Republic of the Philippines
SURIGAO DEL NORTE STATE UNIVERSITY
Narciso Street, Surigao City 8400, Philippines

"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

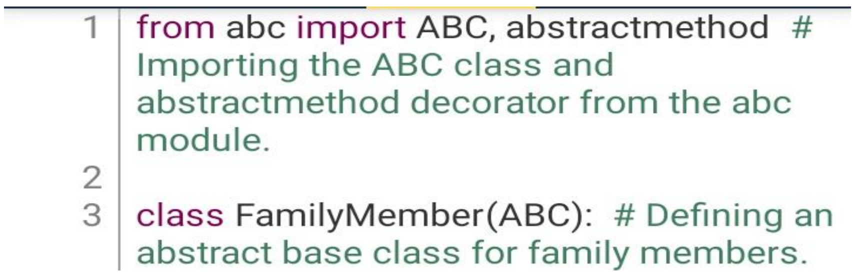
Software:

● Pydroid 3

PRINCIPLES OF OOP BEING USED IN THE CODE:

Abstraction with Inheritance:

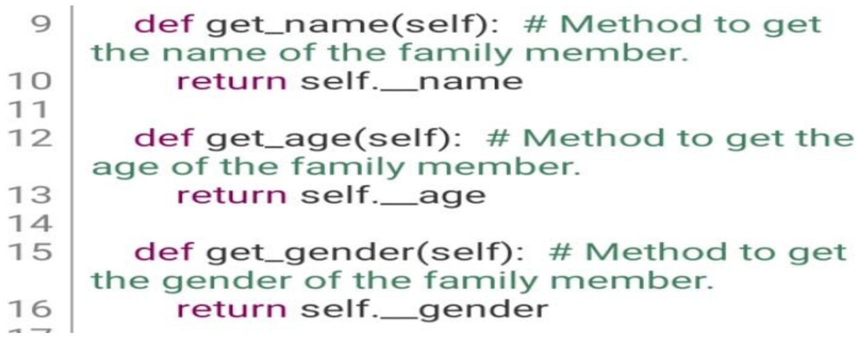
This line imports the ABC (Abstract Base Class) and abstract method from the abc module, which are used for defining abstract base classes and abstract methods. The class FamilyMember has a parameter of ABC which defines as abstract base class and inherits from ABC. The FamilyMember class that will serve 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.



```
from abc import ABC, abstractmethod  # Importing the ABC class and abstractmethod decorator from the abc module.

class FamilyMember(ABC):  # Defining an abstract base class for family members.
```

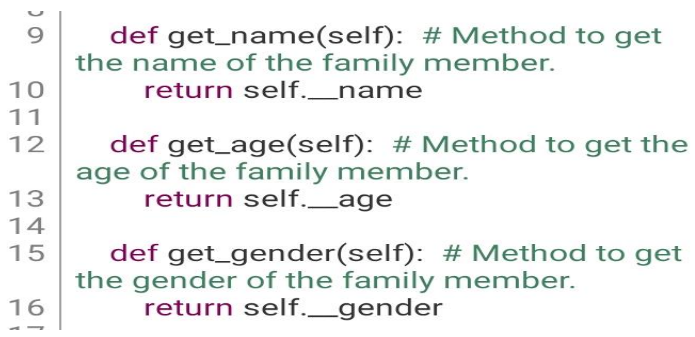
Encapsulation:

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.



```
    def get_name(self):  # Method to get
the name of the family member.
        return self.__name

    def get_age(self):  # Method to get the
age of the family member.
        return self.__age

    def get_gender(self):  # Method to get
the gender of the family member.
        return self.__gender
```

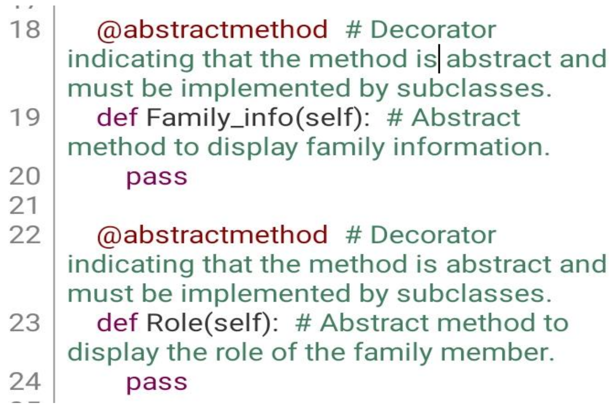
This line of code is defining a method to retrieve the name, age and gender of a FamilyMember. These three methods provide controlled access to the private attribute name, age and gender by returning its value when called. It encapsulates the data by providing a getter method to retrieve the name, age,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



```
    def get_name(self):  # Method to get
the name of the family member.
        return self.__name

    def get_age(self):  # Method to get the
age of the family member.
        return self.__age

    def get_gender(self):  # Method to get
the gender of the family member.
        return self.__gender
```

Abstraction:

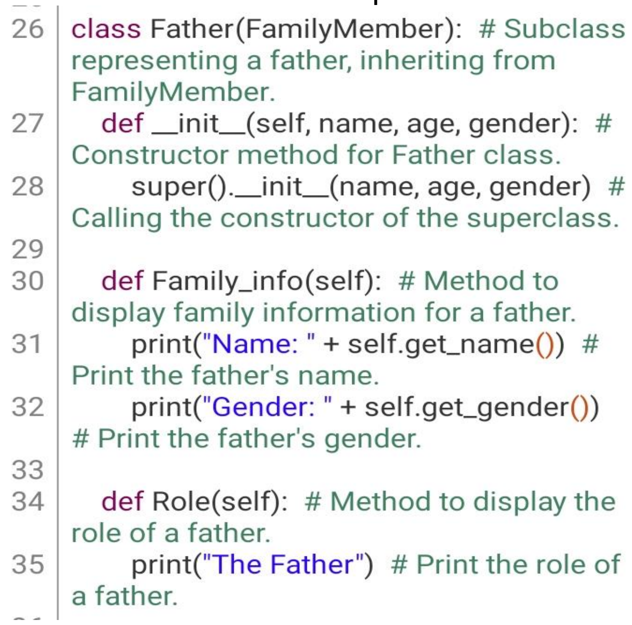
The purpose is to declaring an abstract method Family_info and Role This line marks the Family_info and Role method as abstract, indicating that any subclass of FamilyMember must provide its own implementation of this method. The pass statement serves as a placeholder, indicating that the method hasnoimplementation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



```
    @abstractmethod  # Decorator
indicating that the method is abstract and
must be implemented by subclasses.
    def Family_info(self):  # Abstract
method to display family information.
        pass

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

Inheritance:

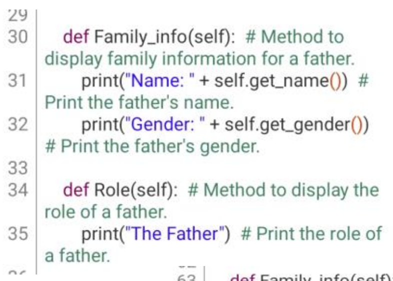
This line defines a subclass Father that inherits from the FamilyMember class, the Father class inherits attributes and methods from the FamilyMember class. It specializes in the FamilyMember class to represent a father. Similarly, the classes Mother, FirstChild, SecondChild, and. ThirdChild follow the 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.



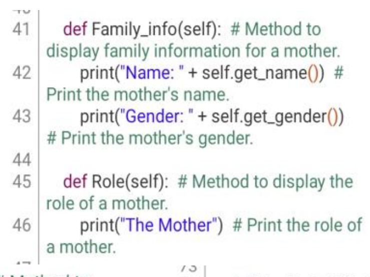
```python
class Father(FamilyMember):  # Subclass representing a father, inheriting from FamilyMember.
    def __init__(self, name, age, gender):  # Constructor method for Father class.
        super().__init__(name, age, gender)  # Calling the constructor of the superclass.

    def Family_info(self):  # Method to display family information for a father.
        print("Name: " + self.get_name())  # Print the father's name.
        print("Gender: " + self.get_gender())  # Print the father's gender.

    def Role(self):  # Method to display the role of a father.
        print("The Father")  # Print the role of a father.
```

Polymorphism:

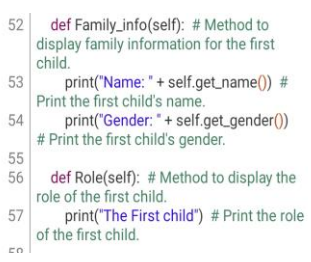
The Family_info and Role method is a polymorphism, and the rest of the of the subclass has this method name for Family_info and Role. Similarly, the classes Mother, FirstChild, SecondChild, and ThirdChild follow the same pattern of inheritance from the FamilyMember class.



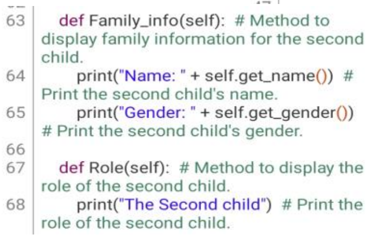
```python
    def Family_info(self):  # Method to display family information for a father.
        print("Name: " + self.get_name())  # Print the father's name.
        print("Gender: " + self.get_gender())  # Print the father's gender.

    def Role(self):  # Method to display the role of a father.
        print("The Father")  # Print the role of a father.
```



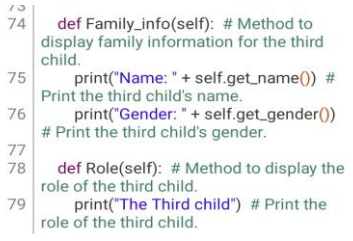
```python
    def Family_info(self):  # Method to display family information for a mother.
        print("Name: " + self.get_name())  # Print the mother's name.
        print("Gender: " + self.get_gender())  # Print the mother's gender.

    def Role(self):  # Method to display the role of a mother.
        print("The Mother")  # Print the role of a mother.
```



```python
    def Family_info(self):  # Method to display family information for the first child.
        print("Name: " + self.get_name())  # Print the first child's name.
        print("Gender: " + self.get_gender())  # Print the first child's gender.

    def Role(self):  # Method to display the role of the first child.
        print("The First child")  # Print the role of the first child.
```

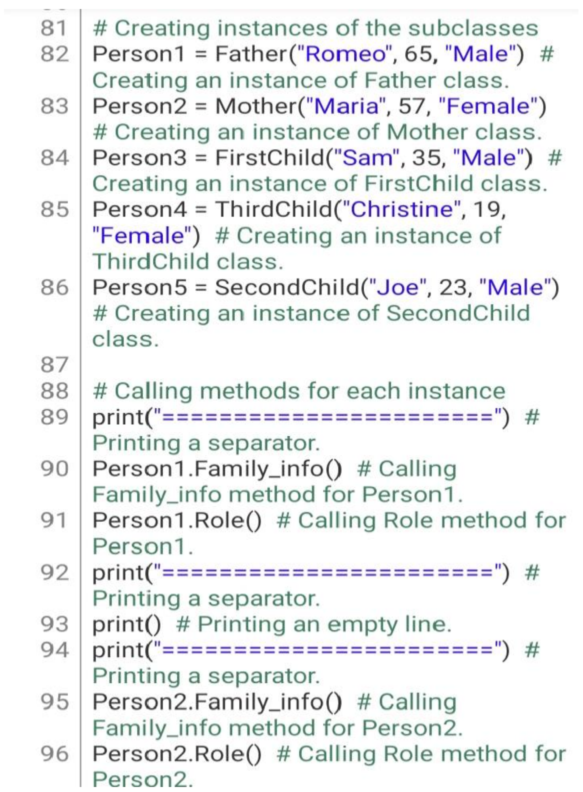


```python
    def Family_info(self):  # Method to display family information for the second child.
        print("Name: " + self.get_name())  # Print the second child's name.
        print("Gender: " + self.get_gender())  # Print the second child's gender.

    def Role(self):  # Method to display the role of the second child.
        print("The Second child")  # Print the role of the second child.
```

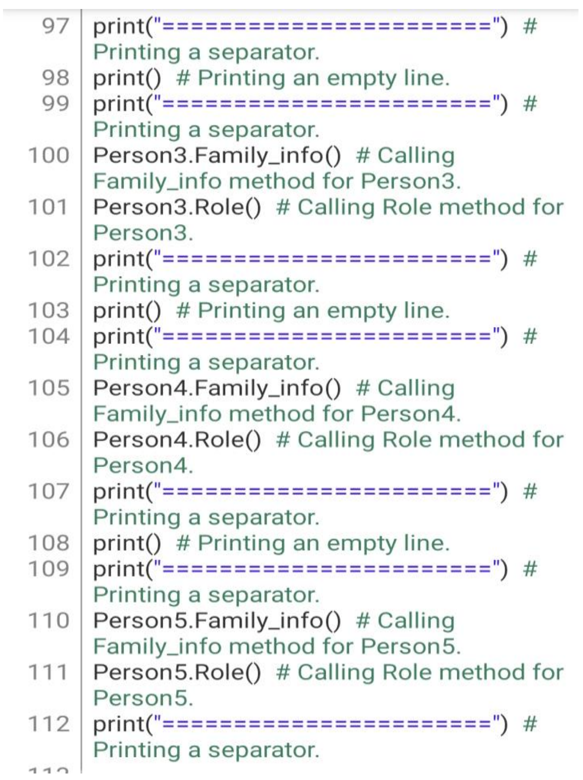


```python
    def Family_info(self):  # Method to display family information for the third child.
        print("Name: " + self.get_name())  # Print the third child's name.
        print("Gender: " + self.get_gender())  # Print the third child's gender.

    def Role(self):  # Method to display the role of the third child.
        print("The Third child")  # Print the role of the third child.
```

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.

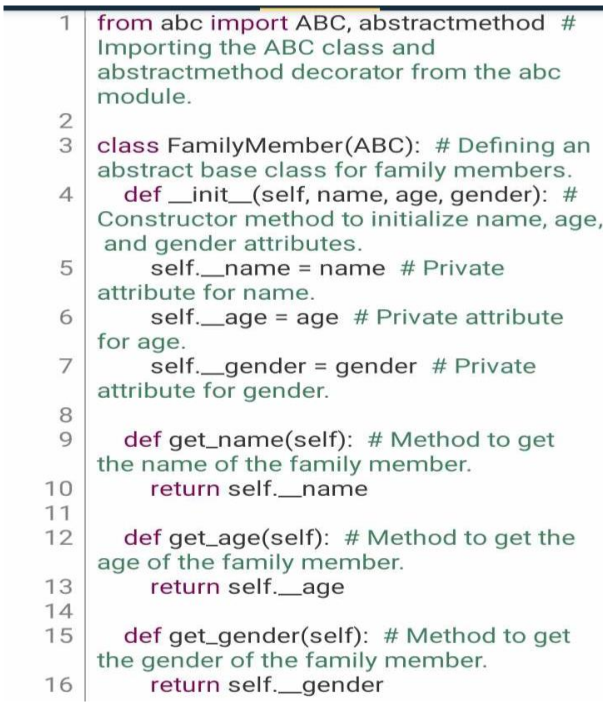


```python
# Creating instances of the subclasses
Person1 = Father("Romeo", 65, "Male")  # Creating an instance of Father class.
Person2 = Mother("Maria", 57, "Female")  # Creating an instance of Mother class.
Person3 = FirstChild("Sam", 35, "Male")  # Creating an instance of FirstChild class.
Person4 = ThirdChild("Christine", 19, "Female")  # Creating an instance of ThirdChild class.
Person5 = SecondChild("Joe", 23, "Male")  # Creating an instance of SecondChild class.

# Calling methods for each instance
print("=======================")  # Printing a separator.
Person1.Family_info()  # Calling Family_info method for Person1.
Person1.Role()  # Calling Role method for Person1.
print("=======================")  # Printing a separator.
print()  # Printing an empty line.
print("=======================")  # Printing a separator.
Person2.Family_info()  # Calling Family_info method for Person2.
Person2.Role()  # Calling Role method for Person2.
```

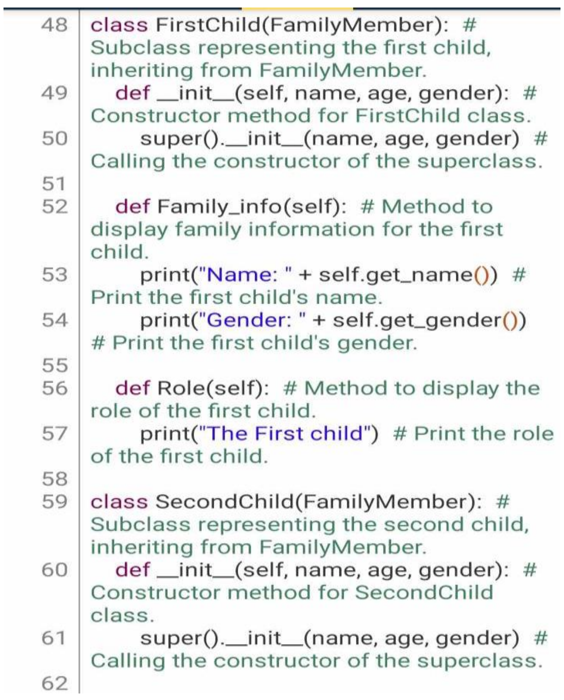


```python
print("=======================")  # Printing a separator.
print()  # Printing an empty line.
print("=======================")  # Printing a separator.
Person3.Family_info()  # Calling Family_info method for Person3.
Person3.Role()  # Calling Role method for Person3.
print("=======================")  # Printing a separator.
print()  # Printing an empty line.
print("=======================")  # Printing a separator.
Person4.Family_info()  # Calling Family_info method for Person4.
Person4.Role()  # Calling Role method for Person4.
print("=======================")  # Printing a separator.
print()  # Printing an empty line.
print("=======================")  # Printing a separator.
Person5.Family_info()  # Calling Family_info method for Person5.
Person5.Role()  # Calling Role method for Person5.
print("=======================")  # Printing a separator.
```

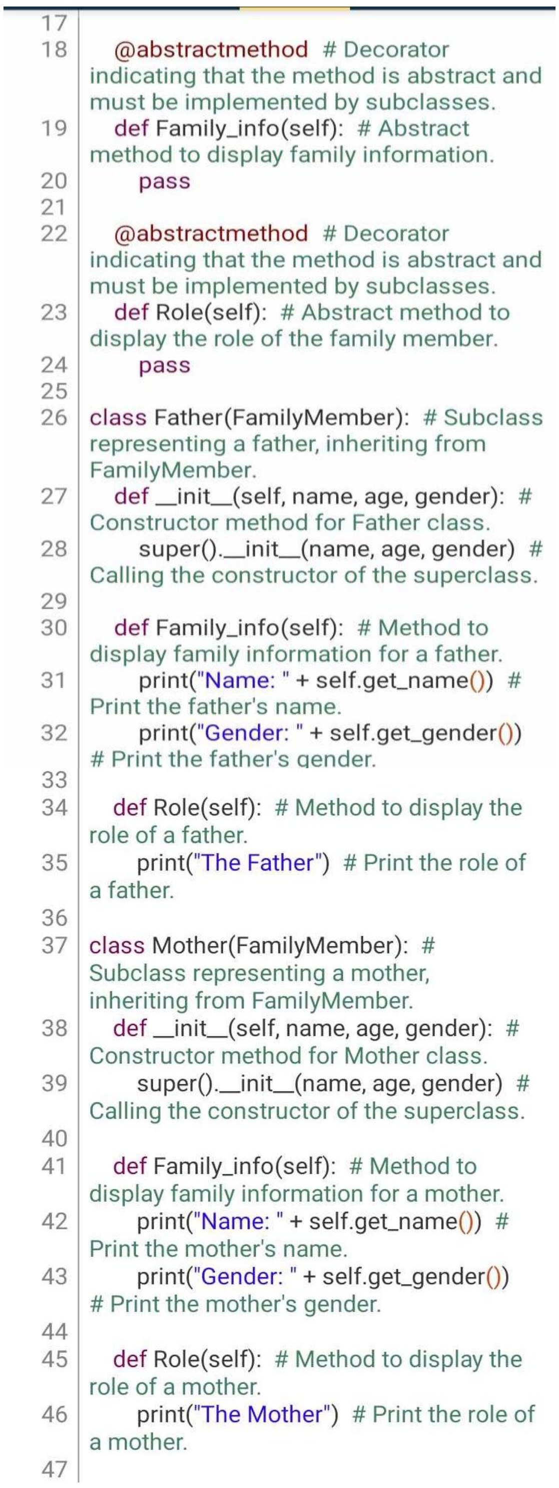
Code:



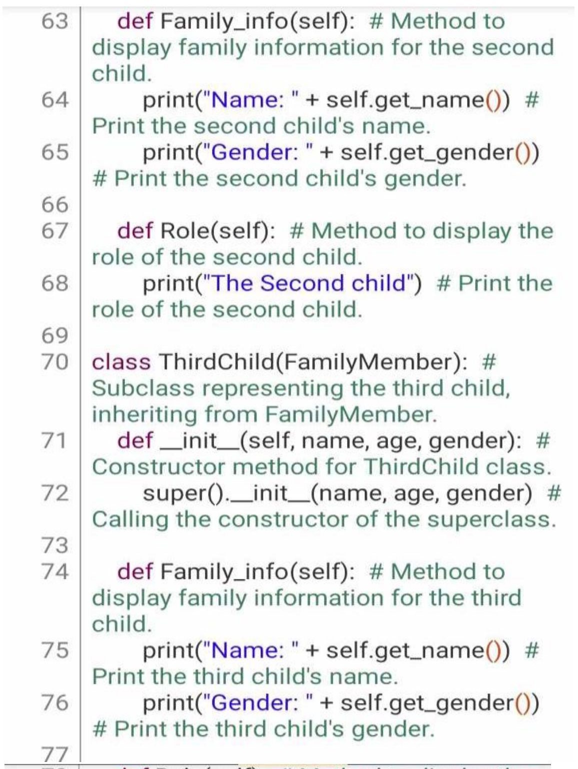
```python
from abc import ABC, abstractmethod  # Importing the ABC class and abstractmethod decorator from the abc module.

class FamilyMember(ABC):  # Defining an abstract base class for family members.
    def __init__(self, name, age, gender):  # Constructor method to initialize name, age, and gender attributes.
        self.__name = name  # Private attribute for name.
        self.__age = age  # Private attribute for age.
        self.__gender = gender  # Private attribute for gender.

    def get_name(self):  # Method to get the name of the family member.
        return self.__name

    def get_age(self):  # Method to get the age of the family member.
        return self.__age

    def get_gender(self):  # Method to get the gender of the family member.
        return self.__gender
```



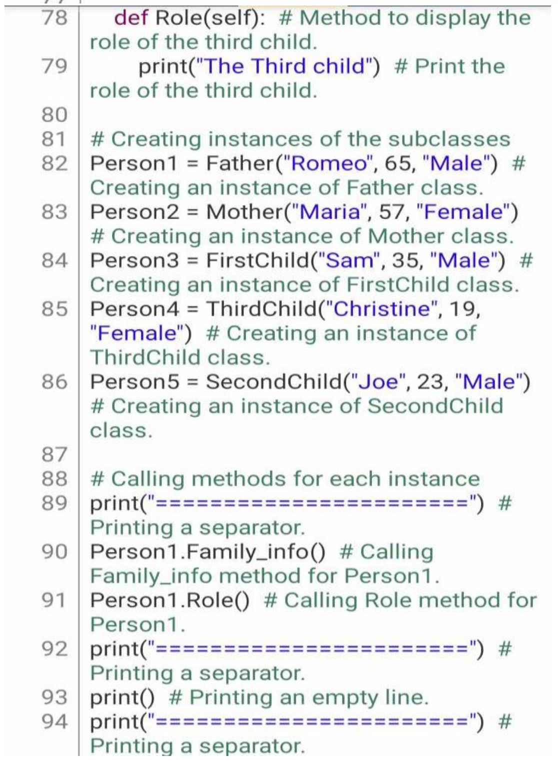
```python
class FirstChild(FamilyMember):  # Subclass representing the first child, inheriting from FamilyMember.
    def __init__(self, name, age, gender):  # Constructor method for FirstChild class.
        super().__init__(name, age, gender)  # Calling the constructor of the superclass.

    def Family_info(self):  # Method to display family information for the first child.
        print("Name: " + self.get_name())  # Print the first child's name.
        print("Gender: " + self.get_gender())  # Print the first child's gender.

    def Role(self):  # Method to display the role of the first child.
        print("The First child")  # Print the role of the first child.

class SecondChild(FamilyMember):  # Subclass representing the second child, inheriting from FamilyMember.
    def __init__(self, name, age, gender):  # Constructor method for SecondChild class.
        super().__init__(name, age, gender)  # Calling the constructor of the superclass.

```



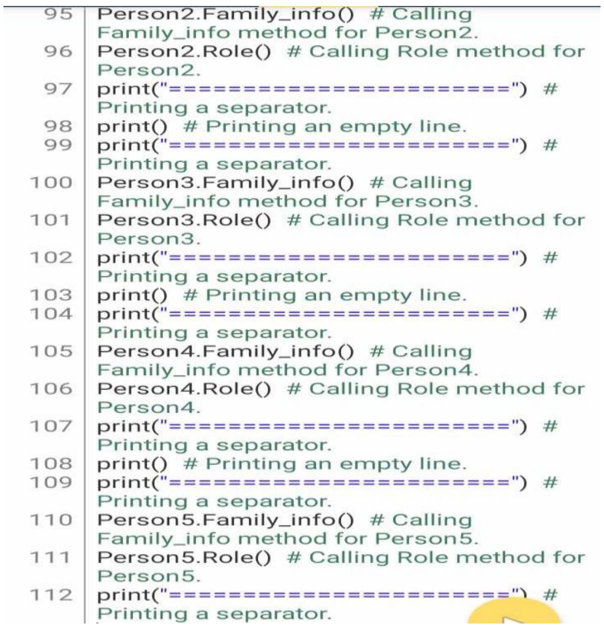
```python

    @abstractmethod  # Decorator indicating that the method is abstract and must be implemented by subclasses.
    def Family_info(self):  # Abstract method to display family information.
        pass

    @abstractmethod  # Decorator indicating that the method is abstract and must be implemented by subclasses.
    def Role(self):  # Abstract method to display the role of the family member.
        pass

class Father(FamilyMember):  # Subclass representing a father, inheriting from FamilyMember.
    def __init__(self, name, age, gender):  # Constructor method for Father class.
        super().__init__(name, age, gender)  # Calling the constructor of the superclass.

    def Family_info(self):  # Method to display family information for a father.
        print("Name: " + self.get_name())  # Print the father's name.
        print("Gender: " + self.get_gender())  # Print the father's gender.

    def Role(self):  # Method to display the role of a father.
        print("The Father")  # Print the role of a father.

class Mother(FamilyMember):  # Subclass representing a mother, inheriting from FamilyMember.
    def __init__(self, name, age, gender):  # Constructor method for Mother class.
        super().__init__(name, age, gender)  # Calling the constructor of the superclass.

    def Family_info(self):  # Method to display family information for a mother.
        print("Name: " + self.get_name())  # Print the mother's name.
        print("Gender: " + self.get_gender())  # Print the mother's gender.

    def Role(self):  # Method to display the role of a mother.
        print("The Mother")  # Print the role of a mother.

```



```python
    def Family_info(self):  # Method to display family information for the second child.
        print("Name: " + self.get_name())  # Print the second child's name.
        print("Gender: " + self.get_gender())  # Print the second child's gender.

    def Role(self):  # Method to display the role of the second child.
        print("The Second child")  # Print the role of the second child.

class ThirdChild(FamilyMember):  # Subclass representing the third child, inheriting from FamilyMember.
    def __init__(self, name, age, gender):  # Constructor method for ThirdChild class.
        super().__init__(name, age, gender)  # Calling the constructor of the superclass.

    def Family_info(self):  # Method to display family information for the third child.
        print("Name: " + self.get_name())  # Print the third child's name.
        print("Gender: " + self.get_gender())  # Print the third child's gender.

```



```python
    def Role(self):  # Method to display the role of the third child.
        print("The Third child")  # Print the role of the third child.

# Creating instances of the subclasses
Person1 = Father("Romeo", 65, "Male")  # Creating an instance of Father class.
Person2 = Mother("Maria", 57, "Female")  # Creating an instance of Mother class.
Person3 = FirstChild("Sam", 35, "Male")  # Creating an instance of FirstChild class.
Person4 = ThirdChild("Christine", 19, "Female")  # Creating an instance of ThirdChild class.
Person5 = SecondChild("Joe", 23, "Male")  # Creating an instance of SecondChild class.

# Calling methods for each instance
print("======================")  # Printing a separator.
Person1.Family_info()  # Calling Family_info method for Person1.
Person1.Role()  # Calling Role method for Person1.
print("======================")  # Printing a separator.
print()  # Printing an empty line.
print("======================")  # Printing a separator.
```



```
Person2.Family_info()  # Calling Family_info method for Person2.
Person2.Role()  # Calling Role method for Person2.
print("=======================")  # Printing a separator.
print()  # Printing an empty line.
print("=======================")  # Printing a separator.
Person3.Family_info()  # Calling Family_info method for Person3.
Person3.Role()  # Calling Role method for Person3.
print("=======================")  # Printing a separator.
print()  # Printing an empty line.
print("=======================")  # Printing a separator.
Person4.Family_info()  # Calling Family_info method for Person4.
Person4.Role()  # Calling Role method for Person4.
print("=======================")  # Printing a separator.
print()  # Printing an empty line.
print("=======================")  # Printing a separator.
Person5.Family_info()  # Calling Family_info method for Person5.
Person5.Role()  # Calling Role method for Person5.
print("=======================")  # Printing 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.

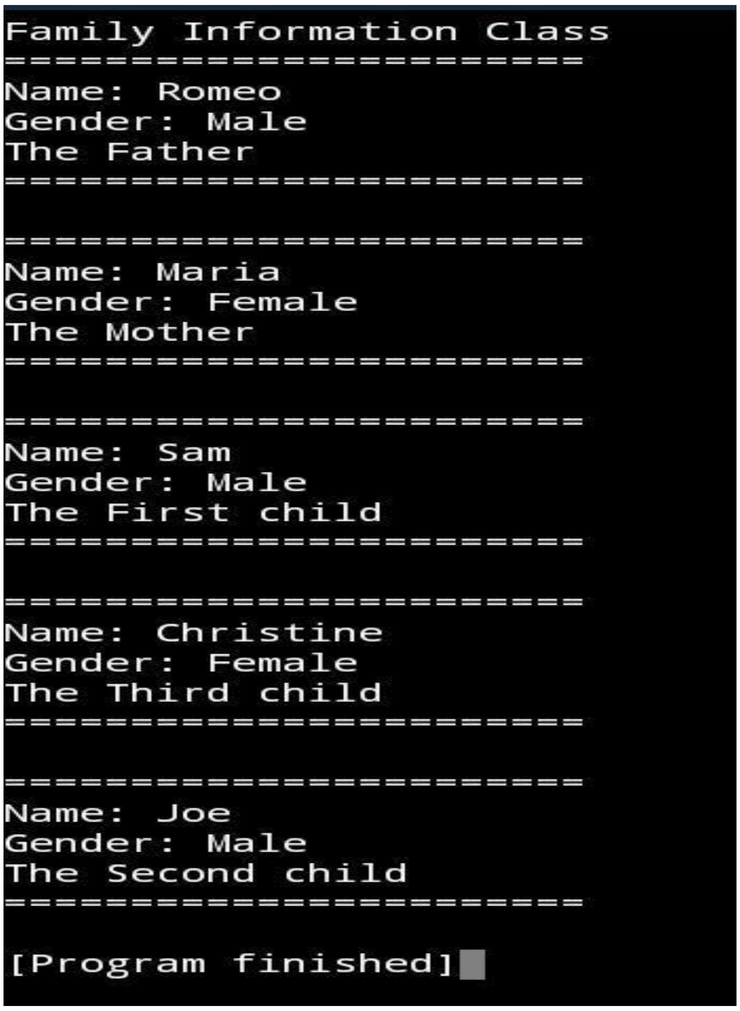
Step3:

- 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:



```
Family 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/