CME 3202 - Concepts of Programming Languages

Laboratory Worksheet #11

Laboratory Aim

In this laboratory section, you are expected to exercise on different functionalities of **Object-Oriented Programming.**

Inheritance

"Inheritance" is the central theme in OOP. Inheritance allows new classes defined in terms of existing ones, i.e., by allowing them to inherit common parts. C++ class can be derived from an existing class, which is then its parent, or base, class.

The data defined in a class definition are called *data members* of that class, and the functions defined in a class definition are called *member functions (*methods*)* of that class. Some or all of the members of the base class may be inherited by the derived class, which can also add new members and modify inherited member functions. The syntactic form of a derived class is:

```
class derived_class_name : derivation_mode base_class_name
{data member and member function declarations};
```

The derivation_mode can be either public or private. In a public-derived class, the public and protected members of a base class are also public and protected, respectively. In a private-derived class, both the public and protected members of the base class are private. Private members of a base class are inherited by a derived class, but they are not visible to the members of that derived class and are therefore of no use there. Consider the following example:

```
class base_class {
      private:
          int a;
          float x;
      protected:
          int b;
          float y;
      public:
          int c;
          float z;
};
class subclass_1 : public base_class {. . .};
class subclass_2 : private base_class {. . .};
class subclass_3 : private base_class {
      base_class :: c;
      . . .
}
```

In `subclass_1`, `b` and `y` are protected, and `c` and `z` are public. In `subclass_2`, `b`, `y`, `c`, and `z` are private. No derived class of `subclass_2` can have members with access to any member of `base_class`. The data members `a` and `x` in `base_class` are not accessible in either `subclass_1` or `subclass_2`. Unlike `subclass_2`, instances of `subclass_3` can access `c`. As far as `c` is concerned, it is as if the derivation had been public.

Exercise 1

Apply the following steps, and implement a sample for inheritance in C++.

Step 1

Open your browser, and open the website - ***http://ideone.com/***. Select C++ as your programming language, and paste the code in the following.

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

class single_linked_list {
```

```
	private:
		class node {
			public:
				node *link;
				int contents;
		};
		node *head;
	public:
		single_linked_list() {head = 0;};
		void insert_at_head(int data)
		{
			 node* n = new node();
		 n->contents = data;
		 n->link = head;
		head = n;
		};
		void insert_at_tail(int data)
		{
			node* n = new node();
			n->contents = data;
			if(empty())
			{
				head = n;
			}
			else
			{
				node* temp = head;
				while(temp->link){
					temp = temp->link;
				}
				temp->link = n;
		}};
		int remove_at_head()
		{
			int ret_val = head->contents;
			head = head->link;
			return ret_val;
		};
		int empty(){
			return ( this->head == 0 );
		};

};

class stack : private single_linked_list {
	public:
		stack() {}
		void push(int value) {
			single_linked_list :: insert_at_head(value);
		}
		int pop() {
			return single_linked_list :: remove_at_head();
		}
		int empty() {
			return	single_linked_list:: empty();
		}

};

int main(int argc, char** argv) {
		stack* s = new stack();
		cout<<s->empty()<<endl;  // boşsa 1 yazdırıyor boş değilse 0
		s->push(1);
		s->push(2);
		s->push(3);
		cout<<s->pop()<<endl;
		cout<<s->empty()<<endl;
}
```

Step 3

Click ***"Run"*** button, and paste the output in the following.

```
1
3
0
```

Question 1

```
int main(int argc, char** argv) {
            stack* s = new stack();
            s->insert_at_head(1);
}
```

Change the code in main function with the code given above. Paste your output in the following and explain the result.

```
prog.cpp:66:22: error: 'single_linked_list' is not an accessible base of 'stack'
  s->insert_at_head(1);

This is because the single_linked_list is derived as "private".
```

Question 2

Change the stack code as queue implementation. Enqueue the numbers 1, 2 and 3, and then dequeue and print an item. Paste your code and your output in the following.

```
class queue : private single_linked_list {
        public:
                queue() {}
                void enqueue(int value) {
                        single_linked_list :: insert_at_tail(value);
                }
                int dequeue() {
                        return single_linked_list :: remove_at_head();
                }
                int empty() {
                        return   single_linked_list:: empty();
                }
};


1
2
3
```

Question 3

What is the purpose of private derivation in `class stack : private single_linked_list ?`

```
It's for preventing the access to the parent methods from stack.
```

Question 4

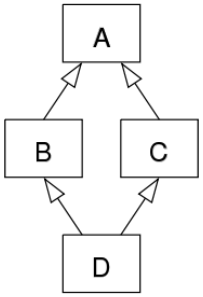
What is the double colon (::) used for in `single_linked_list::insert_at_head(value)` ?

It's used for calling the methods of superclass.

Multiple Inheritance

If a new class is a subclass of a single parent class, then the derivation process is called **single inheritance**. If a class has more than one parent class, the process is called **multiple inheritance**. C++ supports multiple inheritance.

Diamond Problem



The diamond problem occurs when two super classes of a class have a common base class. The "diamond problem" is an ambiguity that arises when two classes B and C inherit from A, and class D inherits from both B and C.

Exercise 2

Apply the following steps, and implement multiple inheritance in C++.

Step 1

Open your browser, and open the website - ***http://ideone.com/*** . Select C++ as your programming language, and paste the following code.

```
#include <iostream>

  using namespace std;

  class Animal{
  public:
      Animal(){
          cout << "animal constructor" << endl;
       }
     int age;
     void walk(){
         cout << "animal walks" << endl;
     }
 };

 class Tiger :public Animal{
 public:
     Tiger(){
         cout << "constructor of tiger" << endl;
     }
 };

 class Lion :public Animal{
 public:
     Lion(){
         cout << "constructor of Lion" << endl;
     }
 };

 class Liger : public Tiger , public Lion{
 public:
     Liger(){
         cout << "constructor of Liger" << endl;
     }
 };

 int main()
 {
     Liger lg;
```

```
    lg.walk();
    return 0;
}
```

Step 3

Click ***"Run"*** button, and paste the output in the following. Explain the reason of the output and correct the code fragment.

```
prog.cpp:40:9: error: request for member 'walk' is ambiguous
   lg.walk();
     ^~~~
```

The compiler can't resolve which base class we should resolve from, so we must specify which base class to resolve the symbol from.

```
lg.Tiger::walk();
lg.Lion::walk();
```

Interface

Interfaces are used as an alternative to multiple inheritance. They provide some of the benefits of multiple inheritance but have fewer disadvantages. For example, the problems of **diamond inheritance** are avoided when **interfaces**, rather than multiple inheritance, are used.
Java directly supports only single inheritance. However, it includes a kind of abstract class, called an interface, which provides partial support for multiple inheritance. Java doesn't allow multiple inheritance to avoid the ambiguity caused by it.

Step 1

Open the website - ***http://ideone.com/*** . Select Java as your programming. Click "Run" button, and paste the output in the following.

```
import java.util.*;
import java.lang.*;
import java.io.*;

/* Name of the class has to be "Main" only if the class is public. */
interface AnimalEat {
   void eat();
}
interface AnimalTravel {
   void travel();
}
class Animal implements AnimalEat, AnimalTravel {
   public void eat() {
      System.out.println("Animal is eating");
   }
   public void travel() {
      System.out.println("Animal is travelling");
   }
}
class Demo {
   public static void main(String args[]) {
      Animal a = new Animal();
      a.eat();
      a.travel();
```

```
    }
}
```

```
Animal is eating
Animal is travelling
```

Question 1

When we need to use extends and implements in Java?

We need to use extends when we are deriving from a class, and we need to use implements when we are implementing an interface to a class.

Question 2

Compare the multiple inheritance of C++ with that provided by Interfaces in Java.

C++ allows multiple inheritance, but Java does not. To solve problems in a similar way, Java supports Interfaces, which does not suffer from the Diamond problem.