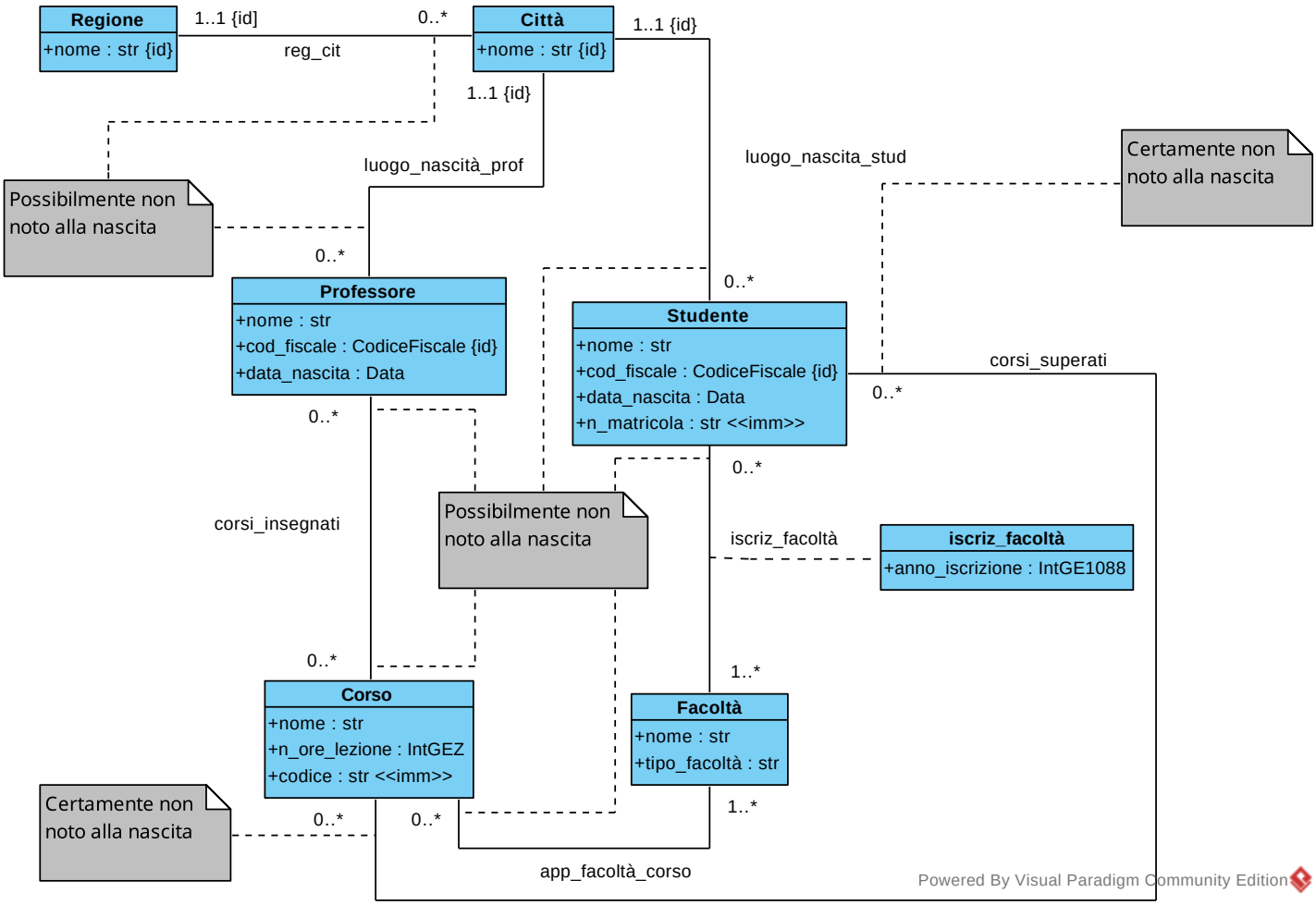
Regione
+nome : str {id}
1..1 {id}
reg_cit
0..*
Città
+nome : str {id}
1..1 {id}
1..1 {id}
luogo_nascità_prof
Possibilmente non noto alla nascita
0..*
Professore
+nome : str
+cod_fiscale : CodiceFiscale {id}
+data_nascita : Data
0..*
luogo_nascita_stud
Certamente non noto alla nascita
0..*
Studente
+nome : str
+cod_fiscale : CodiceFiscale {id}
+data_nascita : Data
+n_matricola : str <<imm>>
corsi_superati
0..*
0..*
corsi_insegnati
Possibilmente non noto alla nascita
iscriz_facoltà
iscriz_facoltà
+anno_iscrizione : IntGE1088
0..*
1..*
Corso
+nome : str
+n_ore_lezione : IntGEZ
+codice : str <<imm>>
Facoltà
+nome : str
+tipo_facoltà : str
Certamente non noto alla nascita
0..*
0..*
1..*
app_facoltà_corso
Powered By Visual Paradigm Community Edition

Custom Types

```
from typing import Self
import re


class IntGEZ(int):
    # Tipo di dato Intero >= 0

    def __new__(cls, v: int | float | str | bool | Self) -> Self:
        # Invoco il metodo new della superclasse, che è 'integer'
        n: int = super().__new__(cls, v)

        if n >= 0:
            return n

        raise ValueError(f"Il numero inserito {v} è inferiore a zero o non è intero!")


class IntGE1088(int):
    # Tipo di dato Intero >= 1088

    def __new__(cls, v: int | float | str | bool | Self) -> Self:
        # Invoco il metodo new della superclasse, che è 'integer'
        n: int = super().__new__(cls, v)

        if n >= 1088:
            return n

        raise ValueError(f"Il numero inserito {v} non è superiore a '1088' o intero!")


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)
```

Regione

```
from typing import Any

class Regione:

    _nome: str # noto alla nascita
```

```
    def __init__(self, nome: str) -> None:
        self.set_nome(nome)

    def nome(self) -> str:
        return self._nome

    def set_nome(self, n: str) -> None:
        self._nome: str = n

    def __hash__(self) -> int:
        return hash(self.nome())

    def __eq__(self, other: Any) -> bool:
        if other is None or not isinstance(other, type(self)) or hash(self) != hash(other):
            return False
        return (self.nome()) == (other.nome())

    def __repr__(self) -> str:
        return f"Regione(nome={self._nome})"
```

Città

```
from typing import Any


class Città:

    _nome: str # noto alla nascita

    def __init__(self, nome: str) -> None:
        self.set_nome(nome)

    def nome(self) -> str:
        return self._nome

    def set_nome(self, n: str) -> None:
        self._nome: str = n

    def __hash__(self) -> int:
        return hash(self.nome())

    def __eq__(self, other: Any) -> bool:
        if other is None or not isinstance(other, type(self)) or hash(self) != hash(other):
            return False
        return (self.nome()) == (other.nome())
```

```
    def __repr__(self) -> str:
        return f"Città(nome={self._nome})"
```

Professore

```
from custom_types import CodiceFiscale
from typing import Any
import datetime


class Professore:

    _nome: str # noto alla nascita
    _cf: CodiceFiscale # noto alla nascita
    _nascita: datetime.date # noto alla nascita

    def __init__(self, nome: str, cf: CodiceFiscale, nascita: datetime.date) -> None:
        self.set_nome(nome)
        self.set_cf(cf)
        self.set_nascita(nascita)

    def nome(self) -> str:
        return self._nome

    def set_nome(self, n: str) -> None:
        self._nome: str = n

    def cf(self) -> CodiceFiscale:
        return self._cf

    def set_cf(self, cf: CodiceFiscale) -> None:
        self._cf: CodiceFiscale = cf

    def nascita(self) -> datetime.date:
        return self._nascita

    def set_nascita(self, n: datetime.date) -> None:
        self._nascita: datetime.date = n

    def __hash__(self) -> int:
        return hash((self.nome(), self.cf(), self.nascita()))

    def __eq__(self, other: Any) -> bool:
        if other is None or not isinstance(other, type(self)) or hash(self) != hash(other):
            return False
        return (self.nome(), self.cf(), self.nascita()) == (other.nome(), other.cf(), other.nascita())
```

```
    def __repr__(self) -> str:
        return f"Professore(nome={self._nome})"
```

Studente

```
from custom_types import CodiceFiscale
from typing import Any
import datetime


class Studente:

    _nome: str # noto alla nascita
    _cf: CodiceFiscale # noto alla nascita
    _nascita: datetime.date # noto alla nascita
    _n_matricola: str # immutabile, noto alla nascita

    def __init__(self, nome: str, cf: CodiceFiscale, nascita: datetime.date, n_matricola: str) ->
None:
        self.set_nome(nome)
        self.set_cf(cf)
        self.set_nascita(nascita)
        self._n_matricola = n_matricola

    def nome(self) -> str:
        return self._nome

    def set_nome(self, n: str) -> None:
        self._nome: str = n

    def cf(self) -> CodiceFiscale:
        return self._cf

    def set_cf(self, cf: CodiceFiscale) -> None:
        self._cf: CodiceFiscale = cf

    def nascita(self) -> datetime.date:
        return self._nascita

    def set_nascita(self, n: datetime.date) -> None:
        self._nascita: datetime.date = n

    def n_matricola(self) -> str:
        return frozenset(self._n_matricola)
```

```
    def __hash__(self) -> int:
        return hash((self.nome(), self.cf(), self.nascita(), self.n_matricola()))

    def __eq__(self, other: Any) -> bool:
        if other is None or not isinstance(other, type(self)) or hash(self) != hash(other):
            return False
        return (self.nome(), self.cf(), self.nascita(), self.n_matricola()) == (other.nome(),
other.cf(), other.nascita(), other.n_matricola())

    def __repr__(self) -> str:
        return f"Studente(nome={self._nome})"
```

Iscrizione Facoltà

```
from custom_types import IntGE1088

class iscriz_facoltà:

    _anno_iscrizione: IntGE1088 # noto alla nascita

    def __init__(self, anno_iscrizione: IntGE1088) -> None:
        self.set_anno_iscrizione(anno_iscrizione)

    def anno_iscrizione(self) -> IntGE1088:
        return self._anno_iscrizione

    def set_anno_iscrizione(self, ann: IntGE1088) -> None:
        self._anno_iscrizione: IntGE1088 = ann
```

Facoltà

```
from typing import Any

class Facoltà:

    _nome: str # noto alla nascita
    _tipo_facoltà: str # noto alla nascita

    def __init__(self, nome: str, tipo_facoltà: str) -> None:
        self.set_nome(nome)
        self.set_tipo_facoltà(tipo_facoltà)
```

```
    def nome(self) -> str:
        return self._nome

    def set_nome(self, n: str) -> None:
        self._nome: str = n

    def tipo_facoltà(self) -> str:
        return self._tipo_facoltà

    def set_tipo_facoltà(self, tf: str) -> None:
        self._tipo_facoltà: str = tf

    def __hash__(self) -> int:
        return hash((self.nome(), self.tipo_facoltà()))

    def __eq__(self, other: Any) -> bool:
        if other is None or not isinstance(other, type(self)) or hash(self) != hash(other):
            return False
        return (self.nome(), self.tipo_facoltà()) == (other.nome(), other.tipo_facoltà())

    def __repr__(self) -> str:
        return f"Facoltà(nome={self._nome})"
```

Corso

```
from custom_types import IntGEZ
from typing import Any

class Corso:

    _nome: str # noto alla nascita
    _n_ore_lezione: IntGEZ # noto alla nascita
    _codice: str # immutabile, noto alla nascita

    def __init__(self, nome: str, n_ore_lezione: IntGEZ, codice: str) -> None:
        self.set_nome(nome)
        self.set_n_ore_lezione(n_ore_lezione)
        self._codice = codice

    def nome(self) -> str:
        return self._nome

    def set_nome(self, n: str) -> None:
```

```
        self._nome: str = n

    def n_ore_lezione(self) -> IntGEZ:
        return self._n_ore_lezione

    def set_n_ore_lezione(self, nol: IntGEZ) -> None:
        self._n_ore_lezione: IntGEZ = nol

    def codice(self) -> str:
        return frozenset(self._codice)

    def __hash__(self) -> int:
        return hash((self.nome(), self.n_ore_lezione(), self.codice()))

    def __eq__(self, other: Any) -> bool:
        if other is None or not isinstance(other, type(self)) or hash(self) != hash(other):
            return False
        return (self.nome(), self.n_ore_lezione(), self.codice()) == (other.nome(),
other.n_ore_lezione(), other.codice())

    def __repr__(self) -> str:
        return f"Corso(nome={self._nome})"
```