CS 247: Software Engineering Principles

Information Hiding, Immutable ADTs

Readings: Eckel, Vol. 1

Ch. 11 References and the Copy Constructor

Ch. 12 Operator Overloading (operator=)

PImpl Idiom

Encapsulate the data representation in a nested private structure (which can be in a separate file).

```
class Rational {
public:
    Rational (int numer = 0, int denom = 1);
    int numerator() const;
    int denominator() const;
private:
    struct Impl;
    Impl* rat_;
public:
    ~Rational();
    Rational ( const Rational& );
    Rational& operator= ( const Rational& );
};
```

Mutable vs. Immutable Objects

Entity Objects are ***mutable***

- their objects can change value via mutators, other functions

Value-based Objects are often ***immutable***

- their objects cannot change value
- instead, variables of the ADT are assigned a different object

- all data members are private
- no mutator member functions
- assignment and copy are deep copies
- if a data member isn't primitive or isn't an immutable ADT, then make a deep copy
    - of any new value
    - when passing its value to client code.

Summary of ADT Design Decisions

Range of legal values

- Default initial value?
- Construct object from client-provided values?
- `explicit` constructors or allow type conversion?
- Throw exception if constructor or mutator is given an illegal value.

Attributes (virtual data members)

- Accessors, mutators; consistent naming scheme

Value-based ADTs

- Overloaded operators
- Mutable or immutable?

Implementation

- Data representation
- Private implementation (PImpl)?
- Private operators (entity ADT?)
- Deep or shallow copy / equality?
- Essential operators: ours or the compiler's?