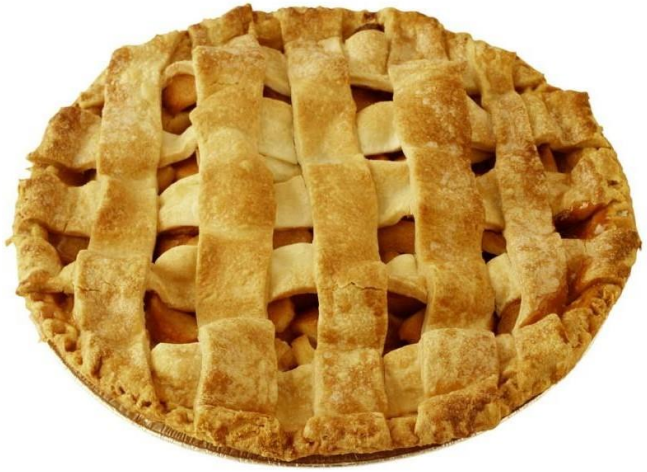
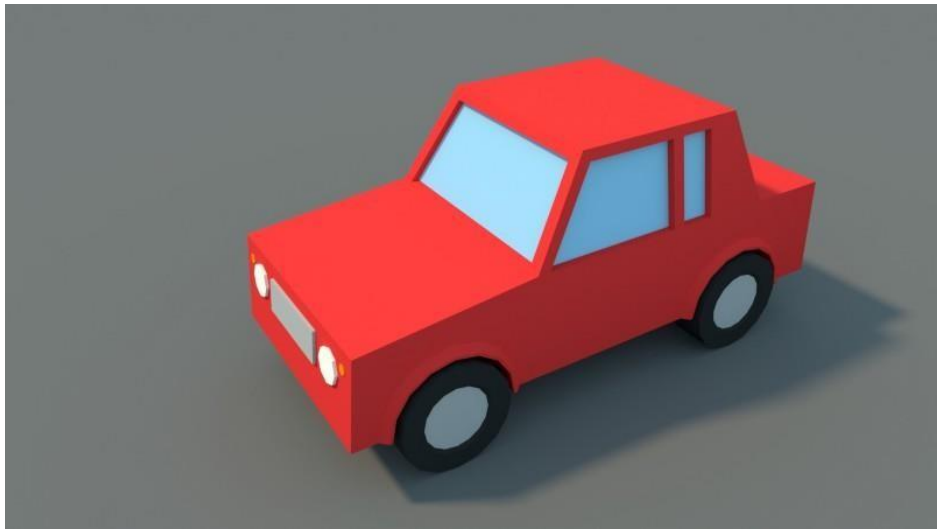
Lab 3

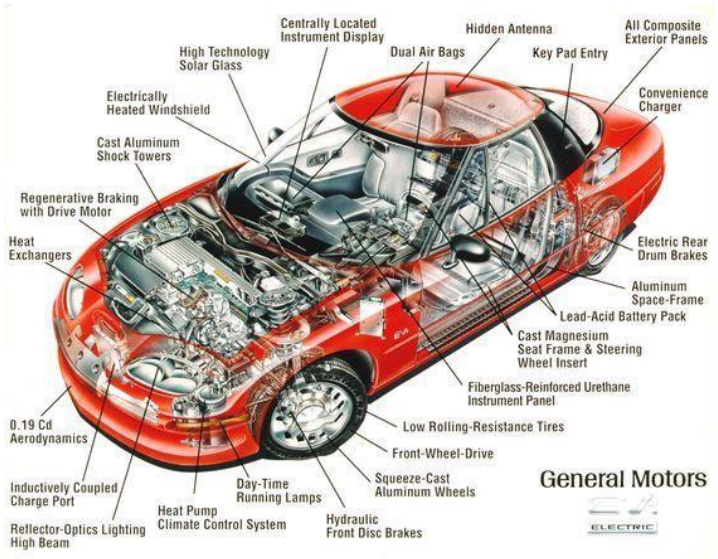
OOP – Abstraction, Polymorphism, Inheritance, Encapsulation



1. Abstraction

- Abstraction is a process where you show only “relevant” data and “hide” unnecessary details of an object from the user.

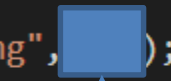




Abstraction -example

```
#define YEAR 2020
class Person{
private :
    string first_name;
    string last_name;
    int birthDay; //20191231
public :
    Person(string f,string l,int b){
        first_name=f;
        last_name=l;
        birthDay=b;
    }
    void showName(){
        cout<<"name is "<<first_name<<' '<<last_name<<endl;
    }
    void showAge(){
        int old=YEAR-(birthDay/10000)+1;
        cout<<"Korean age is "<<old<<endl;
    }
};
```

```
int main(){
    Person p("dongkyu","sung",20191231);
    p.showName();
    p.showAge();
    return 0;
}
```

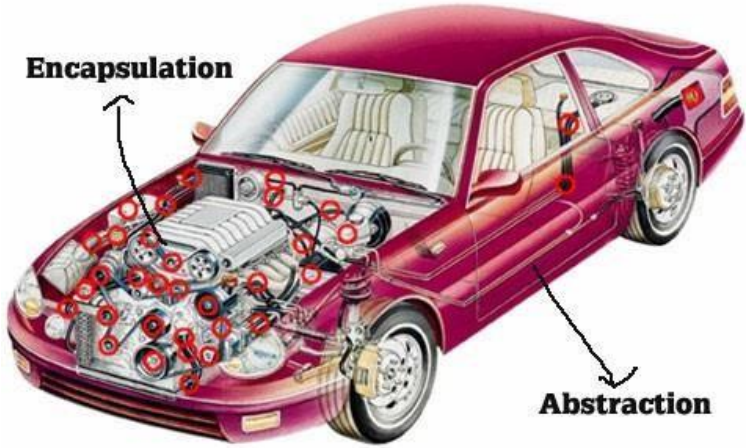


20191231

Output:
Just Full Name
Just age not birthday

2. Encapsulation

- Encapsulation is a process where you keep all the inner works of the system together hidden. The important works are stored hidden to keep it safe from the average user which ensures the integrity of the system as it was designed.



2. Encapsulation

Encapsulation

Need to prevent outside code from accessing data that is private.

Therefore, you need getter and setter for this purpose, when you program!

Setter: method that takes parameter(s) and assign them to private variable(s) (It prevents outside from directly modifying the data)

Getter: method that takes no parameter and just returns values of private variable(s) (It prevents outside from directly retrieving the data)

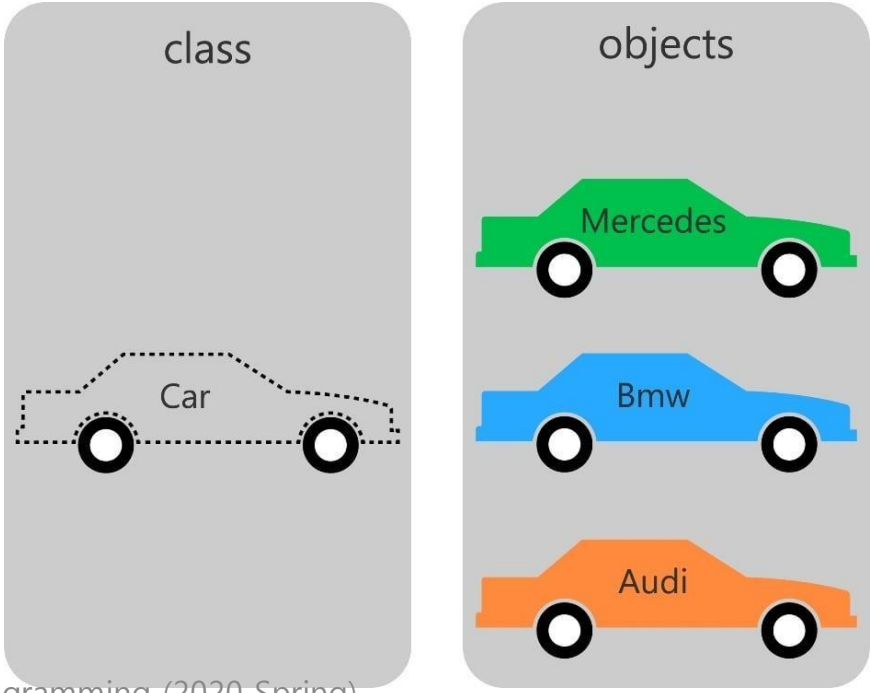
Encapsulation -example

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

#define YEAR 2020
class Person{
private :
    string first_name;
    string last_name;
    int birthDay; //20191231
    int calcOld(){
        return YEAR-(birthDay/10000)+1;
    }
public :
    Person(string f,string l,int b){
        first_name=f;
        last_name=l;
        birthDay=b;
    }
    string getFirstName(){
        return first_name;
    }
    void setFirstName(string fn){
        first_name=fn;
    }
    string getLastName(){
        return last_name;
    }
    void setLastName(string ln){
        last_name=ln;
    }
    int getBirthDay(){
        return birthDay;
    }
    void setBirthDay(int b){
        birthDay=b;
    }
};
```

3. Inheritance

- Inheritance is the mechanism by which an object acquires the some/all properties of another object.
- It supports the concept of hierarchical classification.



3. Inheritance

Inheritance -example

```
// C++ program to demonstrate implementation
// of Inheritance

#include <iostream>
using namespace std;

//Base class
class Parent
{
    public:
      int id_p;
};

// Sub class inheriting from Base Class(Parent)
class Child : public Parent
{
    public:
      int id_c;
};

//main function
int main()
   {

        Child obj1;

        // An object of class child has all data members
        // and member functions of class parent
        obj1.id_c = 7;
        obj1.id_p = 91;
        cout << "Child id is " <<  obj1.id_c << endl;
        cout << "Parent id is " <<  obj1.id_p << endl;

        return 0;
   }
```

Output:
Child id is 7
Parent id is 91

Inheritance -example

```
class A
{
public:
    int x;
protected:
    int y;
private:
    int z;
};

class B : public A
{
    // x is public
    // y is protected
    // z is not accessible from B
};

class C : protected A
{
    // x is protected
    // y is protected
    // z is not accessible from C
};

class D : private A    // 'private' is default for classes
{
    // x is private
    // y is private
    // z is not accessible from D
};
```

Base class member access specifier	Type of Inheritence		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	Not accessible (Hidden)	Not accessible (Hidden)

Types of Inheritance in C++

Inheritance –example(single inheritance)

```
// C++ program to explain
// Single inheritance
#include <iostream>
using namespace std;

// base class
class Vehicle {
  public:
    Vehicle()
    {
      cout << "This is a Vehicle" << endl;
    }
};

// sub class derived from two base classes
class Car: public Vehicle{

};

// main function
int main()
{
    // creating object of sub class will
    // invoke the constructor of base classes
    Car obj;
    return 0;
}
```

Output:
This is a vehicle

Inheritance –example(multilevel inheritance)

```
// C++ program to implement
// Multilevel Inheritance
#include <iostream>
using namespace std;

// base class
class Vehicle
{
  public:
    Vehicle()
    {
      cout << "This is a Vehicle" << endl;
    }
};
class fourWheeler: public Vehicle
{  public:
    fourWheeler()
    {
      cout<<"Objects with 4 wheels are vehicles"<<endl;
    }
};
// sub class derived from two base classes
class Car: public fourWheeler{
   public:
     car()
     {
       cout<<"Car has 4 Wheels"<<endl;
     }
};

// main function
int main()
{
    //creating object of sub class will
    //invoke the constructor of base classes
    Car obj;
    return 0;
}
```

Output :

```
This is a Vehicle
Objects with 4 wheels are vehicles
Car has 4 Wheels
```

4. Polymorphism

- Polymorphism means to process objects differently based on their data type.
- One method with multiple implementation, for a certain class of action.

4. Polymorphism

Polymorphism -example

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

class Shape {
   protected:
      int width, height;

   public:
      Shape( int a = 0, int b = 0){
         width = a;
         height = b;
      }
      int area() {
         cout << "Parent class area :" <<endl;
         return 0;
      }
};
class Rectangle: public Shape {
   public:
      Rectangle( int a = 0, int b = 0):Shape(a, b) { }

      int area () {
         cout << "Rectangle class area :" <<endl;
         return (width * height);
      }
};

class Triangle: public Shape {
   public:
      Triangle( int a = 0, int b = 0):Shape(a, b) { }

      int area () {
         cout << "Triangle class area :" <<endl;
         return (width * height / 2);
      }
};
```

```
// Main function for the program
int main() {
   Shape *shape;
   Rectangle rec(10,7);
   Triangle  tri(10,5);

   // store the address of Rectangle
   shape = &rec;

   // call rectangle area.
   shape->area();

   // store the address of Triangle
   shape = &tri;

   // call triangle area.
   shape->area();

   return 0;
}
```

Output:

```
Parent class area :
Parent class area :
```

Polymorphism -example

```
class Shape {
   protected:
      int width, height;

   public:
      Shape( int a = 0, int b = 0) {
         width = a;
         height = b;
      }
      virtual int area() {
         cout << "Parent class area :" <<endl;
         return 0;
      }
};
```

```
Rectangle class area
Triangle class area
```

Constructor

- Form of the Constructor
  - The class has a function that is named same as class name.
  - Return type not declared, not actually returned.
  - A kind of function that allows initializing variables (using parameters)

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

class Constructor
{
	int num1;
	int num2;

public:
	Constructor()
	{
		num1=0;
		num2=0;
	}
	Constructor(int n)
	{
		num1=n;
		num2=0;
	}
	Constructor(int n1, int n2)
	{
		num1=n1;
		num2=n2;
	}
	/* default parameter constructor
	Constructor(int  n1=0,  int n2=0)
	{
		num1=n1;
		num2=n2;
	}
	*/

	void ShowData() const
	{
		cout<<num1<<' '<<num2<<endl;
	}
};

int main(void) {
	Constructor sc1;
	sc1.ShowData();

	Constructor sc2(100);
	sc2.ShowData();

	Constructor sc3(100, 200);
	sc3.ShowData();
	return 0;
}
```

When sc1, sc2, sc3 objects are being made, they use different constructors based on what are passed into as parameter(s).

Copy Constructor

- Copy Constructor
  - constructor that initializes an object using another object of the same class
  - default copy constructor does what we call “shallow copy”

```
#include <iostream>
#include <cstring>
using namespace std;
class Book
{
private:
	char * bookName;
	int bookNum;
public:
	Book(char * tempName, int tempNum)
	{
		int len=strlen(tempName)+1;
		bookName=new char[len];
		strcpy(bookName, tempName);
		bookNum=tempNum;
	}
	void ShowBookInfo() const
	{
		cout<<"Book Name : "<<bookName<<endl;
		cout<<"Book Number : "<<bookNum<<endl;
	}
	~Book()
	{
		delete []bookName;
		cout<<"destructor"<<endl;
	}
};
```

```
int main(void)
{
	Book book1("Computer Programming", 2001001);
	Book book2("This is C++", 400010);
	Book book3(book2);
	book1.ShowBookInfo();
	book2.ShowBookInfo();
	book3.ShowBookInfo();
	return 0;
}
```

- Even if you don't define a copy constructor, a default copy constructor is used (C++ does it automatically).
- Book book3(book2) <- using copy constructor
- If you run the code above, you only get "destructor" output TWICE.
- This is because book3 (that is created using default copy constructor) and book2 are pointing to the same memory address of variable bookName(char*).
- book3's bookName(char*) is deleted first by destructor. Then book2's bookName(same memory addr) cannot be deleted because it has already been deleted.
- This might cause an error when you program.

Copy Constructor

- To solve this problem, it needs to copy this book name part into another memory location.
- This is called "deep copy".

```
// 기본 생성자 사용 (shallow copy)  Book(Book& b) {
                    bookName=b.bookName;
                    bookNum=b.bookNum;
                    }
// 복사 생성자 사용 (deep copy)  Book(Book& b) {
                    int len=strlen(b.bookName)+1;
                    bookName=new char[len];
                    strcpy(bookName, b.bookName);
                    bookNum=b.bookNum;
}
```

참고용:
https://lesslate.github.io/cpp/%EC%96%95%EC%9D%80%EB%B3%B5%EC%82%AC-%EA%B9%8A%EC%9D%80%EB%B3%B5%EC%82%AC/

Destructor

- Destruct the resources which is allocated by constructor.
- If there is memory space allocated by new operator, then destructor destructs this memory space (that is, retrieve data space for another process to use!)
- reference>> new and delete
  - They are compared to malloc and free respectively (in c).
  - When you generate objects, you have to use "new".

Exercise

Let's write classes to practice inheritance & polymorphism

1. Class GameCharacter.
    a. It will serve as a parent class
    b. It has five variables: strength, dexterity, intelligence, luck, power (double type)
    c. Getter and setters for all variables (I know it is plenty.. But do it for your own benefit)
    d. It need constructors
        i. Default constructor that takes no parameter and initializes variables to 0
        ii. Overriden constructor that takes five parameters to initialize all variables.
    e. Virtual void attack() method
        i. It prints out "attack!!"
    f. Virtual double damage() method
        i. Leave it empty for now

Exercise

2. Class Warrior

a. It inherits from GameCharacter class.
b. Constructor that takes two parameters (default param1 value=10 and param2 value=30)
    i. Assign 1st parameter value to strong (larger than 10)
    ii. Assign 2nd parameter value to power (larger than 30)
    iii. Assign the rest to value of (parameter1 - 10)
c. void attack() method
    i. It prints out “Warrior: attack!! Clang! Clang!”
    ii. Also prints out damage
d. double damage() method
    i. It calculates damage based on variables and returns it.
    ii. Equation: damage = (strong*4 + dexterity + luck*0.1 + intelligence *0.1) * power

Exercise

3. Class Magician

a. It inherits from GameCharacter class.
b. Constructor that takes two parameters (default param1 value=10 and param2 value=30)
    i. Assign 1st parameter value to intelligence (larger than 10)
    ii. Assign 2nd parameter value to power (larger than 30)
    iii. Assign the rest to value of (parameter1- 10)
c. void attack() method
    i. It prints out “Magician: attack!! magic balt!”
    ii. Also prints out damage
d. double damage() method
    i. It calculates damage based on variables and returns it.
    ii. Equation: damage = (intelligence*4 + luck + dexterity*0.1 + strength*0.1) * power

Exercise

4. Class Bowman

a. It inherits from GameCharacter class.
b. Constructor that takes two parameters (default param1 value=10 and param2 value=30)
    i. Assign 1st parameter value to dexterity (larger than 10)
    ii. Assign 2nd parameter value to power (larger than 30)
    iii. Assign the rest to value of (parameter1 - 10)
c. void attack() method
    i. It prints out “Bowman: attack!! zing zing!”
    ii. Also prints out damage
d. double damage() method
    i. It calculates damage based on variables and returns it.
    ii. Equation: damage = (dexterity*4 + strength + intelligence*0.1 + luck*0.1) * power

Exercise

5. Class Thief

a. It inherits from GameCharacter class.
b. Constructor that takes two parameters (default param1 value=10 and param2 value=30)
    i. Assign 1st parameter value to luck (larger than 10)
    ii. Assign 2nd parameter value to power (larger than 30)
    iii. Assign the rest to value of (parameter1 - 10)
c. void attack() method
    i. It prints out “Thief: attack!! ping ping!”
    ii. Also prints out damage
d. double damage() method
    i. It calculates damage based on variables and returns it.
    ii. Equation: damage = (luck*4 + dexterity + intelligence*0.1 + strength*0.1) * power

Important Note for writing program with multiple header & cpp files!!

- You need to know what is preprocessor directives (one example is #include you already known)
- We have to use preprocessor directives such as #ifndef … #define … #endif … to avoid multiple declaration of class or something.
- For building program using multiple files
  - We recommend not to use build shortcuts we set up for vscode but
  - Use terminal on the bottom and write command on it to build the program.
  - g++ -o (execuatble name) (all cpp file names you use seperated by space)
    - ex) g++ -o cars bmw.cpp benz.cpp hyundai.cpp . . . main.cpp