```
from abc import ABC, abstractmethod

# HERENCIA
class Figura:
    def __init__(self, nombre, color):
        self.nombre = nombre
        self.color = color

    def mostrar(self):
        print(f"{self.nombre} - Color: {self.color}")

class Circulo(Figura):
    def __init__(self, color, radio):
        super().__init__("Círculo", color)
        self.radio = radio

    def area(self):
        return 3.14 * self.radio ** 2

    def perimetro(self):
        return 2 * 3.14 * self.radio

    def mostrar(self):
        super().mostrar()
        print(f"Área: {self.area()}")
        print(f"Perímetro: {self.perimetro()}")

class Rectangulo(Figura):
    def __init__(self, color, ancho, alto):
        super().__init__("Rectángulo", color)
        self.ancho = ancho
        self.alto = alto

    def area(self):
        return self.ancho * self.alto

    def perimetro(self):
        return 2 * (self.ancho + self.alto)

    def mostrar(self):
        super().mostrar()
        print(f"Área: {self.area()}")
        print(f"Perímetro: {self.perimetro()}")

# ABSTRACCIÓN
class Vehiculo(ABC):
    @abstractmethod
    def encender(self): pass
```

```
    @abstractmethod
    def apagar(self): pass

    @abstractmethod
    def conducir(self): pass

class Coche(Vehiculo):
    def encender(self): print("Coche encendido")
    def apagar(self): print("Coche apagado")
    def conducir(self): print("Conduciendo coche")

class Bicicleta(Vehiculo):
    def encender(self): print("La bici no se enciende")
    def apagar(self): print("La bici no se apaga")
    def conducir(self): print("Pedaleando")

# ENCAPSULAMIENTO
class Cuenta:
    def __init__(self, numero, saldo):
        self.__numero = numero
        self.__saldo = saldo

    def depositar(self, monto):
        self.__saldo += monto
        print(f"Depositado: {monto}. Saldo: {self.__saldo}")

    def retirar(self, monto):
        if monto <= self.__saldo:
            self.__saldo -= monto
            print(f"Retirado: {monto}. Saldo: {self.__saldo}")
        else:
            print("Saldo insuficiente")

    def get_saldo(self):
        return self.__saldo

# POLIMORFISMO
class Empleado(ABC):
    @abstractmethod
    def calcular_salario(self): pass

class PorHora(Empleado):
    def __init__(self, horas, tarifa):
        self.horas = horas
        self.tarifa = tarifa

    def calcular_salario(self):
```

```
        return self.horas * self.tarifa

class Fijo(Empleado):
    def __init__(self, sueldo):
        self.sueldo = sueldo

    def calcular_salario(self):
        return self.sueldo

# COMBINACIÓN DE CONCEPTOS
class Estudiante(ABC):
    def __init__(self, nombre):
        self._nombre = nombre

    @abstractmethod
    def obtener_promedio(self): pass

    @abstractmethod
    def mostrar_info(self): pass

class Grado(Estudiante):
    def __init__(self, nombre, notas):
        super().__init__(nombre)
        self.__notas = notas

    def obtener_promedio(self):
        return sum(self.__notas) / len(self.__notas)

    def mostrar_info(self):
        print(f"{self._nombre} (Grado) - Promedio: {self.obtener_promedio():.2f}")

class Posgrado(Estudiante):
    def __init__(self, nombre, tesis):
        super().__init__(nombre)
        self.__tesis = tesis

    def obtener_promedio(self):
        return 10.0 if self.__tesis else 6.0

    def mostrar_info(self):
        print(f"{self._nombre} (Posgrado) - Tesis: {'Sí' if self.__tesis else 'No'} - Promedio:
{self.obtener_promedio()}")

# EJECUCIÓN
if __name__ == "__main__":
    print("\n--- HERENCIA ---")
    c = Circulo("Azul", 3)
    r = Rectangulo("Rojo", 4, 5)
```

```
c.mostrar()
r.mostrar()

print("\n--- ABSTRACCIÓN ---")
v1 = Coche()
v2 = Bicicleta()
v1.encender(); v1.conducir(); v1.apagar()
v2.encender(); v2.conducir(); v2.apagar()

print("\n--- ENCAPSULAMIENTO ---")
cuenta = Cuenta("123", 1000)
cuenta.depositar(300)
cuenta.retirar(200)
cuenta.retirar(2000)

print("\n--- POLIMORFISMO ---")
lista_empleados = [PorHora(40, 15), Fijo(3000)]
for emp in lista_empleados:
    print(f"Salario: {emp.calcular_salario()}")

print("\n--- COMBINACIÓN ---")
estudiantes = [
    Grado("Lucas", [8, 9, 10]),
    Posgrado("Ana", True)
]
for est in estudiantes:
    est.mostrar_info()
```