Aufgabe 1 [Persistenztechniken für Objekte]

In der Vorlesung wurden die folgenden Persistenztechniken für Objekte vorgestellt:

- Objekt-relationale Mapping-Tools [ORMT]
- Bytecode-Postprozessor [BCP]
- Objekt-Orientierte Datenbanksysteme [OODBMS]
- Serialisierung [Ser]
- manuelles Objekt-relationales Mapping [ORM]
- direkte Mapper [dM]
- Container-Managed Persistence [CMP]
- Speicherung von Objekten/Beziehugen als Tupel in Tabellen [O>T]

a) Erzeugen Sie eine Hierarchie (Vererbungsbaum) dieser Techniken. Zur Vereinfachung können Sie die in eckigen Klammer angegebenen Abkürzungen verwenden.

b) Besonders erwähnt wurden die beiden Produkte **Java Data Objekts (JDO)** und **FastObjects**. Ordnen Sie diese der entsprechenden speziellsten Technologie (d.h. einem Blatt der oben entstandenen Hierarchie) zu.

c) Gegeben sei das objektorientierte Schema nach Anhang A der persistent zu haltenden Klassen einer objektorientierten Anwendung. Eine Gruppe von Technologien aus der Hierarchie aus a) ist aus Effizienzgründen nicht für diese Anwendung geeignet. Welche und warum nicht?

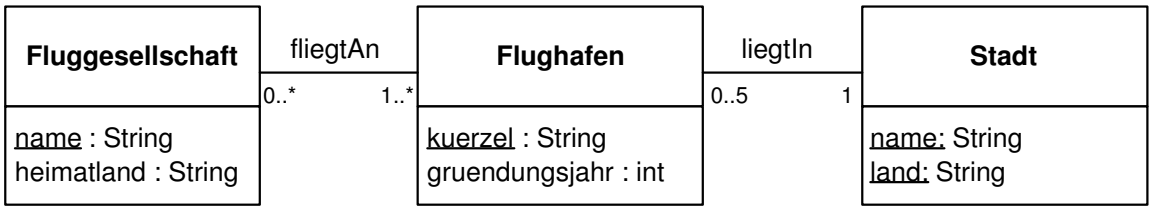
Aufgabe 2 [Objekt-relationales Mapping]

Eine häufig verwendete Technik zur persistenten Speicherung von Objekten ist das *objekt-relationale Mapping*. Hierbei wird die objektorientierte Struktur in Tabellen des relationalen Modells übertragen.

a) Nennen Sie das allgemeine Hauptproblem dieser Vorgehensweise.

b) Beschreiben Sie allgemein, wie Klassen, 1:n-Beziehungen und n:m-Beziehungen des objektorientierten Modells in Relationen übertragen werden und erzeugen Sie mit diesem Vorgehen die Relationen für das abgebildete UML-Klassendiagramm.

Anmerkungen: Die jeweiligen Schlüssel der Klassen sind unterstrichen. Unterstreichen Sie auch die Schlüssel ihrer Relationen. Die Datentypen (String, int etc.) brauchen nicht mitangegeben zu werden.



Anhang A: Objektorientiertes Schema

```
class Kontinent (extent AlleKontinente) {
  attribute string name;
  relationship SET(staat) staaten inverse Staat::kontinent;

}

class Staat (extent AlleStaaten) {
  attribute string name;
  attribute string regierungsform;
  attribute double flaeche;
  relationship Kontinent kontinent inverse Kontinent::staaten;
  relationship SET(land) laender inverse Land::staat;
  relationship Stadt hauptstadt;
}

class Land (extent AlleLaender) {
  attribute string name;
  attribute long einwohnerzahl;
  relationship Staat staat inverse Staat::laender;
  relationship set(Ort) orte inverse Ort::land;
  relationship Stadt hauptstadt;
}

class Ort (extent AlleOrte) {
  attribute string name;
  attibute string buergermeister;
  attribute long einwohnerzahl;
  relationship Land land inverse Land::orte;
  relationship set(Ortsteil) ortsteile inverse Ortsteil::ort;
}

class Dorf (extent AlleDoerfer) extends Ort;

class Stadt (extent AlleStaedte) extends Ort {
  attribute int stadtrechteSeit;
}

class Ortsteil (extent AlleOrtsteile) {
  attribute string name;
  relationship Ort ort inverse Ort::ortsteile;
  relationship set(Strasse) strassen inverse Strasse::ortsteile;
}
```

```
class Strasse (extent AlleStrassen) {
  attribute string name;
  attribute double laenge;
  relationship set(Ortsteil) ortsteile inverse Ortsteil:strassen;
  relationship set(Haus) haeuser inverser Haus::strasse;
}

class Haus (extent AlleHaeuser) {
  attribute integer nummer;
  attribute integer erbauungsjahr;
  relationship Strasse strasse inverse Strasse::haeuser;
}
```