02

Relatório de horas

Nesse último capítulo, queremos desenvolver um novo relatório para o projeto que listará qual foi o número de horas normais e extras que foram trabalhadas pelo usuário logado por dia.

Para essa funcionalidade, precisamos inicialmente de um novo método no dao que devolverá a lista de horas lançadas de um determinado usuário:

```
public List<HoraLancada> horasDoUsuario(Usuario usuario){
    String jpql = "select h from HoraLancada h where h.usuario = :usuario order by h.data";
    TypedQuery<HoraLancada> query = manager.createQuery(jpql, HoraLancada.class);
    query.setParameter("usuario", usuario);
    return query.getResultList();
}
```

E agora, vamos utilizar esse novo método dentro do `HoraLancadaController`, mas para isso precisaremos do usuário logado da aplicação:

```
@Controller
public class HoraLancadaController{
    private UsuarioLogado usuarioLogado;

    // outros atributos

    @Inject
    public HoraLancadaController(HoraLancadaDao horaLancadaDao,
            Validator validator, Result result, UsuarioLogado usuarioLogado) {
        this.horaLancadaDao = horaLancadaDao;
        this.validator = validator;
        this.result = result;
        this.usuarioLogado = usuarioLogado;
    }

    // outros métodos

    public void relatorioHoras(){
        List<HoraLancada> horas = horaLancadaDao.horasDoUsuario(
            usuarioLogado.getUsuario());
    }
}
```

Mas temos um problema, o usuário pode lançar diversas horas para um mesmo dia, então precisamos agregar os resultados pelo dia, calculando qual é o total de horas normais e extras por dia. Esse é um trabalho complicado e, por isso, será isolado dentro de uma nova classe no projeto chamada `RelatorioDeHoras`.

Como o `RelatorioDeHoras` precisa da lista de `HoraLancada`, passaremos essa informação no construtor da classe:

```
public class RelatorioDeHoras{
    public RelatorioDeHoras(List<HoraLancada> horas){
```

```
        // processa a lista de horas
    }
}
```

No construtor dessa classe, faremos o agrupamento das informações do banco de dados e guardaremos o resultado desse processamento dentro de objetos do tipo `HorasPorDia` :

```
public class RelatorioDeHoras{
    private List<HorasPorDia> horasPorDia = new ArrayList<>();

    public RelatorioDeHoras(List<HoraLancada> horas){
        // processa a lista de horas
    }
}
```

A classe `HorasPorDia` vai simplesmente armazenar o número de horas normais e extras de um determinado dia:

```
public class HorasPorDia {

    private double horasNormais;

    private double horasExtras;

    private Calendar data;

    public HorasPorDia(double horasNormais, double horasExtras, Calendar data) {
        this.horasNormais = horasNormais;
        this.horasExtras = horasExtras;
        this.data = data;
    }
    // getters
}
```

Para finalizar, implementaremos a lógica do `RelatorioDeHoras` começando pelo código do teste para garantir que quando o teste passar, teremos o código pronto.

```
public class RelatorioDeHorasTest{

    @Test
    public void calculaRelatorioQuandoDatasSaoIguais(){
        Calendar data = new GregorianCalendar(2014, 11, 11);
        HoraLancada hora1 = novaHoraLancada("10:00", "18:00", data);
        HoraLancada hora2 = novaHoraLancada("18:00", "20:00", data);
        RelatorioDeHoras relatorio = new RelatorioDeHoras(Arrays.asList(hora1, hora2));

        HorasPorDia horasPorDia = relatorio.getHorasPorDia().get(0);
        Assert.assertEquals(1, relatorio.getHorasPorDia().size());
        Assert.assertEquals(8.0, horasPorDia.getHorasNormais(), 0.001);
        Assert.assertEquals(2.0, horasPorDia.getHorasExtras(), 0.001);
        Assert.assertEquals(data, horasPorDia.getData());
    }

    @Test
```

```
    public void calculaRelatorioQuandoDatasSaoDiferentes(){
        Calendar data1 = new GregorianCalendar(2014, 10, 11);
        Calendar data2 = new GregorianCalendar(2014, 10, 12);

        HoraLancada hora1 = novaHoraLancada("10:00", "18:00", data1);
        HoraLancada hora2 = novaHoraLancada("10:00", "20:00", data2);

        RelatorioDeHoras relatorio = new RelatorioDeHoras(Arrays.asList(hora1, hora2));

        HorasPorDia horasPorDia1 = relatorio.getHorasPorDia().get(0);
        HorasPorDia horasPorDia2 = relatorio.getHorasPorDia().get(1);

        Assert.assertEquals(2, relatorio.getHorasPorDia().size());
        Assert.assertEquals(8.0, horasPorDia1.getHorasNormais(), 0.001);
        Assert.assertEquals(0.0, horasPorDia1.getHorasExtras(), 0.001);
        Assert.assertEquals(data1, horasPorDia1.getData());

        Assert.assertEquals(8.0, horasPorDia2.getHorasNormais(), 0.001);
        Assert.assertEquals(2.0, horasPorDia2.getHorasExtras(), 0.001);
        Assert.assertEquals(data2, horasPorDia2.getData());
    }

    private HoraLancada novaHoraLancada(String inicio, String fim, Calendar data){
        HoraLancada hora = new HoraLancada();
        hora.setData(data);
        hora.setHoraInicial(inicio);
        hora.setHoraFinal(fim);
        return hora;
    }
}
```

E agora que o teste está pronto, podemos implementar a lógica do relatório que será chamada pelo construtor da classe:

```
public class RelatorioDeHoras {

    private List<HorasPorDia> horasPorDia = new ArrayList<>();

    public RelatorioDeHoras(List<HoraLancada> horas) {
        calculaHorasPorDia(horas);
    }

    private void calculaHorasPorDia(List<HoraLancada> horas) {
        if (!horas.isEmpty()) {
            double horasDoDia = 0.0;
            Calendar dataAtual = horas.get(0).getData();
            for (HoraLancada hora : horas) {
                if(!hora.getData().equals(dataAtual)){
                    double horasNormais = Math.min(horasDoDia, 8);
                    double horasExtras = Math.max(horasDoDia - 8, 0.0);
                    horasPorDia.add(new HorasPorDia(horasNormais, horasExtras, dataAtual));
                    dataAtual = hora.getData();
                    horasDoDia = 0.0;
                }
                horasDoDia += hora.getDuracao();
            }
            double horasNormais = Math.min(horasDoDia, 8);
```

```
            double horasExtras = Math.max(horasDoDia - 8, 0.0);
            horasPorDia.add(new HorasPorDia(horasNormais, horasExtras, dataAtual));
        }
    }

    public List<HorasPorDia> getHorasPorDia() {
        return horasPorDia;
    }

    public double getTotalHorasNormais() {
        return totalHorasNormais;
    }

    public double getTotalHorasExtras() {
        return totalHorasExtras;
    }
}
```

E agora que a lógica do relatório está desenvolvida e testada, podemos utilizá-la dentro do código do controller:

```
public void relatorioHoras(){
    List<HoraLancada> horas = horaLancadaDao.horasDoUsuario(
        usuarioLogado.getUsuario());
    RelatorioDeHoras relatorio = new RelatorioDeHoras(horas);
    result.include("relatorio", relatorio);
}
```

Para terminar, precisamos desenvolver a camada de visualização (`WEB-INF/jsp/horaLancada/relatorioHoras.jsp`):

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:import url="/WEB-INF/jsp/header.jsp"/>

<h2>Relatório de horas cadastradas</h2>

<table class="table">
    <thead>
        <tr>
            <th>Data</th>
            <th>Horas Normais</th>
            <th>Horas Extras</th>
        </tr>
    </thead>
    <tbody>
        <c:forEach var="horasDoDia" items="${relatorio.horasPorDia}">
            <tr>
                <td>${horasDoDia.data.time}</td>
                <td>${horasDoDia.horasNormais}</td>
                <td>${horasDoDia.horasExtras}</td>
            </tr>
        </c:forEach>
    </tbody>
</table>

<c:import url="/WEB-INF/jsp/footer.jsp"/>
```