Object-oriented programming:
Generalization hierarchies

Topics:
– Generalization, inheritance, and the is-a relation

Generalization

Defn: relationship between class and one or more refined versions of it

Benefits:
– Powerful abstraction for sharing similarities among classes, while preserving their differences
– Inheritance is helpful as a vehicle for reusing code

Note: Not a synonym for inheritance!
– Can inherit from class that is not a generalization
– Can define generalizations that cannot (easily) be implemented using inheritance
Motivation

Recall STL template class list

template <typename T>
class list {
  public:
    bool empty() const;
    void push_front( const T& );
    void push_back( const T& );
    void pop_front();
    void pop_back();
};

Suppose we want to define template class stack by inheriting from list

template <typename T>
class stack : public list<T> {
  public:
    ...
};

Question: Why is this a bad idea?

Accessing Inappropriate Operations

template <typename T> Foo(list<T>* parm) {...}

Legal call

| stack<int> s |
| Foo(s) |

Now Foo can do this

| parm->pop_front(); |

...which doesn’t make sense for a stack.

Design error: A stack is not a list.
Liskov substitution principle (LSP)

Design principle: A derived class should be a logical, consistent extension of its base class

Liskov substitution principle (LSP):
An instance of a class should function as an instance of its base class

LSP helps one use inheritance to implement generalization/specialization (i.e., the is-a relation)

Question

Could we fix the error by having list inherit from stack? Why or why not?