

VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING



Advanced Programming (CO2039)

Final Report

Advisors: Trương Tuấn Anh
Student's name: Lê Đức Cầm
Student id: 1952588

HO CHI MINH CITY, August 2021



Contents

1	Key advantages of Polymorphism in C++	2
1.1	Definition	2
1.2	Important advantages with explanation	2
1.3	Code demonstration	3
2	Object Oriented Programming in C++ and Java	5
2.1	The similarities	5
2.2	Main differences	6
2.3	Conclusions	6
3	Object Oriented Programming and Functional Programming	7
3.1	Comparison	7
3.1.1	Similarities	7
3.1.2	Differences	7
3.2	Some code	7
3.3	Pros and Cons of each programming paradigm	8
3.4	Conclusions	8
4	References	9



1 Key advantages of Polymorphism in C++

1.1 Definition

★ The word polymorphism means that the same entity can behave in different ways, in simple words, we can say that functions or objects which share the same name may exist in many forms and have different characteristics. For example, car, bicycle, train, motorbike, plane,... all of them are considered as vehicle.

★ In C++, polymorphism is mainly divided into two types:

- Compile time Polymorphism: a function is called at the time of program compilation (function overloading and operator overloading).
- Runtime Polymorphism: functions are called at the time of program execution, in other words, it occurs when a child class redefines for one of the member functions of the parent class (function overriding).

1.2 Important advantages with explanation

★ **Same interface could be used for creating methods with different implementations.** For example, more than one method with the same name but with different number or type of arguments.

★ **Reduces the volume of work in terms of distinguishing and handling various objects.** Considering class "SUV" and "Sedan" both of them have method "honk()", because they are just cars so we can just reuse that method from class "Car" and we don't need to spend too much time on it.

★ **Make it easier for extending systems and applications.** Because polymorphism enables us to write clear, clean, simply-structured code which knows only a little (or maybe nothing) about the types we are dealing with.

★ **Provides easier maintenance of application.** If the bug occurs somewhere, we just have a look into the arguments of the method which cause the bug and determine where it may happen easily.

1.3 Code demonstration

★ We can see that after creating an object, depends on the input arguments, the suitable constructor will be called => flexibility.

Function overloading - constructor overloading example:

```
using namespace std;
// Le Duc Cam - 1952588
class Car{
protected:
    unsigned int license_plate;
    string type;
public:
    Car(){
        this->license_plate = 0;
        this->type = "";
    }
    Car(int license_plate){
        this->license_plate = license_plate;
        this->type = "";
    }
    Car(int lp, string s){
        this->license_plate = lp;
        this->type = s;
    }
}
```

★ Every car has its own kind of horn so there will be various sound when driver press the horn. So we will make a little abstract method for honking => scalability

Function overloading - method overloading example:

```
    }
    Car(int lp, string s){
        this->license_plate = lp;
        this->type = s;
    }
    void honk(){
        cout<<"honkk"<<'\\n';
    }
    void honk(int type){
        cout<<"hank";
    }
    void honk(string s, int id){
        cout<<s<<'\\n';
    }
    void honk(int *a){
        cout<<*a<<'\\n';
    }
    bool operator ==(Car obj){
```

★ There are hundreds type of car with thousand of versions each, so how can we consider two models are the same? operator "==" mostly used for comparing two numbers, two operands, but now we can apply polymorphism to solve our concern on comparing two car models by using

operator overloading => efficiency

Operator overloading example:

```
bool operator ==(Car obj){  
    if (obj.license_plate == this->license_plate) return true;  
    else return false;  
}
```

★ Because various type of car will have different sound when they honk, so if there exist a different of honking sound, we just need to rewrite honk function for a specific type of car => safe time
Function overriding example:

```
};  
class Sedan:public Car{  
public:  
    void honk(){  
        cout<<"bip bip\n";  
    }  
};
```

· Let's test our small program a little bit:

```
int main(){  
    Car c1(1111);  
    Car *c2 = new Car(1234);  
    Sedan s;  
    cout<<"Car1's honking sound: ";  
    c1.honk("bembem",5);  
    cout<<"sedan car's sound: ";  
    s.honk();  
    int b = 99;  
    int *a = &b;  
    cout<<"test polymorphism func with passing a pointer: ";  
    c1.honk(a);  
    cout<<"Car1 and car2 are the same? ";  
    if(c1 == *c2 ) cout<<"true";  
    else cout<<"false";  
    return 0;  
}
```

· Here the result:

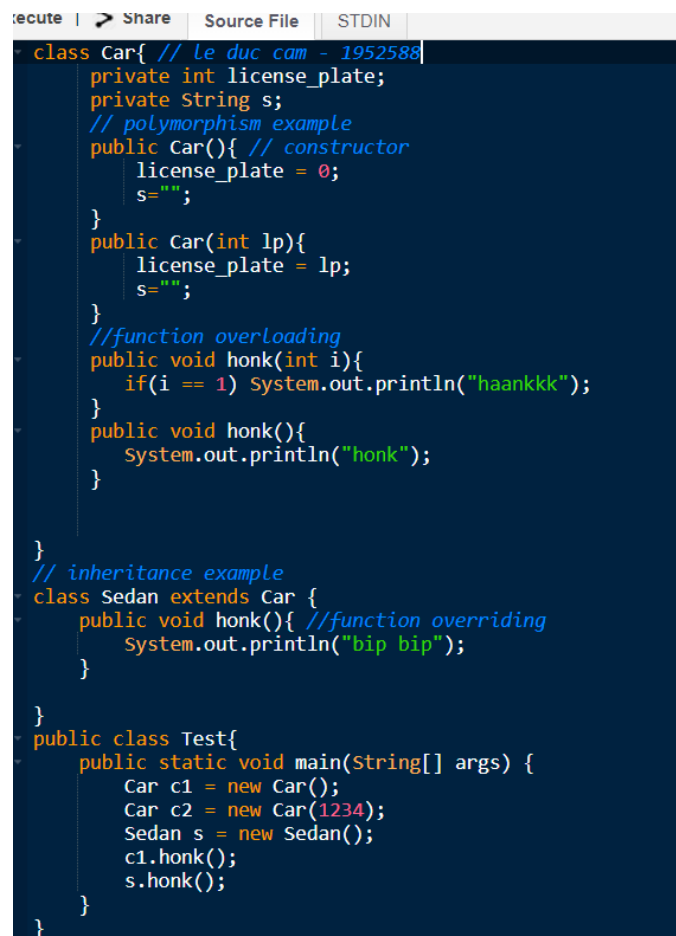
```
car1 and car2 are the same? false  
PS C:\Users\HP\Desktop\OOP> cd "c:\Users\HP\Desktop\OOP\  
Car1's honking sound: bembem  
sedan car's sound: bip bip  
test polymorphism func with passing a pointer: 99  
Car1 and car2 are the same? false
```

⇒ Polymorphism is a strong tool and amazing features of OOP, developer should take advantage of it.

2 Object Oriented Programming in C++ and Java

2.1 The similarities

★ Object Oriented Programming in C++ and Java have vast applications in the industry.
★ Both of them ensure and provide several OOP features such as abstraction, encapsulation, single inheritance, polymorphism, static and dynamic binding.
We have seen an OOP example in C++ above, so there is a picture about how to do same thing with Java right below:



```
recute | ➤ Share | Source File | STDIN
class Car{ // le duc cam - 1952588
    private int license_plate;
    private String s;
    // polymorphism example
    public Car(){ // constructor
        license_plate = 0;
        s="";
    }
    public Car(int lp){
        license_plate = lp;
        s="";
    }
    //function overloading
    public void honk(int i){
        if(i == 1) System.out.println("haankkk");
    }
    public void honk(){
        System.out.println("honk");
    }
}

// inheritance example
class Sedan extends Car {
    public void honk(){ //function overriding
        System.out.println("bip bip");
    }
}

public class Test{
    public static void main(String[] args) {
        Car c1 = new Car();
        Car c2 = new Car(1234);
        Sedan s = new Sedan();
        c1.honk();
        s.honk();
    }
}
```

2.2 Main differences

No.	C++	Java
1	supports both operator overloading and method overloading	only supports method overloading
2	supports manual object management with "the help of "new" and "delete"	depends on built-in automatic garbage collection
3	Flexible model with constant protection available	The cumbersome model encourages weak encapsulation
4	provide multiple inheritance	doesn't provide multiple inheritance
5	Methods can be called by value or by reference	Supports only call by value

2.3 Conclusions

C++ and Java were created in different period of time, with different purposes (but also share a lot of things in common) and each of them has its own merits and demerits. C++ is suitable for computer graphics, real-time applications, low-level and system programming or gaming applications while Java has been a "super star" in building normal users applications like networks, web, desktop and mobile apps, TV applications...

→ So we should evaluate beforehand the pros and cons, the uniqueness of each language based on our demand of new applications and the solutions for our problems, then make a decision on which is the most suitable for our developing app to achieve the best result.



3 Object Oriented Programming and Functional Programming

3.1 Comparison

3.1.1 Similarities

- ★ They are both developed to solve a certain programming task.
- ★ Each of them has strategies, principles, and rules which help developer to build a software.

3.1.2 Differences

Object Oriented Programming	Functional Programming
Focus on the concept of objects	Focus on function evaluation
Data can be change	Data is immutable
Imperative programming model	Declarative programming model
Does not support parallel programming	Parallel programming is available
Execution in a particular order	Statements can be executed in any order
Uses loops for iteration	Uses recursion for iteration
Fundamental elements: objects and methods	Fundamental elements: variables and functions
Has three access specifiers	Does not have any access specifier
Secured programs	Non-secured programs
Easy to add new data and functions	Hard to add new data
The methods may have side effect	Pure functions do not have side-effects

3.2 Some code

- ★ This is an example of implementing quick sort and merge sort to show how haskell is quite different compare with C++.


```
qsort :: Ord a => [a] -> [a]
qsort [] = []
qsort (x:xs) =
    (qsort smaller) ++ [x] ++ (qsort larger)
    where
        smaller = [a | a <- xs, a <= x]
        larger = [b | b <- xs, b > x]

merge :: Ord a => [a] -> [a] -> [a]
merge [] [] = []
merge [] z = z
merge z [] = z
merge (x:xs) (y:ys) =
    if x > y then [y] ++ (merge (x:xs) ys) else [x] ++ (merge xs ([y]++ys))

msort :: Ord a => [a] -> [a]
msort [] = []
msort [x] = [x]
msort x = do
    let mid = length x `div` 2
    let y = take mid x
    let z = drop mid x
    merge (msort y) (msort z)
```

3.3 Pros and Cons of each programming paradigm

★Functional Programming Pros and Cons:

- (+)Using pure and transparent functions leads to reliable results without side effects.
- (+)It uses a more declarative style that focuses more on what needs to be done and less on how to do it, with an emphasis on efficiency and optimisation.
- (+)Easy to handling and debug
- (-)It requires much knowledge,skill and even maths to combining pure functions in a hard task.
- (-)It is a relatively new paradigm and sometimes it is not so easy to find documentation or information compared to Object-oriented Programming.
- (-)Have to develop a completely different mindset on thinking mostly everything recursively.

★ Object-oriented Programming Pros and Cons:

- (+)Objects and methods are explicit and easy to understand.
- (+)Use an imperative style, in which the code is read like a simple set of instructions.
- (+)The code is reusable and really easy to maintain
- (+)It allows for parallel development
- (-)May lead to unspecified and wanted results in the case that a parallel code that would have access to a common critical section and create unexpected output after execution.
- (-)Its methods can have side effects and may put an impact on processors.
- (-)It can be inefficient due to massive amount of bloated, unnecessary code

3.4 Conclusions

It is very difficult to clearly determine which one is better because the concepts of the two programming paradigms are quite different and their purpose are also diverse. Like the OOP comparison between Java and C++ above, there will be no winner. Depends on demands, environments, our purposes and available technologies, we can choose an approach that will our development process productive and effortless.



4 References

- Provided slides on Bkel
- <https://sdacademy.dev/>
- <https://www.geeksforgeeks.org/>
- <https://www.tutorialspoint.com/>
- <https://www.guru99.com/cpp-vs-java.html>
- <https://nguyenvanhieu.vn/series/huong-doi-tuong-cpp/>