Tarea semana 6: Programación Orientada a Objetos con PythonJorge Peñaloza

Programación Avanzada II

Instituto IACC

27 de septiembre de 2021

**Desarrollo**

**Respuestas del problema**

***Contexto***

Lea atentamente la siguiente instrucción, analice la información de acuerdo a los contenidos revisados en la semana y conteste las preguntas que se presentan a continuación:

Imagine que usted ha sido seleccionado para diseñar lógicamente (puede usar elementos gráficos como formas de Word, tipo un mapa conceptual) un sistema para el registro de estudiantes de colegios privados de la Región Metropolitana de Santiago, este sistema debe manejar las personas dentro del proceso como son: estudiantes con todos sus atributos (nombre, apellido, rut, dirección) y apoderados también con sus atributos (nombre, apellido, Rut, dirección), que año y asignaturas cursan cada uno de los estudiantes, si realizan actividades extras, especificando cual actividad realizan.

Ahora bien, basándose en esa situación ponga en práctica sus conocimientos adquiridos de Python y ejemplifique lo siguiente:

* La abstracción de clases, Interfaz de objetos, clases anónimas y sobrecarga en la POO (3 puntos).
* Explique la aplicación de iteración de objetos, métodos mágicos y clonación de objetos para casos presentados (3 puntos).
* Diseñe la abstracción de clases, sobrecarga y métodos mágicos en los casos dados (3 puntos).

Recuerde que debe mostrar ejemplos de código en Python y justificar su elaboración.

***Respuesta a pregunta 1***

La clase *PersonaA* es una clase abstracta que tiene el método obligatorio rutEsValido(). Se encuentra definido en la figura 1 y se encuentra implementado según código de programación 1.

Figura 1

*Diagrama UML de clase PersonaA*

Tabla

Descripción generada automáticamente

Fuente: elaboración propia

Código de programación 1

class PersonaA(metaclass=ABCMeta):

    @abstractmethod

    def rutEsValido(self):

        pass

Fuente: Elaboración propia

La interfaz *Validar* es una interfaz que tiene el método implementado esRut() y se encuentra definido en la figura 2 y se encuentra implementado según código de programación 2. Además, se debe mencionar que en esta interfaz se implemento la “sobrecarga de método” para comprobación de esta.

Figura 2

*Diagrama UML de interfaz Validar*

Imagen que contiene Tabla

Descripción generada automáticamente

Fuente: elaboración propia

Código de programación 2

class Validar():

    def esRut(rut):

        pass

    def esRut(rut):

        rut = rut.upper();

        rut = rut.replace("-","")

        rut = rut.replace(".","")

        aux = rut[:-1]

        dv = rut[-1:]

        revertido = map(int, reversed(str(aux)))

        factors = cycle(range(2,8))

        s = sum(d \* f for d, f in zip(revertido,factors))

        res = (-s)%11

        if str(res) == dv:

            return True

        elif dv=="K" and res==10:

            return True

        else:

            return False

Fuente: Elaboración propia

La clase *Persona* es la implementación que tiene el método obligatorio rutEsValido() además de los atributos requeridos. Se encuentra definido en la figura 3 y se encuentra implementado según código de programación 3. La clase persona hereda métodos de la clase abstracta *PersonaA* y la interfaz *Validar.*

Figura 3

*Diagrama UML de clase Persona*

Tabla

Descripción generada automáticamente con confianza media

Fuente: elaboración propia

Código de programación 3

class Persona(PersonaA, Validar):

    def \_\_init\_\_(self, rut, nombre, apellido, direccion):

        self.rut = rut

        self.nombre = nombre

        self.apellido = apellido

        self.direccion = direccion

    def rutEsValido(self):

        return Validar.esRut(self.rut)

Fuente: Elaboración propia

La clase *Apoderado* se encuentra definido en la figura 4 y se encuentra implementado según código de programación 4. La clase persona hereda métodos de la clase *Persona.*

Figura 4

*Diagrama UML de clase Apoderado*

Tabla

Descripción generada automáticamente

Fuente: elaboración propia

Código de programación 4

class Apoderado(Persona):

    def \_\_init\_\_(self, rut, nombre, apellido, direccion):

        self.estudiantes = []

        super().\_\_init\_\_(rut, nombre, apellido, direccion)

    def addEstudiantes(self, rut):

        self.estudiantes.append(rut)

    def delEstudiantes(self, rut):

        self.estudiantes.remove(rut)

    def getEstudiantes(self):

        return self.estudiantes

Fuente: Elaboración propia

La clase *Estudiante* se encuentra definido en la figura 5 y se encuentra implementado según código de programación 5. La clase persona hereda métodos de la clase *Persona.*

Figura 5

*Diagrama UML de clase Estudiante*

Tabla

Descripción generada automáticamente

Fuente: elaboración propia

Código de programación 5

class Estudiante(Persona):

    def \_\_init\_\_(self, \*arg):

        if len(arg) == 5:

            self.anio = arg[-1]

        super().\_\_init\_\_(\*arg[0:-1])

Fuente: Elaboración propia

La clase *Asignatura* se encuentra definido en la figura 6 y se encuentra implementado según código de programación 6.

Figura 6

*Diagrama UML de clase Asignatura*

Tabla

Descripción generada automáticamente

Fuente: elaboración propia

Código de programación 6

class Asignatura:

    def \_\_init\_\_(self, codigo, descripcion, planEstudios) :

        self.codigo = codigo

        self.descripcion = descripcion

        self.planEstudios = planEstudios

        self.estudiantes = []

    def addEstudiantes(self, rut):

        self.estudiantes.append(rut)

    def delEstudiantes(self, rut):

        self.estudiantes.remove(rut)

    def getEstudiantes(self):

        return self.estudiantes

Fuente: Elaboración propia

La clase *Actividad* se encuentra definido en la figura 7 y se encuentra implementado según código de programación 7.

Figura 7

*Diagrama UML de clase Actividad*

Tabla

Descripción generada automáticamente

Fuente: elaboración propia

Código de programación 7

class Actividad:

    def \_\_init\_\_(self, codigo, descripcion) :

        self.codigo = codigo

        self.descripcion = descripcion

        self.estudiantes = []

    def addEstudiantes(self, rut):

        self.estudiantes.append(rut)

    def delEstudiantes(self, rut):

        self.estudiantes.remove(rut)

    def getEstudiantes(self):

        return self.estudiantes

Fuente: Elaboración propia

Ya definidas y mostradas las implementaciones de cada clase e interfaz, se puede observar la relación de todas las clases mediante el diagrama UML de la figura 8, además, en código de programación 8 se puede apreciar la implementación completa más su ejecución en la figura 9.

Figura 8

*Diagrama UML*

Diagrama

Descripción generada automáticamente

Fuente: elaboración propia

En código de programación 8 se puede observar que se utilizó lo siguiente:

* Definición de e implementación de clases, clases abstractas, interfaz y sobrecarga de método
* Implementación de iteración de objetos, métodos mágicos y clonación de objetos.

Código de programación 8

*Archivo: proyecto.py*

from abc import ABCMeta, abstractmethod

from itertools import cycle

class Asignatura:

    def \_\_init\_\_(self, codigo, descripcion, planEstudios) :

        self.codigo = codigo

        self.descripcion = descripcion

        self.planEstudios = planEstudios

        self.estudiantes = []

    def addEstudiantes(self, rut):

        self.estudiantes.append(rut)

    def delEstudiantes(self, rut):

        self.estudiantes.remove(rut)

    def getEstudiantes(self):

        return self.estudiantes

class Actividad:

    def \_\_init\_\_(self, codigo, descripcion) :

        self.codigo = codigo

        self.descripcion = descripcion

        self.estudiantes = []

    def addEstudiantes(self, rut):

        self.estudiantes.append(rut)

    def delEstudiantes(self, rut):

        self.estudiantes.remove(rut)

    def getEstudiantes(self):

        return self.estudiantes

class PersonaA(metaclass=ABCMeta):

    @abstractmethod

    def rutEsValido(self):

        pass

class Validar():

    def esRut(rut):

        pass

    def esRut(rut):

        rut = rut.upper();

        rut = rut.replace("-","")

        rut = rut.replace(".","")

        aux = rut[:-1]

        dv = rut[-1:]

        revertido = map(int, reversed(str(aux)))

        factors = cycle(range(2,8))

        s = sum(d \* f for d, f in zip(revertido,factors))

        res = (-s)%11

        if str(res) == dv:

            return True

        elif dv=="K" and res==10:

            return True

        else:

            return False

class Persona(PersonaA, Validar):

    def \_\_init\_\_(self, rut, nombre, apellido, direccion):

        self.rut = rut

        self.nombre = nombre

        self.apellido = apellido

        self.direccion = direccion

    def rutEsValido(self):

        return Validar.esRut(self.rut)

class Estudiante(Persona):

    def \_\_init\_\_(self, \*arg):

        if len(arg) == 5:

            self.anio = arg[-1]

        super().\_\_init\_\_(\*arg[0:-1])

class Apoderado(Persona):

    def \_\_init\_\_(self, rut, nombre, apellido, direccion):

        self.estudiantes = []

        super().\_\_init\_\_(rut, nombre, apellido, direccion)

    def addEstudiantes(self, rut):

        self.estudiantes.append(rut)

    def delEstudiantes(self, rut):

        self.estudiantes.remove(rut)

    def getEstudiantes(self):

        return self.estudiantes

estudiante1 = Estudiante("13834450-1","Andres","Santillana","Santa Isabel 10852",2019)

estudiante2 = Estudiante("13834570-k","Jorge","Peñaloza","Maria Elena 334",2020)

estudiante3 = Estudiante("20552211-1","Valentina","Peñaloza","Las Magnolias 1122",2021)

apoderado1 = Apoderado("6225225-8","Segundo","Peñaloza","Los Copihues 2234")

apoderado1.addEstudiantes(estudiante1.rut)

apoderado1.addEstudiantes(estudiante2.rut)

apoderado1.addEstudiantes(estudiante3.rut)

apoderado2 = Apoderado("8258963-1","Ruben","Santillana","Los Magnus 34")

apoderado2.addEstudiantes(estudiante1.rut)

print(apoderado1.rut, apoderado1.nombre, apoderado1.apellido, apoderado1.direccion, apoderado1.rutEsValido())

print("Hay un estudiante que no corresponde")

for rutEstudiante in apoderado1.getEstudiantes():

    print(apoderado1.nombre, "es apoderado de", rutEstudiante)

apoderado1.delEstudiantes(estudiante1.rut)

print("Ahora si estan los 2 estudiantes que corresponden")

for rutEstudiante in apoderado1.getEstudiantes():

    print(apoderado1.nombre, "es apoderado de", rutEstudiante)

print("El otro apoderado es")

print(apoderado2.rut, apoderado2.nombre, apoderado2.apellido, apoderado2.direccion, apoderado2.rutEsValido())

for rutEstudiante in apoderado2.getEstudiantes():

    print(apoderado2.nombre, "es apoderado de", rutEstudiante)

print("Los estudiantes son")

print(estudiante1.rut, estudiante1.nombre, estudiante1.apellido, estudiante1.direccion, estudiante1.anio, estudiante1.rutEsValido())

print(estudiante2.rut, estudiante2.nombre, estudiante2.apellido, estudiante2.direccion, estudiante2.anio, estudiante2.rutEsValido())

print(estudiante3.rut, estudiante3.nombre, estudiante3.apellido, estudiante3.direccion, estudiante3.anio, estudiante3.rutEsValido())

clon = estudiante1

print("el clon es")

clon.nombre = "Andrea"

print(estudiante1.rut, estudiante1.nombre, estudiante1.apellido, estudiante1.direccion, estudiante1.anio, estudiante1.rutEsValido())

ingles = Asignatura(101,"Ingles Basico", "Verbos")

ingles.addEstudiantes(estudiante1.rut)

print("Asignatura",ingles.codigo, ingles.descripcion, "Alumno",ingles.getEstudiantes()[0])

natacion = Actividad(522,"Natacion en estadio municipal")

natacion.addEstudiantes(estudiante2.rut)

natacion.addEstudiantes(estudiante3.rut)

print("Asignatura",natacion.codigo, natacion.descripcion, "Alumno",natacion.getEstudiantes()[0])

print("Asignatura",natacion.codigo, natacion.descripcion, "Alumno",natacion.getEstudiantes()[1])

Fuente: Elaboración propia

Figura 9

*Vista de ejecución de Código de programación 8*

Texto

Descripción generada automáticamente

Fuente: elaboración propia

Bibliografía

IACC. (2020). Programación Orientada a Objetos con Python. En *Programación avanzada 2. Semana 6.*

IACC. (2021). Tarea semana 6.