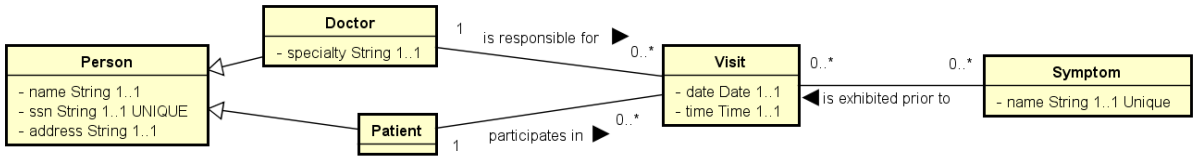
Suggested solutions SD1 - SD4 (lesson 1)

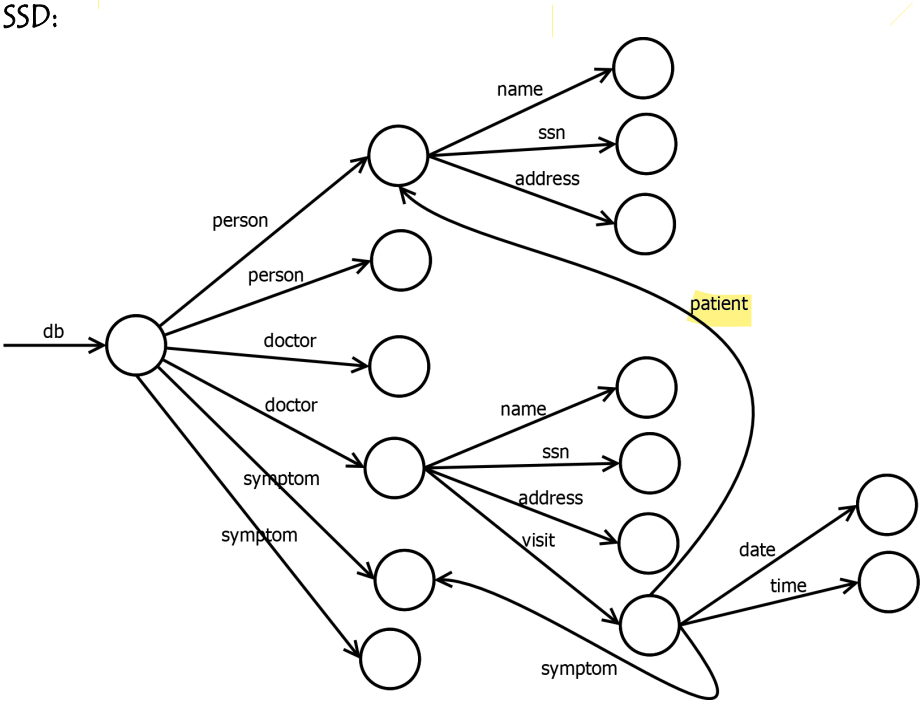
EXERCISE SD1

Create suitable structures in SSD, XML and JSON for the following conceptual model! Do not think about how a relational database would turn out! Think of the possibilities and limitations of semi-structured data structures!



We can see that Person and Symptom are the strong classes and that makes them candidates for appearing immediately under the root. A visit requires a doctor and a patient, so we can try to at least get one of those requirements automatically by nesting the visits in a suitable way. Here is a suggestion:

SSD:



This is just a fragment in order to show the structure. All the leaf nodes would of course have a value and since this is not a tree, node/object identifiers could be used (for example when referring to a patient or to a symptom). In textual form (as an SSD expression) it could look like this:

```
{db : {person : {name : "", ssn : "", address : ""},
       person : {name : "", ssn : "", address : ""},
       doctor : {name : "", ssn : "", address : "", specialty : "",
                  visit : {date : "", time : "", patient : "", symptom : "", symptom : ""},
                  visit : {date : "", time : "", patient : "", symptom : "", symptom : ""}},
       doctor : {name : "", ssn : "", address : "", specialty : ""},
       symptom : "",
       symptom : ""}
}
```

Here we have person and doctor, so any person that is a doctor gets the label doctor and has specialty and zero or more visits. Each visit still has a reference to its patient (that is a person or a doctor)

XML:

```
<DB>
    <Person id="" name="" ssn="" address="">
        <Doctor specialty="">
            <Visit date="" time="" patient="">
                <Symptom refid="" />
                <Symptom refid="" />
            </Visit>
            <Visit date="" time="" patient="">
                <Symptom refid="" />
                <Symptom refid="" />
            </Visit>
        </Doctor>
    </Person>
    <Person id="" name="" ssn="" address="">
        <Doctor specialty="">
            <Visit date="" time="" patient="">
                <Symptom refid="" />
                <Symptom refid="" />
            </Visit>
        </Doctor>
    </Person>
    <Person id="" name="" ssn="" address=""/>
    <Person id="" name="" ssn="" address=""/>
    <Symptom id="" name=""/>
    <Symptom id="" name=""/>
</DB>
```

Visit has a date time and patient field

Every person that is a doctor has a Doctor element
any visit to the doctor is inside the Doctor element.

<Visit date="" time="" patient="">
Visit refers to patient who is a different element.

Every person that is a doctor has a Doctor element and any visit with that doctor is inside the Doctor element. The visit still needs to refer to the patient, who is a (different) person.

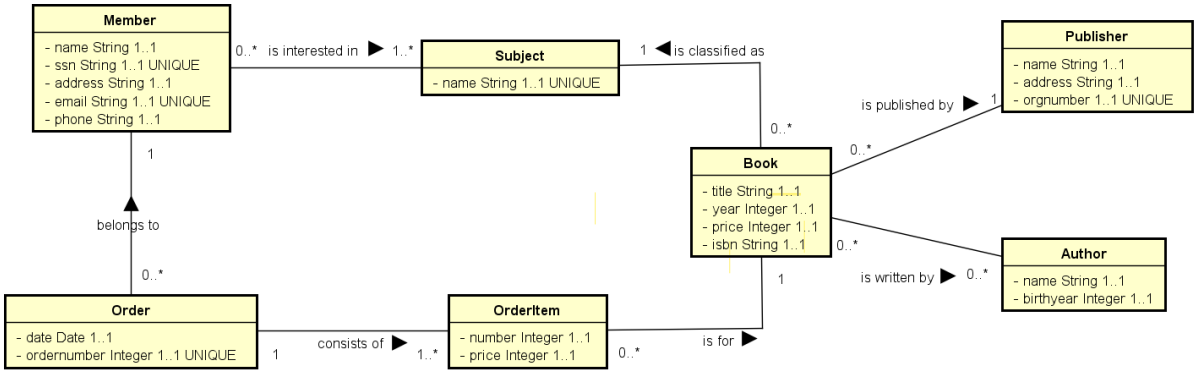
JSON:

```
{"people" : [{"id" : "", "name" : "", "ssn" : "", "address" : ""},
             {"id" : "", "name" : "", "ssn" : "", "address" : ""},
             {"id" : "", "name" : "", "ssn" : "", "address" : "",
               "doctor" : {"specialty" : "",
                           "visits" : [{"date" : "", "time" : "", "patient" : "", "symptoms" : ["", ""]},
                                       {"date" : "", "time" : "", "patient" : "", "symptoms" : [""]}]
                           }
              }
            ],
"symptoms" : [{"id" : "", "name" : ""}, {"id" : "", "name" : ""}]
}
```

The values of the ids can be numbers and we could separate the doctors from the other people. There are many possibilities.

EXERCISE SD2

Create suitable structures in SSD, XML and JSON for the following conceptual model!



Let's start with XML. All the points apply in a similar way if we would be designing an SSD graph or expression.

This model has multiple strong classes that are suitable to be directly under the root. Member, Publisher, Subject and Author are all suitable. Every choice will probably have pros and cons. Perhaps more redundancy can lead to less references. There are many ways to adequately represent the information of the model. What are the advantages and disadvantages of the following solution?

```
<DB>
   <Publisher name="" address="" orgnumber="">
      <Book id="" title="" year="" isbn="" price="" subject="">
         <Author name="" birthyear="" />
         <Author name="" birthyear="" />
      </Book>
      <Book id="" title="" year="" isbn="" price="" subject="">
         <Author name="" birthyear="" />
      </Book>
   </Publisher>
   <!-- more publishers with books -->
```

```
Publisher
   Book
      Author
      Author
```

```
    <Member name="" ssn="" address="" email="" phone="">
        <Order date="" number="">
            <Item number="" price="" book=""/>
            <Item number="" price="" book=""/>
        </Order>
        <Order date="" number="">
            <Item number="" price="" book=""/>
        </Order>
        <Subject>???</Subject>
        <Subject>???</Subject>
    </Member>
    <!-- more members with zero or more orders and one or more subjects -->
</DB>
```

Member
Order
Item number

It is of course possible to use elements with text nodes instead of attributes, perhaps for things like the book title and the book subject. The above structure has some redundancy due to the repetition of the subjects per book. We could group the books per subject in order to reduce the redundancy and at the same time guarantee that each book has a publisher and a subject:

```
<DB>
    <Publisher name="" address="" orgnumber="">
        <Subject name="">
            <Book id="" title="" year="" isbn="" price="">
                <Author name="" birthyear="" />
                <Author name="" birthyear="" />
            </Book>
            <Book id="" title="" year="" isbn="" price="">
                <Author name="" birthyear="" />
            </Book>
        </Subject>
        <Subject name="">
            <Book id="" title="" year="" isbn="" price="">
                <Author name="" birthyear="" />
                <Author name="" birthyear="" />
            </Book>
        </Subject>
        <!-- More subjects with one or more books -->
    </Publisher>
    <!-- more publishers with books grouped by subject -->
    <Member name="" ssn="" address="" email="" phone="">
        <Order date="" number="">
            <Item number="" price="" book=""/>
            <Item number="" price="" book=""/>
        </Order>
        <Order date="" number="">
            <Item number="" price="" book=""/>
        </Order>
        <Subject>???</Subject>
        <Subject>???</Subject>
    </Member>
    <!-- more members with zero or more orders and one or more subjects -->
</DB>
```

The subjects could also be placed directly under the root with books and members placed inside each subject. Perhaps with references for the members, but with the full books. The Publisher-Subject relationship could be flipped and, to avoid redundancy, the publishers could be placed under the root. Perhaps like this:

```
<DB>
    <Publisher name="" address="" orgnumber=""/>
    <Publisher name="" address="" orgnumber=""/>
    <Publisher name="" address="" orgnumber=""/>
    <Subject name="">
        <Publisher orgnumber="">
            <Book id="" title="" year="" isbn="" price="">
                <Author name="" birthyear="" />
                <Author name="" birthyear="" />
            </Book>
            <Book id="" title="" year="" isbn="" price="">
                <Author name="" birthyear="" />
            </Book>
        </Publisher>
        <Publisher orgnumber="">
            <Book id="" title="" year="" isbn="" price="">
                <Author name="" birthyear="" />
                <Author name="" birthyear="" />
            </Book>
        </Publisher>
        <!-- More publishers with one or more books of this subject -->
        <InterestedMember ssn=""/>
        <InterestedMember ssn=""/>
        <InterestedMember ssn=""/>
    </Subject>
    <!-- More subjects with zero or more publishers with books and zero or more interested members -->
    <Member name="" ssn="" address="" email="" phone="">
        <Order date="" number="">
            <Item number="" price="" book=""/>
            <Item number="" price="" book=""/>
        </Order>
        <Order date="" number="">
            <Item number="" price="" book=""/>
        </Order>
    </Member>
    <!-- more members with zero or more orders -->
</DB>
```

If we would like to avoid having the same element names at different places in the structure, we could perhaps change the Publisher elements inside the Subject elements to BooksByPublisher.

As for JSON, perhaps the following structure would be suitable:

```
{"publishers" : [{"name" : "", "address" : "", "orgnumber" : ""}, {"name" : "", "address" : "", "orgnumber" : ""}],
"subjects" : [{"name" : "", "books" : [{"id" : "", "title" : "",  "year" : "", "isbn" : "", "price" : "", "publisher" : ""},
                                       {"id" : "", "title" : "",  "year" : "", "isbn" : "", "price" : "", "publisher" : ""}],
                    "interestedmembers" : ["", "", ""]},
              {"name" : "", "books" : [{"id" : "", "title" : "",  "year" : "", "isbn" : "", "price" : "", "publisher" : ""},
                                       {"id" : "", "title" : "",  "year" : "", "isbn" : "", "price" : "", "publisher" : ""}]}],
"members" : [{"name" : "", "ssn" : "", "address" : "", "email" : "", "phone" : "",
              "orders" : [{"date" : "", "number" : "", "items" : [{"book" : "", "number" : "", "price" : ""},
                                                                  {"book" : "", "number" : "", "price" : ""}]},
                          {"date" : "", "number" : "", "items" : [{"book" : "", "number" : "", "price" : ""},
                                                                  {"book" : "", "number" : "", "price" : ""}]}]},
              {"name" : "", "ssn" : "", "address" : "", "email" : "", "phone" : ""}]
}
```

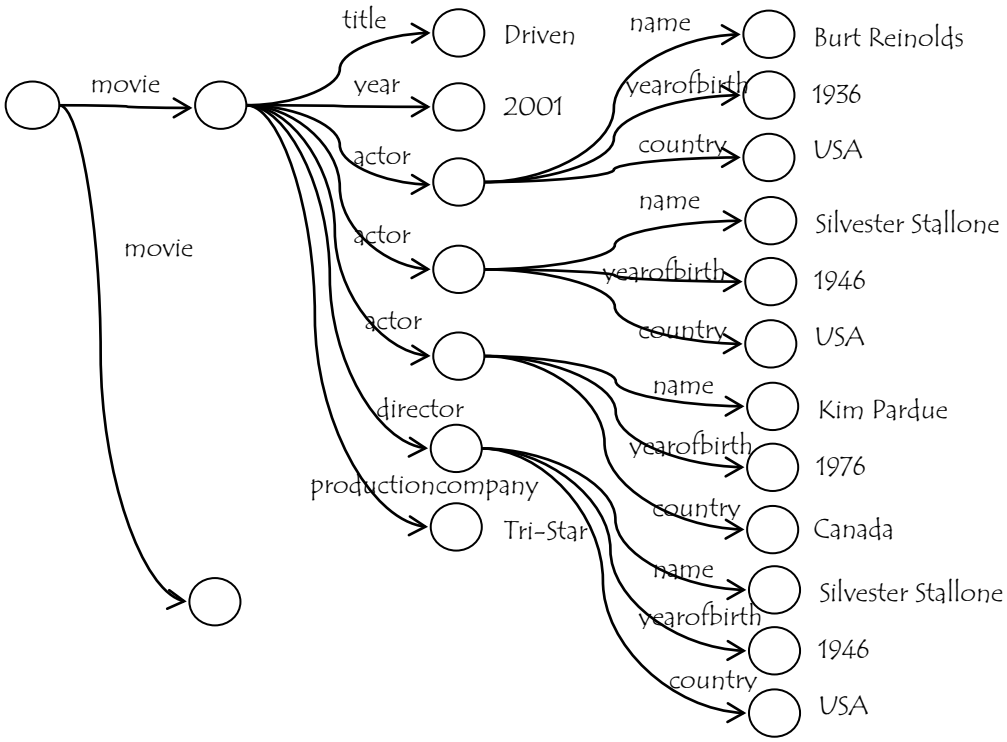
This structure is similar to the previous XML structure, but the books are not grouped by publisher. All the places that expect values are denoted by "", but numeric values will be without the quotes.

EXERCISE SD3

Convert the movie database (the XML document) into an SSD graph and an SSD expression! The first two movies should be enough.

SSD

```
db:{movie:{title:"Driven", year:2001,
      actor:{name:"Burt Reinolds", yearofbirth:1936, country:"USA"},
      actor:{name:"Silvester Stallone", yearofbirth:1946, country:"USA"},
      actor:{name:"Kip Pardue", yearofbirth:1976, country:"Canada"},
      director:{name:"Silvester Stallone", yearofbirth:1946, country:"USA"},
      productioncompany:"Tri-Star"},
movie:{title:"Antz", year:1998,
      actor:{name:"Woody Allen", yearofbirth:1935, country:"USA"},
      actor:{name:"Silvester Stallone", yearofbirth:1946, country:"USA"},
      actor:{name:"Sharon Stone", yearofbirth:1958, country:"USA"},
      director:{name:"Eric Darnell", yearofbirth:1961, country:"Ireland"},
      productioncompany:"Universal"},
...
}
```



Write Lorel statements for the following:

1. Retrieve the name of the production company of the movie "True Lies"!

```
select result:P
from db.movie M, M.title T, M.productioncompany P
where T = "True Lies"
```

2. Retrieve the name of the actors of the movie "Sliver"!

```
select actor:N
from db.movie M, M.actor.name N, M.title T
where T = "Sliver"
```

3. Retrieve the title and the director's name for movies where the director is also one of the actors!

```
select result:{title:T, director:DN}
from db.movie M, M.director.name DN, M.title T
where DN in M.actor.name
```

EXERCISE SD4

Convert the movie database (the XML document) into JSON! The first two movies should be enough. Construct a corresponding JSON Schema!

JSON

```
[{"title" : "Driven", "year" : 2001,
    "actors":[{"name" : "Burt Reinolds", "yearofbirth" : 1936, "country" : "USA"},
                {"name" : "Silvester Stallone", "yearofbirth" : 1946, "country" : "USA"},
                {"name" : "Kip Pardue", "yearofbirth" : 1976, "country" : "Canada"}],
    "director" : {"name" : "Silvester Stallone", "yearofbirth" : 1946, "country" : "USA"},
    "productioncompany" : "Tri-Star"},
 {"title" : "Antz", "year" : 1998,
    "actors" : [{name:"Woody Allen", "yearofbirth" : 1935, "country" : "USA"},
                {name:"Silvester Stallone", "yearofbirth" : 1946, "country" : "USA"},
                {name:"Sharon Stone", "yearofbirth" : 1958, "country" : "USA"}],
    "director" : {"name" : "Eric Darnell", "yearofbirth" : 1961, "country" : "Ireland"},
    "productioncompany" : "Universal"}
]
```

JSON Schema

```
{"$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "http://dsv.su.se/SDXML/jsonschema/movie",
  "type": "object",
  "title": "A movie",
  "description": "An object representation of a movie",
  "properties": {
      "title": {"type": "string"},
      "year": {"type": "integer"},
      "actors": {
          "type": "array",
          "items": {
              "type": "object",
              "properties": {
                  "name": {"type": "string"}, "yearofbirth": {"type": "integer"}, "country": {"type": "string"}
              },
              "additionalProperties" : false,
              "required": ["name", "yearofbirth", "country"]
          }
      },
      "director": {
          "type": "object",
          "properties": {
              "country": {"type": "string"}, "name": {"type": "string"}, "yearofbirth": {"type": "integer"}
          },
          "additionalProperties" : false,
          "required": ["name", "yearofbirth", "country"]
      },
      "productioncompany": {"type": "string"}
  },
  "additionalProperties" : false,
  "required": ["title", "year", "actors", "director", "productioncompany"]
}
```

Since the actors and the directors have the same structure, we could create a schema for person and reuse it:

```
{
   "$schema": "https://json-schema.org/draft/2020-12/schema",
   "$id": "http://dsv.su.se/SDXML/jsonschema/movieperson",
   "type": "object",
   "title": "A movie contributor",
   "description": "An object representation of a person that can be an actor of a director",
    "properties": {"name": {"type": "string"},
                        "yearofbirth": {"type": "integer"},
                        "country": {"type": "string"}},
    "additionalProperties" : false,
    "required": ["name", "yearofbirth", "country"]
}
{"$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "http://dsv.su.se/SDXML/jsonschema/movie",
  "type": "object",
  "title": "A movie",
  "description": "An object representation of a movie",
  "properties": {
        "title": {"type": "string"},
        "year": {"type": "integer"},
        "actors": {
            "type": "array",
            "items": {"$ref" : "http://dsv.su.se/SDXML/jsonschema/movieperson"}
        },
        "director": {"$ref" : "http://dsv.su.se/SDXML/jsonschema/movieperson"},
        "productioncompany": {"type": "string"}
  },
  "additionalProperties" : false,
  "required": ["title", "year", "actors", "director", "productioncompany"]
}
```