Lab 3
OOP – Abstraction, Polymorphism, Inheritance, Encapsulation

A. P. I. E.
1. Abstraction

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

No "first_name" and "last_name", Simply ask the object who it is or how old it is.

It returns nicely formatted strings that interpolate the data we have stored or data we have calculated.

class Person
  def initialize(first_name, last_name, birthday)
    @first_name = first_name
    @last_name = last_name
    @birthday = Date.parse(birthday)
  end

  def who_am_i?
    "My name is #{first_name} #{last_name}" # => We abstract these variables into practical methods
  end

  def how_old_am_i?
    "I am #{age} years old"
  end

  private # => We further encapsulate by making data and methods private
    attr_reader :first_name, :last_name, :birthday

  def age
    Date.today.year - @birthday.year
  end
end

john = Person.new('John', 'Smith', '18/05/1986')

john.who_am_i? # => I am John Smith
john.how_old_am_i? # => I am 31 years old
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

```java
private void shutter_click() {
    very secret code;
}

private void memory_save() {
    very secret code;
}

public void shot_button() {
    shutter_click();
    memory_save();
}
```
2. Encapsulation

We encapsulate the data (first name, last name, age) and the functions needed for that data (methods for names and a method for date) inside the class. We have packaged all relevant information together and only the methods within that package can directly manipulate that information.
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
3. Inheritance

As you can see in the example Jane can tell you who she is and how old she is, but she can also tell you what grade she is in. We are able to extend the functionality of a child class without duplicating all the code from the parent. It makes our code reusable and keeps us from repeating ourselves.
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
4. Polymorphism

If we override the “who_am_i?” method in the child class to only offer a first name but additionally offer an age, we still have not changed the interface with the code from the user perspective.
Constructor

• Form of the Constructor
  • The class has the same name as the function.
  • Return type not declared, not actually returned.
  • A kind of function that allows default values to be set for overload and parameter.
#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 pass overloaded constructor. If you use the default parameter constructor, then you erase other constructors. The result is same.
Constructor

• The initialization using member initializer
  • Use member initializer when you call constructors of the objects which is declared as member variable.
  • Not initialize at the body, initialize at the next of the parameters.
```cpp
#include <iostream>
using namespace std;

class Constructor
{
    int num1;
    int num2;

public:
    Constructor(int n1, int n2) : num1(n1), num2(n2)
    {
    }

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

int main(void)
{
    Constructor sc(100, 200);
    sc.ShowData();
    return 0;
}
```
Destructor

• Destruct the resources which is allocated by constructor.

• If there is memory space allocated by new operator, then destructor destruct this memory space

• reference>> new and delete
  • They are compared to malloc and free respectively.
  • When you generate objects, you have to use "new".
```cpp
#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);
        book1.ShowBookInfo();
        book2.ShowBookInfo();
        return 0;
    }
};
```
Copy Constructor

• Copy Constructor
  • When you recall name which is generated in parameter, copy the object.
  • If you are not definite copy constructor, default copy constructor insert automatically.

• Kinds of copy constructor through conversions
  • implicit conversion : = , explicit conversion : (object)
  • you have to use explicit to prevent implicit conversion
Copy Constructor

• Call point of copy generator
  • 1. Initialize a new object using a already generated object.
     Point x2(x1);
  • 2. Call-by-value : pass the object as a parameter during the function calling
     Point copyFunc(Point obj)
     {
       return obj;
     }
  • 3. return the object which is not returned by the references.
     Point copyFunc(Point obj)
     {
       return obj;
     }
```cpp
#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;
}
```

- if you not define any copy constructor, a default copy constructor copies member to member.
- Upper code has an error, the default copy constructor points same book name part, destructor destruct at book2, and destructor destruct at book3, too. But there is nothing to destruct. because string already destructed. so, error appears.
Copy Constructor

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

```cpp
// 기본 생성자 사용 (shallow copy)
TestClass(int a) {
    num = a;
}

// 복사 생성자 (deep copy)
TestClass(TestClass& tc) {
    num = tc.num;
}
```
Vector

• An array-based container that supports a random access iterator.

• Elements are stored consecutively in one memory block

```cpp
template<typename T, typename Allocator = allocator<T>>
class vector

v.pop_back() : Remove the last element of v.
v.push_back() : Add the element to the end of v
```

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

int main(void)
{
    vector<int> v;
    v.push_back(5);
    v.push_back(2);
    v.pop_back();
}
```
Vector

- `vector<int> v;`
  - 비어있는 `vector v`를 생성합니다.

- `vector<int> v(5);`
  - 기본값(0)으로 초기화된 5개의 원소를 가진 `vector v`를 생성합니다.

- `vector<int> v(5, 2);`
  - 2로 초기화된 5개의 원소를 가진 `vector v`를 생성합니다.

- `vector<int> v1(5, 2);
  vector<int> v2(v1);`
  - `v2`는 `v1` `vector`를 복사해서 생성됩니다.

- `vector<int> v1; vector<int> v2;` 가 있고, 내부에 인자들이 있다고 했을때.

Vector

- `v.assign(5, 2);`
  - 2의 값으로 5개의 원소 할당.

- `v.at(idx);`
  - idx번째 원소를 참조합니다.
  - v[idx]보다 속도는 느리지만, 범위를 점검하므로 안전합니다.

- `v[idx];`
  - idx 번째 원소를 참조합니다.
  - 범위를 점검하지 않으므로 속도가 v.at(idx)보다 빠릅니다.

- `v.front();`
  - 첫번째 원소를 참조합니다.

- `v.back();`
  - 마지막 원소를 참조합니다.

- `v.clear();`
  - 모든 원소를 제거합니다.
  - 원소만 제거하고 메모리는 남아있습니다.
  - size만 줄어들고 capacity는 그대로 남아있습니다.

- `v.begin();`
  - 첫번째 원소를 가리킵니다. (iterator와 사용)

- `v.end();`
  - 마지막의 "다음"을 가리킵니다 (iterator와 사용)

- `v.begin();`
  - reverse begin을 가리킨다 (거꾸로 해서 첫번째 원소를 가리집니다)
  - iterator와 사용.

- `v.end();`
  - reverse end 을 가리킨다 (거꾸로 해서 마지막의 다음을 가리집니다)
  - iterator와 사용.

- `v.reserve(n);`
  - n개의 원소를 저장할 위치를 예약합니다 (마리 동적할당 해늘 nữa)

- `v.resize(n);`
  - 크기를 n으로 변경합니다.
  - 더 커졌을 경우 default값인 0으로 초기화 합니다.

- `v.resize(n, 3);`
  - 크기를 n으로 변경합니다.
  - 더 커졌을 경우 인자의 값을 3으로 초기화합니다.
Polymorphism

• Same sentence but different result.

• Polymorphism : the method of implementing all of the super-class' member. Sub-class has its own member and super-class' member.

• Is-a relation.
class Person
{
private:
  int age;
  char name[50];
public:
  Person(int myage, char * myname) : age(myage)
  {
    strcpy(name, myname);
  }
  void ShowName() const
  {
    cout << "My name is" << name << endl;
  }
  void ShowAge() const
  {
    cout << "My age is" << age << endl;
  }
};

class Student : public Person
{
private:
  char major[50];
public:
  Student(char * myname, int myage, char * mymajor) : Person(myage, myname)
  {
    strcpy(major, mymajor);
  }
  void ShowStudent() const
  {
    ShowName();
    ShowAge();
    cout << "My major is" << major << endl; // Removed the duplicate endl
  }
};

Student is a person. (is-a relation), student class inherits person class.
Student is implemented by 'public Person'.
And Student is inherited from Person's member.