UNIPÊ
Pós-graduação

Desenvolvimento Web Frameworks

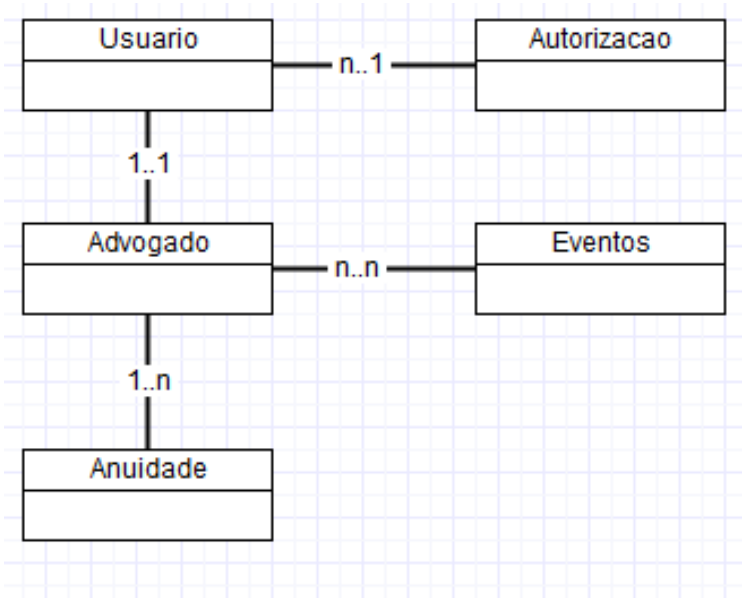
MSc. Moacir Lopes de Mendonça Junior

Agenda

- Hibernate
- JPA
- Spring Core
- Spring MVC
- Spring Security

Projeto Exemplo

- O Projeto exemplo está disponível em https://github.com/moacirlmjr/devwebunipe.git
- Modelo de Dados



Hibernate

- Baixa produtividade no desenvolvimento de Queries
- Apesar do padrão ANSI, SGBDs apresentam diferenças
- Paradigma de POO difere do esquema entidade relacional
- Baixa produtividade ao transformar objetos em registros e registros em objetos

Hibernate

- Ferramentas ORM se tornaram populares
- Inspirou o JPA
- Abstração do SQL
- Geração do SQL pensando no dialeto do BD

Benefícios do Hibernate

- Suporte a mapeamento de classes java para tabelas e vice-versa
- Suporte a gerenciamento de transações, garantido que não há inconsistência nos dados presentes
- Através do mapeamento das classes o Hibernate provê uma camada de abstração entre a aplicação e a base de dados

Benefícios do Hibernate

- Ajuda no mapeamento de joins, collections
  - Podemos visualizar com facilidade como nossas classes estão representando as tabelas
- Possui poderosa linguagem de SQL, o HQL orientado a objetos
- Possui o Hibernate Validator, implementação da especificação do Beans Validator
- Tem integração com Spring

Hibernate e JPA

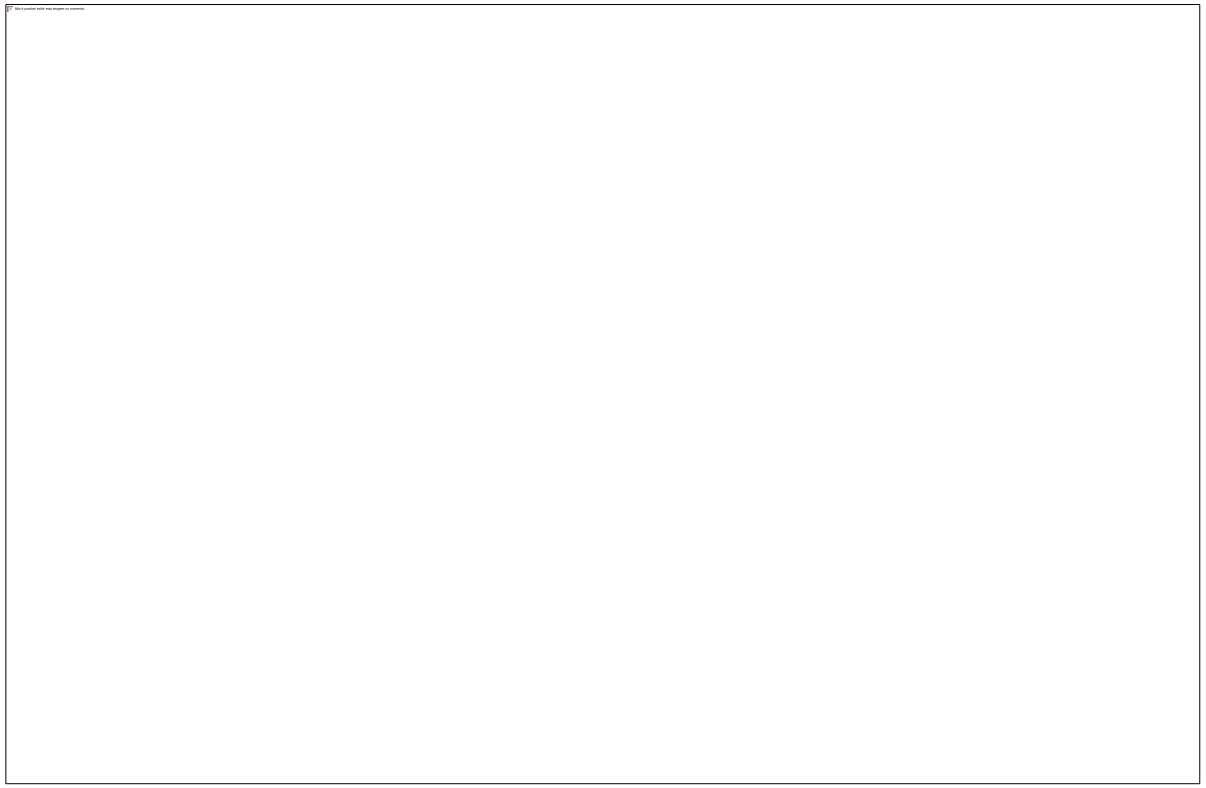
- Implementa toda a especificação do Java Persistence API

- Pode usar o annotations do JPA para modelar as classes

Hibernate

- Ciclo de Vida
- Configuração
- Mapeamento de Objetos
- Annotations
- Validator
- Versionamento
- Pooling
- Envers
- Cache

Ciclo de Vida



- Transient – objeto novo
- Persistent – tem representação no Banco de dados
- Detached – objeto existente, mas com sessão fechada
- Removed – objeto removido

Hibernate Session

- A sessão é usada para obter conexão com a base de dados.
- É designado instanciado toda vez que há interação com a base de dados.
- Objetos de sessão não devem ser mantidos abertos por muito tempo
- A sua função principal é oferecer acesso as operações de criação, leitura e remoção para as instâncias de classes mapeadas.

Configurando o hibernate.cfg.xml

- Propriedade de Conexão
  - hibernate.dialect - Faz Hibernate gerar SQL apropriado
  - hibernate.connection.driver_class - Classe JDBC
  - hibernate.connection.url - Jdbc URL para a base de dados
  - hibernate.connection.username.
  - hibernate.connection.password.

Hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</property>
        <property name="hibernate.connection.driver_class">org.postgresql.Driver</property>
        <property name="hibernate.connection.url">jdbc:postgresql://localhost:5432/advogados</property>
        <property name="hibernate.connection.username">postgres</property>
        <property name="hibernate.connection.password">mv13wavaty</property>

        <property name="hibernate.current_session_context_class">org.hibernate.context.internal.ThreadLocalSessionContext</property>
        <property name="hbm2ddl.auto">create</property>
    </session-factory>
</hibernate-configuration>
```

Dialetos

Database	Dialect Property
DB2	org.hibernate.dialect.DB2Dialect
HSQLDB	org.hibernate.dialect.HSQLDialect
HypersonicSQL	org.hibernate.dialect.HSQLDialect
Informix	org.hibernate.dialect.InformixDialect
Ingres	org.hibernate.dialect.IngresDialect
Interbase	org.hibernate.dialect.InterbaseDialect
Microsoft SQL Server 2000	org.hibernate.dialect.SQLServerDialect
Microsoft SQL Server 2005	org.hibernate.dialect.SQLServer2005Dialect
Microsoft SQL Server 2008	org.hibernate.dialect.SQLServer2008Dialect
MySQL	org.hibernate.dialect.MySQLDialect
Oracle (any version)	org.hibernate.dialect.OracleDialect
Oracle 11g	org.hibernate.dialect.Oracle10gDialect
Oracle 10g	org.hibernate.dialect.Oracle10gDialect
Oracle 9i	org.hibernate.dialect.Oracle9iDialect
PostgreSQL	org.hibernate.dialect.PostgreSQLDialect
Progress	org.hibernate.dialect.ProgressDialect
SAP DB	org.hibernate.dialect.SAPDBDialect
Sybase	org.hibernate.dialect.SybaseDialect
Sybase Anywhere	org.hibernate.dialect.SybaseAnywhereDialect

Obtendo SessionFactory

```java
public class HibernateUtil implements Serializable {

    private static SessionFactory sessionFactory;

    public static SessionFactory getSessionFactory() {
        if (sessionFactory == null) {
            try {
                sessionFactory = new Configuration().configure().buildSessionFactory();

            } catch (Throwable ex) {
                System.err.println("Initial SessionFactory creation failed." + ex);
                throw new ExceptionInInitializerError(ex);
            }
            return sessionFactory;
        } else {
            return sessionFactory;
        }
    }

    public static void main(String[] args) {
        HibernateUtil.getSessionFactory();
    }

}
```

Mapeamento

- Um dos principais objetivos dos frameworks ORM é estabelecer o mapeamento entre o modelo orientado a objetos e o modelo relacional.
- Tipos Primitivos

Mapping type	Java type	ANSI SQL Type
integer	int or java.lang.Integer	INTEGER
long	long or java.lang.Long	BIGINT
short	short or java.lang.Short	SMALLINT
float	float or java.lang.Float	FLOAT
double	double or java.lang.Double	DOUBLE
big_decimal	java.math.BigDecimal	NUMERIC
character	java.lang.String	CHAR(1)
string	java.lang.String	VARCHAR
byte	byte or java.lang.Byte	TINYINT
boolean	boolean or java.lang.Boolean	BIT
yes/no	boolean or java.lang.Boolean	CHAR(1) ('Y' or 'N')
true/false	boolean or java.lang.Boolean	CHAR(1) ('T' or 'F')

Mapeamento

- Tipos Data

Mapping type	Java type	ANSI SQL Type
date	java.util.Date or java.sql.Date	DATE
time	java.util.Date or java.sql.Time	TIME
timestamp	java.util.Date or java.sql.Timestamp	TIMESTAMP
calendar	java.util.Calendar	TIMESTAMP

- Tags
  - Class
  - Id
  - Generator
  - Property
  - Many-to-one
  - Set
  - Key
  - Column
  - One-to-many
  - Many-to-many

Mapeando o Objeto Usuário

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD//EN"
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="br.com.unipe.gerenciamentoAdvogados.model.vo.Usuario" table="usuario">
        <id name="id" type="long">
            <generator class="native"></generator>
        </id>
        <property name="createdOn" type="timestamp">
        </property>
        <property name="nome" type="string"></property>
        <property name="email" type="string"></property>
        <property name="telefone" type="string"></property>
        <property name="username" type="string"></property>
        <property name="password" type="string"></property>
    </class>
</hibernate-mapping>
```

Exercício

- Desenvolver o Mapeamento para os Objetos:
  - Autorização
  - Anuidade
  - Advogado

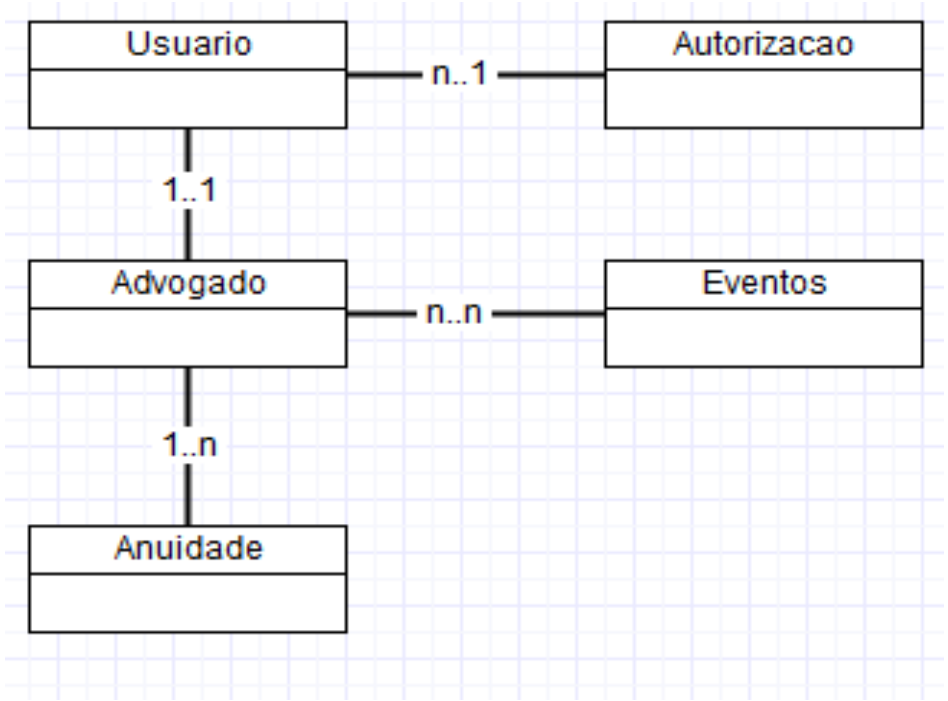
- moacir.lopes.jr@gmail.com

Mapeamento entre associações

- Há quatro maneiras que a cardinalidade dos relacionamentos entre objetos pode ser expressado
  - Many-to-one
  - One-to-one
  - One-to-many
  - Many-to-many

Modelo de Dados

- Modelo de Dados



Many-to-one

- Nesta associação o objeto pode estar associado com múltiplos objetos.

```
<many-to-one name="autorizacao" column="autorizacao_id"
        not-null="true" cascade="all"
    class="br.com.unipe.gerenciamentoAdvogados.model.vo.Autorizacao">
</many-to-one>
```

One-to-one

- Está é similar a many-to-one, a diferença está na adição da propriedade unique=”true”.

```
<many-to-one name="autorizacao" column="autorizacao_id"
        not-null="true" cascade="all" unique="true"
class="br.com.unipe.gerenciamentoAdvogados.model.vo.Autorizacao">
</many-to-one>
```

One-to-many

- Esta associação liga o parente com um ou mais filhos.
- Unidirecional:

```
<set name="anuidades" inverse="true">
    <key>
        <column name="advogado_id" not-null="true"></column>
    </key>
    <one-to-many class="br.com.unipe.gerenciamentoAdvogados.model.vo.Anuidade" />
</set>
```

- Bidirecional:

```
<many-to-one name="advogado" column="advogado_id"
    class="br.com.unipe.gerenciamentoAdvogados.model.vo.Advogado"
    cascade="all"></many-to-one>
```

Many-to-many

- Neste mapeamento o hibernate criará uma terceira tabela com a chave dos dois objetos mapeados.

Em Advogados:

```
<set name="eventos" table="advogados_eventos" inverse="true">
    <key>
        <column name="advogado_id" not-null="true"></column>
    </key>
    <many-to-many class="br.com.unipe.gerenciamentoAdvogados.model.vo.Evento">
        <column name="evento_id" not-null="true"></column>
    </many-to-many>
</set>
```

Em Eventos

```
<set name="advogados" table="advogados_eventos">
    <key>
        <column name="evento_id" not-null="true"></column>
    </key>
    <many-to-many class="br.com.unipe.gerenciamentoAdvogados.model.vo.Evento">
        <column name="advogado_id" not-null="true"></column>
    </many-to-many>
</set>
```

Metodos do Hibernate Session

- Salvar: session.save(obj);
- Atualizar: session.update(obj);
- Delete: session.delete(obj)
- Obter os dados:
  - Criteria, Hql, Native Query, session.get(Obj.class,id)

Operações em cascata

- O hibernate/jpa permite realizar operações de persistência em cascata,

- As opções em que você pode utilizar na propriedade cascade é save, update, saveOrupdate, delete, all.

Mapeamento com Annotations

- O hibernate também permite o mapeamento das entidades através de anotações, estendidas dos JPA
  - @Entity
  - @Table
  - @Id
  - @GeneratedValue
  - @Column
  - @Enumarated
  - @Temporal
  - @Mappedsuperclass
  - @transient
  - @One-to-many
  - @Many-to-one
  - @One-to-one
  - @Many-to-many
  - @Jointable

Exercício

- Desenvolver o Dao e o Mapeamento para os Objetos:
  - Anuidade
  - Advogado
  - Eventos

- moacir.lopes.jr@gmail.com

Versionamento

- No Hibernate você pode utilizar *locking* otimista através da propriedade versão em seus objetos mapeados

- Para especificar a versão para seu objeto, simplesmente adicione a propriedade *version* no seu mapeamento

Versionamento

- Hibernate usará a o número de versão para checar se a linha foi atualizada ou não desde a última vez que o objeto foi persistido

- Caso a versão tenha sido atualizada, o framework lançará a exceção *StaleObjectStateException* e a transação fará um *rollback*.

- Adicionar propriedade “version” ou anotação @Version

Pooling

- Qualquer aplicação web que acesse bancos de dados precisa estar preparada para receber vários acessos simultâneos
- Mas o que acontece quando o número de usuários é muito grande?
- Tempo de latência é grande ao ficar abrindo e fechando conexões com o banco

Pooling

- O pool de conexões é excelente para melhorar a performance do hibernate
- Cria diversas conexões idles, para serem usadas pela aplicação
- Para configurar o pool iremos utilizar a lib C3P0 do hibernate

Propriedades do C3P0

- Adicionar no hibernate.cfg.xml
  - <property name="hibernate.c3p0.min_size">5</property>
  - <property name="hibernate.c3p0.max_size">20</property>
  - <property name="hibernate.c3p0.timeout">300</property>

Auditoria

- Auditoria pode ser considerada como um exame sistemático das atividades desenvolvidas em determinada empresa ou setor

- Com o surgimento dos sistemas computacionais, as empresas passaram a confiar seus dados à área de Tecnologia da Informação (TI)

- Como essas informações, na maioria das vezes, ficam armazenadas em bancos de dados, ele é o ponto de partida para iniciar uma auditoria

Auditoria

- Auditar a persistência requer uma análise direta de operações.
- Uma das funções da auditoria é monitorar quando e como o dado foi inserido, com o intuito de prevenir e detectar problemas no cumprimento das regras de negócio.
- Outra função tem relação com questões de segurança, objetivando garantir que usuários não autorizados não estejam acessando o banco de dados.

Auditoria

- Uma boa auditoria de persistência deve fornecer informações sobre as operações realizadas nas tabelas envolvidas

- Com estas informações o analista poderá recuperar uma versão antiga ou mesmo identificar o autor daquela alteração.

- Independentemente do porte da aplicação, é cada vez mais comum a necessidade real de monitorar as ações do usuário frente ao sistema

Envers

- O Hibernate Envers é uma biblioteca que nos facilita oferecendo um conjunto de funcionalidades sobre auditoria

- Ele possui um mecanismo embutido para manter o histórico de tabelas no Hibernate.

- O Envers trabalha de forma similar a um Controlador de Versão

Envers

- Algumas características do Hibernate Envers:
  - Auditar todos os mapeamentos definidos pela especificação do JPA ou hibernate
  - Consultas de dados históricos.

- Para o objeto mapeado o envers gera duas tabelas revinfo e [nometabela]_aud

JPA

- JPA é um framework leve, baseado em POJOS (Plain Old Java Objects) para persistir objetos Java.

- A JPA (Java Persistence API) é a solução ORM padrão para as plataformas Java SE e Java EE.

- Atualmente temos que praticamente todas as aplicações de grande porte utilizam JPA

JPA

- Sua criação e consequentes atualizações efetuadas foram e são estabelecidas seguindo o mesmo procedimento de evolução das tecnologias da plataforma Java, através de especificações

- Uma especificação é elaborada por meio de uma JSR (Java Specification Request), que consiste em uma proposta formal e estabelece quais e como os recursos serão oferecidos, além de determinar o comportamento esperado destes nas implementações.

JPA

- JPA 2.1 foi especificada pela JSR 338 em 2013
- Sua implementação de referência é o EclipseLink, disponibilizado no servidor GlassFish.
- Por ser considerada uma das especificações mais maduras do Java EE 6, seus recursos já podem ser utilizados nas aplicações Java, desde que a implementação JPA utilizada já suporte essa nova especificação.

JPA

- Vamos usar a implementação do Hibernate do JPA, que já implementa as especificações do JPA 2.1

EntityManager

- Os Entity Managers são configurados para serem capazes de persistir ou gerenciar tipos específicos de objetos

- É implementado por um Provedor de Persistência (persistence provider) como Hibernate, TopLink, JDO, entre outros.

Persistence.xml

- A configuração que descreve a unidade de persistência é definida em um arquivo XML chamado persistence.xml.
- Cada unidade de persistência é nomeado
- Um simples persistence.xml pode conter uma ou mais configurações de unidade de persistência nomeada

Persistence.xml

- O arquivo persistence.xml é armazenado no diretório META-INF.

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
    version="1.0">
    <persistence-unit name="PersistenceUnit" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
        <properties>
            <property name="connection.driver_class" value="org.postgresql.Driver" />
            <property name="hibernate.connection.url" value="jdbc:postgresql://localhost:5432/advogados" />
            <property name="hibernate.connection.username" value="postgres"></property>
            <property name="hibernate.connection.password" value="mv13wavaty"></property>
            <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect" />
<!--            <property name="hibernate.hbm2ddl.auto" value="update" /> -->
            <property name="hibernate.show_sql" value="true" />
            <property name="hibernate.format_sql" value="true" />
        </properties>
    </persistence-unit>
</persistence>
```

Obtendo o EntityManagerFactory

- Um Entity Manager é sempre obtido através de um EntityManagerFactory que determina os parâmetros de configuração que vão dizer como será o funcionamento do Entity Manager.

- O método estático createEntityManagerFactory() na classe Persistence retorna o EntityManagerFactory para um nome de unidade de persistência específico.

Obtendo o EntityManagerFactory

```
public class EntityManagerUtil implements Serializable {
    private static EntityManagerFactory factory;

    public static EntityManagerFactory getEntityManagerFactory() {
        if (factory == null) {
            try {
                factory = Persistence.createEntityManagerFactory("PersistenceUnit");

            } catch (Throwable ex) {
                System.err.println("Initial SessionFactory creation failed." + ex);
                throw new ExceptionInInitializerError(ex);
            }
            return factory;
        } else {
            return factory;
        }
    }
```

Metodos do EntityManager

- Persistir: em.persist(obj);
- Atualizar: em.merge(obj);
- Remover: em.remove(obj);
- Obtendo dados:
  - Criteria, JQL, Native Query, em.find(Obj.**class**, id);

DAO with Generics

- Um “DAO genérico” é uma maneira conveniente de ter uma única classe que consiga fazer operações de persistência em tipos de entidade diferentes.

```
public class GenericDAOImpl<T, I extends Serializable>  {
```

Spring Framework

- Com mais de 10 anos de existência, o Spring ainda é um dos mais importantes e mais usados frameworks do ecossistema Java.

- O Spring veio com o objetivo de facilitar o desenvolvimento e deployment de aplicações Java enterprise

- A introdução de conceitos e facilidades inexistentes até então na plataforma Java EE o transformou na solução preferida a ser adotada em novos projetos.

Spring Framework

- Com o passar do tempo, muitas das suas funcionalidades passaram a fazer parte da especificação Java EE

- Ao mesmo tempo, novos recursos do Java também ajudaram em sua evolução, como por exemplo, o uso de Annotations ao invés dos seus longos arquivos XML de configuração.

- Desde a criação do Spring, sempre existiu muita discussão comparando-o com a Java EE

Spring Framework

- O que parece ser mais razoável é que de início o Spring realmente atuava como um substituto às tecnologias “padrões” da Java EE

- É verdade, porém, que muito de seu sucesso é relacionado com o surgimento do Hibernate, que atuou e ainda atua como seu grande “parceiro” em muitos projetos.

Spring Framework

- No entanto, com o passar do tempo e com a evolução da especificação Java EE, esses frameworks também passaram a ser vistos como ferramentas que podem ser usadas em conjunto com a especificação, e não somente como substitutos.

- Muitas das funcionalidades do Spring são baseadas em tecnologias padrões, como JNDI, JMS e JTA. O próprio Hibernate passou a ser uma implementação da especificação do JPA.

Spring Framework

- Suas versões sempre trouxeram novos projetos visando novas funcionalidades e facilidades, como o spring-integration, spring-data, spring-mvc, spring-social, entre outros

- O projeto Spring cresceu tanto com o passar do tempo que atualmente ele possui algumas dezenas de subprojetos, o que torna o um verdadeiro “canivete suíço”.

Configuração Inicial

- Web.xml

```
<servlet>
	<servlet-name>Spring MVC</servlet-name>
	<servlet-class>
		org.springframework.web.servlet.DispatcherServlet
	</servlet-class>
	<init-param>
		<param-name>contextConfigLocation</param-name>
		<param-value>
			/WEB-INF/application-context.xml
		</param-value>
	</init-param>
	<load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
	<servlet-name>Spring MVC Dispatcher Servlet</servlet-name>
	<url-pattern>/</url-pattern>
</servlet-mapping>
```

Application Context

- No application-context definimos todas as configurações do Spring Framework

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-4.0.xsd">

    <context:component-scan base-package="br.com.unipe.gerenciamentoAdvogados" />
    <mvc:annotation-driven />

</beans>
```

Injeção de dependência

- O padrão de injeção de dependência trabalha baseado em abstrações, sejam elas classes abstratas ou interfaces

- Programe para uma interface e nunca para uma implementação

- Diminuição o acoplamento entre as classes do nosso modelo

Injeção de dependência

- Podemos trabalhar com a injeção de dependência de três formas:
  - injeção por construtor (constructor injection);
  - injeção por propriedade ou getters e setters no caso do Java (setter injection);
  - injeção por interface (interface injection)

Spring DI

- Os mecanismos do Spring para implementar injeção de dependência, são baseadas na especificação Java para injeção de dependências:
  - @Component
  - @Repository
  - @Service
  - @Autowired

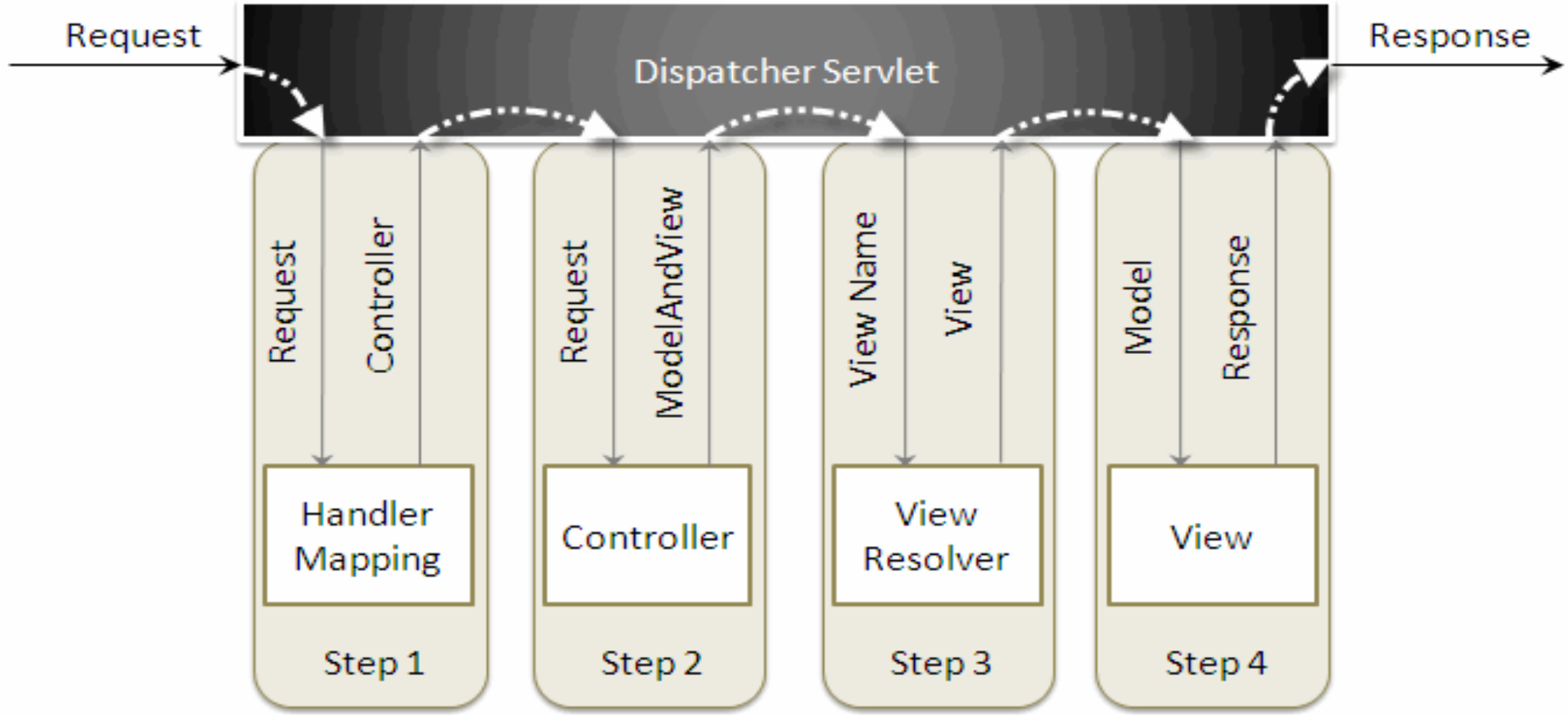
Spring MVC

- Spring é um framework que inicialmente não foi criado para o desenvolvimento web.

- Na essência o Spring é um container leve que visa fornecer serviços para sua aplicação como por exemplo o gerenciamento de objetos ou transação.

- O Spring MVC é um framework moderno que usa os recursos atuais da linguagem além de usar todo poder do container Spring.

Ciclo de Vida



Application-context

- É necessário adicionar no application-context.xml configuramos onde ficarão as views a serem criadas e a extensão das mesmas.

```
<mvc:annotation-driven />

<bean
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/views/" />
    <property name="suffix" value=".jsp" />
</bean>
```

Criando o Controller de Autorizacao

```
@Controller
public class AutorizacaoBean {

    @Autowired
    private AutorizacaoDAO autorizacaoDAO;

}
```

Cadastro de Autorizacao

```
@RequestMapping("prepararCadastroAutorizacao")
public String prepararCadastro(Model model) {
model.addAttribute("autorizacao", new Autorizacao());
    return "cadastroAutorizacao";
}

@RequestMapping("/addAutorizacao")
private String cadastro(Autorizacao a) {
    autorizacaoDAO.create(a);
    return "redirect:/prepararListarAutorizacao";
}
```

View de Cadastro de Autorizacao

- Após a criação do jsp colocar a taglib abaixo:

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
```

View de Cadastro de Autorizacao

```
<h2>Cadastro de Autorização!</h2>
<form:form method="POST" action="addAutorizacao" modelAttribute="autorizacao">
    Nome:
    <form:input path="nome" type="text" id="nome" />
    <br />

    <input type="submit" value="Enviar" />
</form:form>
```

Listar Autorização

```
@RequestMapping("/prepararListarAutorizacao")
private String list(Model model) {
    List<Autorizacao> list = autorizacaoDAO.listAll();
    model.addAttribute("autorizacoes", list);
    return "listAutorizacao";
}
```

View para Listar Autorização

```
<a href="index.jsp">Inicio</a><br />
<a href="prepararCadastroAutorizacao">Criar nova autorizacao</a><br /><br />
<table>
    <tr>
        <th>Id</th>
        <th>Nome</th>
        <th>Remover</th>
        <th>Modificar</th>
    </tr>
    <c:forEach items="${autorizacoes}" var="autorizacao">
        <tr>
            <td>${autorizacao.id}</td>
            <td>${autorizacao.nome}</td>
            <td><a href="removerAutorizacao?id=${autorizacao.id}">Remover</a></td>
            <td><a href="prepararAtualizarAutorizacao?id=${autorizacao.id}">Modificar</a></td>
        </tr>
    </c:forEach>
</table>
```

Atualizar Autorização

```
@RequestMapping("/prepararAtualizarAutorizacao")
private String modificar(Long id, Model model) {
    model.addAttribute("autorizacao", autorizacaoDAO.findById(id));
    return "alterarAutorizacao";
}

@RequestMapping("/updateAutorizacao")
private String update(Autorizacao a) {
    autorizacaoDAO.update(a);
    return "redirect:/prepararListarAutorizacao";
}
```

View para Atualizar Autorizacao

```
<h2>Atualização de Autorização!</h2>
<form:form method="POST" action="updateAutorizacao" modelAttribute="autorizacao">
    <form:hidden path="id" />
    Nome:
    <form:input path="nome" type="text" id="nome" /> <br />
    <input type="submit" value="Alterar" />
</form>
```

Deletar Autorização

```
@RequestMapping("/removerAutorizacao")
private String remover(Autorizacao a) {
    autorizacaoDAO.delete(autorizacaoDAO.findById(a.getId()));
    return "redirect:/prepararListarAutorizacao";
}
```

Obrigado