COMP S380F Lecture 8: Data Access Object (DAO), Hibernate, Spring Data JPA

Dr. Keith Lee

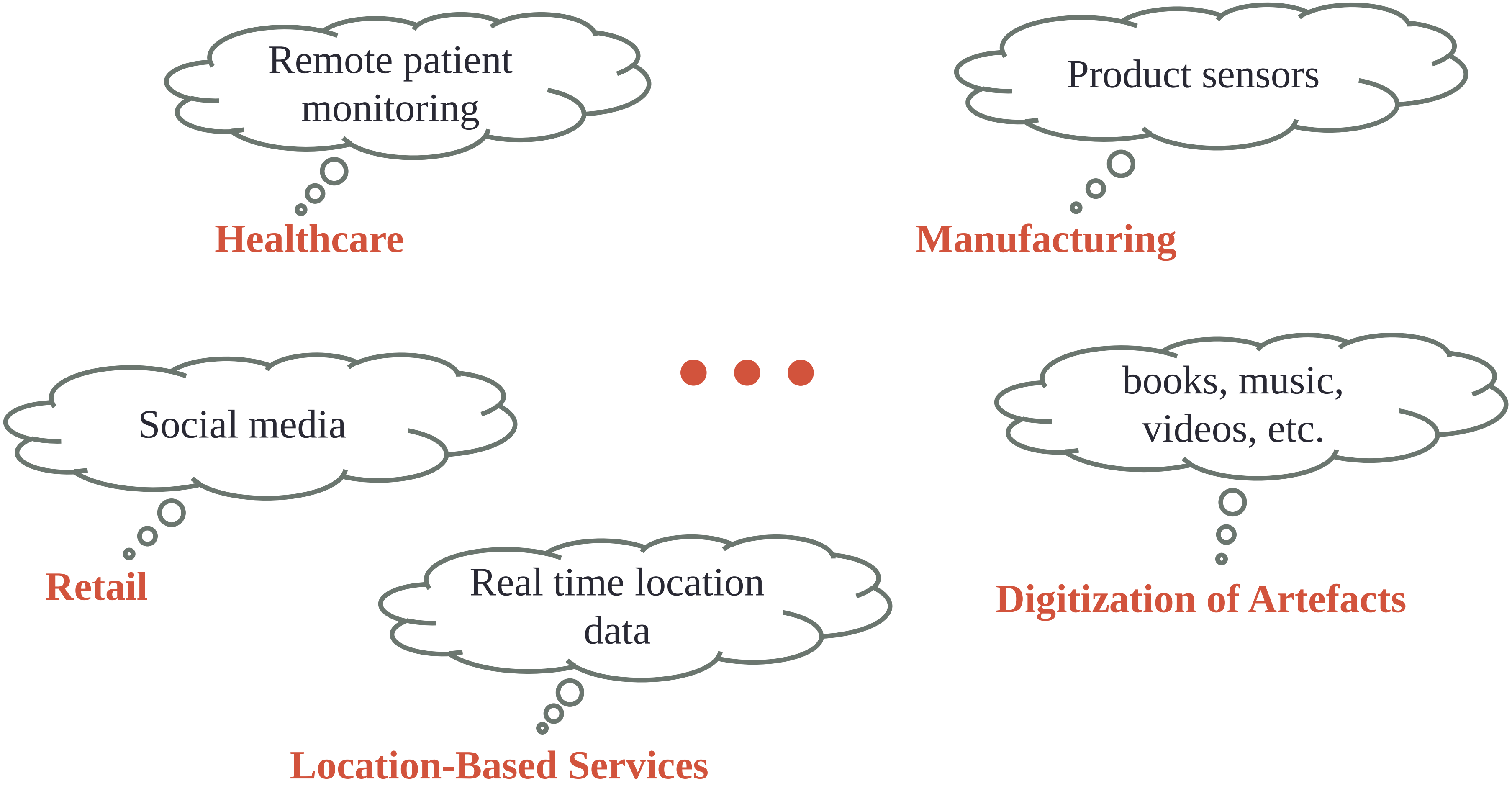
School of Science and Technology

Hong Kong Metropolitan University

Overview of this lecture

- Review on JDBC
- Object-Relational Mapping (ORM) tool: **Hibernate**
  - Entity classes
- **Data Access Object (DAO) pattern**
  - Repository interface & implementation
- Webapp example: **HelloSpringHibernate**
- JPA vs. Hibernate
- **Spring Data JPA** with Hibernate
  - Webapp example: **HelloSpringDataJPA**
- Automatic JPA repository with Spring Data JPA
- Defining repository method
  - By name convention
  - By @Query annotation
  - By implementation

We are generating vast amount of data!!



Remote patient monitoring

Healthcare

Product sensors

Manufacturing

Social media

Retail

• • •

books, music, videos, etc.

Digitization of Artefacts

Real time location data

Location-Based Services

We are generating vast amount of data!! (cont’)

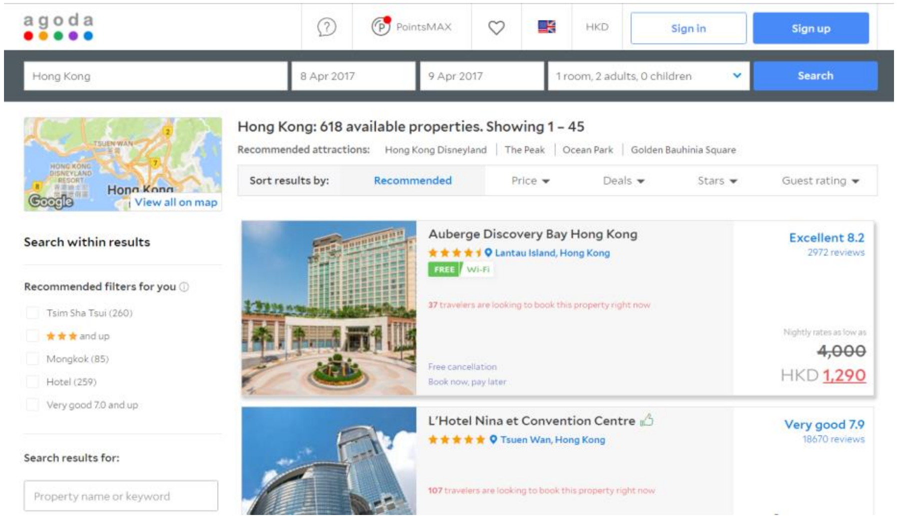
- Air Bus A380:
  - ➢ Generate 10 TB of data every 30 min
- Twitter (X):
  - ➢ Generate ~12 TB of data per day
- Facebook:
  - ➢ Facebook data grows by over 500 TB daily
- New York Stock:
  - ➢ Exchange 1 TB of data everyday

Challenge

- How do we store and access this data over the web?

E-commerce website

- Data operations are mainly transactions (Reads and Writes)
- Operations are mostly online
- Response time should be quick but it is important to maintain **security and reliability of transactions**
- ACID properties are important



Hong Kong | 8 Apr 2017 | 9 Apr 2017 | 1 room, 2 adults, 0 children

Hong Kong: 618 available properties. Showing 1 – 45

Recommended attractions: Hong Kong Disneyland | The Peak | Ocean Park | Golden Bauhinia Square

Sort results by: Recommended | Price | Deals | Stars | Guest rating

Search within results

Recommended filters for you

□ Tsim Sha Tsui (260)
□ ★★★ and up
□ Mongkok (85)
□ Hotel (259)
□ Very good 7.0 and up

Search results for:

Property name or keyword

Auberge Discovery Bay Hong Kong
Lantau Island, Hong Kong
FREE Wi-Fi
37 travelers are looking to book this property right now
Free cancellation
Book now, pay later
Excellent 8.2
2972 reviews
Nightly rates as low as
~~4,000~~
HKD 1,290

L'Hotel Nina et Convention Centre
Tsuen Wan, Hong Kong
107 travelers are looking to book this property right now
Very good 7.9
18670 reviews

Challenge

- How do we store and access this data over the web?

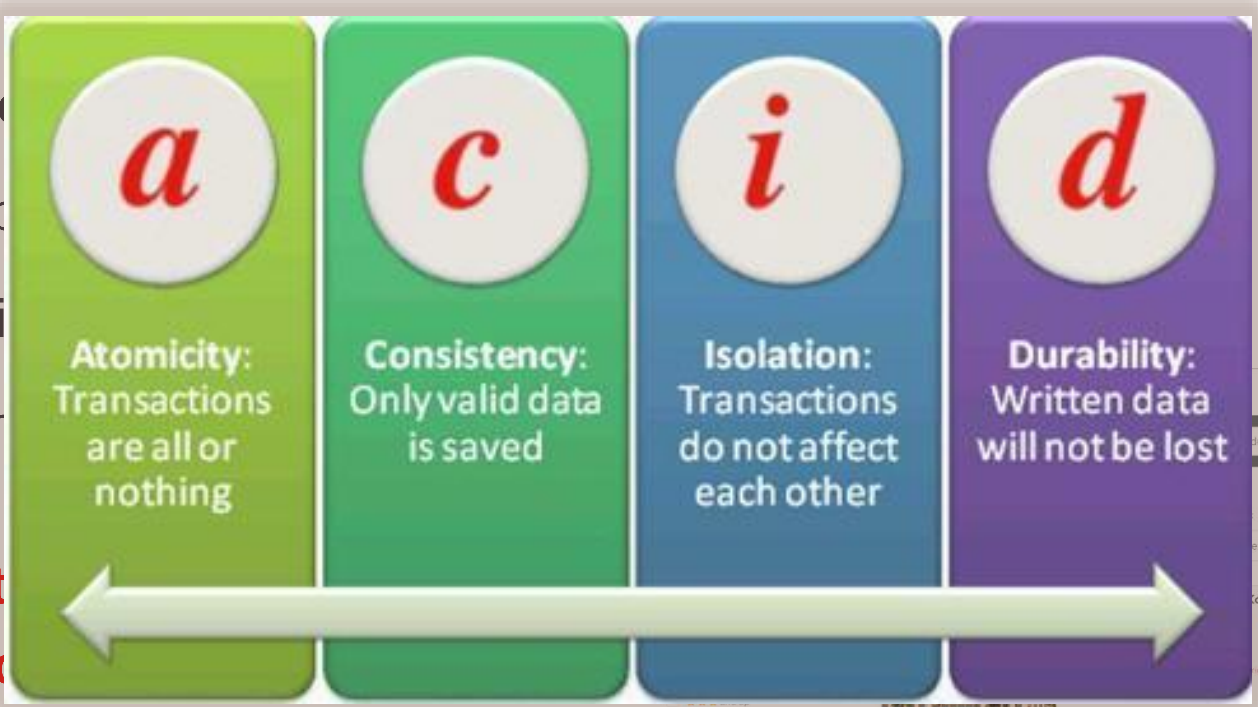
E-commer

- Data op
- Operati
- Respor

  but it is

  securit

  transac

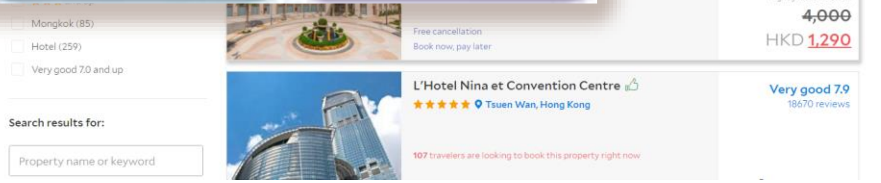


Atomicity: Transactions are all or nothing

Consistency: Only valid data is saved

Isolation: Transactions do not affect each other

Durability: Written data will not be lost



Golden Bauhinia Square

eals

ong

Excellent 8.2
2972 reviews

ight now

Nightly rates as low as
~~4,000~~
HKD 1,290

Mongkok (85)
Hotel (259)
Very good 7.0 and up

Search results for:

Property name or keyword

Free cancellation
Book now, pay later

L'Hotel Nina et Convention Centre
★★★★★ Tsuen Wan, Hong Kong

107 travelers are looking to book this property right now

Very good 7.9
18670 reviews

- ACID properties are important

Challenge (cont’)

- How do we store and access this data over the web?

Image serving website

- Data operations are mainly fetching large files (Reads)
- ACID requirements can be **relaxed**
  - E.g., It is hard to maintain *consistency* due to lack of direct control of data from users, so we may relax the requirement to only ensure consistent outcomes from the data.
  - E.g., Locking shared data may result in denial of service, so the level of *isolation* may not be as high as traditional system.
- Operations are mainly online
- High bandwidth requirement

Challenge (cont’)

- How do we store and access this data over the web?

Search engine

- Data operations are mainly reading index files for answering queries (Reads)
- ACID requirements can be relaxed
- Index compilation is performed offline due to the large size of source data (the entire Web)
- Response time must be as fast as possible

Data persistence

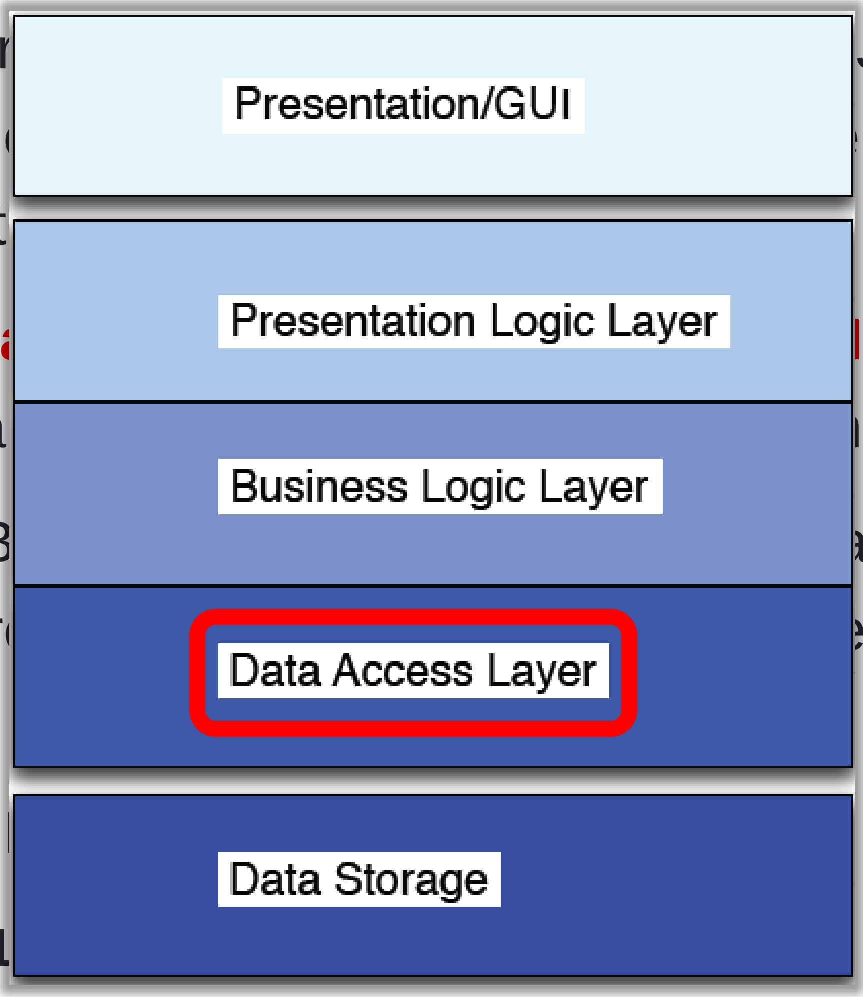
- When we talk about persistence in Java, we normally mean storing data in a relational database using SQL.
- Relational database technology is a common denominator for many disparate systems and technology platforms.
- Relational database provides a way of **sharing data** across different applications or technologies that form part of the same application.
- The relational data model is often the common enterprise-wide presentation of business entities.

Data persistence (cont’)

- When you work with a relational database in a Java application, the Java code issues SQL statements to the database via the JDBC API.
- The **Java Database Connectivity (JDBC) API** provides universal data access from the Java programming language.
- Using the JDBC API, you can access virtually any data source, from relational databases to spreadsheets and flat files.
- The JDBC API is comprised of two packages:
  - ➢ **`java.sql`**
  - ➢ **`javax.sql`**

Data persistence (cont’)

- When you wor Java application, th nts to the database via t
- The **Java Data** provides universal data ing language.
- Using the JDB any data source, from r eets and flat files.
- The JDBC AP
  - ➢ **`java.sql`**
  - ➢ **`javax.sql`**



Presentation/GUI

Presentation Logic Layer

Business Logic Layer

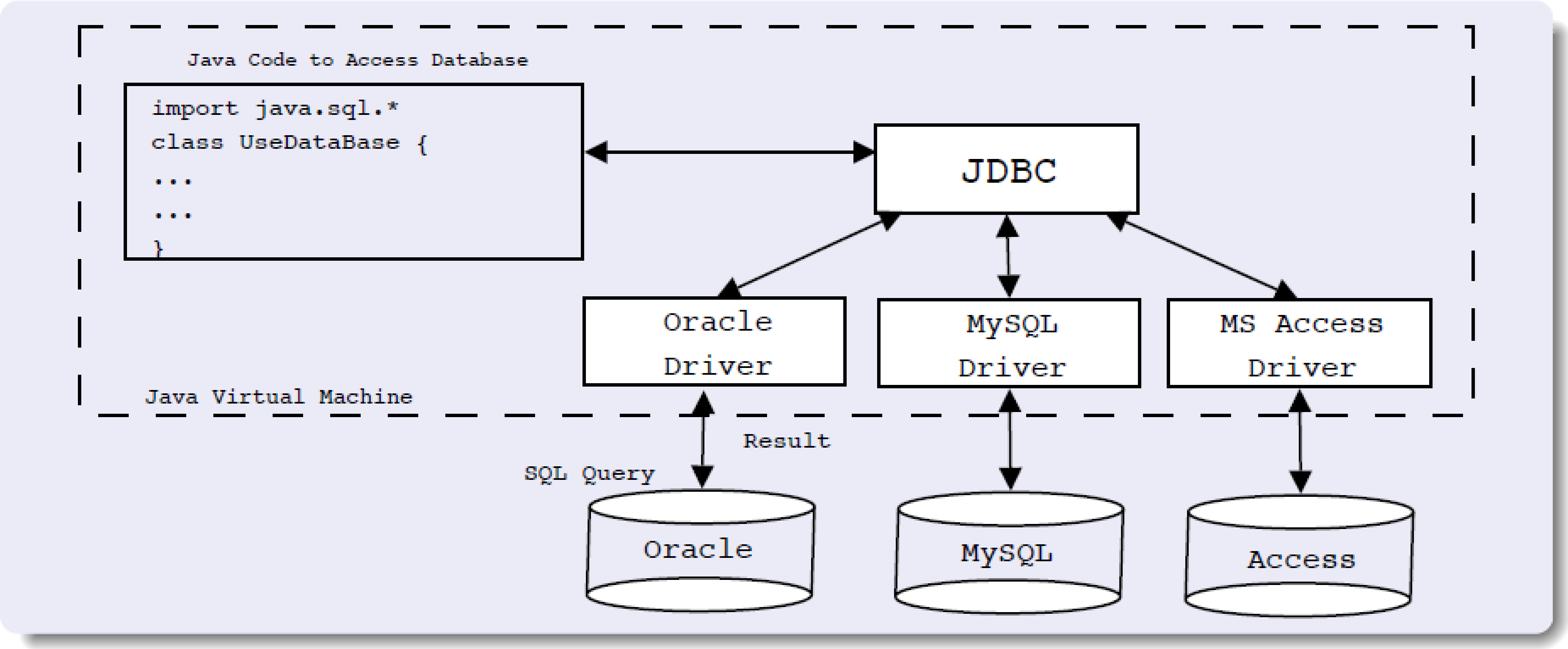
Data Access Layer

Data Storage

JDBC: Accessing DB from an application

- JDBC (Java Database Connectivity) provides access to relational database management system (RDBMS) via SQL.
  - E.g., MySQL, Oracle, PostgreSQL, H2, Apache Derby
- JDBC offers simplicity (easy to use) and database independence.
  - Developers do not need to worry about differences in vendor-specific database connection instructions.
- A vendor would have to produce JDBC drivers and build their proprietary connection management code beneath a common Java API.
- A **JDBC driver** is a segment of code designed to enable access to a *particular kind* of database.

JDBC: Accessing DB from an application (cont')



- Java classes in the JDBC package (`java.sql.*`) are interfaces.
- RDBMS vendors implements the interfaces to fit with their own products.

JDBC concepts

- When developers use JDBC, they construct SQL statements that can be executed. E.g.,

  SELECT name FROM employee WHERE age = ?

- The above SQL is combined with local data structures so that regular Java objects can be mapped to the bindings in the string.

  ➢ E.g., an Integer object with the value of 42 can be mapped:

  SELECT name FROM employee WHERE age = 42

- The results of execution, if any, are combined in a set returned to the caller. E.g., the above SQL may return:

```
name
----------
Peter
Mary
John
```

→ We can browse this result set as necessary.

JDBC interfaces

- **`java.sql.Statement`**
  - Represent a SQL statement (SELECT or UPDATE) to be sent to DBMS
  - Related methods:
    - Execution: `execute, executeQuery, executeUpdate`
    - Creation: `Connection.createStatement`
- **`java.sql.ResultSet`**
  - Hold the result of executing an SQL query (i.e., the result relation)
  - Handle access both to rows and columns within rows
  - Related methods:
    - Iteration: `next`
    - Accessing data: `get{Type}(position|name)`
      - E.g., `getInt(4), getString("name")`

PreparedStatement object

- A more realistic case is that the same kind of SQL statement is processed over and over (rather than a static SQL statement).

```
SELECT * FROM employee WHERE id = 3;
SELECT * FROM employee WHERE id = 7;
SELECT * FROM employee WHERE id = 25;
SELECT * FROM employee WHERE id = 21;
...
SELECT * FROM employee WHERE id = ?
```

- In **PreparedStatement**, a placeholder (?) will be bound to an incoming value before execution (no recompilation).

```
PreparedStatement ps =
       conn.prepareStatement("SELECT * FROM exmployee WHERE id=?");
ResultSet rs;
for (int i = 0; i < 1000; i++) {
   ps.setInt(1, i);
   rs = ps.executeQuery();
   /* Do something more */
}
```

Object-Relational Mapping (ORM)

- JDBC is a primitive way for applications to talk to databases.
- **Object-relational mapping (ORM)** aims to give you automated (and transparent) persistence of objects in Java application to the tables in a relational database, using metadata that describes the mapping between the objects and the database.
- There are many tools / frameworks in this area:
  - JDO (Java Data Objects): http://db.apache.org/jdo/
  - EclipseLink: https://eclipse.dev/eclipselink/
  - **Hibernate**: http://hibernate.org/
  - **JPA (Jakarta Persistence API, formerly Java Persistence API)**: https://projects.eclipse.org/projects/ee4j.jpa
- In this course, we will use Hibernate, which is one of the popular ORM tools (e.g., WildFly has built-in Hibernate support).

Features of ORM tools

- Object to Relational mapping using ordinary POJO (**Entity class**)
- **Lazy loading**:
  - ➢ As object graphs become more complex (e.g., one-to-many, many-to-many mappings), you do not want to fetch entire relationships immediately.
  - ➢ Lazy loading allows you to grab data only as it is needed.
    - E.g., A ***Teacher*** has many ***Courses*** to teach (one to many).
    - The ***Teacher*** object is fetched, and the related ***Course*** objects are fetched only when needed.
- **Eager fetching**:
  - ➢ Opposite of lazy loading.
  - ➢ Eager fetching allows you to grab an entire object graph in one query, and thus saving you from costly round-trips in some cases.
- **Cascading**: Changes to one table result in changes to other tables.

Entity class

Web app example: lecture08-hellospringhibernate

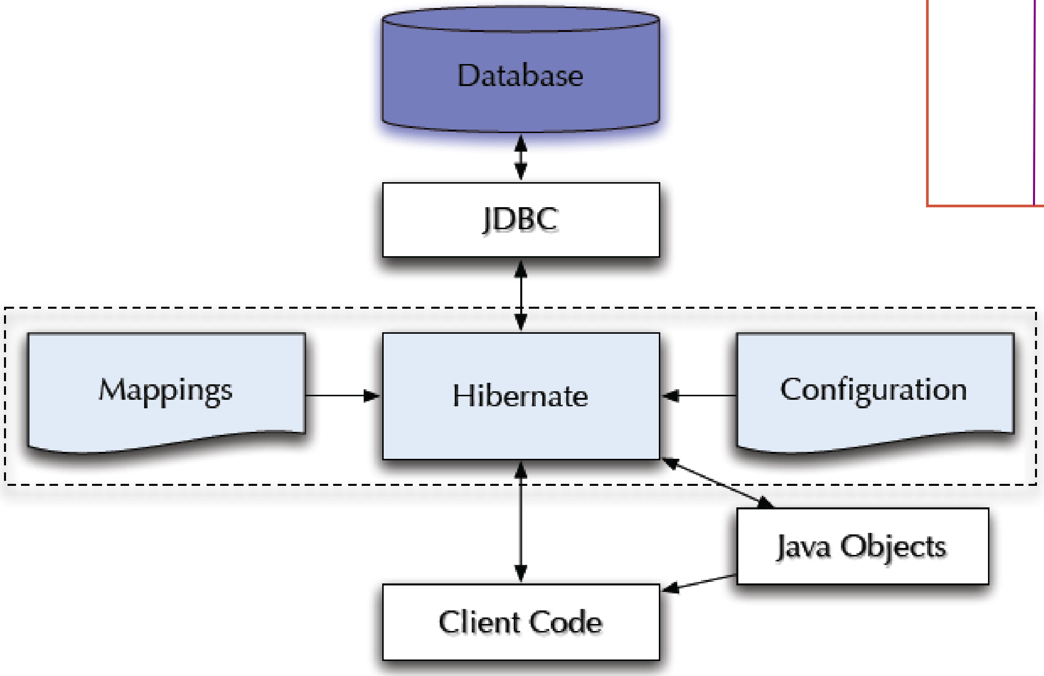
- An **entity class** maps to a database table for manipulating its records.
- It is often a POJO and is marked with **field annotations**.

Relational DB

GUESTBOOK		
PK	id	INTEGER
	name	VARCHAR(50)
	message	VARCHAR(200)
	date	TIMESTAMP

Entity class

```
GuestBookEntry
id: Integer
name: String
message: String
date: Date
```



Entity class: Field annotations

GuestBookEntry.java

```
@Entity
@Table(name = "guestbook")
public class GuestBookEntry {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String message;

    @Temporal(TemporalType.TIMESTAMP)
    private Date date;

    …
}
```

- **@Entity** specifies an entity class.
- By default, the table name equals the class name.
  - ➢ Use **@Table** on the class if the table and class have different names.

Entity class: Field annotations (cont')

GuestBookEntry.java

```
@Entity
@Table(name = "guestbook")
public class GuestBookEntry {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String message;

    @Temporal(TemporalType.TIMESTAMP)
    private Date date;

    …
}
```

For converting between java.sql.Timestamp & java.util.Date

Can have value DATE, TIME, or TIMESTAMP.

- By default, the table column name equals the property name.
  - ➢ Use **@Column** if they are different:

```
@Column(name = "employees_number")
private Integer emplNumber;
```

- **@Id** specifies the primary key of the table.
- **@GeneratedValue** specifies that property value will be automatically generated. **strategy** specifies how the primary key is generated.

Entity class (cont')

- Similar to JavaBean, an Entity class requires an no-arg constructor.
- To ease debugging, we may override the `toString()` method such that we can print the data in an entity class using `System.out.println()`.

GuestBookEntry.java

```
@Entity
@Table(name = "guestbook")
public class GuestBookEntry {
    ...

  // getters and setters for id, name, message, date

  @Override
  public String toString() {
    return "GuestBookEntry{" +
            "id=" + id +
            ", name='" + name + '\'' +
            ", message='" + message + '\'' +
            ", date=" + date +
            '}';
  }
}
```

Configuration for using Hibernate with Spring Boot

- Starter dependency for Spring Data JPA

```
implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
```

- Dependency for a particular database vendor, e.g.,
  - H2:

```
runtimeOnly 'com.h2database:h2'
```

  - MySQL:

```
runtimeOnly 'com.mysql:mysql-connector-j'
```

  - Apache Derby:

```
runtimeOnly 'org.apache.derby:derby'
```

Configuration for using Hibernate with Spring Boot (con’t)

Properties in **application.properties**

AUTO_SERVER=TRUE enables H2’s *Automatic Mixed Mode*, which allows multiple processes to access the database.

Data source: e.g.,

- H2 data source (. is the root project directory):

```
spring.datasource.url=jdbc:h2:./Data/myDB;AUTO_SERVER=TRUE
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
```

H2 Tutorial: http://h2database.com/html/tutorial.html

- Apache Derby data source:

```
spring.datasource.url=jdbc:derby://localhost:1527/myDB;create=true
spring.datasource.driver-class-name=org.apache.derby.iapi.jdbc.AutoloadedDriver
spring.datasource.username=sa
spring.datasource.password=password
```

Apache Derby 10.17 Manual: https://db.apache.org/derby/docs/10.17/ref/index.html

Properties in **application.properties**

- **Auto-initialize DB:** Hibernate has a feature to generate database tables according to the entity classes (which is useful for demo and testing).

```
spring.jpa.hibernate.ddl-auto=update
```

- `create`: Hibernate first drops existing tables, then creates new tables.
- `update`: The object model created based on the mappings (annotations or XML) is compared with existing schema, and then Hibernate updates the schema accordingly (but never deletes existing tables/columns).
- `create-drop`: Similar to `create`, with the addition that Hibernate will drop the database after all operations are completed (useful for unit testing).
- `validate` – Hibernate only validates whether the tables and columns exist; if not, it throws an exception.
- `none` – This value effectively turns off the DDL generation.

More details: https://docs.spring.io/spring-boot/docs/current/reference/html/howto.html#howto.data-initialization.using-hibernate

Properties in **application.properties**

- **Initialize DB using SQL files:** Spring Boot can also create database tables and insert data into them using SQL files on the classpath.

```
spring.jpa.hibernate.ddl-auto=none
spring.sql.init.mode=always
spring.sql.init.schema-locations=classpath:sql/schema.sql
spring.sql.init.data-locations=classpath:sql/data.sql
```

- By default, SQL database initialization is only performed for an embedded ***in-memory*** database, so we need `spring.sql.init.mode=always` .

/resources/**sql/schema.sql**

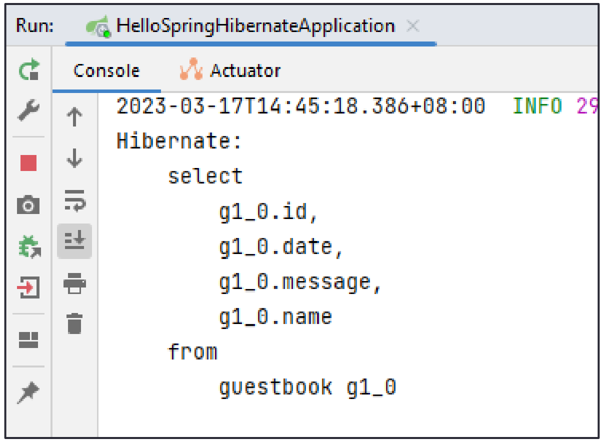
```
CREATE TABLE IF NOT EXISTS guestbook (
    id BIGINT NOT NULL GENERATED ALWAYS AS IDENTITY,
    name VARCHAR(50),
    message VARCHAR(255),
    date TIMESTAMP,
    PRIMARY KEY (id)
);
```

/resources/**sql/data.sql**

```
INSERT INTO guestbook (date, message, name)
    VALUES ('2024-03-15 08:39:01.629000', 'Hello', 'Keith');
INSERT INTO guestbook (date, message, name)
    VALUES ('2024-03-15 08:39:33.152000', 'Hi', 'John');
```

Details: https://docs.spring.io/spring-boot/docs/current/reference/html/howto.html#howto.data-initialization.using-basic-sql-scripts

Properties in **application.properties**



```
2023-03-17T14:45:18.386+08:00  INFO 29
Hibernate:
    select
        g1_0.id,
        g1_0.date,
        g1_0.message,
        g1_0.name
    from
        guestbook g1_0
```

- **Show hibernate-generated SQL in console**

```
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

- **Hibernate's Session vs. JPA's Entity Manager**

➢ Hibernate's **Session** object (`org.hibernate.Session`) provides basic data-access functionality such as the ability to save, update, delete, and load entity objects from the database (similar to a database connection).

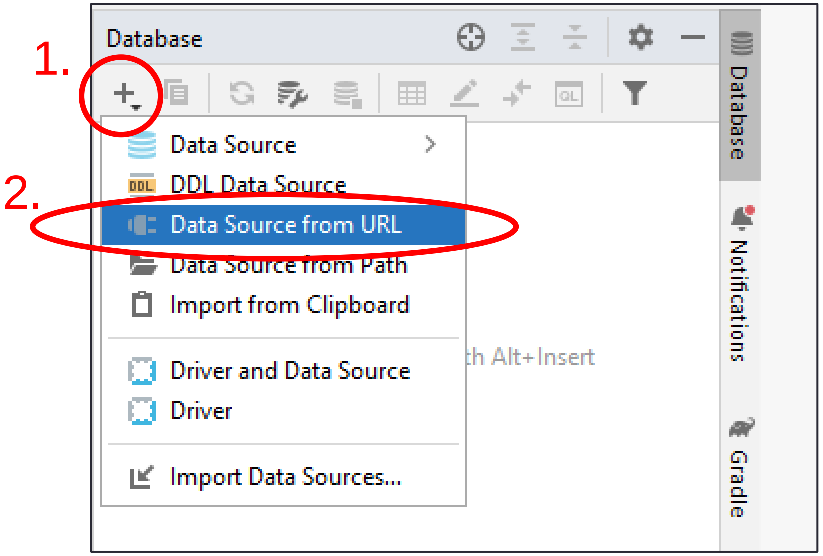
➢ JPA uses **EntityManager** object (`jakarta.persistence.EntityManager`) for similar purpose, which involves Hibernate's Session under the hood.

➢ To obtain Hibernate's Session from EntityManager, we need the property:

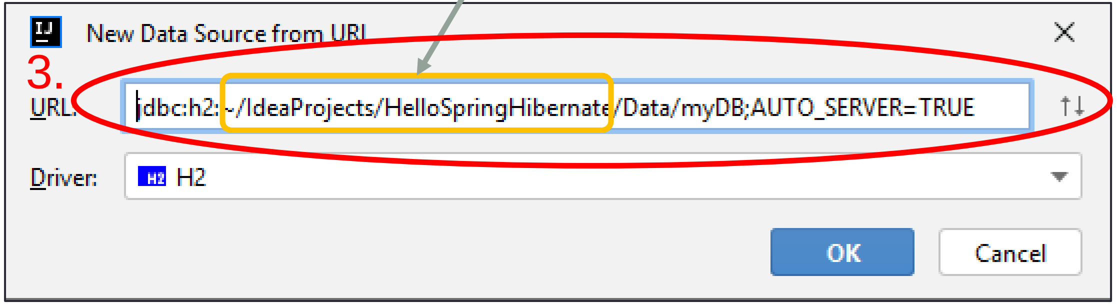
```
spring.jpa.properties.hibernate.current_session_context_class
   = org.springframework.orm.hibernate5.SpringSessionContext
```

Database view in IntelliJ

- It is convenient to interact with the database in IntelliJ IDEA Ultimate.
- We can create a data source connection (after running the web app once), as follows:



Our web app project folder

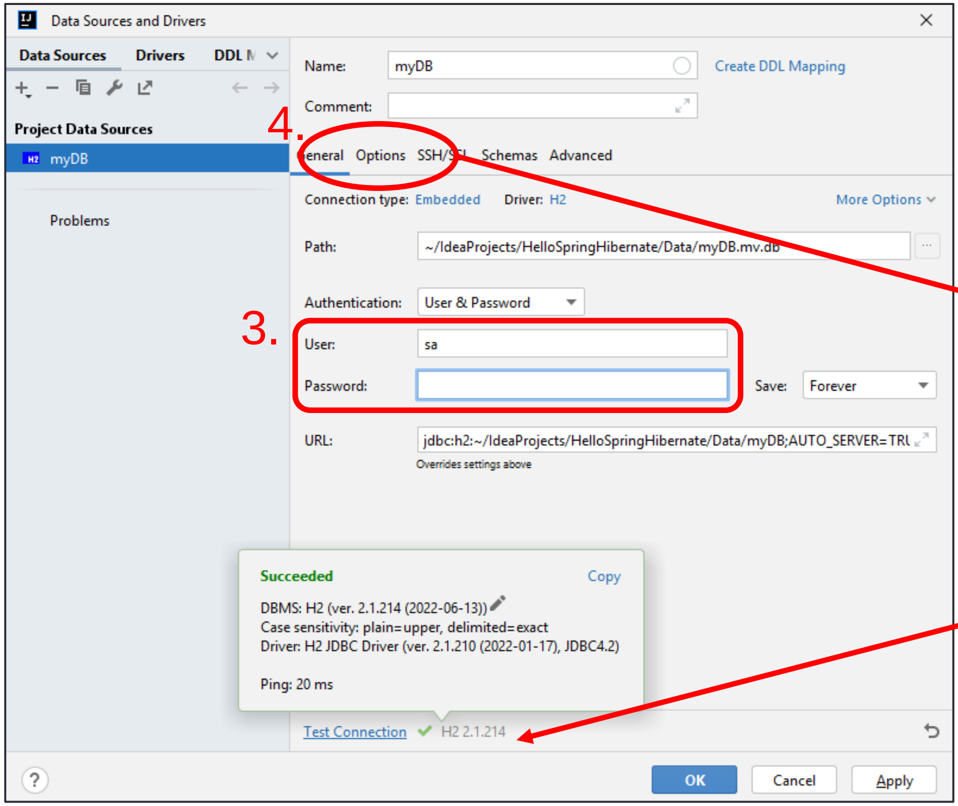


URL: jdbc:h2:~/IdeaProjects/HelloSpringHibernate/Data/myDB;AUTO_SERVER=TRUE

Driver: H2

Database view in IntelliJ (cont’)

- To configure the User and Password fields of the DB, enter the user and password we set in our `application.properties` file (`sa, password`).
- If there is a warning about missing H2 drivers, click on `Download missing driver files`.



Data Sources and Drivers

Data Sources | Drivers | DDL M

Project Data Sources

myDB

Problems

Name: myDB — Create DDL Mapping

Comment:

General | Options | SSH/SSL | Schemas | Advanced

Connection type: Embedded Driver: H2 — More Options

Path: ~/IdeaProjects/HelloSpringHibernate/Data/myDB.mv.db

Authentication: User & Password

User: sa

Password:

Save: Forever

URL: jdbc:h2:~/IdeaProjects/HelloSpringHibernate/Data/myDB;AUTO_SERVER=TRU

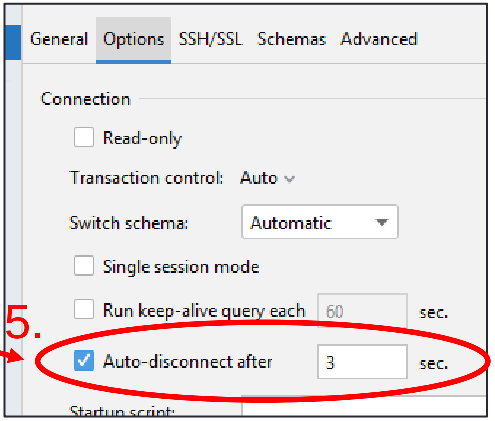
Overrides settings above

Succeeded — Copy

DBMS: H2 (ver. 2.1.214 (2022-06-13))
Case sensitivity: plain=upper, delimited=exact
Driver: H2 JDBC Driver (ver. 2.1.210 (2022-01-17), JDBC4.2)

Ping: 20 ms

Test Connection ✓ H2 2.1.214



General | Options | SSH/SSL | Schemas | Advanced

Connection

☐ Read-only

Transaction control: Auto

Switch schema: Automatic

☐ Single session mode

☐ Run keep-alive query each 60 sec.

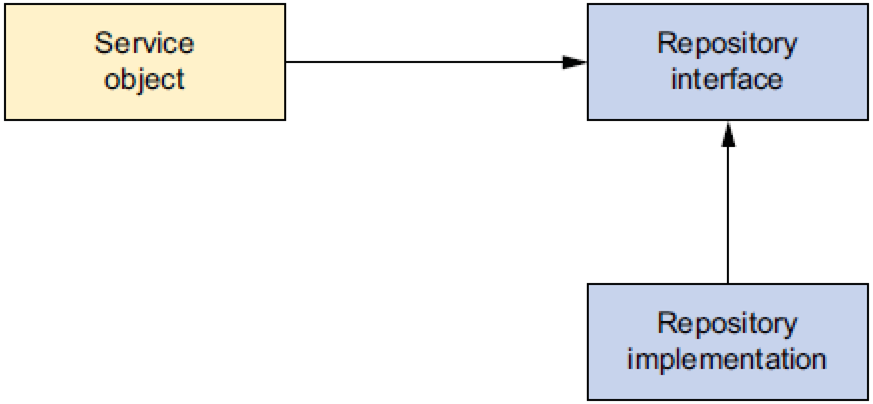
☑ Auto-disconnect after 3 sec.

Startup script:

- We can “`Test Connection`” before confirming the config.

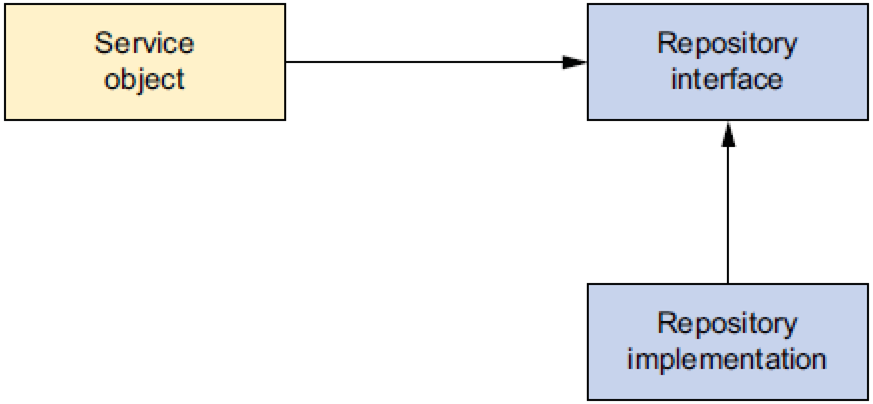
Data Access Object (DAO) pattern

- Data access strategies often differ depending on the storage mechanism.
  - ➢ Accessing relational tables is different from accessing XML files
  - ➢ JDBC and Hibernate are different ways to access relational tables
- To avoid scattering persistence logic across the application, we factor database access into one or more components.
- Such component is called **Data Access Object (DAO)** or a **repository**.
- DAO is a design pattern that provides abstract interface to the retrieval of data from a data resource.
- It aims to provide a uniform access interface for persistent storages.



DAO: Advantages

- Changes to persistence access layer do not affect DAO clients as long as the interface remains correctly implemented.
- DAO is used to insulate an application from the numerous, complex, and varied Java persistence technologies (e.g., JDBC, EJB CMP, TopLink, Hibernate, iBATIS).
- Using Data Access Objects means the underlying technology can be upgraded or swapped without changing other parts of the application.

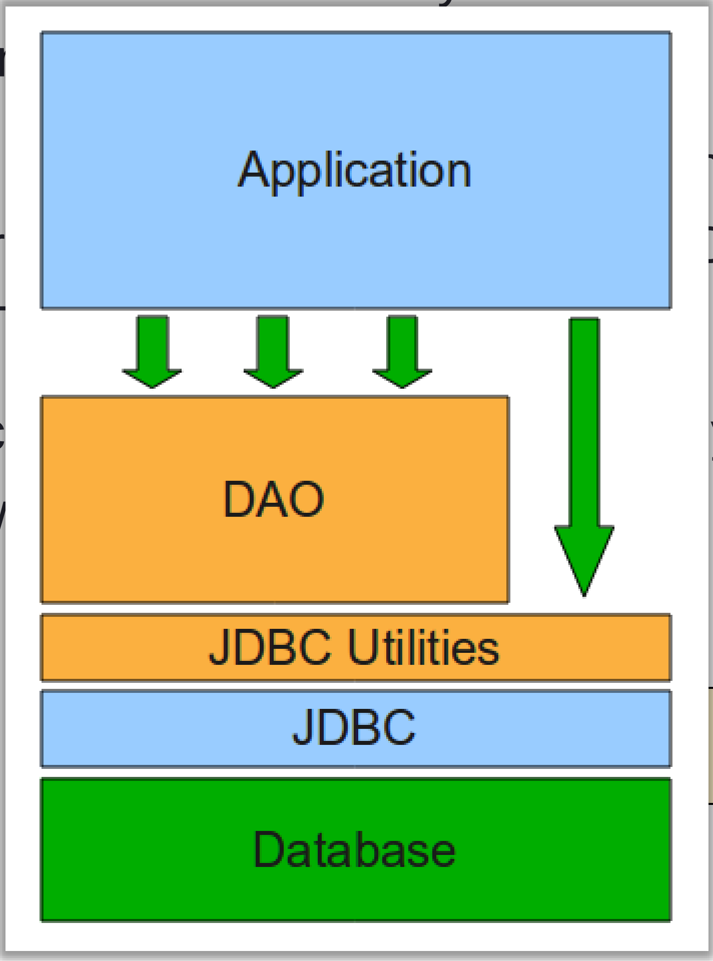


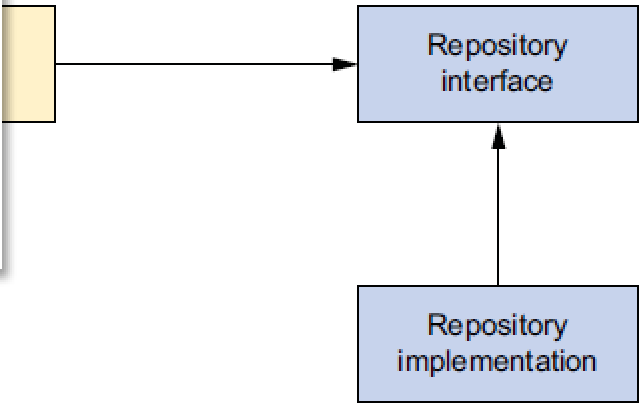
DAO: Advantages

- Changes to persistence access layer do not affect DAO clients as long as the interface re

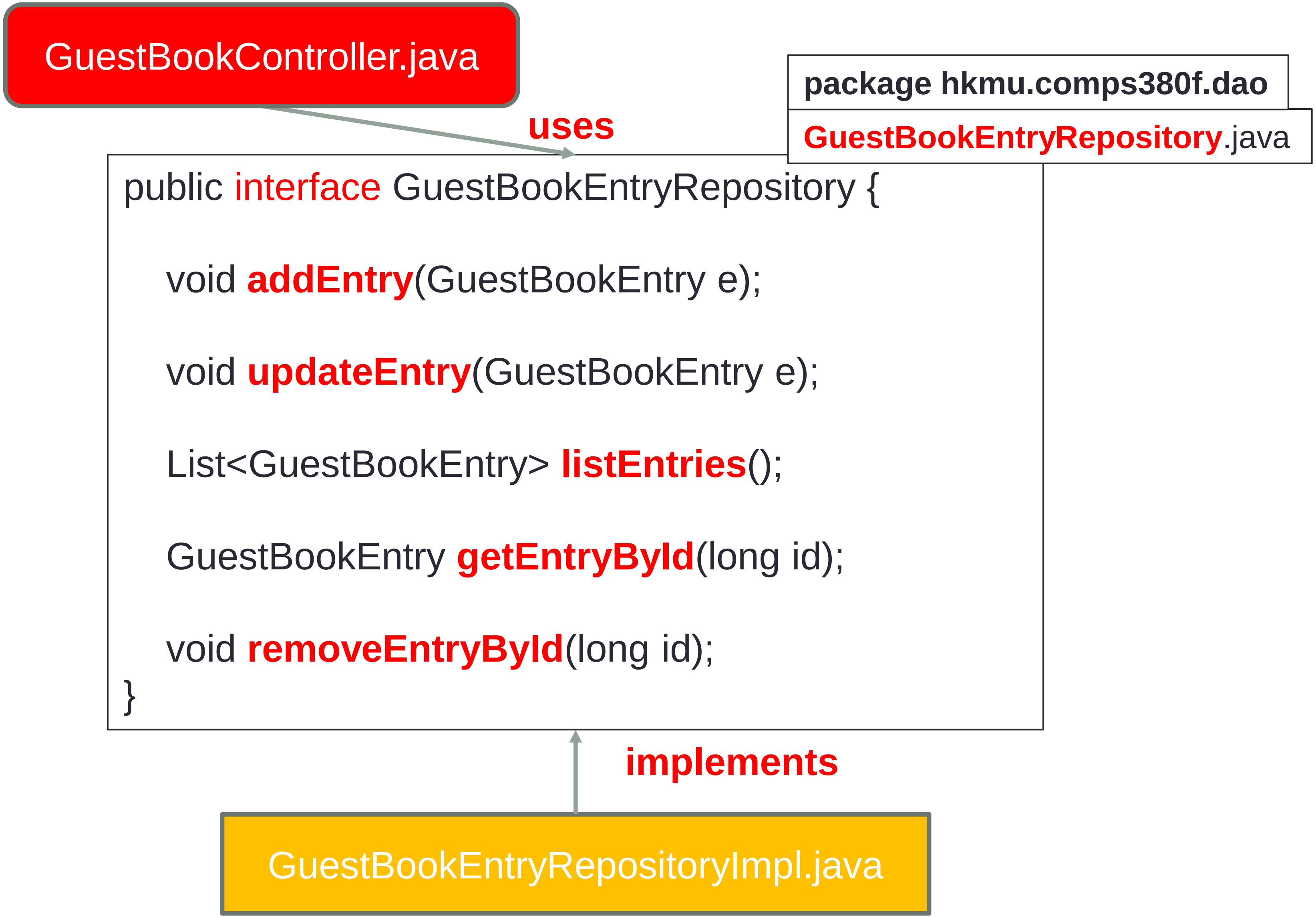
- DAO is used to ne numerous, complex, and varied Java per DBC, EJB CMP, TopLink, Hibernate, iBAT

- Using Data Acc ying technology can be upgraded or sw parts of the application.





HelloSpringHibernate: Repository interface



GuestBookController.java

uses

package hkmu.comps380f.dao

GuestBookEntryRepository.java

```
public interface GuestBookEntryRepository {

    void addEntry(GuestBookEntry e);

    void updateEntry(GuestBookEntry e);

    List<GuestBookEntry> listEntries();

    GuestBookEntry getEntryById(long id);

    void removeEntryById(long id);
}
```

implements

GuestBookEntryRepositoryImpl.java

HelloSpringHibernate: Repository implementation

GuestBookEntryRepositoryImpl.java

```
package hkmu.comps380f.dao;

import hkmu.comps380f.model.GuestBookEntry;
import jakarta.persistence.EntityManager;
import org.hibernate.Hibernate;
import org.hibernate.Session;
import org.hibernate.query.Query;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;
import java.util.List;

@Repository
public class GuestBookEntryRepositoryImpl implements GuestBookEntryRepository {
        …
}
```

- **`@Repository`** tells Spring to create a Spring bean for this class.
- We will use the annotation **`@Transactional`** for Spring’s database transaction management.

HelloSpringHibernate: Repository implementation (cont')

GuestBookEntryRepositoryImpl.java

```
@Repository
public class GuestBookEntryRepositoryImpl implements GuestBookEntryRepository {

    private final Session session;

    @Autowired
    public GuestBookEntryRepositoryImpl(EntityManager entityManager) {
        this.session = entityManager.unwrap(Session.class);
    }

    // … Implementation of repository methods
}
```

- We obtain Hibernate's Session object from JPA's **entityManager** in the constructor.
- **@Autowired** automatically finds the matched Spring bean (by type) in the method arguments. Thus, the **entityManager** will be automatically matched.
- **@Autowired** is a Spring's feature called **Dependency Injection**. It be used to annotate an instance method (for auto-loading arguments) and to annotate an instance variable (for setting it automatically).

Repository implementation: CRUD

- **Create** a new guest book entry:

```
@Override
@Transactional
public void addEntry(GuestBookEntry e) {
    this.session.persist(e);
}
```

- **Read** all guest book entries:

```
@Override
@Transactional
public List<GuestBookEntry> listEntries() {
    String hql = "FROM GuestBookEntry";
    Query<GuestBookEntry> query = this.session.createQuery(hql, GuestBookEntry.class);
    List<GuestBookEntry> entriesList = query.list();
    return entriesList;
}
```

Hibernate Query Language

Repository implementation: CRUD (cont’)

- **Read** a guest book entry with a particular ID:

```
@Override
@Transactional
public GuestBookEntry getEntryById(long id) {
    GuestBookEntry e = this.session.getReference(GuestBookEntry.class, id);
    Hibernate.initialize(e);
    return e;
}
```

- Due to lazy loading, the GuestBookEntry object **e** may not be accessible after the Hibernate Session is closed.
- We need to use **Hibernate.initialize(e)** to make sure the object **e** is loaded before the session is closed.

Repository implementation: CRUD (cont’)

- **Update** a guest book entry with a particular ID:

```
@Override
@Transactional
public void updateEntry(GuestBookEntry e) {
    this.session.merge(e);
}
```

- **Delete** a guest book entry with a particular ID:

```
@Override
@Transactional
public void removeEntryById(long id) {
    GuestBookEntry e = this.session.getReference(GuestBookEntry.class, id);
    if (e != null) {
        this.session.remove(e);
    }
}
```

HelloSpringHibernate: Controller

GuestBookController.java

```
@Controller
@RequestMapping("/guestbook")
public class GuestBookController  {

    @Resource
    private GuestBookEntryRepository gbeRepo;

    @GetMapping({"", "/"})
    public String index(ModelMap model) {
        model.addAttribute("entries", gbeRepo.listEntries());
        return "GuestBook";
    }

    @GetMapping("/add")
    public ModelAndView addCommentForm() {
        return new ModelAndView("AddComment", "command", new GuestBookEntry());
    }

    @PostMapping("/add")
    public View addCommentHandle(@ModelAttribute("entry") GuestBookEntry gbEntry) {
        gbEntry.setDate(new Date());
        gbeRepo.addEntry(gbEntry);
        return new RedirectView(".");
    }
```

Work the same as **@Autowired**

HelloSpringHibernate: Controller (cont’)

GuestBookController.java

```
@GetMapping("/edit/{id}")
public String editCommentForm(@PathVariable("id") long entryId, ModelMap model) {
    GuestBookEntry entry = gbeRepo.getEntryById(entryId);
    if (entry == null) {
        return "redirect:/guestbook";
    }
    model.addAttribute("entry", entry);
    return "EditComment";
}

@PostMapping("/edit/{id}")
public String editCommentHandle(@PathVariable("id") long entryId,
                                          @ModelAttribute("entry") GuestBookEntry gbEntry) {
    if (gbEntry.getId() == entryId) {
        gbEntry.setDate(new Date());
        gbeRepo.updateEntry(gbEntry);
    }
    return "redirect:..";
}
```

HelloSpringHibernate: Controller (cont’)

GuestBookController.java

```
    @GetMapping("/delete/{id}")
    public String deleteEntry(@PathVariable("id") long entryId) {
        GuestBookEntry entry = gbeRepo.getEntryById(entryId);
        if (entry == null) {
            return "redirect:/guestbook";
        }
        gbeRepo.removeEntryById(entryId);
        return "redirect:/";
    }
}
```

JDBC vs. ORM

- Relational databases are organized in rows and columns.

- Using JDBC:
  - It is tedious to read data from database in a row-by-row manner to object-oriented (OO) programs.
  - It is also tedious to update data from OO programs to database.

- Using ORM tool (e.g., Hibernate):
  - ORM tool maps data in relational database to entity object.
  - It simplifies the data reading and updating operations in OO programs.

Jakarta Persistence API (JPA) vs. Hibernate

- **Hibernate** is an **ORM tool**.
  - Other ORM tool examples: TopLink, Java Data Objects (JDO).

- **Jakarta Persistence API (JPA)** is a specification of an **interface**. You can choose any implementation available on the market. E.g.,
  - The programs can be written according to the JPA interface.
  - We can configure our programs to use **Hibernate** as the implementation, and switch to another implementation (e.g., EclipseLink) later.

- If a program is written to use **Hibernate** without JPA, it is locked in and switching to use another persistence provider requires **a fair amount of code change**.

JPA Components

- **ORM**: mechanism to map objects to relational data
- **Entity manager (~Hibernate Session)** to perform CRUD operations (Create, Read, Update and Delete)
- **Jakarta Persistence Query Language (JPQL)** to retrieve data with an OO query language
- Transaction and locking mechanism to protect data from being corrupted under concurrent access
- Callbacks and Listeners

- When using **JPA**, we still need **a lot of boilerplate code** for interacting directly with the Entity Manager.
- **Spring Data JPA** is a better solution.

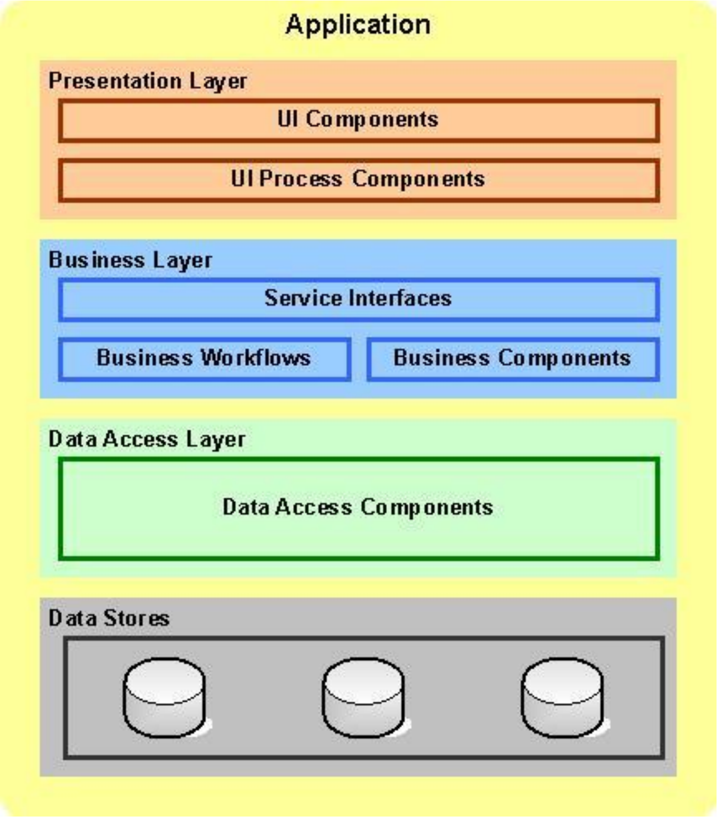
Spring Data

Motivations:

- Best practice indicates you should have Data Access Components or a Data Service Layer.
- People often make simple mistakes interacting with EntityManager.

Spring Data:

- An open-source project managed by SpringSource.
- Rely on Spring Framework.
- Provide a set of common patterns for persisting data using existing libraries.
- **Provide a JPA Repository Interface**
- **Automatic JPA Repository Implementation**



Spring Data JPA with Hibernate

Webapp example: *HelloSpringDataJPA*

lecture08-hellospringdatajpa

- Starter dependency for Spring Data JPA:

```
implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
```

- Properties in `application.properties`:

```
spring.datasource.url=jdbc:h2:./Data/myDB;AUTO_SERVER=TRUE
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

➢ Note that we don’t need the property `spring.jpa.properties.hibernate.current_session_context_class`

Spring Data JPA: Automatic JPA Repository

GuestBookEntryRepository.java

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface GuestBookEntryRepository extends JpaRepository<GuestBookEntry, Long>{
}
```

- JpaRepository is parameterized such that it knows
  - This is a repository for persisting **GuestBookEntry** objects, and
  - **GuestBookEntry** objects have an ID of type **Long**.
- We only need to define the above interface and we **do not need to implement repository methods**.
- Spring Data JPA will ***automatically generate*** a set of CRUD implementations, including
  - `findAll, count, delete, deleteById, deleteAll, exists, findById, save`
  - See this link for the complete list: http://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html

Spring Data JPA: Automatic JPA Repository (cont’)

GuestBookEntryRepository.java

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface GuestBookEntryRepository extends JpaRepository<GuestBookEntry, Long>{
}
```

Spring Data JPA will ***automatically generate*** CRUD implementations, e.g.,

- `GuestBookEntry` **`save`**`(GuestBookEntry entity)`
  - Insert a new entry into the table
- **`Optional`**`<GuestBookEntry>` **`findById`**`(Long id)`
  - Find the entry with the given id
- `List<GuestBookEntry>` **`findAll`**`()`
  - Find all the guestbook entries in the table
- `void` **`delete`**`(GuestBookEntry entity)`
  - Delete the guestbook entry from the table
- Besides generated methods, we can also implement our own methods.

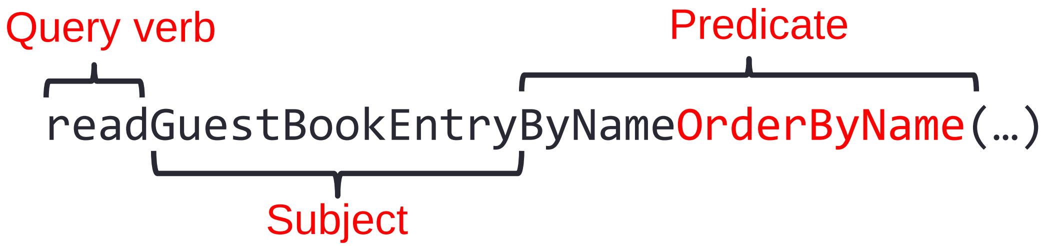
Define repository method 1: By name convention

- There are three ways to define our own functions:
  1. by **name convention**
  2. by **@Query annotation**
  3. by **implementation**
- We can create a method in the repository interface by following a name convention of Spring Data JPA.
- Then, its implementation will be generated automatically by Spring Data.

E.g.,

```
List<GuestBookEntry> readGuestBookEntryByName(String name);
```

- This method will give all guestbook entries from the specified user.
- Naming convention:



More details: **https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html**

Define repository method 2: By @Query

- We can also create a method in the repository interface by giving custom **JPQL query**. Spring Data JPA will automatically generate its implementation.
- E.g.,

```
@Query("select s from Spitter s where s.email like '%gmail.com' ")
List<Spitter> findAllGmailSpitters();
```

- This method is also useful when the naming convention gives a long method name and you want to replace it with a shorter one.

Define repository method 3: By implementation

- We can follow the **traditional JPA way** to define a repository method.

```
public class SpitterRepositoryImpl implements SpitterSweeper {
   @PersistenceContext
   private EntityManager em;

   public int eliteSweep() {
      String update =
         "UPDATE Spitter spitter " +
         "SET spitter.status = 'Elite' " +
         "WHERE spitter.status = 'Newbie' " +
         "AND spitter.id IN (" +
         "SELECT s FROM Spitter s WHERE (" +
         " SELECT COUNT(spittles) FROM s.spittles spittles) > 10000" +
         ")";
      return em.createQuery(update, Spitter.class).executeUpdate();
   }
}
```

```
public interface SpitterSweeper {
   int eliteSweep();
}
```

```
public interface SpitterRepository
   extends JpaRepository<Spitter, Long>, SpitterSweeper {
     ....
}
```

Reference: Spring in Action, 6th edition, Manning

HelloSpringDataJPA: Controller

GuestBookController.java

```
@Controller
@RequestMapping("/guestbook")
public class GuestBookController  {

    @Resource
    private GuestBookEntryRepository gbeRepo;

    @GetMapping({"", "/"})
    public String index(ModelMap model) {
        model.addAttribute("entries", gbeRepo.findAll());
        return "GuestBook";
    }

    @GetMapping("/add")
    public ModelAndView addCommentForm() {
        return new ModelAndView("AddComment", "command", new GuestBookEntry());
    }

    @PostMapping("/add")
    public View addCommentHandle(@ModelAttribute("entry") GuestBookEntry gbEntry) {
        gbEntry.setDate(new Date());
        gbeRepo.save(gbEntry);
        return new RedirectView(".");
    }
```

Generated by Spring Data JPA

No implementation is needed

No implementation is needed

HelloSpringDataJPA: Controller (cont’)

GuestBookController.java

```
@GetMapping("/edit/{id}")
public String editCommentForm(@PathVariable("id") long entryId, ModelMap model) {
    GuestBookEntry entry = gbeRepo.findById(entryId).orElse(null);
    if (entry == null) {
        return "redirect:/guestbook";
    }
    model.addAttribute("entry", entry);
    return "EditComment";
}

@PostMapping("/edit/{id}")
public String editCommentHandle(@PathVariable("id") long entryId,
                                                  @ModelAttribute("entry") GuestBookEntry gbEntry) {
    if (gbEntry.getId() == entryId) {
        gbEntry.setDate(new Date());
        gbeRepo.save(gbEntry);
    }
    return "redirect:..";
}
```

No implementation is need.

`findById` returns an `Optional<GuestBookEntry>` object.

No implementation is needed

HelloSpringDataJPA: Controller (cont’)

GuestBookController.java

```
    @GetMapping("/delete/{id}")
    public String deleteEntry(@PathVariable("id") long entryId) {
        GuestBookEntry entry = gbeRepo.findById(entryId).orElse(null);
        if (entry == null) {
            return "redirect:/guestbook";
        }
        gbeRepo.deleteById(entryId);
        return "redirect:/";
    }
}
```

No implementation is need.

`findById` returns an `Optional<GuestBookEntry>` object.

No implementation is needed