CS 617 Object Oriented Systems
Lecture 6
Classes Implementing Interfaces
Abstract Classes
Open-Closed Principle
Self References (This)
3:30-5:00 pm Mon, Jan 21

Rushikesh K Joshi

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

Outline

1. Classes: Implementing Interfaces
2. Abstract Classes
3. What's Frozen in a Class, and What can Change?
4. This: The Self Reference (A Runtime View)
5. Single Inheritance

Outline

1. Classes: Implementing Interfaces
2. Abstract Classes
3. What's Frozen in a Class, and What can Change?
4. This: The Self Reference (A Runtime View)
5. Single Inheritance

Implementing Interfaces

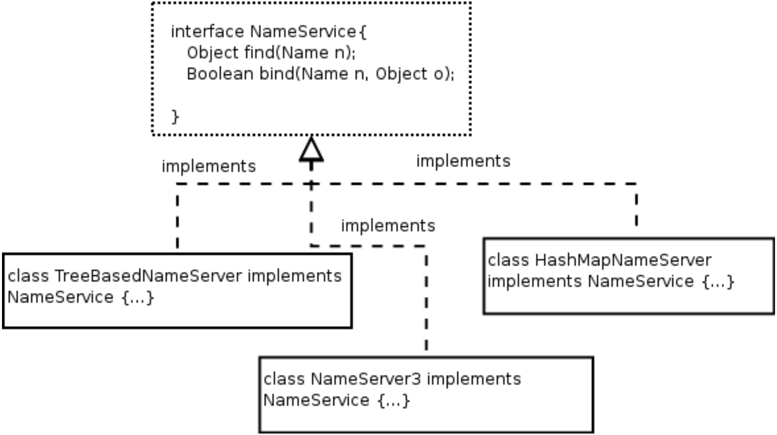
A class provides an implementation of an interface.

An interface: Defines a set of messages through a set of abstract member functions (only the type signatures)

A Class: Provides their implementations, i.e. the method bodies and it exports an interface.

Distinction between Messages and Methods

Interfaces: Explicit Vs. Implicit I



```
interface NameService{
   Object find(Name n);
   Boolean bind(Name n, Object o);

}
```

class TreeBasedNameServer implements NameService {...}

class HashMapNameServer implements NameService {...}

class NameServer3 implements NameService {...}

Interfaces: Explicit Vs. Implicit II

```
Class Account {

public:
int balance():
int withdraw (int amount);
int deposit (int amount);
}
```

- The interface is embedded in class description in the above example.
- Everything kept in public visibility contribute to the interface.

Interfaces: Explicit Vs. Implicit III

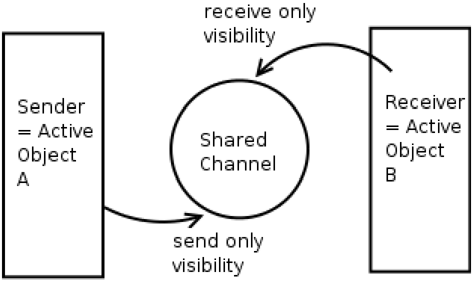
- What's the meaning of exporting a variable through the interface?

Class Implementing Multiple Interfaces I

```
interface Send {
        public void send (int value);
}
interface Receive {
        public int receive ();
}
class UniChannel implements Send, Receive {
int buffer;

  public void send (int value) {buffer=value;}
  public void receive (int value) {return value;}
}
```

Class Implementing Multiple Interfaces II



Separating Interfaces from Classes

- Interfaces provide the protocols for interactions
- Classes act as implementors
- Application classes can be implemented purely in terms of interfaces
- Application classes can be fully unaware of implementation classes
- Implementations classes can vary without having to change the application classes that use implementation classes
- Implementation classes can also be changed without having to change the application classes that use them

Role Modeling with Multiple Interfaces

- Common implementation class
- The implementation class implements many interfaces
- Application classes get restricted contracts through a narrow window of the interface that they know

Outline

1. Classes: Implementing Interfaces
2. Abstract Classes
3. What's Frozen in a Class, and What can Change?
4. This: The Self Reference (A Runtime View)
5. Single Inheritance

Abstract Classes

Partial Implementations

No Implementation == Interface

Abstract Classes as Interfaces I

```
#include<iostream>
using std::cout;
class Channel {
  public:
        virtual void send(int data) = 0;
        virtual int receive () = 0;
};
class BufferedChannel: public Channel {
  int buffer[1];
  public:
        BufferedChannel () { };
        virtual void send(int data);
```

Abstract Classes as Interfaces II

```
        virtual int receive ();
};
void BufferedChannel::send (int data)
                          {buffer[0]=data;}
int BufferedChannel::receive ()
                          {return buffer[0];}
main () {
Channel *c = new BufferedChannel();

   c->send(10);
   c->send(20);
   cout << c->receive() << "\n";
}
```

Abstract Classes Holding Partial Implementation I

```
class Channel {
protected:
  int buffer[10];
  int size;

  public:
        virtual void send(int data) = 0;
        virtual int receive () = 0;
};

class FIFOChannel: public Channel {
  public:
```

Abstract Classes Holding Partial Implementation II

```
        FIFOChannel () { };
        virtual void send(int data);
        virtual int receive ();
};
class LIFOChannel: public Channel {
  public:
        LIFOChannel () { };
        virtual void send(int data);
        virtual int receive ();
};
```

Class Member Visibilities

- Private
  – Committed only Locally
- Public
  – Committed to External Classes
- Protected
  – Committed to Subclasses
- Restricted
  – Committed to a Subset of External Classes

Choosing the right visibilities is important for Contracts
The right level of encapsulation enforces the abstraction
Visibility has impact on refinability

Outline

1. Classes: Implementing Interfaces
2. Abstract Classes
3. What’s Frozen in a Class, and What can Change?
4. This: The Self Reference (A Runtime View)
5. Single Inheritance

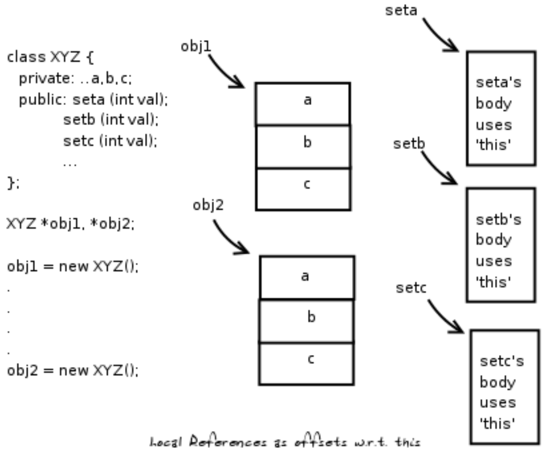
The Open-Closed Principle: Applying to Classes

- Never Change an interface of a class once the class is published.
- The Contract (in our case, the interface) is closed for changes.
- However, the implementation can be changed.
- The implementation is open for refinements
- Unique Ids for component Interfaces

Outline

1. Classes: Implementing Interfaces
2. Abstract Classes
3. What's Frozen in a Class, and What can Change?
4. This: The Self Reference (A Runtime View)
5. Single Inheritance

This: The Self Reference I



```
class XYZ {
  private: ..a,b,c;
  public: seta (int val);
          setb (int val);
          setc (int val);
          ...
};

XYZ *obj1, *obj2;

obj1 = new XYZ();
.
.
.
.
obj2 = new XYZ();
```

Local References as offsets w.r.t. this

This: The Self Reference II

```
#include <iostream>
using std::cout;

class A {
int x;
int y;
public:
        A() {}
        void printthis() { cout << this << "\n";}
        void f(int p, int q) { x=p; y=q;}
        void printstate(){cout<<x<<" "<<y<<"\n";}
};
```

This: The Self Reference III

```
main () {


 A *a1 = new A();
 A *a2 = new A();
 cout << a1 << "\n";
 a1->printthis();

 cout << a2 << "\n";
 a2->printthis();

}
```

Uses of Self References

- For sharing method bodies across instances of a class
- Returning self – e.g. cascaded operations
- Self in parameter passing
- Dynamic Binding in Inheritance Hierarchies

Self Reference Bindings

Is a self reference available in abstract classes?

Is a self reference available in class members?

Is a self reference available in instance members?

Outline

1. Classes: Implementing Interfaces
2. Abstract Classes
3. What's Frozen in a Class, and What can Change?
4. This: The Self Reference (A Runtime View)
5. Single Inheritance

Inheritance for Conceptually Compatible Classes

- Contract Conformance (Conceptual Inheritance)
- Extension
- Refinement

Is Kind Of Relationship

Subclass (Derived Class)

Superclass (Base Class)