Julius Ranoa
CSC 121 001 Computer Science I
Homework – Chapter 7 Introduction to Classes and Objects

Part I. Review Questions @ Page 493 – 496.
Qn. 1, 2, 4 – 8, 11 – 22, 24, 28 – 30, 32 – 35, 38, 45 – 46.

1. What does ADT stand for? *Abstract Data Type*
2. Which of the following must a programmer know about an ADT to use it?
   A. What values it can hold; and B. What operations it can perform.
   An ADT's implementation is NOT needed to use it.
4. *Procedural* programming is centered around functions or procedures whereas *Object-Oriented* programming is centered around objects.
5. An object is a software entity that combines both *variables (data)* and *functions (procedures)* in a single unit.
6. An object is a(n) *instance* of a class.
7. Creating a class object is often called *instantiating* the class.
8. Once a class is declared, how many objects can be created from it? *C. Many*
11. Bundling together an object's data and procedures is called *encapsulation*.
12. An object's members can be declared *public* or *private*.
    A public member can be accessed by *functions outside the class*.
    A private member can be accessed by *functions that are members of the same class*.
13. Normally, a class's *member variables* are declared to be private and its *member functions* are declared to be public.
14. A class member function that uses, but does not change, the value of a member variable is called a(n) *accessor (getter)*.
15. A class member function that changes the value of a member variable is called a(n) *mutator (setter)* function.
16. When a member function's body is written inside a class declaration, the function is a(n) *inline* function.
17. A class constructor is a member function with the same name as the *class*.
18. A constructor is automatically called when an object is *created*.
19. Constructors cannot have a(n) *return* type.
20. A(n) *default* constructor is one that requires no arguments.
21. A destructor is a member function that is automatically called when an object is *destroyed (e.g. function scope where the object is created ends)*.

22. A destructor has the same name as the class but is preceded by a(n) *tilde (~)* character.

28. When a member function performs a task internal to the class and should be not be called by a client program, the function should be made *private*.

29. True or false: A class object can be passed to a function but cannot be returned by a function. *False. Classes are considered to data types and can be set as return types.*

30. True or false: C++ class objects are always passed to functions by reference. *False. Objects by default are passed by value.*

32. If you were writing a class declaration for a class named *Canine* and wanted to place it in its own file, what should you name the file? *Canine.h*

33. If you were writing the definitions for *Canine* class member functions and wanted to place these in their own file, what should you name the file? *Canine.cpp*

34. A structure is like a class but normally only contains member variables and no *member functions*.

35. By default, are the members of a structure public or private? *Public.*

38. The *dot* operator is used to access structure members.

45. Indicate whether each of the following enumerated data type definitions is valid or invalid. If it is invalid, tell what is wrong with it.

    A. `enum Holiday { Easter, Halloween, Thanksgiving, Christmas };`
    Valid.

    B. `Enum Holiday { Easter, Halloween, Thanksgiving, Christmas };`
    Invalid – enum must be in all lowercase. C++ is case-sensitive.

    C. `enum Holiday { “EASTER”, “HALLOWEEN”, “THANKSGIVING”, “CHRISTMAS” };`
    Invalid – enum constants cannot be strings like other variable names.

    D. `enum Holiday { EASTER, HALLOWEEN, THANKSGIVING, CHRISTMAS } nextHoliday;`
    Valid.

46. An enumerated data type and several variables have been defined like this:

```
enum Department {Purchasing, Manufacturing, Warehouse, Sales};
Department floor1, floor2;
int dNum = 2;
```

Indicate whether each of the following statements is valid or invalid. If it is invalid, tell what is wrong with it.

A. `floor1 = Sales;` *Valid.*

B. `dNum = Sales;` *Valid. Implicit conversion of enum constant to int.*

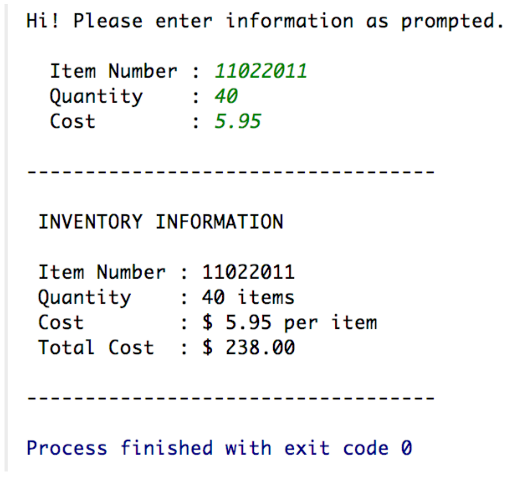
C. `dNum = floor1;` *Valid. Implicit conversion of enum constant to int.*

D. `floor2 = dNum;` *Invalid. Int must be cast as enum constant.*
e.g. `floor2 = static_cast<Department>(dNum);`

Part II. Programming Challenge @ Page 501 – 502.

Qn. 7 – Inventory Class.

Screenshot of Runtime:



```
Hi! Please enter information as prompted.

  Item Number : 11022011
  Quantity    : 40
  Cost        : 5.95

-----------------------------------

 INVENTORY INFORMATION

 Item Number : 11022011
 Quantity    : 40 items
 Cost        : $ 5.95 per item
 Total Cost  : $ 238.00

-----------------------------------

Process finished with exit code 0
```

Source Code:

1. *Inventory.h*
2. *Inventory.cpp*
3. *main.cpp*

The source code is also stored at Github.

Link below:

https://github.com/TheLoneWoof1102/FA17_CSC121001/tree/master/Source%20Code/Homework-Ch7.Qn7

main.cpp
```cpp
#include "Inventory.h"
#include <iostream>
#include <iomanip>
using namespace std;

bool safeGetInteger(int&);
bool safeGetDouble(double&);
void printStuff(Inventory);

int main() {
    Inventory inv;
    int tempN;
    double tempD;

    cout << "Hi! Please enter information as prompted." << endl << endl;

    cout << "  Item Number : ";
    while ( !safeGetInteger(tempN) ) {
        cout << "    Invalid input. Try again: ";
    }
    inv.setItemNumber(tempN);

    cout << "  Quantity    : ";
    while ( !safeGetInteger(tempN) ) {
        cout << "    Invalid input. Try again: ";
    }
    inv.setQuantity(tempN);

    cout << "  Cost        : ";
    while ( !safeGetDouble(tempD) ) {
        cout << "    Invalid input. Try again: ";
    }
    inv.setCost(tempD);

    cout << endl;
    printStuff(inv);

    return 0;
}

bool safeGetInteger(int &n) {
    string raw_input;
    getline(cin, raw_input);
    try {
        n = stoi(raw_input);
        return true;
    } catch (exception &e) { return false; }
}

bool safeGetDouble(double &d) {
    string raw_input;
    getline(cin, raw_input);
    try {
        d = stod(raw_input);
        return true;
    } catch (exception &e) { return false; }
}
```
***// main.cpp* is continued next page.**

main.cpp – cont'd.

```
void printStuff(Inventory inv) {
    string border;
    border.assign(35, '-');

    cout << border << endl << endl;
    cout << " INVENTORY INFORMATION" << endl << endl;
    cout << " Item Number : " << inv.getItemNumber() << endl;
    cout << " Quantity    : " << inv.getQuantity() << " items" << endl;
    cout << " Cost        : $ " << fixed << showpoint << setprecision(2)
         << inv.getCost() << " per item" << endl;
    cout << " Total Cost  : $ " << fixed << showpoint << setprecision(2)
         << inv.getTotalCost() << endl;
    cout << endl << border << endl;
}
```

// END of *main.cpp*.

Inventory.h

```
//
// Created by TheLoneWoof on 10/8/17.
//

#ifndef HOMEWORK_CH7_QN7_INVENTORY_H
#define HOMEWORK_CH7_QN7_INVENTORY_H

class Inventory {

private:
    int itemNumber;
    int quantity;
    double cost;
    double totalCost;
    void setTotalCost();

public:
    Inventory();
    Inventory(int, int, double);

    void setItemNumber(int);
    void setQuantity(int);
    void setCost(double);

    int getItemNumber();
    int getQuantity();
    double getCost();
    double getTotalCost();
};


#endif //HOMEWORK_CH7_QN7_INVENTORY_H
```

// END of *Inventory.h*.

Inventory.cpp

```cpp
//
// Created by TheLoneWoof on 10/8/17.
//

#include "Inventory.h"

// Private Methods
void Inventory::setTotalCost() {
    totalCost = quantity * cost;
}

// Constructors
Inventory::Inventory() {
    itemNumber = quantity = 0;
    cost = 0;
}

Inventory::Inventory(int n, int qty, double cst) {
    itemNumber = n;
    quantity = qty;
    cost = cst;
}

// Mutator Methods
void Inventory::setItemNumber(int n) {
    itemNumber = n;
}

void Inventory::setQuantity(int qty) {
    quantity = qty;
    setTotalCost();
}

void Inventory::setCost(double itemCost) {
    cost = itemCost;
    setTotalCost();
}

// Accessor Methods
int Inventory::getItemNumber() {
    return itemNumber;
}

int Inventory::getQuantity() {
    return quantity;
}

double Inventory::getCost() {
    return cost;
}

double Inventory::getTotalCost() {
    return totalCost;
}
```

// END of *Inventory.cpp*.