Lösung 1 - Delegation anwenden

circle_cuboid.py

```
class Point:
    """
    Die ist ein Punkt auf den Koordinaten x, y
    """

    def __init__(self, x: float, y: float):
        """
        Erzeugt einen Punkt auf den Koordinaten x und y
        :param x: Horizontale Komponente
        :param y: Vertikale Komponente
        """
        self.__x = x
        self.__y = y

    @property
    def x(self) -> float:
        return self.__x

    @x.setter
    def x(self, x: float):
        self.__x = x

    @property
    def y(self) -> float:
        return self.__y

    @y.setter
    def y(self, y: float):
        self.__y = y

    def __str__(self) -> str:
        return f"({round(self.x)}|{round(self.y)})"


class Cuboid:
    """
    Das ist ein Quadrat mit der Seitenlänge 2*u
    """
    def __init__(self, u):
        """
        Erzeugt ein Quadrat mit 4 Eckpunkten und der Seitenlänge 2*u
        :param u: entspricht demnach einer halben Seitenlänge
        """
        self.points = [
            Point(u, u),
            Point(-u, u),
```

```
            Point(u, -u),
            Point(-u, -u)
        ]

    def scale(self, f: float) -> None:
        """
        Skaliert die Kompenenten der Eckpunkte um den Faktor f
        :param f: Skalierungsfaktor
        """
        for p in self.points:
            p.x = p.x * f
            p.y = p.y * f

    @property
    def area(self) -> float:
        """
        Berechnet die Fläche und gibt sie retour.
        :return: Fläche
        """
        p = self.points[0]
        return round(2 * p.x * 2 * p.y)

    def __str__(self) -> str:
        points = ",".join([str(p) for p in self.points])
        return f"Fläche: {self.area}, Punkte: {points}"


class Circle:
    """
    Kreis mit Radius u
    """

    def __init__(self, u: float):
        """
        Erzeugt ein Kreis mit dem Radius u.
        :param u: Radius
        """
        self.__radius = u

    @property
    def radius(self) -> float:
        return round(self.__radius)

    def scale(self, f: float) -> None:
        """
        Skaliert den Radius um den Faktor f
        :param f: Skalierungsfaktor
        """
        self.__radius *= f
```

```
    @property
    def area(self) -> float:
        """
        Berechnet die Fläche und gibt sie retour.
        :return: Fläche
        """
        r = self.__radius
        PI = 3.142
        return round(PI * r * r)

    def __str__(self) -> str:
        return f"Fläche: {self.area}, Radius: {self.radius}"


class CircleCuboid():
    """
    Quadrat mit Inkreis und der Seitenlänge 2*u
    """
    def __init__(self, u: float):
        """
        Erzeugt ein Quadrat mit Inkreis
        :param u: Radius des Inkreises
        """
        self.__circle_ref = Circle(u)
        self.__cuboid_ref = Cuboid(u)

    @property
    def circle(self): return self.__circle_ref

    @property
    def cuboid(self): return self.__cuboid_ref

    def scale(self, f):
        """
        Skaliert die Komponenten um den Faktor f
        :param f: Skalierungsfaktor
        """
        self.__circle_ref.scale(f)
        self.__cuboid_ref.scale(f)

    @property
    def area_difference(self):
        """
        Berechnet die Flächendifferenz der beiden Komponenten und gibt
sie retour.
        :return: Flächendifferenz
        """
        return round(self.__cuboid_ref.area - self.__circle_ref.area)

    def __str__(self):
```

```
        return f"Kreiseck: " + \
               f"\n\tKreis -> {self.circle}" + \
               f"\n\tEck ---> {self.cuboid}" + \
               f"\n\tDie Flächendifferenz ist {self.area_difference}"


if __name__ == '__main__':
    print("Darstellung der Klasse 'Punkt' anhand von zwei Beispielen:")
    p1 = Point(1, 2)
    p2 = Point(3, 4)
    print(p1, p2)

    print("\nKreis-Eck Demo:")
    cc = CircleCuboid(3)
    print(cc)

    factor = 20
    cc.scale(factor)
    print(f"\nEs wurde um den Faktor {factor} skaliert")
    print(cc)
```