Projektweiterführung – Nutzung von Interfaces, Adapterklassen und Default-Implementierungen

Einleitung

Ihr habt bereits ein Projekt erstellt, das eine grundlegende Klassenhierarchie abbildet. Nun möchten wir dieses Projekt erweitern, um **Interfaces**, **Adapterklassen** und **Default-Implementierungen** einzubeziehen. Ziel ist es, die Funktionalität eures Projekts zu verbessern und gleichzeitig eure Kenntnisse über OOP-Prinzipien wie Polymorphismus, Sichtbarkeiten und Vererbung zu vertiefen.

Aufgabenstellung

1. Erweiterung der Klassenhierarchie

Interface `Lebewesen`

- **Eigenschaften**:
  - `alter` (z. B. in Jahren, `int`).
- **Methoden**:
  - `bewegen()` – Abstrakte Methode, die das Bewegungsverhalten beschreibt.
  - `lebenserwartung()` – **Default-Methode**, die basierend auf der Klasse eine Standardausgabe über die Lebenserwartung gibt.

Abstrakte Klasse `Tier` (Adapterklasse)

- Implementiert das Interface `Lebewesen`.
- **Eigenschaften**:
  - `nahrung` (z. B. Fleisch, Pflanzen, `String`).
- **Methoden**:
  - `fressen()` – Gibt aus, was das Tier frisst.
  - Abstrakte Methode `lautGeben()` – Muss in Unterklassen implementiert werden.

Klasse `Pflanze`

- Implementiert das Interface `Lebewesen`.
- **Methoden**:
  - Überschreibt `bewegen()`, um das Wachstum der Pflanze anzuzeigen.
  - Verwendet die Default-Implementierung von `lebenserwartung()`.

---

2. Spezifische Klassen

Klasse `Mensch`

- **Eigenschaften**:
  - `sprache` (z. B. Deutsch, Englisch, `String`).

- **Methoden**:
  - Überschreibt `bewegen()` – Gibt aus, dass der Mensch läuft und spricht.

Klasse `Katze`

- Erbt von `Tier`.
- **Methoden**:
  - Implementiert `lautGeben()` – Gibt z. B. "miaut" aus.

Erweiterung: Klasse `Hund`

- Erbt von `Tier`.
- **Methoden**:
  - Implementiert `lautGeben()` – Gibt z. B. "bellt" aus.
  - Neue Methode `spielen()` – Gibt aus, dass der Hund spielt.

---

4. Sichtbarkeiten und Kapselung

1. Verändern Sie die Sichtbarkeit der Eigenschaften:
   - Verwenden Sie `protected` für Eigenschaften, die Unterklassen benötigen (z. B. `nahrung`).
   - Verwenden Sie `private` für Eigenschaften, die nur innerhalb einer Klasse verwendet werden sollen (z. B. `alter`).
2. Fügen Sie **Getter und Setter** hinzu, um den kontrollierten Zugriff auf die Eigenschaften zu ermöglichen.

---

5. Abgabeanforderungen

1. Eine funktionierende **Klassenhierarchie** mit:
   - Dem Interface `Lebewesen` mit Default-Methoden.
   - Einer Adapterklasse `Tier`.
   - Den konkreten Klassen Mensch, Katze, Hund und Pflanze.
2. Eine **klar strukturierte Package-Organisation**:
   - Beispiel:

```
src/
├── main/
│   ├── lebewesen/
│   │   ├── Lebewesen.java
│   │   ├── Tier.java
│   │   ├── Pflanze.java
│   ├── spezies/
│   │   ├── Mensch.java
│   │   ├── Katze.java
│   │   ├── Hund.java
│   ├── app/
│   │   ├── Main.java
```

3. Eine Hauptklasse `Main`, die:
    - Objekte erstellt und Methoden testet.
    - Polymorphismus demonstriert.

Erwartetes Ergebnis

Am Ende dieser Aufgabe sollte das Programm:

1. Eine funktionierende Klassenhierarchie mit allen oben genannten Anforderungen implementieren.
2. Eine strukturierte Package-Organisation enthalten.
3. Ein UML-Diagramm und eine Dokumentation der Hierarchie enthalten.
4. Eine gezippte Datei mit allen Quellcodes und der Dokumentation als Abgabe bereitstellen.
