Inicjatywa NoSQL na przykładzie db4o.

Marcin Stachniuk
mstachniuk@gmail.com
http://mstachniuk.blogspot.com

14 grudnia 2010

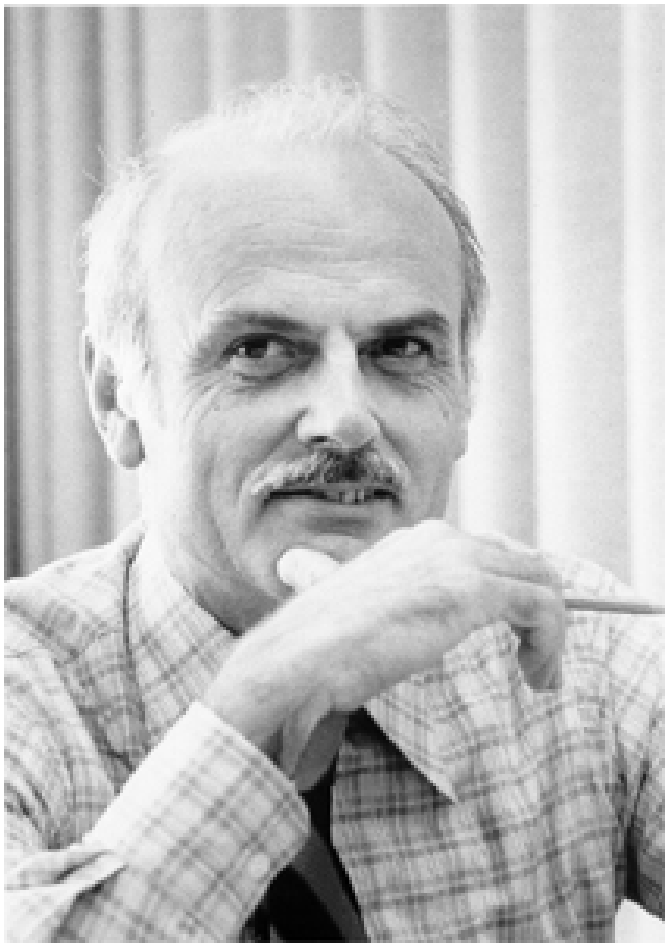
WARNING!

kontrowersyjny / „śliski” temat

mimo wszystko zachęcam do dyskusji :)

Tak to się zaczęło...

Edgar Frank Codd

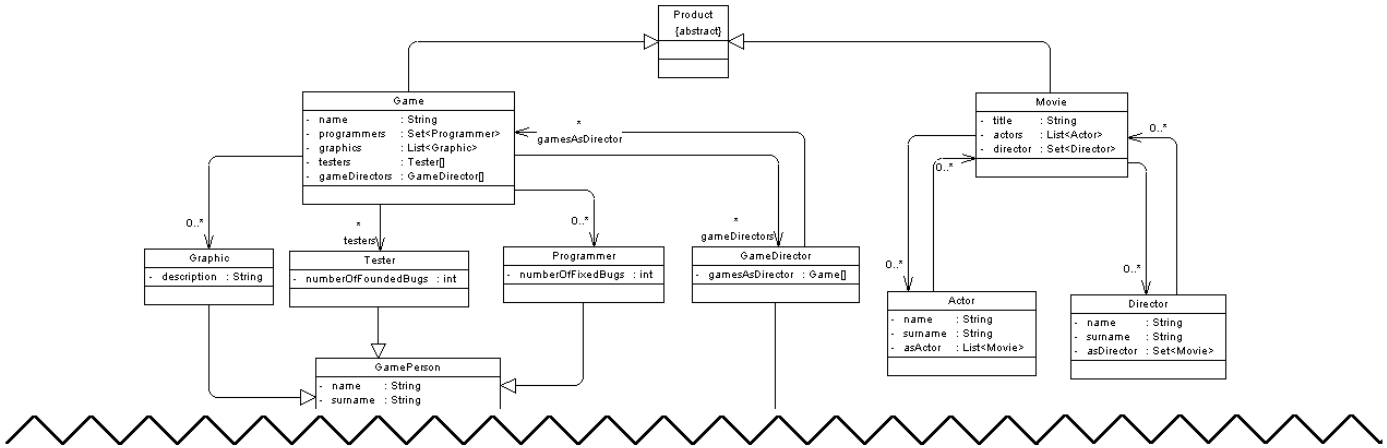


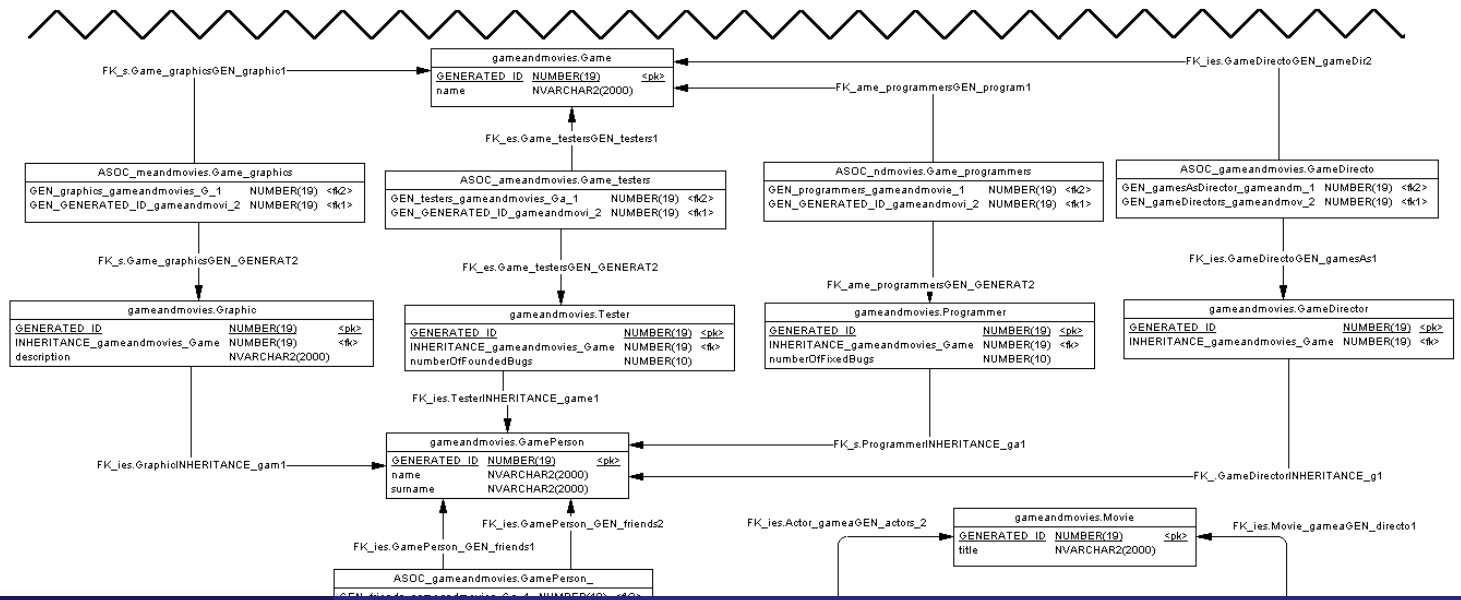
A Relational Model of Data for Large Shared Data Banks, **1970**

...a to było później

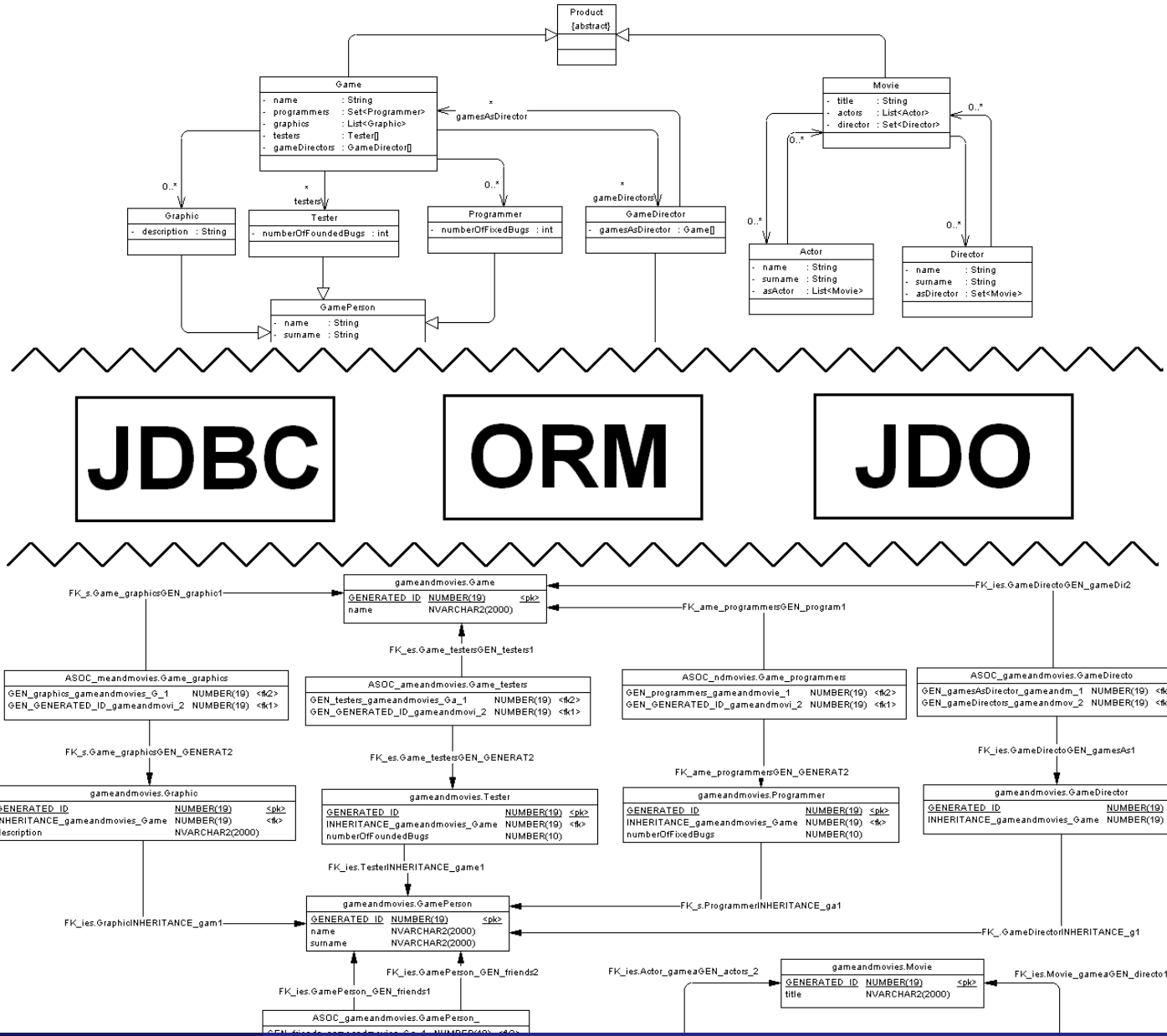
- **1972** Pojawienie się Smalltalk'a
- **1979** Pojawienie się Oracle'a, pierwszej relacyjnej, komercyjnej bazy danych
- **1979** Pojawienie się C++
- **1980** Pojawienie się Smalltalk-80
- **1995** Pojawienie się Javy

Model relacyjny i obiektowy





Model relacyjny i obiektowy



Niezgodność typów danych

Model relacyjny	Model obiektowy
VARCHAR2(size), NVARCHAR2(size), CHAR(size), NCHAR(size)	`java.lang.String`
NUMBER(5)	`short`
NUMBER(10)	`int`
NUMBER(19)	`long`
LONG	`ResultSet.getBinaryStream()`
NUMBER	`double`
DATE	`java.sql.Timestamp`
TIMESTAMP	`oracle.sql.TIMESTAMP`

Na przykładzie Oracle 10g

SQL Injection

co jeszcze...

- *Są głosy (H.Baker), że model relacyjny opóźnił rozwój przemysłu baz danych o 10 lat.*[1]
- *Można skonstruować kompletny język zapytań/programowania oparty wyłącznie o algebrę relacji. Tę demagogię propaguje utwór „The Third Manifesto" H.Darwena i C.J.Date.*[2]
- *SQL jest oparty na logice matematycznej. Przekręt w wykonaniu J. Ullmana.*[2]
- *W odróżnieniu od systemów obiektowych, systemy relacyjne i SQL posiadają solidne podstawy matematyczne. Nieprawda. [...] SQL posiada wiele operatorów nie rozpatrywanych przez relacyjne teorie, np. operatory aktualizacyjne, uporządkowanie [...]*[2]

[1] Słownik terminów z zakresu obiektowości, Kazimierz Subieta

[2] Obiektowe Bazy Danych kontra Relacyjne Bazy Danych, Kazimierz Subieta

co jeszcze...

- *Są głosy (H.Baker), że model relacyjny opóźnił rozwój przemysłu baz danych o 10 lat.*[1]
- *Można skonstruować kompletny język zapytań/programowania oparty wyłącznie o algebrę relacji. Tę demagogię propaguje utwór „The Third Manifesto" H.Darwena i C.J.Date.*[2]
- *SQL jest oparty na logice matematycznej. Przekręt w wykonaniu J. Ullmana.*[2]
- *W odróżnieniu od systemów obiektowych, systemy relacyjne i SQL posiadają solidne podstawy matematyczne. Nieprawda. [...] SQL posiada wiele operatorów nie rozpatrywanych przez relacyjne teorie, np. operatory aktualizacyjne, uporządkowanie [...]*[2]

[1] Słownik terminów z zakresu obiektowości, Kazimierz Subieta

[2] Obiektowe Bazy Danych kontra Relacyjne Bazy Danych, Kazimierz Subieta

co jeszcze...

- *Są głosy (H.Baker), że model relacyjny opóźnił rozwój przemysłu baz danych o 10 lat.*[1]
- *Można skonstruować kompletny język zapytań/programowania oparty wyłącznie o algebrę relacji. Tę demagogię propaguje utwór „The Third Manifesto" H.Darwena i C.J.Date.*[2]
- *SQL jest oparty na logice matematycznej. Przekręt w wykonaniu J. Ullmana.*[2]
- *W odróżnieniu od systemów obiektowych, systemy relacyjne i SQL posiadają solidne podstawy matematyczne. Nieprawda. [...] SQL posiada wiele operatorów nie rozpatrywanych przez relacyjne teorie, np. operatory aktualizacyjne, uporządkowanie [...]*[2]

[1] Słownik terminów z zakresu obiektowości, Kazimierz Subieta

[2] Obiektowe Bazy Danych kontra Relacyjne Bazy Danych, Kazimierz Subieta

co jeszcze...

- *Są głosy (H.Baker), że model relacyjny opóźnił rozwój przemysłu baz danych o 10 lat.*[1]
- *Można skonstruować kompletny język zapytań/programowania oparty wyłącznie o algebrę relacji. Tę demagogię propaguje utwór „The Third Manifesto" H.Darwena i C.J.Date.*[2]
- *SQL jest oparty na logice matematycznej. Przekręt w wykonaniu J. Ullmana.*[2]
- *W odróżnieniu od systemów obiektowych, systemy relacyjne i SQL posiadają solidne podstawy matematyczne. Nieprawda. [...] SQL posiada wiele operatorów nie rozpatrywanych przez relacyjne teorie, np. operatory aktualizacyjne, uporządkowanie [...]*[2]

[1] Słownik terminów z zakresu obiektowości, Kazimierz Subieta
[2] Obiektowe Bazy Danych kontra Relacyjne Bazy Danych, Kazimierz Subieta

co jeszcze...

- *Systemy relacyjne umożliwiającą matematyczną weryfikację poprawności zaprojektowanej bazy danych. Nie umożliwiają. [...] Pojęcia takie jak 2NF, 3NF, 4NF, BCNF oraz związane z nimi teorie [...] służą prawie wyłącznie jako pseudo-naukowy muł zapychający programy nauczania i podręczniki dla studentów. Projektant instynktownie unika sytuacji, które nie są zgodne z 3NF, 4NF, itd...*[3]
- *Standard SQL-92 zapewnia pełną przenaszalność oprogramowania pomiędzy różnymi systemami relacyjnych baz danych. Nie zapewnia. Są opinie, że ponad 95% programów zgodnych z SQL-92 nie jest przenaszalna.*[3]
- jeszcze by się coś znalazło...

[3] Obiektowe Bazy Danych kontra Relacyjne Bazy Danych, Kazimierz Subieta

co jeszcze...

- *Systemy relacyjne umożliwiającą matematyczną weryfikację poprawności zaprojektowanej bazy danych. Nie umożliwiają. [...] Pojęcia takie jak 2NF, 3NF, 4NF, BCNF oraz związane z nimi teorie [...] służą prawie wyłącznie jako pseudo-naukowy muł zapychający programy nauczania i podręczniki dla studentów. Projektant instynktownie unika sytuacji, które nie są zgodne z 3NF, 4NF, itd...*[3]
- *Standard SQL-92 zapewnia pełną przenaszalność oprogramowania pomiędzy różnymi systemami relacyjnych baz danych. Nie zapewnia. Są opinie, że ponad 95% programów zgodnych z SQL-92 nie jest przenaszalna.*[3]
- jeszcze by się coś znalazło...

[3] Obiektowe Bazy Danych kontra Relacyjne Bazy Danych, Kazimierz Subieta

co jeszcze...

- *Systemy relacyjne umożliwiającą matematyczną weryfikację poprawności zaprojektowanej bazy danych. Nie umożliwiają. [...] Pojęcia takie jak 2NF, 3NF, 4NF, BCNF oraz związane z nimi teorie [...] służą prawie wyłącznie jako pseudo-naukowy muł zapychający programy nauczania i podręczniki dla studentów. Projektant instynktownie unika sytuacji, które nie są zgodne z 3NF, 4NF, itd...*[3]
- *Standard SQL-92 zapewnia pełną przenaszalność oprogramowania pomiędzy różnymi systemami relacyjnych baz danych. Nie zapewnia. Są opinie, że ponad 95% programów zgodnych z SQL-92 nie jest przenaszalna.*[3]
- jeszcze by się coś znalazło...

[3] Obiektowe Bazy Danych kontra Relacyjne Bazy Danych, Kazimierz Subieta

Rozwiązanie

W odpowiedzi na niedoskonałość relacyjnych baz danych powstała inicjatywa:

NoSQL

promująca rozwiazania bazodanowe, niewykorzystujące SQL'a.

Not only SQL

Rozwiązania NoSQL

CouchDB
Objectivity/DB
Tuple space
Core Data
Rocket U2
Tokyo Cabinet
MongoDB
InterSystems Caché
MarkLogic Server
Redis
ZODB
Jackrabbit
DEX
memcached
Graph
NMDB
SimpleDB
XML database
FlockDB
CDB
AllegroGraph
GemStone/S
eXist
Memcachedb
Hbase
BigTable
Berkeley DB
OpenQM
membase
Cassandra
Db4o
GT.M
Neo4j
Paxos
Dynamo
BigTable
Project Voldemort
Tuple space
Mnesia
Hibari
Velocity
Keyspace
Apache River
Hypertable

NoSQL w praktyce

Kto używa NoSQL:

- Twitter (Hadoop / HBase, FlockDB, Cassandra, Pig)
- Facebook (Cassandra)
- Google (BigTable)
- Digg.com (Cassandra)
- SourceForge.net (MongoDB)
- LinkedIn (Project Voldemort)
- Amazon (Dynamo)
- Subversion (BerkleyDB)

db4objects

BY VERSANT

dVP 2010

Charakterystyka db4o

Czym jest db4o?

- Obiektowa baza danych open-source
- Trzy licencje: darmowe (GPL[4], dOCL[5]) i komercyjna[6]
- Natywna implementacja dla języków Java i .NET
- Duża społeczność (60000 zarejestrowanych developerów)
- Ponad milion ściągnięć

[4] GNU General Public License
[5] db4o Opensource Compatibility License (dOCL)
[6] Commercial Packages and Prices

Charakterystyka db4o

Firmy korzystające z db4o:

- BMW
- Bosch
- Intel
- Seagate
- Merrill Lynch
- Macrix Software
- Eastern Data
- Die Mobilanten
- Pragmatyxs
- Juvander
- Long Island Housing Services
- Novator
- Syft Technologies
- Boeing
- IBM
- Ricoh
- INDRA Sistemas
- Postbank
- Mandala IT
- Riege Software International
- ITAnyplace
- Electrabel
- TradeWeapon
- MR Controls
- Clarity Medical
- Arum Systems

Charakterystyka db4o

Dla kogo db4o?

- Małe i średnie projekty (studenckie projekty!!!)
- Nowe niskobudżetowe projekty
- Gdzie nie ma narzuconej technologii wykorzystywanej bazy danych
- Baza może działać na pliku dyskowym (podobnie jak: SQLite)
- Brak konieczności stawiania osobnego serwera bazodanowego (ale oczywiście można)
- Świetnie się nadaje do aplikacji typu standalone

Wersje db4o

Dostępne wersje db4o:

- 7.4 – **Stable** Recommended for product shipment
- 7.12 – **Production** Recommended for development with db4o
- 8.0 – **Development** Beta release, Ideal for testing new features

Jak zacząć?

Podpiąć JAR'a pod projekt:

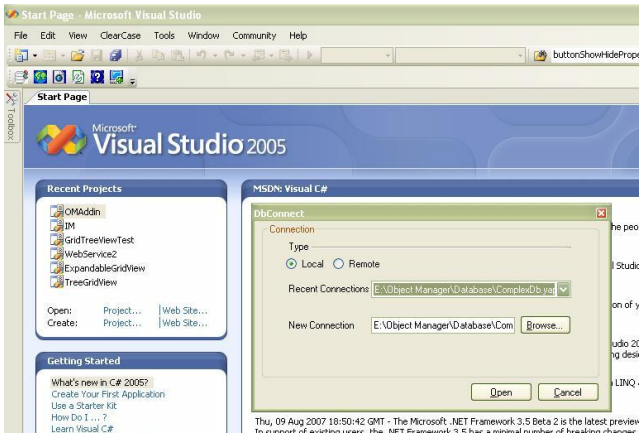
db4o-7.4.106.13438-java5.jar
Executable Jar File
1 223 KB

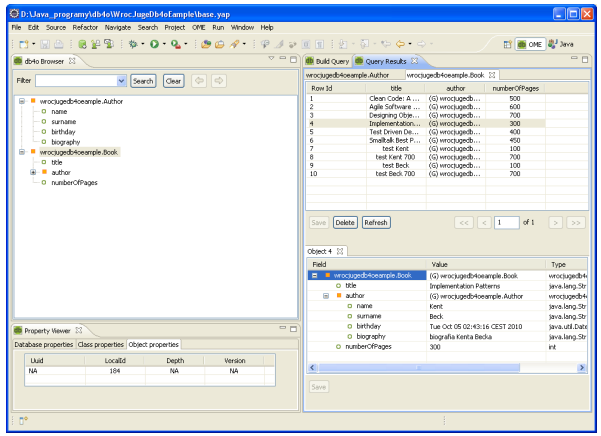
OR

```
1: <repositories>
2:    <repository>
3:       <id>db4o</id>
4:       <url>http://source.db4o.com/maven/</url>
5:    </repository>
6: </repositories>
7:
8: <dependency>
9:    <groupId>com.db4o</groupId>
10:   <artifactId>db4o-java5</artifactId>
11:   <version>7.4-SNAPSHOT</version>
12: </dependency>
```

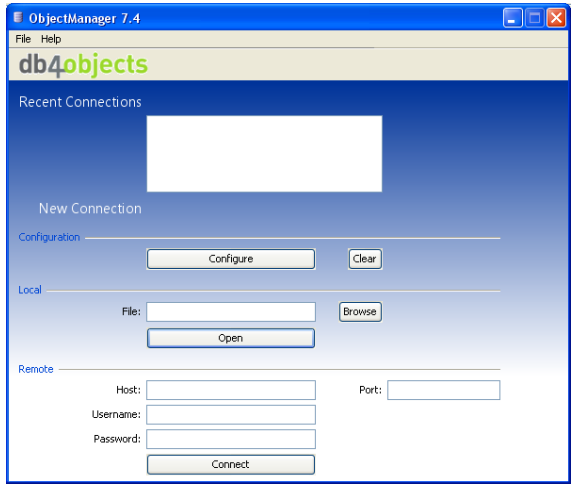
A co się jeszcze przyda?

ObjectManager Enterprise (OME)



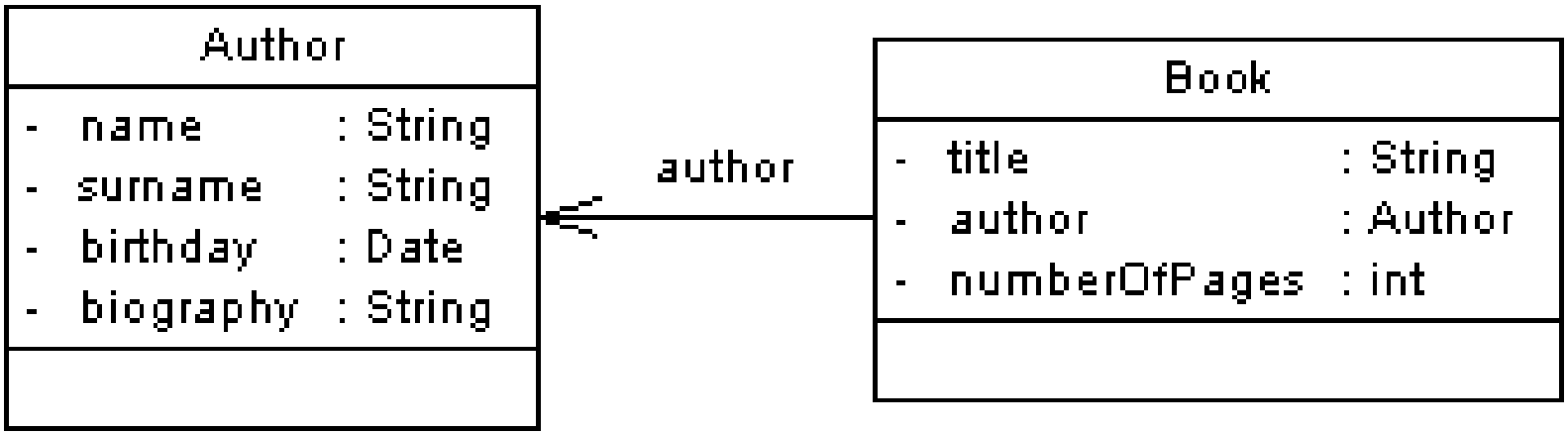


Object Manager



Czas na przykład

Diagram klas



INSERT INTO ... VALUES ...

```
1: ObjectContainer db = Db4o.openFile("base.yap");
2: db.store(new Book(...));
3: db.close();
```

Właściwości metody store()

Właściwości metody `store()`:

- Zapisuje graf obiektow w bazie
- Wykonywana jako jedna transakcja
- Pozwala zapisać dwa „takie same” obiekty (!!!)

SELECT * FROM ...

```
1: ObjectContainer db = Db4o.openFile("base.yap");
2: ObjectSet<Book> books = db.query(Book.class);
3: for (Book book : books) {
4:     System.out.println(book);
5: }
6: db.close();
```

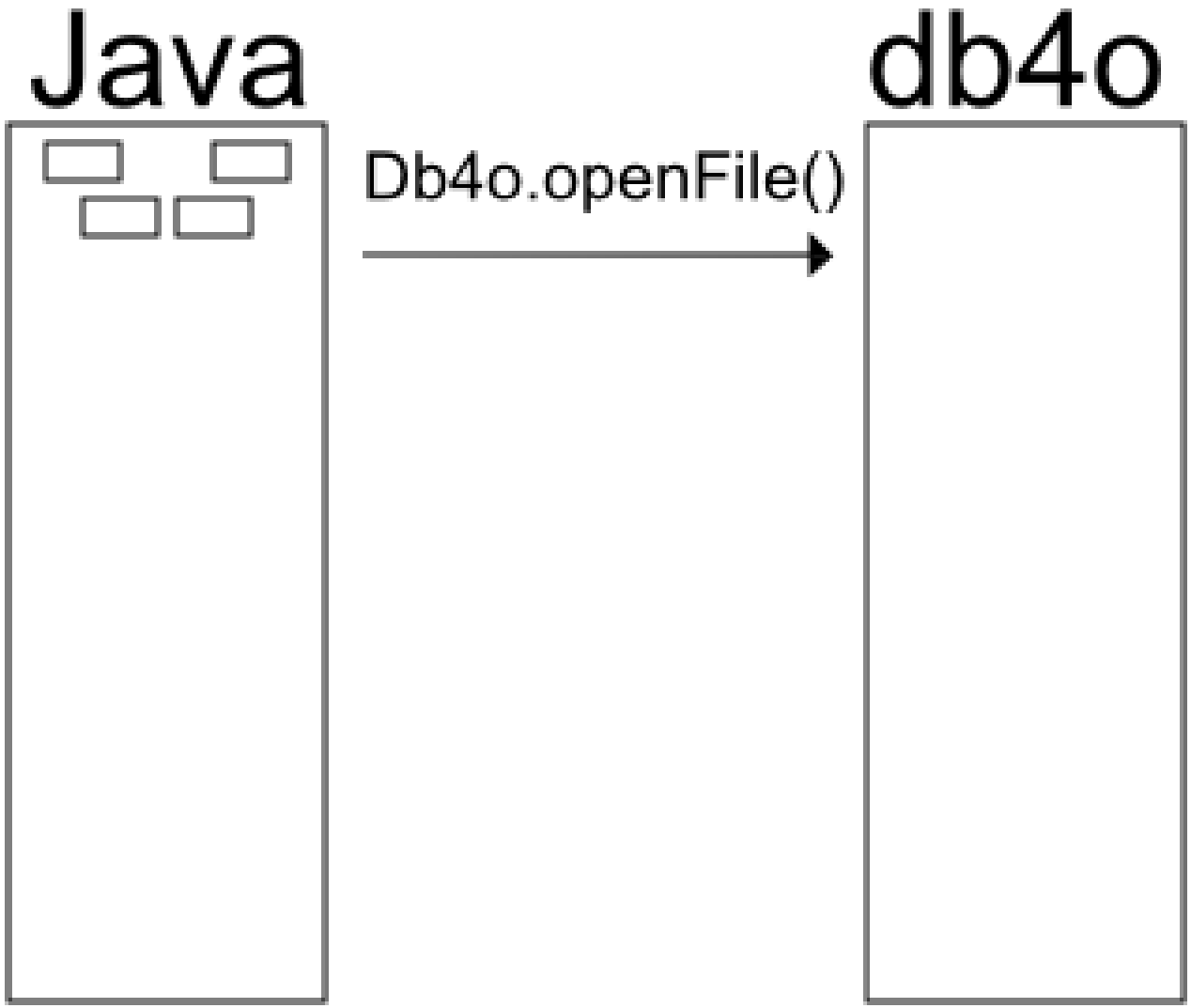
UPDATE ... SET ...

Modyfikację obiektu wykonujemy za pomocą znanej już metody store():

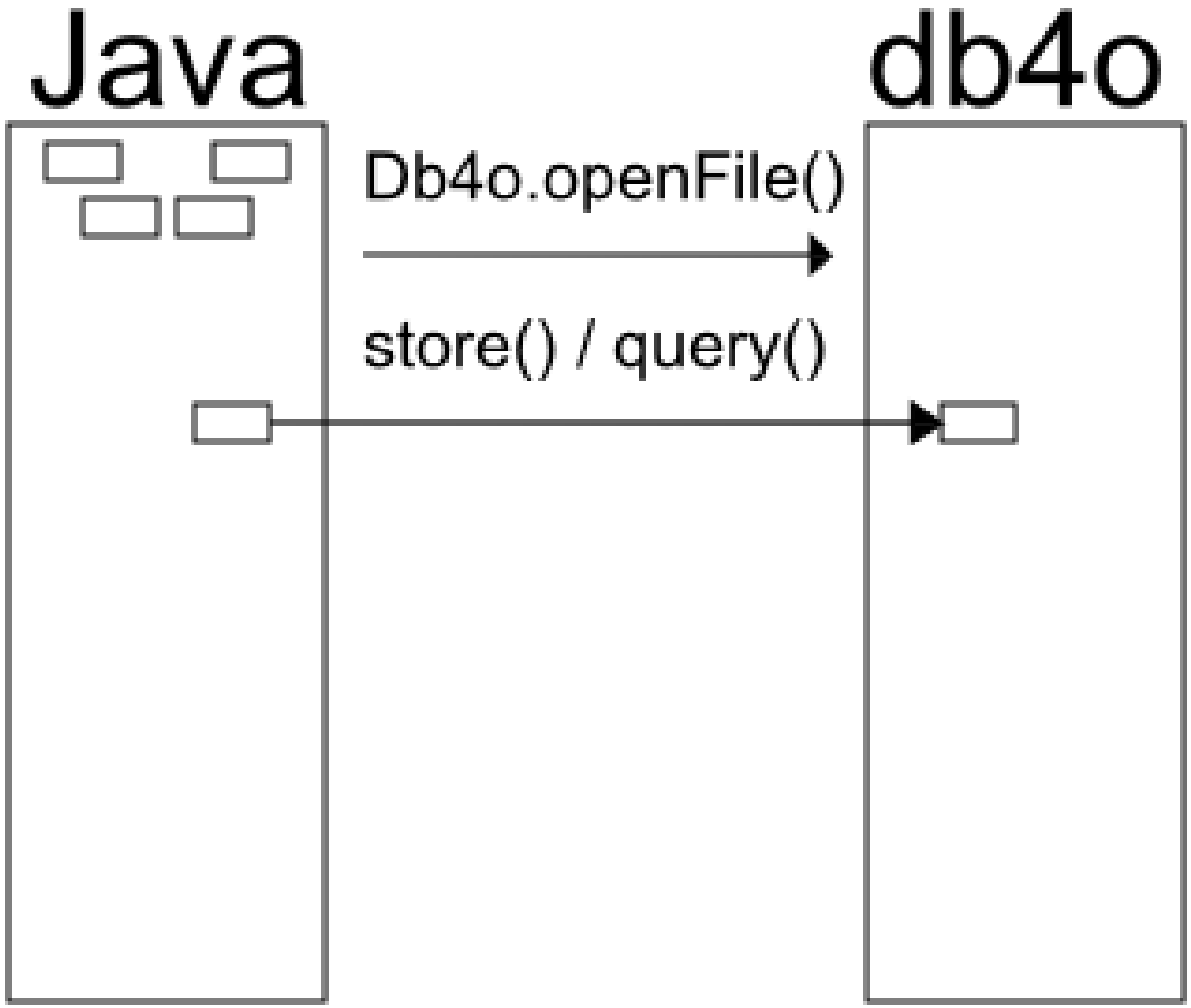
```
1: ObjectContainer db = Db4o.openFile("base.yap");
2: ObjectSet<Book> books = db.query(Book.class);
3: for (Book book : books) {
4:     book.setNumberOfPages(1000);
5:     db.store(book);
6: }
7: db.close();
```

Cykl życia obiektów

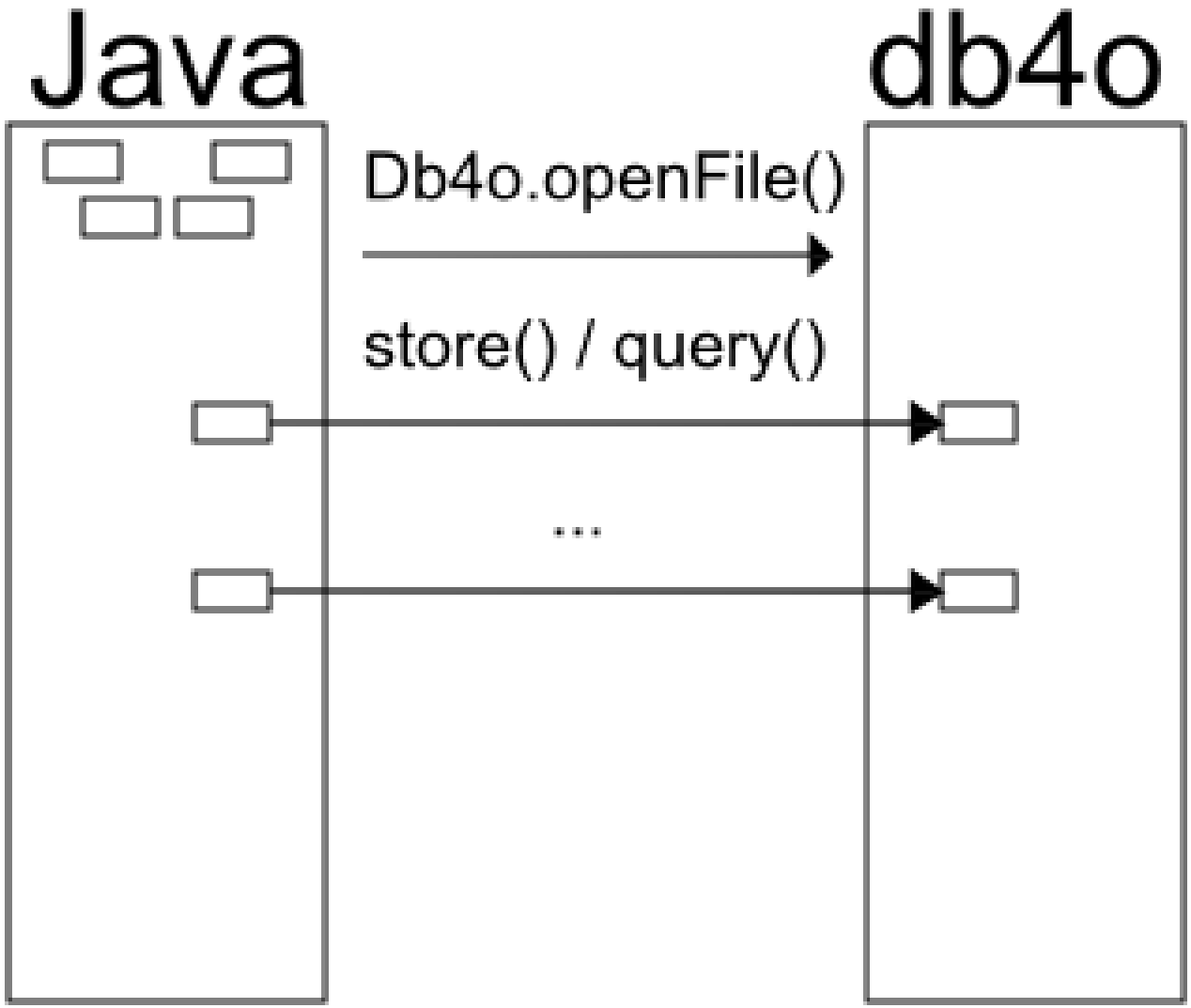
Cykl życia obiektów



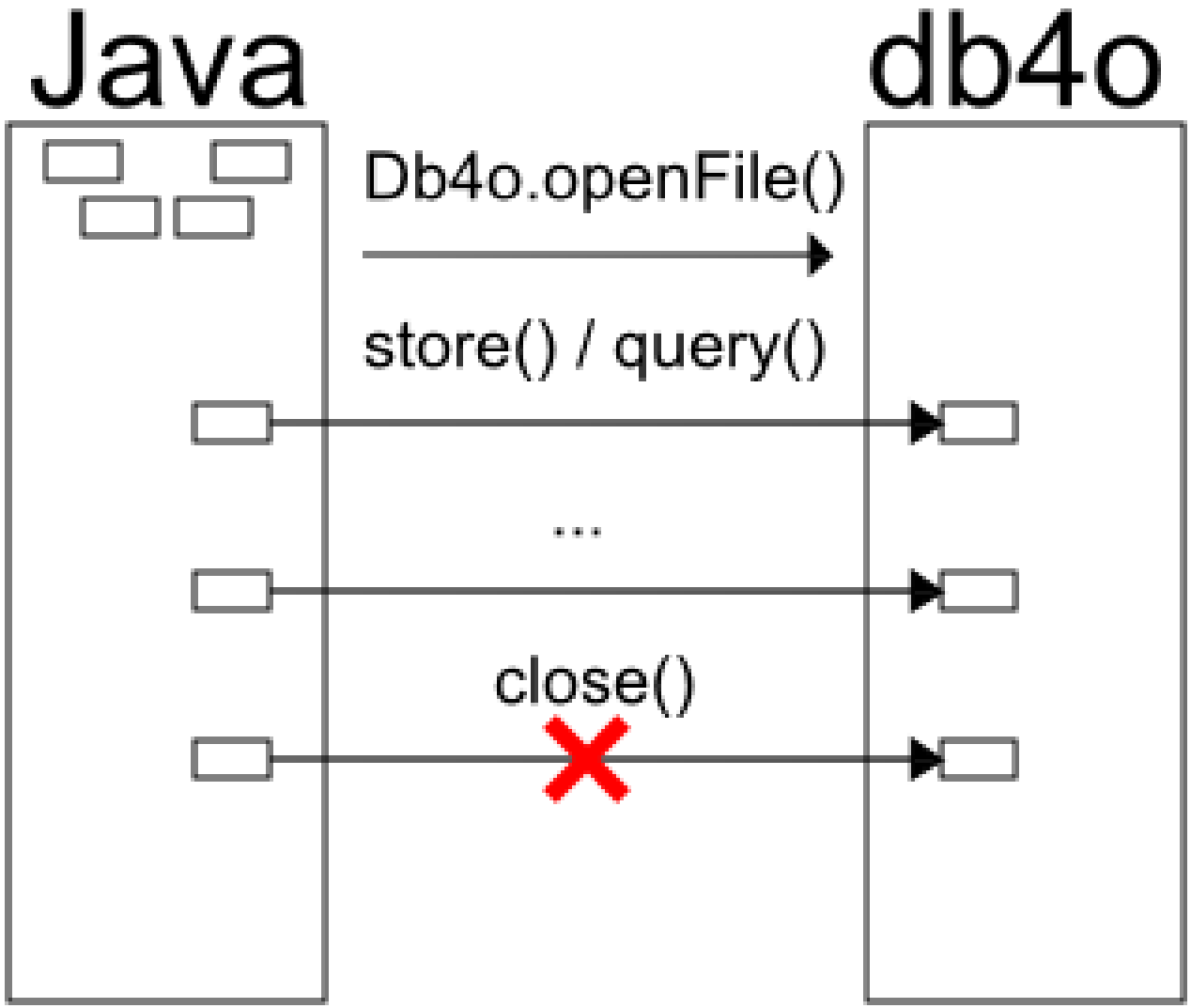
Cykl życia obiektów



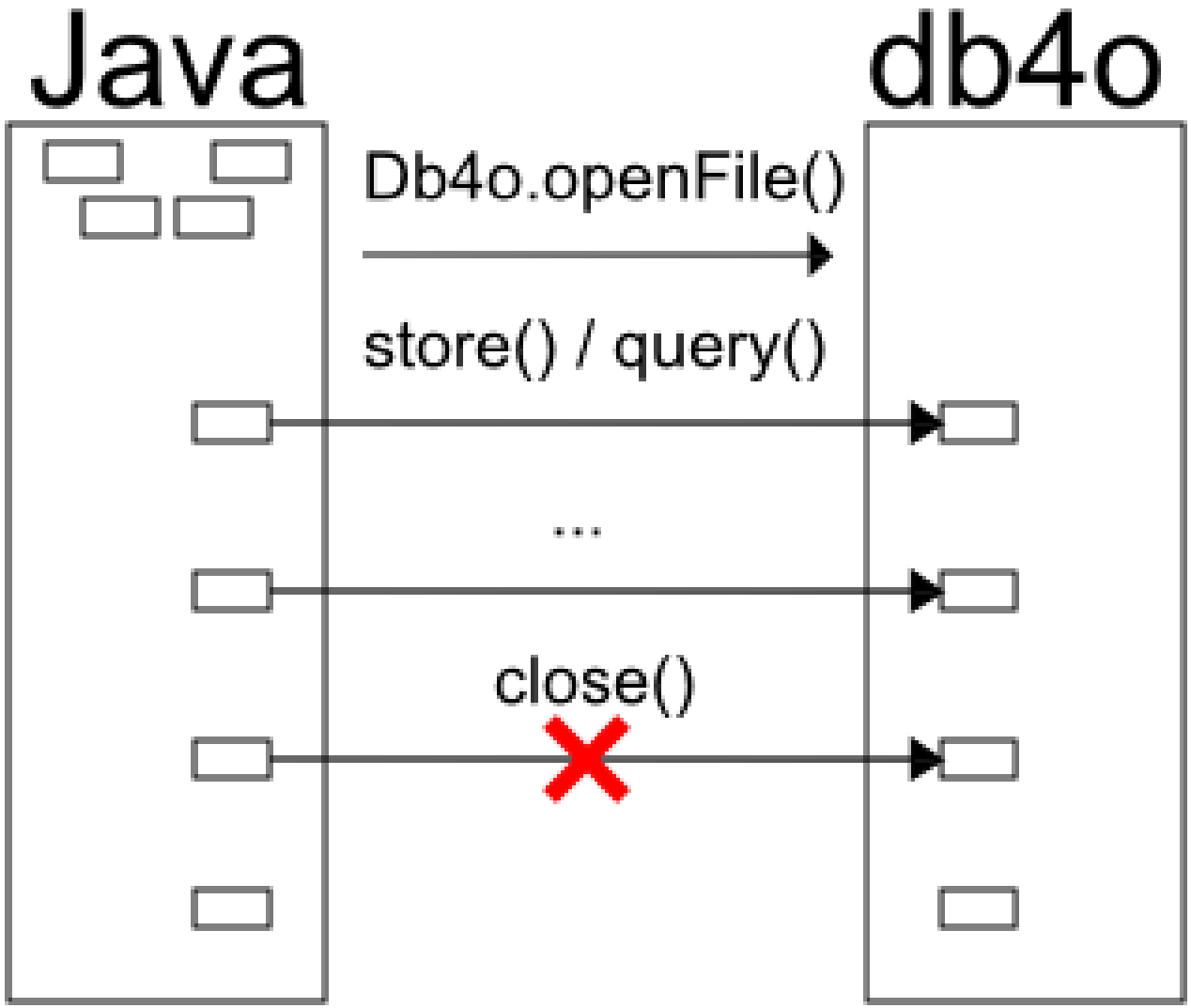
Cykl życia obiektów



Cykl życia obiektów



Cykl życia obiektów



DELETE FROM ... WHERE ...

```
1: ObjectContainer db = Db4o.openFile("base.yap");
2: ObjectSet<Book> books = db.query(Book.class);
3: for (Book book : books) {
4:     db.delete(book);
5: }
6: db.close();
```

Aktualizacja usuwanych obiektów

TIP!

Gdy usuwamy / aktualizujemy obiekty z `ObjectSet`, to aby zaktualizować zbiór, trzeba ponownie wykonać zapytanie, w celu aktualizacji zbioru obiektów.

Komplikacje

Problemy:

- Zamknięcie połączenia powoduje utratę „skojarzenia” obiektów
- Ciągłe otwieranie i zamykanie pliku jest kosztowne czasowo
- Przechowywanie nazwy pliku z bazą w wielu miejscach nie jest dobrym rozwiązaniem (zasada DRY)
- W danym czasie może być nawiązane jedno połączenie z plikiem (w trybie pracy na pliku)

Rozwiązanie: utrzymywać połączenie / pulę połączeń
zastosować **Db4oUtil**[7][8]

[7] Db4oUtil (aka. The Easiest Way to Get Started With Db4o)
http://developer.db4o.com/Projects/useful_snippets/db4outil_aka._the_easiest_way_to_get_started_with_db4o.html

[8] http://www.spaceprogram.com/knowledge/2006/07/db4outil-aka-easiest-way-to-get.html

Komplikacje

Problemy:

- Zamknięcie połączenia powoduje utratę „skojarzenia” obiektów
- Ciągłe otwieranie i zamykanie pliku jest kosztowne czasowo
- Przechowywanie nazwy pliku z bazą w wielu miejscach nie jest dobrym rozwiązaniem (zasada DRY)
- W danym czasie może być nawiązane jedno połączenie z plikiem (w trybie pracy na pliku)

Rozwiązanie: utrzymywać połączenie / pulę połączeń
zastosować **Db4oUtil**[7][8]

[7] Db4oUtil (aka. The Easiest Way to Get Started With Db4o)
`http://developer.db4o.com/Projects/useful_snippets/db4outil_aka._the_easiest_way_to_get_started_with_db4o.html`

[8] `http://www.spaceprogram.com/knowledge/2006/07/db4outil-aka-easiest-way-to-get.html`

Użycie Db4oUtil.java

Schemat postępowania z Db4oUtil.java:

```
1: // ustawienie nazwy pliku
2: Db4oUtil.setDatabaseFilename("baza.yap");
3:
4: // pobranie obiektu ObjectContainer
5: ObjectContainer db = Db4oUtil.getObjectContainer();
6:
7: // Kolejne operacje...
8:
9: // Zamknięcie ObjectContainer gdy koniec operacji
10: Db4oUtil.closeObjectContainer();
11:
12: // zamkniecie wszystkiego, przy kończeniu aplikacji
13: Db4oUtil.shutdown();
```

SELECT * FROM ... WHERE ...

Zapytania poprzez Query By Example:

- Tworzymy obiekt który będzie przykładem tego co mamy znaleźć
- Ustawione pola są dopasowywane „dokładnie”
- Dla pól z wartością domyślną wszystko pasuje

SELECT * FROM ... WHERE ...

Poszukajmy książek mających 500 stron:

```
1: ObjectContainer db = Db4oUtil.getObjectContainer();
2: Book exampleBook = new Book(null, null, 500);
3: ObjectSet<Book> books = db.queryByExample(exampleBook);
4: for (Book book : books) {
5:     System.out.println(book);
6: }
```

SELECT * FROM ... WHERE ...

Zapytania poprzez Native Queries:

- Rozszerzamy klasę `com.db4o.query.Predicate<ExtentType>`
- Przeciążamy metodę: `public boolean match(ExtentType et)`
- Wywołujemy: `db.query(Predicate predicate);`

SELECT * FROM ... WHERE ...

Poszukajmy książek Kenta Beck'a lub mające 700 stron.
„Normalnie" (czytaj w SQL'u[9]) by było to jakoś tak:

```
1: SELECT "Book"."ID", "title", "numberOfPages", "authorID"
2: FROM "Book" LEFT OUTER JOIN "Author" ON
3: "Book"."authorID" = "Author"."ID"
4: WHERE ("name"='Kent' AND "surname"='Beck')
5: OR "numberOfPages"=700
```

[9] Oracle 10 XE

SELECT * FROM ... WHERE ...

Poszukajmy książek Kenta Beck'a lub mające 700 stron.
W db4o będzie to tak:

```
 1: final String name = "Kent";
 2: final String surname = "Beck";
 3:
 4: ObjectContainer db = Db4oUtil.getObjectContainer();
 5:
 6: ObjectSet<Book> books = db.query(new Predicate<Book>() {
 7:
 8:     @Override
 9:     public boolean match(Book book) {
10:         if((book.getAuthor().getName().equals(name) &&
11:                 (book.getAuthor().getSurname().equals(
12:                     surname))) ||
13:                 book.getNumberOfPages() == 700)
14:             return true;
15:         return false;
16:     }
17: });
```

SELECT * FROM ... WHERE ...

Niby działa, ale ubezpieczmy się na wartość `null`:

```
 1: ObjectSet<Book> books = db.query(new Predicate<Book>() {
 2:
 3:     @Override
 4:     public boolean match(Book book) {
 5:         if((book.getAuthor().getName() != null &&
 6:                 book.getAuthor().getName().equals(name) &&
 7:                 (book.getAuthor().getSurname() != null &&
 8:                 book.getAuthor().getSurname().equals(
 9:                     surname))) ||
10:                 book.getNumberOfPages() == 700)
11:             return true;
12:         return false;
13:     }
14: });
```

Linie 5 i 7

SELECT * FROM ... WHERE ... ORDER BY ...

Sortowanie wyników przy Native Queries:

- Implementujemy `Comparator<T>`
- Wywołujemy: `db.query(Predicate predicate, Comparator comparator)`

SELECT * FROM ... WHERE ... ORDER BY ...

Kod komparatora:

```
1: public class BooksTitleComparator
2:         implements Comparator<Book> {
3:
4:     public int compare(Book b1, Book b2) {
5:         return b1.getTitle().compareToIgnoreCase(
6:             b2.getTitle());
7:     }
8:
9: }
```

Wywołanie:

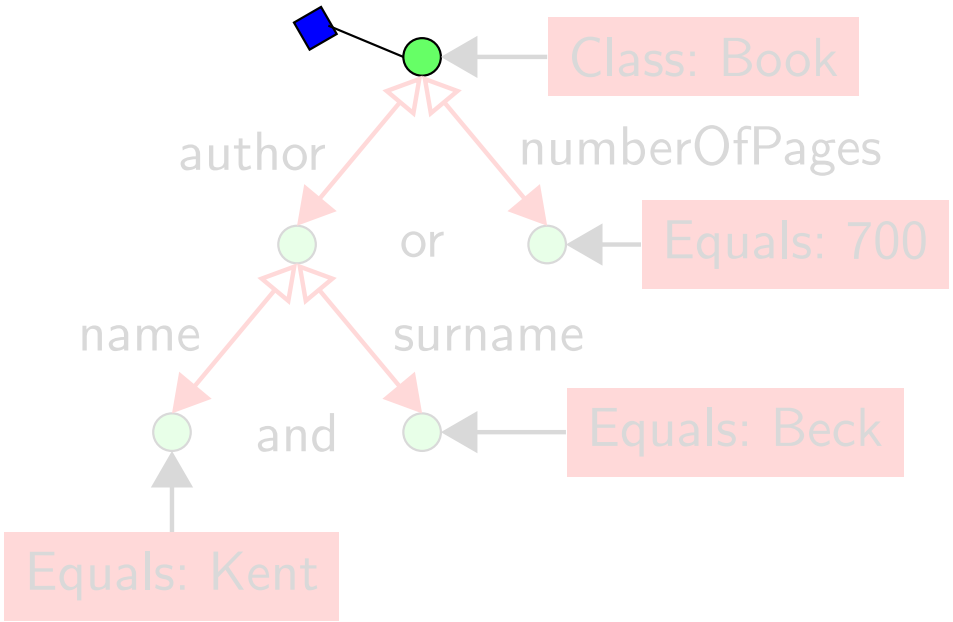
```
1: ObjectSet<Book> books = db.query(new Predicate<Book>() {...},
2:         new BooksTitleComparator());
```

Zapytania SODA Simple Object Data Access

Zapytania SODA:

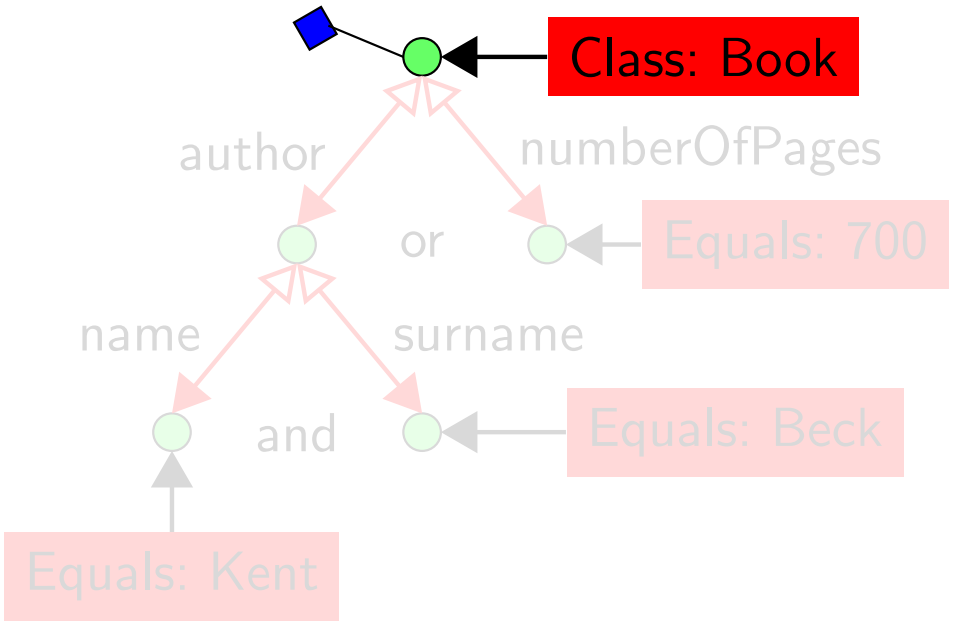
- Niskopoziomowe zapytania db4o
- Budowa drzewiasta
- Idealne do dynamicznego budowania zapytań
- Szybsze niż Native Queries
- Brak kontroli typów (w przeciwieństwie do NQ)
- Nie trzeba się martwić o `null`

SODA Simple Object Data Access



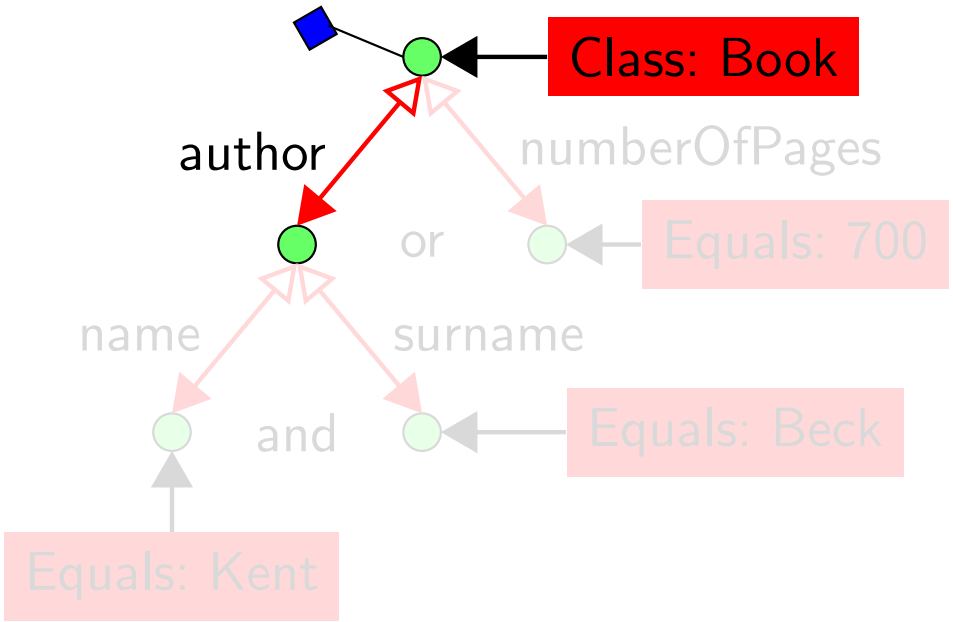

```
query.constrain(Book.class);
Constraint pagesConstraint =
query.descend("numberOfPages").constrain(700);
Constraint surnameConstraint = query.descend("author")
    .descend("surname").constrain("Beck");
query.descend("author").descend("name").constrain("Kent")
    .and(surnameConstraint).or(pagesConstraint);
```

SODA Simple Object Data Access



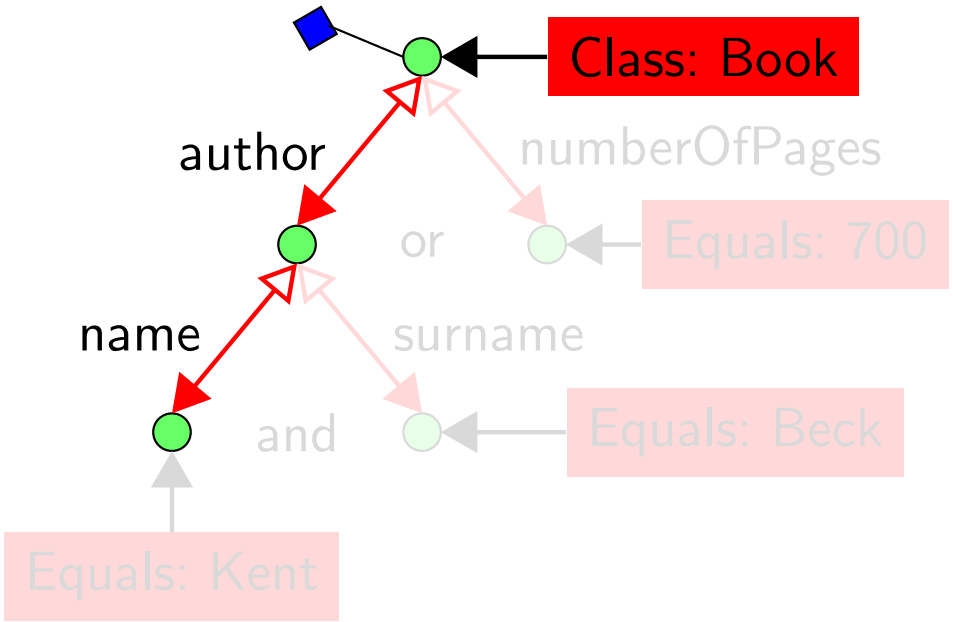

```
query.constrain(Book.class);
Constraint pagesConstraint =
query.descend("numberOfPages").constrain(700);
Constraint surnameConstraint = query.descend("author")
    .descend("surname").constrain("Beck");
query.descend("author").descend("name").constrain("Kent")
    .and(surnameConstraint).or(pagesConstraint);
```

SODA Simple Object Data Access



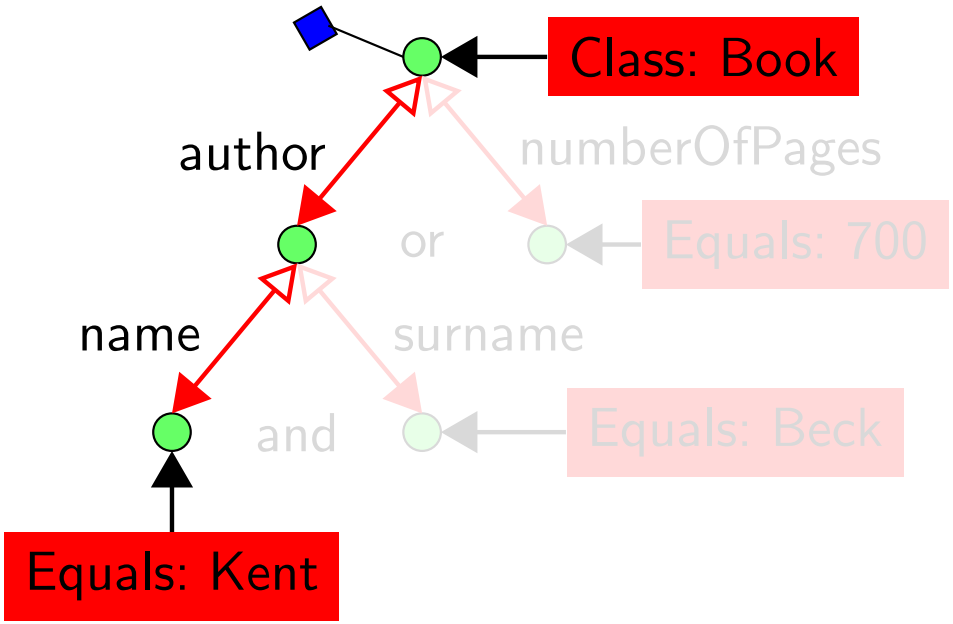

```
query.constrain(Book.class);
Constraint pagesConstraint =
query.descend("numberOfPages").constrain(700);
Constraint surnameConstraint = query.descend("author")
    .descend("surname").constrain("Beck");
query.descend("author").descend("name").constrain("Kent")
    .and(surnameConstraint).or(pagesConstraint);
```

SODA Simple Object Data Access



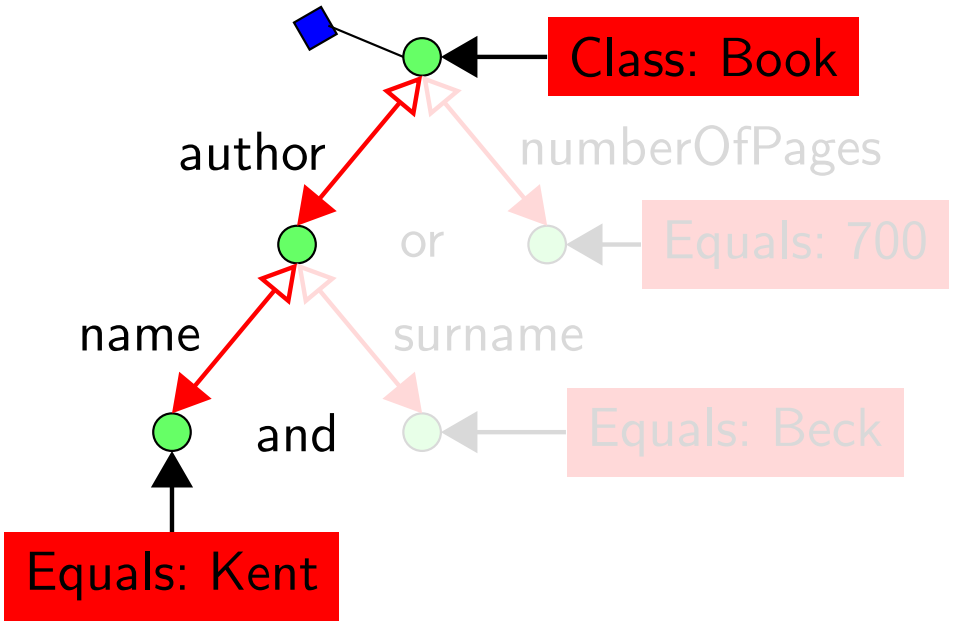

```
query.constrain(Book.class);
Constraint pagesConstraint =
query.descend("numberOfPages").constrain(700);
Constraint surnameConstraint = query.descend("author")
    .descend("surname").constrain("Beck");
query.descend("author").descend("name").constrain("Kent")
    .and(surnameConstraint).or(pagesConstraint);
```

SODA Simple Object Data Access



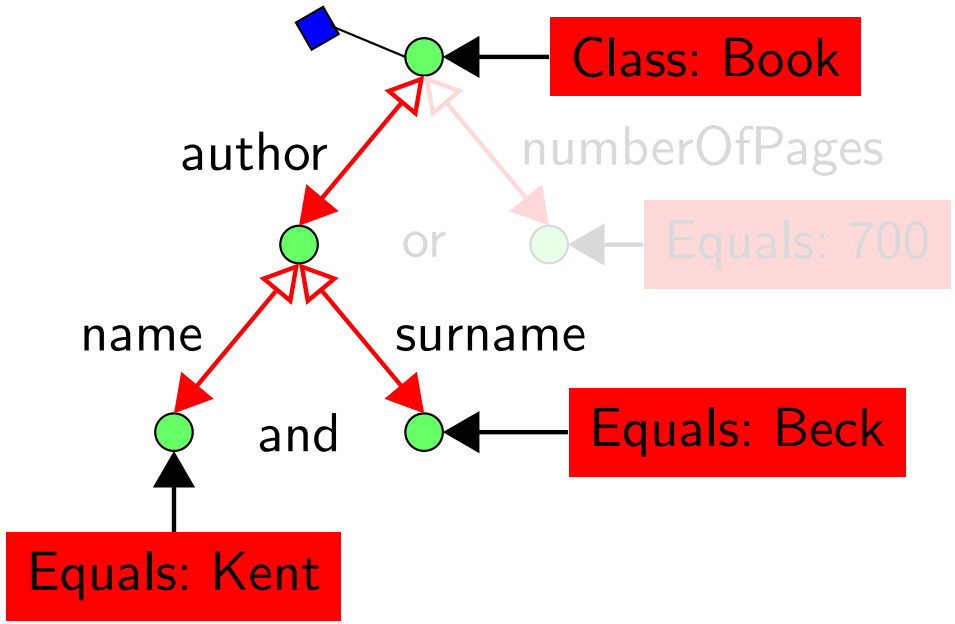

```
query.constrain(Book.class);
Constraint pagesConstraint =
query.descend("numberOfPages").constrain(700);
Constraint surnameConstraint = query.descend("author")
    .descend("surname").constrain("Beck");
query.descend("author").descend("name").constrain("Kent")
    .and(surnameConstraint).or(pagesConstraint);
```

SODA Simple Object Data Access



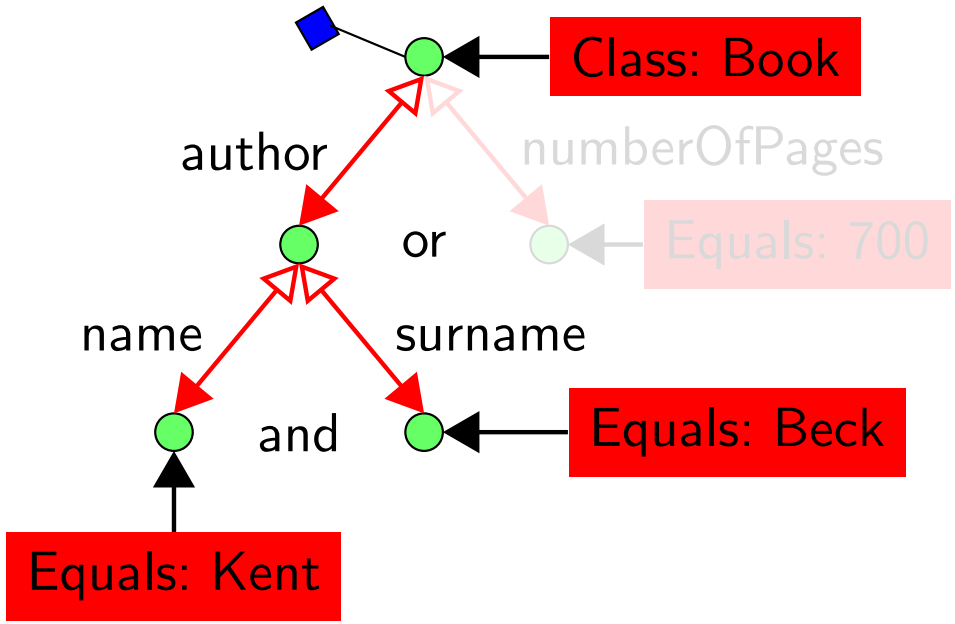

```
query.constrain(Book.class);
Constraint pagesConstraint =
query.descend("numberOfPages").constrain(700);
Constraint surnameConstraint = query.descend("author")
    .descend("surname").constrain("Beck");
query.descend("author").descend("name").constrain("Kent")
    .and(surnameConstraint).or(pagesConstraint);
```

SODA Simple Object Data Access



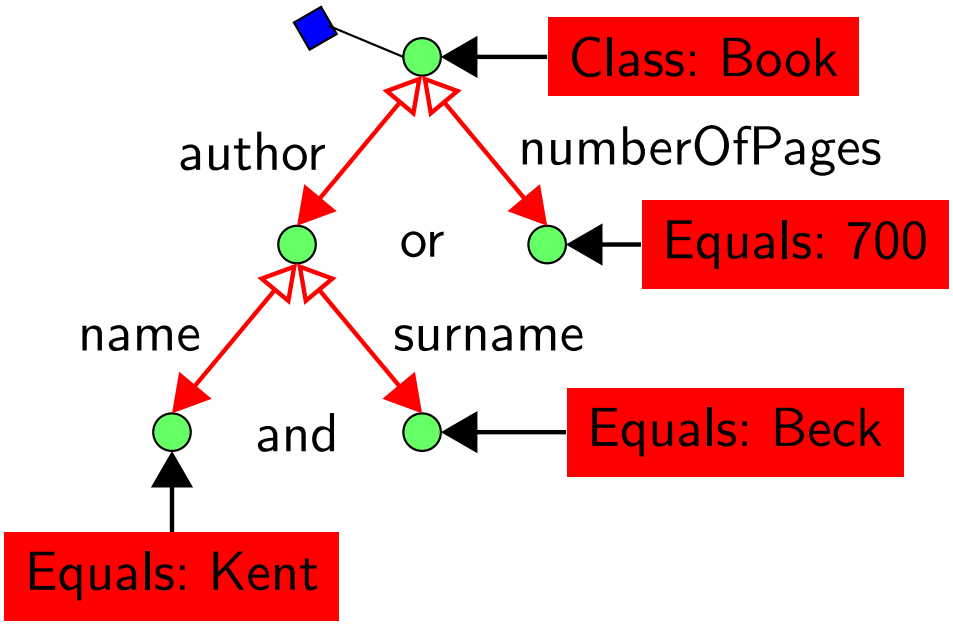

```
query.constrain(Book.class);
Constraint pagesConstraint =
query.descend("numberOfPages").constrain(700);
Constraint surnameConstraint = query.descend("author")
    .descend("surname").constrain("Beck");
query.descend("author").descend("name").constrain("Kent")
    .and(surnameConstraint).or(pagesConstraint);
```

SODA Simple Object Data Access




```
query.constrain(Book.class);
Constraint pagesConstraint =
query.descend("numberOfPages").constrain(700);
Constraint surnameConstraint = query.descend("author")
    .descend("surname").constrain("Beck");
query.descend("author").descend("name").constrain("Kent")
    .and(surnameConstraint).or(pagesConstraint);
```

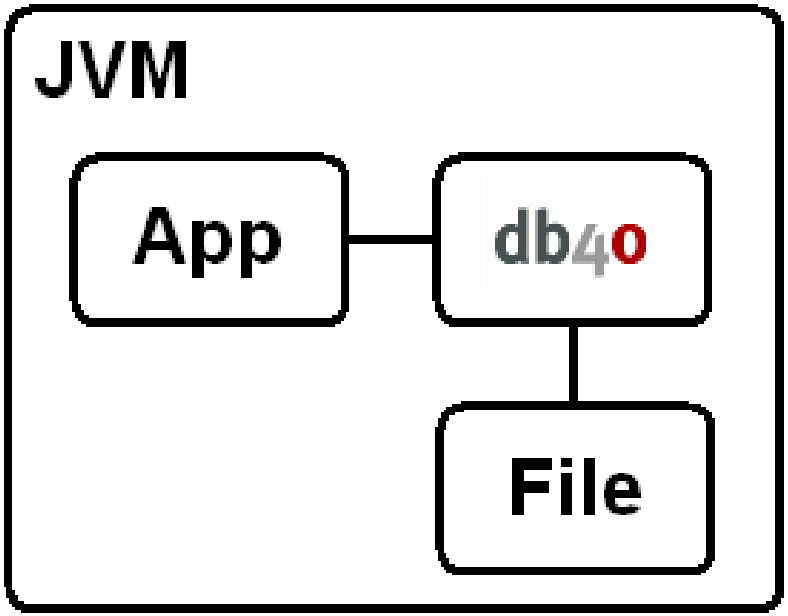
SODA Simple Object Data Access



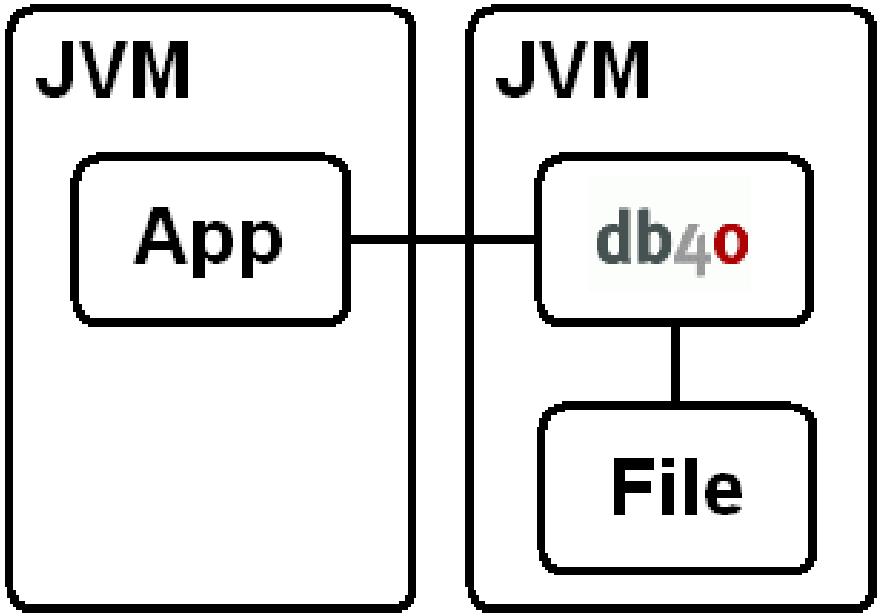

```
query.constrain(Book.class);
Constraint pagesConstraint =
query.descend("numberOfPages").constrain(700);
Constraint surnameConstraint = query.descend("author")
    .descend("surname").constrain("Beck");
query.descend("author").descend("name").constrain("Kent")
    .and(surnameConstraint).or(pagesConstraint);
```

Tryb pracy klient serwer

Tryb pracy na pliku



Tryb pracy klient serwer



Tryb pracy klient serwer

Uruchomienie serwera (klasa `StartServer`):

```
1: ObjectServer db4oServer = Db4o.openServer(FILE, PORT);
2: db4oServer.grantAccess(USER, PASS);
3:
4: // ustawienie nasłuchiwacza obiektów
5: db4oServer.ext().configure().clientServer().↩
       setMessageRecipient(this);
6:
7: try {
8:     if (!stop) {
9:         // wait forever for notify() from close()
10:        this.wait(Long.MAX_VALUE);
11:    }
12: } catch (Exception e) {
13:    e.printStackTrace();
14: }
15: db4oServer.close();
```

Tryb pracy klient serwer

Zatrzymanie serwera (wysłanie obiektu do serwera):

```
 1: public class StopServer {
 2:     // ...
 3:
 4:     db = Db4oUtil.getObjectContainer();
 5:
 6:     // get the messageSender for the ObjectContainer
 7:     MessageSender messageSender = db.ext().configure().
 8:         clientServer().getMessageSender();
 9:
10:     // send an instance of a StopServer object
11:     messageSender.send(new StopServer());
12:
13:     // ...
14: }
```

Tryb pracy klient serwer

Zatrzymanie serwera (odebranie obiektu od klienta):

```
 1: public class StartServer implements MessageRecipient {
 2:     // ...
 3:
 4:     // Implementacja metody z MessageRecipient
 5:     public void processMessage(MessageContext con,
 6:             Object message) {
 7:         if (message instanceof StopServer) {
 8:             close();
 9:         }
10:     }
11: }
```

Tryb pracy klient serwer

Użycie serwera (modyfikacja `Db4oUtil`):

```
1: oc = Db4o.openClient(HOST, PORT, USER, PASS);
```

Reszta taka sama :)

Transakcje

Transakcje:

- ACID
- Read committed
- Domyślnie każda operacja jest pojedyńczą transakcją
- Ale mamy metody `commit()` i `rollback()` :)
- W przypadku wywołania `rollback()` należy odświeżyć „żywe obiekty”:
  `db.ext().refresh(author, Integer.MAX_VALUE);`

Android & db4o

db4objects BY VERSANT

Ale dla db4o $>$ 7.4

WROCŁAW JAVA USER GROUP

Co dalej?

Co jeszcze nie zostało powiedziane:

- Konfigurowanie połaczenia
- Dziedziczenie, kolekcje, tablice
- Kaskadowe wstawianie / modyfikacja / usuwanie
- Głębokość aktywacji
- Transprentna aktywacja
- Pobieranie metadanych
- Indeksowanie pól klas
- Możliwość przechowywania obiektów typu BLOB
- Możliwość replikacji bazy

Więcej informacji

- Strona projektu
- Developer Community
- db4objects and the Dual Licensing Model
- Object Manager 7.4
- db4o-netbeans
- db4o Tutorial for Java
- Db4oUtil (aka. The Easiest Way to Get Started With Db4o)
  `http://developer.db4o.com/Projects/useful_snippets/db4outil_aka._the_easiest_way_to_get_started_with_db4o.html`
  `http://www.spaceprogram.com/knowledge/2006/07/db4outil-aka-easiest-way-to-get.html`
- Using db4o in an Android application
- MapMe (aplikacja na androida wykorzystujaca db4o)
- Słownik terminów z zakresu obiektowości, Kaziemierz Subieta
- Obiektowe Bazy Danych kontra Relacyjne Bazy Danych, Kazimierz Subieta

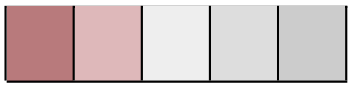
Pytania

Pytania?

Pewnie ktoś się zapytał o wydajność...

Barcelona Benchmarks – wydajność transakcji, pobierającej 100 obiektów z puli 30000 rekordów z 5cio stopniową strukturą dziedziczenia

Barcelona Benchmarks	read	write	query	delete
Native / db4o 6.4	1,0	1,0	1,0	1,0
Hibernate / hsqldb	15,8	3,7	2583,1	4,3
Hibernate / mysql	48,0	26,1	14,4	26,9
JDBC / MySQL	40,8	19,5	9,3	15,8
JDBC / JavaDB	27843,7	20,5	47993,1	17,7
JDBC / HSQLDB*	1,9	1,1	2554,4	0,5
JDBC / SQLite	8,8	519,1	1,1	362,1

fastest  slowest

*JDBC / HSQLDB not ACID transaction safe

http://developer.db4o.com/LearnMore/RealTimePerformance.aspx

Inicjatywa NoSQL na przykładzie db4o.

Marcin Stachniuk
mstachniuk@gmail.com
http://mstachniuk.blogspot.com

14 grudnia 2010

Dziękuję za uwagę!