SESSION NOTES and EXAMPLES

- Copy Constructor

2 types of Copy Constructor

1. Default Copy Constructor → we use an assignment operator (=)
2. User-defined Copy Constructor → class_name(const class_name&)

- Destructor: Each class may have only one destructor.
- 'this' pointer and arrow (→) sign
- static members and methods
- Constant (const) members and methods

EXAMPLE

```
//rectangle.h
class rectangle{
private:
	int length, width;
	static int counter; //they do not belong to any specific object.But they belong to the whole
class
                                                    //all objects of the class may share it!!!
                                                    // should be initialized outside the class!!!!
public:
	rectangle()
	{
		this->length = 0;
		this->width = 0;
		counter++;
	}
	rectangle(int length, int width)
	{
		this->length = length;
		this->width = width;
		counter++;
	}
	rectangle(const rectangle & robj)
	{
		this->length = robj.length;
		this->width = robj.width;
		counter++;
	}
	~rectangle()
	{
		cout << "The object having values " << this->length << " " << this->width
			<< " has been destroyed." << endl;
```

```
		counter--;
	}
	int getLength() const
	{
		return this->length;
	}
	int getWidth() const
	{
		return this->width;
	}
	void setLength(int length)
	{
		this->length = length;
	}
	void setWidth(int width)
	{
		this->width = width;
	}
	int getArea()
	{
		int area;
		area = this->length*this->width;
		return area;
	}
	static int getCounter()
	{
		return counter;
	}
};
int rectangle::counter = 0;

//rectangle.cpp
#include<iostream>
using namespace std;
#include"rectangle.h"
void main()
{
	{
		cout << "Initial number of objects:" << rectangle::getCounter() << endl;
		rectangle robj1; //default constructor
		rectangle robj2(50, 30); //parameterized constructor
		rectangle robj3 = robj1; //default (implicit) copy constructor
		rectangle robj4(robj2); //user-defined copy constuctor

		robj1.setLength(40);
		robj1.setWidth(20);

		robj2.setLength(60);
```

```
        cout << robj1.getLength() << " " << robj1.getWidth() <<" "<< robj1.getArea()<< endl;
        cout << robj2.getLength() << " " << robj2.getWidth() <<" "<< robj2.getArea() << endl;
        cout << robj3.getLength() << " " << robj3.getWidth() <<" "<< robj3.getArea() << endl;
        cout << robj4.getLength() << " " << robj4.getWidth() <<" "<< robj4.getArea() << endl;

        cout << "Final number of objects:" << rectangle::getCounter() << endl;
    }//destructor exectues for each object
cout << "Now, after destructor executed the number of objects:" << rectangle::getCounter() << endl;
system("pause");
}
```

Exercises

1. Write a C++ code to

a. create 'Sale' class

Sale
taxRate:double total:double calcSale(double):void
Sale( ) Sale(trate:double,cost:double) Sale(cost:double) Sale(const Sale &sobj) ~Sale( ) GetTotal():double

- **calcSale ( ):** Private method that calculates the total of the sale when necessary.
  - The formula to calculate total is: (total= cost + (cost *taxRate)
- **Default Constructor:** Initializes taxRate and total with default values (preferabbly zero (0))
- **Parameterized Constructor (with two parameters):** Accepts taxRate and cost as parameters and initializes the taxRate and uses cost to calculate the total by calling calcSale (private)method.
- **Parameterized Constructor (with single parameter):** Accepts cost as parameter and sets taxRate to zero. Since the sale is tax free, to total of the sale should be equal to the cost.
- **Copy Constructor:** Copies the taxRate and total values of old object to new object.
- **Destructor:** Displays an appropriate message when an object is destroyed.

b. Write a main( ) function to test the class. (Create (at least) four objects that each one should invoke different types of constructors)

SOLUTION

```
//sale.h
class sale{
	double taxrate, total;

	void calcSale(double cost)
	{
		this->total = cost + (cost*this->taxrate);
	}
public:
	static int counter;
	sale()
	{
		this->taxrate = 0;
		this->total = 0;
		counter++;
	}
	sale(double taxrate, double cost)
	{
		this->taxrate = taxrate;
		calcSale(cost);
		counter++;
	}
	sale(double cost)
	{
		this->taxrate = 0.0;
		this->total = cost;
		counter++;
	}
	sale(const sale& sobj)
	{
		this->taxrate = sobj.taxrate;
		this->total = sobj.total;
		counter++;
	}
	~sale()
	{
		cout << "The object with total=" << this->total << " has been destroyed." << endl;
		counter--;
	}
	double getTotal()
	{
		return this->total;
	}

};
int sale::counter; //by default the initial value is ZERO (0)
```

```
//sale.cpp
#include<iostream>
using namespace std;
#include"sale.h"
void main()
{
	{
		cout << "The number of sale objects:" << sale::counter<<endl;
		sale sobj1;
		sale sobj2(0.3, 550.50);
		sale sobj3(85.75);
		sale sobj4(sobj1);
		cout << sobj1.getTotal() << endl;
		cout << sobj2.getTotal() << endl;
		cout << sobj3.getTotal() << endl;
		cout << sobj4.getTotal() << endl;
		cout << "The number of sale objects:" << sale::counter<<endl;
	}
	cout << "The number of sale objects:" << sale::counter << endl;
	system("pause");
}
```

2. Write a C++ code to

Product
prodNo:int name:string year:int price:double prodCount:static int

Product( ) // default constructor
~Product( ) //destructor
Product(const &Product) //copy constructor
int getProdNo( ) const
string getName( ) const
int getYear( ) const
static int getprodCount( )
double getPrice( ) const
void setName(string)
void setYear(int)
void setPrice(double)

a) Create 'product.h' that will include the following;

- The Default Constructor
  - accepts data from keyboard for private data members (name, year and price),
  - increases prodCount by 1,
  - assigns the current value of prodCount to prodNo so that each product can have a unique value.
- Destructor decrements the prodCount by one and prints the name of the object that has been destroyed. (i.e: "Product **Anything** has been deleted!")
- Getter methods should return the corresponding data member.
- Setter methods should receive parameter from main() and change the corresponding data member.
  - A year value is invalid if it is less than 0 or greater than 4000. If the parameter is an invalid, set year to 0 (zero).
  - A price value is invalid if it is less than 0.0. If the parameter is an invalid, set price to 0.0 (zero).
- Initialize the static data member prodCount to 0.

b) Create 'product.cpp' <main( ) function should include the following steps:>

- Include necessary header files,
- Create 2 product objects,
- Display the name of the both objects,
- Create an array object for 5 products,
- Display the prodno, name, year and price of the array object,
- Display the total count of all products,
- Delete all objects that you have created.