|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C:\Users\Admin\Desktop\download.jpg | USMAN INSTITUTE OF TECHNOLOGY | | | | | |  |
|  |  |  |  |  |  |  |  |
|  | Department of Computer Science  CS121 Object Oriented Programming | | | | | |  |
|  |  | Lab # 10  Abstract Classes | | | |  |  |
|  | Objective:  This experiment introduces the students to the concept of Abstract Classes and their use in Inheritance | | | | | |  |
|  | **Name of Student:**  **Roll No: Sec.**  **Date of Experiment:** | | | | | |  |
|  | **Marks Obtained/Remarks:**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Signature:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | |  |

**Lab 10: Abstract Classes**

An abstract class can be considered as a blueprint for other classes. It allows you to create a set of methods that must be created within any child classes built from the abstract class. A class which contains one or more abstract methods is called an abstract class. An abstract method is a method that has a declaration but does not have an implementation. While we are designing large functional Units we use an abstract class. When we want to provide a common interface for different implementations of a component, we use an abstract class.

In object-oriented programming, an abstract class is a class that cannot be instantiated. However, other classes can be created that inherit from an abstract class. Generally, an abstract class is used to create a blueprint for other classes.

**Abstract Methods**

Abstract classes contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation. Since abstract classes cannot be instantiated, subclasses are required to provide implementations for the abstract methods. It is important to note that a class that is derived from an abstract class cannot be instantiated unless all of its abstract methods are overridden.

**Python *abc* Module**

Python on its own doesn't provide abstract classes. However, it does come with a module that provides the base for defining Abstract Base classes(ABC). In order to define an abstract class, abc (abstract base class) module is used. abc module provides the infrastructure for defining abstract base classes in Python.

*Example 1*

* from abc import ABC, abstractmethod
* class Polygon(ABC):
* @abstractmethod
* def noofsides(self):
* pass
* class Triangle(Polygon):
* # overriding abstract method
* def noofsides(self):
* print("I have 3 sides")
* class Pentagon(Polygon):
* # overriding abstract method
* def noofsides(self):
* print("I have 5 sides")
* class Hexagon(Polygon):
* # overriding abstract method
* def noofsides(self):
* print("I have 6 sides")
* class Quadrilateral(Polygon):
* # overriding abstract method
* def noofsides(self):
* print("I have 4 sides")

# Student Exercise

Following UML diagram represent an Employee class and Salaried Employee and Hourly Employee classes being inherited from it

|  |
| --- |
| ***Employee*** |
| - name: String  - cnic: String |
| Employee(name: String, cnic: String)  *monthlyEarnings(): float*  getDetails(): String |

|  |
| --- |
| **SalariedEmployee** |
| - weeklySalary |
| SalariedEmployee(name: String, cnic: String, weeklySalary: float)  MonthlyEarnings(): float  getDetails(): String |

|  |
| --- |
| **HourlyEmployee** |
| - wage: float  - hours: float |
| HourlyEmployee(name: String, cnic: String, wage: float, hours: float)  MonthlyEarnings(): float  getDetails(): String |

Function details are given in the following table

|  |  |  |
| --- | --- | --- |
|  | MonthlyEarnings() | getDetails() |
| Employee | Abstract Method | Return name and cnic |
| Salaried Employee | Four times the weekly salary | Return name, cnic, and monthly earnings |
| Hourly Employee | Product of wages and hours.  For every extra hour above 40 hours, 1.5 times the wage | Return name, cnic, and monthly earnings |

Exercise 1

Write Python code for implementing the classes

**Code:**

from abc import ABC, abstractmethod

class Employee(ABC):

    def \_\_init\_\_(self, name, cnic):

        self.\_\_name = name

        self.\_\_cnic = cnic

    @abstractmethod

    def monthlySalary(self):

        pass

    def getDetails(self):

        return "Name: {}, CNIC: {}, Monthly Salary: {}".format(self.\_\_name, self.\_\_cnic, self.monthlySalary())

class SalariedEmployee(Employee):

    def \_\_init\_\_(self, name,cnic, weeklySalary):

        super().\_\_init\_\_(name, cnic)

        self.\_\_weeklySalary = weeklySalary

    def monthlySalary(self):

        return self.\_\_weeklySalary \* 4

    def getDetails(self):

        return super().getDetails()

class HourlyEmployee(Employee):

    def \_\_init\_\_(self, name,cnic, wage,hours):

        super().\_\_init\_\_(name, cnic)

        self.\_\_wage = wage

        self.\_\_hours = hours

    def monthlySalary(self):

        return self.\_\_wage \* self.\_\_hours \* 4

    def getDetails(self):

        return super().getDetails()

Exercise 2

Implement a program to create three employees

Given the user input, initialize each of them as either a Salaried Employee or an Hourly Employee

Display details and Monthly Earnings of all the employee

**Code:**

print("======== Salaried Employee =========")

#SalariedEmployee

name=input("Enter Name: ")

cnic=input("Enter CNIC: ")

weeklySalary=input("Enter weekly salary: ")

weeklySalary = float(weeklySalary)

obj = SalariedEmployee(name, cnic, weeklySalary)

print('\n',obj.getDetails())

print('\n')

print("======== Hourly Employee =========")

#HourlyEmployee

name=input("Enter Name: ")

cnic=input("Enter CNIC: ")

wage=input("Enter wage: ")

hours=input("Enter hours: ")

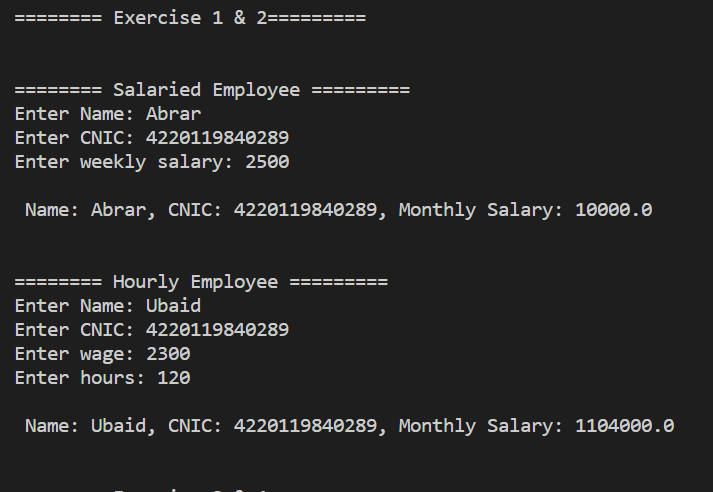
wage = float(wage)

hours = float(hours)

obj = HourlyEmployee(name, cnic, wage, hours)

print('\n',obj.getDetails())

**Output:**



Following UML Class diagram represent classes for an Undergraduate and a Graduate student both being inherited from the Student class. For an undergraduate student, the grade would be ‘Pass’ if the test score is greater than or equals to 60, and ‘Fail’ otherwise. However, for a graduate student the grade would be ‘Pass’ if the test score is greater than or equal to 70, and ‘Fail’ otherwise

|  |
| --- |
| ***Employee*** |
| - name: String  - cnic: String  - basicSalary: float |
| Employee (String, String, float)  //accessors  setBasicSalary(float): None  *monthlyEarnings(): float* |

|  |
| --- |
| **Manager** |
| - department: String  - houseAllowance: float |
| Manager (String, String, float, String, float)  //accessors  //mutators  MonthlyEarnings(): float |

|  |
| --- |
| **Developer** |
| - technology: String |
| Developer(String, String, float, String)  //accessor  //mutator  MonthlyEarnings(): float |

|  |
| --- |
| **CEO** |
| - medicalAllowance: float |
| CEO (String, String, float, String, float, float)  //accessor  //mutator  MonthlyEarnings(): float |

Manager’s monthly earning is equal to the basic salary plus house allowance

CEO’s monthly earning is equal to the basic salary plus house allowance and medical allowance

Developer’s monthly earning is equal to the basic salary

Exercise 3

Write Python code for defining the classes

**Code:**

class Employee(ABC):

    def \_\_init\_\_(self, name, cnic, basicSalary):

        self.\_\_name = name

        self.\_\_cnic = cnic

        self.\_\_basicSalary = basicSalary

    def getName(self):

        return self.\_\_name

    def getCnic(self):

        return self.\_\_cnic

    def getBasicSalary(self):

        return self.\_\_basicSalary

    def setBasicSalary(self, basicSalary):

        self.\_\_basicSalary = basicSalary

    @abstractmethod

    def monthlyEarnings(self):

        pass

class Manager(Employee):

    def \_\_init\_\_(self, name, cnic, basicSalary, department, houseAllowance):

        super().\_\_init\_\_(name, cnic, basicSalary)

        self.\_\_department = department

        self.\_\_houseAllowance = houseAllowance

    #accessors

    def getDepartment(self):

        return self.\_\_department

    def getHouseAllowance(self):

        return self.\_\_houseAllowance

    #mutators

    def setDepartment(self, department):

        self.\_\_department = department

    def setHouseAllowance(self, houseAllowance):

        self.\_\_houseAllowance = houseAllowance

    def monthlyEarnings(self):

        return self.getBasicSalary() + self.getHouseAllowance()

class CEO(Employee):

    def \_\_init\_\_(self, name, cnic, basicSalary, department, houseAllowance, medicalAllowance):

        super().\_\_init\_\_(name, cnic, basicSalary)

        self.\_\_department = department

        self.\_\_houseAllowance = houseAllowance

        self.\_\_medicalAllowance = medicalAllowance

    #accessors

    def getDepartment(self):

        return self.\_\_department

    def getHouseAllowance(self):

        return self.\_\_houseAllowance

    def getMedicalAllowance(self):

        return self.\_\_medicalAllowance

    #mutators

    def setDepartment(self, department):

        self.\_\_department = department

    def setHouseAllowance(self, houseAllowance):

        self.\_\_houseAllowance = houseAllowance

    def setMedicalAllowance(self, medicalAllowance):

        self.\_\_medicalAllowance = medicalAllowance

    def monthlyEarnings(self):

        return self.getBasicSalary() + self.getHouseAllowance() + self.getMedicalAllowance()

class Developer(Employee):

    def \_\_init\_\_(self, name, cnic, basicSalary, technology):

        super().\_\_init\_\_(name, cnic, basicSalary)

        self.\_\_technology = technology

    #accessors

    def getTechnology(self):

        return self.\_\_technology

    #mutators

    def setTechnology(self, technology):

        self.\_\_technology = technology

    def monthlyEarnings(self):

        return self.getBasicSalary()

Exercise 4

Implement a program to create three employees

Given the user input, initialize each of them as either a Manager, a CEO, or a Developer

Display Monthly Earnings for all the employees

Revise particulars of employees as follows

* Increase Basic Salary by 10%
* Increase House Allowance by 7%
* Increase Medical Allowance by 5%

Display Monthly Earnings of all the employees after the revision

**Code:**

print("======== Manager =========")

name=input("Enter Name: ")

cnic=input("Enter CNIC: ")

basicSalary=input("Enter Basic  Salary: ")

department=input("Enter Basic  Department: ")

houseAllowance=input("Enter houseAllowance: ")

basicSalary = float(basicSalary)

houseAllowance = float(houseAllowance)

obj = Manager(name, cnic, basicSalary, department, houseAllowance)

print('Name: ',obj.getName())

print('Cnic: ',obj.getCnic())

print('Basic Salary: ',obj.getBasicSalary())

print('Department: ',obj.getDepartment())

print('House Allowance: ',obj.getHouseAllowance())

print('Monthly Earnings: ',obj.monthlyEarnings())

obj.setBasicSalary(float(obj.getBasicSalary()) \* 1.1)

obj.setHouseAllowance(float(obj.getHouseAllowance()) \* 1.07)

print(obj.monthlyEarnings())

print('\n')

print("======== CEO =========")

name=input("Enter Name: ")

cnic=input("Enter CNIC: ")

basicSalary=input("Enter Basic  Salary: ")

department=input("Enter Basic  Department: ")

houseAllowance=input("Enter house Allowance: ")

medicalAllowance=input("Enter Medical Allowance: ")

obj1 = CEO(name, cnic, basicSalary, department, houseAllowance, medicalAllowance)

print('Name: ',obj1.getName())

print('CNIC: ',obj1.getCnic())

print('Basic Salary: ',obj1.getBasicSalary())

print('Department: ',obj1.getDepartment())

print('House Allowance: ',obj1.getHouseAllowance())

print('Medical Allowance: ',obj1.getMedicalAllowance())

print('Monthly Earnings: ',obj1.monthlyEarnings())

obj1.setBasicSalary(float(obj1.getBasicSalary()) \* 1.1)

obj1.setHouseAllowance(float(obj1.getHouseAllowance()) \* 1.07)

obj1.setMedicalAllowance(float(obj1.getMedicalAllowance()) \* 1.05)

print(obj1.monthlyEarnings())

print('\n')

print("======== Developer =========")

name=input("Enter Name: ")

cnic=input("Enter CNIC: ")

basicSalary=input("Enter Basic  Salary: ")

technology=input("Enter Technology: ")

obj = Developer(name, cnic, basicSalary, technology)

print('Name: ',obj.getName())

print('CNIC: ',obj.getCnic())

print('Basic Salary: ',obj.getBasicSalary())

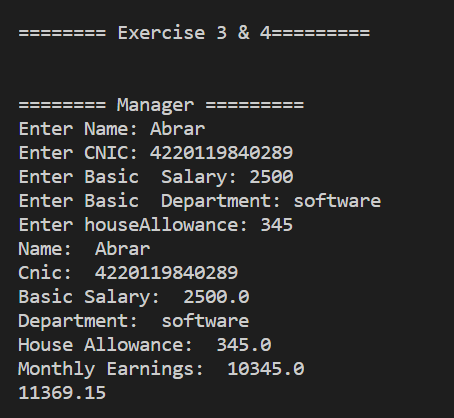
print('Technology: ',obj.getTechnology())

print('Earnings: ',obj.monthlyEarnings())

obj.setBasicSalary(float(obj.getBasicSalary()) \* 1.1)

print(obj.monthlyEarnings())

**Output:**



Text

Description automatically generated