Data Structures & Programming

Object Oriented Design

Golnar Sheikhshab

Object-Oriented Design Goals & Principles

- Goals
  - Robustness
  - Adaptability (Evolvability)
  - Reusability
  -
- Principles
  - Abstraction (related to ADT and Interfaces)
  - Encapsulation (related to class)
  - Modularity (related to hierarchy and inheritance)

Abstraction - A List ADT

```
Template <typename E>
class ListInterface{
public:
    /** Adds a new entry to this list.
       @post If successful, newEntry is stored in the list and
       the count of items in the list has increased by 1.
       @param newEntry The object to be added as a new entry.
       @return True if addition was successful, or false if not. */
    virtual bool add(const E& newEntry) = 0;
    /** Documentation... */
    virtual bool removeFirst(const E& elem) = 0;
    virtual bool empty() = 0;
    virtual int count() = 0;
    virtual ~ListInterface() = 0;
    virtual void print() = 0;
};
```

Interface

Does the interface reveal anything about how things are implemented?

Can this interface have an implementation?

How is this interface useful?

Using the interface

```
template <typename E>
class ArrayList: ListInterface<E>{
public:
        ArrayList(int capacity){  body.... }
        bool empty() const{return n_==0;};
        int count() const{return n_;}
        void print() const{ body ...  }
        bool removeFirst(const E& elem){ body ...  }
        bool add(const E& new_entry){ body ...  }
        ~ArrayList() { delete [] listArray_; }
protected:
int findFirst(const E& elem) const { body ... }
        E* listArray_;
        int n_, capacity_; };
```

ArrayList implementation (inside ArrayList.h)

```
ArrayList(int capacity){
        std::cout << "ArrayList construction is running...\n";
        listArray_ = new E[capacity];  //dynamic array
        n_=0;
        capacity_ = capacity;
    }

void print() const{
        for (int i=0; i<n_; i++)
            std::cout << listArray_[i] << " ";
        std::cout << std::endl;
    }
```

ArrayList implementation (inside ArrayList.h)

```
bool removeFirst(const E& elem){
        int index = findFirst(elem);
        if (index==-1){
            std::cout << "removing " << elem << " failed!\n";
            return false;
        }
        for (int i=index; i<n_-1; i++)
            listArray_[i] = listArray_[i+1];
        n_--;
        std::cout << "just removed " << elem << std::endl;
        return true;
    }
```

ArrayList implementation (inside ArrayList.h)

```
bool add(const E& new_entry){
        if (n_+1 > capacity_){ // array is full
            std::cout << "adding " << new_entry << " failed\n";
            return false;
        }
        listArray_[n_] = new_entry;
        n_++;
        std::cout << "just added " << new_entry << std::endl;
        return true;
    }
```

ArrayList implementation (inside ArrayList.h)

```
~ArrayList() {
        std::cout << "ArrayList destruction is running...\n";
        delete [] listArray_;
    }


int findFirst(const E& elem) const{
    for (int i=0; i<n_; i++)
        if (listArray_[i] == elem)
            return i;
    return -1;
}
```

Inheriting even more

SortedArrayList "is a" ArrayList

Inheritance is used for specialization and extension

```
template <typename E>
class SortedArrayList: public ArrayList<E>{
        SortedArrayList(int capacity): ArrayList<E>(capacity) { body ...}
         bool add(const E& elem){ body ... }            // use inheritance for specialization
         bool fastSearch(const E& elem){ body ...}  // use inheritance for extension
         ~SortedArrayList(){ body ... }
protected:
         int getRightPlace(const E& elem){ body ... }
         bool binarySearch(const E& elem, int start_index, int end_index){ body ... }
 };
```

SortedArrayList implementation (inside header file)

```
SortedArrayList(int capacity): ArrayList<E>(capacity) {
        std::cout << "SortedArrayList construction is running...\n";
    }

bool fastSearch(const E& elem){
        return binarySearch(elem, 0, this->n_-1);
    }

~SortedArrayList(){
        std::cout << "SortedArrayList destruction is running...\n";
    }
```

SortedArrayList implementation (inside header file)

```
bool add(const E& elem){
        if (this->n_+1> this->capacity_){
            std::cout << "adding " << elem << " failed!" << std::endl;
            return false;
        }
        int right_place = getRightPlace(elem);
        // shift everything right
        for (int i=this->n_-1; i>=right_place; i--)
            this->listArray_[i+1] = this->listArray_[i];
        this->listArray_[right_place] = elem;
        this->n_++;
        std::cout << "just added " << elem << std::endl;
        return true;
    }
```

SortedArrayList implementation (inside header file)

```
int getRightPlace(const E& elem){
        for (int i=0; i<this->n_; i++)
            if (this->listArray_[i]>= elem)
                return i;
        return this->n_;
    }
```

SortedArrayList implementation (inside header file)

```
bool binarySearch(const E& elem, int start_index, int end_index){
        if (start_index > end_index)
            return false;
        int middle_index = (start_index + end_index)/2;
        if (this->listArray_[middle_index]>elem)
            return binarySearch(elem, start_index, middle_index-1);
        else if (this->listArray_[middle_index]<elem)
            return binarySearch(elem, middle_index+1, end_index);
        else // this->listArray_[middle_index]==elem
            return true;
    }
```

Order of constructors and destructors

compile and run test_ArrayList.cpp and test_SortedArrayList.cpp

Does the constructor of parent class runs first or the constructor of derived class?

How about the destructors?

Access Control

- Public : everybody can access
- Protected: derived classes can access
- Private (default): the class itself and friends can access

- Public inheritance: public members will stay public
- Protected inheritance: public members will become protected
- Private inheritance (default): public & protected members will become private

Virtual methods

Look at intArrayList.h and stringArrayList.h

The method add is virtual in stringArrayList but not in intArrayList

Now look at the code in illustrate_polymorphism.cpp

Which add function is being called in each of the following cases?

```
intArrayList* int_lists[2];
int_lists[0] = new intArrayList(10);
int_lists[1] = new intSortedArrayList(10);

int_lists[0]->add(5);
int_lists[1]->add(5);
```

```
stringArrayList* string_lists[2];
string_lists[0] = new stringArrayList(10);
string_lists[1] = new stringSortedArrayList(10);

string_lists[0]->add("5");
 string_lists[1]->add("5");
```

Virtual methods

Compile and run the illustrate_polymorphism.cpp to verify your answers

To compile

1. Compile the intArrayList and intSortedArrayList as objects
2. Compile the stringArrayList and stringSortedArrayList as objects
3. run

    g++ -o illustrate_polymorphism.o illustrate_polymorphism.cpp int/intSortedArrayList.o string/stringSortedArrayList.o int/intArrayList.o string/stringArrayList.o

To run

./illustrate_polymorphism.o

Virtual destructors

As a rule of thumb, whenever you have any virtual function in a class, you should make a virtual destructor too just to be safe. It's OK if it doesn't do anything.

This is to make sure all the destructor of the derived class will run and consequently the destructor of the parent (all the way up the hierarchy).

static binding and dynamic binding

will the following compile?

```
stringArrayList* string_lists[2];
 string_lists[0] = new stringArrayList(10);
 string_lists[1] = new stringSortedArrayList(10);
 if (string_lists[1]->fastSearch("7"))   // will not compile
        cout << "fast_search found 7.\n";
 else
        cout << "fast_search did not find 7.\n";
```

what if we replace the first line with the following?

```
if (dynamic_cast<stringSortedArrayList*>(string_lists[1])->fastSearch("7"))
```

Reading material

Read chapter 2 of your book