Software Design

Julian Huber & Matthias Panny

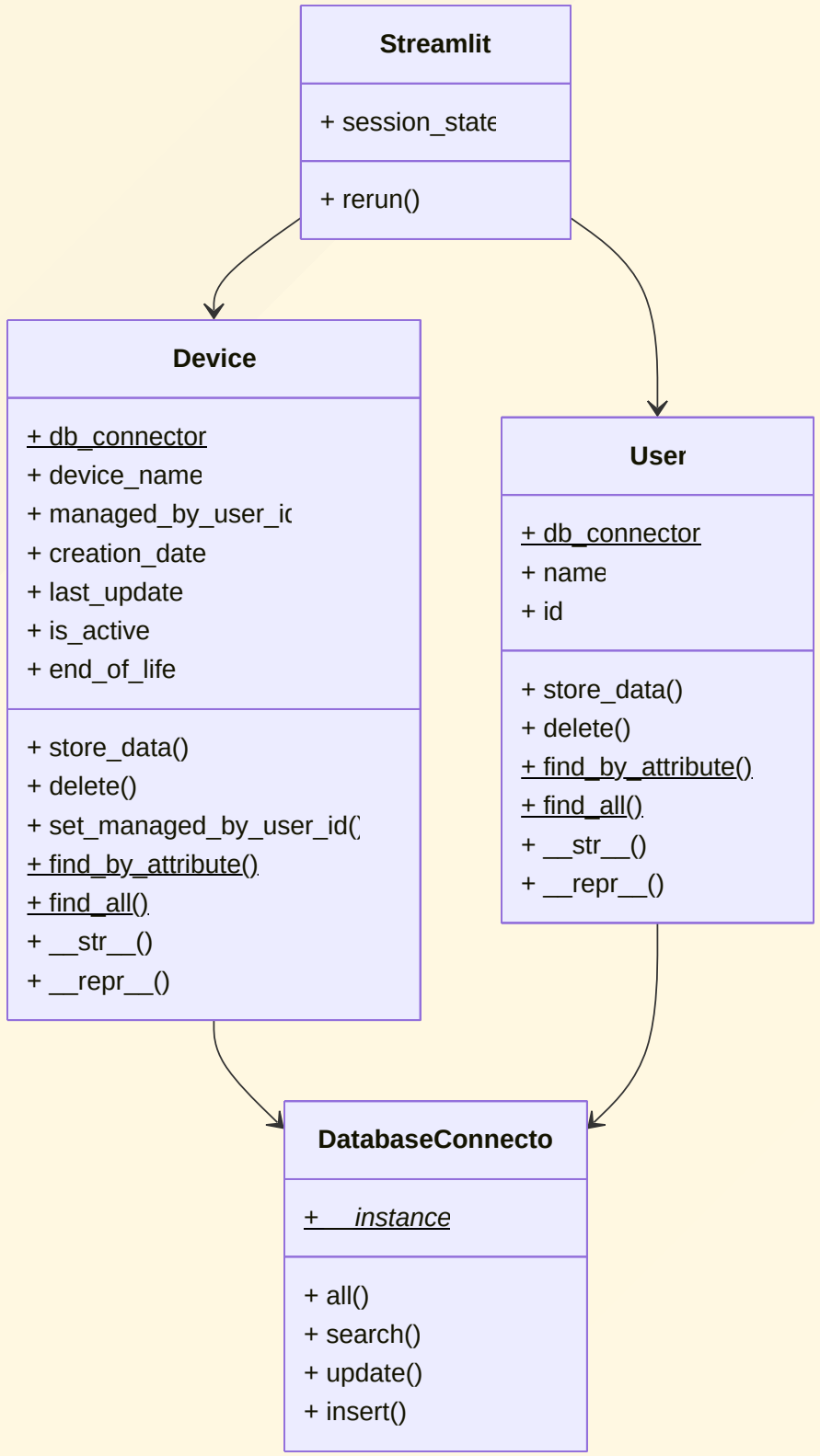
Anwendung von Vererbung

Design-Patterns

- Design-Patterns arbeiten faktisch immer mit Objektorientierung
- Wir wollen nun nochmals mit den Prinzipien aus Kapitel
  - 01_05_Objektorientierung_I und
  - 01_06_Objektorientierung_II arbeiten

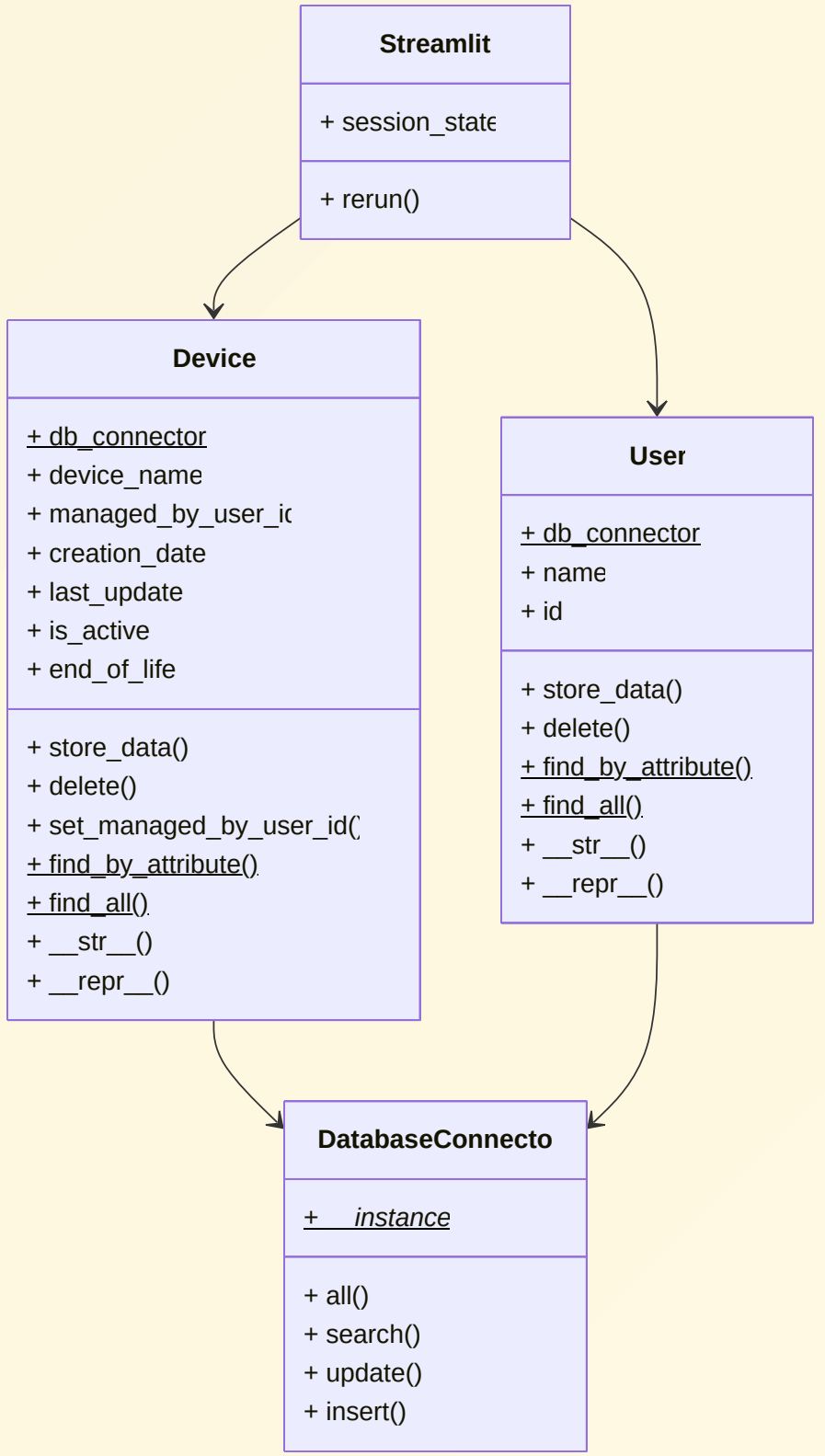
Lösungsvorschlag aus dem letzten Foliensatz

- Wir müssen sowohl Objekte vom Typ `Device` als auch vom Typ `User` verwalten und in der Datenbank speichern
- Für Reservierungs- & Wartungsverwaltung müssen auch Daten in der Datenbank gespeichert werden



Lösungsvorschlag aus dem letzten Foliensatz

- Können wir uns das Prozedere des Speicherns & Ladens von Objekten vereinfachen?
- Welche Attribute benötigen/verwenden wir für die Speicherung?



🎨 Aufgabe

- Bestimmen Sie die Attribute & Methoden, die ausschließlich für die Speicherung der Objekte benötigt werden
- Welche davon scheinen in mehreren Klassen auf?
- Können Attribute auch mit dem Konzept des Singleton aus der letzten Einheit in Verbindung gebracht werden?

✍️ Lösung

- Es werden folgende Attribute für die Speicherung benötigt:
  - Attribute:
    - Eine `id` → z.B.: `user_id` oder `device_name`, etc.
    - Ein `db_connector` → Zugang zur Datenbank
  - Methoden:
    - `store_data()`
    - `find_by_attribute()`
    - `find_all()`
    - `delete()`
- Alle davon scheinen sowohl in `User` als auch in `Device` auf
- Nicht jede Klasse braucht einen eigenen `db_connector` → Singleton aus der letzen LV-Einheit ist hier sinnvoll

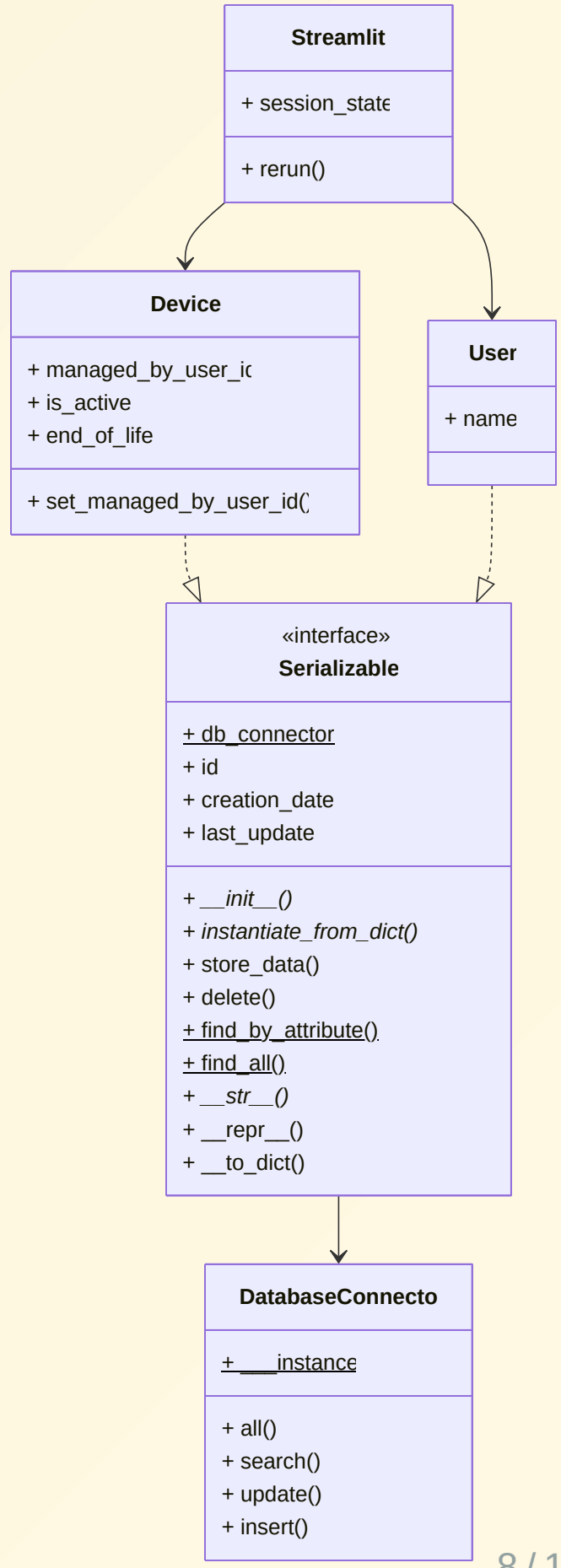
Handlungsbedarf?

- Sind diese mehrfachen Implementierungen sinnvoll?
- Was könnte hier nun getan werden?

Lösungsvorschlag

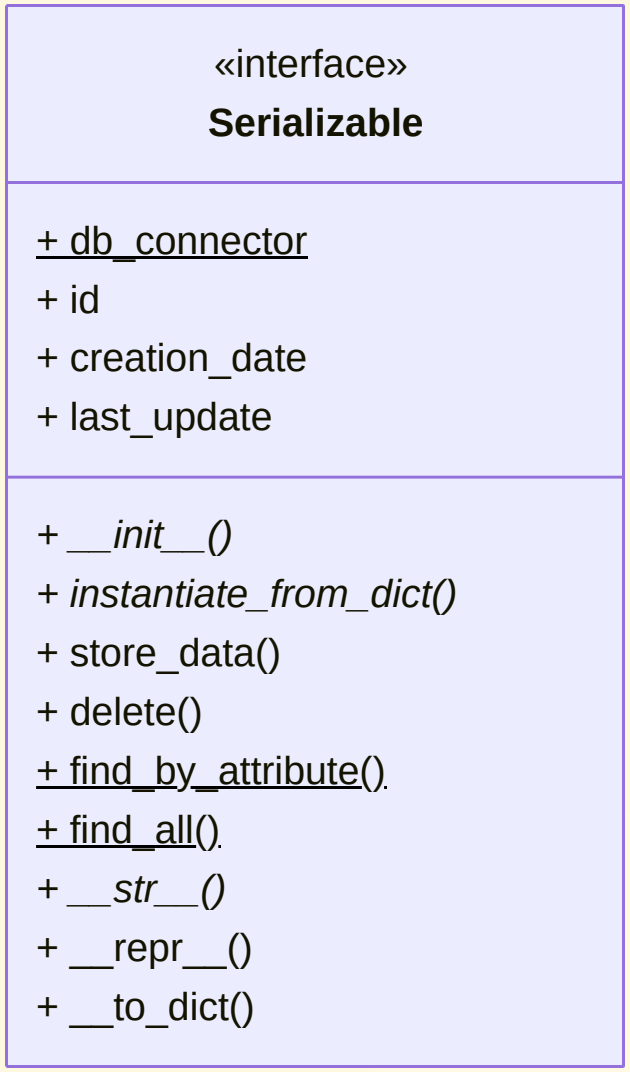
Vererbung oder Interface (ABC)

- Vererbung:
  gemeinsame Attribute und Methoden werden in eine *Super-Klasse* ausgelagert. Methoden, die voneinander abweichen werden überschrieben
  (z.B. `find_all()` und `db_connector` wegen unterschiedlicher `table`, die die Daten speichert)
  → es entsteht ein Objekt `Serializable`
- Abstract Base Class (ABC):
  Die Klasse `Serializable` ist hilfreich, aber soll nie direkt verwendet werden → eine Abstract Base Class
  🔁 Was bedeutet *abstract* in diesem Kontext?



`Serializable`-Klasse

- Es soll die Klasse `Serializable` erstellt werden die als ABC das gemeinsame Interface für alle Objekte die in der Datenbank gespeichert werden sollen darstellt
- Abstrakte Methoden, die angepasst werden müssen sind *kursiv*



Attribute & Methoden

- Welche Attribute und Methoden sollen in der Klasse `Serializable` implementiert werden?
- Welche davon waren bereits in den Implementierungen in `User` und `Device` vorhanden?
- Welche davon sind spezifisch für die Klasse `Serializable`?

✍️ Aufgabe

- Wir wollen nun unsere Klasse `Serializable` implementieren und in unsere Projekt integrieren
- Wie könnte ein sinnvoller Workflow für diesen Prozess aussehen, wenn wir noch nicht genau wissen wie die Implementierung aussehen soll?

Ausgangslage - im Ordner `Vererbung/Ausgangslage`

- `database.py` → ident zur Version aus der letzten Einheit
- `devices_start.py`
- `users_start.py`
- `serializable_start.py`

✍️ Aufgabe

- Wir wollen nun Schritt für Schritt die Attribute und Methoden aus `User` und `Device` in die Klasse `Serializable` auslagern

Musterlösung - im Ordner `Vererbung/Final`

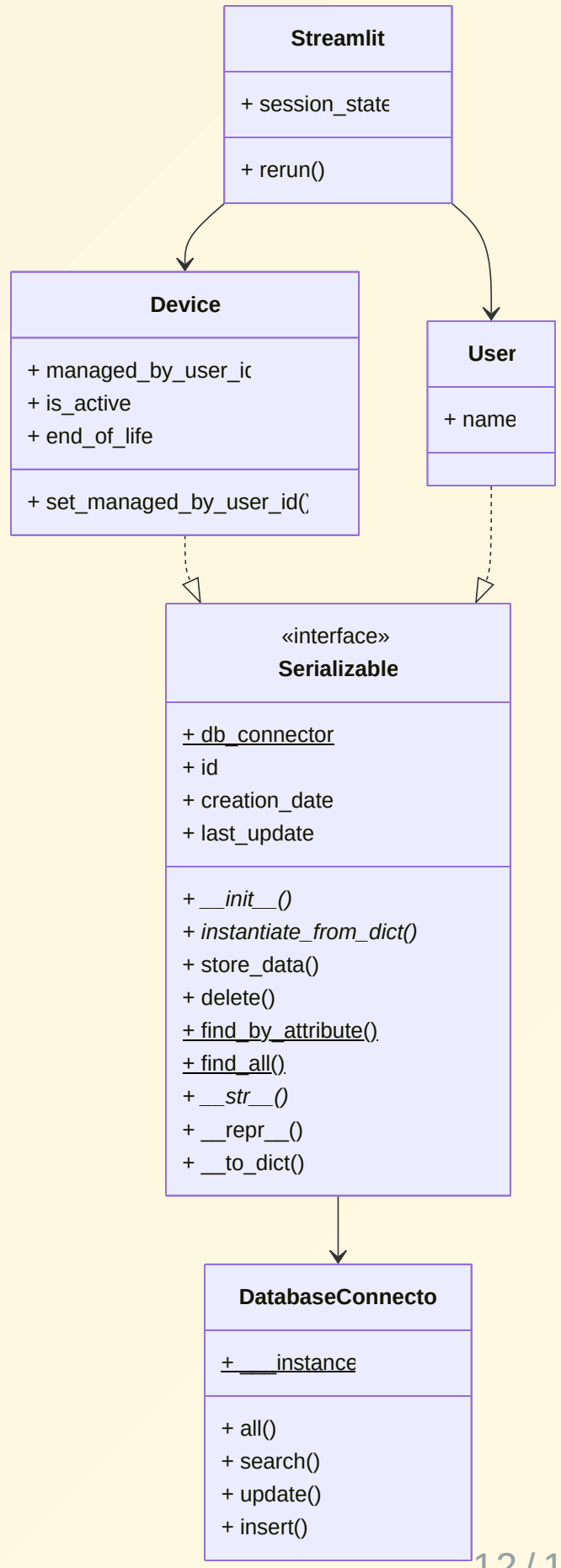
- `database.py` → ident zur Version aus der letzten Einheit
- `devices_inheritance.py`
- `users_inheritance.py`
- `serializable.py`

Lösungsvorschlag mit Vererbung

- Welche Eigenschaften hat diese Lösung?
- Welche Arbeitsschritte sind notwendig um eine Klasse `Reservation` zu implementieren?

`Reservation`-Klasse

- Muss in der Datenbank gespeichert werden
- Geräte speichern welche Reservierung zu ihnen gehört



✍️ Aufgabe

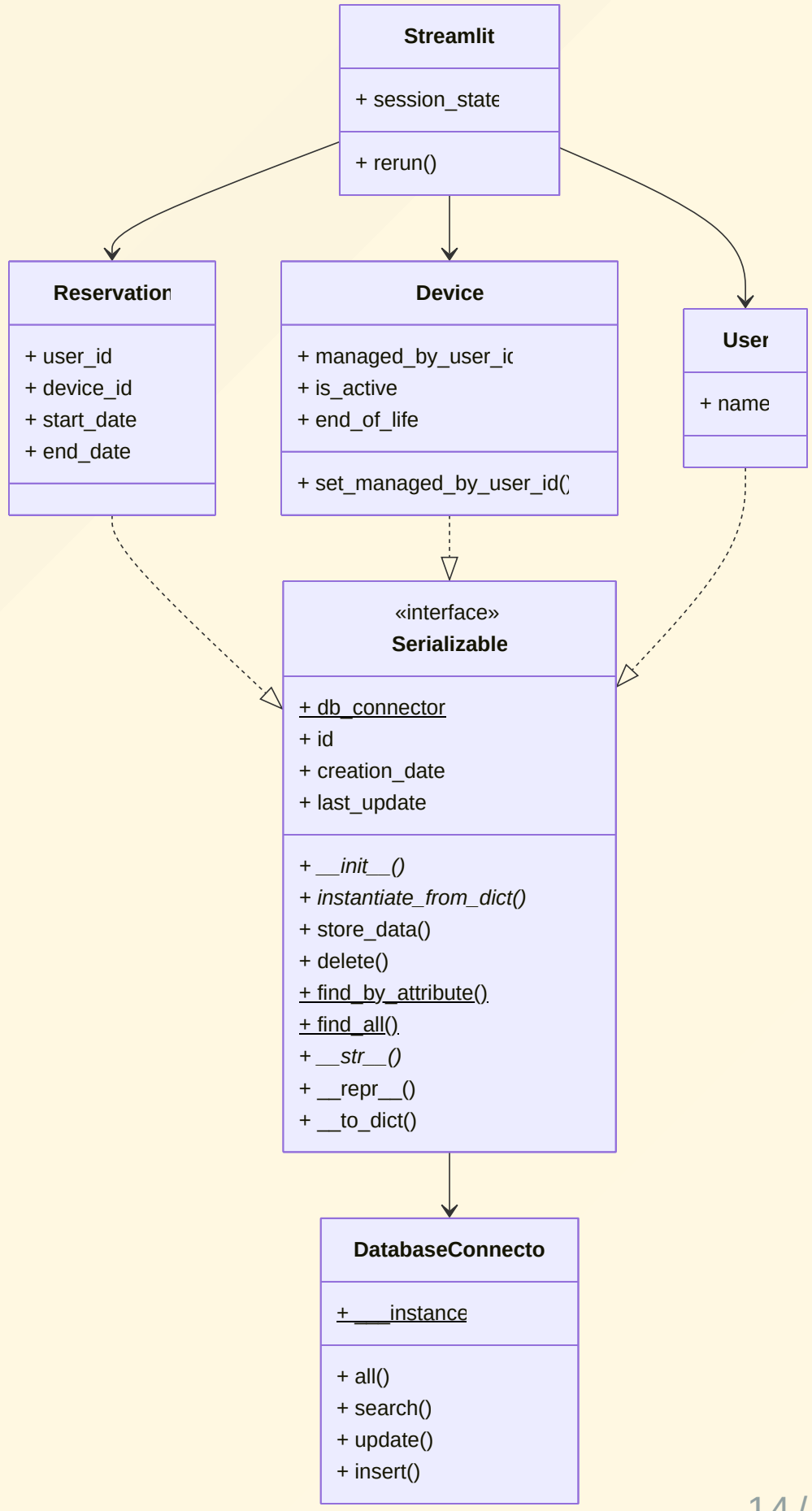
- Wir wollen nun die zusätzliche `Reservation`-Klasse implementieren

Musterlösung - im Ordner `Vererbung/Final`

- `database.py` → ident zur Version aus der letzten Einheit
- `devices_inheritance.py`
- `users_inheritance.py`
- `serializable.py`
- `reservations.py`

Lösungsvorschlag mit `Reservation`

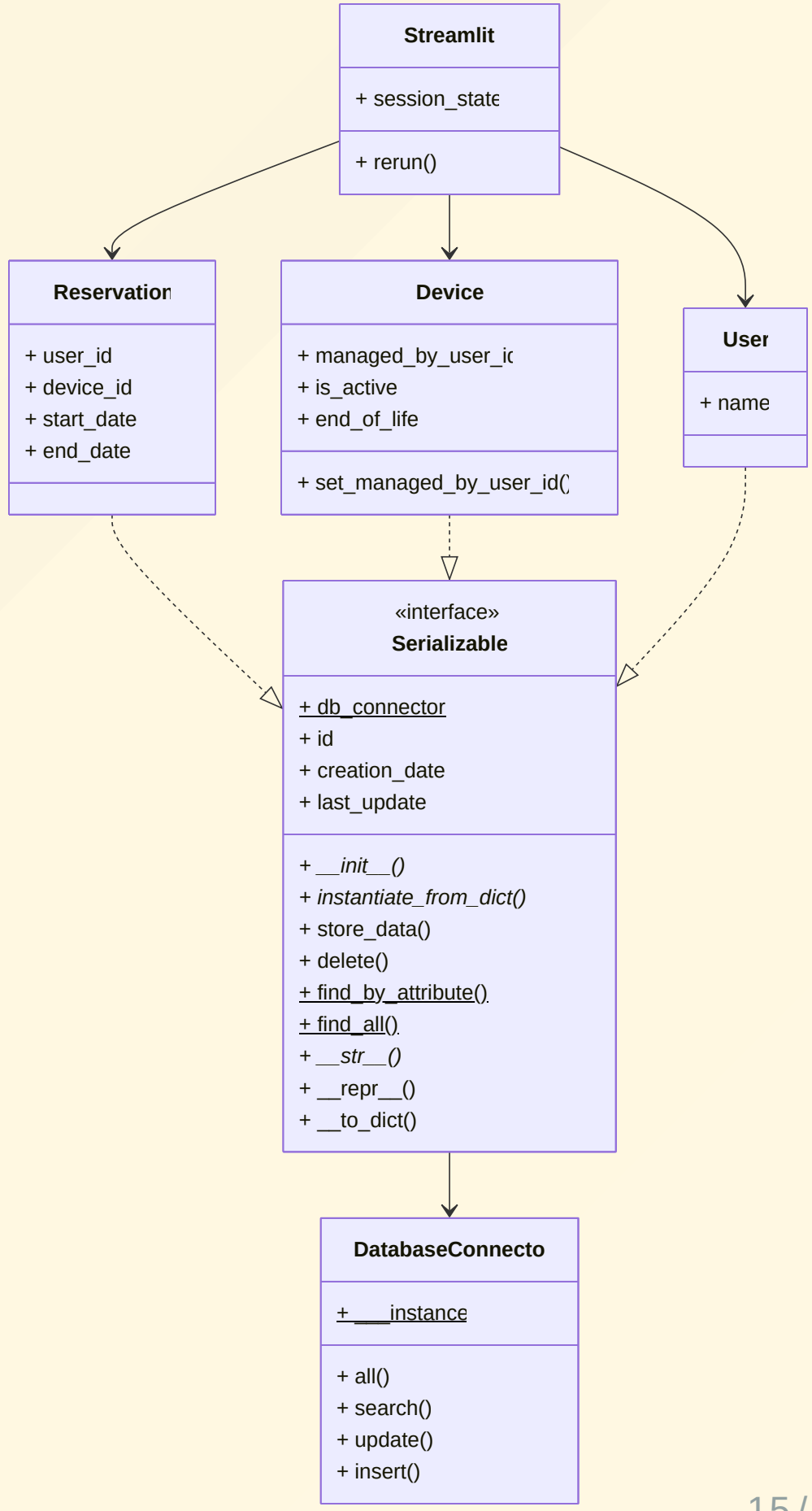
- Der Lösungsvorschlag lässt `Reservation` erneut von `Seralizable` erben → kann direkt in Datenbank gespeichert werden
- `DatabaseConnector` enthält eine neue Tabelle
- `Reservation` speichert die `device_id` und `user_id`
- Wir benötigen eine Methode um alle Reservierungen zu einem Gerät zu finden (und nicht nur einen Teil davon `find_by_attribute()`)



Lösungsvorschlag mit `Reservation`

Diskussion des Ansatzes

- Wenig zusätzlicher Aufwand für die Implementierung → Erweiterbarkeit, da wir schon den `Serializable`-Ansatz gewählt haben
- Leicht zu warten, da es keine Interaktionen zwischen den Klassen gibt außer, dass wird die IDs speichern



Lösungsvorschlag mit `Reservation`

Erweitern des Lösungsvorschlags

- Beim Erstellen von Reservierungen müssen wir sicherstellen, dass das Gerät und der User existieren und das Gerät noch nicht reserviert ist. Hierzu können wir eine neue Klasse `ReservationService` erstellen, die diese Aufgaben übernimmt

Schritte zur Implementierung

- Um unsere gesamte Case Study zu implementieren müssen wir auch noch das Wartungsmanagement implementieren
- Wie könnten die hierfür notwendigen Schritte aussehen?
- Welche neuen Klassen braucht es?
- Wovon müssen diese Klassen erben? → UML-Klassendiagramm
- Wie muss die Datenbank ergänzt werden?