Meeting #44

EBean Java Persistence

(Dominik Dorn)

CoffeeScript & **SASS**

(Clemens Helm)

Brunch (w/ backbone.js)

(Nik Graf – lightning talk)

EBean

Java Persistence
just
simple & powerful

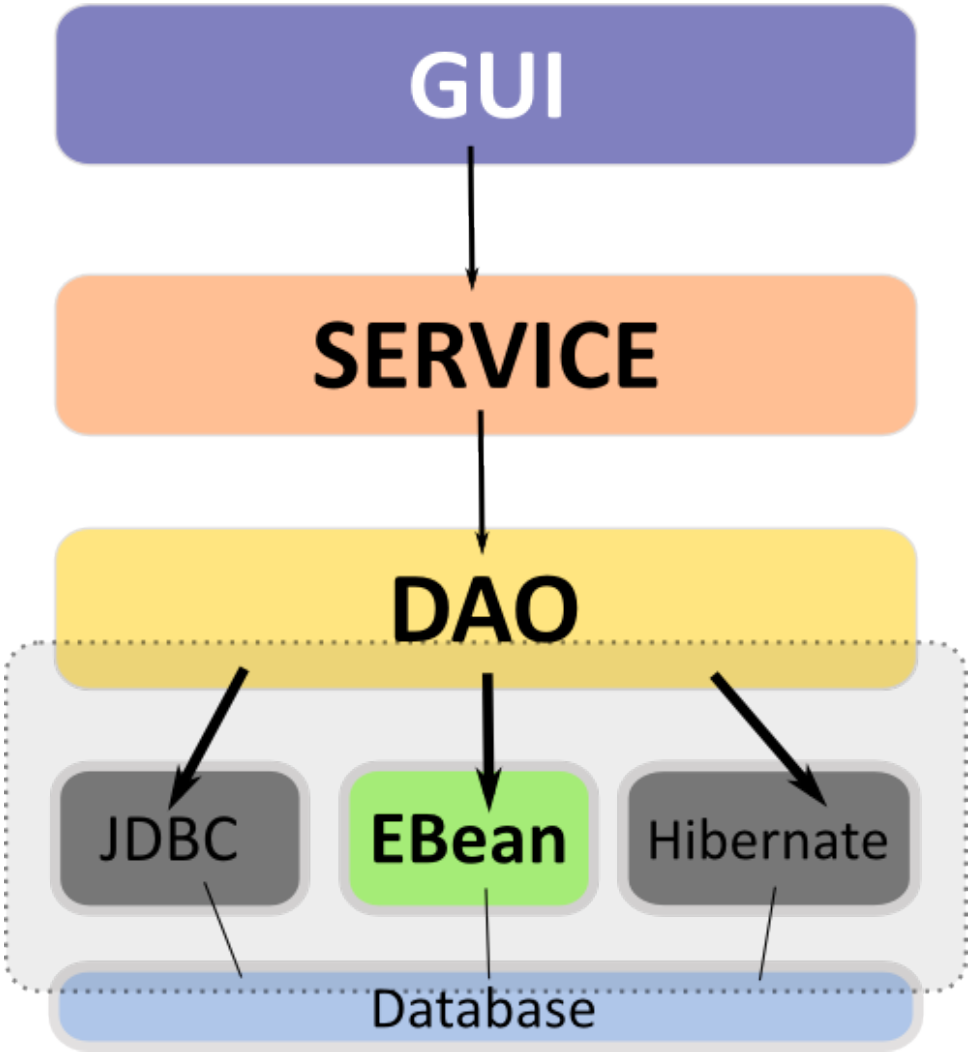
Overview

- What is EBean / Java Persistence + JPA?
- Architecture – where does it fit?
- Partial Objects + Fetch Graphs
- EBean Features
- Code... (maybe :))

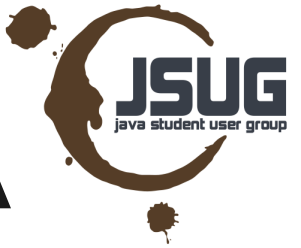
What is Ebean?

- open source **Object Relational Mapping** tool
- simple alternative to **JPA**
  (not like hibernate, eclipselink, etc.)
- “sessionless” API and simple query language
  (explained later)
- Fetch graphs + Partial Objects
- Lazy loading that just works.

Architecture – Where does it fit?

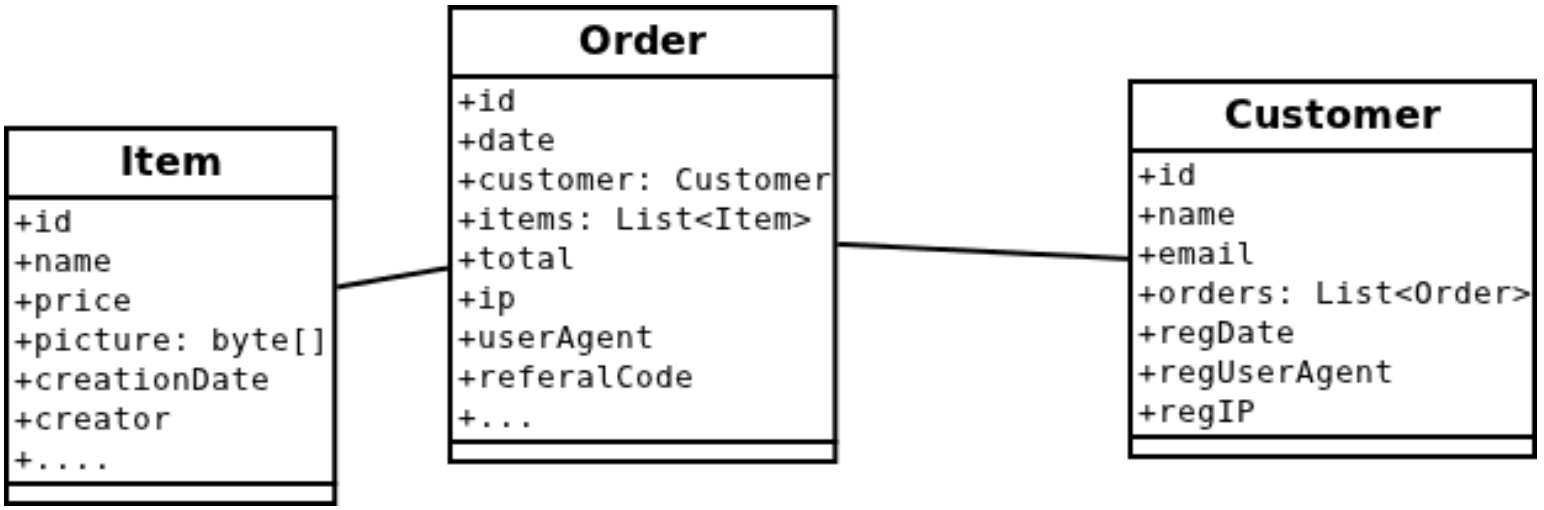


Differences to Hibernate/JPA



- Sessionless
  - No EntityManager / Session / UnitOfWork
    - No merge(), persist(), attach() or detach()
    - Only save() and delete()
- Query Language
  - Slightly different
  - Optimized for fetch graphs and partial objects
- No hacks
  - For wide tables / blobs & clobs etc.

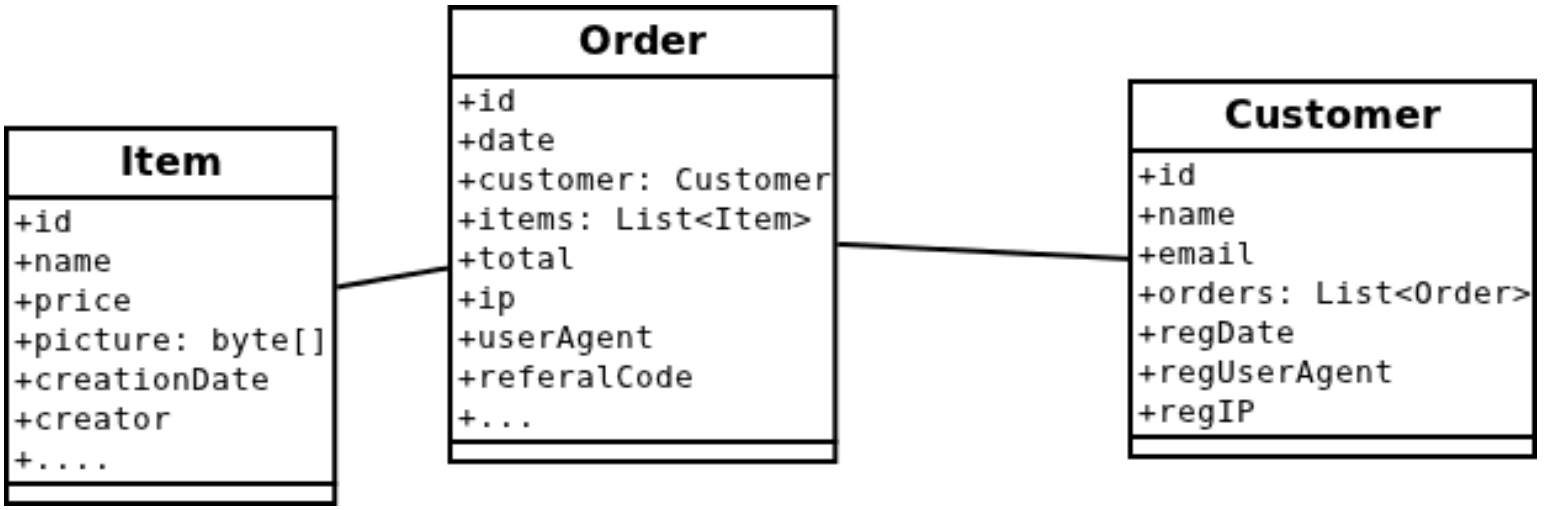
Partial Objects & Fetch Graphs



- **TASK**: Print an Invoice

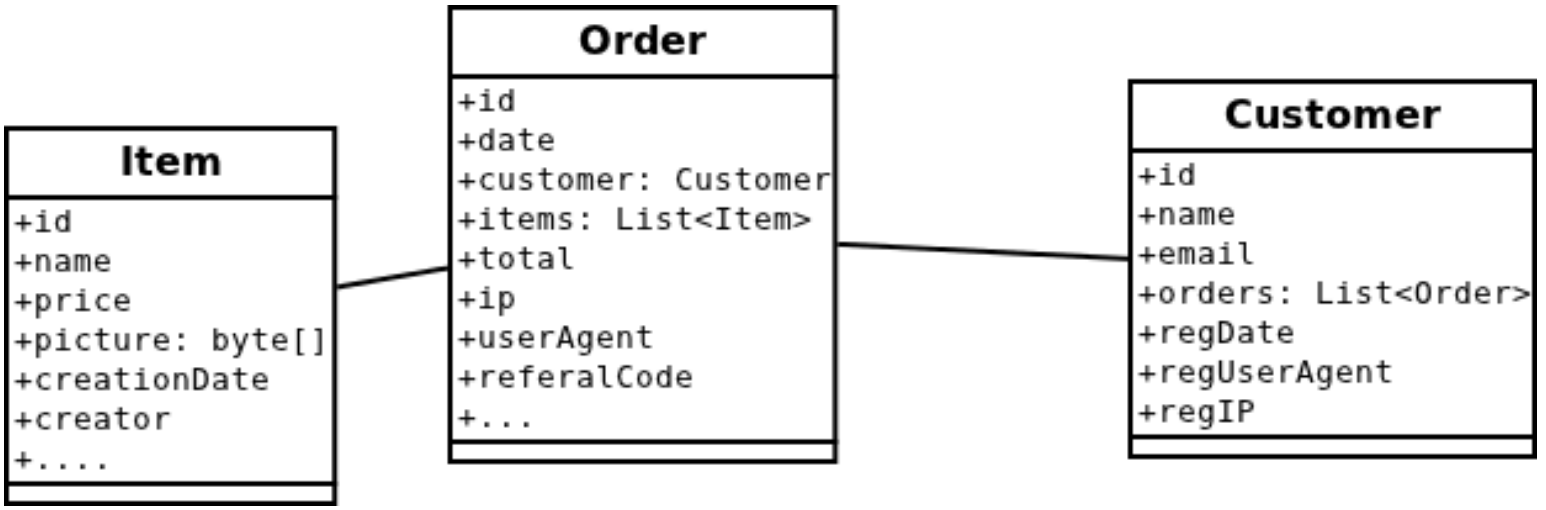
  → get Order(date, items, total) + customer (name, email) + items (name, price)

Partial Objects & Fetch Graphs



- **Hibernate**
  createQuery("SELECT o FROM Order o WHERE o.id = ?")
  → Lazy/eager loading Customer + Items
  → "Huge" traffic and memory requirements (blobs etc.)

Partial Objects & Fetch Graphs



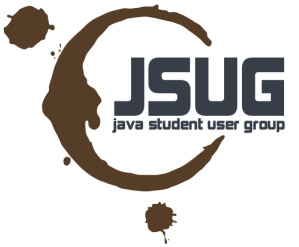
- **EBean**

```
Order o = Ebean.find(Order.class)
    .select("date, total")
    .fetch("customer", "name, email")
    .fetch("items", "name, price")
    .where().eq("id", 42).findUnique();
```

Ebean (nonfunctional) features

- Optimal fetching
  - fast queries right from the start
    - make your DB-Admin happy and your applications fly!
  - Use only the index when possible!
- Still supports **lazy loading**
- Edit/Delete only partially fetched objects
- Transparent Encryption Support
- **Multi DB Support**

Ebean (functional) features



- RawSQL Beans (Reports!)
- AutoFetch – **Automatic Query Tuning**
- Smart Caching
- **Asynchronous Queries**!
- @EnumMapping
- @Formula

  // (Spring & JAX-RS Integration)

  // (Used in PlayFramework 2.0, available in 1.x)

RawSQL Beans

```
@Entity
@Sql
public class OrderAggregate {
	@OneToOne
	Order order;

	Double totalAmount;
	Double totalItems;
//getters setters etc
…
}
```

```
String sql
= " select order_id, o.status, c.id, c.name,
sum(d.order_qty*d.unit_price) as totalAmount"
+ " from o_order o"
+ " join o_customer c on c.id = o.kcustomer_id "
+ " join o_order_detail d on d.order_id = o.id "
+ " group by order_id, o.status ";

RawSql rawSql = RawSqlBuilder.parse(sql)
// map result columns to bean properties
	.columnMapping("order_id", "order.id")
	.columnMapping("o.status", "order.status")
	.columnMapping("c.id", "order.customer.id")
	.columnMapping("c.name", "order.customer.name")
.create();
```

```
Query<OrderAggregate> query = Ebean.find(OrderAggregate.class);
query.setRawSql(rawSql) // we can even add own expressions now...
	.where().gt("order.id", 0).having().gt("totalAmount", 20);

List<OrderAggregate> list = query.findList();
```

AutoFetch

“automatically tune your queries for optimal performance by using profiling information.”

- Analyzes your code at runtime
- Automatically creates fetch profiles
- *Optimal* queries with 0 intervention required

Smart Caching

- e.g. “Read Only and Shared Instances”

```
// Cache countries. Use readOnly=true so unless explicitly
// stated in the query we will
// return read only/shared instances
@CacheStrategy(readOnly=true, warmingQuery="order by name")
@Entity
@Table(name="o_country")
public class Country {
```

Asynchronous Queries

```
// Methods on Query for asynchronous execution

public FutureList<T> findFutureList();
public FutureIds<T> findFutureIds();
public FutureRowCount<T> findFutureRowCount();
```

Asynchronous Queries

```
// example
Query<Order> query = Ebean.find(Order.class);
// find list using a background thread
FutureList<Order> futureList = query.findFutureList();

// do something else in the meantime...

if (!futureList.isDone()){
      // you can cancel the query. If supported by the JDBC
      // driver and database this will actually cancel the
      // sql query execution on the database
      futureList.cancel(true);
}
// wait for the query to finish ... no timeout
List<Order> list = futureList.get();

// wait for the query to finish ... with a 30sec timeout
List<Order> list2 = futureList.get(30, TimeUnit.SECONDS);
```

@EnumValue

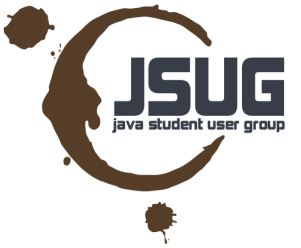
```
public enum UserStatus {
      ACTIVE, INACTIVE, NEW
}
```

```
public enum EnumType
extends java.lang.Enum<EnumType> {
 ORDINAL, STRING
}
```

```
public enum UserStatus {

      @EnumValue(“D”)
      DELETED,

      @EnumValue(“A”)
      ACTIVE,

      @EnumValue(“I”)
      INACTIVE,

      @EnumValue(“N”)
      NEW
}
```

@Formula

```
@Entity
@Table(name="s_user")
public class User {
	@Id
	Integer id;

	@Formula(select="(case when ${ta}.id > 4 then 'T' else 'F' end)")
	boolean caseForm;

	@Formula(select="(select count(*) from f_topic as _b
		where _b.user_id = ${ta}.id)")
	int countAssiged;
```



That's it!

EBean HP
http://www.avaje.org/

twitter: @domdorn

mail: {firstname}@{firstname}{lastname}.com ;)