workshop 
August 21, 2001 

Michael H. Scott
UniaxialMaterial Functionality

• Represent \textit{all} one-dimensional material models
  – Stress-strain ($\sigma-\varepsilon$)
  – Force-deformation (s-e)

• Where are these objects used in OpenSees?
  – Truss elements ($\sigma-\varepsilon$)
  – Zero length elements (s-e)
  – Beam section fibers ($\sigma-\varepsilon$), e.g., steel, concrete
  – Resultant beam sections (s-e), e.g., $M-\kappa$

• Calling object (element or other material object) determines appropriate meaning, $\sigma-\varepsilon$ or s-e
UniaxialMaterial ADT

• Base class, UniaxialMaterial, provides its subclasses with a common interface
• The base class defines no data, only provides virtual methods, some with default behavior
• Subclasses provide specific implementations of the UniaxialMaterial interface
• Defines an abstract data type (ADT)
  – Set of objects of the data type
  – Operations for objects in the set
**C++ Class Interface**

```cpp
class UniaxialMaterial : public Material {
public:
    UniaxialMaterial(int tag, int classTag);
    virtual ~UniaxialMaterial();

    virtual int setTrialStrain(double strain, double strainRate = 0.0) = 0;
    virtual double getStrain(void) = 0;
    virtual double getStrainRate(void);
    virtual double getStress(void) = 0;
    virtual double getTangent(void) = 0;
    virtual double getDampTangent(void);
    virtual double getSecant(void);
    virtual int commitState(void) = 0;
    virtual int revertToLastCommit(void) = 0;
    virtual int revertToStart(void) = 0;
    virtual UniaxialMaterial *getCopy(void) = 0;

protected:

private:
};
```

*Can be overridden by subclass*  

*Must be overridden by subclass, “pure virtual”*
Constructor

UniaxialMaterial::UniaxialMaterial(int tag, int classTag): Material(tag, classTag)
{
    // Does nothing
}

• This constructor calls its base class constructor (Material), which in turn calls the TaggedObject and MovableObject constructors

• Subclass constructors will take material parameters (floating point values or other data types) as arguments
Destructor

UniaxialMaterial::~UniaxialMaterial()
{
    // Does nothing
}

• This destructor does nothing

• Subclass destructors will deallocate any dynamically allocated memory
virtual int setTrialStrain(double strain,
                       double strainRate = 0.0) = 0;

- Must be implemented by subclasses
- Takes a strain and strain rate as arguments
- The default value passed for the strain rate is 0.0
- To store the current strain and strain rate (if used)
virtual double getStrain(void) = 0;

• Must be implemented by subclasses
• To return the current value of strain

double
UniaxialMaterial::getStrainRate(void)
{
    return 0.0;
}

• This is the default implementation for strain rate
getStress

virtual double getStress(void) = 0;

\[ s = s(e, \dot{e}) \]

- Must be implemented by subclasses
- To return the current value of material stress including both strain and strain rate (if any) effects
getTangent/getDampTangent

virtual double getTangent(void) = 0;

- Must be implemented by subclasses
- To return the current material tangent

double UniaxialMaterial::getDampTangent(void)
{
    return 0.0;
}

- This is the default implementation for the damping tangent

\[ D_t = \frac{\partial s}{\partial \dot{\varepsilon}} \]

\[ ? = \frac{\partial s}{\partial \dot{\varepsilon}} \]
double
UniaxialMaterial::getSecant(void)
{
    double strain = this->getStrain();
    double stress = this->getStress();

    if (strain != 0.0)
        return stress/strain;
    else
        return this->getTangent();
}

• Default implementation for the material secant

\[ D_s = \frac{S}{e} \]
commitState

virtual int commitState(void) = 0;

• Must be implemented by subclasses
• UniaxialMaterial objects must manage their own history variables
• This method is invoked upon convergence during a time step
• To update the history variables for path dependence (if any) from time \( t_n \) to \( t_{n+1} \)
• To return 0 if successful and a negative number otherwise
virtual int revertToLastCommit(void) = 0;
• To revert to last commit, i.e., at time $t_n$
• For most materials, nothing needs to be done here
• To return 0 if successful and a negative number otherwise

virtual int revertToStart(void) = 0;
• To revert to initial state
• To return 0 if successful and a negative number otherwise
getCopy

virtual UniaxialMaterial *getCopy(void) = 0;

• Must be implemented by subclasses
• To return a pointer to a new object which is a copy of this UniaxialMaterial object
• Invoked by a calling object, which is responsible for deallocating the dynamically allocated memory
sendSelf/recvSelf

virtual int sendSelf(int cTag, Channel &theChannel) = 0;

virtual int recvSelf(int cTag, Channel &theChannel,
                      FEM_ObjectBroker &theBroker) = 0;

• Two methods inherited from MovableObject for parallel processing and database programming
• To return 0 if successfully moved, a negative number otherwise
• Just make them return –1, then we’ll sort it out for you later!
virtual void Print(ostream &s, int flag = 0) = 0;

• A utility method inherited from TaggedObject
• To print relevant information about the material, i.e., class name, tag, and material parameters to the stream object s passed as an argument
• For example:

    s << "ElasticMaterial, tag: " << this->getTag() << endl;
    s << "E: " << E << endl;
Implementation Exercise

Now it’s time to see how it really works …

• Save NewUniaxialMaterial.h/.cpp as J2Material.h/.cpp and add these new files to the VC++ workspace (or Makefile)

• Provide definitions for all the pure virtual methods in the UniaxialMaterial class interface to implement the uniaxial J2 return map algorithm (see handout)

• Use the OpenSees/Tk UniaxialMaterial tester to see if your implementation works!!!