ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH



**ADVANCED PROGRAMING (CO2039)**

**FINAL REPORT**

GVHD: Trương Tuấn Anh

SVTH: Trần Nguyễn Phước Nhân

MSSV: 1952893

Tp. Hồ Chí Minh, Tháng 8/2021

Mục lục

[**TOPIC I: OBJECT-ORIENTED PROGRAMING AND ITS PROPERTIES** 3](#_Toc79680542)

[**TOPIC II: OBJECT-ORIENTED PROGRAMING BETWEEN JAVA AND C++** 6](#_Toc79680543)

[**TOPIC III: OBJECT-ORIENTED PROGRAMING vs FUNCTIONAL PROGRAMING** 7](#_Toc79680544)

# **TOPIC I: OBJECT-ORIENTED PROGRAMING AND ITS PROPERTIES**

1. **What is Object-Oriented Programing (OOP)?**

* Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.
* OOP works based on the objects that developers want to handle rather than the logic to handle them. The approach is well-fitted for programs that are big, complex, frequently updated or maintained.

1. **What is the structure of OOP?**

* Classes are user-defined data type that acts as the blueprints for some objects, attributes, and methods.
* Objects are instances of class created with defined data. It can be related to real-world objects or just an abstract entity. Whenever a class is initialized, the description is the only object that is defined.
* Methods are functions that are defined inside a class to describe behaviors of an object. Programmers use methods for reusability or keeping functionality encapsulated inside object.
* Attributes are defined in the class template to represent the state of an object.

1. **What are the principles of OOP?**

* **Encapsulation:** This principle by which we hide information that we don’t want the users to see and show users what we want them to see instead.
* **Abstraction:** This principle is used to hide implementation details. It hides and handles complex tasks from users. Users will know what it can does but does not need to know how it was implemented.
* **Inheritance:** This principle is to rewrite the exact same logic in the base class to the derived class that requires that logic. For example, vehicles are the base class, then cars would be the derived class which inherits some basic properties of vehicles like fuel, tires, engines, …
* **Polymorphism:** This principle means “many shapes”. In OOP, it means that the same entity (function or object) behaves differently in different scenarios. For example, “walking” in living creatures may vary from no legs, one legged, two-legs, … in which functions differently in different situations.

1. **What are the key advantages of Polymorphism in C++?**

Polymorphism in C++ is mainly divided into two types:

* **Compile time Polymorphism:** Achieved through functions overloading or operator overloading.
* **Function Overloading:** When there exist multiple functions with the same name but different parameters then those function are said to be ***overloaded***. Function can be overloaded by change in numbers of arguments or change in type of arguments.
* **Operator Overloading:** When we need to assign an operator for specific tasks of our defined data type, we use operator overloading to handle that task, minimizing the time for getting and setting new data, minimize the space required for that task.
* **Runtime Polymorphism:** Achieved through function overriding.
* **Function Overriding:** This occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.

In OOP, polymorphism is defined as how to carry out different processing steps by a function having the same messages. Polymorphism treats objects of related classes in a generic manner.

* Advantages of Polymorphism in C++:
* It allows programmers to reuse, evaluate and execute the program, modules, forms written once. In certain aspects, they can be repeated.
* You may use the odd variable name to stock variables of different types of data, such as Int, Float, etc.).
* Polymorphism tends to reduce the pairing of multiple functionalities.
* Method overloading can be extended to builders that allow multiple ways of initializing class objects. It helps you to identify several builders for managing various forms of initializations.
* Method overriding functions along with inheritance without the need for re-compilation to allow code reuse of existing groups.

Example code of Polymorphism in C++:

Assume we have an interface of List Data Type, name IList, which share common properties such as: ***add, remove, search****.* And also, SLinkedList for Singly Linked List derived from IList. IList has pure virtual functions that can be seen below. And SLinkedList has its detailed definition of each method in SLinkedList.h.

#ifndef ILIST\_H

#define ILIST\_H

using namespace std;

template<class T>

class IList{

public:

virtual void add(T e)=0; // for adding at the end

virtual void add(int index, T e)=0; // for adding at specific index

virtual T removeAt(int index)=0; //remove at specific index

virtual bool removeItem(T item)=0; // remove item

virtual T& get(int index)=0; // search for item

virtual bool empty()=0;

virtual int size()=0;

virtual void clear()=0;

};

SLinkedList.h

*template <class T>*

*T SLinkedList<T>::removeAt(int index){*

*Node \*prev = head;*

*Node \*cur = head->next;*

*int idx = 0;*

*while (idx < index){*

*prev = prev->next;*

*cur = cur->next;*

*idx++;*

*}*

*prev->next = cur->next;*

*cur->next = 0;*

*count--;*

*T backup = cur->data;*

*delete cur;*

*if (index == count - 1)*

*tail->next = prev;*

*return backup;*

*}*

*template <class T>*

*void SLinkedList<T>::add(int index, T e){*

*if ((index < 0) || (index > count))*

*throw std::out\_of\_range("The index is out of range!");*

*Node \*prev = head;*

*int ite = -1;*

*while (ite < index - 1) {*

*prev = prev->next;*

*ite += 1;*

*}*

*Node \*current = prev->next;*

*prev->next = newNode;*

*newNode->next = current;*

*if (index == count)*

*tail->next = newNode;*

*count += 1;*

*}*

*…. Some other implementations*

*…..*

*#include IList.h*

*template <class T>*

*class SLinkedList : public IList<T>*

*{*

*….. //some implementation*

*public:*

*void add(T e);*

*void add(int index, T e);*

*T removeAt(int index);*

*bool removeItem(T item, void (\*removeItemData)(T) = 0);*

*bool empty();*

*int size();*

*void clear();*

*T &get(int index);*

*}*

*template <class T>*

*void SLinkedList<T>::add(T e)*

*{*

*Node \*node = new Node(e, tail);*

*tail->next->next = node;*

*tail->next = node;*

*count += 1;*

*}*

*template <class T>*

*T &SLinkedList<T>::get(int index)*

*{*

*Node \*prev = head;*

*int cnt = -1;*

*while (cnt < index - 1)*

*{*

*prev = prev->next;*

*cnt++;*

*}*

*return prev->next->data;*

*}*

# **TOPIC II: OBJECT-ORIENTED PROGRAMING BETWEEN JAVA AND C++**

C++ is a both procedural and object-oriented programming language while Java is the object-oriented programming language. We can outlined the comparison by below table:

|  |  |  |
| --- | --- | --- |
| Properties | C++ | Java |
| Object-oriented? | Is a procedural and object-oriented language | Is a completely object-oriented language |
| Access specifiers | Public, protected, private.  Default is private if not declared. | Public, protected, private and default. The default is “default”. |
| Operator overloading | C++ allow operator overloading.  E.g: << >> + - \* == -= += \*, … | Java does not allow operator overloading |
| Multiple inheritance | C++ supports. One class can be inherited from many class | Java does not support |
| Interface | C++ does not have interface | Java supports interface to control the behavior of classes |
| Leaf class | C++ does not have | Java has leaf class which has “final” flag for a class, then no other classes can inherit from it. |
| Friend class | C++ supports friend class | Java does not support friend class |
| Organization inside class | C++ can group attributes or methods with the same access specifier.  C++ can declare prototypes inside .h files and implementation in .cpp files | Java require declare the access specifier first unless it will set to “default”  Java declare and define methods inside one file .java |
| Protected access scope | Can access in defined class and derived class |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# **TOPIC III: OBJECT-ORIENTED PROGRAMING vs FUNCTIONAL PROGRAMING**