```
from enum import *

class Genere(StrEnum):
	uomo = auto()
	donna = auto()
import re
class Indirizzo:
	def __init__(self, via: str, civico: str, cap: str):
		via = via.strip()
		civico = civico.strip()
		cap = cap.strip()
		if not self._valid_via(via):
			raise ValueError(f"Via non valida: '{via}'")
		if not self._valid_civico(civico):
			raise ValueError(f"Civico non valido: '{civico}'")
		if not self._valid_cap(cap):
			raise ValueError(f"CAP non valido: '{cap}'")
		self._via = via
		self._civico = civico
		self._cap = cap
	def _valid_via(self, via):
		return re.fullmatch(r"(via|piazza|corso|viale|lungo|strada|rotonda)?\s+", via,
	re.IGNORECASE) is not None
	def _valid_civico(self, civico):
		return re.fullmatch(r"\d+[A-Z]?", civico) is not None
	def _valid_cap(self, cap):
		return re.fullmatch(r"\d{5}", cap) is not None
	def via(self):
```

```
            return self._via

        def civico(self):

            return self._civico

        def cap(self):

            return self._cap

        def __eq__(self, other) -> bool:

            if other is None or not isinstance(other, Indirizzo) or hash(self) != hash(other):

                return False

            return (self.via(), self.civico(), self.cap()) == (other.via(), other.civico(),
        other.cap())

        def __hash__(self):

            return hash((self.via(), self.civico(), self.cap()))

    class Denaro:

        def __init__(self, valuta: str, importo: str):

            if not self._valid_valuta(valuta):

                raise ValueError(f"Valuta non valida: '{valuta}'")

            if not self._valid_importo(importo):

                raise ValueError(f"Importo non valido: '{importo}'")

            self._valuta = valuta

            self._importo = importo

        def _valid_valuta(self, valuta):

            return re.fullmatch(r"[A-Z]{3}", valuta) is not None

        def _valid_importo(self, importo):

            try:

                return float(importo) > 0

            except (ValueError, TypeError):

                return False
```

```
    def valuta(self):
        return self._valuta
    def importo(self):
        return self._importo
    def __eq__(self, other):
        if other is None or not isinstance(other, Denaro) or hash(self) != hash(other):
            return False
        return (self._valuta, self._importo) == (other._valuta, other._importo)
    def __hash__(self):
        return hash((self._valuta, self._importo))
class Email(str):
    def __new__(cls, mail: str):
        if not re.fullmatch(r"[^@]+@[^@]+\.[^@]+", mail):
            raise ValueError(f"Email non valida: '{mail}'")
        return str.__new__(cls, mail)
class Telefono(str):
    def __new__(cls, numero: str):
        numero = numero.strip()
        if re.fullmatch(r"(\+39)?3\d{8}", numero):
            return super().__new__(cls, numero)
        if re.fullmatch(r"0\d+", numero):
            return super().__new__(cls, numero)
        raise ValueError(f"Numero di telefono non valido: '{numero}'")
class CodiceFiscale(str):
    def __new__(cls, cf: str):
        if not re.fullmatch(r"[A-Z]{6}\d{2}[A-Z]\d{2}[A-L]\d{3}[A-Z0-9]", cf):
```

```
                raise ValueError(f"Codice fiscale non valido: '{cf}'")
            return str.__new__(cls, cf)
class PartitaIva(str):
    def __new__(cls, iva: str):
        if not re.fullmatch(r"\d{11}", iva):
            raise ValueError(f"Partita Iva non valida: '{iva}'")
        return str.__new__(cls, iva)
class Targa(str):
    def __new__(cls, targa: str):
        if not re.fullmatch(r"[A-Z]{2}\d{3}[A-Z]{2}", targa):
            raise ValueError(f"Targa non valida: '{targa}'")
        return str.__new__(cls, targa)

“””link repository github: https://github.com/JacopoSfolgor1/Jacopo-Sfolgori-UML.git”””
```