Advanced Hibernate

EHSDI

eBuzima

Configuration

- `new Configuration()`

- This object is only needed at startup
- Creating a new instance causes Hibernate to look for configuration files in the project root
- Configuration files specify things like:
  - Database server details
  - Connection pooling options
  - Mapping files

SessionFactory

- Created from a configuration at startup, e.g.

```
SessionFactory factory
   = new Configuration.configure().buildSessionFactory()
```

- Will produce sessions for working with the database specified in the configuration

- Thread-safe, used by all application threads

Sessions

- Intended for a single unit of work
- Can be created from a session factory, e.g.

```
Session session = factory.openSession();
```

- And flushed and closed when the work is done...

```
session.flush();
session.close();
```

- Is not thread-safe!

Transactions

- Typically a session consists of transactions
- A transaction is a small unit of work, to be executed in isolation from others
- It's all-or-nothing – either completes entirely or should have no effect at all
  - For example – moving money in two steps:
    - Debit one account $100
    - Credit another account $100

  This needs to happen in a single transaction, or the money could be lost

Transactions

- Created from a session, e.g.

```
Transaction tx = session.beginTransaction();
```

- ...and committed when the work is done

```
tx.commit();
```

- ...or rolled back if something went wrong

```
tx.rollback();
```

- Equivalent to START TRANSACTION, COMMIT and ROLLBACK in SQL

Session management

- We can let Hibernate manage sessions for us
- `getCurrentSession` gives a session which is associated with a single transaction, e.g.

```
Session session = factory.getCurrentSession();
session.beginTransaction();

// Do something!

session.getTransaction().commit();
```

Session is automatically flushed and closed

Persistent classes

- Is a class with an associated mapping XML file (or annotations)

```
<hibernate-mapping package="eh203">
  <class name="Person" table="person">
    <id name="personId" column="person_id">
      <generator class="native" />
    </id>
    <property name="name" type="string" column="name" />
  </class>
</hibernate-mapping>
```

```
class Person {
  ...
  public int getPersonId() {...}
  public String getName() {...}
}
```

Persistent class instance states

- **Transient**
  - Not as yet associated with a session
  - No primary key value
- **Persistent**
  - Obtained by load, get or a query
  - Associated with a session
  - Has a primary key value
- **Detached**
  - Was associated with a session once – session may have been closed
  - Has a primary key

Instance states example

```
Session session = factory.openSession();
session.beginTransaction();

Person p1 = new Person("Bob Smith");

Person p2 = (Person)session.load(Person.class, 4);

session.getTransaction().commit();
session.close();

Person p3 = p2;
```

p1 is *transient*

p2 is *persistent*

p2 is now *detached* because session is closed

Object identity

- Database identity means that 2 objects have the same primary key, e.g.
  - `p1.getPersonId.equals(p2.getPersonId())`
- JVM identity means that 2 objects reference the same location in memory, e.g.
  - `p1 == p2`
- Hibernate guarantees only for persistent objects that database identity is equivalent to JVM identity

Creating persistent objects

- Load creates an instance of a persistent class and loads its values from the row with that id

```
class Person {
  ...
  public int getPersonId() {...}
  public String getName() {...}
}
```

person_id	name
1	Bob Smith
2	Jack Green
3	Anne Tang

```
load(Person.class, 2)
```

Person
person_id = 2
name = "Jack Green"

Proxy objects

- What if the Person class has a collection to be loaded from another table? e.g.

```
class Person {
  ...
  public Set<Address> getAddresses() {...}
}
```

foreign key

person_id	name
1	Bob Smith
2	Jack Green
3	Anne Tang

address_id	street	person_id
1	10 Main St.	2
2	123 Time Ave.	2
3	64 Oak Rd.	3

- Should that be loaded as well?

Proxy objects

- If Hibernate was to load every related row/object from the database, it could potentially load the entire database
- Maybe the user only needs one field from a row?

- Hibernate solves this problem using ***proxy objects***

Proxy objects

- The proxy object only loads data from the database when it is actually needed, e.g.

```
Person p = (Person)session.load(Person.class, 2);
```

Person *(Proxy)*
person_id = 2
name = ?

```
String name = p.getName();
```

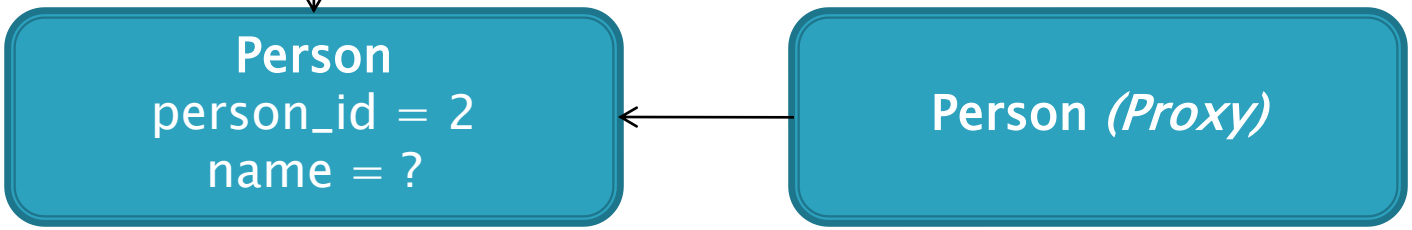
Person *(Proxy)*
person_id = 2
name = "Jack Green"

← Load ←

Proxy objects

- A proxy object is a dynamically created subclass of the persistent class, e.g.

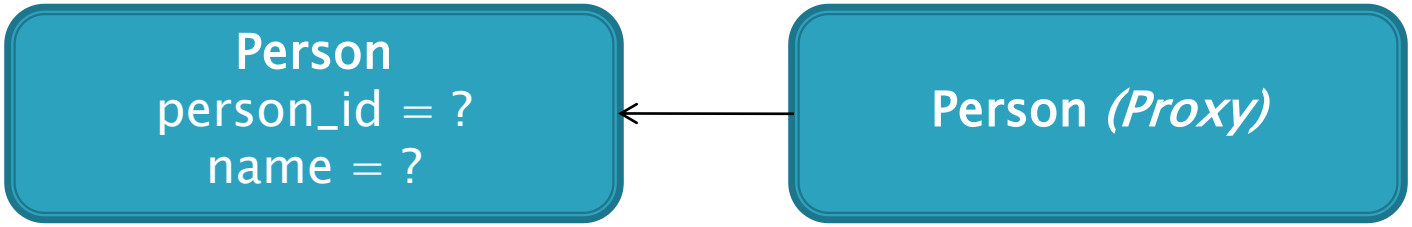
```
Person p = (Person)session.load(Person.class, 2);
```



Because of polymorphism, calling `p.getName()` actually calls `getName()` on the proxy class

Proxy objects

- The overridden methods in the proxy object will load the object if it hasn't already been loaded, e.g.




```
...
public String getName() {
   return name;
}
```

```
...
public String getName() {
   if (!loaded)
      loadFromDatabase();

   return super.getName();
}
```

Proxy objects

- Proxy objects are essential to Hibernate, but can lead to problems if you don't understand how they work, e.g.

```
Session session = sessionFactory.getCurrentSession();
session.beginTransaction();

Person p = (Person)session.load(Person.class, 2);

session.getTransaction.commit();

p.getName();
```

causes session to be closed

throws an exception because session has been closed, so proxy object can't load data from the database

Being lazy

- This behavior of delaying actual loading of data until it is needed is called ***lazy initialization***
- Sometimes it is preferable to disable this behavior for a specific class or property, e.g.

```
<hibernate-mapping package="eh203">
  <class name="Person" table="person" lazy="false">
    ...
```

- But this must be used with caution!

Get vs load

- Both of these methods fetch an object from a database record, however...
- `session.load(...)`
  - Returns a proxy object and delays loading of data until it is requested
  - Throws an `ObjectNotFoundException` if record with that ID doesn't exist
- `session.get(...)`
  - Loads from the database immediately
  - Returns `null` if record with that ID doesn't exist

References

- Websites
  - http://hibernate.org