Software Development Assignment 1

Submitted by: Mostafa Abdel-Hakeem Ahmed El-Baroudi

Submitted to: Eng. Omar Amin

Q1: What is a Class?

A: When a class is declared it's equal to declaration of a special data type (user defined) or data structure. For example, creating a class called class car which will contain some information inside like model, color and license plate number. It should be like

```
Class Car
{

string model;
string color;
int license_plate_number;

}
```

When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object. A class definition starts with the keyword class followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. Classes have functions to be called and specifiers to access of members either public or private. Private members are forbidden to be accessed outside the class.

Q2: How does a class accomplish abstraction, encapsulation, and data hiding?

A: Abstraction aims behavior of objects and essential details of items as an entity, while Encapsulation is about implementations which gives rise to objects' behavior enclosing items as a package and deal with it as a whole.
Information hiding is hiding data which is being affected by implementation. Data to be hidden is identified by abstraction.

Q3: What is the relationship between an object and a class?

A: Class defines behavior and specifications for objects represented by the preceding OOP properties.
An Object displays the property and behavior introduced by its class, in most cases an object is like an instance of a class

Q4: In what way, aside from being functions, are class function members different from class data members?

A: The public data members of objects of a class can be accessed using the direct member access operator (.). A member function of a class is a function that has its definition or its prototype within the class definition like any other variable. It operates on any object of the class of which it is a member, and has access to all the members of a class for that object.

Q5: Define a class to represent a bank account. Data members should include the depositor's name, the account number (use a string), and the balance. Member functions should allow the following: n Creating an object and initializing it. n Displaying the depositor's name, account number, and balance n Depositing an amount of money given by an argument n Withdrawing an amount of money given by an argument Just show the class declaration, not the method implementations. (Programming Exercise 1 provides you with an opportunity to write the implementation.)

A: Answered later in Ex1.

Q6: When are class constructors called? When are class destructors called?

A: Constructors are member functions of classes that has same name of the class and has no return values not even void, executes whenever objects of that class are created and usually used for member variables initialization. Destructors are executed whenever an object gets out of scope or on deletion of a pointer to an object of a class, have same name of class but prefixed with (~) and returns no values and takes no parameters. Used mainly for releasing resources before coming out of the program like closing files and erasing memories.

Q8: What is a default constructor? What is the advantage of having one?

A: In computer programming languages the term default constructor can refer to a constructor that is automatically generated by the compiler in the absence of any programmer-defined constructors (e.g. in Java), and is usually a null constructor. In other languages (e.g. in C++) it is a constructor that can be called without having to provide any arguments, irrespective of whether the constructor is auto-generated or user-defined. Note that a constructor with formal parameters can still be called without arguments if default arguments were provided in the constructor's definition.

Q10: What are this and *this?

A: '*' is a pointer in C++. The 'this' pointer is passed as a hidden argument to all non-static member function calls and is available as a local variable within the body of all non-static functions. 'this' pointer is a constant pointer that holds the memory address of the current object. 'this' pointer is not available in static member functions as static member functions can be called without any object (with class name). For a class X, the type of this pointer is 'X* const'. Also, if a member function of X is declared as const, then the type of this pointer is 'const X *const'.

Ex1: Provide method definitions for the class described in Review Question 5 and write a short program that illustrates all the features.

A: #include <iostream>

#include <string>

```
using namespace std;

class BankAccount
{
private:
   string DepositorName;
   string AccountNumber;
   double Balance;
public:
   BankAccount();
   BankAccount(string depositorname, string accountnumber, double balance);
   ~BankAccount();
   void ShowInfo() const;
   void Withdraw(double amount);
   void Deposit(double depositamount);
};
BankAccount::BankAccount()
{
   DepositorName = "No Name";
   AccountNumber = "00000000";
   Balance = 0;
}
BankAccount::BankAccount(string depositorname, string accountnumber, double balance)
{
   DepositorName = depositorname;
   AccountNumber = accountnumber;
   Balance = balance;
```

```
}
BankAccount::~BankAccount()
{
}
void BankAccount::ShowInfo() const
{
   cout << "Depositor's name: " << DepositorName << endl ;
   cout << "Account number: " << AccountNumber << endl ;
   cout << "Balance: $" << Balance;
}
void BankAccount::Withdraw(double amount)
{
   if (amount < 0)
      cout << "Please enter positive number for the amount to withdraw.";
   else
      Balance -= amount;
}
void BankAccount::Deposit(double depositamount)
{
   if (depositamount < 0)
      cout << "Please enter positive number for the amount to deposit.";
   else
      Balance += depositamount;
}
```

Ex2: Here is a rather simple class definition: class Person { private: static const LIMIT = 25; string lname; // Person's last name char fname[LIMIT]; // Person's first name public: Person() {lname = ""; fname[0] = '\0'; } // #1 Person(const string & ln, const char * fn = "Heyyou"); // #2 // the following methods display lname and fname void Show() const; // firstname lastname format void FormalShow() const; // lastname, firstname format }; (It uses both a string object and a character array so that you can

compare how the two forms are used.) Write a program that completes the implementation by providing code for the undefined methods.The program in which you use the class should also use the three possible constructor calls (no arguments, one argument, and two arguments) and the two display methods. Here's an example that uses the constructors and methods: Person one; // use default constructor Person two("Smythecraft"); // use #2 with one default argument Person three("Dimwiddy", "Sam"); // use #2, no defaults one.Show(); cout << endl; one.FormalShow();

// etc. for two and three

A: #include <string>

#include <iostream>

using namespace std;

```
class Person {
   private:
      const int Limit = 25;
      string lastname; // Person's last name
      char firstname[Limit]; // Person's first name
   public:
      Person() {lastname = ""; firstname[0] = '\0'; } // #1
      Person(const string & ln, const char * fn = "Heyyou"); // #2
      ~Person();
      void Show() const; // firstname lastname format
      void FormalShow() const; // lastname, firstname format
};
Person::Person(const string & ln, const char * fn)
{
   lastname = ln;
   strcpy(firstname,fn);
}
Person::~Person()
{
```

```
}
void Person::Show() const
{
    cout << firstname << " " << lastname;
}

void Person::FormalShow() const
{
    cout << lastname << ", " << firstname;
}
int main()
{
    Person one;
    Person two("Smythecraft");
    Person three("Dimwiddy", "Sam");
    one.Show();
    cout << endl;
    one.FormalShow();
    two.FormalShow();
    two.Show();
    three.FormalShow();
    three.Show();
    return 0;
}
```

Ex6: Here's a class declaration:

```
class Move
```

```
{
private:
double x;
double y;
public:
Move(double a = 0, double b = 0); // sets x, y to a, b
showmove() const; // shows current x, y values
Move add(const Move & m) const;
// this function adds x of m to x of invoking object to get new x,
// adds y of m to y of invoking object to get new y, creates a new
// move object initialized to new x, y values and returns it
reset(double a = 0, double b = 0); // resets x,y to a, b
};
```

Create member function definitions and a program that exercises the class.

A:

```
#include <iostream>
using namespace std;
class Move
{
   private:
      double x;
      double y;
   public:
      Move(double a = 0, double b = 0);
      void showmove() const;
      Move add(const Move & m) const;
      void reset(double a = 0, double b = 0); // resets x,y to a, b
};
Move::Move(double a, double b)
```

```
{
   x = a;
   y = b;
}
void Move::showmove() const
{
   cout << "x = " << x << ", y = " << y;
}
Move Move::add(const Move &m) const
{
   Move temp;
   temp.x = x + m.x;
   temp.y = y + m.y;
   return temp;
}
void Move::reset(double a, double b)
{
   x = a;
   y = b;
}
int main()
{
   Move obj1(5,6);
   Move obj2(10,11);
   Move obj3;
   obj1 = obj1.add(obj2);
   cout << "obj1: ";
```

```
    obj1.showmove();
    obj2 = obj2.add(obj1);
    cout << "nobj2: ";
    obj2.showmove();
    obj3 = obj1.add(obj2);
    cout << "nobj3: ";
    obj3.showmove();
    cin.get();
    cin.get();
    return 0;
}
```