```
import re
import datetime
from abc import ABC, abstractmethod

class CodiceFiscale:
def __new__(cls, codice_fiscale: str):
if len(codice_fiscale) != 16 or not re.fullmatch(r'^[A-Z]{6}[0-9]{2}[A-Z]{1}[0-9]{2}[A-Z]{1}[0-9]{3}[A-Z]{1}$',
codice_fiscale):
raise ValueError(f"Errore. Codice Fiscale {codice_fiscale} non valido.")
istanza = super().__new__(cls)
istanza.codice_fiscale = codice_fiscale
return istanza

def __str__(self):
return self.codice_fiscale

def __eq__(self, other):
return isinstance(other, Persona) and self.codice_fiscale == other.codice_fiscale
def __hash__(self):
return hash(self.codice_fiscale)

class DataNascita:
def __new__(cls, data_nascita: str):
if not re.fullmatch(r'^(0[1-9]|[12][0-9]|3[01])/(0[1-9]|1[0-2])/((19|20)\d{2})$', data_nascita):
raise ValueError(f"Errore. Data di nascita {data_nascita} non valida.")
try:
parsed = datetime.datetime.strptime(data_nascita, "%d/%m/%Y").date()
except ValueError:
raise ValueError(f"Errore. La data '{data_nascita}' non esiste nel calendario.")

today = datetime.date.today()
max_data_valida = datetime.date(1895, 1, 1)
if parsed > today:
raise ValueError(f"Errore. La data '{data_nascita}' è nel futuro.")
if parsed < max_data_valida:
raise ValueError(f"Errore. La data '{data_nascita}' è troppo vecchia.")
istanza = super().__new__(cls)
istanza.data_nascita = parsed
return istanza
def __str__(self):
return self.data_nascita.strftime("%d/%m/%Y")
def __eq__(self, other):
return isinstance(other, Persona) and self.data_nascita == other.data_nascita
def __hash__(self):
return hash(self.data_nascita)
```

```
class Persona:
def __init__(self, nome: str, cognome: str, data_nascita: DataNascita, codice_fiscale: CodiceFiscale):
self.setNome(nome)
self.setCognome(cognome)
self.setDataNascita(data_nascita)
self.setCodiceFiscale(codice_fiscale)

def __str__(self):
return f"Nome: {self.nome}\nCognome: {self.cognome}\nCodice Fiscale: {self.codice_fiscale}\nData di nascita:
{self.data_nascita}"

def setNome(self, nome: str) -> str:
if not isinstance(nome, str) or nome.strip() == "":
raise ValueError(f"Il nome inserito non è valido.")
else:
self.nome = nome

def setCognome(self, cognome: str) -> str:
if not isinstance(cognome, str) or cognome.strip() == "":
raise ValueError(f"Il cognome inserito non è valido.")
else:
self.cognome = cognome
def setDataNascita(self, data_nascita):
self.data_nascita = data_nascita

def setCodiceFiscale(self, codice_fiscale):
self.codice_fiscale = codice_fiscale

def getNome(self):
return self.nome

def getCognome(self):
return self.cognome

def getDataNascita(self):
return self.data_nascita

def getCodiceFiscale(self):
return self.codice_fiscale

def __eq__(self, other):
return isinstance(other, Persona) and self.codice_fiscale == other.codice_fiscale

def __hash__(self):
return hash(self.codice_fiscale)
```

```
class PosizioneMilitare:
def __init__(self, nome: str):
if not isinstance(nome, str) or nome.strip == "":
raise ValueError("Inserire una posizione militare valida.")
self.nome = nome

def __str__(self):
return f"{self.nome}"

def __eq__(self, other):
return isinstance(other, PosizioneMilitare) and self.nome == other.nome

def __hash__(self):
return hash(self.nome)


class Uomo(Persona):
def __init__(self, nome: str, cognome: str, data_nascita: DataNascita, codice_fiscale: CodiceFiscale,
posizione_militare: PosizioneMilitare):
super().__init__(nome, cognome, data_nascita, codice_fiscale)
self.posizione_militare = posizione_militare

def __str__(self):
return f"{super().__str__()}\nPosizione militare: {self.posizione_militare}"


class Donna(Persona):
def __init__(self, nome: str, cognome: str, data_nascita: DataNascita, codice_fiscale: CodiceFiscale,
numero_maternità: int):
super().__init__(nome, cognome, data_nascita, codice_fiscale)
if not isinstance(numero_maternità, int) or numero_maternità < 0:
raise ValueError("Inserire un numero di maternità valido.")
self.numero_maternità = numero_maternità

def __str__(self):
return f"{super().__str__()}\nNumero di figli: {self.numero_maternità}."


class Studente(Persona):
def __init__(self, nome: str, cognome: str, data_nascita: DataNascita, codice_fiscale: CodiceFiscale,
numero_matricola: int):
super().__init__(nome, cognome, data_nascita, codice_fiscale)
if not isinstance(numero_matricola, int) or numero_matricola < 0:
raise ValueError("Numero di matricola non valido.")
self.numero_matricola = numero_matricola

def __eq__(self, other):
```

```
return isinstance(other, Studente) and self.numero_matricola == other.numero_matricola
def __hash__(self):
return hash(self.numero_matricola)

def __str__(self):
return f"{super().__str__()}\nNumero di matricola: {self.numero_matricola}."


class Impiegato(Persona, ABC):
def __init__(self, stipendio: float):
if not isinstance(stipendio, float) or stipendio < 0:
raise ValueError("Stipendio non valido.")
self.stipendio = stipendio

@abstractmethod
def ruolo(self):
pass

class Direttore(Impiegato):
def __init__(self, stipendio: float):
super().__init__(stipendio)

def ruolo(self):
return "Direttore"

def __str__(self):
return f"Direttore con stipendio di {self.stipendio} euro."


class Segretario(Impiegato):
def __init__(self, stipendio: float):
super().__init__(stipendio)

def ruolo(self):
return "Segretario"

def __str__(self):
return f"Segretario con stipendio di {self.stipendio} euro."


class Progettista(Impiegato):
def __init__(self, stipendio: float, is_responsabile:bool, nome_prog: str = None):
super().__init__(stipendio)
self.is_responsabile = is_responsabile
if is_responsabile:
if not isinstance(nome_prog, str) or nome_prog.strip() == "":
raise ValueError("Inserire un nome di progetto valido.")
```

```
self.nome_prog = nome_prog
else:
self.nome_prog = None

def ruolo(self):
return "Progettista"

def __str__(self):
if self.is_responsabile:
return f"Il progettista è responsabile del progetto {self.nome_prog} e ha uno stipendio di {self.stipendio} euro."
else:
return f"Il progettista non è responsabile di alcun progetto ma ha uno stipendio di {self.stipendio} euro."



p1: Persona = Persona("Lisa", "Po", DataNascita("22/11/2022"), CodiceFiscale("BDFILS76G32F738C"))
p2: Persona = Persona("Gio", "Lu", DataNascita("01/01/1900"), CodiceFiscale("GBUSGI87F23G649K"))
p3: Persona = Persona("Virginia", "Bianchi", DataNascita("14/11/1996"), CodiceFiscale("VRGBNH96T62D892T"))

print(p1)

print(("\n--------------\n"))

print(p2)

print(("\n--------------\n"))

print(p3)

print(("\n--------------\n"))

print(p1 == p2)

print(("\n--------------\n"))

print(p2 == p3)

print(("\n--------------\n"))

direttore: Direttore = Direttore(1900.0)
print(direttore)

print("\n--------------\n")

segretario: Segretario = Segretario(1400.125)
print(segretario)
```

```
print("\n--------------\n")

progettista: Progettista = Progettista(2800.01, True, "Ponzio")
print(progettista)

print("\n--------------\n")

progettista2: Progettista = Progettista(2366.05, False)
print(progettista2)

print("\n--------------\n")

data_uomo = DataNascita("10/07/1950")
codice_uomo = CodiceFiscale("FINRTI67F74G938V")
posizione = PosizioneMilitare("Maresciallo")
uomo = Uomo("Mario", "Rossi", data_uomo, codice_uomo, posizione)
print(uomo)

print("\n--------------\n")

data_donna = DataNascita("18/01/1925")
codice_donna = CodiceFiscale("DKIGRI67F89G647D")
donna = Donna("Antonella", "Clerici", data_donna, codice_donna, 2)
print(donna)

print("\n--------------\n")
data_stud = DataNascita("19/04/1989")
codice_stud = CodiceFiscale("BHNJIY76T54G567I")
studente = Studente("Alessio", "Pono", data_stud, codice_stud, 856932)
print(studente)
```