Keep me this data for later!

(Persisting data in Pharo)

As always, several options

- Relational
  - MySQL
  - Postgress
  - SQLite3
  - ODBC
- Object
  - OmniBase
  - Gemstone
- NoSQL
  - Mongo
  - Redis
  - EJDB
  - UnQLite
- Snapshot

… and of course there are more around

Snapshot

- This is very easy and enough for many cases.
  - Just… save the image with the data you want. They will be there when you open it back.
- It can be combined with some serialization ***ad-hoc*** mechanism.
  - Fuel
  - STON

Object databases

- They match one-to-one the object model.
- They are frequently coupled to the environment.

Relational

- Usually the preferred option (even if not always appropriate) but useful when needing projections.
- As an open source project, we have better support for open source alternatives: PostgreSQL, MySQL (MariaDB) and SQLite3
- But we also have ODBC support, which solves most of the problems for commercial databases.

Relational example on Pharo

```
connection := SQLite3Connection memory.
connection open.
connection execute: 'CREATE TABLE collections (
     id INTEGER PRIMARY KEY AUTOINCREMENT,
     name TEXT NOT NULL
);'.
connection execute: 'CREATE TABLE authors (
     id INTEGER PRIMARY KEY AUTOINCREMENT,
     collectionId INTEGER,
     name TEXT NOT NULL,
     FOREIGN KEY (collectionId) REFERENCES collections(id)
);'.
connection execute: 'CREATE TABLE books (
     id INTEGER PRIMARY KEY AUTOINCREMENT,
     authorId INTEGER,
     name TEXT NOT NULL,
     FOREIGN KEY (authorId) REFERENCES authors(id)
);'.
connection execute: 'INSERT INTO collections(name) VALUES (?);' value: 'SF'.
connection execute: 'INSERT INTO authors(name, collectionId) VALUES (?, ?);' value: 'Philip K.
Dick' value: 1.
```

No-relational

- Hyped on 2010+, and they have become very common since then.
- No single approach, hard to categorize (Key-Value, JSON documents, column based, …).
- Somehow they feel more appropriate for object-oriented solutions (but not perfect).

NoSQL example on Pharo (Mongo)

```
mongo := Mongo host: 'localhost' port: 27017.
mongo open.

collection := mongo databaseNamed: 'test' getCollection: 'collections'.
collection add: (Dictionary new
    at: #name put: 'SF';
    at: #authors put: (Dictionary new
        at: #name put: 'Philip K. Dick';
        yourself);
    yourself).
```

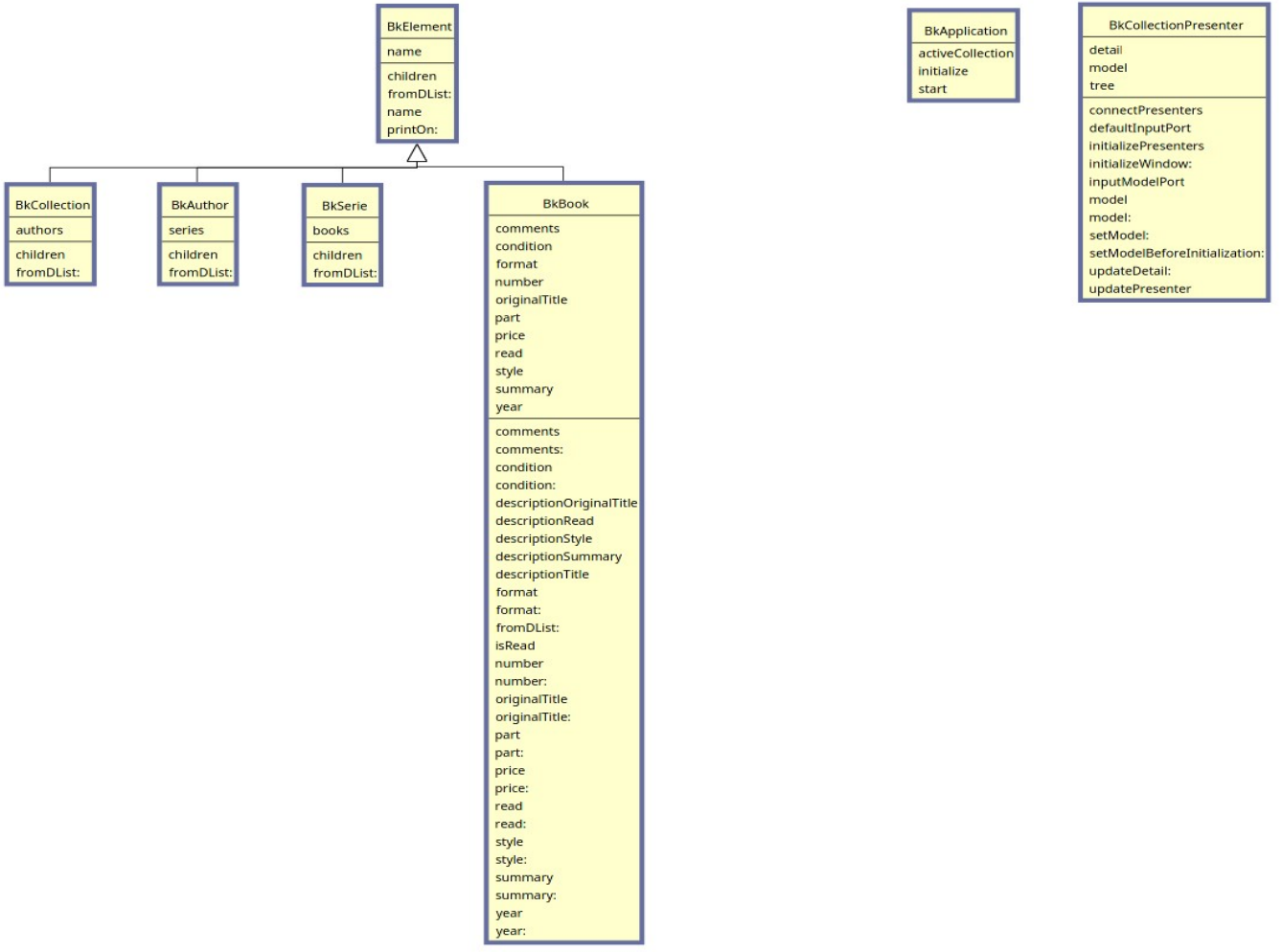
Frameworks

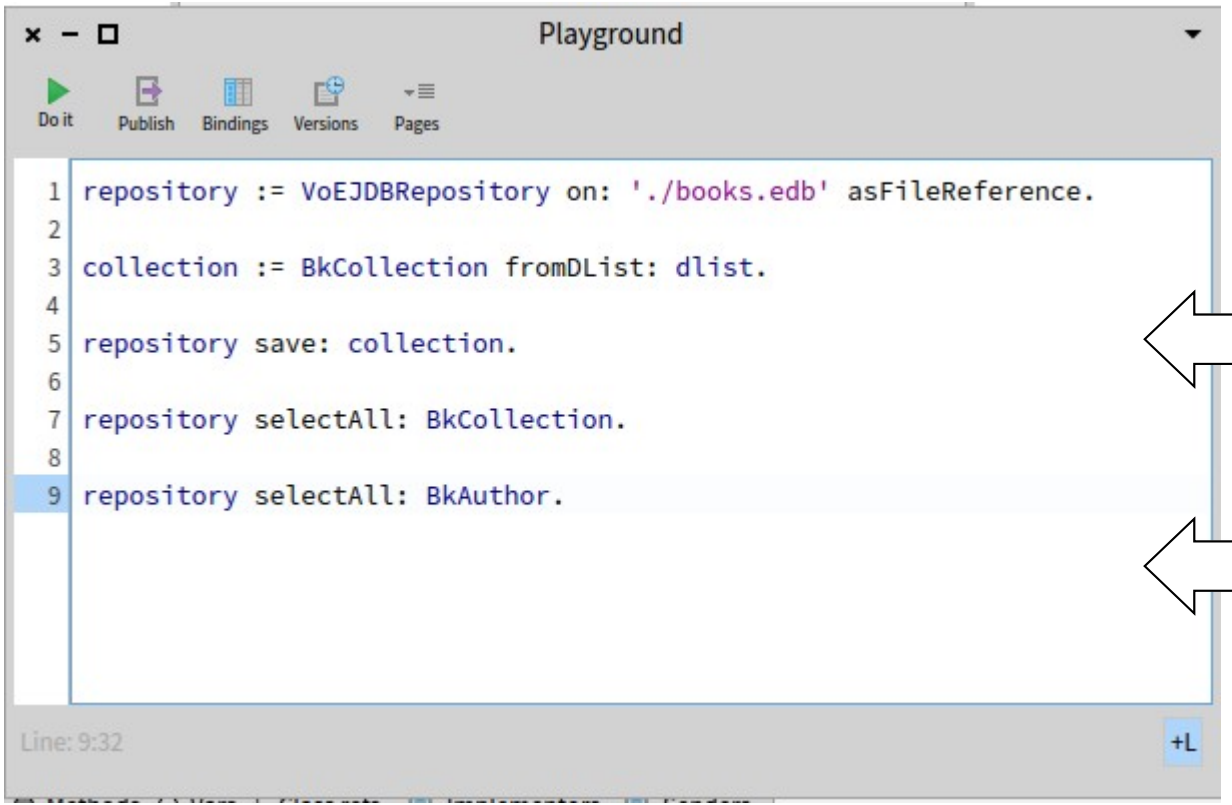
- GLORP (relational), think on Hibernate
- ReStore (relational) → Watch tomorrow's talk.
- Voyage (NoSQL), useful for mapping document databases (like MongoDB) to Objects.
- (Object databases do not need a framework other than themselves)

Voyage example (connecting to Mongo)

```
repository := VOMongoRepository host: 'localhost'.
bookCollection := BkCollection new
    name: 'SF';
    addAuthor: (BkAuthor new
        name: 'Philip K. Dick';
        yourself);
    yourself.
repository save: bookCollection.
```

Showcase: Saving Stef books in a database


BkElement
name
children
fromDList:
name
printOn:
BkCollection
authors
children
fromDList:
BkAuthor
series
children
fromDList:
BkSerie
books
children
fromDList:
BkBook
comments
condition
format
number
originalTitle
part
price
read
style
summary
year
comments
comments:
condition
condition:
descriptionOriginalTitle
descriptionRead
descriptionStyle
descriptionSummary
descriptionTitle
format
format:
fromDList:
isRead
number
number:
originalTitle
originalTitle:
part
part:
price
price:
read
read:
style
style:
summary
summary:
year
year:
BkApplication
activeCollection
initialize
start
BkCollectionPresenter
detail
model
tree
connectPresenters
defaultInputPort
initializePresenters
initializeWindow:
inputModelPort
model
model:
setModel:
setModelBeforeInitialization:
updateDetail:
updatePresenter



(**BkCollection** and **BkAuthor** as roots)

```
BkCollection class>>isVoyageRoot
    ^ true
```

```
BkAuthor class>>isVoyageRoot
    ^ true
```