



OBJECT ORIENTED PROGRAMMING

LAB # 10

ASSIGNMENT

NAME: S.M.IRTIZA

ROLL NO. : 22K-4638

CLASS: 2-F

QUESTION#01:

1 Create a class hierarchy for a car dealership. Start with a base class called Car and create subclasses

for specific types of cars, such as Sedan and SUV.

Data members of classes:

- Car
 - brand (string): the brand of the car
 - model (string): the model of the car
 - price (float): the price of the car in USD
 - color (string): the color of the car
 - features (list): a list of Feature objects representing the features of the car
- Sedan
 - fuel_type (string): the type of fuel the sedan uses
 - engine_capacity (float): the engine capacity of the sedan in liters
- SUV
 - seating_capacity (int): the maximum number of passengers the SUV can seat
 - cargo_capacity (float): the maximum cargo capacity of the SUV in cubic feet

Methods:

- Car
 - Default and parameterized constructor
 - add_feature(self, feature): adds a Feature object to the car's list of features
 - get_features(self): returns a list of Feature objects representing the car's features
- Sedan.
 - Default and parameterized constructor
 - start_engine(self): starts the engine of the sedan
 - stop_engine(self): stops the engine of the sedan
- SUV
 - Default and parameterized constructor
 - open_trunk(self): opens the trunk of the SUV
 - close_trunk(self): closes the trunk of the SUV

CODE:

```
//NAME: S.M.IRTIZA | ROLL NO.: 22K-4638
```

```
#include<iostream>
```

```
#include<string>
```

```
#include<vector>
```

```
using namespace std;
```

```
class Feature{
```

```
    public:
```

```
    string name;
```

```
    Feature(){}
```

```
    Feature(string name):name(name){}
```

```
};
```

```
class Car{
```

```
    string brand;
```

```
    string model;
```

```
    float price;
```

```
    string color;
```

```
vector<Feature> features;
```

```
public:
```

```
Car(){};
```

```
Car(string brand,string model,float price,string  
color):brand(brand),model(model),price(price),color(color){}
```

```
    string getBrand() {  
        return this->brand;  
    }
```

```
    void setBrand(string brand) {  
        this->brand = brand;  
    }
```

```
    string getModel() {  
        return this->model;  
    }
```

```
    void setModel(string model) {  
        this->model = model;  
    }
```

```
    float getPrice() {  
        return this->price;  
    }
```

```
    void setPrice(float price) {  
        this->price = price;  
    }
```

```
    string getColor() {  
        return this->color;  
    }
```

```
    void setColor(string color) {  
        this->color = color;  
    }
```

```
    vector<Feature> getFeatures() {  
        return this->features;  
    }
```

```
    void setFeatures(vector<Feature> features) {  
        this->features = features;  
    }
```

```
void add_feature(Feature feature){  
    features.push_back(feature);
```

```
}

vector<Feature> getFeature(){
    return features;
}

};

class Sedan:public Car{
    string fuel_type;
    float engine_capacity;

public:
    Sedan(){}
    Sedan(string brand,string model,float price,string color,string fuel_type,float
engine_capacity):Car(brand,model,price,color),fuel_type(fuel_type),engine_capacity(engine
_capacity){}
    string getFuel_type()
    {
        return this->fuel_type;
    }

    void setFuel_type(string fuel_type)
    {
        this->fuel_type = fuel_type;
    }

    float getEngine_capacity()
    {
        return this->engine_capacity;
    }

    void setEngine_capacity(float engine_capacity)
    {
        this->engine_capacity = engine_capacity;
    }

    void start_engine(){
        cout<<"Engine started"<<endl;
    }

    void stop_engine(){
        cout<<"Engine stopped"<<endl;
    }

};
```

```
class SUV: public Car{
    int seating_capacity;
    float cargo_capacity;

public:
    SUV(){}
    SUV(string brand,string model,float price,string color,int seating_capacity,float
cargo_capacity):Car(brand,model,price,color),seating_capacity(seating_capacity),cargo_ca
pacity(cargo_capacity){}

    int getSeating_capacity()
    {
        return this->seating_capacity;
    }

    void setSeating_capacity(int seating_capacity)
    {
        this->seating_capacity = seating_capacity;
    }

    float getCargo_capacity()
    {
        return this->cargo_capacity;
    }

    void setCargo_capacity(float cargo_capacity)
    {
        this->cargo_capacity = cargo_capacity;
    }

    void open_trunk(){
        cout<<"Trunk opened"<<endl;
    }
    void close_trunk(){
        cout<<"Trunk closed"<<endl;
    }
};

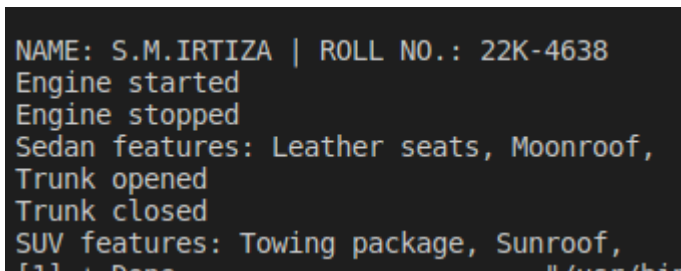
int main(){
    cout<<"NAME: S.M.IRTIZA | ROLL NO.: 22K-4638"<<endl;
    Sedan mySedan("Toyota", "Corolla", 25000.00, "Red", "Gasoline", 2.0);
    mySedan.add_feature(Feature("Leather seats"));
    mySedan.add_feature(Feature("Moonroof"));
    mySedan.start_engine();
    mySedan.stop_engine();
    vector<Feature> sedanFeatures = mySedan.getFeatures();
```

```
cout << "Sedan features: ";
for (auto feature : sedanFeatures) {
    cout << feature.name << ", ";
}
cout << endl;

SUV mySUV("Jeep", "Grand Cherokee", 35000.00, "Black", 5, 50.0);
mySUV.add_feature(Feature("Towing package"));
mySUV.add_feature(Feature("Sunroof"));
mySUV.open_trunk();
mySUV.close_trunk();
vector<Feature> suvFeatures = mySUV.getFeatures();
cout << "SUV features: ";
for (auto feature : suvFeatures) {
    cout << feature.name << ", ";
}
cout << endl;

return 0;
}
```

OUTPUT:

A screenshot of a terminal window showing the output of a C++ program. The text is as follows:

```
NAME: S.M.IRTIZA | ROLL NO.: 22K-4638
Engine started
Engine stopped
Sedan features: Leather seats, Moonroof,
Trunk opened
Trunk closed
SUV features: Towing package, Sunroof,
[1] Done
```

QUESTION # 02:

A company called TechCo needs a software system to manage its employees. The system should include classes to represent employees, departments, and managers. Each employee belongs to a specific department, and each department has one or more managers.

- Create a base class called Employee that includes the following properties:
 - employee_id (int): the unique ID assigned to the employee
 - name (string): the name of the employee
 - salary (float): the salary of the employee
 - position (string): the position of the employee
 - department (string): the name of the department the employee belongs to
 - manager (string): the name of the employee's manager
- Create a subclass called Department that inherits using protected mode from the Employee class and includes the following properties:
 - department_id (int): the unique ID assigned to the department
 - department_name (string): the name of the department
 - employees (list): a list of employees in the department
 - manager (string): the name of the department manager
- Create a subclass called Manager that inherits from Department and includes the following properties:
 - manager_id (int): the unique ID assigned to the manager
 - manager_name (string): the name of the manager
 - subordinates (list): a list of employees managed by the manager
- Create methods for each class to manage the employees:
 - Employee:
 - Default and parameterized constructor
 - assign_department(self, department): assigns the employee to the specified department
 - Department:
 - Default and parameterized constructor
 - add_employee(self, employee): adds an employee to the department
 - get_employees(self): returns a list of employees in the department
 - Manager:
 - Default and parameterized constructor
 - add_subordinate(self, employee): adds an employee to the manager's list of subordinates
 - get_subordinates(self): returns a list of subordinates managed by the manager.
- Create a program that demonstrates how the class hierarchy works. The program should allow the user to create new employees, departments, and managers, add and remove employees from departments, and add and remove subordinates from managers. The program should also allow the user to view the employees in a specific department, view the subordinates of a specific manager, and search for employees by various criteria (such as name, position, and salary).

CODE:

```
//NAME: S.M.IRTIZA | ROLL NO.: 22K-4638
#include<iostream>
#include<string>
#include<vector>
using namespace std;
class Departement;
class Employee{
    int employee_id;
```

```
string name;  
float salary;  
string position;  
string departement;  
string manager;
```

```
public:
```

```
    Employee(){}  
    Employee(int employee_id,string name,float salary,string position,string departement,string
```

```
manager){
```

```
    this->employee_id=employee_id;  
    this->name=name;  
    this->salary=salary;  
    this->position=position;  
    this->departement=departement;  
    this->manager=manager;  
}
```

```
int getEmployee_id()
```

```
{  
    return this->employee_id;  
}
```

```
void setEmployee_id(int employee_id)
```

```
{  
    this->employee_id = employee_id;  
}
```

```
string getName()
```

```
{  
    return this->name;  
}
```

```
void setName(string name)
```

```
{  
    this->name = name;  
}
```

```
float getSalary()
```

```
{  
    return this->salary;  
}
```

```
void setSalary(float salary)
```



```
{  
    this->salary = salary;  
}
```

```
string getPosition()  
{  
    return this->position;  
}
```

```
void setPosition(string position)  
{  
    this->position = position;  
}
```

```
string getDepartement()  
{  
    return this->departement;  
}
```

```
void setDepartement(string departement)  
{  
    this->departement = departement;  
}
```

```
string getManager()  
{  
    return this->manager;  
}
```

```
void setManager(string manager)  
{  
    this->manager = manager;  
}
```

```
void assign_departement(string departement){  
    this->departement=departement;  
}  
};
```

```
class Department: protected Employee{  
    int departement_id;  
    string departement_name;  
    vector<Employee> employees;
```

```
    string manager;
```

```
public:
```

```
    Department(){};
```

```
    Department(int employee_id,string name,float salary,string position,string manager,int  
departement_id,string departement_name):Employee(employee_id, name,  
salary,position,departement_name, manager) {  
        this->departement_id=departement_id;  
        this->departement_name=departement_name;  
        this->manager=name;  
    }
```

```
    int getDepartement_id()
```

```
    {  
        return this->departement_id;  
    }
```

```
    void setDepartement_id(int departement_id)
```

```
    {  
        this->departement_id = departement_id;  
    }
```

```
    string getDepartement_name()
```

```
    {  
        return this->departement_name;  
    }
```

```
    void setDepartement_name(string departement_name)
```

```
    {  
        this->departement_name = departement_name;  
    }
```

```
    vector<Employee> getEmployees()
```

```
    {  
        return this->employees;  
    }
```

```
    void setEmployees(vector<Employee> employees)
```

```
    {  
        this->employees = employees;  
    }
```

```
    string getManager()
```

```
    {
```

```
        return this->manager;
    }

    void setManager(string manager)
    {
        this->manager = manager;
    }

    void add_employee(Employee employee){
        employee.assign_departement(departement_name);
        employees.push_back(employee);
    }

    vector<Employee> get_employees(){
        return employees;
    }
};

class Manager: public Department{
    int manager_id;
    string manager_name;
    vector<Employee> subordinates;
public:
    Manager(){}
    Manager(int employee_id,string name,float salary,string position,int departement_id,string
departement_name,int manager_id,string
manager_name):Department( employee_id,name,salary,position,
manager_name,departement_id,departement_name){
        this->manager_id=manager_id;
        this->manager_name=manager_name;
    }
    int getManager_id()
    {
        return this->manager_id;
    }

    void setManager_id(int manager_id)
    {
        this->manager_id = manager_id;
    }

    string getManager_name()
    {
        return this->manager_name;
    }
}
```

```

void setManager_name(string manager_name)
{
    this->manager_name = manager_name;
}

void add_subordinate(Employee employee){
    subordinates.push_back(employee);
}

vector<Employee> get_subordinates(){
    return subordinates;
}

};

int main(){
    cout<<"NAME: S.M.IRTIZA | ROLL NO.: 22K-4638"<<endl;
// create new employees
    Employee emp1(1, "John Smith", 50000, "Manager", "", "");
    Employee emp2(2, "Jane Doe", 40000, "Salesperson", "", "");
    Employee emp3(3, "Bob Johnson", 45000, "Accountant", "", "");//-----

    // create new department
    Department dep1(emp1.getEmployee_id(), emp1.getName(), emp1.getSalary(),
emp1.getPosition(), emp1.getName(),1, "Sales");

    // add employees to department
    dep1.add_employee(emp2);
    dep1.add_employee(emp3);//-----

    // create new manager
    Manager mgr1(emp1.getEmployee_id(), emp1.getName(), emp1.getSalary(),
emp1.getPosition(),dep1.getDepartement_id(), dep1.getDepartement_name(),1, "Sarah Lee");

    // add subordinates to manager
    mgr1.add_subordinate(emp2);
    mgr1.add_subordinate(emp3);

    // view employees in a specific department
    vector<Employee> sales_employees = dep1.get_employees();
    cout << "Employees in Sales department:" << endl;
    for (Employee emp : sales_employees) {
        cout << emp.getName() << " (" << emp.getPosition() << ")" << endl;
    }

    // view subordinates of a specific manager
    vector<Employee> sarah_subordinates = mgr1.get_subordinates();
    cout << "Subordinates of Sarah Lee:" << endl;

```

```
for (Employee emp : sarah_subordinates) {  
    cout << emp.getName() << " (" << emp.getPosition() << ")"<<endl;  
}  
return 0;  
}
```

OUTPUT :

```
NAME: S.M.IRTIZA | ROLL NO.: 22K-4638  
Employees in Sales department:  
Jane Doe (Salesperson)  
Bob Johnson (Accountant)  
Subordinates of Sarah Lee:  
Jane Doe (Salesperson)  
Bob Johnson (Accountant)
```

QUESTION # 03:

3. Create four classes: Vehicle, Car, Truck, and Motorcycle with the following specifications:

- The class Vehicle should have data members make, model, and year.
- The classes Car, Truck, and Motorcycle should inherit from the class Vehicle.
- The Car class should have an attribute num_doors indicating the number of doors on the car. The function getdata should get the input from the user of Car Class. The function putdata should print the data of Car Class.
- The Truck class should have an attribute payload_capacity indicating the maximum weight the truck can carry. The function getdata should get the input from the user of Truck Class. The function putdata should print the data of Truck Class.
- The Motorcycle class should have an attribute engine_size indicating the size of the motorcycle's engine in cc. The function getdata should get the input from the user of Motorcycle Class. The function putdata should print the data of Motorcycle Class.

Sequential id's should be assigned to each object being created of the four classes starting from 1.

CODE:

```
//NAME: S.M.IRTIZA | ROLL NO.: 22K-4638
```

```
#include<iostream>
#include<string>
using namespace std;
class Vehicle{
    string make;
    int model;
    int year;

public:
    static int id;
    Vehicle(){}
    Vehicle(string make,int model,int year){
        this->make=make;
        this->model=model;
        this->year=year;
    }
    string getMake()
    {
        return this->make;
    }
    void setMake(string make)
    {
        this->make = make;
    }
    int getModel()
    {
        return this->model;
    }

    void setModel(int model)
    {
        this->model = model;
    }

    int getYear()
    {
        return this->year;
    }

    void setYear(int year)
```

```

    {
        this->year = year;
    }
};
int Vehicle::id(0);

class Car: public Vehicle{
    int num_doors;
    int num;
public:
    Car(){
        ++id;
        num=id;
    }

    int getNum_doors()
    {
        return this->num_doors;
    }

    void setNum_doors(int num_doors)
    {
        this->num_doors = num_doors;
    }
    int getNum(){
        return num;
    }
    void getdata(){
        string make; int model,year;
        cout<<"Enter the maker: "<<endl;
        cin>>make;
        setMake(make);
        cout<<"Enter the model: "<<endl;
        cin>>model;
        setModel(model);
        cout<<"Enter the year: "<<endl;
        cin>>year;
        setYear(year);
        cout<<"Enter the number of door: "<<endl;
        cin>>num_doors;
        setNum_doors(num_doors);
    }

    void putdata(){
        cout<<"Id: "<<getNum()<<endl;
        cout<<"make: "<<getMake()<<endl;
        cout<<"model: "<<getModel()<<endl;
        cout<<"year: "<<getYear()<<endl;
        cout<<"NUM OF DOORS: "<<getNum_doors()<<endl;
    }

};

class Truck: public Vehicle{
    int payload_capacity;
    int num;
public:
    Truck(){

```

```

    ++id;
    num=id;
}

int getPayload_capacity()
{
    return this->payload_capacity;
}

void setPayload_capacity(int payload_capacity)
{
    this->payload_capacity = payload_capacity;
}

int getNum(){
    return num;
}

void getdata(){
    string make;
    int model,year;
    cout<<"Enter the maker: "<<endl;
    cin>>make;
    setMake(make);
    cout<<"Enter the model: "<<endl;
    cin>>model;
    setModel(model);
    cout<<"Enter the year: "<<endl;
    cin>>year;
    setYear(year);
    cout<<"Enter the pay load capacity: "<<endl;
    cin>>payload_capacity;
    setPayload_capacity(payload_capacity);
}

void putdata(){
    cout<<"Id: "<<getNum()<<endl;
    cout<<"make: "<<getMake()<<endl;
    cout<<"model: "<<getModel()<<endl;
    cout<<"year: "<<getYear()<<endl;
    cout<<"pay load capacity: "<<getPayload_capacity()<<endl;
}

};

class Motorcycle: public Vehicle{
    int engine_size;
    int num;
public:
    Motorcycle(){
        ++id;
        num=id;
    }

    int getEngine_size()
    {
        return this->engine_size;
    }

    void setEngine_size(int engine_size)

```



```
{
    this->engine_size = engine_size;
}
int getNum(){
    return num;
}
void getdata(){
    string Make;
    int Model,Year;
    cout<<"Enter the maker: "<<endl;
    cin>>Make;
    setMake(Make);
    cout<<"Enter the model: "<<endl;
    cin>>Model;
    setModel(Model);
    cout<<"Enter the year: "<<endl;
    cin>>Year;
    setYear(Year);
    cout<<"Enter the engine size: "<<endl;
    cin>>engine_size;
    setEngine_size(engine_size);
}

void putdata(){
    cout<<"Id: "<<getNum()<<endl;
    cout<<"make: "<<getMake()<<endl;
    cout<<"model: "<<getModel()<<endl;
    cout<<"year: "<<getYear()<<endl;
    cout<<"Engine Size: "<<getEngine_size()<<endl;
}

};

int main(){
    cout<<"NAME: S.M.IRTIZA | ROLL NO.: 22K-4638"<<endl;
    cout<<"Enter the car details:"<<endl;
    Car c1;
    c1.getdata();
    c1.putdata();
    cout<<"Enter the truck details: "<<endl;
    Truck t1;
    t1.getdata();
    t1.putdata();
    cout<<"Enter the Motorcycle details: "<<endl;
    Motorcycle m1;
    m1.getdata();
    m1.putdata();
    return 0;
}
```

OUTPUT:

```
NAME: S.M.IRTIZA | ROLL NO.: 22K-4638
Enter the car details:
Enter the maker:
honda
Enter the model:
2020
Enter the year:
2018
Enter the number of door:
4
Id: 1
make: honda
model: 2020
year: 2018
NUM OF DOORS: 4
Enter the truck details:
Enter the maker:
shehzor
Enter the model:
1999
Enter the year:
2018
Enter the pay load capacity:
1000
Id: 2
make: shehzor
model: 1999
year: 2018
pay load capacity: 1000
Enter the Motorcycle details:
Enter the maker:
unique
Enter the model:
```

```
make: unique
model: 2018
year: 2020
Engine Size: 70
```

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 | ROLL NO.: 22K-4638
#include<iostream>
#include<string>
#include<vector>
using namespace std;

class Student{
    string name;
    int age;
    string std_Id;
public:
    Student(){}
    Student(string name,int age,string std_Id){
        this->name=name;
        this->age=age;
        this->std_Id=std_Id;
    }

    string getName()
    {
        return this->name;
    }

    void setName(string name)
    {
        this->name = name;
    }

    int getAge()
    {
        return this->age;
    }

    void setAge(int age)
    {
        this->age = age;
    }

    string getStd_Id()
    {
        return this->std_Id;
    }
}
```

```

    }

    void setStd_Id(string std_Id)
    {
        this->std_Id = std_Id;
    }

};

class Honorsstudent: public Student{
    float GPA;
    vector<string>honorclasses;
public:
    Honorsstudent(){}
    Honorsstudent(string Name,int Age,string id,float
    GPA,vector<string>classes):Student(Name, Age,id),honorclasses(classes)
    {
        this->GPA=GPA;
    }

    float getGPA()
    {
        return this->GPA;
    }

    void setGPA(float GPA)
    {
        this->GPA = GPA;
    }

    void setHonorsClasses(vector<string> newClasses) { honorclasses =
newClasses; }
    vector<string> getHonorClasses(){
        return honorclasses;
    }
    void printdata(){
        cout<<"name: "<<getName()<<endl;
        cout<<"age: "<<getAge()<<endl;
        cout<<"ID: "<<getStd_Id()<<endl;
        cout<<"GPA: "<<getGPA()<<endl;
        for (auto& cls : getHonorClasses()) {
            cout << cls << " ";
        }
        cout << endl;
    }

};

int main(){
    cout<<"NAME: S.M.IRTIZA | ROLL NO.: 22K-4638"<<endl;
    cout<<"single level inheritance "<<endl;
    Honorsstudent honorsStudent("Bob", 19, "54321", 3.8, {"Honors", "
Science","MATH"});
    honorsStudent.printdata();
}

```

OUTPUT :

```
NAME: S.M.IRTIZA | ROLL NO.: 22K-4638
single level inheritance
name: Bob
age: 19
ID: 54321
GPA: 3.8
Honors Science MATH
```


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
```