

# Utveckling av desktopapplikationer

## Föreläsning 9

---

*VT2017*

*NAZILA H.*

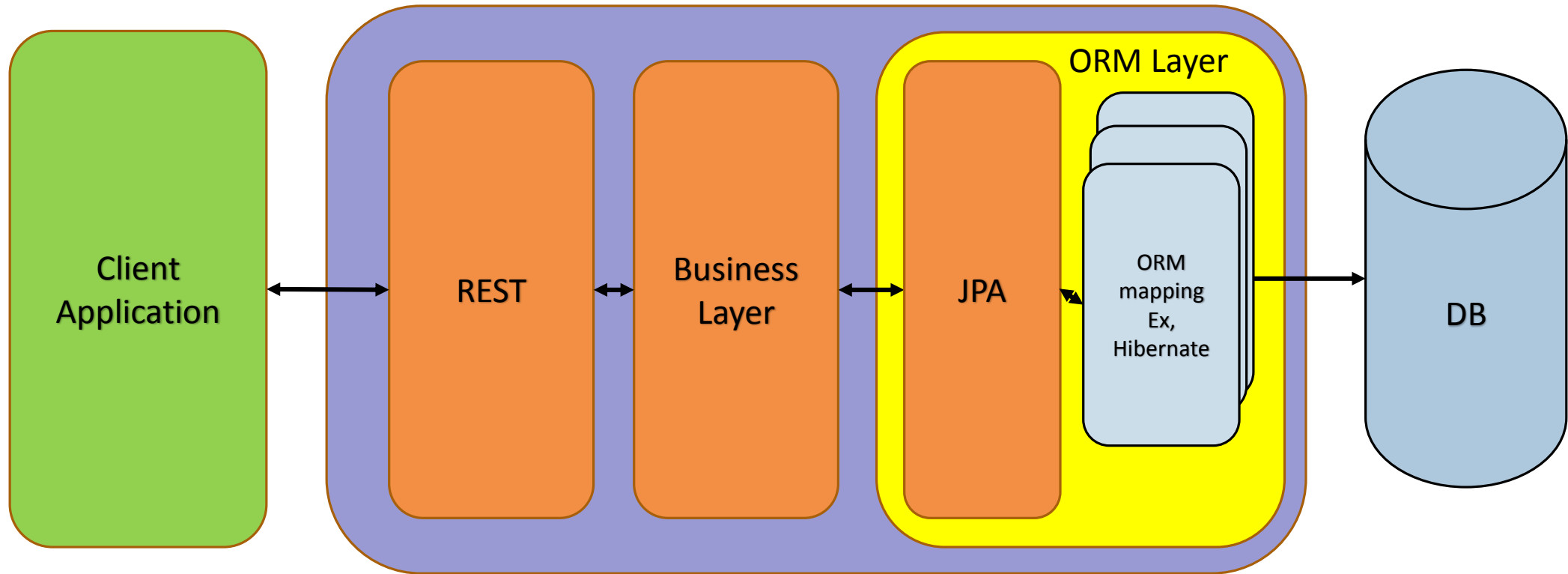
# Innehåll

---

- Hibernate fortsättning
- Relationer (1:n) (m:n)

# Java Persistence API

---



# One-To-One

---

Exempel:

Två Entity-klasser : Book och Author

**Senario 1:** En bok har en författare och en författare skriver en bok (One-to-One relationship).

@Entity

```
public class Book {  
    @Id@GeneratedValue  
    private int id;  
    private String name;
```

@OneToOne

```
    private Author author;
```

```
    .... //getter och setter
```

@Entity

```
public class Author {  
  
    @Id@GeneratedValue  
    private int id;  
    private String name;
```

```
    ...//getter och setter
```

Hibernate.cfg.xml

```
<property name="hibernate.show_sql">true</property>  
<property name="hbm2ddl.auto">create</property>  
<mapping class="model.Book"/>  
<mapping class="model.Author"/>  
</session-factory>  
</hibernate-configuration>
```

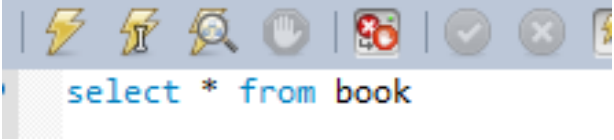
# One-To-One, Add a book

```
Book b1 = new Book();  
b1.setName("book 1");
```

```
Author author = new Author();  
author.setName("Sara");
```

```
b1.setAuthor(author);
```

```
Session session = sessionFactory.openSession();  
session.beginTransaction();  
session.save(b1);  
session.save(author);  
session.getTransaction().commit();
```



The screenshot shows a database query tool interface. At the top, there is a toolbar with icons for various database operations. Below the toolbar, the SQL query `select * from book` is entered. The results are displayed in a table with the following structure:

	id	name	author_id
▶	1	book 1	1
*	NULL	NULL	NULL

# One-To-Many, med extra tabell

---

## Senario 2: En bok har många författare

@Entity

```
public class Book {
```

@Id@GeneratedValue

```
private int id;
```

```
private String name;
```

@OneToMany

```
List<Author> authors;
```

```
...//getter och setter
```

@Entity

```
public class Author {
```

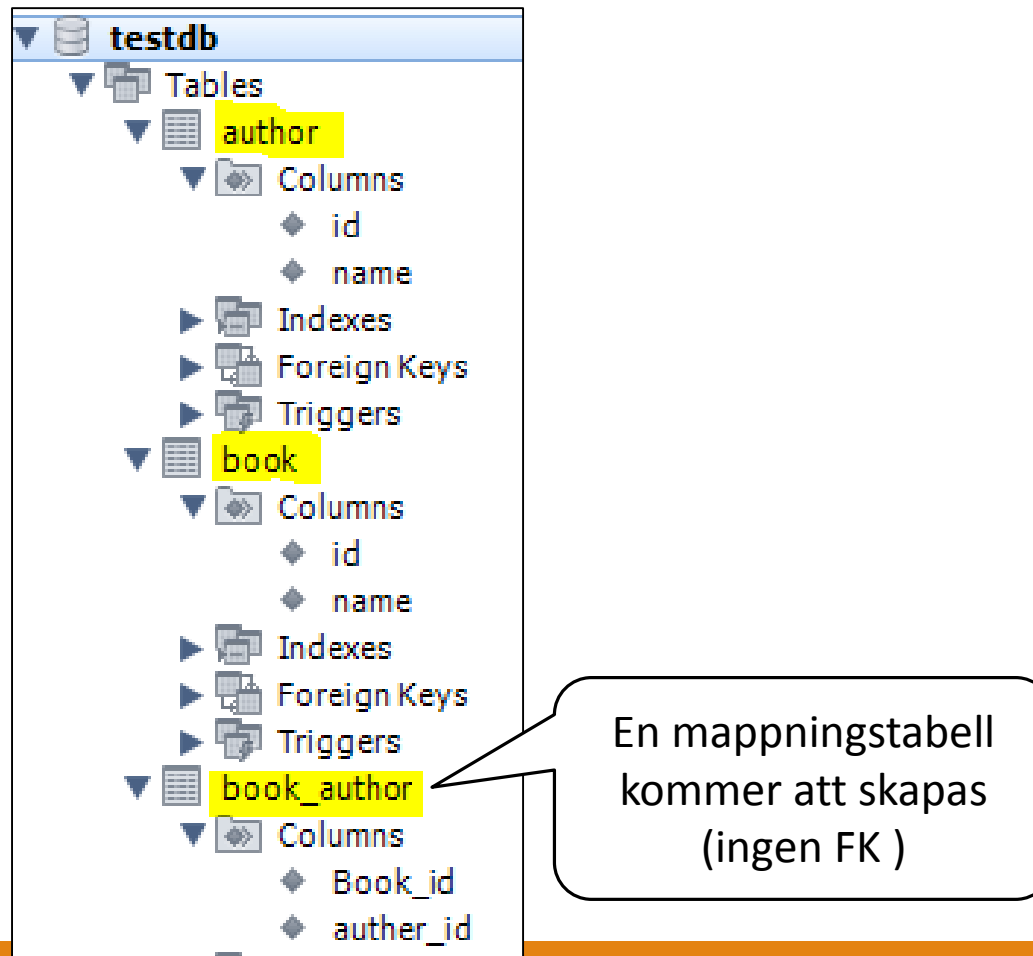
@Id@GeneratedValue

```
private int id;
```

```
private String name;
```

```
...//getter och setter
```

# One-To-Many, med extra tabell



# One-To-Many (med FK)

Alternativt kan man välja att ha FK istället för en extra tabell.

```
@Entity
public class Book {
    @Id@GeneratedValue
    private int id;
    private String name;

    @OneToMany(mappedBy = "book")
    @JsonBackReference
    private List<Author> author;

    .....// getters och setters
```

```
@Entity
public class Author {

    @Id@GeneratedValue
    private int id;
    private String name;

    @ManyToOne
    @JsonManagedReference
    @JsonIgnore
    Book book;

    ....//getter och setter
```

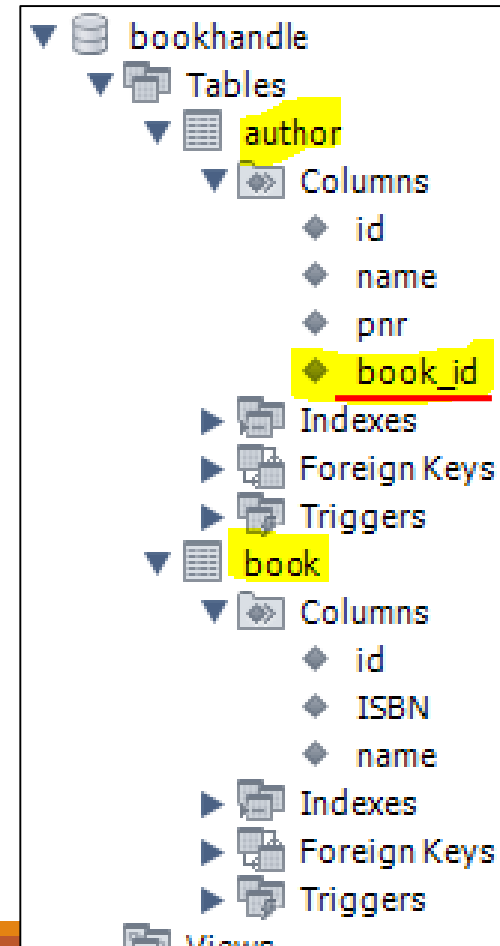


# One-To-Many (med FK), Add book

```
public void addBook(Book book)
{
    Session session = HibernateUtil.getSession();
    Transaction tx = session.beginTransaction();
    Book myBook = new Book();
    myBook.setName(book.getName());
    myBook.setISBN(book.getISBN());
    session.save(myBook);
    List<Author> authors = (List<Author>) book.getAuthors();
    myBook.setAuthors(authors);

    session.saveOrUpdate(myBook);

    tx.commit();
    session.close();
}
```



# Many-To-Many

@Entity

```
public class Book {
```

@Id@GeneratedValue

```
private int id;
```

```
private String name;
```

@ManyToMany(mappedBy = "books")

@JsonBackReference

```
private Collection<Author> author;
```

```
public Collection<Author> getAuther() {
```

```
    return author;
```

```
}
```

```
....//getter och setter
```

Titta i klassen  
Author(nästa slide) för att  
hitta books

Kan vara List istället för  
Collection

# Many-To-Many

---

```
@Entity
public class Author {

    @Id@GeneratedValue
    private int id;
    private String name;

    @ManyToMany
    @JsonManagedReference
    @JsonIgnore
    private Collection<Book> books;

    ....//getter och setter
```

# Innehåll

---

- Lazy Loading i Hibernate
- HQL

# Fråga

---

- Följande kod hämtar en book som har id=1 från databasen:

*Book book =(Book)session.get(Book.class,1);*

- Book innehåller en lista av klassen Author (en bok kan ha många författare)
- Frågan är att när vi hämtar en bok från databasen kommer alla relaterade författare att hämtas också?

# Problem

---

## ➤ Om ja:

- Detta kommer då påverka systemets prestanda.
- Ifall vi bara är intresserade av namn på boken, varför behöver vi då vänta i onödan tills alla relaterade författare ska hämtas? (om de är många så kommer det ta tid)

## ➤ Om nej:

- Ska vi skriva en separat query att hämta bokens författare efter att vi har hämtat boken? Det är inte heller optimalt.

# Lösning, Lazy Loading

---

- Hibernate har, som standard, en strategi som heter **Lazy Loading** vilket innebär att:
- Relaterade objekten kommer inte att hämtas vid en get-Request från databasen.
- **Men** så fort man försöker få tillgång till objektets lista ( ex, anropa `getAuthors()` på objektet) då kommer Hibernate automatiskt att hämta alla relaterade objekten (alla relaterade författare i det här fallet) med.

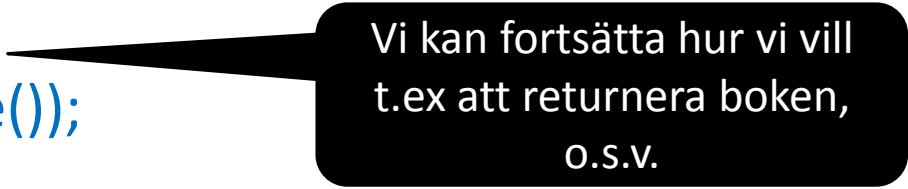
# Exempel

---

```
SessionFactory sessionFactory = NewHibernateUtil.getSessionFactory();  
Session session = sessionFactory.openSession();
```

```
    session.beginTransaction();
```

```
        Book book =(Book)session.get(Book.class,1);  
        Collection<Author> authors = book.getAuthors();  
        for (Author name : authors)  
            System.out.println(name.getName());
```



Vi kan fortsätta hur vi vill  
t.ex att returnera boken,  
o.s.v.

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



# Att hämta ett objekt från DB

---

```
public Book getBook(int id)
{
    Session session = HibernateUtil.getSession();
    session.beginTransaction();
    Book book = (Book)session.get(Book.class, id);
    book.getAuthors().size();
    session.getTransaction().commit();
    session.close();
    System.out.println(book.getName() + " " + book.getISBN() + " " + book.getId());
    System.out.println("The size of getauthors is: " + book.getAuthors().size());
    return book;
}
```

# HQL

---

- Hibernate Query Language
- Ett alternativ till Hibernate metoder för att kommunicera med databasen.
- Mer kontroll på data.
- Liknar SQL.
- Till skillnad från SQL som använder tabeller och kolumner, använder HQL namn på klasser och attribut.

# HQL

---

- Ingen select
- klassens namn istället för tabell
- klassens attribut istället för kolumn

**SQL:** **SELECT** \* **FROM** *tabell-namn* **WHERE** *kolumn-namn* = .....

**HQL:** **FROM** *Klass-namn* **WHERE** *attribut-namn* = .....

**SQL:** **SELECT** \* **FROM** *Author* **WHERE** *Author\_Age* > 65

**HQL:** **FROM** *Author* **WHERE** *age* > 65

# Exempel 1

---

➤ Query: package *org.Hibernate.Query*

```
session.beginTransaction();

    Query query = session.createQuery("from Author where age > 65");
    List<Author> oldAuthors = query.list();
    .....
    .....

session.getTransaction().commit();

session.close();
```

# Exempel 2

---

Alternativ 1 för  
att hämta alla  
böcker!

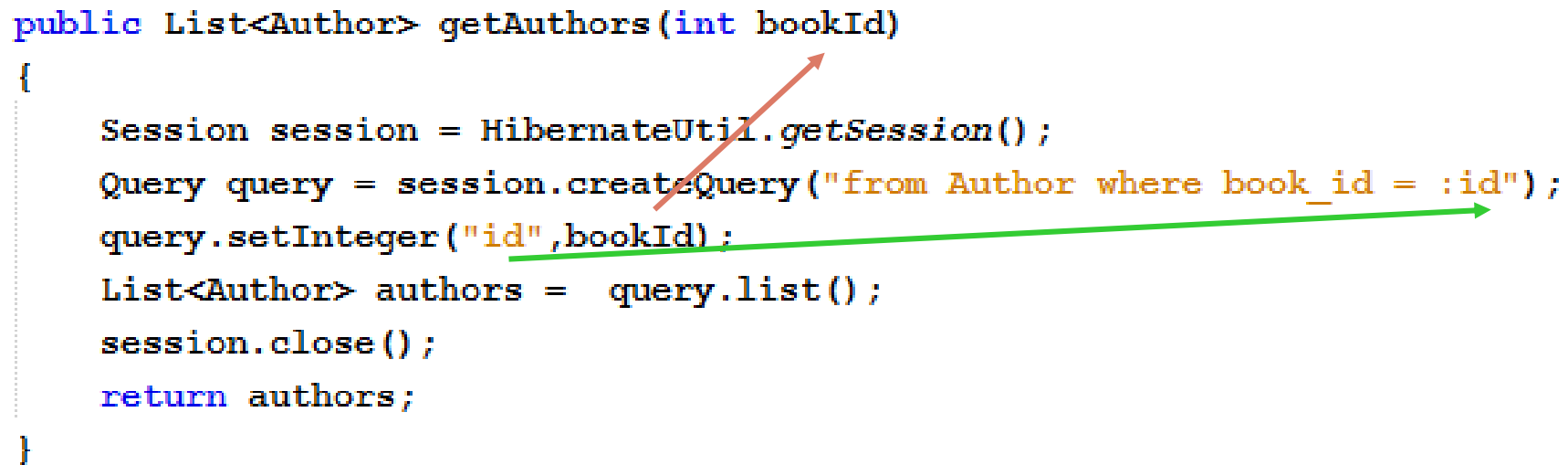
```
public List<Book> getBooks() {  
    Session session = HibernateUtil.getSession();  
    // List<Book> books = session.createCriteria(Book.class).list();  
    Query query = session.createQuery("from Book");  
    List<Book> books = query.list();  
    for(Book b:books)  
    {  
        b.getAuthors().size();  
    }  
    session.close();  
    return books;  
}
```

Alternativ 2 för  
att hämta alla  
böcker!

# Exempel 3

---

```
public List<Author> getAuthors(int bookId)
{
    Session session = HibernateUtil.getSession();
    Query query = session.createQuery("from Author where book_id = :id");
    query.setInteger("id",bookId);
    List<Author> authors = query.list();
    session.close();
    return authors;
}
```



# HQL, update

---

```
session.beginTransaction();
```

```
Query query = session.createQuery("update Book set name = :newName where id = :theid");
```

```
query.setParameter("newName", "Learning Java");
```

```
query.setParameter("theid", 1);
```

```
int result = query.executeUpdate();
```

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

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

	DTYPE	id	name	numberOfPages	size
▶	TextBook	1	<u>Learning Java</u>	670	NULL
	DigitalBook	2	DigitalBook	NULL	45
★	NULL	NULL	NULL	NULL	NULL