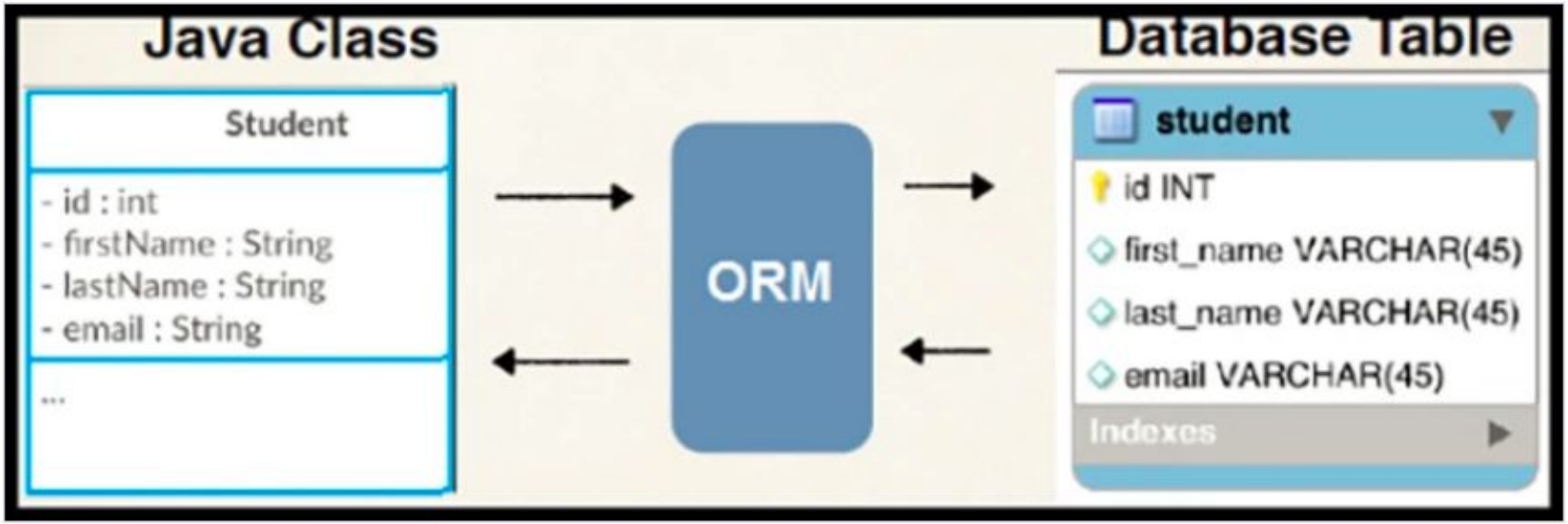
Spring [2]

Backend ITMO 2021

Object-Relational Mapping (ORM)



Java Persistence API (JPA)

- Entity
  - @Id - Access Type
  - property-based (Getter/Setter)
  - field-based
- Persistence Context
- Entity Manager
  - .persist(entity)

```java
@Entity
@Table(name = "games")
@EntityListeners(AuditingEntityListener.class)
public class Game {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    @Column(name = "player1", nullable = false)
    private long player1;

    @Column(name = "player2", nullable = false)
    private long player2;

    @Column(name = "field", nullable = false)
    private String field;
```

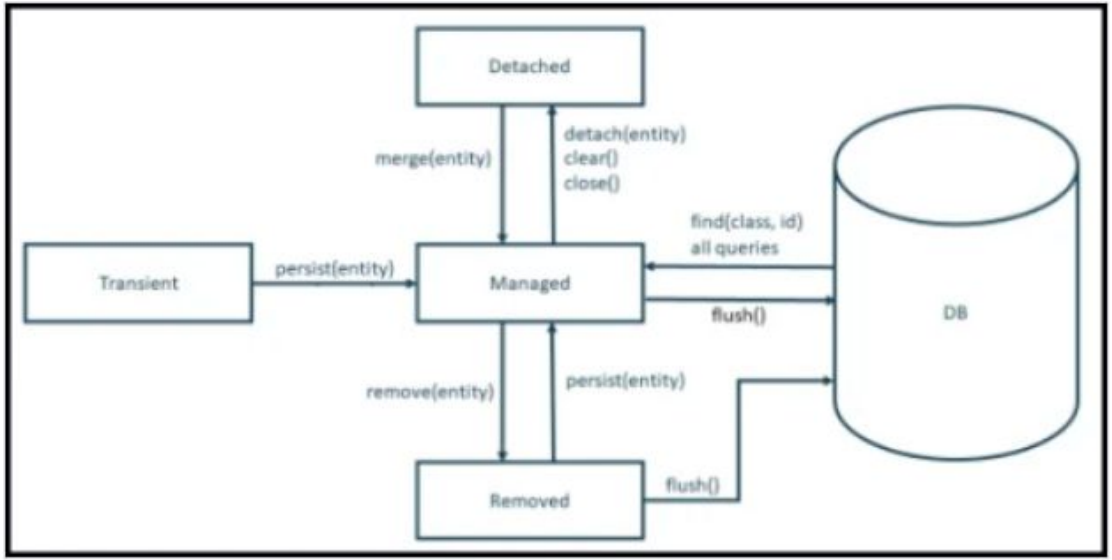
JPA [2]

- Transaction
  - begin()
  - commit()

```
1  @Before
2  public void init() {
3      EntityManagerFactory emf = Persistence.createEntityManagerFactory( "JavaRush" );
4      em = emf.createEntityManager();
5      em.getTransaction().begin();
6  }
7  @After
8  public void close() {
9      if (em.getTransaction().isActive()) {
10         em.getTransaction().commit();
11         }
12     em.getEntityManagerFactory().close();
13     em.close();
14 }
```

JPA [3]

- Entity Lifecycle
  - persist
  - detach
  - merge
  - remove



JPA [4]

- Mapping
  - Category [One] <- Topic [Many]
  - In Topic:

```
@ManyToOne
@JoinColumn(name = "category_id")
private Category category;
```

  - In Category:

```
@OneToMany(cascade = CascadeType.ALL)
@JoinColumn(name = "topic_id")
private Set<Topic> topics = new HashSet<>();
```

  - Setter in Topic:

```
public void setCategory(Category category) {
    category.getTopics().add(this);
    this.category = category;
}
```

JPA [5]

- Java Persistence Query Language (JPQL)

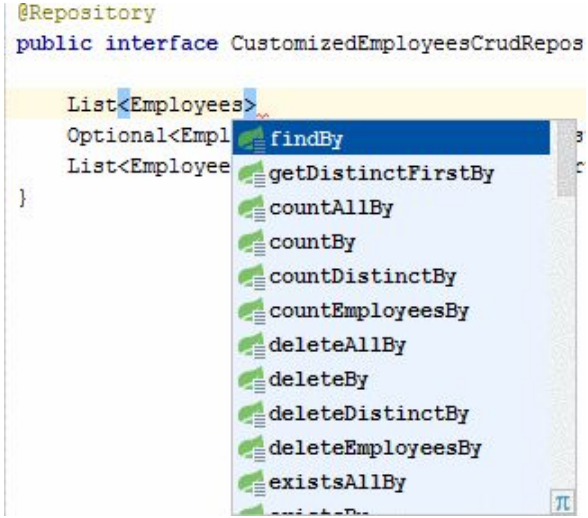
```
Query query = em.createQuery("SELECT c from Category c WHERE c.title = 'query'");
assertNotNull(query.getSingleResult());
```

- Criteria API
  - SELECT c FROM Category c

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Category> query = cb.createQuery(Category.class);
Root<Category> c = query.from(Category.class);
query.select(c);
List<Category> resultList = em.createQuery(query).getResultList();
```

Repository [1]

- public interface CrudRepository<T, ID extends Serializable> extends Repository<T, ID>
- Queries by method name




```
S save(S var1);
Iterable<S> saveAll(Iterable<S> var1);
Optional<T> findById(ID var1);
boolean existsById(ID var1);
Iterable<T> findAll();
Iterable<T> findAllById(Iterable<ID> var1);
long count();
void deleteById(ID var1);
void delete(T var1);
void deleteAll(Iterable<? extends T> var1);
void deleteAll();
```

Repository [2]

- Custom methods

```
public class CustomizedEmployeesImpl implements CustomizedEmployees {

    @PersistenceContext
    private EntityManager em;

    @Override
    public List getEmployeesMaxSalary() {
        return em.createQuery("from Employees where salary = (select max(sal
ary) from Employees )", Employees.class)
                .getResultList();
    }
}
```

- @Query

```
    @Query("select e from Employees e where e.salary > :salary")
    List<Employees> findEmployeesWithMoreThanSalary(@Param("salary") Long sa
lary, Sort sort);
```

Repository [3]

- Modifying queries -> @Modifying

```
@Modifying
@Query("update Employees e set e.firstName = ?1 where e.employeeId = ?2")
int setFirstnameFor(String firstName, String employeeId);
```

- Generic types

```
@Query("select t from #{#entityName} t where t.deleted = ?1")
List<T> findMarked(Boolean deleted);
```

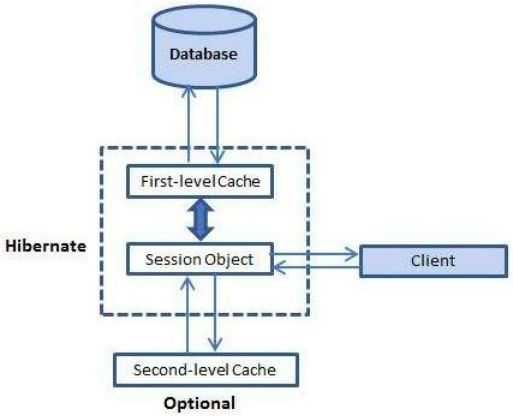
- save() creates transaction by default

Spring Data and Hibernate

- **JPA** - API which provides specification for persisting, reading, managing data
- **Hibernate** - provider which implements JPA. Most popular, default in Spring.
- **Spring Data** offers a solution to the Repository pattern or the legacy GenericDao custom implementations. It can also generate JPA queries on your behalf through method name conventions.

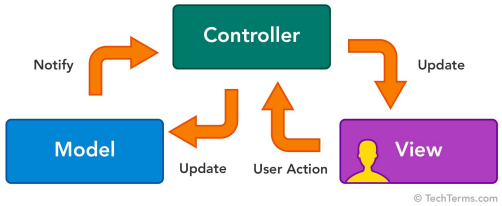
Caching

- Cache memory stores recently used data items in order to reduce the number of database hits as much as possible
- First-Level (Session)
  - If you issue multiple updates to an object, Hibernate tries to delay doing the update as long as possible to reduce the number of update SQL statements issued. If you close the session, all the objects being cached are lost and either persisted or updated in the database.
- Second-Level (Optional)
  - Can be configured on a per-class and per-collection basis and mainly responsible for caching objects across sessions
- Query-level
  - An optional feature and requires two additional physical cache regions that hold the cached query results and the timestamps when a table was last updated



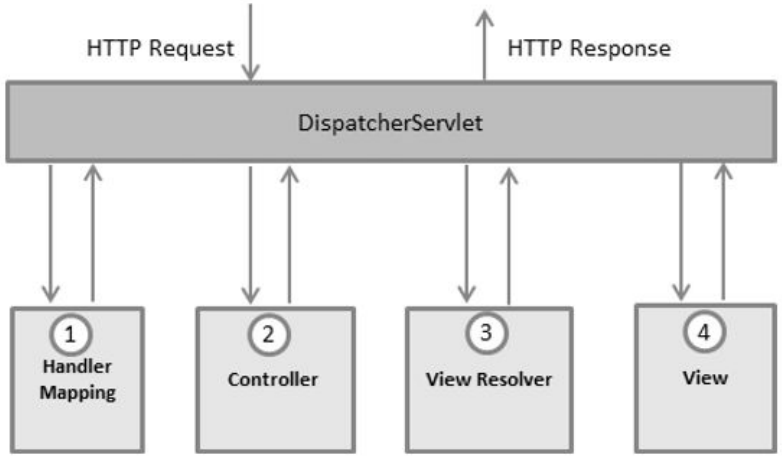
Model-View-Controller Pattern

- Model
  - Dynamic data structure, independent of the user interface
  - Responsible for managing the data of the application, it receives user input from the controller
- View
  - Any representation of information such as a chart, diagram or table
  - Renders presentation of the model in a particular format
- Controller
  - Accepts input and converts it to commands for the model or view
  - Responds to the user input and performs interactions on the data model objects



Spring MVC

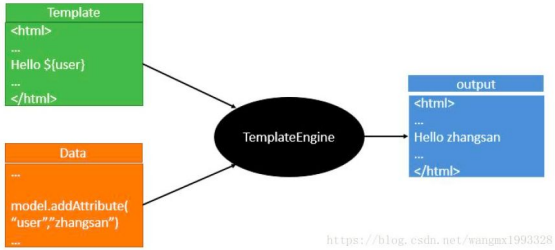
- Handler selects the controller and sends request to it
- Controller calls inner logic and return name of view to dispatcher
- Dispatcher selects exact view using View Resolver
- Dispatcher injects model arguments into created View



Thymeleaf

- Server-Side **Java Template Engine**
- HTML, XML, Javascript, CSS, Plain Text

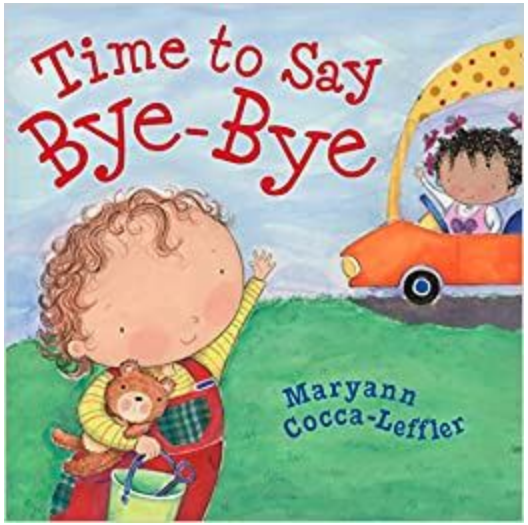
```
<table>
  <thead>
    <tr>
      <th th:text="#{msgs.headers.name}">Name</th>
      <th th:text="#{msgs.headers.price}">Price</th>
    </tr>
  </thead>
  <tbody>
    <tr th:each="prod: ${allProducts}">
      <td th:text="${prod.name}">Oranges</td>
      <td th:text="${#numbers.formatDecimal(prod.price, 1, 2)}">0.99</td>
    </tr>
  </tbody>
</table>
```



Deploying to cloud

- Heroku (PaaS)
  - heroku create
  - git push heroku master
  - heroku addons:create heroku-postgresql
  - Heroku will automatically populate the environment variables SPRING_DATASOURCE_URL, SPRING_DATASOURCE_USERNAME, and SPRING_DATASOURCE_PASSWORD. These environment variables should allow your Spring Boot application to connect to the database without any other configuration as long as you add a PostgreSQL JDBC driver to your dependencies
  - https://devcenter.heroku.com/articles/deploying-spring-boot-apps-to-heroku

The End



SB

Backend ITMO 2021