OBJECT ORIENTED PROGRAMMING

Virtual Functions and Abstract Classes

Zuhaib A. Shaikh,
Asst. Prof., CSE Deptt.,QUEST
Web: zuhaib-shaikh.neocities.org

Introduction

- A pointer of base class can be used to point base class object as well as its child class objects
- Dynamic memory objects are very advantageous but sometimes are problematic
- During runtime-polymorphism, the method of corresponding instant is called
  - e.g.

```
class A
{     public:
      void show() { cout<<“A”; }
};
class B: public A
{     public:
      void show() { cout<<“B”; }
};
class C: public A
{     public:
      void show() { cout<<“C”; }
};
```

```
A *Obj;
Obj = new A;
Obj ->show();
delete Obj;
Obj = new B;
Obj ->show();
delete Obj;
Obj = new C;
Obj ->show();
delete Obj;
```

- The solution to this problem is virtual functions

Virtual function

- Methods of base class expected to be redefined (overriding) in child classes
- Virtual functions tends to be called by an object pointer, depending on which object it is pointing called "*Dynamic Binding*"
  - e.g.

```
class A
{     public:
      virtual void show() { cout<<"A"; }
};
class B: public A
{     public:
      void show() { cout<<"B"; }
};
class C: public A
{     public:
      void show() { cout<<"C"; }
};
```

```
A *Obj;
Obj = new A;
Obj ->show();
delete Obj;
Obj = new B;
Obj ->show();
delete Obj;
Obj = new C;
Obj ->show();
delete Obj;
```

- A virtual function (starts with virtual keyword) should be public and not a constructor, not a friend and/or static function

Virtual destructor

- Destructor of base class should have a virtual destructor
- Otherwise, delete operator will always invoke destructor with the type of point object
  - e.g.

```
class A
{    public:
     virtual ~A() { cout<<“A Destroyed”; }
};
class B: public A
{    public:
     ~B() { cout<<“B Destroyed”; }
};
```

```
A *Obj;
Obj = new A;
delete Obj;
Obj = new B;
delete Obj;
```

Virtual class

- A class can be inherited as virtual to child class
- Resolves an ambiguity occurs in multiple inheritance
  - e.g.

```
class Person                                    class Student: public Person
{    protected:                                 {
     int ID;                                    };
};
class Employee: public Person
{
};
class Internee: public Employee, public Student
{
     cout<<ID;                                  //Ambigious error
};
```

- Inheriting class A virtual will create only single copy hence resolves issue

Abstract Class

- Base classes which cannot be instantiated but can be inherited
- Useful when inheritance is compulsory required
- Created with help of pure virtual function
- A method declared with =0 is considered as pure virtual function
  - e.g.

```
class A
{     public:
      virtual void show() = 0
};
class B: public A
{     public:
      void show() { cout<<“B”; }
};
class C: public A
{     public:
      void show() { cout<<“C”; }
}
```

```
A *Obj;
Obj = new A;
Obj ->show();
delete Obj;
Obj = new B;
Obj ->show();
delete Obj;
Obj = new C;
Obj ->show();
delete Obj;
```

Questions