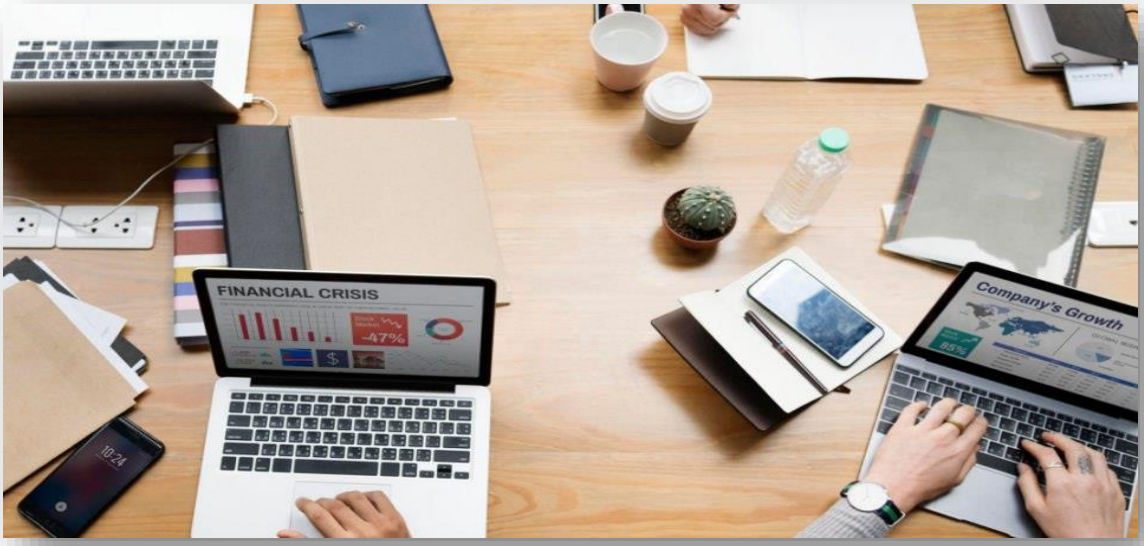
Orange Digital Cente

TECHIE KIT

University System



Team member

NAME
• **AYA OMAR** • **NADA HAMADA**

Supervisor: Abdel Sattar Ahmed

Project team :

Name	Role
• Nada Hamada	• Responsible for creating classes(Person, Singleton Manager, Student, Employee, Instructor ,Course) • Testing and Designing system.
• Aya Omar	• Responsible for creating classes(University, Faculty, Department) • Testing system

1.introduction

Because of the increase number of students it makes harder for university to track the increase number of students and their information so that the software systems are designed to support the diverse needs of a university and its stakeholders, including students, faculty, staff, administrators, and external partners. They encompass a wide range of functionalities and modules that interact with each other to facilitate efficient and effective management of university operations.

2. Abstract:

the software university system provides help to the employee to track stakeholders who work in the university and their information like how many people work here and their salary ,name ,age and id and their role and also help to track the students information like where faculty they go and their courses and their department and save all that in system so that it can’t be deleted

Table content :

Person Class

```
#include <iostream>
#include<vector>
#include<string>
#include <fstream>

using namespace std;
class Singleton;
class Person
{
public:
    string name;
    int age;
    int id;
    Person(string n,int a,int ID)
    {
        name=n;
        age=a;
        id=ID;
    }
    virtual void role()=0;
};
//////////////////////////////////////
```

The "Person" class acts as an abstract base class, providing a common structure and functionality for person objects. It allows derived classes to inherit its attributes and behaviors while also requiring them to implement their own version of the role() function (virtual) to define the specific role of the person in their respective contexts.

Manager Singleton

```
////////////////////////////////////
class Singleton:public Person
{
    int salary;
    static Singleton* instance;
public:
    Singleton(string n,int a,int ID):Person(n, a, ID)
    {
        //salary=s;
    }
    static Singleton* getInstance(string n, int a, int ID)
    {
        if(instance==nullptr)
        {
            instance = new Singleton(n, a, ID);;
        }
        return instance;
    }
    void setSalary(int s)
    {
        salary=s;
    }
    int getSalary()
    {
        return salary;
    }
    void role()
    {
        cout<<"This is Manager person"<<endl;
    }
};
Singleton* Singleton::instance=nullptr;
////////////////////////////////
```

The Singleton class is designed to ensure that only one instance of the class can exist at any given time. The getInstance method is used to retrieve the single instance, and the setSalary and getSalary methods can be used to modify and access the salary property of the singleton object.

Employee class

```
class Employee: public Person
{
public:
    Employee(string n,int a,int ID):Person(n,a,ID)
    {
    }
    int salary;
    void role()
    {
        cout<<"I am Employee"<<endl;
    }
```

The Employee class in the provided code is a derived class of the Person class. It represents an employee and inherits the properties and behaviors of a person. The Employee class extends the functionality of the Person class by adding a salary property specific to an employee. The role() method is overridden to provide an employee-specific implementation.

Student Class

```
//////////////////////////////////
class Student:public Person
{
public:
    float GPA;
    Student(string n,int a,int ID,float gpa):Person( n, a, ID)
    {
        GPA=gpa;
    }
    void role()
    {
        cout<<"I am a student "<<endl;
    }

};
//////////////////////////////////////////////
```

The Student class extends the functionality of the Person class by adding a GPA property specific to a student. The role() method is overridden to provide a student-specific implementation.

Instructor class

```
////////////////////////////////////////////////
class Instructor:public Person
{
public:
    Instructor(string n,int a,int ID):Person(n,a,ID)
    {
    }
    int salary;
    int hours;
    void role()
    {
        cout<<"I am Instructor"<<endl;
    }
};
////////////////////////////////////////////////
```

The Instructor class in the provided code is a derived class of the Person class. It represents an instructor and inherits the properties and behaviors of a person.

The Instructor class extends the functionality of the Person class by adding salary and hours properties specific to an instructor. The role() method is overridden to provide an instructor-specific implementation.

The university class

```
class University
{
public:

    string title;
    vector<Faculty>facultys;
    Singleton *manager;
    University(Faculty f)
    {
        facultys.push_back(f);

    }


    void addFaculty( Faculty s)
    {

        ofstream output;
        output.open("university.txt",ios_base::app);

        if (!output)
        {
            cout<< "Error" << endl;
        }

        output<<s.title <<endl;
        output.close();
    }
    void detail()
    {

        cout<<"this university "<<title;
    }
```

the "University" class represents an abstraction for a university entity. It allows for the addition of faculties, stores them in a vector, and provides a method to display the university's title. The `addFaculty()` method takes an instance . and writes the title of the faculty to a file named "university.txt"

Course class

```
class Course
{
  public:
      string title;
      int icourse_id;
       void details()
    {

        cout<<" this is course "<<endl;
    }

};
```

This class represents a course offered at a university/college.

It contains information like the course title, id, description etc.

Department Class

```
//////////////////////////////////////////////////
class Department{
    public:
    string title;
    vector<Course>courses;
      void addCourse( Course s)
    {

        ofstream output;
        output.open("university.txt",ios_base::app);

        if (!output)
        {
            cout<< "Error" << endl;
        }

        output <<  s.title<<" Course"<<endl;
        output.close();
    }
    void detail()
    {

        cout<<" this is department"<<endl;
    }

};
```

the "Department" class represents an abstraction for a department within a university. It allows for the addition of courses and stores them in a vector. It also provides a method to display a generic message indicating that it is a department. The addCourse(Course s) method takes an instance of the "Course" class. It writes the title of the course followed by "Course" to a file named "university.txt". The file is opened in append mode.

Faculty Class

```
class Faculty
{
public:

    string title;
    Faculty()
    {

    }
    vector<Student>students;
     vector<Instructor>instructors;
      vector<Employee>employees;
      vector<Department>departments ;
      void detail()
      {

          cout<<"this is faculty"<<endl;
      }
      void addStudent(Student s)
      {

          ofstream output;
          output.open("university.txt",ios_base::app);

          if (!output)
          {
              cout<< "Error" << endl;
          }

          output <<  s.name<<"   Student "<< endl;
          output.close();
```

```
void addEmployee( Employee s)
{

    ofstream output;
    output.open("university.txt",ios_base::app);

    if (!output)
    {
        cout<< "Error" << endl;
    }

    output <<  s.name<<"   Instructor "<< endl;
    output.close();
}


};
```

```
void addInstractor( Instructor s)
{

    ofstream output;
    output.open("university.txt",ios_base::app);

    if (!output)
    {
        cout<< "Error" << endl;
    }

    output <<  s.name<<"   Instructor "<< endl;
    output.close();
}
void addDepartment( Department s)
{

    ofstream output;
    output.open("university.txt",ios_base::app);

    if (!output)
    {
        cout<< "Error" << endl;
    }

    output <<  s.title<<endl;
    output.close();
}
```

the "Faculty" class represents an abstraction for a faculty within a university. It allows for the addition of students, instructors, employees, and departments associated with the faculty. It also provides a method to display a generic message indicating that it is a faculty.

It has `addEmployee(Employee s)`:method takes an instance of the "Employee" class as a parameter. It also writes the name of the employee followed by "Instructor" to the "university.txt"

`It also has addDepartment(Department s)` method takes an instance of the "Department" class as a parameter and it writes the title of the department to the "university.txt" file.

And `addInstructor(Instructor s)`method takes an instance of the "Instructor" class as a parameter and it writes the name of the instructor followed by "Instructor" to the "university.txt" file.

`And addStudent(Student s)` method takes an instance of the "Student" class as a parameter and It also writes the name of the student followed by "Student" to a file named "university.txt". The file is opened in append mode.

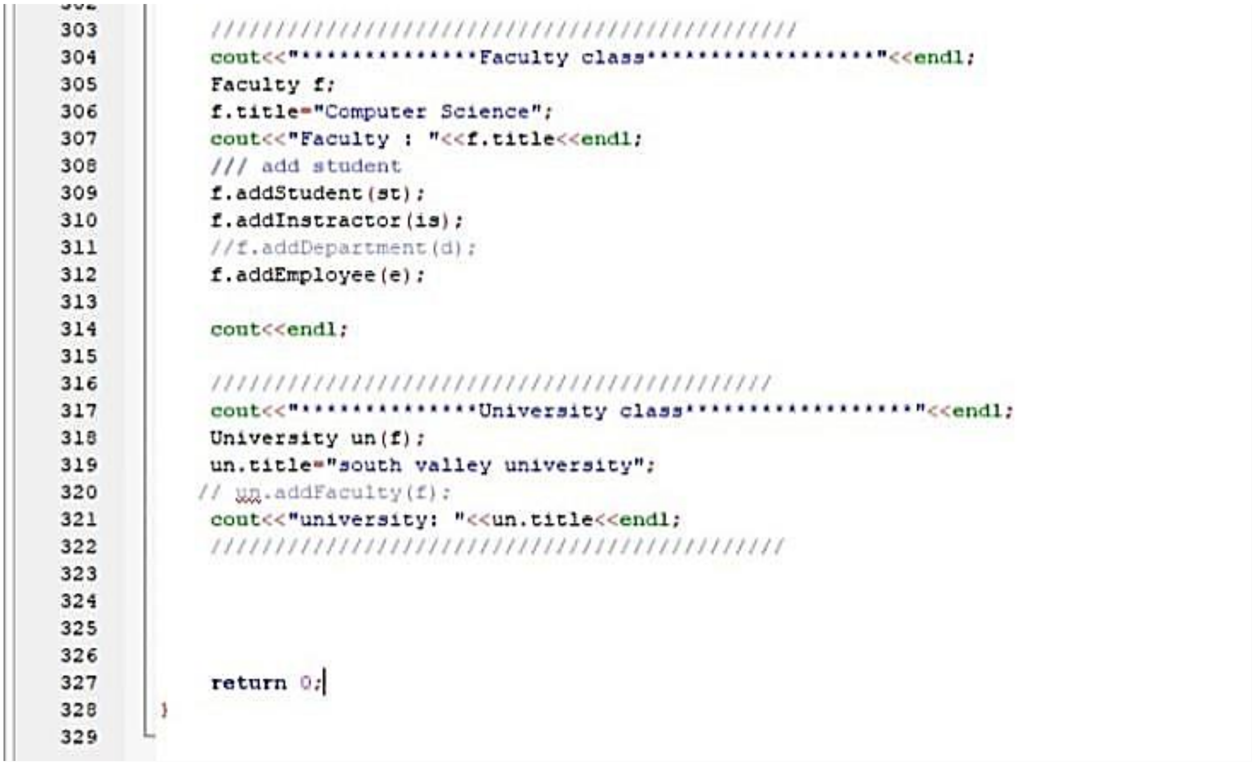
Main prototype:

```
 ///////////////////////////////////////////////
  cout<<"**************course class*****************"<<endl;
Course c;
c.title="Data Structure";
cout<<"Course: "<<c.title<<endl;
cout<<endl;
////////////////////////////////////////////////////////
 cout<<"**************Department class*****************"<<endl;

Department d;
d.title="CS";
cout<<"Department : "<<d.title<<endl;
// d.addCourse(c);
cout<<endl;


////////////////////////////////////////////////
cout<<"**************Faculty class*****************"<<endl;
Faculty f;
f.title="Computer Science";
cout<<"Faculty : "<<f.title<<endl;
/// add student
f.addStudent(st);
f.addInstractor(is);
//f.addDepartment(d);
f.addEmployee(e);

cout<<endl;
```



```
    ////////////////////////////////////////////////
    cout<<"**************Faculty class*****************"<<endl;
    Faculty f;
    f.title="Computer Science";
    cout<<"Faculty : "<<f.title<<endl;
    /// add student
    f.addStudent(st);
    f.addInstractor(is);
    //f.addDepartment(d);
    f.addEmployee(e);

    cout<<endl;

    //////////////////////////////////////////////
    cout<<"**************University class*****************"<<endl;
    University un(f);
    un.title="south valley university";
   // un.addFaculty(f);
    cout<<"university: "<<un.title<<endl;
    //////////////////////////////////////////////




    return 0;
}
```

```
int main()
{
    cout<<"****************class Manager**************** "<<endl;
    Singleton* s1=Singleton::getInstance("aye",22,201977);
    cout<<"Nmae Manager: "<<s1->name<<" Age: "<<s1->age<<" ID : "<<s1->id<<endl;
    s1->setSalary(2000);
    cout<<"Salary: "<<s1->getSalary()<<endl;
    cout<<endl;

    ////////////////////////////////////////////
    cout<<"********class student*********************"<<endl;
    Student st("nada",19,202144,3.4);
    cout<<"Name: "<<st.name<<" ID: "<<st.id<<" Age: "<<st.age<<" GPA: "<<st.GPA<<endl;
    cout<<endl;

    ////////////////////////////////////////////////
    cout<<"*********Employee class******************"<<endl;
    Employee e("ali",10,272782);
    e.salary=400;
    cout<<"Name: "<<e.name<<" ID: "<<e.id<<" Age: "<<e.age<<" Salary: "<<e.salary<<endl;
    cout<<endl;

    ////////////////////////////////////////////////
    cout<<"**************Instructor class******************"<<endl;
    Instructor is("noor",18,353873);
    is.salary=4000;
    is.hours=8;
    cout<<"Name: "<<is.name<<" ID: "<<is.id<<" Age: "<<is.age<<" Salary: "<<is.salary<<" Hors: "<<is.hours<<endl;
    cout<<endl;
    ////////////////////////////////////////////////////////////////////////////
```

, the code demonstrates the creation and usage of various classes like `Singleton`, `Student`, `Employee`, `Instructor`, `Course`, `Department`, `Faculty`, and `University`. It showcases the initialization of objects, accessing their attributes, and calling methods on the instances of these classes.