03

Salvando a compra

Transcrição

Voltando ao nosso checkout, precisamos salvar os itens junto com o usuário. Ou seja, precisamos indicar o carrinho no CheckoutBean:

```
@Inject
private CarrinhoCompras carrinho;

@Transactional
public void finalizar() {
    carrinho.finalizar(usuario);
    usuarioDao.salvar(usuario);
}
```

É interessante passar o usuário em `carrinho.finalizar(usuario)` porque o próprio Carrinho finaliza a compra passando o usuário e guardando todos os itens no banco de dados. Então agora o `usuarioDao.salvar(usuario)` vai para dentro do método `finalizar()` em `CarrinhoCompras.java` :

```
public void finalizar(Usuario usuario) {
    usuarioDao.salvar(usuario);
}
```

E o `private UsuarioDao usuarioDao;` será removido do Bean e injetado no CarrinhoCompras:

```
private static final long serialVersionUID = 1L;

@Inject
private UsuarioDao usuarioDao;

private Set<CarrinhoItem> itens =
        new HashSet<>();
```

Nós também iremos salvar os itens do carrinho. Chamaremos essa entidade de `Compra` informando quem é o usuário e quais os itens dela, além de pedir para salvar:

```
public void finalizar(Usuario usuario) {
    Compra compra = new Compra();
    compra.setUsuario(usuario);
    compra.setItens(itens);
    usuarioDao.salvar(usuario);
    compraDao.salvar(compra);
}
```

Como de costume, precisamos criar essa Classe "Compra", criando o arquivo `Compra.java` :

```
package br.com.casadocodigo.loja.models;

import javax.persistence.Entity;

@Entity
public class Compra {

}
```

Cada compra deve ter seu próprio Id:

```
package br.com.casadocodigo.loja.models;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Compra {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;
}
```

E vamos colocar os dados da compra:

```
import java.util.List;

//...

public class Compra {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;

    private Usuario usuario;

    private List<CarrinhoItem> itens;
}
```

Damos um CTRL + SHIFT + O para importarmos todos. Assim temos a relação da nossa compra, com Id próprio, Usuário e Itens do carrinho de compras. Falta gerarmos os getters e os setters:

```
public Integer getId() {
    return id;
}

public void setId(Integer Id) {
    this.id = id;
```

```
}

public Usuario getUsuario() {
    return usuario;
}

public void setUsuario(Usuario usuario) {
    this.usuario = usuario;
}

public List<CarrinhoItem> getItens() {
    return itens;
}

public void setItens(List<CarrinhoItem itens) {
    this.itens = itens;
}
```

Na Classe do Carrinho de Compras precisamos mudar de `setItens(item)` dentro do método `finalizar()` para pegar direto pelo get:

```
public void finalizar(Usuario usuario) {
    //...
    compra.setItens(getItens());
    //...
}
```

Como precisamos salvar a compra toda, criaremos uma variável local `CompraDao` :

```
//...

private CompraDao compraDao

public void add(CarrinhoItem item) {
    itens.add(item)
}

//...
```

E criamos a Classe "CompraDao" no pacote Daos:

```
package br.com.casadocodigo.loja.daos;

public class CompraDao {

}
```

Criamos o `EntityManager` e um método para salvar a compra, o processo não é diferente do que fizemos com o Usuário:

```
package br.com.casadocodigo.loja.daos;
```

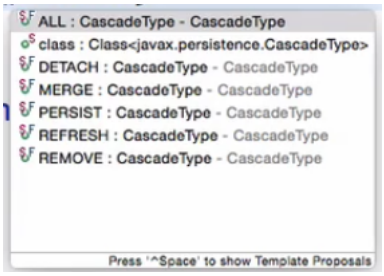
```
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import br.com.casadocodigo.loja.models.Compra

public class CompraDao {

    @PersistenceContext
    private EntityManager manager;

    public void salvar(Compra compra) {
        manager.persist(compra);
    }
}
```

Podemos pedir para que o Usuário seja salvo junto com a Compra. Em `Compra.java` temos a relação do usuário com a compra. Primeiramente temos que dizer qual o tipo de relação:

Uma conta possui apenas um usuário, mas um usuário pode possuir mais de uma conta. Logo Many-to-One.

```
@ManyToOne(cascade=CascadeType.PERSIST)
private Usuario usuario;
```

Desse modo estamos informando par o JPA que toda vez que quisermos salvar uma compra, o usuário seja salvo juntamente, isto é o que faz o *Cascade Persist*. Perceba que temos vários outros cascades:



A nossa lista de itens vai salvar cada `<CarrinhoItem>` e relacioná-los no banco de dados. Porém o `<CarrinhoItem>` não é uma entidade e só representa o item que está no carrinho antes da compra, não é um item vendido para o usuário. Então vamos mudar seu nome para ficar mais semântico:

```
private List<ItemVendido> itens;
```

Poderemos salvar uma lista desses itens em que cada um se transforma em um item vendido. Isto é importante pois, caso o valor do item mude, compras passadas não serão afetadas.

Mas façamos algo diferente agora. Deixaremos os itens como String!

```
private String itens;

//...

public String getItens() {
    return itens;
```

```
}

public void setItens(String itens) {
    this.itens = itens;
}
```

Que estranho! Os itens em String vão salvar texto no lugar de uma lista... Mas vejamos o que é texto:

- XML
- JSON

Estes formatos são os mais comuns para transportar dados de um sistema para outro! Então salvar os itens como texto vai nos permitir essa flexibilidade e evoluir nosso sistema.

> ATENÇÃO: não estamos dizendo que salvar em formatos JSON ou XML é melhor. Estamos fazendo isso com propósito experimental e queremos nos aprofundar em novas tecnologias do Java EE.

Tendo isso em vista iremos usar a [JSR-353 (http://www.oracle.com/technetwork/articles/java/json-1973242.html)](http://www.oracle.com/technetwork/articles/java/json-1973242.html) (Java API for JSON Processing). Essa API vai nos permitir transformar o CarrinhoItem para JSON:

```
public void finalizar(Usuario usuario) {
    Compra compra = new Compra();
    compra.setUsuario(usuario);
    compra.setItens(this.toJson());
    compraDao.salvar(compra);
}
```

Aproveitando, não precisamos mais do UsuarioDao, já que estamos salvando em cascata. Também excluiremos o `private UsuarioDao usuarioDao .`

Criamos o método `toJson()` , fazendo com que ele retorne, por enquanto, um JSON vazio:

```
public String toJson() {
    return "{}";
}
```

Antes de testarmos, só falta um detalhe, no `CompraDao.java` :

```
public class CompraDao implements Serializable {

    private static final long serialVersionUID = 1L;

    @PersistenceContex
    private EntityManager manager;

    public void salvar(Compra compra) {
        manager.persist(compra);
    }
}
```

O nome do usuário também não estava sendo salvo foi faltava um `type` no xhtml:

```
<input class="form-control" type="text" .../>
```

Apesar de no xhtml ser facultativo, o JSF pede o *type*.

Subindo o servidor e testando, adicionamos um item no carrinho, finalizamos a compra, preenchemos os dados do usuário e clicamos em "Finalizar compra". Vejamos o resultado no banco de dados:

```
mysql> select * from usuario;
+----+--------------------------+-------------+--------+
| id | email                    | nome        | senha  |
+----+--------------------------+-------------+--------+
|  1 | paulo.alvez@caelum.com.br | NULL        | 123456 |
|  2 | paulo.alvez@caelum.com.br | Paulo Alves | 123456 |
+----+--------------------------+-------------+--------+
2 rows in set (0,00 sec)
```

Vejamos todas as tabelas que o banco de dados possui:

```
mysql> show tables;
+-------------------------------+
| Tables_in_casadocodigo_javaee |
+-------------------------------+
| Autor                         |
| Compra                        |
| Livro                         |
| Livro_Autor                   |
| Usuário                       |
+-------------------------------+
5 rows in set (0,00 sec)
```

Agora temos a tabela "Compra"!

```
mysql> select * from Compra;
+----+-------+------------+
| id | itens | usuario_id |
+----+-------+------------+
|  1 | {}    |          2 |
+----+-------+------------+
1 row in set (0,00 sec)
```

Na tabela observamos 1 compra, nenhum item e um usuário de id=2, que é exatamente o segundo usuário da tabela de usuários.

O próximo passo é fazer com que a compra seja salva realmente, afinal nós inserimos um item no carrinho.