



# **OBJECT ORIENTED PROGRAMMING**

## **LAB # 10**

## **ASSIGNMENT**

**NAME: S.M.IRTIZA**

**ROLL NO. : 22K-4638**

**CLASS: 2-F**

**QUESTION#01:**

Write a C++ program in which a MobilePhone class with a constructor that takes two parameters, manufacturer and model, and initializes the corresponding member variables. We then define three overloaded member functions: call, sendMessage (with two parameters), and sendMessage (with three parameters). The call function takes a single parameter phoneNumber and simply prints a message indicating that the phone is calling that number. The first sendMessage function takes two parameters, phoneNumber and message, and prints a message indicating that the phone is sending a message to that number with the specified message. The second sendMessage function takes three parameters, phoneNumber, message, and isEncrypted, and checks if the isEncrypted parameter is true. If it is, the function prints a message indicating that the phone is sending an encrypted message to the specified number with the specified message. If not, it calls the first sendMessage function to send a regular message. In the main function, we create an object of the MobilePhone class with the manufacturer "Nokia" and model "3310". We then test the call and sendMessage functions with different parameters, including a regular message and an encrypted message.

**CODE:**

```
//NAME: S.M.IRTIZA | NU ID: 22K-4638
```

```
#include<iostream>
```

```
#include<string>
```

```
using namespace std;
```

```
class MobilePhone{
```

```
string manufacturer;
```

```
string model;
```

```
string getManufacturer()
```

```
{
```

```
return this->manufacturer;
```

```
}
```

```
void setManufacturer(string manufacturer)
```

```
{
```

```
this->manufacturer = manufacturer;
```

```
}
```

```
string getModel()
```

```
{
```

```
return this->model;
```

```
}
```

```
void setModel(string model)
```

```
{
```

```
this->model = model;
```

```
}
```

```
public:
```

```
MobilePhone(){}
```

```
MobilePhone(string a,string b): manufacturer(a),model(b){}
```

```
void call(string a){
```

```
cout<<"calling "<<a<<endl;
```

```
}
```

```
void sendMessage(string a,string message){
```

```
cout<<"sending message to "<<a<<": "<<message<<endl;
}
void sendMessage(string a,string message, bool isEncrypted){
if(isEncrypted==true){
cout<<"sending encrpyted message to "<<a<<": "<<message<<endl;
}
else{
sendMessage(a,message);
}
}
};

int main(){
cout<<"NAME: S.M.IRTIZA | NU ID: 22K-4638"<<endl;
MobilePhone obj("Nokia","3310");
obj.call("0318-1122-703");
obj.sendMessage("0318-1122-703","Hello World!");
obj.sendMessage("0318-1122-703","Hello World!",true);
return 0;
}
```

## OUTPUT :

```
NAME: S.M.IRTIZA | NU ID: 22K-4638
calling 0318-1122-703
sending message to 0318-1122-703: Hello World!
sending encrpyted message to 0318-1122-703: Hello World!
```

**QUESTION # 02:**

Write a C++ program in which a base class RAM and two derived classes, DDR3 and DDR4. The RAM class has two member variables, type and capacity, which store the type and capacity of the RAM. It also has a virtual member function printDetails, which prints the details of the RAM. The DDR3 class is a derived class of RAM and has a constructor that takes the capacity of the DDR3 RAM as a parameter. It also overrides the printDetails function to print the details of the DDR3 RAM. The DDR4 class is also a derived class of RAM and has a constructor that takes the capacity of the DDR4 RAM as a parameter. It also overrides the printDetails function to print the details of the DDR4 RAM. In the main function, we create objects of the DDR3 and DDR4 classes with different capacities, and then call the printDetails function for each object.

**CODE:**

```
//NAME: S.M.IRTIZA | NU ID: 22K-4638
#include<iostream>
#include<string>
using namespace std;

class RAM{
    string type;
    int capacity;

public:
    RAM(){}
    RAM(string a, int b):type(a),capacity(b){}

    string getType()
    {
        return this->type;
    }

    void setType(string type)
    {
        this->type = type;
    }

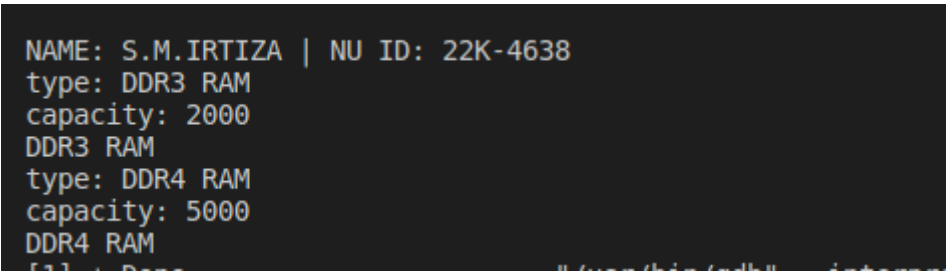
    int getCapacity()
    {
        return this->capacity;
    }

    void setCapacity(int capacity)
    {
        this->capacity = capacity;
    }

    virtual void printDetails(){
        cout<<"type: "<<type<<endl;
        cout<<"capacity: "<<capacity<<endl;
```

```
}  
};  
  
class DDR3:public RAM{  
public:  
DDR3(){}  
DDR3(int b):RAM("DDR3 RAM",b){}  
void printDetails(){  
RAM::printDetails();  
cout<<"DDR3 RAM "<<endl;  
}  
};  
  
class DDR4:public RAM{  
public:  
DDR4(){}  
DDR4(int b):RAM("DDR4 RAM",b){}  
void printDetails(){  
RAM::printDetails();  
cout<<"DDR4 RAM "<<endl;  
}  
};  
  
int main(){  
cout<<"NAME: S.M.IRTIZA | NU ID: 22K-4638"<<endl;  
DDR3 obj1(2000);  
DDR4 obj2(5000);  
obj1.printDetails();  
obj2.printDetails();  
return 0;  
}
```

## OUTPUT :

A screenshot of a terminal window with a dark background and light-colored text. The output shows the program's execution results, including the user's name and ID, and the details of two RAM objects (DDR3 and DDR4) created. The text is as follows:

```
NAME: S.M.IRTIZA | NU ID: 22K-4638  
type: DDR3 RAM  
capacity: 2000  
DDR3 RAM  
type: DDR4 RAM  
capacity: 5000  
DDR4 RAM
```

**QUESTION # 03:**

Write a C++ program in which a class called Mobile represents a mobile phone. The class has two private member variables, brand and model, which store the brand and model of the mobile phone. Mobile class has a constructor that takes the brand and model as parameters, and two getter functions that return the brand and model. The interesting part of this program is the operator overloading. Overload the == operator to compare two Mobile objects. In this case, consider two mobile phones to be the same if they have the same brand and model. Define the overloaded == operator as a member function of the Mobile class that takes another Mobile object as a parameter. In the main function, create three Mobile objects, two of which have the same brand and model. Then compare these objects using the overloaded == operator.

**CODE:**

```
//NAME: S.M.IRTIZA | NU ID: 22K-4638
#include<iostream>
#include<string.h>
using namespace std;

class Mobile{
string brand;
string model;

public:
Mobile(){}
Mobile(string a,string b):brand(a),model(b){}

string getBrand(){
return brand;
}
string getModel(){
return model;
}

bool operator==(const Mobile& obj){
    if((brand==obj.brand)&& (model==obj.model)){
        return true;
    }
    else{
        return false;
    }
}
};

int main(){
cout<<"NAME: S.M.IRTIZA | NU ID: 22K-4638"<<endl;
Mobile M1("motorola","E9"), M2("iphone","i11"),M3("motorola","E9");
cout<<"(M1==M3):"<<(M1==M3)<<endl;
cout<<"(M1==M2):"<<(M1==M2)<<endl;
return 0;
}
```

**OUTPUT :**

```
NAME: S.M.IRTIZA | NU ID: 22K-4638  
(M1==M3):1  
(M1==M2):0
```

## QUESTION # 04

Write a C++ program in which a base class base class Compiler and two derived classes, CCompiler and CppCompiler, which represent C and C++ compilers, respectively. Each of these derived classes overrides the compile function to provide its own implementation of the compiler. Also define a class TurboCompiler, which derives from both CCompiler and CppCompiler. This class provides its own implementation of the compile function, which calls both the CCompiler::compile and CppCompiler::compile functions. This program also demonstrates the concept of the diamond problem, which arises when a class inherits from two or more classes that in turn inherit from a common base class. In this case, TurboCompiler inherits from both CCompiler and CppCompiler, which both inherit from Compiler. the main function begins by creating an object of the TurboCompiler class named turbo. Then, call the compile function on turbo three times. The first two calls use the scope resolution operator :: to explicitly call the compile function of the CCompiler and CppCompiler classes, respectively. The third call to compile calls the compile function of the TurboCompiler class, which in turn calls the compile functions of both CCompiler and CppCompiler

### CODE:

```
//NAME: S.M.IRTIZA | NU ID: 22K-4638
#include<iostream>
#include<string.h>
using namespace std;
class Compiler{
public:
virtual void CompilerFunction(){
cout<<"compiler"<<endl;
}
};

class CCompiler:public Compiler{
public:
void CompilerFunction(){
cout<<"C Compiler"<<endl;
}
};

class CppCompiler:public Compiler{
public:
void CompilerFunction(){
cout<<"C++ Compiler"<<endl;
}
};

class TurboCompiler: virtual public CCompiler,virtual public CppCompiler{
public:
void CompilerFunction(){
CCompiler::CompilerFunction();
CppCompiler::CompilerFunction();
cout<<"Turbo Compiler"<<endl;
}
};

int main(){
cout<<"NAME: S.M.IRTIZA | NU ID: 22K-4638"<<endl;
TurboCompiler turbo;
turbo.CCompiler::CompilerFunction();
turbo.CppCompiler::CompilerFunction();
turbo.CompilerFunction();
return 0;
}
```



```
}
```

**OUTPUT :**

```
NAME: S.M.IRTIZA | NU ID: 22K-4638
C Compiler
C++ Compiler
C Compiler
C++ Compiler
Turbo Compiler
```

**QUESTION# 05:**

Write a C++ program in which a base class Printer and two derived classes, InkjetPrinter and LaserPrinter, which represent inkjet and laser printers, respectively. Each of these derived classes overrides the print function to provide its own implementation of printing. In the main function, create two objects of the InkjetPrinter and LaserPrinter classes, respectively. Then, create a pointer to the Printer class named printer. then set the printer pointer to point to the inkjet object and call the print function on printer. Since printer is a pointer to the base class Printer, late binding is used to determine which version of the print function to call based on the actual object being pointed to at runtime. Similarly, set the printer pointer to point to the laser object and call the print function on printer. Again, late binding is used to determine which version of the print function to call based on the actual object being pointed to at runtime.

**CODE:**

```
//NAME: S.M.IRTIZA | NU ID: 22K-4638
```

```
#include<iostream>
```

```
#include<string>
```

```
using namespace std;
```

```
class Printer{
public:
virtual void print(){
cout<<"printer"<<endl;
}
};
```

```
class InkjetPrinter:public Printer{
public:
void print(){
cout<<"Inkjet Printer"<<endl;
}
};
class LaserPrinter:public Printer{
public:
void print(){
cout<<"Laser printer"<<endl;
}
};
```

```
int main(){
cout<<"NAME: S.M.IRTIZA | NU ID: 22K-4638"<<endl;
InkjetPrinter I1;
LaserPrinter L1;
Printer* p1;
p1=&I1;
p1->print();
p1=&L1;
p1->print();
return 0;
}
```

**OUTPUT :**

```
NAME: S.M.IRTIZA | NU ID: 22K-4638  
Inkjet Printer  
Laser printer
```