BUBT | BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY

ASSIGNMENT

ASSIGNMENT NO -01,02,03

Course NO :121

Course Name : Object Oriented Programming Language Lab

Submission Date : 10-04-2023

Submitted To
Name:Khan Md. Hasib Assistant Professor Department of Computer Science & Engineering
Submitted By
Name: S.M. Fazla Rabby Ashik **ID:21225103213 INATKE:49 SECTION:06**

CO1

C++ program to implement inheritance with following requirements,
Classes :- Animal.cpp, bird.cpp, canine.cpp, main.cpp (to test it) :-
-You may need getters and setters also.

1. Declare and define an animal base class:
animal stores an age (int), a unique long ID (a boolean that is true if it is alive, and a location (a pair of double).
animal requires a default constructor and a 3 parameter constructor. Both constructors should set the unique ID automatically. They should also set the alive boolean to true. The three parameter constructor should accept an int for the age and two doubles for the set of coordinates. The default should set these three values to 0.

animal requires a virtual move method which accepts two doubles that represent two coordinates, and 'moves' the animal to the set of coordinates.

animal requires a copy constructor and a virtual destructor. The destructor should be virtual.

animal requires a virtual sleep method and a virtual eat method. Both methods return void. Both methods should print an appropriate message to cout.

animal requires a setAlive function which accepts a boolean. This is a void function that changes the alive data member to the value of the boolean that is passed in.
Overload the insertion operator for the animal.

Soln:

```
#include <iostream>
#include<conio.h>
#include<string>
using namespace std;
class Animal {
private:
int age;
long id;
bool alive;
pair<double, double> location;
public:
Animal() : age(0), id(generateUniqueId()), alive(true), location({0.0, 0.0}) {}
Animal(int age, double x, double y) : age(age), id(generateUniqueId()), alive(true),
```

```
location({x, y}) {}
virtual ~Animal() {}
Animal(const Animal& other) : age(other.age), id(generateUniqueId()), alive(true),
location(other.location) {}
void setAlive(bool status) { alive = status; }
bool isAlive() const { return alive; }
void move(double x, double y) {
cout << "Animal moved to (" << x << ", " << y << ")" << endl;
location = {x, y};
}
virtual void sleep() { cout << "Animal is sleeping" << endl; }
virtual void eat() { cout << "Animal is eating" << endl; }
friend ostream& operator<<(ostream& os, const Animal& animal) {
os << "Animal: age = " << animal.age << ", id = " << animal.id << ", alive = " <<
animal.alive
<< ", location = (" << animal.location.first << ", " << animal.location.second << ")";
return os;
}
private:
static long generateUniqueId() {
static long id = 0;
return ++id;
}
};
class Bird : public Animal {
public:
void fly(double x, double y) {
cout << "Bird is flying to (" << x << ", " << y << ")" << endl;
move(x, y);
}
void sleep() override { cout << "Bird is sleeping" << endl; }
void eat() override { cout << "Bird is eating seeds" << endl; }
};
class Canine : public Animal {
public:
void run(double x, double y) {
cout << "Canine is running to (" << x << ", " << y << ")" << endl;
move(x, y);
}
void sleep() override { cout << "Canine is sleeping" << endl; }
void eat() override { cout << "Canine is eating meat" << endl; }
};
int main() {
Animal animal1;
cout << animal1 << endl;
Bird bird1( 1.0, 2.0);
```

```
cout << bird1 << endl;
bird1.fly(3.0, 4.0);
bird1.sleep();
bird1.eat();
Canine canine1(5, 5.0, 6.0);
cout << canine1 << endl;
canine1.run(7.0, 8.0);
canine1.sleep();
canine1.eat();
getch();
}
```

CO2

Imagine a publishing company that markets both book and audiocassette versions of its works. Create a class publication that stores the title (a string) and price (type float) of a publication. From this class derive two classes: book, which adds a page count (type int), and tape, which adds a playing time in minutes (type float). Each of these three classes should have a get Data() function to get its data from the user at the keyboard, and a put Data() function to display its data.

Write a main program to test the book and tape classes by creating instances of them, asking the user to fill in data with get Data(), and then displaying the data with put Data().

Soln:-

```
#include <iostream>
#include<conio.h>
#include <string>
using namespace std;
class Publication {
private:
string title;
float price;
public:
void getData() {
cout << "Title name: ";
cin>>title;
cout << "price: ";
cin >> price;
}
void putData()
const
{
cout << "Title: " << title << endl;
cout << "Price: $" << price << endl;
}
};
class Book : public Publication {
private:
int Count;
public:
void getData() {
Publication::getData();
cout << "PAGE Count: ";
cin >>Count;
```

```
}
void putData() const {
Publication::putData();
cout << "Page count: " << Count << endl;
}
};
class Tape : public Publication {
private:
float Time;
public:
void getData() {
Publication::getData();
cout << "Enter playing time in minutes: ";
cin >>Time;
}
void putData() const {
Publication::putData();
cout << "Playing time: " << Time << " minutes" << endl;
}
};
int main() {
Book book;
Tape tape;
cout << "DATA FOR BOOK"<<endl;
book.getData();
cout << "DATA FOR TAPE"<<endl;
tape.getData();
cout << "BOOK DATA"<<endl;
book.putData();
cout << "TAPE DATA"<<endl;
tape.putData();
getch();
}
```

CO3

CO3:-

Demonstrate a C++ code that creates a class called Fraction. The class Fraction has two attributes: numerator and denominator.

• In your constructor (inyour__init__ method), verify(assert?) that the numerator and denominator passed in during initiation are both of type int. If you want to be thorough, also check to make sure that the denominator is not zero.

• Write a .reduce() method that will reduce a fraction to lowest terms.

• Override the Object class's __str__ and __repl__ methods so that your objects will print out nicely. Remember that__str__ is more for humans; __repl__ is more for programmers. Ideally ,the__ repl __ method will produce a string that you can run through the eval() function to clone the original fraction object.

• Override the + operator. In your code, this means that you will implement the special method __add__. The signature of the __add__ function will be def __add__(self, other): , and you'll return a new Fraction with the result of the addition. Run your new Fraction through the reduce() function before Returning.

Soln:

```
#include <iostream>
#include<conio.h>
#include <string>
using namespace std;
class Fraction {
private:
int numerator;
int denominator;
public:
Fraction(int numerator, int denominator)
{
assert(denominator != 0 && "Denominator can't be zero!");
assert((typeid (numerator) == typeid(int)) && (typeid(denominator) == typeid(int)) &&
"Numerator and denominator must be integers!");
this->numerator = numerator;
this->denominator = denominator;
}
void reduce() {
int gcd = gcd (numerator, denominator);
```

```
numerator /= gcd;
denominator /= gcd;
}
string str() const {
return to_string(numerator) + "/" + to_string(denominator);
}
friend ostream & operator<<( ostream & os , const Fraction& fraction) {
os << fraction .str();
return os;
}
Fraction operator+(const Fraction& other) const {
int num = numerator * other.denominator + other.numerator * denominator;
int denom = denominator * other.denominator;
Fraction result(num, denom);
result. reduce();
return result;
}
};
int main() {
Fraction a(3, 4);
Fraction b(1, 2);
Fraction c = a + b;
cout << "a: " << a << endl;
cout << "b: " << b << endl;
cout << "c: " << c << endl;
getch();
}
```