CS544 EA

Hibernate

Lazy and Eager

What is Lazy (or Eager)?

- Lazy means it does not load it
  - Until it absolutely needs it
  - By default all **Collections** (*ToMany) **are lazy**

- Eager gets loaded right away
  - As soon as it knows about it
  - By default all **references** (*ToOne) **are eager**

Changing FetchType

- It's **possible to change**
  - References to LAZY
  - collections to EAGER
- Generally not needed

```
@Entity
public class Customer {
      @Id
      @GeneratedValue
      private Long id;
      private String firstName;
      private String lastName;
      @ManyToOne(fetch=FetchType.LAZY)
      private SalesRep salesRep;
      @OneToMany(fetch=FetchType.LAZY)
      @JoinColumn
      private List<Book> books
            = new ArrayList<>();
```

LAZY

LAZY or EAGER

- Some say that all associations should be lazy
  - EAGERly loaded objects may never be used
- Usually Hibernate loads *ToOne with Joins
  - Less expensive then separate selects
  - Still takes overhead, wasted if object not used

- On the whole **you can be safe never modifying**
  - Premature optimization is the root of all evil

@LazyCollection

- A Hibernate extension that:
  - Can be useful **for big collections**

- By default the entire collection is retrieved for:
  - .size(), .isEmpty(), .contains()
  - Instead of **using the DB** to count / check
  - ‘Extra lazy’ fixes that

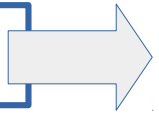
@LazyCollection

```java
import org.hibernate.annotations.LazyCollection;
import org.hibernate.annotations.LazyCollectionOption;

@Entity
public class Customer {
     @Id
     @GeneratedValue
     private Long id;
     private String firstName;
     private String lastName;
     @OneToMany(cascade=CascadeType.ALL)
     @JoinColumn
     @LazyCollection(LazyCollectionOption.EXTRA)
     private List<Movie> movies
          = new ArrayList<>();
```

Extra Lazy

```java
Customer c = em.find(Customer.class, 1L);
System.out.println(c.getMovies().size());
```



```
Hibernate:
    select
        customer0_.id as id1_1_0_,
        customer0_.firstName as firstNam2_1_0_,
        customer0_.lastName as lastName3_1_0_,
        customer0_.salesRep_id as salesRep4_1_0_
    from
        Customer customer0_
    where
        customer0_.id=?

Hibernate:
    select
        count(id)
    from
        Movie
    where
        movies_id =?
```

Lazy Properties

- It is possible to make **individual properties lazy**
  - Needs Property access (getters)
  - @Basic(fetch=FetchType.LAZY) (on getters)

- **Needs ByteCode instrumentation** to work
  - Rewrites your getters (after compilation) for ability to load data

- Generally not recommended
  - **DTO projection is better** / easier solution

Hibernate Documentation calls it a “Marketing Feature”

ByteCode Instrumentation Ant file

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="ByteCodeInstrument" default="instrument">
  <description>Byte Code instrument example</description>
  <property name="src" location="src" />
  <property name="build" location="bin" />

  <target name="compile">
    <javac srcdir="${src}" destdir="${build}" />
  </target>

  <target name="instrument" depends="compile">
    <taskdef name="instrument"
        classname="org.hibernate.tool.instrument.cglib.InstrumentTask">
      <classpath>
        <fileset dir="c:/hibernatetraining/libraries/">
          <include name="**/*.jar" />
        </fileset>
      </classpath>
    </taskdef>
    <instrument verbose="true">
      <fileset dir="${build}/when/properties/">
        <include name="**/*.class" />
      </fileset>
    </instrument>
  </target>
</project>
```

Lazy Properties

```
@Entity
public class Book {
  ...

  @Id
  public String getIsbn() { return isbn; }
  public String getTitle() { return title; }
  public String getAuthor() { return author; }

  @Basic(fetch=FetchType.LAZY)
  public java.sql.Clob getSummary() {
    return summary;
  }
  @Basic(fetch=FetchType.LAZY)
  public java.sql.Blob getCover() {
    return cover;
  }
```

Annotations on getters

```
Book b = (Book)session.get(Book.class, "978-0545139700");
System.out.println(b.getTitle());

java.sql.Clob sumData = b.getSummary();
int length = (int)sumData.length();
System.out.println(sumData.getSubString(1, length));
```

```
Hibernate:
    select
        book0_.isbn as isbn0_0_,
        book0_.title as title0_0_,
        book0_.author as author0_0_
    from
        Book book0_
    where
        book0_.isbn=?

Harry Potter and the Deathly Hallows

Hibernate:
    select
        book_.summary as summary0_,
        book_.cover as cover0_
    from
        Book book_
    where
        book_.isbn=?

Readers beware. The brilliant,
breathtaking conclusion to J.K.
Rowling's spellbinding series is not for
the faint of heart
```

Loads Summary when needed

Also loads cover

Instead use DTO Projection

- Already discussed during **SELECT new Object**
  - Can select (project) only the properties you need
  - Better than lazy-loading properties!

```
TypedQuery<Home> query = em.createQuery(
     "select new hibernate06.Home(p, a) "
     + "from Person p " + "join p.address a ", Home.class);
List<Home> homes = query.getResultList();

Person p = null;
Address a = null;
for (Home home : homes) {
     p = home.getPerson();
     a = home.getAddress();

     System.out.println(p.getFirstName()
     + " " + p.getLastName()
     + " has a home in " + a.getCity());
}
```

Not an Entity
but a DTO class

```
public class Home {
     private Person person;
     private Address address;

     public Home(Person p, Address a) {
          this.person = p;
          this.address = a;
     }
```

Lazy and Eager Summary

- Lazy and Eager **specify WHEN not HOW**
  - When is generally not the problem

- It’s good to know about these options
  - To understand how Hibernate works
  - While not the biggest source of problems or solutions