Enterprise Application Frameworks

- **Overview**
- **Spring Framework**
- **Movie Rental Application**

Enterprise Application Frameworks

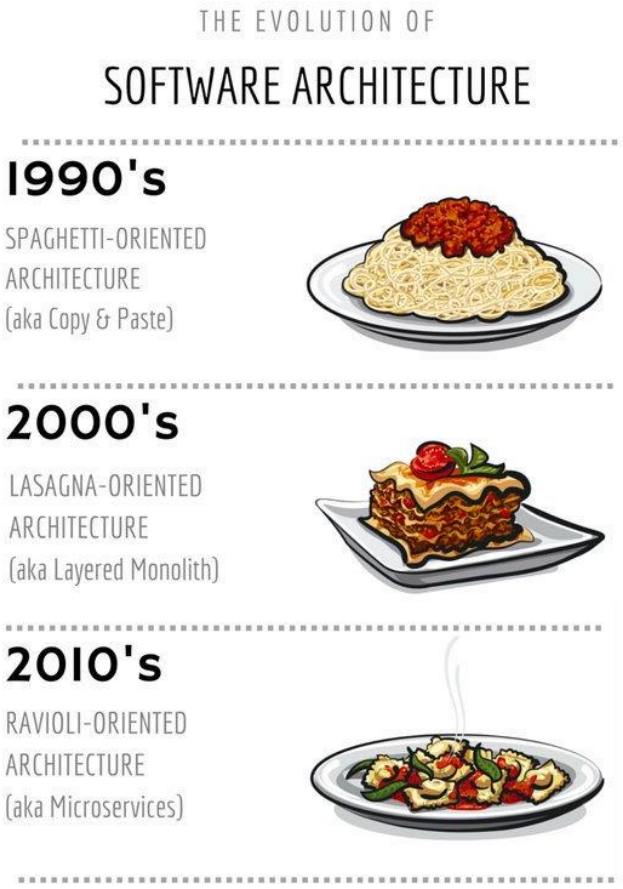
- **Teachers**
  - Dominik Gruntz, dominik.gruntz@fhnw.ch
  - Jürg Luthiger, juerg.luthiger@fhnw.ch

- **Materials**
  - \\fsemu18.edu.ds.fhnw.ch\e_18_data11$\E1811_Unterrichte_Bachelor\E1811_Unterrichte_I\5iw\eaf
  - https://gitlab.fhnw.ch/eaf/hs21

Enterprise Application Frameworks

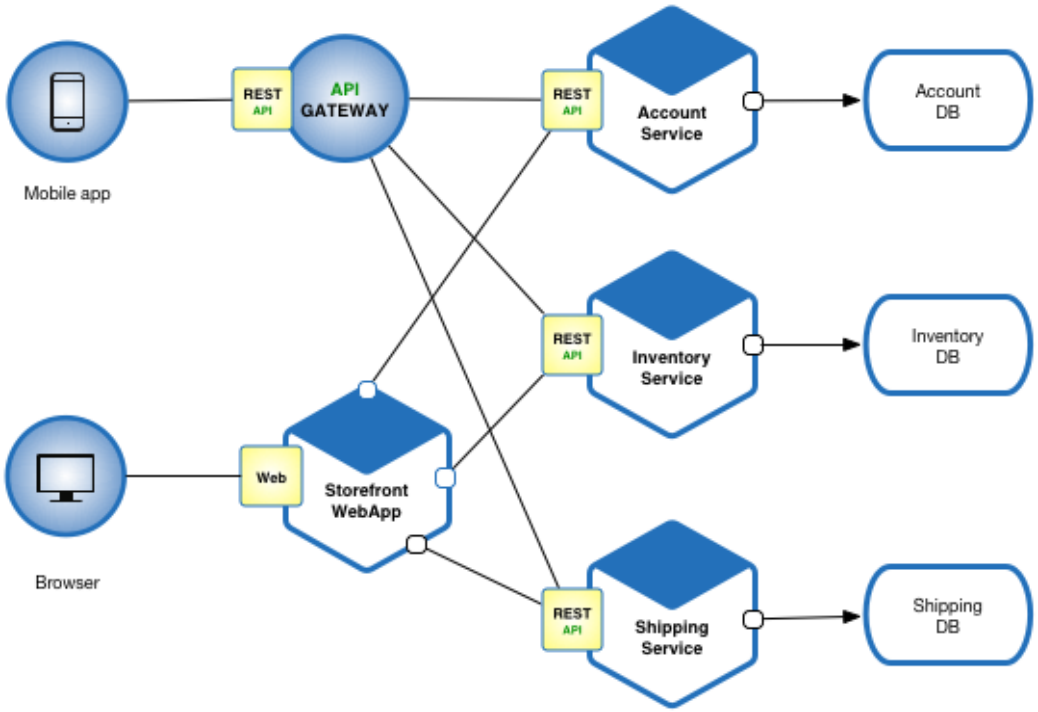
- **Topic**
  - The student learns how to realize powerful distributed applications with new technologies.
- **Learning Objectives**
  - The participants
    - know the most important architecture patterns for distributed applications
    - know the parts of Spring / Spring Boot
    - know the advantages of *Dependency Injection* (DI)
    - can model orthogonal aspects with *Aspect-Oriented Programming* (AOP)
    - can develop Spring Beans
    - can access a database with an O/R mapper (ORM: JPA, Spring Repositories)
    - can integrate different software components on one integration platform (such as Spring)
    - can implement *Design Patterns* for distributed applications

Enterprise Application Frameworks



Enterprise Application Frameworks

- **Microservices Architecture**



  - Spring Boot is well suited for the development of micro services

Enterprise Application Frameworks

- **Program:**
  - **Introduction** (1W)
    - Dependency Injection
  - **Persistence** (4W)
    - JPA: Relations, Queries
    - Spring Repositories
  - **Architecture** (4W)
    - Spring Boot & Docker, RESTful APIs
    - AOP & Transactions
  - **Reactive Programming** (2W)
    - Reactive Streams, Reactor, Spring WebFlux
  - **Spring in Production** (3W)
    - Testing
    - Security

Enterprise Application Frameworks

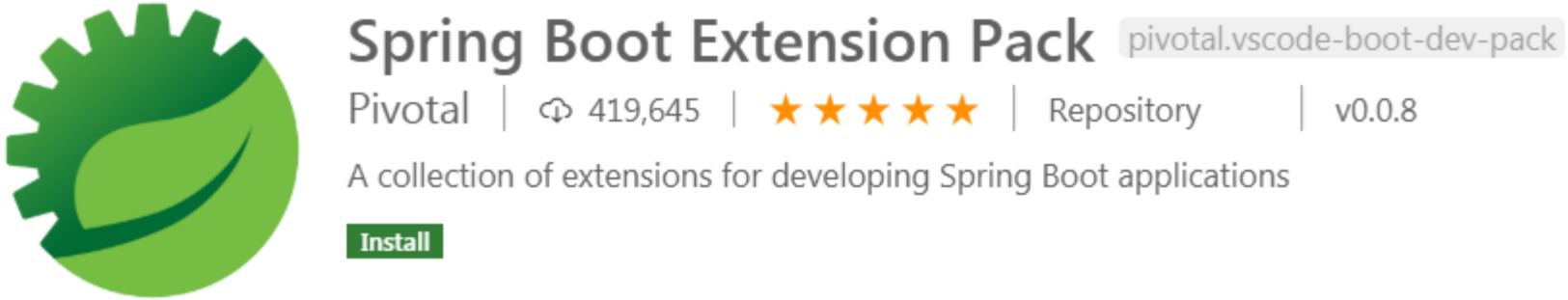
- **Examination**
  - Tuesday, November 16, 2020, 09:15 – 10:45, 6.-1D13
    - Part 1: 30 Min, without materials (closed book)
    - Part 2: 60 Min, open book, i.e. arbitrary written/printed material, but no electronic devices
  - Final written exam, date/room not yet fixed
    - Part 1: 30 Min, without materials (closed book)
    - Part 2: 60 Min, open book

Enterprise Application Frameworks

- **Resources**
  - Spring Website: https://spring.io
  - Spring Guides: https://spring.io/guides
    - Getting Started Guides
    - Topical Guides
    - Tutorials
  - Spring Blog: https://spring.io/blog
  - Spring Initializr: https://start.spring.io/

Spring Boot & Visual Studio Code

- **Spring Boot Extension Pack**



  - Spring Boot Tools
    - Language server providing support for application.* and *.java
  - Spring Initializr Java Support
    - Generation of new Spring Boot projects
  - Spring Boot Dashboard
    - Managing of Spring Boot projects

Spring Boot Extension Pack

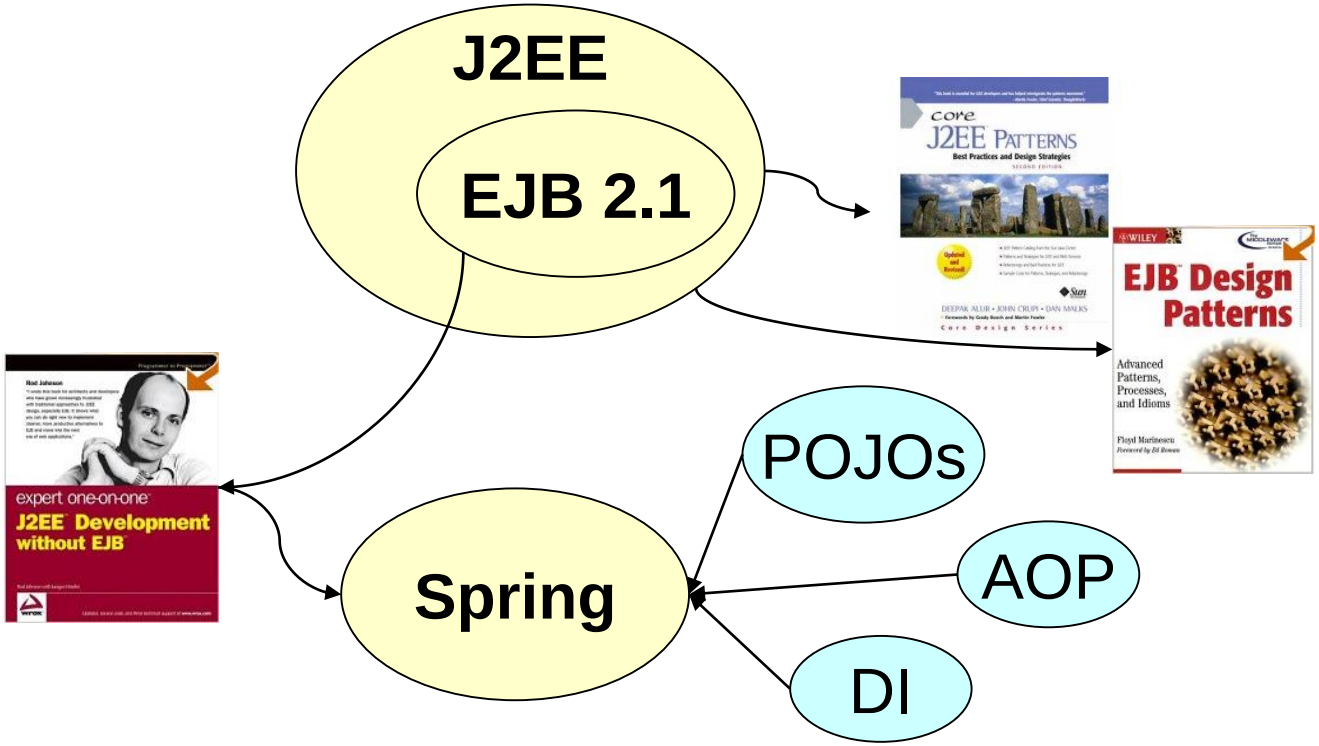
- **Spring Boot Tools**
  - Easy navigation to Spring-specific elements
  - Property file support: validation, code completion, information hoovers
    - application*.properties / application*.yml
- **Spring Initializr Java Support**
  - Based on spring initalizr (https://start.spring.io/)
  - Commands (F1 or CTRL-SHIFT-P)
    - Spring Initializr: Create a Gradle Project…
    - Spring Initializr: Create a Maven Project…
    - Spring Initializr: Add Starters…
      - Edit starters is only supported on Maven projects
- **Spring Boot Dashboard**
  - Display of all Spring Boot apps in current workspace
  - Starting/Stopping/Debugging Spring Boot applications

Enterprise Application Frameworks

- **Overview**
- **Spring Framework**
  - Core Concepts
  - Dependency Injection
  - Annotation based DI
- **Movie Rental Application**

Spring Framework

- **History**



Spring Framework

- **Core Concepts**
  - Dependency Injection (DI)
    - DI is a technique in which an object (client) receives other objects (services) on which it depends
    - Instead of the client accessing or creating its services, the dependencies are injected by an injector or assembler
    - DI enables configurability and testability
  - Aspect Oriented Programming (AOP)
    - AOP allows to isolate technical, non-functional aspects such as transactions, caching or security and keeps the actual code free of them
  - Templates
    - Templates are designed to simplify working with some APIs by automatically cleaning up resources and handling error situations consistently

Dependency Injection

- **Implementations**
  - Spring Framework
  - Google Guice
  - Hilt / Dagger (for Android)

- **Participants**
  - Components (Clients and Services)
  - Schema (assembly chart)
    - Defines which services are provided and required
  - Injector / Assembler
    - Does the actual wiring according to the schema
    - Provides access to the wired components

Dependency Injection in Spring

- **Components: Spring Beans**
  - Normal Java Objects (POJO = Plain Old Java Object)
  - A POJO which is under control of the Spring container is a Spring Bean
- **Injector/Assembler: Spring Container**
  - Spring container contains and controls the configuration and life-cycle of the spring beans (=> Application Context)
  - Spring container is light-weight, can be restarted for each unit test
- **Schema: Spring Configuration**
  - XML (=> Modul Design Patterns)
  - Annotations
  - Java Configuration
  - Convention over Configuration

@Component

- **@Component**
  - Marks a class as a Spring Bean
  - By default, the bean has the same name as the class name with a lowercase initial
  - A different name can be specified using the optional *value* argument of this annotation

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Indexed
public @interface Component {
   String value() default "";
}
```

@Component

- **Meta annotations of @Component**
  - @Service
    - Indicates that a class is a service or a business service facade.
  - @Controller
    - Indicates the a class is a web controller defining request mappings
  - @Repository
    - Indicates that a class is a repository (DDD), i.e. a mechanism for encapsulating CRUD operations on objects, also called DAOs
    - Provides automatic persistence exception translation
  - @Configuration
    - Indicates that a class declares one or more @Bean methods which return Spring Beans

@Bean

- **@Bean**
  - Indicates that a method produces a bean to be managed by the Spring container. Can be used to add external objects into the Spring context
  - Method is declared in a Bean marked with @Configuration
  - By default, the method name defines the name of the bean

```
@Configuration
public class DataSourceConfig {
  @Bean
  public DataSource getDataSource() {
    DataSourceBuilder dataSourceBuilder = DataSourceBuilder.create();
    dataSourceBuilder.driverClassName("org.h2.Driver");
    dataSourceBuilder.url("jdbc:h2:mem:test");
    return dataSourceBuilder.build();
  }
}
```

@Autowired

- **@Autowired**
  - Can be applied on (private) fields

```
@Service
public class MovieServiceImpl implements MovieService {
   @Autowired
   private MovieRepository movieRepo;
   …
```

  - Can be applied on setter methods
  - Can be applied on arbitrary methods with arbitrary arguments
  - Can be applied on constructors
    - At most one constructor can carry the @Autowired annotation
    - If the class has only one constructor, annotation is no longer necessary

@Autowired

- **Field vs Setter vs Constructor Injection**
  - With setter injection the setters are called one by one, i.e. not all fields are initially set which might lead to NPEs
  - With field injection the fields are initialized *after* the constructor was called
  - With constructor injection all dependencies are available at initialization time
  - With constructor injection less annotations are necessary
  - If field injection is used (without declaring setters), then the bean can only be initialized using the framework, i.e. the bean could not be created explicitly in a config class
  - With constructor injection the fields can be final!
  - With field injection the fields can be final as well, but must be initialized e.g. with null.
  - With constructor injection cyclic dependencies are not possible.

Qualifiers

- **@Primary**
  - Indicates that a particular bean should be given preference when multiple beans are candidates to be autowired to a single-valued dependency
- **@Qualifier**
  - Can be specified on individual constructor arguments or methods parameters or on fields; argument is the desired bean
  - Beans can be marked with a Qualifier, as a fall back the bean name can be used.

```
@Component
@Qualifier("Version1.1")
class MovieServiceImplV11 implements MovieService { … }
```

```
@Autowired
@Qualifier("Version1.1")
MovieService movieService;
```

Qualifiers

- **@Qualifier**
  - @Qualifier can also be used to define custom qualifiers (meta annotations)

```
@Target({ElementType.FIELD, ElementType.PARAMETER, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Qualifier
public @interface Version {
   String value();
}
```

```
@Component
@Version("1.1")
class MovieServiceImplV11 implements MovieService { … }
```

```
@Autowired
@Version("1.1")
MovieService movieService;
```

Profiles

- **@Profile**
  - Can be used to define profile-specific beans, e.g.
    - `@Profile("test")`
    - `@Profile("dev")`
  - Active profile can be defined in application.properties
    - `spring.profiles.active=dev`
  - Specific property-files can be defined to override profile-specific properties in application.properties
    - `application-test.properties`
    - `application-dev.properties`

Annotation-based Container Configuration

- **@Value**
  - Can be used to inject values from property files
  - Used on fields and method and constructor parameters
  - Argument supports ${} placeholder and default values
  - Property can be specified in file application.properties

```
@Configuration
public class DataSourceConfig {
  @Bean
  public DataSource getDataSource(
          @Value("${db.driverName:org.h2.Driver}") String driver) {
    DataSourceBuilder dataSourceBuilder = DataSourceBuilder.create();
    dataSourceBuilder.driverClassName(driver);
    return dataSourceBuilder.build();
  }
}
```

```
# application.properties file
db.driverName=com.oracle.Drive
```

Annotation-based Container Configuration

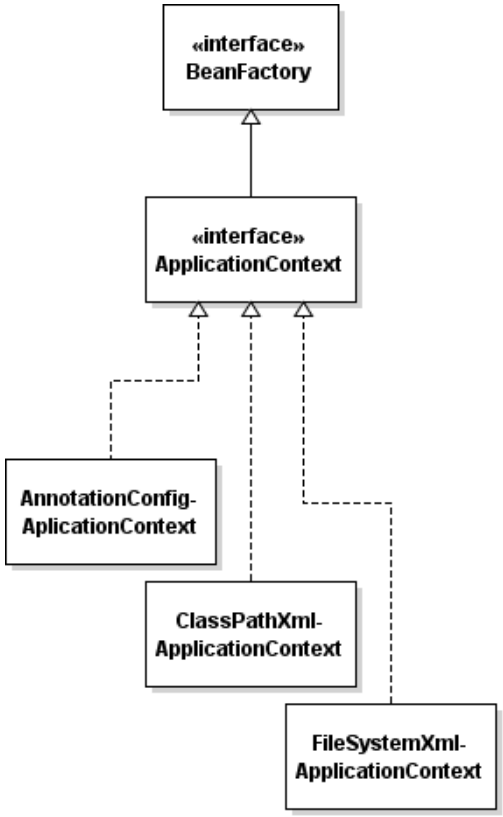
- **BeanFactory is the root factory to access beans**

```
public interface BeanFactory {
   Object getBean(String name) throws BeansException;
   <T> T getBean(Class<T> requiredType) throws BeansException;
   <T> T getBean(String name, Class<T> requiredType)
                                              throws BeansException;
   boolean containsBean(String name);
   boolean isSingleton(String name)
                               throws NoSuchBeanDefinitionException;
   boolean isPrototype(String name)
                               throws NoSuchBeanDefinitionException;
}
```

  - Maps names to beans
  - Can be accessed with `@Autowired BeanFactory ctx;` but typically it does not appear in program code!

Annotation-based Container Configuration

- **`BeanFactory`**
  - Beans are created on demand, i.e. when method getBean is called
- **`ApplicationContext`**
  - Instantiates singleton beans when the container is started
  - **`AnnotationConfigApplicationContext`**
    - Registers annotated components as beans
  - **`ClassPathXmlAplicationContext`**
    - Beans are defined in an XML file (on the classpath)
  - **`FileSystemXmlApplicationContext`**
    - XML file can bei stored anywhere



Spring Boot

- **@SpringBootApplication**
  - Is a meta annotation which contains

```
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan
public @interface SpringBootApplication {
```

    - `SpringBootConfiguration`
      - is a @Configuration class indicating that a Spring boot app is provided
    - `EnableAutoConfiguration`
      - enables auto-configuration
    - `ComponentScan`
      - defines the location where configured classes are searched
      - default is below the package which contains this annotated class

Spring Boot

- **Dependencies**

```
dependencies {
  implementation("org.springframework.boot:spring-boot-starter")
```

  - **Spring Boot Starter**

Compile Dependencies (6)

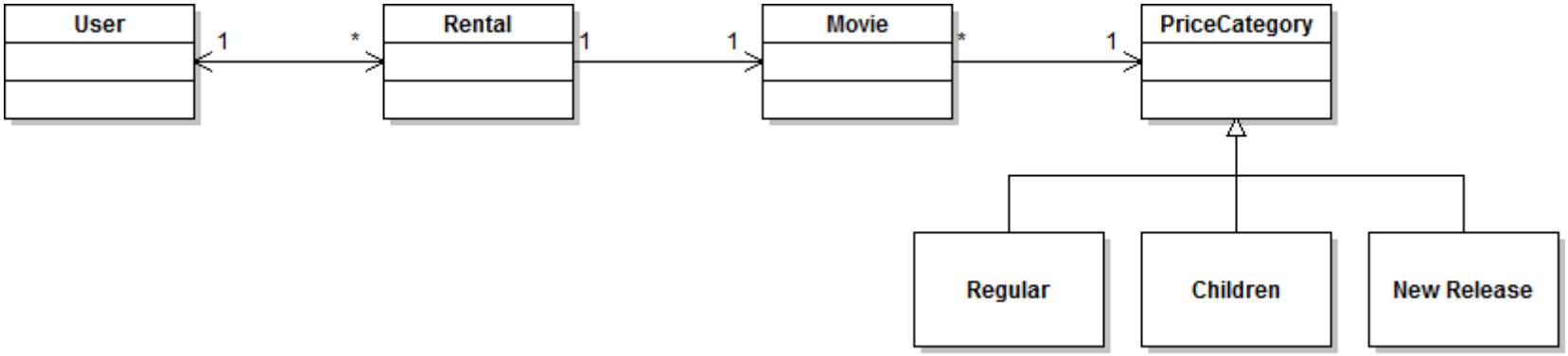
Category/License	Group / Artifact	Version	Updates
EPL 2.0	jakarta.annotation » jakarta.annotation-api	1.3.5	2.0.0
Core Utils, Apache 2.0	org.springframework » spring-core	5.3.9	✓
Apache 2.0	org.springframework.boot » spring-boot	2.5.4	✓
Apache 2.0	org.springframework.boot » spring-boot-autoconfigure	2.5.4	✓
Apache 2.0	org.springframework.boot » spring-boot-starter-logging	2.5.4	✓
YAML, Apache 2.0	org.yaml » snakeyaml	1.28	1.29

Enterprise Application Frameworks

- **Overview**
- **Spring Framework**
- **Movie Rental Application**

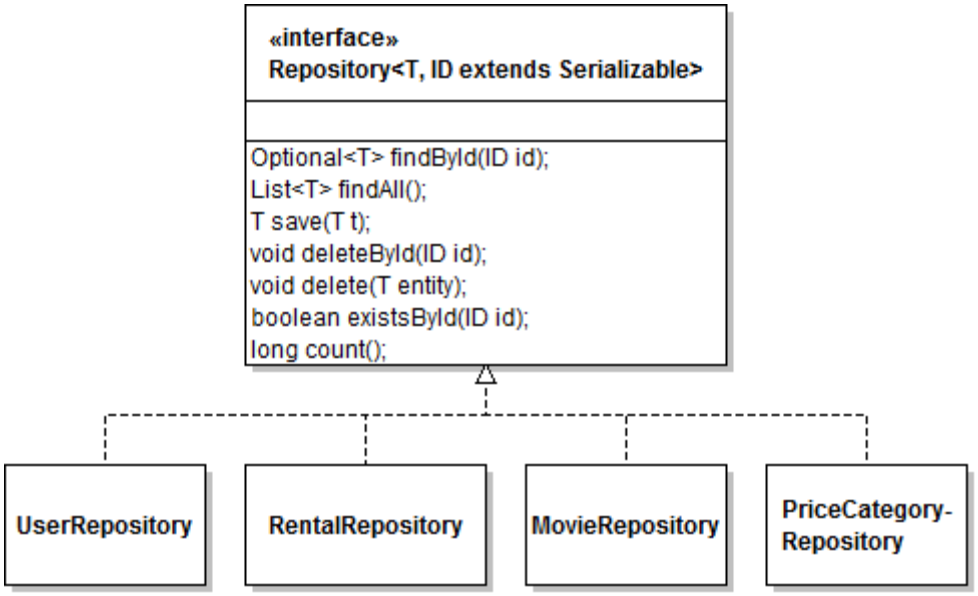
Movie Rental Application

- **Model Classes**



Movie Rental Application

- **Repository Interfaces**



Movie Rental Application

- **Repository Interface**

```
public interface Repository<T, ID extends Serializable> {
   Optional<T> findById(ID id);
   List<T>     findAll();

   T           save(T t);

   void        deleteById(ID id);
   void        delete(T entity);

   boolean     existsById(ID id);
   long        count();
}
```

Movie Rental Application

- **Component Wiring**