Generalisierung/Spezialisierung

Die Begriffe Generalisierung und Spezialisierung beziehen sich auf zwei zentrale Konzepte der objektorientierten Programmierung und Datenmodellierung.

Generalisierung ist der Prozess, bei dem gemeinsame Eigenschaften (Attribute) und Funktionen (Methoden) in einer allgemeineren Klasse (auch als Superklasse bezeichnet) gebündelt werden. Diese Superklasse enthält die Attribute und Methoden, die allen Entitäten gemeinsam sind. Dies ermöglicht eine effiziente Wiederverwendung von Code und eine klare Strukturierung des Programms.

Spezialisierung hingegen ist der Prozess, bei dem spezifischere Klassen (auch als Subklassen bezeichnet) erstellt werden, die von der Superklasse erben. Diese Subklassen enthalten spezifische Attribute und Methoden, die nur für sie zutreffen, und erben die Eigenschaften der Superklasse. Dies ermöglicht eine genaue Modellierung von spezifischen Verhaltensweisen und Eigenschaften.

Zusammenfassung: die **Generalisierung** dient dazu, gemeinsame Merkmale in einer Superklasse zu bündeln die **Spezialisierung** dient dazu, spezifische Merkmale in Subklassen zu definieren.

Beide Konzepte sind eng miteinander verbunden und bilden die Grundlage für die Vererbung in der objektorientierten Programmierung.

Beispiel: Eine Superklasse `Fahrzeug` mit Eigenschaften wie `Geschwindigkeit` und `Farbe` und Methoden wie `beschleunigen` und `bremsen`. Subklassen wie `Auto` und `Motorrad`, die zusätzliche Eigenschaften und Methoden haben, aber auch die von `Fahrzeug` erben.

In [ ]:
```python
# Superklasse
class Fahrzeug:
    def __init__(self, geschwindigkeit, farbe):
        self.geschwindigkeit = geschwindigkeit
        self.farbe = farbe

    def beschleunigen(self):
        pass  # Diese Methode wird in den Subklassen überschrieben

    def bremsen(self):
        pass  # Diese Methode wird in den Subklassen überschrieben

# Subklasse
class Auto(Fahrzeug):
    def __init__(self, geschwindigkeit, farbe, marke):
        super().__init__(geschwindigkeit, farbe)
        self.marke = marke

    def hupen(self):
        return "Das Auto hupt!"

# Subklasse
```

```
class Motorrad(Fahrzeug):
    def __init__(self, geschwindigkeit, farbe, typ):
        super().__init__(geschwindigkeit, farbe)
        self.typ = typ

    def wheelie_machen(self):
        return "Das Motorrad macht einen Wheelie!"
```

Die Superklasse, auch als Basisklasse oder Elternklasse bezeichnet, definiert gemeinsame Attribute und Methoden, die von den erbenden Klassen, den sogenannten Subklassen oder Kindklassen, genutzt werden können.

Die Vererbung ermöglicht es, dass eine Subklasse die Attribute und Methoden einer Superklasse erbt und zusätzlich eigene spezifische Attribute und Methoden hinzufügt oder die geerbten überschreibt. Dies fördert die Wiederverwendbarkeit des Codes und die logische Strukturierung des Programms.

Nehmen wir wieder ein Beispiel aus WoW. Wir könnten eine Superklasse `Charakter` erstellen, die allgemeine Attribute und Methoden für alle Charaktere im Spiel definiert. Dann könnten wir spezifischere Subklassen wie `Krieger` und `Magier` erstellen, die von `Charakter` erben.

In [ ]:
```
class Charakter:
    def __init__(self, name, level):
        self.name = name
        self.level = level

    def angreifen(self):
        pass  # Diese Methode wird in den Subklassen überschrieben

class Krieger(Charakter):
    def __init__(self, name, level, waffe):
        super().__init__(name, level)
        self.waffe = waffe

    def angreifen(self):
        return f"{self.name} greift mit {self.waffe} an!"

class Magier(Charakter):
    def __init__(self, name, level, zauber):
        super().__init__(name, level)
        self.zauber = zauber

    def angreifen(self):
        return f"{self.name} wirkt {self.zauber}!"
```

In diesem Beispiel erben sowohl `Krieger` als auch `Magier` von der Superklasse `Charakter`. Sie erben die Attribute `name` und `level` und fügen ihre eigenen spezifischen Attribute `waffe` bzw. `zauber` hinzu. Sie überschreiben auch die Methode `angreifen`, um spezifisches Verhalten zu implementieren. Die Methode `super().__init__(name, level)` wird verwendet, um die Initialisierungsmethode der Superklasse aufzurufen und die geerbten Attribute zu initialisieren.

isinstance/issubclass

In Python können Sie die eingebauten Funktionen `isinstance()` und `issubclass()` verwenden, um zu überprüfen, ob ein Objekt eine Instanz einer bestimmten Klasse ist oder ob eine Klasse eine Unterklasse einer anderen Klasse ist.

Die Funktion `isinstance(objekt, klasse)` prüft, ob ein bestimmtes Objekt eine Instanz einer bestimmten Klasse oder einer ihrer Subklassen ist.

In [ ]:
```
krieger = Krieger("Cônan", 10, "Schwert")
magier = Magier("Gandálf", 20, "Feuerball")

print(isinstance(krieger, Krieger))  # Gibt True aus
print(isinstance(krieger, Charakter))  # Gibt True aus
print(isinstance(magier, Magier))  # Gibt True aus
print(isinstance(magier, Charakter))  # Gibt True aus
```

Die Funktion `issubclass(klasse1, klasse2)` prüft, ob `klasse1` eine Unterklasse von `klasse2` ist.

In [ ]:
```
print(issubclass(Krieger, Charakter))  # Gibt True aus
print(issubclass(Magier, Charakter))  # Gibt True aus
print(issubclass(Charakter, Krieger))  # Gibt False aus
```

In diesem Beispiel sind sowohl `Krieger` als auch `Magier` Subklassen der Klasse `Charakter`, daher gibt `issubclass(Krieger, Charakter)` und `issubclass(Magier, Charakter)` True aus. Aber `Charakter` ist keine Subklasse von `Krieger` oder `Magier`, daher gibt `issubclass(Charakter, Krieger)` False aus.

Übungen:

Pseudocode:

1. Erstellen Sie eine Superklasse `Tier` mit den Attributen `name` und `alter` und einer Methode `lautgeben`.
2. Erstellen Sie eine Subklasse `Hund` und eine Subklasse `Katze`, die beide von `Tier` erben.
3. Fügen Sie der Klasse `Hund` das Attribut `rasse` und der Klasse `Katze` das Attribut `fellfarbe` hinzu.
4. Überschreiben Sie die Methode `lautgeben` in beiden Subklassen, so dass ein Hund "Wuff!" und eine Katze "Miau!" ausgibt.

Python:

```
# Übung 1
class Tier:
    def __init__(self, name, alter):
        self.name = name
        self.alter = alter

    def lautgeben(self):
        pass
```

```python
class Hund(Tier):
    def __init__(self, name, alter, rasse):
        super().__init__(name, alter)
        self.rasse = rasse

    def lautgeben(self):
        return "Wuff!"

class Katze(Tier):
    def __init__(self, name, alter, fellfarbe):
        super().__init__(name, alter)
        self.fellfarbe = fellfarbe

    def lautgeben(self):
        return "Miau!"
```

Aufgaben:

1. Erzeugen Sie Instanzen von `Hund` und `Katze` und rufen Sie ihre Methoden auf.
2. Experimentieren Sie mit verschiedenen Attributen und Methoden in den Klassen.
3. Überlegen Sie, wie Sie die Klassen weiter spezialisieren könnten (z.B. könnten Sie weitere Subklassen wie `Dackel` oder `Perserkatze` erstellen).
4. Überlegen Sie, wie Sie die Klassen weiter generalisieren könnten (z.B. könnten Sie eine Superklasse `Säugetier` erstellen, von der `Tier` erbt).
5. Erzeugen Sie Instanzen von `Hund` und `Katze` und verwenden Sie `isinstance()`, um zu überprüfen, ob diese Instanzen tatsächlich Instanzen der Klassen `Hund`, `Katze` und `Tier` sind.
6. Verwenden Sie `issubclass()`, um zu überprüfen, ob `Hund` und `Katze` tatsächlich Subklassen von `Tier` sind.
7. Erstellen Sie eine neue Subklasse von `Tier`, z.B. `Vogel`, mit einem zusätzlichen Attribut `flügelspannweite`. Erzeugen Sie dann eine Instanz von `Vogel` und verwenden Sie `isinstance()` und `issubclass()`, um zu überprüfen, ob Ihre neue Klasse und Instanz korrekt funktionieren.