TECH ELEVATOR

Java

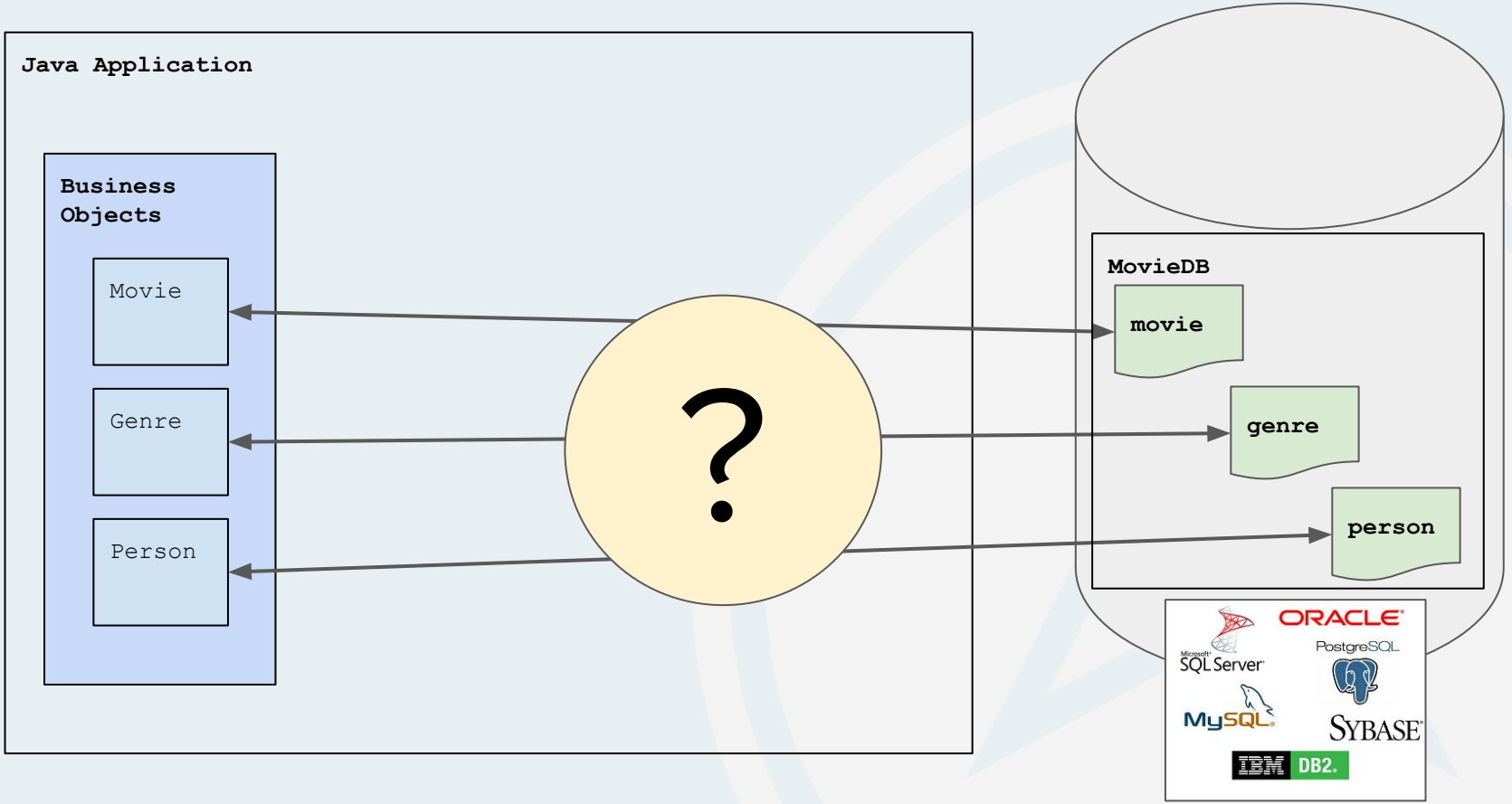
Data Access
Data Security

Module 2 - Week 3

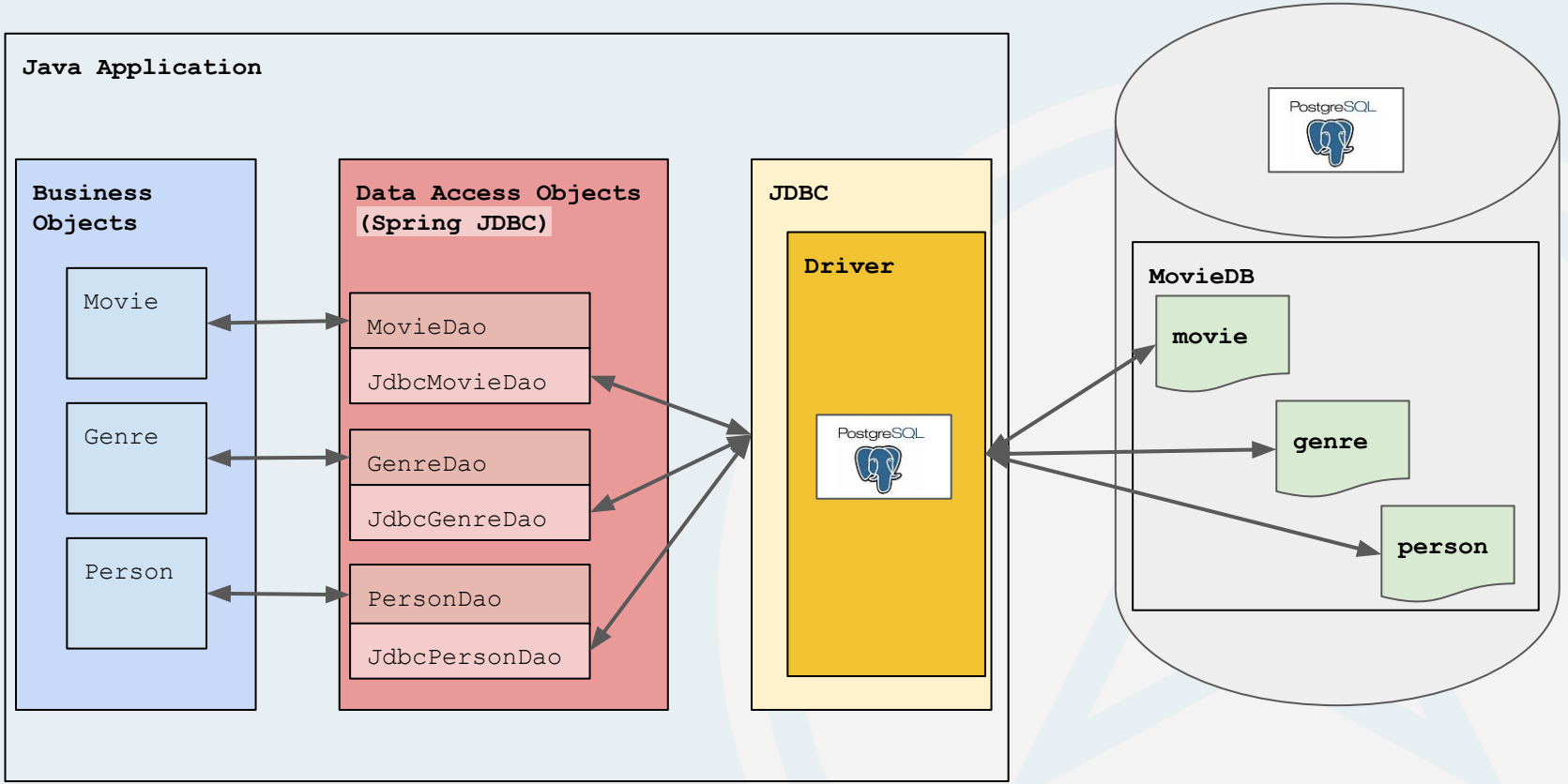
Schedule

Week	Topics
Week 0	Module 2 Orientation / PostgreSQL
Week 1	Intro to databases / Ordering, limiting, and grouping
Week 2	SQL joins / Insert, update, and delete / Database design
Week 3	Data Access / Data security
Week 4	DAO testing
Week 5	**Mid-module project**
Week 6	Postman / NPM / Networking and HTTP / Consuming RESTful APIs
Week 7	Server-side APIs
Week 8	Securing APIs
Week 9	**End-of-module project**
Week 10	**Assessment**

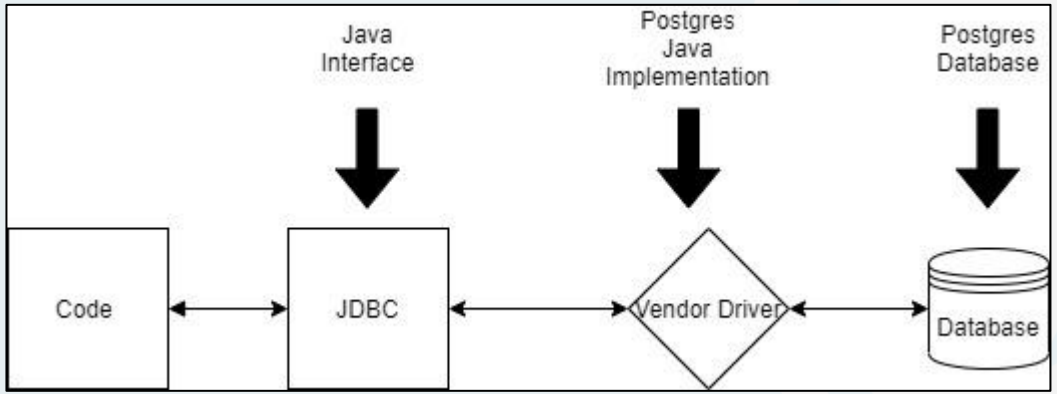
Data Access



Data Access

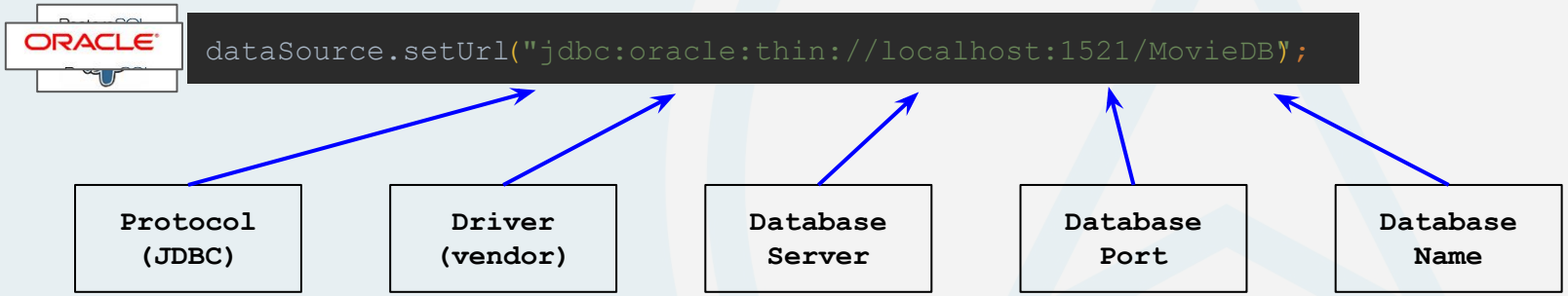


- The Java Database Connectivity (JDBC) API provides universal data access from the Java programming language.
- JDBC acts as an abstraction layer on top of the actual database implementation.
- Each vendor provides an implementation of the interface specific to its code.
- The vendor's implementation is known as a driver.
- We use the JDBC interface to communicate with the vendor's driver.
- This makes it fairly easy to swap out one vendor's database for another by swapping out the JDBC vendor driver.



JDBC DataSource & Connection

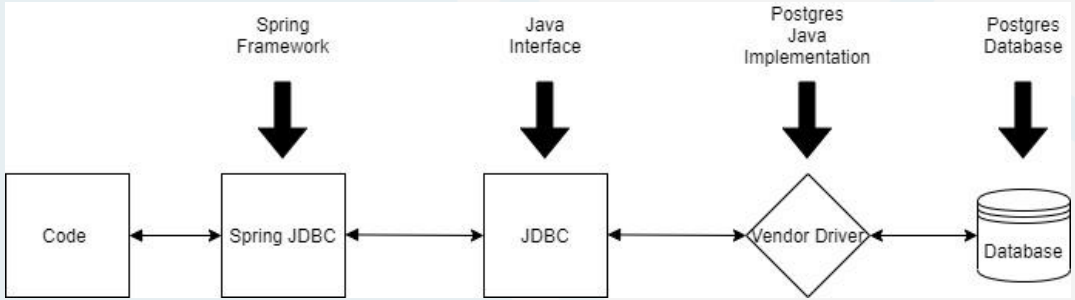
- The JDBC `DataSource` object is the preferred means of getting a connection to a data source. The `DataSource` acts as a connection factory.
- When we interact with a database, we need to create a connection.
  - Connections remain open until they are closed or time out.
  - Connections have overhead when created and opened, thus there is often a finite number of connections.
  - A connection pool can be used to reuse a few connections to conserve resources within an application by allowing the application to acquire a connection and release it when it is no longer needed so it can be reused.



The JDBC interface provides a standard way of working with databases in Java but using it can be very cumbersome.

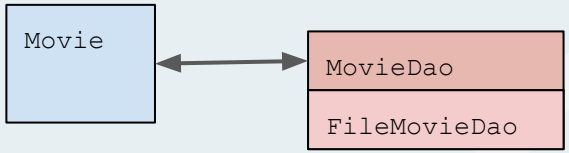
Spring is a popular Java framework. It is made up of many modules including Spring JDBC, which is is intended to solve some of the problems with using JDBC.

- Abstracts away boilerplate code so you can focus on YOUR code rather than all the nuts and bolts of interfacing with JDBC.
- Does most of the clean-up automatically for you.
- Simplifies SQL Exceptions in a way that makes them much easier to handle without a lot of extra code.
- Greatly simplifies working with Transactions.



DAO Pattern

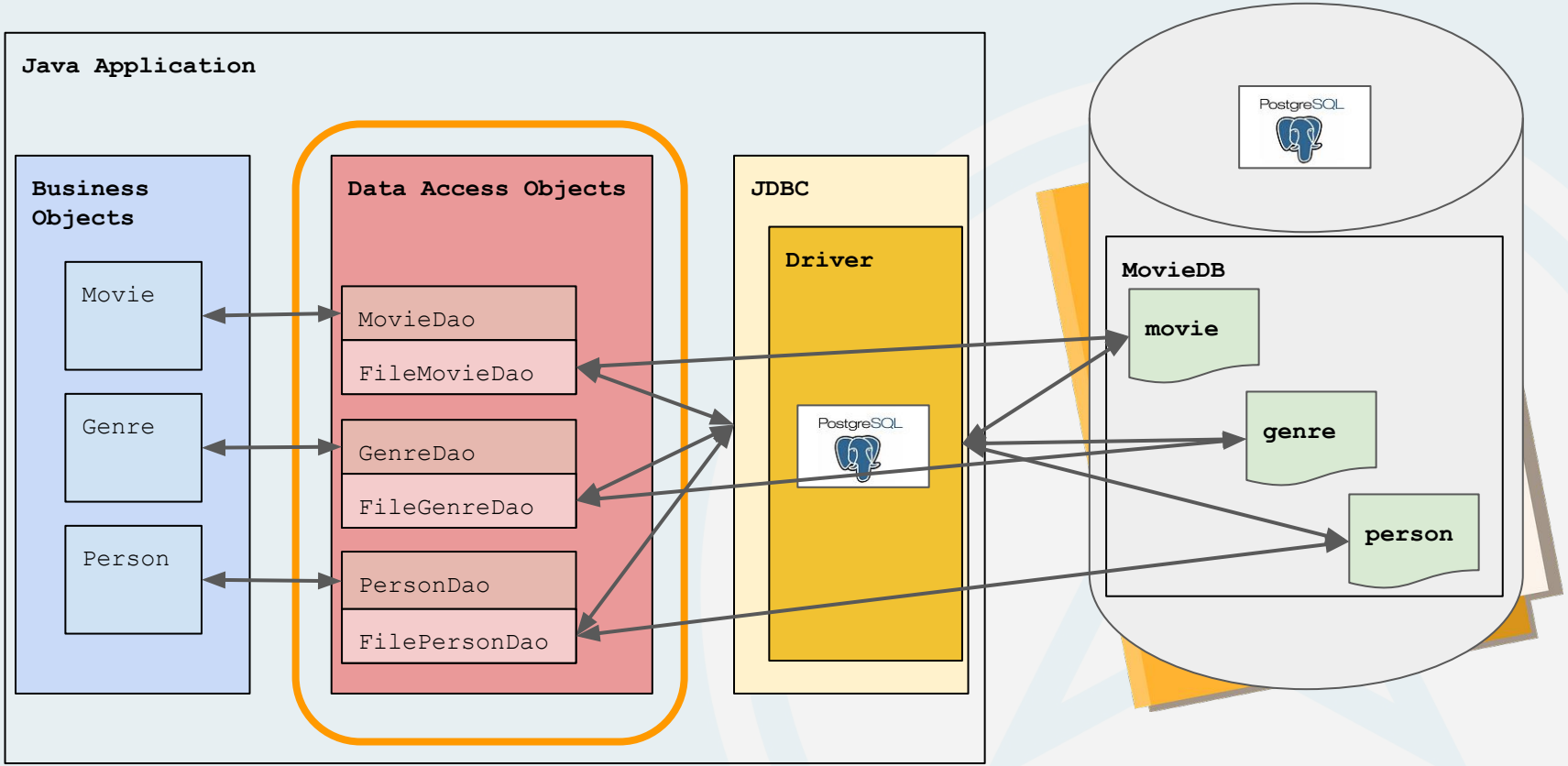
The Data Access Object (DAO) pattern is a structural pattern that allows us to isolate the application/business layer from the persistence layer (usually a relational database) using an abstract API.



The model objects interact with the DAOs through interfaces that expose the access to the associated data. A DAO interface can be different implementations that can be swapped with no change to the model code.

- Encapsulation
  DAO classes keep the logic for interacting with a data store separate from the rest of the application logic.
- Loosely coupled
  DAO interface abstracts away the specifics of the underlying data storage, so that the application and DAO have no knowledge of each other. This allows you to replace the DAO with one that accesses a different data source with little to no change to other code.
- Single responsibility principle
  This principle states that every class or function of an application should have responsibility over a single part of that program's functionality. DAO classes have responsibility over a single type of object, such as a relational database table.

DAO



`JdbcTemplate` is the central class in the Spring JDBC package.

- Simplifies the use of JDBC and helps to avoid common errors
- Executes core JDBC workflow, leaving application code to provide SQL and extract results
- This class executes SQL queries or updates
- Initiates iteration over `ResultSet`
- Heavily overloaded methods: `query`, `queryForRowSet`, `queryForObject`, `update`
- Constructed with a JDBC `Datasource`

```
// Create DataSource
BasicDataSource dataSource = new BasicDataSource();
dataSource.setUrl("jdbc:postgresql://localhost:5432/MovieDB");
dataSource.setUsername("postgres");
dataSource.setPassword("postgres1");

// Create a JdbcTemplate using the Datasource
JdbcTemplate jdbcTemplate =  new JdbcTemplate(dataSource);
```

The `JdbcTemplate queryForRowSet` method allows you to pass a string that contains the SQL query and get back a `SqlRowSet`, which represents the rows returned.

```
SqlRowSet movies = jdbcTemplate.queryForRowSet("SELECT movie_id, title FROM movie");
```

The `JdbcTemplate queryForObject` method allows you to pass a string that contains the SQL query and get back a single object. The returned object will be an object that holds the type of data you are querying for.

```
Integer cnt = jdbcTemplate.queryForObject("SELECT COUNT(*) FROM movie", Integer.class);
```

The `JdbcTemplate query` method allows you to pass a string that contains the SQL query and a `RowMapper` to be used by the Spring framework to map each row of the result to an object.

```
List<Movie> movies = jdbcTemplate.query("SELECT * FROM movie", MAPPER);
```

A `SqlRowSet` contains a data set representing the results of a query through the `JdbcTemplate queryForRowSet` method.

```
SqlRowSet movies = jdbcTemplate.queryForRowSet("SELECT movie_id, title FROM movie");
```

- As the **`next()`** method is called, the cursor advances and points to the next row in the result set.
- We can query the current row by using methods like **`getLong`**, **`getInt`**, and **`getString`** that take the column name as a parameter.
- We can iterate over a result set and map each row to a new instance of the object we are querying for.

Data Output | Explain | Messages | Notifications

	movie_id [PK] integer	title character varying (200)
1	1891	The Empire Strikes Back
2	120	The Lord of the Rings: The Fellowship of the Ring
3	27205	Inception
4	121	The Lord of the Rings: The Two Towers
5	101	Léon: The Professional
6	299534	Avengers: Endgame
7	299536	Avengers: Infinity War

```
while(movies.next()) {
    String title = movies.getString("title");
    Long id = movies.getLong("movie_id");
    // use results to create a new Movie instance from the current row
}
```

SQL Parameters

The `JdbcTemplate` methods can make parameterized queries. We can provide a SQL query string with question marks (`?`) as parameter placeholders along with a list of parameter variables to populate the query. This makes our queries reusable and helps to prevent SQL injection.

With `jdbcTemplate queryForRowSet` ...

```
SqlRowSet movies = jdbcTemplate.queryForRowSet(
    "SELECT * FROM movie WHERE movie_id =  ?", movieId);
```

(Alternate approach) With `jdbcTemplate query` ...

```
List<Movie> movies = jdbcTemplate.query(
    "SELECT * FROM movie WHERE movie_id =  ?", MAPPER, movieId);
```

The `JdbcTemplate update` method allows you to pass a string that contains a SQL statement which will be executed in the database (notice the parameter placeholder `?`). The `update` is used when we don't need to return data.

```
jdbcTemplate.update("UPDATE movie SET length_minutes = 300 WHERE movie_id = ?", id);
```

```
jdbcTemplate.update(
 "INSERT INTO city(city_name, state_abbreviation, population, area)VALUES(?, ?, ?, ?)",
 name, state, population, area);
```

```
jdbcTemplate.update("DELETE FROM movie WHERE movie_id = ?", id);
```

When inserting a record, it is often important to get the id that was generated by PostgreSQL. We can do this using the `INSERT … RETURNING` syntax. We use `queryForObject` rather than

```
Long newID = jdbcTemplate.queryForObject(
  "INSERT INTO city (city_name, state_abbreviation, population, area) " +
  "VALUES (?, ?, ?, ?) RETURNING city_id;", Long.class, name, state, population, area);
```

DTO Pattern

- DTOs or Data Transfer Objects are objects that carry data between processes in order to reduce the number of methods calls.
- Usually these will only have data members and getters/setters.

Mapping Method

We can create a method that can be called to map the current row in the SqlRowSet to a Java Object.

```
List<Movie> movieList = new ArrayList<>();
SqlRowSet rowSet = jdbcTemplate.queryForRowSet("SELECT movie_id, title FROM movie");
while(rowSet.next()) {
    movieList.add(mapRowToMovie(rowSet));
}
```

The benefit of creating a mapping method in your DAO class is that it can be reused for any of your DAO methods that get a row set and need to map each row to an object.

```
private Movie mapRowToMovie(SqlRowSet rowSet) {
     int id = resultSet.getInt("movie_id");
     String title = resultSet.getString("title");
     // … get other Movie attributes
     return new Movie(id, title, ...);
}
```

Break

Objectives

- Define Data Access Object and explain why it's good for decoupling code
- **Describe the SQL injection vulnerability and the process of exploiting it in vulnerable code**
- Write code in Java that implements data access in a safe way

SQL Injection

SQL injection occurs when untrusted data such as user data from application web pages are added to database queries, materially changing the structure and producing behaviors inconsistent with application design or purpose.
Clever attackers exploit SQL injection vulnerabilities to steal sensitive information, bypass authentication or gain elevated privileges, add or delete rows in the database, deny services, and in extreme cases, gain direct operating system shell access, using the database as a launch point for sophisticated attacks against internal systems.

Using our **UnitedStates** database, let's say we let someone enter a city id and then display the information for the city using this SQL String:

```
String query = "SELECT * FROM city WHERE city_id = " + cityId;
```

If **237** is entered for the customer ID, the result from the DB will be the city row with **city_id 237**.

But what happens when the user enters the following?

- **237 OR 1 = 1**
- **237; DELETE FROM park_state**

What can be done to prevent this type of attack?

SQL Injection Prevention

- **Parameterized Queries**: The single most effective thing you can do to prevent SQL injection is to use parameterized queries. If this is done, SQL injection should not be possible.

- **Input Validation**: Limiting or sanitizing the data that is input by a user can certainly be helpful in preventing SQL Injection, but is by no means an effective prevention by itself.

- **Limit Database User Privileges**: A web application should always use a database user to connect to the database that has as few permissions as necessary. For example, if your application never deletes data from a particular table, then the database user should not have permission to delete from that table. This will help to limit the damage in the case that there is a SQL Injection vulnerability.

Break

Objectives

- Define Data Access Object and explain why it's good for decoupling code
- Describe the SQL injection vulnerability and the process of exploiting it in vulnerable code
- **Write code in Java that implements data access in a safe way**

Protecting Sensitive Data

Many data breaches involve sensitive data that has not been stored in a safe manner, meaning if the attacker gets to the data they can easily understand it.
There are many cases in which we need to be able to store data such that it is not readable by unauthorized parties:

- Passwords
- PIN
- Credit card numbers
- Bank account numbers

Depending on the nature of the **data** and how it's used, it **can be protected using** one of two techniques: **hashing** or **encryption**.

Passwords

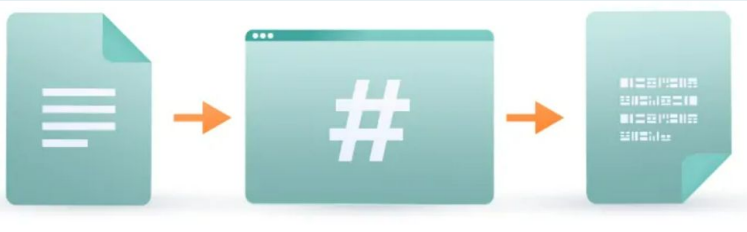
One type of sensitive data systems need to store is user passwords.
If passwords are kept in plain text, anyone with internal access can see them and if the database gets breached, hackers would also see the credentials in plain view.

username	password
alice	happy1
bob	password123
jason	qwerty
mike	happy1

- We need to be able to verify a password but not recover it.
- A system administrator with access to credential data should not be able to determine a password.
- Any hacker that steals a database or set of credentials should not be able to read the passwords.
- Even with super-computing capabilities, no one should be able to access the data within any reasonable amount of time.

Hashing

- Uses a **one-way function to scramble** the plain-text password prior to storage.
- Transforms the password into a completely different string of characters with a set length.



plain text (**password123**) hashing function hashed text (**86861c4aec4eb67df463fb0fe96a3127**)

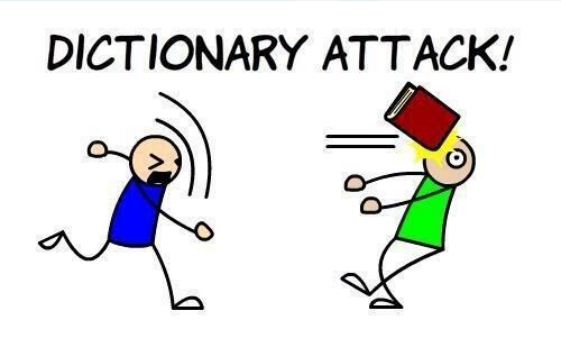
username	password_hash
bob	86861c4aec4eb67df463fb0fe96a3127
alice	3b2ba79307ae54030de4185d09171a9e

- When you log into your account, the password you enter is hashed using the password hashing function.
- The hashed password is compared to the hash stored in the database for your user account.
- If the password hash matches, you're granted access to your account.

```
bob, password123 => 86861c4aec4eb67df463fb0fe96a3127
```

Cracking a Hash

- If you run a specific word like 'java' through a hashing function, the hashed output will always be the same for that word.
- Hackers can use software to generate hashes for millions of password guesses by running them through known hashing functions. A password which is a direct entry in the English dictionary is especially vulnerable.
- Once a database is breached, the hackers can compare the password hash values from the database to hashes from their password guesses.
- If a match is found then the system can be accessed.
- These attacks are known as **dictionary attacks**.
- Adding **“salt”** to a hash helps defend against these types of attacks.



Needs Salt

- The output of a hash function is deterministic; when given the same input, the same hash is always produced.
- If two users have the same password their hash is also the same.
- Once a password is known, the same password can be used to access all the accounts that use that hash.

username	password_hash
alice	3b2ba79307ae54030de4185d09171a9e
bob	86861c4aec4eb67df463fb0fe96a3127
jason	3b2ba79307ae54030de4185d09171a9e

If Alice's password is '**happy1**', what do you think Jason's password is?

- To limit the possibility of a successful dictionary attack, we salt the passwords.
- Salt is a generated value that is added to the input of hash functions to create unique hashes for every input.
- A salt makes a hash function look non-deterministic; when given the same input, the same hash is not produced.

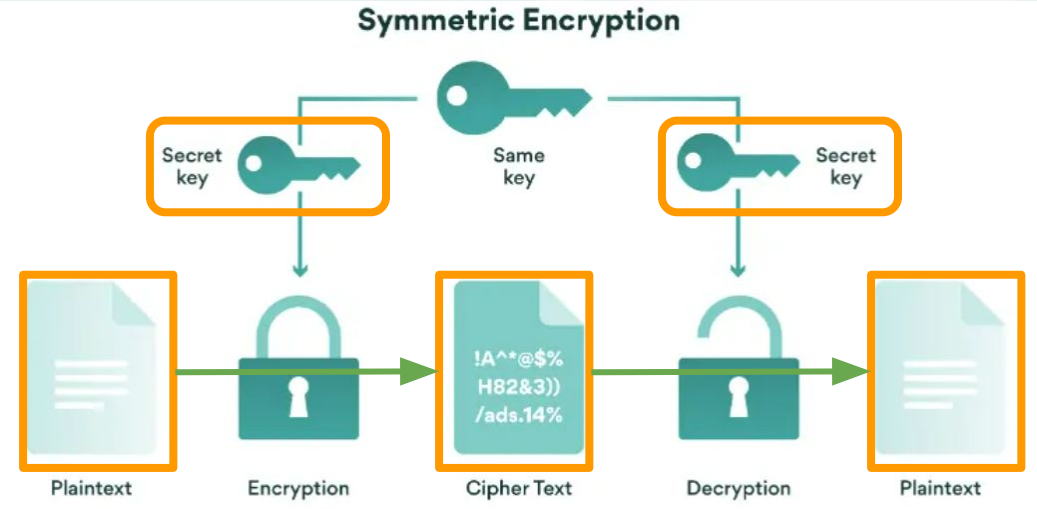
username	password_hash	salt
alice	060b5087575363068163e6e39e0f7b5f	f1nd1ngn3m0
bob	34285ac3f310665d46be96563330df6f	f1nd1ngn22o
jason	08af9b4a340e6367f0793e75eadac596	f1nd1ngn4z1

Encryption

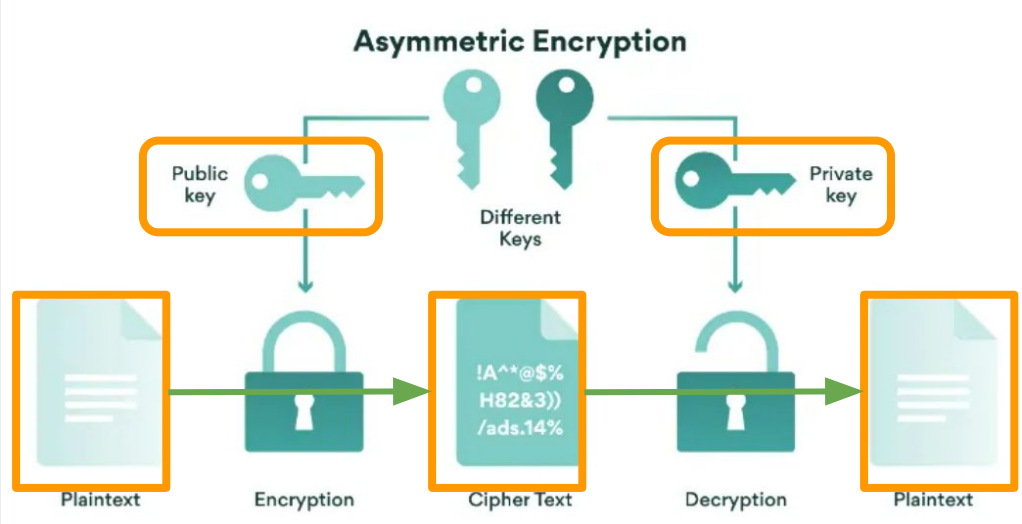
- **Encryption** is the most effective way to achieve data security.
- Information is encoded in such a way that only authorized parties can gain access to it.
- A combination of keys and algorithms is used to encode and decode sensitive information.
- Web encryption:
  - **Secure Socket Layer (SSL) and Transport Layer Security (TLS)** are examples of **asymmetric key encryption**.
  - **SSL** was developed by Netscape in 1994 to secure transactions over the WWW.
  - **TLS** was developed to address some of the security risks of SSL. These days, when people talk about SSL as a protocol, they are usually actually referring to TLS.
  - TLS and SSL are recognized as protocols to provide secure HTTP(S) for internet transactions. It supports authentication, encryption, and data integrity.

Securing Data at Rest

- Data at rest can use a form of encryption called **symmetric key encryption**
- Also known as private key encryption.
- Requires both parties to use a private key to encrypt and decrypt data.
- Any party possessing the key can read the data.
- Difficult to secure the symmetric key between multiple parties.



- Sometimes others need to send you secure data without worrying that it be intercepted.
- Giving the secure key away would not be a good decision.
- **Asymmetric algorithms** allow us to create a **public key** and a **private key**
- The public key is distributed freely.
- The private key is kept secret by the owner.
- If the public key is used for encryption, then the related private key is used for decryption. If the private key is used for encryption, then the related public key is used for decryption.



Person-In-The-Middle Attack

- Even communication performed over **SSL** is subject to a **PITM attack**.
- The browser will set a SSL session with the attacker while the attacker sets an SSL session with the web server.
- The **browser** will try and **warn** the **user** that the **digital certificate is not valid**, but the **user** often **ignores** the warning because the threat is not understood.