National University of Computer & Emerging Sciences, Karachi

Fall-2019 (CS-Department)

Mid-term II Exam

Course Code: CS-217	**Course Name:** Object-oriented Programming
Instructors: Dr. Abdul Aziz and Syed Zain Ul Hassan	
Student ID:	**Section:**
Date: 4th October, 2019	**Time:** 01:00 pm - 02:00 pm

Instructions:

Attempt all tasks.

This exam has 1 question on 2 pages. It has 9 tasks carrying 4 points each

Return the question paper after exam **Max Points**: 36

Question 1:

Nine-Tails Gift Delivery Service allows users to book and send gifts. The service allows gifts to be delivered at the doorstep reliably.

The service delivers three types of gifts; that are perfumes, chocolate cakes and flowers. Furthermore, happy bundle is another gift type in which any two of these three gift items are sent together as gifts.

Each gift type has a unique category ID i.e. CK1 for cakes, FL1 for flowers, PF1 for perfumes and HB1 for happy bundles.

There is another entity involved, the Taxation Department. The Taxation Department must have access to the revenue and payable tax of Nine-Tails Delivery Service (but the Taxation Department does not need to be able to access any other attributes).

The classes for this scenario are presented in *Figure 1*. Carefully read the assumptions below and then perform the tasks given on the next page.

Assumptions:

1. You may create or modify any variables and override any functions if needed to satisfy the requirements of the question. But their role in the program have to be justifiable.
2. The floral type can either be *Rose*, *Tulip* or *Daisy*. The unit price for each of these types is Rs. 50.
3. Perfumes and chocolate cakes are taxable gift items, but tax is not applicable on flowers.
4. The perfumes available are either from GUCCI, VERSACE or CHANEL. The unit price for perfume is Rs. 1200 for GUCCI, Rs. 1100 for VERSACE and Rs. 950 for CHANEL
5. The value given to weight variable for chocolate cake should be assumed as given in pounds
6. The price for chocolate cake is Rs. 700 per pound
7. The tax rate for perfume is 7% of the price and for chocolate cake it is 4% of the price.

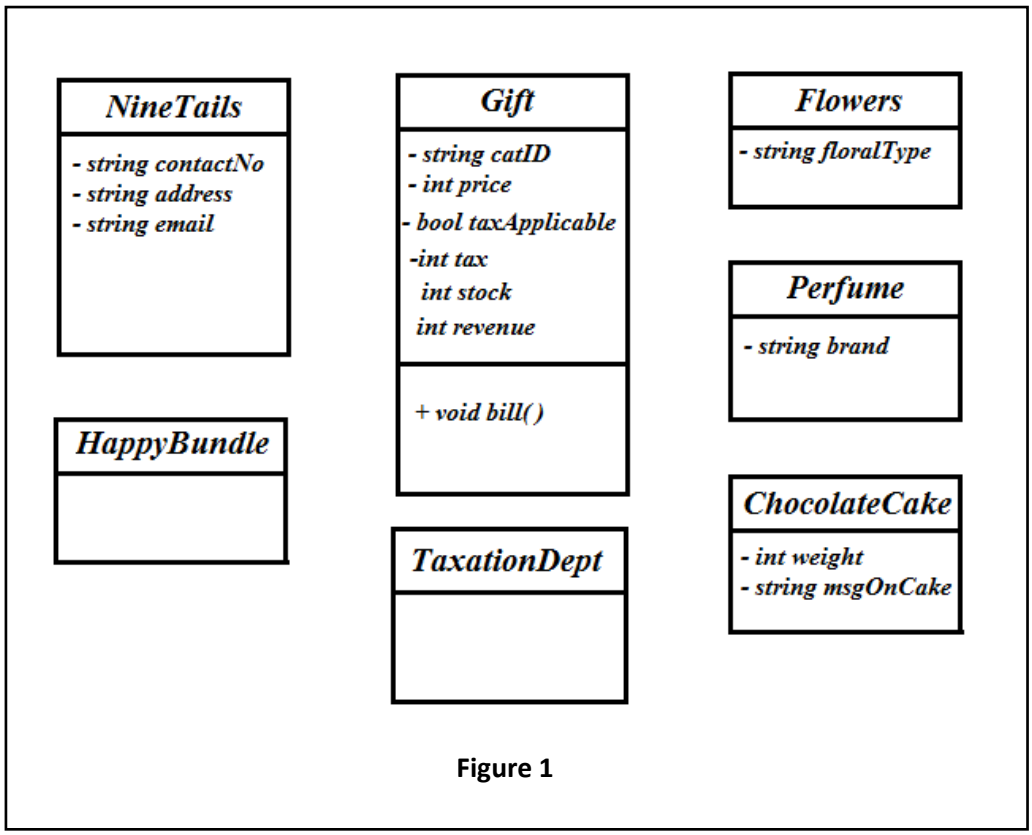

Figure 1

Tasks to be performed:

a. Identify the type(s) of inheritance present between the given classes and for each type of inheritance that you identify, simply list down the classes involved in it.

Hierarchical Inheritance: Gift -> Flowers, Perfume, ChocolateCake
Multiple Inheritance: Flowers, Perfume, ChocolateCake -> HappyBundle

b. Using the inheritance that you identified in Task 1, provide class declarations only.

```
class Gift {        };
class Flowers: virtual public Gift {        };
class Perfume: virtual public Gift {        };
class ChocolateCake: virtual public Gift {        };
class HappyBundle: public Flowers, public Perfume, public ChocolateCake {        };
```

c. In the classes, declare variables and also provide suitable implementation for constructor(s) of each class. Overload the constructors wherever necessary. Also implement setter & getter functions for every private variable.

```
class Gift
{
	string catID;
	bool taxApplicable;
	int tax, price;
	int quantity; // a variable added as per need

	public:
	static int payableTax;
	static int revenue;

	void setPrice(int price) { this->price = price; }
	int getPrice() { return price; }

	void setQuantity(int quantity) { this->quantity = quantity; }
	int getQuantity() { return quantity; }

	virtual void setCatID(string catID) { this->catID = catID; }
	string getCatID() { return catID; }
};

class Flowers: virtual public Gift
{
	string floralType; // can be rose, tulip or daisy
	public:
	static int stock;
	Flowers()
	{
	}

	Flowers(string floralType, int quantity)
	{
		setCatID("FL1");
		setQuantity(quantity);
		this->floralType = floralType;
		setPrice(50);
		stock -= quantity;
		Gift::revenue += getQuantity() * getPrice();
	}

	string getFloralType() { return floralType; }
};
```

```
class Perfume: virtual public Gift
{
	string brand;

	public:
	static int stock;
	Perfume()
	{
	}

	Perfume(string brand, int quantity)
	{
		setCatID("PF1");
		setQuantity(quantity);
		this->brand = brand;
		setPrice(1000);
		stock -= quantity;
		Gift::revenue += getQuantity() * getPrice();
		Gift::payableTax += 0.07 * getPrice();
	}
};

class ChocolateCake: virtual public Gift
{
	int weight; // in pounds
	string msgOnCake;

	public:
	static int stock;
	ChocolateCake()
	{
	}
	ChocolateCake(int weight, string msgOnCake, int quantity)
	{
		setCatID("CK1");
		setQuantity(quantity);
		this->weight = weight;
		this->msgOnCake = msgOnCake;
		setPrice(700);
		stock -= quantity;
		Gift::revenue += getQuantity() * weight * getPrice();
		Gift::payableTax += (getPrice() * 0.04);
	}
};

class HappyBundle: public ChocolateCake,  public Flowers, public Perfume
{
	bool p = false, f = false, c = false; \\ declared as per need
```

```
        public:
        HappyBundle(string brand, string floralType): Perfume(brand, 1), Flowers(floralType, 1)
        {
                setCatID("HB1");
                p = true;
                f = true;
        }

        HappyBundle(string brand, int weight, string msgOnCake): Perfume(brand, 1),
        ChocolateCake(weight, msgOnCake, 1)
        {
                setCatID("HB1");
                p = true;
                c = true;
        }

        HappyBundle(int weight, string msgOnCake, string floralType): Flowers(floralType, 1),
        ChocolateCake(weight, msgOnCake, 1)
        {
                setCatID("HB1");
                f = true;
                c = true;
        }
};
```

d. Override the functions wherever necessary and show their implementations (in both the parent and child classes).

```
class Gift
{
        // other code
        public:
        virtual int bill() = 0;
};

class Flowers: virtual public Gift
{
        // other code
        public:
        int bill()
        {
                return getQuantity() * getPrice();

        }
};

class Perfume: virtual public Gift
{
```

```
        // other code
        public:
        int bill()
        {
                return getQuantity() * getPrice();

        }
};

class ChocolateCake: virtual public Gift
{
        //other code
        public:
        int bill()
        {
                return getQuantity() * weight * getPrice();
        }
};

class HappyBundle: public ChocolateCake,  public Flowers, public Perfume
{
        // other code
        public:
        int bill()
        {
                if(p && f)
                {
                        return Perfume::bill() + Flowers::bill();
                        //cout << "Happy Bundle: " << Perfume::bill() + Flowers::bill();
                }
                else if(p && c)
                {
                        return Perfume::bill() + ChocolateCake::bill();
                        //cout << "Happy Bundle: " << Perfume::bill() + ChocolateCake::bill();
                }
                else
                {
                        return Flowers::bill() + ChocolateCake::bill();
                        //cout << "Happy Bundle: " << Flowers::bill() + ChocolateCake::bill();
                }

        }
};
```

e. Write the code for calling any function of the ***Perfume*** class using pointer variable of its parent class.

```
Gift* g = new Perfume("CHANEL", 2);
cout << "Perfume: " << g->bill() << endl;
```

f. Override the variable ***tax*** in ***Perfume*** class and make it a constant with value 0.07 (for 7% tax).

```
class Perfume: virtual public Gift
{
        // other code
        const int tax = 0.07;
}
```

g. For keeping track of the inventory, there must be a mechanism to find how much of each individual gift item is remaining in stock. Also provide mechanism to see the current revenue (overall profit). Keep in mind that each gift category has a different price.

```
class Gift
{
        // other code
        public:
        static int revenue;
};
int Gift::revenue = 0;     // initial profit/revenue

class Flowers: virtual public Gift
{
        // other code
        public:
        static int stock;
};
int Flowers::stock = 50; // assuming 50 is the max stock limit

class Perfume: virtual public Gift
{
        // other code
        public:
        static int stock;
};
int Perfume::stock = 50; // assuming 50 is the max stock limit

class ChocolateCake: virtual public Gift
{
        // other code
        public:
        static int stock;
};
int Perfume::stock = 50; // assuming 50 is the max stock limit
```

h. Provide a copy constructor for copying objects of ***Flowers*** class.

```
Flowers(Flowers& ob)
{
        setCatID(ob.getCatID());
        setPrice(ob.getPrice());
        floralType = ob.floralType;
        setQuantity(ob.getQuantity());
        stock -= ob.getQuantity();
        Gift::revenue += ob.getQuantity() * ob.getPrice();
}
```

i. Provide mechanism for ***TaxationDept*** class to access the revenue and payable tax amount of Nine-Tails Delivery Service.

```
class NineTails
{
        string contactNo, officeAddress, email;
        int tax, revenue;

        public:
        NineTails()
        {
                tax = Gift::payableTax;
                revenue = Gift::revenue;
        }

        friend class TaxationDept;
};


class TaxationDept
{
        NineTails nt;

        public:
        void calculateTax()
        {
                cout << "Tax Payable: " << nt.tax << endl;
                cout << "Revenue: " << nt.revenue << endl;
        }
};
```

~ Good luck!