Sample Solutions of Tutorial 3 – Class and Object

Question 1

```cpp
#include <iostream>
#include <string>

class Student {
private:
    std::string name;  // Stores the name of the student
    int age;           // Stores the age of the student
    double gpa;        // Stores the GPA of the student

public:
    // Constructor: Initializes name, age, and GPA
    Student(std::string studentName, int studentAge, double studentGPA) {
        name = studentName;
        age = studentAge;
        gpa = studentGPA;
    }

    // Display function: Prints student details
    void displayDetails() const {
        std::cout << "Student Name: " << name << std::endl;
        std::cout << "Age: " << age << std::endl;
        std::cout << "GPA: " << gpa << std::endl;
    }

    // Getters - Return private member values
    std::string getName() const { return name; }
    int getAge() const { return age; }
    double getGPA() const { return gpa; }

    // Setters - Modify private member values
    void setName(std::string newName) { name = newName; }
    void setAge(int newAge) { age = newAge; }
    void setGPA(double newGPA) { gpa = newGPA; }
};

int main() {
    // Creating Student objects
    Student student1("Alice", 20, 3.8);
    Student student2("Charlie", 19, 3.5);

    // Display details of students
    std::cout << "Initial Student Details:\n";
    student1.displayDetails();
    std::cout << std::endl;
    student2.displayDetails();
    std::cout << std::endl;
```

```
    // Modify student1 details using setters
    student1.setName("Bob");
    student1.setAge(22);
    student1.setGPA(3.9);

    // Display updated details
    std::cout << "Updated Student Details:\n";
    student1.displayDetails();

    return 0;
}
```

Question 2

The major issue here is that the default copy constructor automatically provided by the compiler will be used when doing the copy, but the default copy constructor can only do shallow copy, which can result in two consequences:

1. Unintended change to the original counter here;
2. Repeated deletion of the dynamically allocated memory.

Below is the output of Andy's code:

```
Constructor called with the count as 10

Original Counter:
Visitor Count: 10
counterCopy:
Visitor Count: 10

After modifying copied counter...
Original Counter:
Visitor Count: 12
counterCopy:
Visitor Count: 12


Destructor called with the count being 12
Destructor called with the count being 0
outDebug(73620,0x1f78e4f40) malloc: Double free of object
0x12c605e30
outDebug(73620,0x1f78e4f40) malloc: *** set a breakpoint in
malloc_error_break to debug
```

The way to fix it is to implement a **user-defined copy constructor to do deep copy**. Below is the sample code:

```
    // User-defined Copy Constructor (Deep Copy)
    VisitorCounter(const VisitorCounter& other) {
        count = new int(*other.count); // Allocates new memory and copies value
        cout << "Deep Copy Constructor Called." << endl;
    }
```

Question 3

```
#include <iostream>
#include <string>

class Pen {
private:
    std::string color;
    double price;

public:
    // Constructor
    Pen(std::string initialColor, double initialPrice) {
        color = initialColor;
        price = initialPrice;
    }

    // Method to set color (returns *this for method chaining)
    Pen& setColor(std::string newColor) {
        color = newColor;
        return *this; // this: the pointer pointing to the current Pen object. It
makes method chaining possible!
    }

    // Method to set price (returns *this for method chaining)
    Pen& setPrice(double newPrice) {
        price = newPrice;
        return *this; // this: the pointer pointing to the current Pen object. It
makes method chaining possible!
    }

    // Method to display pen details
    void display() const {
        std::cout << "Pen Color: " << color << std::endl;
        std::cout << "Price: $" << price << std::endl;
    }
};

int main() {
    // Creating a Pen object and using method chaining
    Pen myPen("Blue", 1.5);
```

```
    std::cout<< "The original color and price of the pen: " << std::endl;
    myPen.display();

    std::cout<< std::endl<<"The color and price of the pen after setting: " <<
std::endl;
    myPen.setColor("Red")
         .setPrice(2.0)
         .display(); // Method Chaining here

    return 0;
}
```

Question 4

```
#include <iostream>

class Box {
private:
    double length;  // Stores the length of the box
    double width;   // Stores the width of the box
    double height;  // Stores the height of the box

public:
    // Constructor to initialize the box dimensions
    Box(double l, double w, double h) {
        length = l;
        width = w;
        height = h;
    }

    // Member function that calculates and displays the volume
     void calculateVolume() {
         double volume = length * width * height;
         std::cout << "Box Volume: " << volume << " cubic units" <<
std::endl;
     }

    // Declare a friend function to display private members
    friend void displayDimensions(const Box& b);
};

// Define the friend function (that can access private members of Box)
void displayDimensions(const Box& b) {
     std::cout << "Box Dimensions:\n";
     std::cout << "Length: " << b.length << std::endl;
     std::cout << "Width: " << b.width << std::endl;
     std::cout << "Height: " << b.height << std::endl;
     std::cout << std::endl;
```

```
}

int main() {
    // Creating a Box object
    Box myBox(5.0, 3.0, 2.0);

    // Friend function accessing private data
    displayDimensions(myBox);

    // Member function accessing private data
    myBox.calculateVolume();

    return 0;
}
```