lab_inheritance : Insidious Inheritance

Week #4 – February 12-14, 2020

Welcome to Lab Inheritance!

Course Website: https://courses.engr.illinois.edu/cs225/sp2020

Overview

In this week's lab, you will gain experience with the concept of inheritance. Inheritance in OOP is used when you want to extend a base class to build other derived classes. Inheritance enables us to write more flexible code with minimum code redundancy.

Derived From Base

A derived class is an extension of the base class. The derived class inherits the member functions and member variables of the base class. You are allowed to override functions given by the base class in the derived class. The **virtual** keyword signals that if a derived version of the function exists in the class; then it should be executed first. Pure virtual functions in the base class **must** have an implementation in the derived class.

Exercise 1: You are given the class Building as a base class for class Apartment. Complete the implementation of Apartment by filling in the .h and .cpp files. Pay close attention to virtual functions!

Building.h

```
#pragma once

class Building{
 public:
  Building(double ar,
string add);

  virtual void
setName(string n)=0;
//pure virtual function

  virtual string
getName();
 private:
  double area_;
  string address_;
};
```

Building.cpp

```
#include "Building.h"

Building::Building(dou
ble ar, string add){
 area_ = ar;
 address_ = add;
}

string
Building::getName(){
 return "SIEBEL";
}
```

Apartment.h

```
#pragma once

class Apartment : public Building{
 public:
  Apartment(double ar, string add, int c);
  bool addResidents(int r);
                                  //YOUR CODE HERE



private:
  string name_;
  int capacity_;
};

```

Apartment.cpp

```
#include "Apartment.h"

Apartment::Apartment(double ar, string add, int c)
:__________________________           //YOUR CODE HERE
{
name_= "ISR";
__________________________            //YOUR CODE HERE
}
bool Apartment::addResidents(int r){
 if (capacity_ - r >=0) {
   capacity_ -= r;
   return true;
 } else {return false;}
}
//OVERRIDE ALL VIRTUAL FUNCTIONS BELOW:
```

Polymorphism

One of the key features of class inheritance is that a pointer to a derived class is type-compatible with a pointer to its base class. However, the opposite is not true; while an instance of a derived class can be "polymorphed" into its base class, an instance of the base class cannot be "polymorphed" into an instance of its derived classes.

Exercise 2.1: Which building initialization(s) will cause an error?

building 1 2 3 4

Exercise 2.2: What would be printed in lines **11** through **14**?

main.cpp

```
 1  int main() {
 2    Building* building1 = new Building(300, "306
 3  Wright");
 4    Building* building2 = new Apartment(200, "1111
 5  Nevada", 80);
 6    Apartment* building3 = new Apartment(250, "918
 7  Illinois", 500);
 8    Apartment* building4 = new Building(200, "201
 9  Goodwin");
10
11    std::cout<< building2->getName() <<std::endl;
12    std::cout<< building3->getName() <<std::endl;
13    building2->setName("Busey Evans");
14    std::cout<< building2->getName() <<std::endl;
15
16  }
```

Virtual Destructors

When we have a base class pointer pointing to a derived class object, the base class destructor is used to free the memory of the derived class object. A memory leak may occur if the derived class has data allocated on the heap using a member variable that was not given by the base class. To fix the issue, the base class's destructor should be a virtual destructor.

Exercise 3.1: What will be printed out when main() is run?

Exercise 3.2: How would you fix this code so that no memory leak happens? What will be printed out after the fix?

ab.h

```
class A {
 public:
  A(){
   cout<<"Ctor A ";
  }

  ~A(){
   cout<<"Dtor A ";
  }
};
```

ab.h

```
class B : public A{
public:
  int *i;
  B():A(){
    cout<<"Ctor B ";
    i = new int;
  }
  ~B(){
    cout<<"Dtor B ";
    if(i != NULL){
     delete i;
     i = NULL;
}}};
```

main.cpp

```
int main(){
  A *a = new B();
  B *b = new B();
  cout << endl;

  delete a;
  delete b;
  return 0;
}
```

In the programming part of this lab, you will:

- Explore the classes: Drawable, Shape, Circle, Triangle, and Rectangle, and discover how each fits in the inheritance hierarchy.
- Produce drawings of Truck and Flower

As your TA and CAs, we're here to help with your programming for the rest of this lab section! ☺️