Inheritance

"Introduction, Base and Derived classes, Inheritance Types, Members Accessibility, Derived class Constructors"

Fundamentals of OOPs

Shakirullah Waseeb
shakir.waseeb@gmail.com

Nangarhar University

October 18, 2017

Agenda

1. Understanding Inheritance
   - Introduction
   - The "Is a" and "Has a"
   - Base and Derived classes
   - Accessibility in Inheritance
2. Derived class constructors
3. Overriding member functions
4. Types of Inheritance
   - Single Inheritance
   - Multiple Inheritance
5. Questions and Discussion

Agenda

Introduction

- A powerful feature of OOPs after classes itself
- Creation of new classes from existing classes
- Inherits all capabilities of the parent classes, thereby adding new features and refinements of its own while parents are unchanged
- **Superclass:** the class from which other classes inherits its behavior
- **Subclass:** the class that inherits the properties and methods of another classes
- Why?
  - Permits code reusability: permits the ease of distributing class libraries
  - To save time and money (a written, debugged and tested class need not to be touched again)
  - Helps in the original conceptualization of the programming problem (hierarchical way), and the overall program design
  - Might have no access to source code


Agenda

The "Is a" and "Has a" relationship

- Inheritance
  - Considered an *Is a* class relationship
  - e.g. a Circle is a Shape , an Alumni is a Student
- A class contain objects of another class as its member data
  - Considered as *Has a* class relationship
  - e.g. a Student class has an object of class Counter as its data member


Agenda

The base and derived classes

- **Base class:** a class from which other classes inherits
- **Derived class:** a class which inherits from an other class

Vehicle Example

```
#include <iostream >
using namespace std;

class Vehicle {
      private:
            int model;
            string license;
      public:
            Vehicle  () {
                  model = 0;      license = "";
            }
};

class Car : public Vehicle {
      private:
            int NoOfSeats;
};
```


Agenda

Accessibility in Inheritance

- Knowing, when a *member function* in the **base** class can be used by **objects of the derived class**
- How the compiler handles the accessibility issue
  - **Substituting base class constructors** : if don't specify a constructor, the derived class will use an appropriate constructor from the base class
  - **Substituting base class member function**: if a member class not found in the derived class via the object of which it is being called, it will use the member functions from the base class
- **protected** access specifier: another scope for class members, which can be accessed *within child classes*, and can't be accessible outside the class via objects
- Example on board


Accessibility in Inheritance (public, protected, and private inheritance)

- **public inheritance:**
- **protected inheritance:**
- **private inheritance:**

- When we wants to initialize the base class members, it will probably done by the base class no arguments constructor which is called automatically when we create the child class

Vehicle Example

```
#include <iostream >
using namespace std;

class Vehicle {
      private:
            int model;
            string license;
      public:
            Vehicle  () {
                  model = 0;      license = "";
                  cout« "I am base class constructor";                  }
};

class Car : public Vehicle {
      private:
            int NoOfSeats;
};
```

- What? if when we wants to initialize the derived class member variables by using one argument constructor
- We will get a compilation error


- When we have functions in base class and we wants to redefined those functions in the derived class
- Having same name and signature as that of base class
- Then these functions in the derived class is said to be override functions
- When we need to call the parent class overridden method we can access it with scope resolution operator (e.g. className:: functionName(argumentList))


Agenda

- When a class derived from only one class is known as single inheritance
- This can also be done arbitrary numbers of level
- A class hierarchy results from generalizing common characteristics
  - The more general the class , the higher it is on the chart
  - The more specific the class, the lower it is on the chart

Example: Single Multilevel Inheritance

```
class A {
};
class B: A {
};
class C: A {
};
class D: B {
};
```


Agenda

- We might have situation, when we inherit a class from more than one class
- Syntax for multiple is similar to that of single inheritance as shown in prior slide
- The base classes (A and B) from which C is derived are listed, followed by colon in C's specification; separated by comma

Example: Multiple Inheritance

```
class A {
};
class B {
};
class C: A,B {
};
```


Your Turn: Time to hear from you!

1

[1] https://fensafitters.files.wordpress.com/2013/07/3d095.jpg

References

Robert Lafore
Object-Oriented Programming in C++, 4th Edition .
2002.

Piyush Kumar
Object oriented Programming (Using C++)
http://www.compgeom.com/ piyush/teach/3330