Interfaces, Inheritance, Visibilities

Rushikesh K Joshi

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

Abstract Class, a generic Component: Behavior not fully defined

```
class Component {
        public:
                virtual Pinset trigger (Pinset p)=0;
};
```

Properties of Abstract Classes

- Cannot instantiate this class, since it is abstract (not fully implemented)
- Notice the virtual function which is defined to be nil (i.e. 0), This makes it abstract!
- They are allowed to contain implementations for use by their subclasses
- Two main ways to instantiate (but both are not permitted on class Component):
    - Component c;
    - Component *cp = new Component()

Interfaces: Abstract classes, behavior not defined at all

- An abstract class is an interface if it does not contain any implementation
- Cannot instantiate it, since it is abstract
- All functions are declared virtual (in Java this is a default!)

Class Member Visibilities

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

An abstract class

```
class Collection {
public:
        virtual bool insert (Item i)=0;
        virtual Item fetch ()=0;
}
```

- So for, the abstract class is working like an interface

A subclass

```
class Set : public Collection {
public:
        virtual bool insert (Item i)=0;
        virtual Item fetch ()=0;
}
```

- The interface remains the same, Set does not have duplicates

Another subclass

```
class FIFOList : public Collection {
public:
        virtual bool insert (Item i)=0;
        virtual Item fetch ()=0;
}
```

- The interface remains the same, first in first out behavior

Yet another subclass

```
class LIFOList : public Collection {
public:
        virtual bool insert (Item i)=0;
        virtual Item fetch ()=0;
}
```

- The interface remains the same, first in last out behavior
- Now, can we have some common implementation for all subclasses
- .. and push it into the abstract class for automatic use by all?

Modified abstract class

```
class Collection {
int size;
public:
        virtual bool insert (Item i)=0;
        virtual Item fetch ()=0;
}
```

- So, what more can we add here?
- And also, a private variable is not visible to subclasses
- If you make it public, that will be a disaster for the abstraction

Further Modified abstract class

```
class Collection {
protected:
        int size;
public:
        virtual bool insert (Item i)=0;
        virtual Item fetch ()=0;
}
```

- So, what more can we add here?
- And also, a private variable is not visible to subclasses
- If you make it public, that will be a disaster for the abstraction

A subclass of a subclass

```
class OrderedSet : public Set {
public:
        virtual bool insert (Item i)=0;
        virtual Item fetch ()=0;
}
```

- The interface remains the same
- OrderedSet can be used where a Set can be used (remember how ’main’ uses a generic variable!)
- OrderedSet keeps its items in order defined on Items

But how do you define the order among the Items?

```
class Item {
public:
        Item & operator < (Item & i) =0;
}
```

- This is an abstract class
- Users may define their items by inheriting from this class
- The above is a bit difficult concept to understand, we shall continue it in the next class..