

For Nation's Greater

In Partial Fulfillment of the Requirements for the

CS 223 - Object-Oriented Programming

Structured Family Hierarchy

Presented to:

Dr. CAGAS, UNIFE ODVINA

Professor

Prepared by: **GUDITO, FELIPEJR. H.** Student

BSCS: Computer Science May 3, 2024



Project Description:

This program is a structured family hierarchy that aims to represent a family tree by organizing the information of family members and displaying their roles in a console-based interaction. The program serves as a valuable tool for comprehending and managing family roles, making it easier to understand the relationships and responsibilities within a family.

Objectives:

- To implement a family member hierarchy using the four OOP principles such as Inheritance, Polymorphism, Encapsulation and Abstraction.
- To define subclasses for father, mother, and children with unique roles.
- To initialize instances with name, age, and gender attributes for each family member.
- To display family information and roles for each member through method calls.

Importance and Contribution:

The code is used as an educational resource for users, particularly students studying object-oriented programming concepts. By studying and interacting with the code, users can obtain a hands-on understanding of how inheritance, polymorphism, encapsulation, and abstraction function in reality. It also provides a basis for documenting family relationships and tracing ancestral lines, allowing users to build family trees and track historical data, making family history and heritage analyze easier.

Hardware & Software Used:

Hardware:

Android Phone

Software:

• Pydroid 3

PRINCIPLES OF OOP BEING USED IN THE CODE:

Abstraction with Inheritance:

This line imports the ABC (Abstract Base Class) and abstract method from the abc module, which are used for defining abstract base classes and abstract methods. The class FamilyMember has a parameter of ABC which defines as abstract base class and inherits from ABC. The FamilyMember class that will serve as the parent class for all family members. It contains common attributes and methods that all family members share, such as name, age, and gender.

- 1 from abc import ABC, abstractmethod # Importing the ABC class and abstractmethod decorator from the abc module.
- 2
- 3 class FamilyMember(ABC): # Defining an abstract base class for family members.

Encapsulation:

This line defines the constructor method for the FamilyMember class which initializes the name, age, and gender attributes. Defining the constructor method to initialize the attributes name, age, and gender. This method sets the initial state of a FamilyMember object by assigning values to its attributes name, age, and gender. The attributes are prefixed with double underscores to make them private, encapsulatingthe data and restricting direct access from outside the class.



For Nation's Greater

```
9
      def get_name(self): # Method to get
    the name of the family member.
10
        return self.__name
11
12
      def get_age(self): # Method to get the
    age of the family member.
13
        return self.__age
14
      def get_gender(self): # Method to get
15
    the gender of the family member.
16
        return self.__gender
```

This line of code is defining a method to retrieve the name, age and gender of a FamilyMember. These three methods provide controlled access to the private attribute name, age and gender by returning its value when called. It encapsulates the data by providing a getter method to retrieve the name, age, and gender

```
9
      def get_name(self): # Method to get
    the name of the family member.
10
        return self.__name
11
      def get_age(self): # Method to get the
12
    age of the family member.
13
        return self.__age
14
15
      def get_gender(self): # Method to get
    the gender of the family member.
16
        return self.__gender
```

Abstraction:

The purpose is to declaring an abstract method Family_info and Role This line marks the Family_info and Role method as abstract, indicating that any subclass of FamilyMember must provide its own implementation of this method. The pass statement serves as a placeholder, indicating that the method hasnoimplementation in the abstract base class.

```
18
      @abstractmethod # Decorator
    indicating that the method is abstract and
    must be implemented by subclasses.
19
      def Family_info(self): # Abstract
    method to display family information.
20
        pass
22
      @abstractmethod # Decorator
    indicating that the method is abstract and
    must be implemented by subclasses.
23
      def Role(self): # Abstract method to
    display the role of the family member.
24
        pass
```



Inheritance:

This line defines a subclass Father that inherits from the FamilyMember class, the Father class inherits attributes and methods from the FamilyMember class. It specializes in the FamilyMember class to represent a father. Similarly, the classes Mother, FirstChild, SecondChild, and. ThirdChild follow the same. Pattern of inheritance from the FamilyMember class. The family_info method is used to print information about the father, such as his name and gender, and the role method is also used to print information that identifies the person as the father.

```
class Father(FamilyMember): # Subclass
26
    representing a father, inheriting from
    FamilyMember.
27
      def __init__(self, name, age, gender): #
    Constructor method for Father class.
         super().__init__(name, age, gender) #
    Calling the constructor of the superclass.
29
      def Family_info(self): # Method to
30
    display family information for a father.
31
         print("Name: " + self.get_name()) #
    Print the father's name.
         print("Gender: " + self.get_gender())
32
    # Print the father's gender.
33
34
      def Role(self): # Method to display the
    role of a father.
         print("The Father") # Print the role of
35
    a father.
```

Polymorphism:

The Family_info and Role method is a polymorphism, and the rest of the of the subclass has this method name for Family_info and Role. Similarly, the classes Mother, FirstChild, SecondChild, and ThirdChild follow the same pattern of inheritance from the FamilyMember class.

```
def Family_info(self): # Method to
                                                                    def Family_info(self): # Method to
        def Family_info(self): # Method to
                                                                                                                           display family information for the first
                                                                 display family information for a mother.
     display family information for a father
                                                                                                                          child.
                                                            42
                                                                      print("Name: " + self.get_name()) #
31
          print("Name: " + self.get_name()) #
                                                                                                                               print("Name: " + self.get_name()) #
                                                                 Print the mother's name.
                                                                                                                          Print the first child's name.
print("Gender: " + self.get_gender())
     Print the father's name.
                                                                      print("Gender: " + self.get_gender())
                                                            43
32
          print("Gender: " + self.get_gender())
                                                                 # Print the mother's gender.
                                                                                                                           # Print the first child's gender
     # Print the father's gender.
                                                            44
33
                                                                                                                      56
                                                                                                                            def Role(self): # Method to display the
                                                            45
                                                                   def Role(self): # Method to display the
        def Role(self): # Method to display the
34
                                                                                                                          role of the first child.
                                                                 role of a mother.
     role of a father
                                                                                                                      57
                                                                                                                               print("The First child") # Print the role
                                                                      print("The Mother") # Print the role of
                                                            46
          print("The Father") # Print the role of
                                                                                                                           of the first child
     a father.
                        63
                                def Family_info(self): # Method to
                                                                                            def Family_info(self): # Method to
                              display family information for the second
                                                                                          display family information for the third
                              child.
                                   print("Name: " + self.get_name()) #
                                                                                              print("Name: " + self.get_name()) #
                              Print the second child's name.
                                                                                         Print the third child's name.
print("Gender: " + self.get_gender())
                              print("Gender: " + self.get_gender())
# Print the second child's gender.
                        65
                                                                                          # Print the third child's gender
                                def Role(self): # Method to display the
                                                                                            def Role(self): # Method to display the
                              role of the second child.
                                                                                          role of the third child.
                                  print("The Second child") # Print the
                                                                                              print("The Third child") # Print the
                              role of the second child.
                                                                                         role of the third child.
```







"For Nation's Greater

In this line of code, we create an instance of the subclasses used as our basic information and identify the background to our sample data. To access the data, we call each method for each instance and display the output.

81	# Creating instances of the subclasses	97	print("======") #
82	Person1 = Father("Romeo", 65, "Male") #		Printing a separator.
	Creating an instance of Father class.	98	print() # Printing an empty line.
83	Person2 = Mother("Maria", 57, "Female")	99	print("=======") #
	# Creating an instance of Mother class.	(SSSE 18	Printing a separator.
84	Person3 = FirstChild("Sam", 35, "Male") #	100	Person3.Family_info() # Calling
	Creating an instance of FirstChild class.	000000000000000000000000000000000000000	Family_info method for Person3.
85	Person4 = ThirdChild("Christine", 19,	101	Person3.Role() # Calling Role method for
	"Female") # Creating an instance of		Person3.
	ThirdChild class.	102	print("======") #
86	Person5 = SecondChild("Joe", 23, "Male")		Printing a separator.
	# Creating an instance of SecondChild	103	print() # Printing an empty line.
	class.	104	print("======"") #
87			Printing a separator.
88	# Calling methods for each instance	105	Person4.Family_info() # Calling
89	print("=======") #	G2568 22	Family_info method for Person4.
	Printing a separator.	106	Person4.Role() # Calling Role method for
90	Person1.Family_info() # Calling		Person4.
	Family_info method for Person1.	107	print("======") #
91	Person1.Role() # Calling Role method for		Printing a separator.
	Person1.	108	print() # Printing an empty line.
92	print("======"" #	109	print("======"") #
2000	Printing a separator.		Printing a separator.
93	<pre>print() # Printing an empty line.</pre>	110	Person5.Family_info() # Calling
94	print("=======" #		Family_info method for Person5.
2000	Printing a separator.	111	Person5.Role() # Calling Role method for
95	Person2.Family_info() # Calling		Person5.
9509094	Family_info method for Person2.	112	print("======") #
96	Person2.Role() # Calling Role method for	110	Printing a separator.
	Person2.	-111	

Code:

1	from abc import ABC, abstractmethod #	48	class FirstChild(FamilyMember): #
	Importing the ABC class and	40	Subclass representing the first child,
	abstractmethod decorator from the abc	49	inheriting from FamilyMember.
	module.	49	definit(self, name, age, gender): #
2			Constructor method for FirstChild class.
3	class FamilyMember(ABC): # Defining an	50	super()init(name, age, gender) #
	abstract base class for family members.		Calling the constructor of the superclass.
4	definit(self, name, age, gender): #	51	
	Constructor method to initialize name, age,	52	def Family_info(self): # Method to
	and gender attributes.		display family information for the first
5	selfname = name # Private		child.
5	attribute for name.	53	print("Name: " + self.get_name()) #
-			Print the first child's name.
6	selfage = age # Private attribute	54	print("Gender: " + self.get_gender())
	for age.	10-10-1	# Print the first child's gender.
7	selfgender = gender # Private	55	
	attribute for gender.	56	def Role(self): # Method to display the
8			role of the first child.
9	def get_name(self): # Method to get	57	print("The First child") # Print the role
	the name of the family member.		of the first child.
10	return selfname	58	
11		59	class SecondChild(FamilyMember): #
12	def get_age(self): # Method to get the		Subclass representing the second child,
12	age of the family member.		inheriting from FamilyMember.
13		60	definit(self, name, age, gender): #
	return selfage		Constructor method for SecondChild
14			class.
15	def get_gender(self): # Method to get	61	super()init(name, age, gender) #
	the gender of the family member.		Calling the constructor of the superclass.
16	return self. <u>g</u> ender	62	







"For Nation's Greater

17 18	@abstractmethod # Decorator	63	def Family_info(self): # Method to display family information for the second
	indicating that the method is abstract and		child.
	must be implemented by subclasses.	64	print("Name: " + self.get_name()) #
19	def Family_info(self): # Abstract		Print the second child's name.
	method to display family information.	65	print("Gender: " + self.get_gender())
20	pass		# Print the second child's gender.
21	puss	66	
25.50	Cabatrastmethod # Deservice	67	def Role(self): # Method to display the
22	@abstractmethod # Decorator	07	
	indicating that the method is abstract and		role of the second child.
886.586.5	must be implemented by subclasses.	68	print("The Second child") # Print the
23	def Role(self): # Abstract method to		role of the second child.
	display the role of the family member.	69	
24	pass	70	class ThirdChild(FamilyMember): #
25			Subclass representing the third child,
26	class Father(FamilyMember): # Subclass		inheriting from FamilyMember.
SEC. 1	representing a father, inheriting from	71	definit(self, name, age, gender): #
	FamilyMember.		Constructor method for ThirdChild class.
27	definit(self, name, age, gender): #	72	super()init(name, age, gender) #
21	Constructor method for Father class.	, _	Calling the constructor of the superclass.
0.0		73	calling the constructor of the superclass.
28	super()init(name, age, gender) #	74	dof Family info(colf): # Mathod to
	Calling the constructor of the superclass.	74	def Family_info(self): # Method to
29			display family information for the third
30	def Family_info(self): # Method to		child.
	display family information for a father.	75	print("Name: " + self.get_name()) #
31	print("Name: " + self.get_name()) #		Print the third child's name.
	Print the father's name.	76	<pre>print("Gender: " + self.get_gender())</pre>
32	print("Gender: " + self.get_gender())		# Print the third child's gender.
	# Print the father's gender.	77 78	
33		78	def Role(self): # Method to display the
34	def Role(self): # Method to display the		role of the third child.
	role of a father.	79	print("The Third child") # Print the
35	print("The Father") # Print the role of	0.0	role of the third child.
	a father.	80	# O
36	a rather.	81	# Creating instances of the subclasses
37	alaca Mathar(FamilyMamhar): #	82	Person1 = Father("Romeo", 65, "Male") #
3/	class Mother(FamilyMember): #	83	Creating an instance of Father class.
	Subclass representing a mother,	03	Person2 = Mother("Maria", 57, "Female") # Creating an instance of Mother class.
100.00	inheriting from FamilyMember.	84	Person3 = FirstChild("Sam", 35, "Male") #
38	definit(self, name, age, gender): #	04	Creating an instance of FirstChild class.
	Constructor method for Mother class.	85	Person4 = ThirdChild("Christine", 19,
39	super()init(name, age, gender) #	00	"Female") # Creating an instance of
	Calling the constructor of the superclass.		ThirdChild class.
40	caming and content actor or and cap crosses.	86	Person5 = SecondChild("Joe", 23, "Male")
41	def Family_info(self): # Method to	00	# Creating an instance of SecondChild
41			class.
40	display family information for a mother.	87	Cidoo.
42	print("Name: " + self.get_name()) #	88	# Calling methods for each instance
	Print the mother's name.	89	print("========") #
43	print("Gender: " + self.get_gender())	0,5	Printing a separator.
	# Print the mother's gender.	90	Person1.Family_info() # Calling
44			Family_info method for Person1.
45	def Role(self): # Method to display the	91	Person1.Role() # Calling Role method for
	role of a mother.		Person1.
16		92	print("=======") #
46	print("The Mother") # Print the role of		Printing a separator.
	a mother.	93	print() # Printing an empty line.
47		94	print("========") #
			Printing a separator.





95	Person2.Family_info() # Calling
	Family_info method for Person2.
96	Person2.Role() # Calling Role method for
	Person2.
97	print("=======") #
	Printing a separator.
98	print() # Printing an empty line.
99	print("=======") #
	Printing a separator.
100	Person3.Family_info() # Calling
	Family_info method for Person3.
101	Person3.Role() # Calling Role method for
	Person3
102	print("========") #
, 02	Printing a separator.
103	print() # Printing an empty line.
104	print("========") #
	Printing a separator.
105	Person4.Family_info() # Calling
100	Family_info method for Person4.
106	Person4.Role() # Calling Role method for
100	Person4.
107	print("=======" #
107	Printing a separator.
108	print() # Printing an empty line.
109	print() # Finding an empty line.
109	Printing a separator.
110	Person5.Family_info() # Calling
110	Family_info method for Person5.
111	Person5.Role() # Calling Role method for
1.1.1	Person5.
112	print("====================================
112	Printing a separator.
	Filliting a separator.

User Guide:

Step1:

Start the program by running the Pydroid 3 application in android phone.

Step2:

• Find the file of the code and open the file.

Step3:

• After accessing the file, you can run it now just input the button below.

Step4:

• You can now see the output in the console.





Output of the code:

```
amily Information Class
-----------------
Name: Romeo
Gender: Male
The Father
----------------
----------------
Name: Maria
Gender: Female
The Mother
______
----------------
Name: Sam
Gender: Male
The First child
 ______
Name: Christine
Gender: Female
The Third child
Name: Joe
Gender: Male
The Second child
[Program finished]
```

This part of the code displays the output of family members, identifies their family roles, and provides specific basic information about a family hierarchy structure. By using the four types of fundamental principles of OOP, we built a code more efficient with a great organized and clean structure.

Conclusion:

The program serves as a practical starting point for modeling familial relationships in programming, demonstrating concepts like inheritance and polymorphism while offering a foundation for expanding functionalities to suit real-world scenarios, such as managing family events or tracking generational differences.



Reference:

Inheritance Guide: https://www.w3schools.com/python/python_inheritance.asp

Polymorphism Guide: https://www.youtube.com/watch?v=C2QfkDcQ5MU

Encapsulation Guide: https://www.tutorialspoint.com/python/python_encapsulation.htm

Abstraction Guide: https://www.geeksforgeeks.org/abstract-classes-in-python/