**OOP244 – REFERENCE SHEET J. ANCHETA /017433152**

**EXAM DATE: 19 APR TUE 11.30 – 1:30PM @GYM**

\*\*\* Abstraction – identifying the most important components and hiding the less important details.

1. Encapsulation - combining the logic and data together to form a program and hiding the core code from the user.

2. Inheritance - it allows us to define another class using other class properties.

3. Polymorphism - class member functions are defined with the same name but it functions differently.

4. Synergy – cohesiveness or proper interaction of all modules.

Class - is a design or idea. Collection of data and function members. Object - is an instance of a class. Header file contains the class definitions and function prototypes. Methods or implementation file defines the logic within the function.

*class Student {*

*int xyz\_;*

*bool valid\_;*

*public:*

*void display (num, abc) const;*

*}*

Methods/Implementation file:

Queries (accessor methods) – report the state of the object

Modifiers (mutator methods) –change the state of the object

Special(manager methods) – create, assign and destroy obj

Compilation process – the pre-processing stage will insert all header files and define statements included in all source codes. After that, the compiler will create an object for each module. The objects created from the previous process will then be combined together by the linker producing an executable file.

Safeguard for header files:

*#ifndef SICT\_CAR\_H\_*

*#define SICT\_CAR\_H\_*

// statements

*#endif*

Static vs. Dynamic memory – static is a memory reserved by the operating system during load time of the program while Dynamic is a memory reserved by the OS during run time and the program itself allocates and deallocates this reserved memory.

*int\* cpa = nullptr; //pointer declaration*

*cpa = new int[n]; //allocates space in dynamic memory*

*delete[] cpa; // deletes dynamic memory*

Constructors are used to initialize data members of an object at the time of creation. Destructor is called before an object goes out of scope to clean buffers and delete dynamic memory.

*Student(); //default constructor without arguments*

*Student( int, bool = true); //constructor with arguments*

*~Student(); //destructors- to delete dynamic memory*

Safe Empty State is when all data members of an object is set to zero. This can be done automatically thru the definition/methods of default constructor.

*Student::Student(){*

*xyz\_[0] = 0; // if character array*

*valid = 0 ; // false*

*}*

Constructors are being called automatically at the time of creation of an object of a class and cannot be called from main() function or from any methods of the class. Destructor is called before the object goes out of scope.

Explicit vs Implicit parameters – explicit parameters are written clearly on the source code/program while implicit parameters are not written but its inside the object itself.

Return \*this (pass by reference or value) – refers to the current object itself who called the method/member function.

Copy Constructor – initialize a newly created object by copying the data of another existing object (source) of the same class.

*e.g. Class name = Student*

*Student A (20,30); call the constructor with 2 -arguments*

*Student B (A); // same as Student B = A - will call copy constructor and copy the data of Student A and assign it to data members of Student B.*

Assignment operators = copying data from an existing object to another existing object. In the method, it checks for self-assignment (if this != &source).

Abstract base class is a class that contains or inherits a pure virtual function that has yet to be defined. It will create a compiler error when attempting to create an object.

Interface = an abstract base class without data members.

Protected member functions at base class can be access only by the derived class.

Pure virtual member function:

virtual std::fstream& load(std::fstream& file) = 0

Derived class doesn’t inherit constructor, destructor, copy & assignment operators member functions of the base class.

//append subject class to the end of sub\* using temp sub and create new DMA for sub \* and transfer content of temp

Transcript& operator+=(const Subject& S){

Subject\* temp;

int i;

temp = new Subject[noOfSubs\_ + 1];

for (i = 0; i < noOfSubs\_; i++){

temp[i] = subs\_[i];

}

noOfSubs\_++;

temp[noOfSubs\_-1] = S;

delete[] subs\_;

subs\_ = new Subject[noOfSubs\_];

for (i = 0; i < noOfSubs\_; i++){

subs\_[i] = temp[i];}

return \*this;

}

**INHERITANCE**

class Vehicle{

public:

virtual void move() const = 0;

virtual ~Vehicle() { cout << "V Trashed" << endl << "---" << endl; }

void stop()const { cout << "STOPPED!" << endl;}

//if put on virtual, allow child run their own method with the same name as the parent

};

class Car : public Vehicle{

public:

void move() const { cout << "C Go!" << endl; }()

void stop()const { cout << "STOPPED CAR!" << endl;} //will not be run as parent has this method

~Car() { cout << "C Trashed" << endl; }

};

class BMW : public Car{

public:

void move() const { cout << "Brrrom Go!" << endl; }

~BMW() { cout << "B Trashed" << endl; }

};

class Tesla : public Car{

public:

void move() const { cout << "whooosh Go!"<< endl; }

~Tesla() {cout << "Never Trash Tesla!!!"<< endl; }

};

class BMW320 : public BMW{

public:

//if no method in derived class, it will run the method of the parent

~BMW320() {cout << "320 Trashed" << endl; }

};

int main(){

Vehicle\* V[4] = { new Car, new BMW, new Tesla, new BMW320 };

for (int i = 0; i < 4; i++){

V[i]->move();

V[i]->stop();

}

for (int i = 0; i < 4; i++) delete V[i];

//no destructor if there is no delete on DMA

return 0;

}

//don’t allow Contact object to be copied or assigned to another Contact

Contact::Contact(const Contact& cCopy) = delete{

set(cCopy);

}

Contact& Contact::operator=(const Contact& C)=delete{

if (this != &C){

delete[] phoneNumber\_;

set(C);

}

return \*this;

}

//destructor

Virtual ~Contact(){delete[] phoneNumber\_;}

// ACCESSOR WITH <IOMANIP> FORMATTING

void Weather::display() const{

cout << setfill('\_') << setw(10) << left << date\_

<< setw(6) << fixed << setprecision(1) << right

<< high\_<< setw(6) << fixed << setprecision(1)

<< right << low\_ << endl;

}

class MyString{

char data\_[2046];

public:

MyString(const char\* str = ""){set(str);}

MyString& set(const char\* str){

strncpy(data\_, str, 2045);

data\_[2045] = 0;

return \*this;

}

operator const char\*()const{return data\_;}

MyString& operator+=(const MyString& M){

strcat(data\_, M.data\_);

return \*this;

}

MyString& operator=(const char\* s){return set(s);}

};

//extraction operator

ostream& operator<<(ostream& os,const MyString& M){

return os << (const char\*)M;}

//insertion operation NOT be const

const char\* operator+=(char\* s, const MyString& M){

return strcat(s, (const char\*)M);}

class Paragraph{

MyString\* s\_;

int size\_;

public:

Paragraph() :s\_(nullptr), size\_(0){};

Paragraph(MyString\* s, int size){

s\_ = new MyString[size\_ = size];

for (int i = 0; i < size; i++)s\_[i] = s[i];}

~Paragraph(){delete[] s\_;}

int size()const{return size\_;}

MyString& operator[](int i){return s\_[i];}

int main(){

char str[1000] = "Hello ";

MyString S = "there!";

cout << (str += S);

return 0;

}

#include <fstream>

fstream tempFile("temp.txt", ios::out); //to write

tempFile << “Write this text to temp.txt”;

fstream tempFile("temp.txt", ios::in); //to read

char str[50];

tempFile.getline(str, 50,’\n’); //1 line to str

state Methods: fail() ignore() clear() good() bad() is\_open() = boolean

fstream& AmaProduct::load(std::fstream& file){

file.getline(tempSku, MAX\_SKU\_LEN, ',');

file.getline(tempName, 1000, ',');

fstream& AmaProduct::store(std::fstream&file)const{

file << fileTag\_ << "," << sku() << "," << name();