Open Source
in the Corporate World

Hands-On Hibernate

Brian McCallister

CHARIOT SOLUTIONS

Agenda

- Super Quick Intro
- Thinking in Object Graphs
- Transactions (so Overloaded!)
- Inheritance and Polymorphism
- Hibernate Interceptor
- Filters
- Performance Tuning

Mostly Hibernate 3.0

Open Source
in the Corporate World

Super Quick Intro

Don’t Blink!

CHARIOT SOLUTIONS

Transparent Persistence

- Persists regular objects
- Works with object graphs
- Tracks object state for you
- Isn't really transparent!

The Boilerplate

```
// Only ever need to do once
SessionFactory factory = new Configuration()
                    .configure()
                    .buildSessionFactory();


// A Session is like a JDBC Connection
Session session = factory.openSession();


Transaction tx = session.beginTransaction();
try {
    ... do stuff ...
    tx.commit();
} catch (Exception e) {
    tx.rollback();
    throw e;
} finally {
    session.close();
}
```

Creating Stuff

```
public void testCreate() throws Exception
{
    assertEquals(0, Tools.countRows("beer"));
    Beer coors = new DomesticBeer("Coors", 0.75);

    Transaction tx = session.beginTransaction();
    session.save(coors);
    tx.commit();

    assertEquals(1, Tools.countRows("beer"));
}
```

Reading Stuff, Part I

```
session.createQuery("select b from Beer b " +
                    "where b.price < :price")
           .setBigDecimal("price", new BigDecimal(“1.0”))
           .list();
...

select beer0_.id as id,
       beer0_.brand as brand2_,
       beer0_.price as price2_,
       beer0_.country as country2_,
       beer0_.duty as duty2_,
       beer0_.type as type
from beer beer0_
where (beer0_.price<?)
```

Reading Stuff, Part 2

```
session.createQuery("select b from Beer b " +
                    "  inner join fetch b.truck " +
                    "where " +
                     "  b.truck.driverName = :name")
                  .setString("name", "George")
                  .list();

...

select [FIELDS OMITTED]
from
  beer beer0_ inner join trucks truck1_
    on beer0_.truck_id=truck1_.id,
  trucks truck2_
where (
  truck2_.driver_name=?
  and
  beer0_.truck_id=truck2_.id)
```

Reading Stuff, Part 3

```
Beer beer = (Beer) session.load(Beer.class,
                                new Integer(7));

...

Beer same = (Beer) session.get(Beer.class,
                               new Integer(7));
```

Updating Stuff

```
// Properties
Transaction tx = session.beginTransaction();
Truck t = (Truck) session.load(Truck.class,
                                    new Integer(17));
t.setDriverName("Brian McCallister");
tx.commit();


// Relationships
tx = session.beginTransaction();
Beer beer = (Beer) session.load(Beer.class,
                                   new Integer(15));
Truck truck = (Truck) session.load(Truck.class,
                                      new Integer(17));
truck.getCargo().add(beer);
beer.setTruck(truck);
tx.commit();
```

Deleting Stuff

```
Transaction tx = session.beginTransaction();
Truck truck = (Truck) session.load(Truck.class,
                                   new Integer(17));
session.delete(truck);
tx.commit();
```

Always with the XML...

```
<hibernate-mapping>
    <class name="com.example.Truck" table="trucks"
           discriminator-value="not null">
        <id name="id" type="integer"
            unsaved-value="null">
            <generator class="native"/>
        </id>
        <discriminator column="driver_name"
                       insert="false"/>
        <property name="driverName"
                  column="driver_name"/>
        <bag name="cargo" lazy="true"
             cascade="persist" inverse="true">
            <key column="truck_id"/>
            <one-to-many class="com.example.Beer"/>
        </bag>
        <subclass discriminator-value="null"
                  name="com.example.AutoPilotTruck"/>
    </class>
</hibernate-mapping>
```

Transparent Persistence

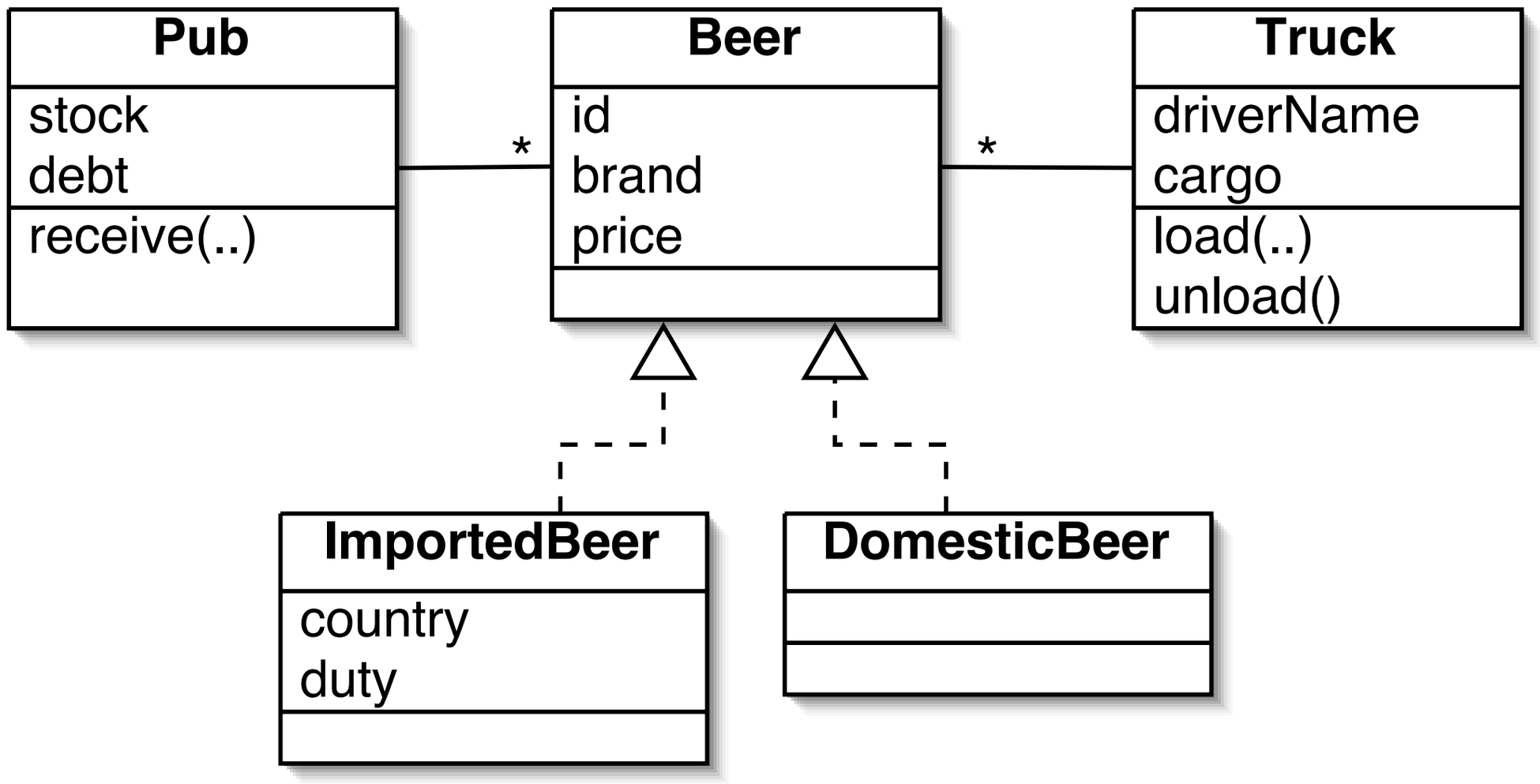
- Persists regular objects
- Works with object graphs
- Tracks object state for you
- Isn't really transparent!
  - But is awfully close!

Open Source
in the Corporate World

Thinking in Object Graphs

CHARIOT SOLUTIONS

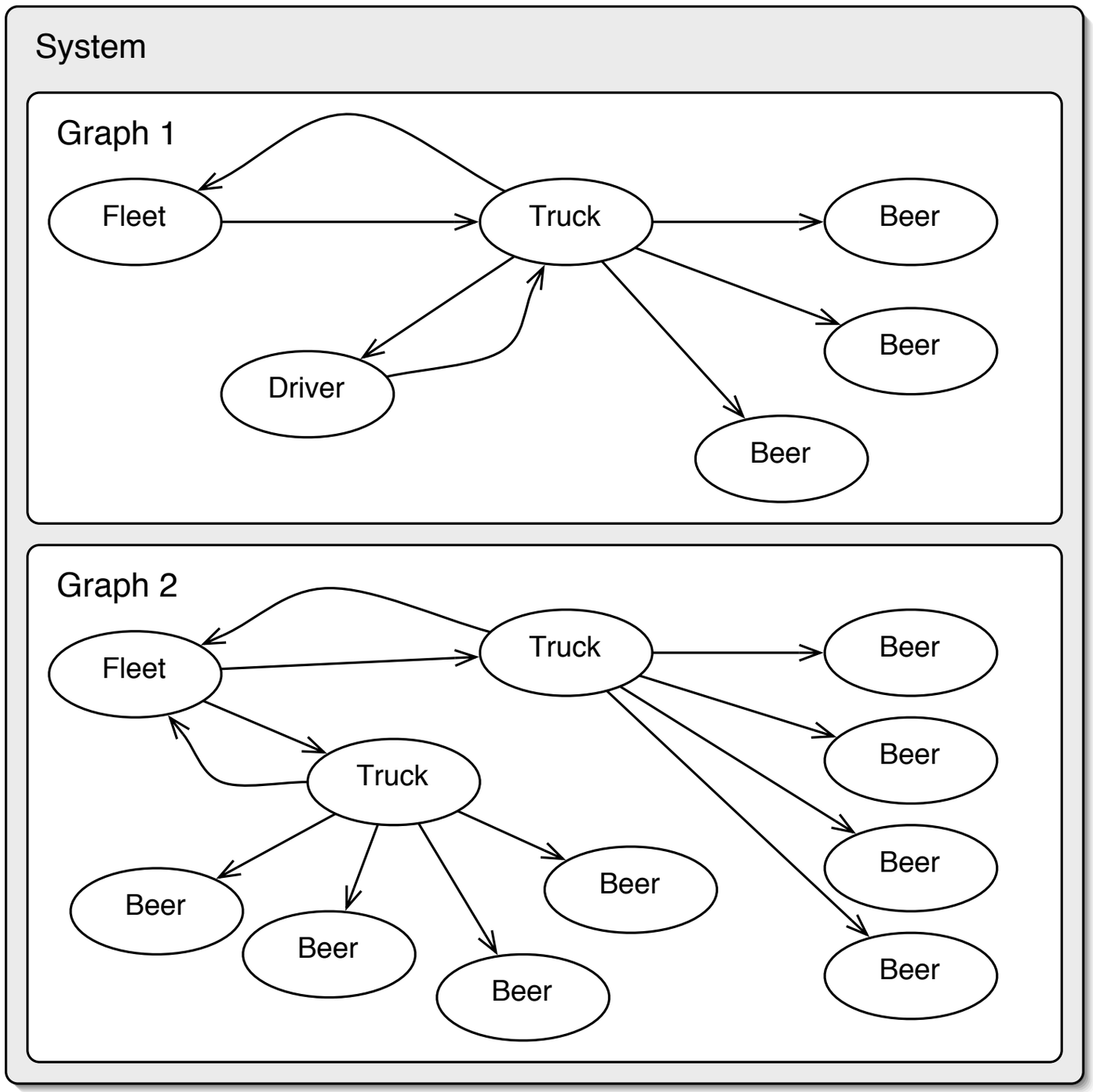
Domain Model



Regular Objects

```
public class DomesticBeer implements Beer {
    private Integer id;
    private String brand;
    private BigDecimal price;
    private Truck truck;

    public DomesticBeer() {
        this("Schlitz", 0.50);
    }

    public DomesticBeer(String brand, BigDecimal price) {
        this.brand = brand;
        this.price = price;
    }

    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }

    ...
```

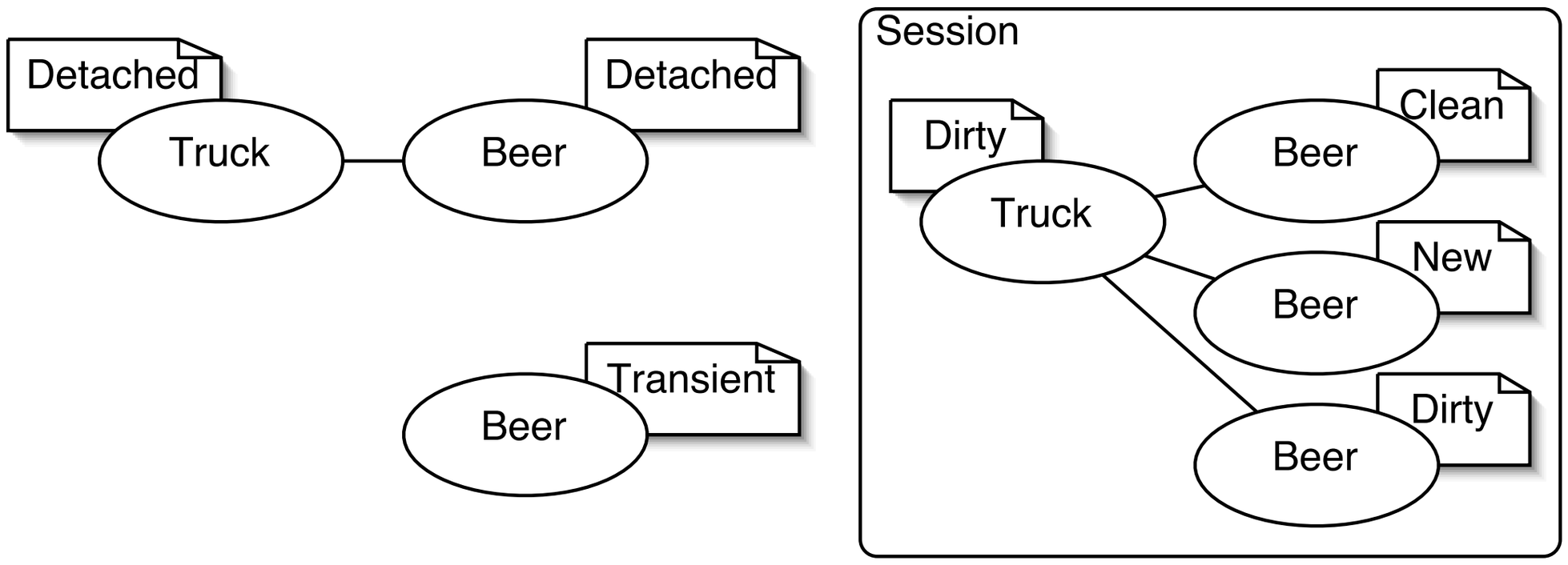
Object Graphs



Persistent States

- Transient
- Persistent Clean
- Persistent Dirty
- Persistent New
- Persistent Deleted
- Detached

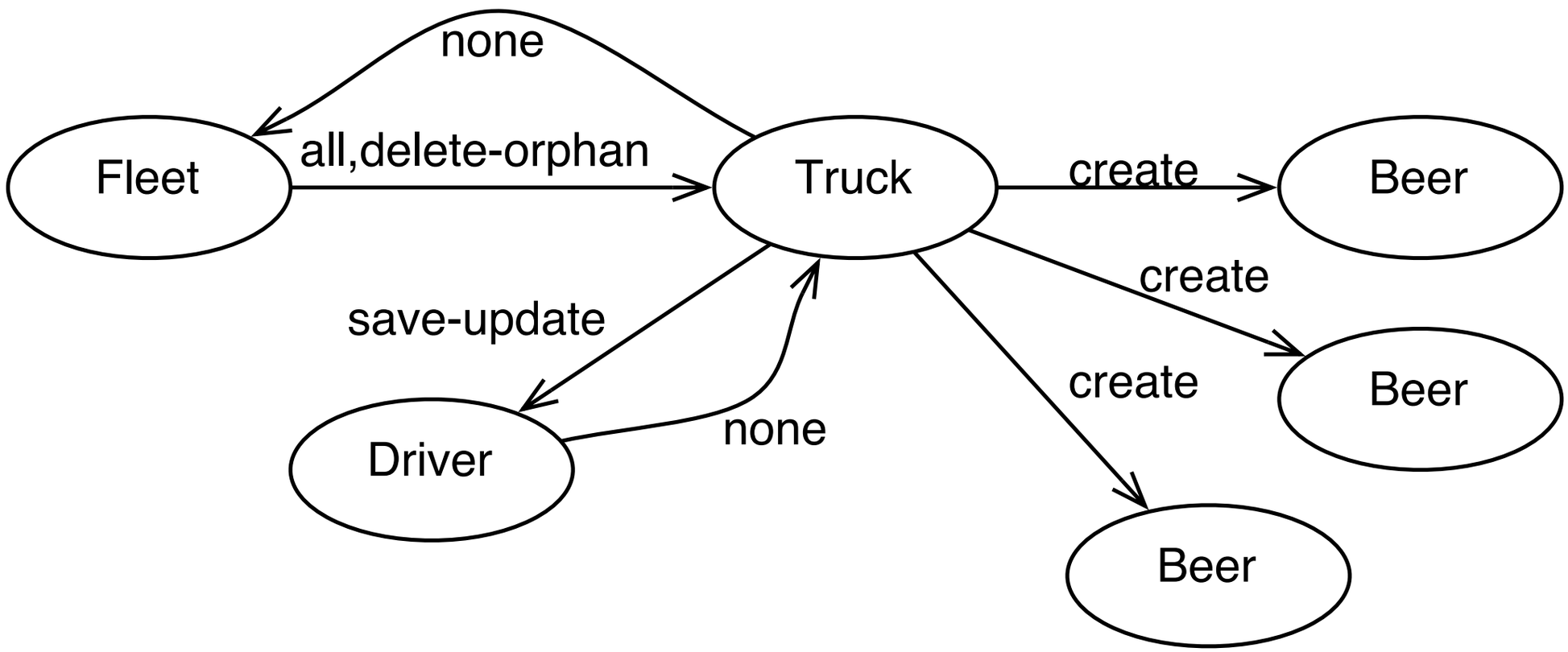
For Example...



Cascades

- Persistence by Reachability
- Define graph traversal behavior
- Configurable per-relationship
- Apply to Session method calls
- Does not apply to state tracking

Cascades and Graphs



Traversing the Graph

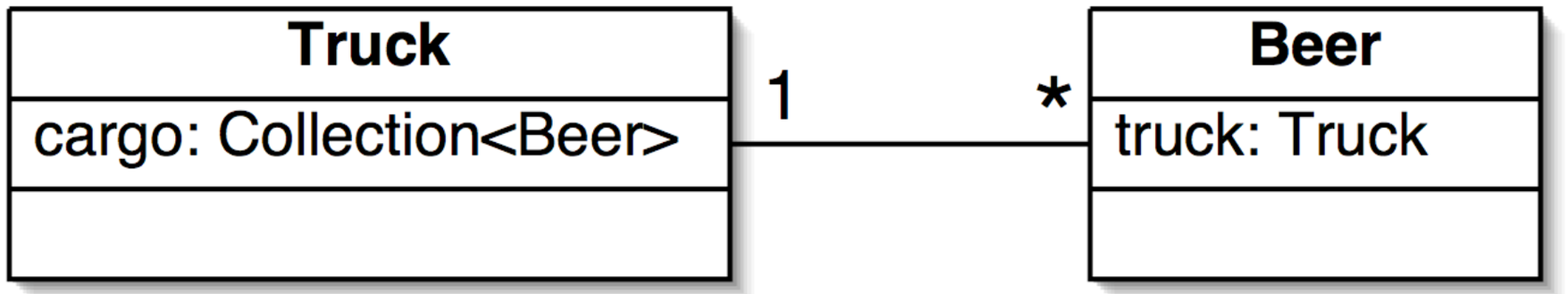
- Cascades
  - create
  - merge
  - delete
  - save-update
  - evict
  - replicate
  - lock
  - refresh
  - delete-orphan
  - all

Relationships

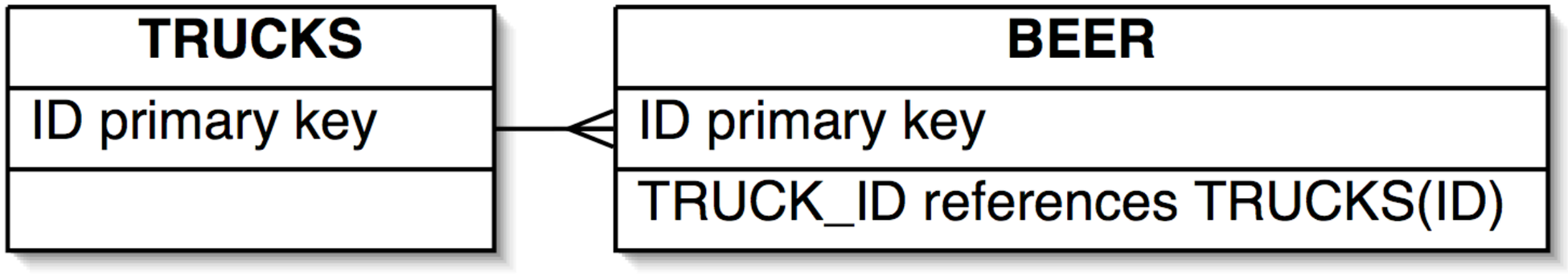
- Object Reference
- Collections
- One-Way
- Two-Way

Relationships

Object Model



Relational Model

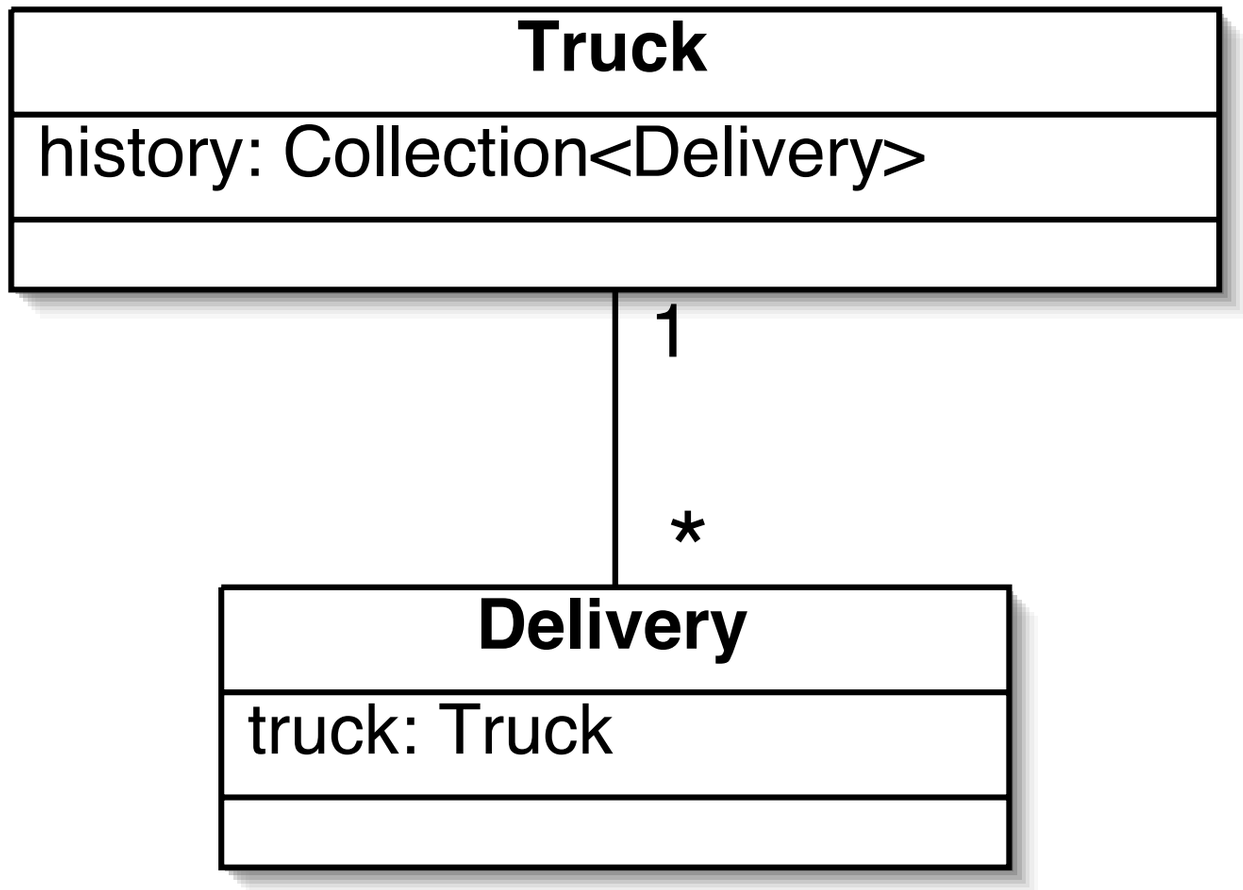


Who Controls Things?

```
<class name="com.example.Truck" .. />
    ..
    <bag name="cargo" inverse="true">
        <key column="truck_id" />
        <one-to-many class="com.example.Beer" />
    </bag>
    ..
</class>

...

<class name="com.example.Beer" .. />
    ..
    <many-to-one name="truck" column="truck_id"/>
    ..
</class>
```

Parent-Child

- Dependent Entities
- Usually < ..cascade=”all, delete-orphan” ..>



Object Graphs

- Need to think in terms of graphs
  - At least a little
- Be aware of nature of relationships
- Be aware of object state

Open Source
in the Corporate World

Transactions (so overloaded)

CHARIOT SOLUTIONS

These are all "Transactions"

- Unit of Work
- Database Transaction
- Local Transaction
- XA Transaction
  - UserTransaction
- Managed Transactions
  - CMT & Spring

Sessions and Changes

- Track changes
- Flushes changes explicitly
- Flushes changes on transaction commits
- Sometimes flushes changes on queries
- Does not flush changes on close
  - by default, can be made to

Sessions and Transactions

- Sessions can span multiple transactions
  - Explicit or Managed
- Sessions can span multiple connections!
  - Long Session

Long Sessions

```
session.setFlushMode(FlushMode.NEVER);
tx = session.beginTransaction();
Truck t = new Truck();
t.getCargo().add(new DomesticBeer());
session.save(t);
tx.commit();
session.disconnect();

assertEquals(0, Tools.countRows("trucks"));
assertEquals(0, Tools.countRows("beer"));

session.reconnect();
tx = session.beginTransaction();
t.setDriverName("Chris");
session.flush();
tx.commit();

assertEquals(1, Tools.countRows("trucks"));
assertEquals(1, Tools.countRows("beer"));
```

Long Sessions

- Long-Running Unit of Work
- Accumulates changes across transactions
- Manually Demarcated
- "Logical Transaction"

However...

- Same Session used elsewhere
  - Session tracking
  - Separate Concerns
- Concurrent Modifications
- Manual Demarcation
- Mixed Strategies
- “Rollback” == Don’t Flush

For Inserts

Reasonable Cascades

```
tx = session.beginTransaction();
Truck t = new Truck();
t.getCargo().add(new DomesticBeer());
tx.commit();
session.close();

// New Request

session = factory.openSession();
tx = session.beginTransaction();
session.save(t);
tx.commit();

assertEquals(1, Tools.countRows("trucks"));
assertEquals(1, Tools.countRows("beer"));
```

Using Long Sessions

- Special Session
- Attached to "Wizard" Context
- Prefer for non-contentious updates
- Do not use by default!

Managing Transactions

- Local
  - session.beginTransaction()
- Bean Managed Transactions
  - UserTransaction
- Container Managed Transactions
  - Declarative on EJBs
- Proprietary Managed
  - Spring, AspectJ, etc

Rules of Thumb

- Same as database transactions
  - It is, really
- Prefer declarative transactions
  - CMT and Spring in particular
- Avoid Long Session unless you need it
  - Then use it!

Open Source
in the Corporate World

Inheritance and Polymorphism

CHARIOT SOLUTIONS

Extents

- Table-Per-Hierarchy
- Table-Per-Class
- Table-Per-Subclass
- Implicit

Type Hierarchy

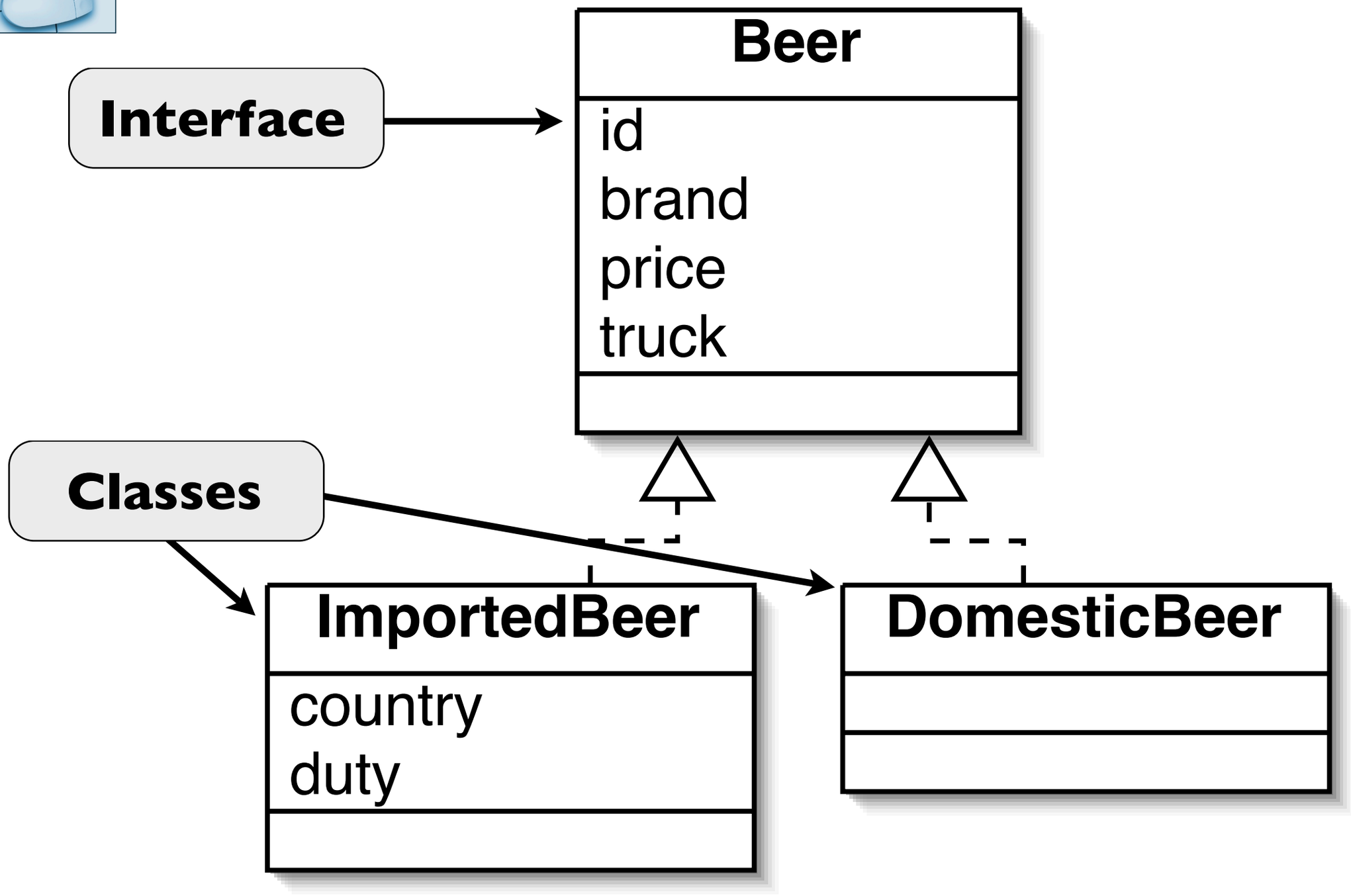


Table Per Hierarchy

BEER
id
brand
price
truck_id
country
duty
type

Table Per Hierarchy

```
<class name="com.example.Beer" table="beer"
       abstract="true">
    <id name="id" type="integer">
        <generator class="native"/>
    </id>
    <discriminator column="type"/>
    <property name="brand"/>
    <property name="price"/>
    <many-to-one name="truck" column="truck_id"/>

    <subclass name="com.example.DomesticBeer"
              discriminator-value="DOMESTIC"/>

    <subclass name="com.example.ImportedBeer"
              discriminator-value="IMPORTED">
        <property name="country"/>
        <property name="duty"/>
    </subclass>
</class>
```

Table Per Subclass

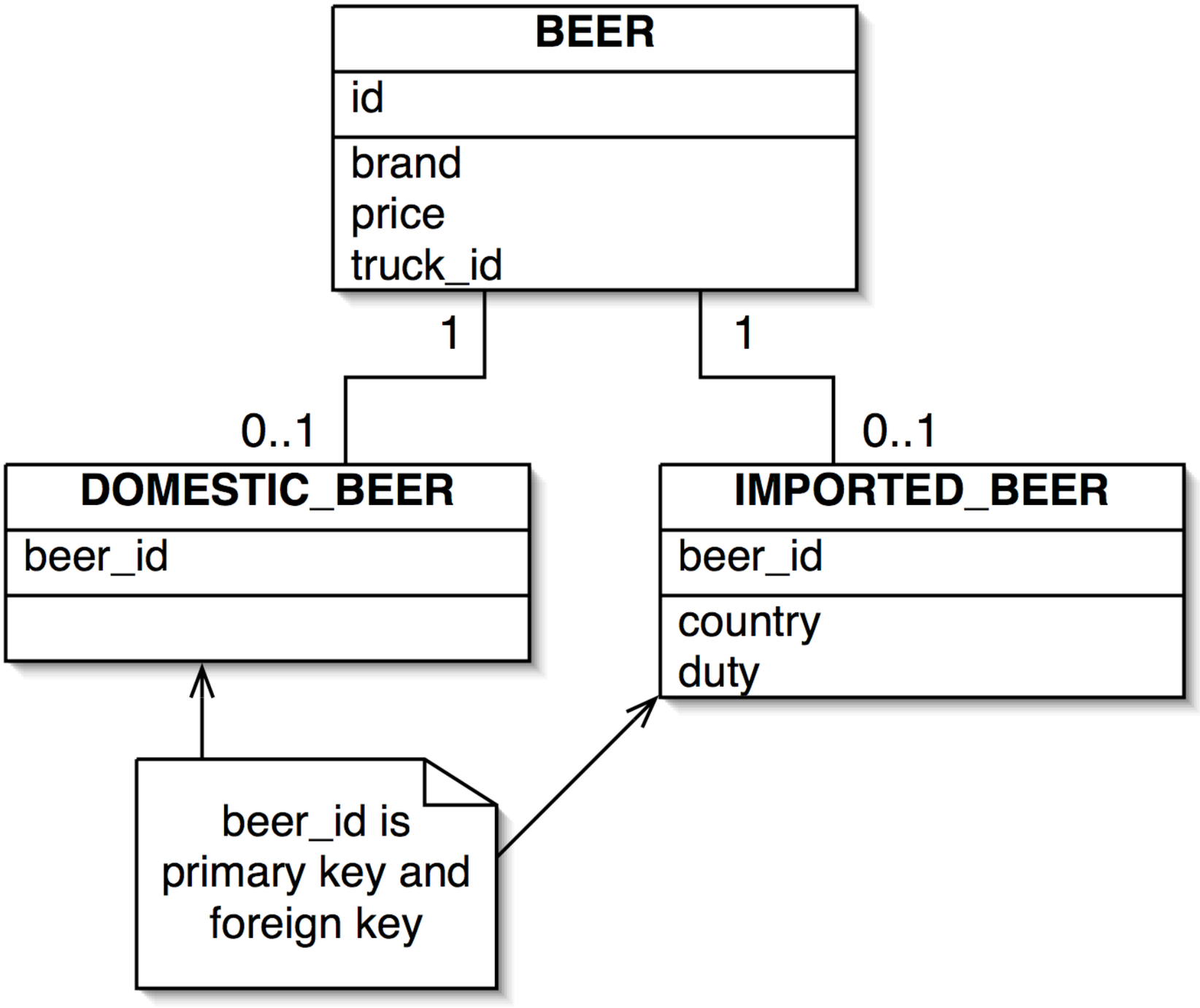


Table Per Subclass

```
<class name="com.example.Beer" table="beer"
        abstract="true" >
   <id name="id" type="integer" unsaved-value="null">
      <generator class="native"/>
   </id>
   <property name="price"/>
   <many-to-one name="truck" column="truck_id"/>

   <joined-subclass name="com.example.DomesticBeer"
                    table="domestic_beer">
      <key column="beer_id"/>
   </joined-subclass>

   <joined-subclass name="com.example.ImportedBeer"
                    table="imported_beer">
      <key column="beer_id" />
      <property name="country"/>
      <property name="duty"/>
   </joined-subclass>
</class>
```

Table Per Concrete Class

DOMESTIC_BEER
id
brand
price
truck_id

IMPORTED_BEER
id
brand
price
truck_id
country
duty

Table Per Concrete Class

```
<class name="com.example.Beer" abstract="true">
   <id name="id" type="integer" unsaved-value="null">
      <generator class="native"/>
   </id>
   <version name="version" column="version"/>

   <property name="brand"/>
   <property name="price"/>
   <many-to-one name="truck" column="truck_id"/>

   <union-subclass name="com.example.DomesticBeer"
                   table="domestic_beer" />

   <union-subclass name="com.example.ImportedBeer"
                    table="imported_beer">
      <property name="country"/>
      <property name="duty"/>
   </union-subclass>
</class>
```

Polymorphic Relationships

- Truck is the same across all of them!
- Map relationship to top of extent
  - The Beer

Relations With Beer

```
<class name="com.example.Truck" .. >
    ...
    <bag name="cargo" cascade="persist">
        <key column="truck_id" />
        <one-to-many class="com.example.Beer"/>
    </bag>
    ...
</class>

...

public class Truck implements Identifiable {
    ...
    private Collection cargo = new ArrayList();
    ...
}
```

Implicit Polymorphism

```
tx = session.beginTransaction();
session.createQuery("from com.example.Identifiable i"
                    + " where i.id = 7")
        .list();
tx.commit();

...

select [columns]
        from beer beer0_
        where (beer0_.id=7)

select [columns]
        from trucks truck0_
        where (truck0_.id=7)
```

Unmapped!

Implicit Limitations

- No Polymorphic Relationships
  - Sort of...
- No "auto-import"
- Query-Per-Type

Open Source
in the Corporate World

Hibernate Interceptor

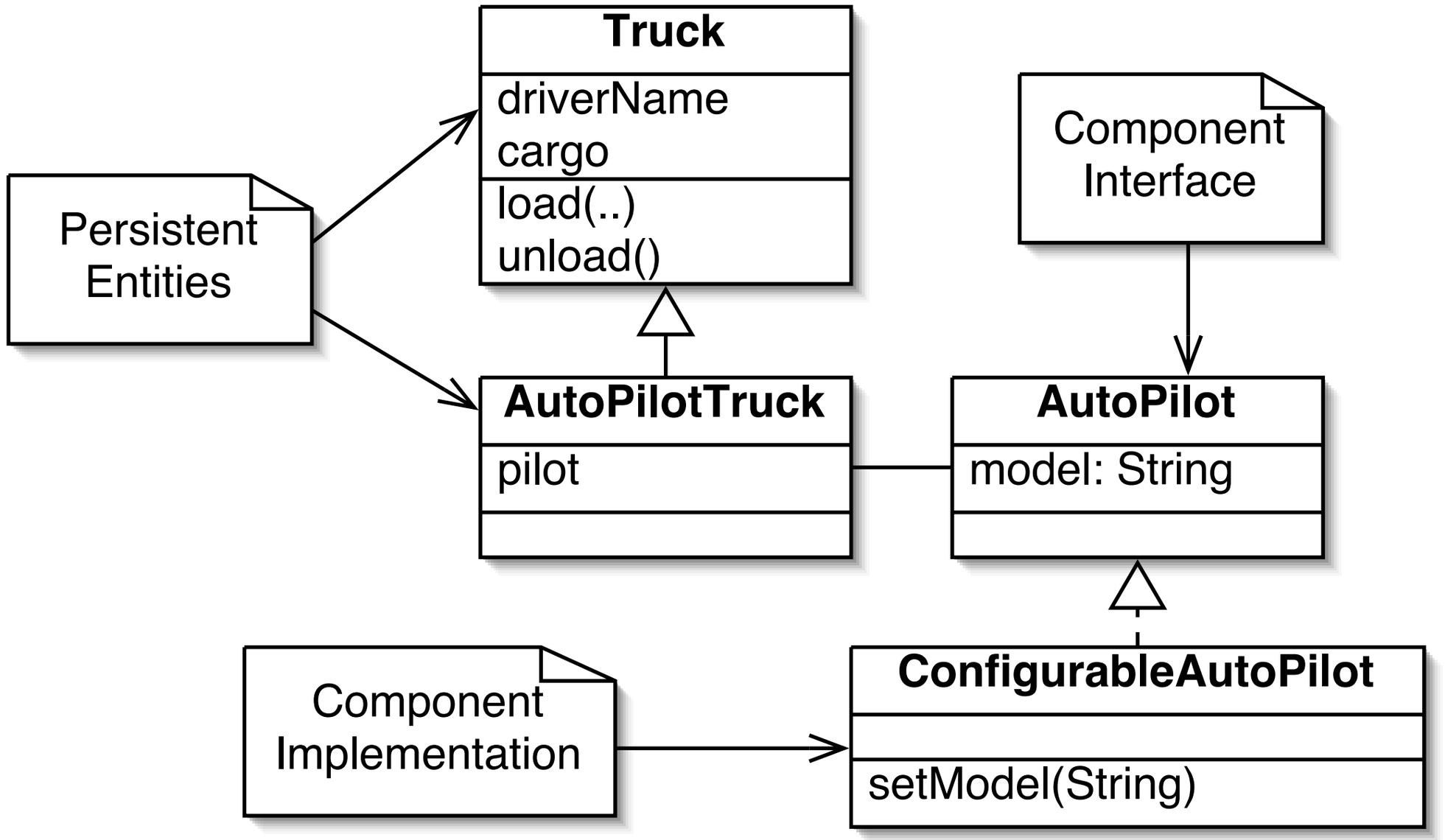
Overriding Behavior

CHARIOT SOLUTIONS

Hibernate Interceptor

- Extension/Override Points
  - Lifecycles
  - Transactions
  - Identity Queries
- 3.0 adds event system

Componentized Domain



We Have AutoPilot!

```
public class AutoPilotTruck extends Truck {

   private AutoPilot pilot;

    // Still need no-arg ctor
    public AutoPilotTruck() {}

    public AutoPilotTruck(AutoPilot pilot) {
        this.pilot = pilot;
    }

    public String getDriverName() {
        return pilot.getModel();
    }

    public void setDriverName(String name) {
        throw new UnsupportedOperationException();
    }
}
```

Umh, What Now?

- AutoPilotTruck depends on a Component
  - The AutoPilot!
- Where do we get the AutoPilot?

Spring is popular, let's try that!

Spring Configuration

```
<beans default-autowire="constructor" >
    <bean class="com.example.Truck"
          singleton="false" />
    <bean class="com.example.DomesticBeer"
          singleton="false" />
    <bean class="com.example.ImportedBeer"
          singleton="false" />

    <bean class="com.example.AutoPilotTruck"
          singleton="false" />

    <bean class="com.example.ConfigurablePilot"
          singleton="true">
        <property name="model">
            <value>Wombat.com SuperPilot</value>
        </property>
    </bean>
</beans>
```

In the Interceptor

```
class FactoryInterceptor implements Interceptor {

  private BeanFactory factory;

  ...

  public Object instantiate(String name,
                            EntityMode mode,
                            Serializable id) {
      Identifiable object =
          (Identifiable) this.factory.getBean(name);

      object.setId(id);
      return object;
  }

  ..

}
```

So What?

- Benefits
  - Container Instantiates Entities
  - Resolves Component Dependencies
- Drawbacks
  - Still need no-arg constructor
  - Complexity
- Neutral
  - Only one Interceptor per session

Open Source
in the Corporate World

Filters

Hibernate 3.0’s Killer Feature

CHARIOT SOLUTIONS

The Problem

- Smart Data Filtering
- Row Level Security
- Context Sensitive Data
  - Which applies across types
  - Which varies across types
- Separating Concerns

Using Filters

```
// 4 beers all in same truck
// Prices .75, .50, 4.00, and 5.00 respectively
assertEquals(4, Tools.countRows("beer"));

session.enableFilter("expensive")
       .setParameter("price", new BigDecimal("2.00"));


List beer = session.createQuery("from Beer").list();
assertEquals(2, beer.size());


Truck t = (Truck) session.createQuery("from Truck")
                         .uniqueResult();
assertEquals(2, t.getCargo().size());
```

Defining Filters

```
<hibernate-mapping>
 <filter-def name="expensive">
  <filter-param name="price" type="java.math.BigDecimal"/>
 </filter-def>
</hibernate-mapping>
```

Filtering Queries

```
<class name="com.example.Beer" .. >
    ...
    <filter name="expensive"
            condition="price > :price" />
</class>
```

Filtering Collections

```
<class name="com.example.Truck" .. >
    ...
    <bag name="cargo" cascade="persist">
        <key column="truck_id" />
        <one-to-many class="com.example.Beer" />
        <filter name="expensive"
                condition="price > :price"/>
    </bag>
    ...
</class>
```

Using Filters (Again)

```
// 4 beers all in same truck
// Prices .75, .50, 4.00, and 5.00 respectively
assertEquals(4, Tools.countRows("beer"));

session.enableFilter("expensive")
       .setParameter("price", new BigDecimal("2.00"));


List beer = session.createQuery("from Beer").list();
assertEquals(2, beer.size());


Truck t = (Truck) session.createQuery("from Truck")
                         .uniqueResult();
assertEquals(2, t.getCargo().size());
```

Filter Gotchas

- Filters are applied at SQL level
  - Cannot add joins
  - Cannot reference relations easily
- Careful with Collections
  - Apply to collections as well as types
- Session Cache
  - If collection is in Session unfiltered, it remains there

Open Source
in the Corporate World

Performance Tuning

The Very Basics

CHARIOT SOLUTIONS

Performance Tuning

- Load Strategies
  - Lazy, Eager, Fetch
- Grokking the Cache
  - Level 1, Level 2
  - Query Cache
  - Distributed Cache

Load Strategies

- Lazy Load by Default
  - It is the default in 3.0, yea!
- Queries are Different
- Join on object references
- Join on one collection
- Beware the Detach

Fetches and Queries

```
<class name="com.example.Beer" ... >
    ...
    <many-to-one name="truck" column="truck_id"
                 fetch="join" />
</class>

...

tx = session.beginTransaction();
Beer beer = (Beer) session.createQuery("from Beer")
                          .uniqueResult();
Truck t = beer.getTruck();
assertFalse(Hibernate.isInitialized(t));
t.getDriverName();
assertTrue(Hibernate.isInitialized(t));
tx.commit();
```

Fetches and Queries

```
tx = session.beginTransaction();

Beer beer = (Beer) session
       .createQuery("from Beer b join fetch b.truck")
       .uniqueResult();

Truck t = beer.getTruck();

assertTrue(Hibernate.isInitialized(t));

tx.commit();
```

Where Fetch Applies

- Proxy Loading
- Session#get
- Session#load

Caching

- First Level Cache
- Second Level Cache
- Query Cache
- Distributed Caches

First Level Cache

- The Session
- Everything loaded by this session

Second Level Cache

hibernate.cfg.xml

```
<class-cache class="com.example.Truck"
                usage="read-write"/>
<collection-cache collection="com.example.Truck.cargo"
                    usage="nonstrict-read-write"/>
```

Truck.hbm.xml

```
<class name="com.example.Truck" ... >
    <cache usage="read-write" />
    <bag name="cargo" cascade="persist"
            inverse="false">
        <cache usage="read-write"/>
        <key column="truck_id" />
        <one-to-many class="com.example.Beer" />
    </bag>
</class>
```

Second Level Cache

- Caches flat data
- Entities and Collections are separate
- Collections require Transactional
- Configuration and Mapping flexibility

Query Cache

hibernate.cfg.xml

```
<hibernate-configuration>
    <session-factory>
        <property name="cache.use_query_cache">
            true
        </property>
        ...
    </session-factory>
</hibernate-configuration>
```

Using it...

```
session.createQuery("select b from Beer b " +
                    "  join fetch b.truck " +
                    "where " +
                    "  b.truck.driverName = :name")
                .setCacheable(true)
                .setString("name", "George")
                .list();
```

Query Cache

- Caches ID's for query results
- Uses second level cache to materialize
- Issues query for the rest

Usage

- read-only
  - Most performant
- nonstrict-read-write
  - If low risk/cost of collision or dirty read
- read-write
  - No serializable tx, Beware of clusters
- transactional
  - Safest, works correctly with transactions

Clusters

- Local Caching in Clusters
- Distributed Caching in Clusters
- Small
  - JBoss TreeCache, SwarmCache, OSCache, ActiveSpace
- Large
  - Memcached*, Commercial

* Custom Provider

Open Source
in the Corporate World

Questions?

CHARIOT SOLUTIONS

What We Looked At

- Super Quick Intro
- Thinking in Object Graphs
- Transactions (so Overloaded!)
- Inheritance and Polymorphism
- Hibernate Interceptor
- Filters
- Performance Tuning

Open Source
in the Corporate World

Thank You!

http://www.chariotsolutions.com/

CHARIOT SOLUTIONS