MAC499 - Trabalho de Formatura Supervisionado
Ginástica Laboral

Luís Felipe Giampaulo Sarro - $n.^o$ USP: 4894666
Professor orientador: Dr. Fábio Kon

6 de fevereiro de 2007

Sumário

1 Introdução

Desde o século passado, presenciamos uma grande revolução na tecnologia. As diversas áreas do conhecimento são invadidas por novas ferramentas, ampliando o leque de possibilidades da atividade humana. O trabalho braçal, cansativo e repetitivo, vai aos poucos dando lugar ao trabalho intelectual, mais criativo e recompensador. Computadores já estão consolidados como ferramenta indispensável na maioria das atividades profissionais, e não conhecê-lo constitui desvantagem competitiva para o mercado de trabalho. No entanto, para que os novos recursos estejam ao nosso favor, é necessário saber usá-los.

Usuários de computadores são freqüentemente acometidos por **lesões de esforço repetitivo** (LER) e **doenças oste-articulares relacionadas ao trabalho** (DORT). Trata-se de ossos do ofício associados à falta de ergonomia durante a operação da máquina por culpa do usuário. Para que tais doenças sejam evitadas, o usuário deve manter uma postura correta, acionar o computador com o menor esforço e praticar atividade física regularmente, evitando o sedentarismo. Fora isso, existem exercícios que, realizados durante o expediente, trazem benefícios para evitar esse incômodo. A prática desses exercícios é chamada **ginástica laboral**, e o *software* que planejamos visa contribuir nesse sentido.

Em 2003, na Reitoria da Universidade de São Paulo, foi implantada a **Ginástica Laboral** como um projeto no curso de Multiplicadores de Qualidade. Seus coordenadores são prof. Christian Klausener e profa. Patrícia Sakai, ambos do **Centro de Práticas Esportivas da USP** (CEPEUSP). Duas vezes por semana, estagiários contratados da **Escola de Educação Física e Esportes** (EEFE) visitam os locais de trabalho participantes e convidam os funcionários para a prática de tal ginástica. Os funcionários afirmam que os exercício aumentam a produtividade, além de promover a socialização e melhorar a auto-estima.

Devido ao sucesso do projeto, outras unidades da USP têm se interessado a receber o atendimento pelo programa. A demanda é grande, bem como a fila de espera. Nosso *software* visa satisfazer essa expectativa.

O objetivo desse trabalho é escrever um programa que permita a todos que trabalham com computadores poder acessar rotinas de exercícios físicos e orientações específicas para realizar a ginástica laboral. Trata-se de uma ferramenta de apoio para quando o estagiário está presente, e também de um complemento quando ele não está. Isso porque realizar a ginástica apenas duas vezes por semana é insuficiente: o ideal é diariamente fazê-la. O *software* será utilizado primeiramente nas unidades da USP, mas, futuramente, o sistema poderá ser oferecido para pessoas e instituições da sociedade em geral.

2 Especificação

O programa deve ser independente de sistema operacional e deve ocupar pouco espaço em disco, para tornar-se acessível a todas as máquinas do *campus*. Atualizado pela *internet*, o sistema baixa a seqüência de exercícios da semana, e, em intervalos de tempo regulares (baseados na configuração do usuário ou do coordenador do grupo), alerta o usuário para fazer uma pausa e realizar a atividade. Essa pausa não é intrusiva, para não atrapalhar o desempenho dos profissionais.

As rotinas de exercícios têm duração média de 15 minutos. Nesse tempo, vários exercícios para várias partes do corpo são sugeridos na tela. Com uma barra de navegação é possível avançar, retroceder e parar a exibição da seqüência. O usuário poderá solicitar exercícios específicos em função de dores no corpo. Para isso, tem de se logar e obter uma autorização de acordo com sua categoria. No futuro, estima-se que o Ginástica Laboral seja oferecido a instituições fora da USP e, embora o *software* seja gratuito, talvez possa ser cobrada uma taxa de serviço caso elas se interessem a ter acesso ao banco de dados personalizado alimentado por profissionais especializados em Educação Física.

O usuário comum pode acessar o banco de dados mas não pode alterá-lo. Os monitores montam a seqüência de exercícios por unidade ou para uma pessoa específica, à distância. Os exercícios, tendo uma representação por seqüência de imagens ou vídeo, são classificados por áreas do corpo e por categoria (sendo elas relaxamento, acupuntura e yoga). Além disso, também são cadastradas dicas de saúde, que são exibidas na inicialização do programa, no estilo das clássicas “dicas do dia”, embora o usuário possa configurar para não vê-las.

3 Metodologia de desenvolvimento

A metodologia de desenvolvimento de *software* utilizada foi a **Programação eXtrema**, também conhecida como **XP**. Trata-se de um método ágil, da mesma família de outras metodologias como SCRUM, DSDM e Crystal. Os princípios do desenvolvimento ágil valorizam:

- indivíduos e interações em vez de processos e ferramentas;
- software funcional em vez de documentação extensa;
- colaboração com clientes em vez de negociação de contratos;
- responder a mudanças em vez de seguir um plano.

Em especial, a **Programação eXtrema** estabelece como valores:

- a **comunicação**

  Cliente e programadores estão em constante interação.

- a **simplicidade**

  Código fácil deve ser detectado e substituído por código simples, que faz somente aquilo que o usuário pediu, a fim de que o escopo do projeto abrace apenas as funcionalidades requisitadas

- **rápido *feedback***

  Pequenas versões prévias do programa (*releases*) são disponibilizadas para o cliente em curto período de tempo.

- **coragem** para abraçar mudanças

  Tendo o cliente contato com versões prévias do produto, é natural que ele clareie suas idéias e solicite mudanças de requisito para que o programa final aproxime-se mais do por ele esperado.

- **respeito** e **qualidade**

XP é indicado para projetos de *software* com requisitos vagos, sujeito a mudanças, e para equipes de porte pequeno ou médio. Das quatro variáveis que as metodologias de desenvolvimento procuram monitorar (são elas tempo, custo, escopo e qualidade), XP adota um **contrato de escopo variável**, afrouxando o escopo para garantir as outras três.

Nos métodos mais tradicionais, supõe-se que o cliente sabe de antemão todas as funcionalidades do projeto, gerando um escopo supostamente completo e imutável. Baseado nisso, as empresas fornecem o custo e o tempo de programação do *software*. Dessa forma, a equipe de desenvolvimento supõe que vai ter pleno controle para conseguir, dentro do tempo estimado, oferecer um *software* de alta qualidade ao cliente, desconsiderando eventuais imprevistos como mudanças na equipe. O problema dessa metodologia é que, na prática, essas quatro variáveis são muitas vezes contraditórias entre si. O cliente pode definir um escopo no começo do projeto

e mudá-lo durante o andamento. Fora isso, imprevistos na empresa de desenvolvimento podem gerar alterações de tempo e custo, sendo necessário sacrificar a qualidade do sistema para poder oferecer ao cliente um produto dentro das estimativas. Nesse ponto, XP prefere garantir o tempo, o custo e a qualidade, sacrificando o escopo se necessário. Com o consentimento do cliente, tarefas de menor prioridade são adiadas ou mesmo canceladas, ao perceber que as estimativas não poderão ser cumpridas (seja por motivo de imprevistos como os já citados ou mesmo devido a mudanças de requisitos por parte do cliente).

Essas decisões são tomadas durante o **jogo do planejamento**, realizado semanalmente: um momento em que cliente troca idéias com os desenvolvedores, estipula prioridade para as novas tarefas e fica sabendo do que já foi e o que será implementado na próxima semana. É fundamental que a comunicação com o cliente seja facilitada. Nem sempre o que profissionais de outras áreas denotam por "rápido", "tipo"ou "lista"corresponde aos conceitos de um programador experiente. Por isso, é preciso traduzir as palavras do cliente para o significado que ele espera dentro do projeto, utilizando, não raramente, **metáforas**.

O time de desenvolvimento busca um **ritmo sustentável** para trabalhar com qualidade durante todo o projeto, sem necessidade de horas extras. No começo de cada dia, é realizada uma reunião curta em que os membros comentam rapidamente sobre tarefas realizadas e a realizar. São **reuniões em pé**, para garantir que não sejam extensas.

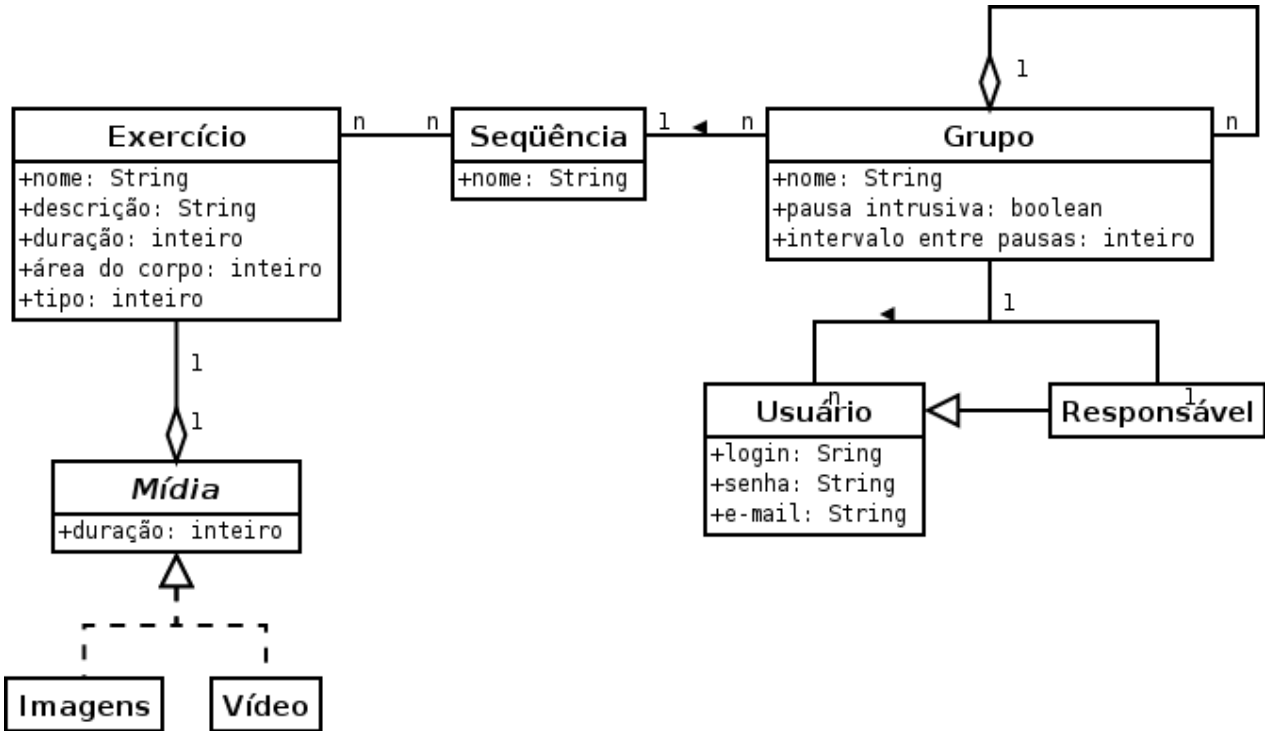
O código do projeto é de **propriedade coletiva**. Nenhuma parte dele possui dono, e qualquer um pode modificar qualquer parte sem aviso prévio. Com isso, busca-se que toda a equipe conheça o sistema por inteiro. Além disso, **padrões de codificação** estabelecidos previamente e respeitados por todos os membros da equipe garantem uniformidade do código, como se tivesse sido gerado por uma mesma pessoa, não importando quantos membros compõem a equipe. A cada nova funcionalidade, a prática da **integração contínua** pede que não se espere para incorporá-la à versão atual do sistema. Integrar de forma contínua diminui a possibilidade de conflitos e erros no código fonte, além de manter o *software* sempre atualizado.

A codificação é feita em duplas (a chamada **programação pareada**). Dois programadores sentam-se em frente a uma mesma máquina e, embora apenas um digite o código, ambos estão revendo, evitando inúmeros bugs e erros de distração causado por trabalhos repetitivos ou cansaço. Fora isso, iniciantes em novas tecnologias podem ser assistidos por programadores experientes, garantindo a evolução da equipe e reforçando a propriedade coletiva do código. O desenvolvimento é todo **orientado a testes**. O código de cada funcionalidade deve ser escrito após seu teste unitário. Além disso, o código da funcionalidade deve ser baseado no teste, evitando conflitos de escopo e favorecendo a simplicidade. Diante das inúmeras mudanças às quais o projeto está sujeito durante o seu desenvolvimento, e considerando o desenvolvimento incremental e a negação de flexibilidades antecipadas, põe-se em questão como garantir a qualidade do código durante meses ou anos de produção. Sabe-se por experiência que código mal-projetado torna-se muito pouco flexível quando novos requisitos são incorporados a ele. A solução são as técnicas de **refatoração**. Existe um catálogo considerável de maneiras de refabricar o código garantindo a compatibilidade com o código existente e melhorando sua qualidade, seja nos aspectos clareza, maximizar coesão ou minizar acoplamento, aumentar reaproveitamento, entre outros.

4 Arquitetura do sistema

Seguindo a especificação do cliente, modelamos as principais estruturas da seguinte forma:

- Uma **seqüência** é uma ordenação de **exercícios**
- Um **exerício** possui uma mídia
- Uma **mídia** pode ser do tipo lista de imagens ou vídeo
- Um **usuário** pertence a um **grupo**
- Um **grupo** tem um **grupo** pai, do qual ele herda as propriedades
- Um **responsável** é um tipo especial de **usuário** com privilégios



O **tipo** do exercício é um inteiro que corresponde às seguintes pré-definições:

```
ACUPUNTURA = 0
ALONGAMENTO = 1
RELAXAMENTO = 2
YOGA = 3
OUTROS = 4
```

A **área do corpo** também é um inteiro pré-definido pelo nosso cliente. Fora isso, o sistema também conta com as **dicas**, que não se relacionam com as estruturas acima citadas.

4.1 Pacotes

A seguir, forneceremos uma breve descrição dos pacotes que compõem nosso sistema.

- **BD**
  Pacote responsável pela persistência dos dados no servidor e no aplicativo *standalone*.
- **comunicacao**
  Responsável pelo autenticação e download de arquivos do servidor.
- **gui**
  Contém classes utilizadas pela interface gráfica. É aqui que os exercícios são visualizados pelo usuário.
- **modelo**
  Implementa basicamente o diagrama UML citado no começo do capítulo.
- **struts**
  Contém as classes para o aplicativo *web*, camada *Control* do arcabouço Struts.

5 Implementação

Devido à necessidade de independência de sistema operacional, escrevemos o nosso *software* em Java para garantir que funcione independente de plataformas. Decidimos que o cadastro de exercícios e rotinas será feito por meio de um portal *web*. Tanto o aplicativo *standalone* como o *web* utilizará um sistema de banco de dados encarregado de armazenar o conteúdo (por parte do administrador) e fazer consultas (por parte do usuário comum).

A seguir, detalhamos que tecnologia adotamos para cada uma das necessidades comentadas.

5.1 Struts

A aplicação *web* é componente importante de diversos sistemas comerciais. Sendo uma aplicação robusta que interage com várias tecnologias e atravessa várias comadas do sistema, não é trivial obter sucesso ao desenvolvê-la. Mas para o nosso auxílio, apoiamo-nos no trabalho de centenas de pesquisadores e adotamos o arcabouço Struts para realizar essa parte da implementação.

A grande maioria das aplicações web são muito parecidas: do navegador, o usuário manda uma requisição HTTP, a página web é gerada, fornecida de volta e lida pelo navegador novamente. O Struts é um arcabouço (e portanto um sistema semicompleto) que se aproveita dessa semelhança para resolver o problema genericamente. Criado por Craig R. McClanahan e doado para a **Fundação Apache** em 2000, Struts envolve diversas tecnologias, implementadas segundo o padrão *Model-View-Controller* (MVC-2). Esse conjunto de tecnologias pode dificultar o aprendizado da ferramenta, mas as inúmeras vantagens que ela oferece faz valer a pena. São elas:

- **internacionalização**

  Os rótulos da interface e as mensagens são guardadas em um arquivo `.properties`. É possível desenvolver interfaces para vários idiomas apenas traduzindo esse arquivo.

- **distribuição de trabalho**

  Sua clara divisão de camadas permite separar o *design* da programação. O *designer* trabalhará apenas na parte *view* do MVC, utilizando as *tags* do struts em conjunto com JSP, Velocity, etc., sem se importar com a lógica de negócios.

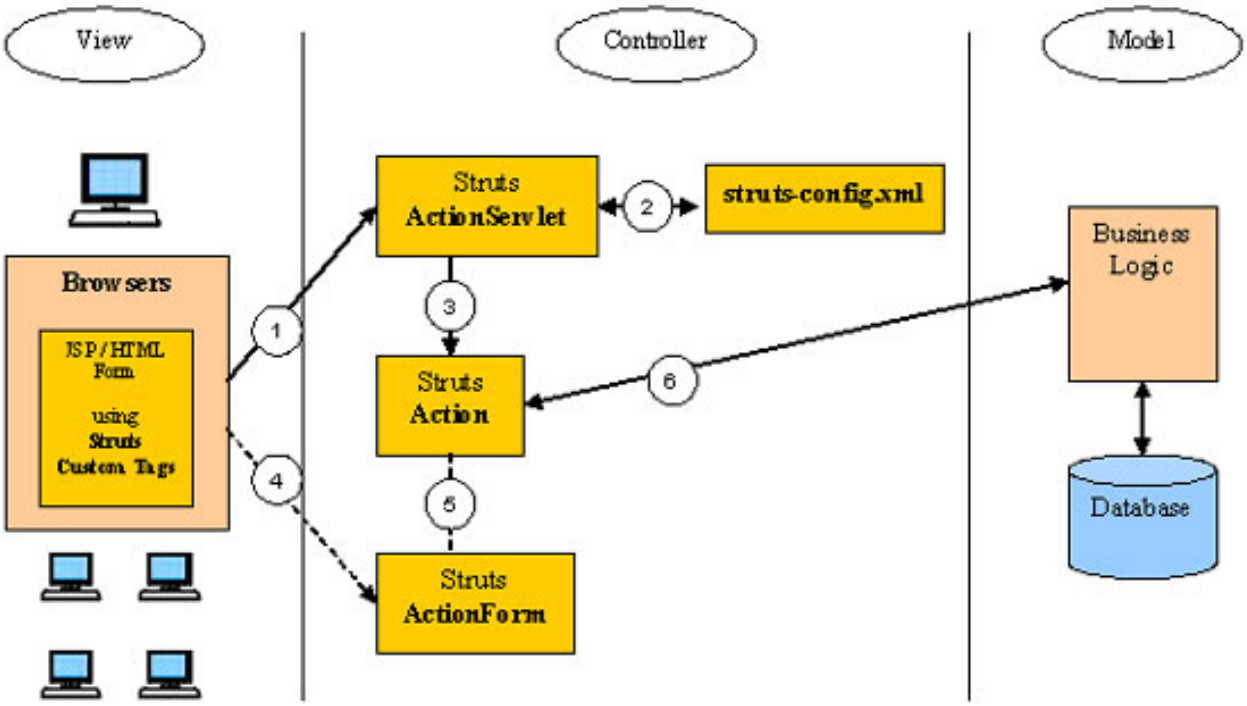
- ***performance***

  Struts é leve e sua utilização melhora o desempenho das aplicações web.

- **reutilização de código**

  As estratégias de localização do Struts reduzem a necessidade de JSPs redundantes. Seus componentes são compatíveis com os padrões e são reutilizáveis pela aplicação.

5.1.1 Fluxo de uma aplicação em Struts



Seguindo a figura acima, há duas maneiras de fazer uma requisição HTTP: (1) utilizando um *servlet* ou (4) por meio de um formulário HTML. No caso do *servlet*, a solicitação é (2) mapeada no arquivo `struts-config.xml`, lida e em seguida o *ActionServlet* define o (3) *Action* correspondente para a solicitação.

Num *Action* pode-se validar a entrada, enviar mensagens de erro, (6) consultar o banco de dados, acessar a lógica de negócios, entre outras coisas.

No caso do formulário HTML, a entrada é armazenada num *JavaBean*, que pode ser considerado uma classe Java com *getter* e *setters* para todos os atributos públicos. O *Action* (5) então pode operar sobre o *FormBean* e armazenar o resultado em um *ResultBean*.

5.1.2 Struts aplicado no nosso projeto

Armazenamos todas nossas *actions* no arquivo `struts-config.xml`. As *actions* têm em geral uma página de entrada e uma página de redirecionamento, após sua execução. Ambas as páginas foram escritas em JSP utilizando as *taglibs* do Struts. As páginas JSP fazem o papel do módulo View no modelo MVC, mas poderíamos ter usado qualquer outro equivalente compatível (por exemplo Velocity). A cada submissão de formulário mal preenchido, a mensagem de erro é exibida no canto superior da página. O Struts troca a tag `<html:erros/>` pela mensagem de erro programada na classe *action* correspondente. Todas as outras mensagens estão reunidas num único arquivo `messages.properties`, mencionado acima.

Estrutura simplificada do nosso `struts-config.xml`:

```
<struts-config>
```

```
<form-beans>
 <form-bean name="exercicioForm" type="struts.ExercicioForm" />
</form-beans>


  <action
   path="/criarExercicio"
   forward="/criarExercicio.jsp" >
  </action>

  <action
   path="/enviarExercicio"
   type="struts.ExercicioAction"
   name="exercicioForm"
   input="/criarExercicio.jsp"
   scope="session"
   validate="true" >
  <forward name="valid" path="/listarExercicios.do" />
  </action>

  <action
   path="/adicionaArquivo"
   type="struts.AdicionaArquivoAction"
   name="exercicioForm"
   input="/criarExercicio.jsp"
   scope="session"
   validate="false" >
  <forward name="valid" path="/criarExercicio.jsp" />
  <forward name="comVideo" path="/criarExercicioVideo.jsp" />
  <forward name="erro" path="/criarExercicioErro.jsp" />
  </action>

  <action
   path="/listarExercicios"
   type="struts.ListarExerciciosAction"
   name="listarExerciciosForm"
   scope="request"
   validate="false" >
   <forward name="valid" path="/listaDeExercicios.jsp" />
  </action>
```

5.2 Hibernate

Nossa aplicação Java utiliza base de dados relacional. Precisamos realizar consultas e atualizações de dados. Para isso, adotamos a utilização do arcabouço **Hibernate** para facilitar a

construção do aplicativo e diminuir a complexidade resultante da convivência dos dois modelos distintos: a **orientação a objetos**, da linguagem Java, e o **modelo relacional**, do nosso SGBD (Sistema Gerenciador de Banco de Dados, no nosso caso HSQLDB).

Dessa forma, por meio de arquivos de configuração do tipo `.hbm.xml` fazemos o mapeamento das classes, heranças e associações dos objetos em relacionamentos de entidades e tabelas para o HSQLDB. No entanto, as transações continuam sendo de responsabilidade do SGBD. Embora o Hibernate faça o mapeamento objeto-relacional, o gerenciamento de transações é delegado para outros elementos da infraestrutura da aplicação.

O primeiro arquivo a ser configurado é o `hibernate.cfg.xml`. Nele, indicamos a classe Java do driver JDBC (API que faz o envio de instruções SQL para qualquer banco de dados relacional), url do arquivo de dados, login e senha para identificação do usuário e a lista dos arquivos de mapeamentos `.hbm.xml`. Nosso `hibernate.cfg.xml` está mais ou menos assim:

```
<hibernate-configuration>

    <session-factory>

        <property name="connection.driver_class"\>org.hsqldb.jdbcDriver</property>
        <property name="connection.url"\>jdbc:hsqldb:file:/home/xp06/caspio/ginastica/Gina
        <property name="connection.username"\>sa</property>
        <property name="connection.password"\></property>

        <property name="connection.pool_size"\>1</property>

        <property name="dialect"\>org.hibernate.dialect.HSQLDialect</property>

        <property name="current_session_context_class"\>thread</property>

        <property name="cache.provider_class"\>org.hibernate.cache.NoCacheProvider</proper

        <property name="show_sql"\>true</property>

        <mapping resource="modelo/Dica.hbm.xml"/>
        <mapping resource="modelo/Exercicio.hbm.xml"/>
        <mapping resource="modelo/Sequencia.hbm.xml"/>
        <mapping resource="modelo/Usuario.hbm.xml"/>
        <mapping resource="modelo/Grupo.hbm.xml"/>
        <mapping resource="gui/midia/Imagem.hbm.xml"/>
        <mapping resource="gui/midia/Midia.hbm.xml"/>

    </session-factory>

</hibernate-configuration>
```

A seguir, veremos como mapear alguns tipos de relacionamento utilizando **Hibernate**.

5.2.1 Relacionamento n-para-n

De acordo com a arquitetura do nosso sistema, embora uma seqüência contenha uma lista de exercícios, cada exercício pode pertencer a várias seqüências diferentes. Por isso, trata-se de um relacionamento n-para-n, e não n-para-1.

Esse tipo de relacionamento exige uma terceira tabela para relacionar os índices das duas entidades. No exemplo de mapeamento abaixo, utilizamos uma coleção do tipo *list* para armazenar os exercícios da seqüência. Fizemos essa escolha pois *list* é uma coleção ordenada. Outra possibilidade seria *set*, mas já não contava com esse recurso.

A *tag* `<many-to-many>` indica o relacionamento n-para-n. É importante notar que não queremos nenhum tipo de efeito "cascata"(isso quer dizer, quando removermos uma seqüência, não queremos que o exercício seja também removido). Por isso o parâmetro `cascade="none"`.

```
<class name="modelo.Sequencia" >

        <id name="ID" >
            <generator class="increment"/>
        </id>

        <property name="nome" />

        <list name="listaDeExercicios" lazy="false"
        cascade="none" table="SEQUENCIA_EXERCICIO" >
        <key column="ID_SEQUENCIA" />
         <index column="idx" />
         <many-to-many column="ID_EXERCICIO" class="modelo.Exercicio"/>
        </list>

</class>
```

5.2.2 Relacionamento n-para-1

Segundo nossa modelagem, cada usuário pertence a um grupo numa relação n-para-1. Esse mapeamento é simples, bastando na *tag* `<many-to-one>` indicar o campo e a classe do elemento de multiplicidade um. O efeito cascata também é indesejável aqui, pois não queremos que todo o grupo seja removido no caso de apenas um usuário deixar de fazer parte dele.

```
<class name="modelo.Usuario" table="USUARIO" >
        <id name="ID">
            <generator class="increment"/>
        </id>
        <property name="tipo"/>
        <property name="nome"/>
        <property name="login"/>
        <property name="senha"/>
        <property name="email"/>
```

```
        <many-to-one name="grupo" class="modelo.Grupo"
         cascade="none" />
</class>
```

O mesmo raciocínio serve para o relacionamento recursivo de grupo e grupo pai, conforme código abaixo.

```
<class name="modelo.Grupo" table="GRUPO" >
        <id name="ID">
            <generator class="increment"/>
        </id>
        <property name="nome" />
        <property name="pausaIntrusiva" />
        <property name="intervaloDoCoordenador" />

        <many-to-one name="pai" class="modelo.Grupo" cascade="none" lazy="false"/>
</class>
```

5.2.3 Relacionamento 1-para-1

No relacionamento 1-para-1, relacionamos duas entidades de tabelas diferentes utilizando o mesmo ID. No nosso programa, mídia relaciona-se univocamente com exercício com esse tipo de relacionamento. A *tag* `<generator class="foreign»` do mapeamento da entidade mídia indica que seu ID é o mesmo que o da entidade exercício com a qual se relaciona. Nesse caso, o efeito "cascata"faz-se necessário.

```
<class name="modelo.Exercicio" >

        <id name="ID" >
            <generator class="increment"/>
        </id>

        <property name="nome"/>
        <property name="descricao"/>
        <property name="duracao"/>
        <property name="area"/>
        <property name="tipo"/>

        <one-to-one
         name="midia"
         class="gui.midia.Midia"
         cascade="all"/>

</class>
```

```
<class name="gui.midia.Midia" >

        <id name="ID" column="exercicio_id" >
            <generator class="foreign" >
             <param name="property" >exercicio</param>
            </generator>
        </id>

       <one-to-one
         name="exercicio"
         class="modelo.Exercicio"
         constrained="true" />

</class>
```

5.2.4 Herança

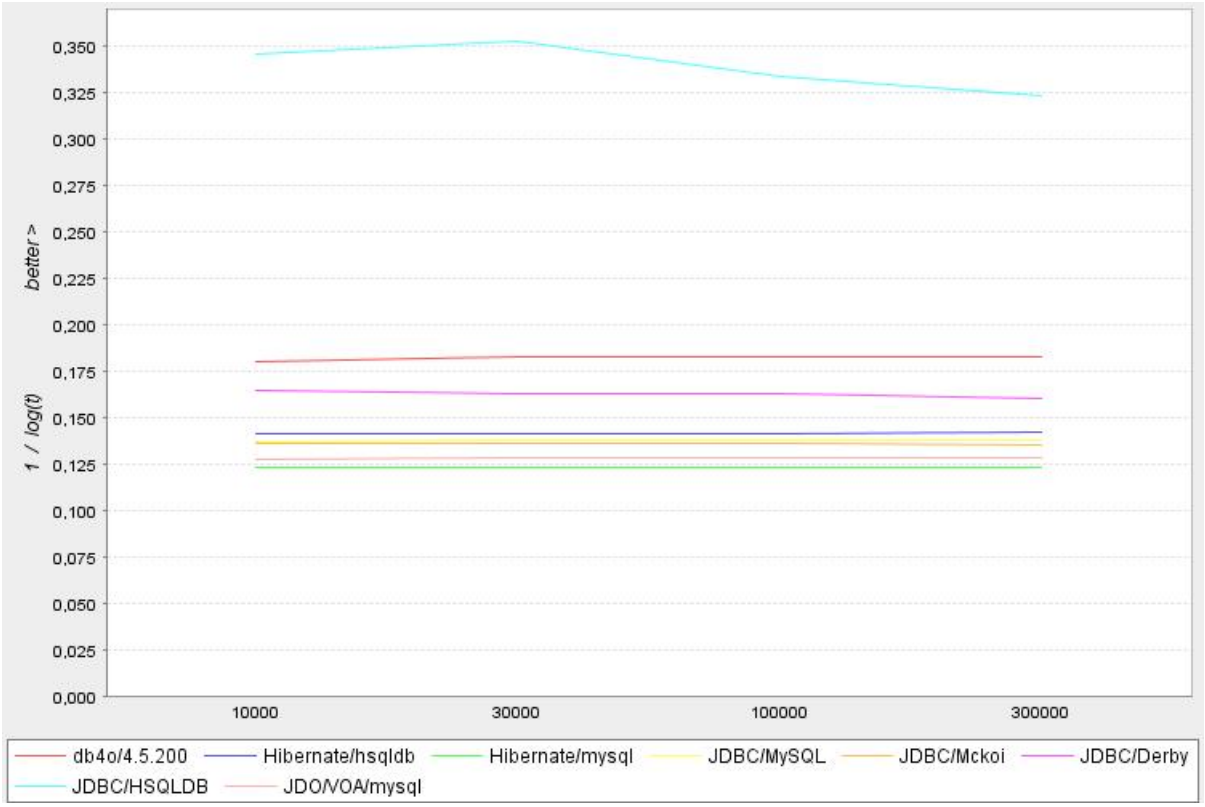
Mídia é uma classe abstrata, podendo ser implementada por uma classe do tipo Imagens ou do tipo Vídeo. Para mapear esse tipo de herança, no mesmo arquivo de mapeamento da classe mãe inserimos o mapeamento das classes filhas dentro da *tag* `<joined-subclass>`, conforme código abaixo:

```
<joined-subclass name="gui.midia.Imagens" table="IMAGENS" >
<key column="Midia_id"/>
        <list name="imagens" lazy="false" cascade="all" >
         <key column="MIDIA_IMAGEM_ID" />
         <index column="IDX" />
          <one-to-many class="gui.midia.Imagem" />
         </list>
</joined-subclass>


<joined-subclass name="gui.midia.Video" table="VIDEO" >
        <key column="Midia_id"/>
         <property name="endereco"/>
</joined-subclass>
```

5.3 HSQLDB

Como citamos anteriormente, o SGBD que elegemos para o nosso sistema é o HSQLDB. Trata-se de um banco de dados livre, com inúmeras vantagens: é flexível, multiplataforma e o *software* de suporte é pequeno, cabendo num disquete e podendo ser fornecido junto com a distribuição do aplicativo. Com ele é possível manipularmos as tabelas em disco, em memória ou em formato texto, e sua *performance* é 20 vezes mais veloz que a dos outros SGBDs do mercado, como a figura abaixo demonstra.



Nosso banco de dados é composto por três arquivos:

- `ginLab.properties`

  Arquivo de propriedades com os ajustes sobre o banco de dados. Versão do SGBD, características do *cache* e se o banco será inicializado em modo somente leitura são algumas das definições dele.

- `ginLab.script`

  Esse arquivo contém o *script* SQL para a criação das tabelas do banco de dados. Gera estrutura de índices, restrições, usuários, além de ser responsável pela persistência.

- `ginLab.log`

  Registro das últimas alterações realizadas sobre o banco.

5.4 JUnit

Já dissemos que um dos pilares no qual a **Programação eXtrema** se sustenta é a programação orientada a testes. Para tornar os testes automatizados, utilizamos o arcabouço **JUnit** durante o desenvolvimento de nosso aplicativo.

JUnit facilita a criação de testes unitários, que são uma maneira de testar o menor dos componentes de um sistema isoladamente. Com ele, executamos os testes rapidamente, sem interromper o processo de desenvolvimento. Também é possível criar uma hierarquia de testes para testarmos apenas parte do sistema.

Cada teste do **JUnit** é composto por uma seqüência de estímulos (chamadas de método) e respostas esperadas. O trabalho do **JUnit** é rodar todos os estímulos numa ordem indeterminada e comparar as respostas obtidas.

Embora seja uma ferramenta robusta e de grande auxílio, sua maior vantagem é o conceito que ela inseriu na comunidade dos desenvolvedores. O arcabouço em si é simples e sua última versão foi lançada há mais de três anos. Isso mostra que se trata de um *software* útil, embora não seja necessário tê-lo para praticar os princípios por ele estabelecidos.

5.5 log4j

Para auxiliar a depuração, configuramos o TomCat em conjunto com o **log4j**, uma ferramenta para registro de logs. A utilização de logs ajuda a depuração e complementa os testes.

Uma das vantagens do **log4j** é o desempenho. Há quem comente que sentenças de log no meio do código acarreta poluição e redução da legibilidade. Na linguagem Java, em que não existe pré-processamento, inserir registros de logs no meio do código diminui a velocidade da aplicação, mesmo quando os logs são desativados. Os efeitos são ampliados conforme o tamanho do programa. Com log4j, as sentenças de log são ativadas em tempo de execução, sem pesar no desempenho. As configurações são editadas num arquivo à parte, sem alterar os binários do sistema, como no exemplo abaixo:

```
log4j.rootLogger=DEBUG, dest1

log4j.appender.dest1=org.apache.log4j.ConsoleAppender
log4j.appender.dest1.layout=org.apache.log4j.PatternLayout
log4j.appender.dest1.layout.ConversionPattern=%d %-5p %-5c{3} %x -> %m%n
log4j.appender.dest1.MaxFileSize=100KB
```

Outro recurso interessante do log4j é a possibilidade de estipular uma hierarquia entre os logs. De acordo com a necessidade, é possível exibir menos ou mais detalhes, segundo um sistema de herança entre as categorias dos logs. Isso permite uma melhor seleção do tipo de saída que se deseja, pois não é sempre que queremos obter todas as informações registradas.

Acreditamos que log4j será ainda mais útil quando o nosso sistema entrar em fase experimental de teste e depois quando já estiver em funcionamento. Dessa forma, teremos controle sobre as operações mal-sucedidas no sistema à distância.

5.6 SVN

Como ferramenta para nos ajudar na **integração contínua**, um dos pilares da metodologia XP, utilizamos o **SVN**. Trata-se de um sistema de controle de versão. Com ele, a cada nova funcionalidade implementada, enviamos os novos arquivos ao repositório e é registrada uma nova versão do sistema. Se precisarmos voltar atrás para recuperar alguma parte que mudamos acidentalmente, **SVN** nos ajuda.

Entretanto, nossa principal utilização do SVN foi para garantir que todos os membros que trabalharam no projeto tivessem acesso à última versão do código da aplicação. Isso nos foi bastante útil no primeiro semestre de 2006, quando contávamos com quatro desenvolvedores

que trabalhavam simultaneamente. SVN avisa quando há conflitos entre as mudanças do código e ajuda a solucionar.

Seu uso diminuiu drasticamente no segundo semestre, quando o sistema tornou-se apenas de minha responsabilidade e a necessidade da integração contínua desapareceu.

5.7 XPlanner

Trata-se de uma ferramenta criada para auxiliar o desenvolvimento de sistemas que utilizam XP. Com ela, registrávamos as histórias que nosso cliente escrevia, estimávamos os cartões, definíamos a duração de cada iteração e podíamos comparar nosso desempenho com auxílio de gráficos e estatísticas geradas.

No segundo semestre, sua função no desenvolvimento do sistema Ginástica Laboral reduziu-se à de uma agenda onde eu anotava o que já foi implementado e o que ainda deveria ser. XP já não era aplicado em sua totalidade, pois havia apenas um desenvolvedor.

6 Resultados obtidos

Com a utilização da metodologia XP, logo no começo de 2006 notamos um desenvolvimento rápido dos alicerces do sistema, o que me pareceu impressionante. Logo tínhamos os requisitos mais básicos do *software* em funcionamento, o que causou uma boa impressão ao nosso cliente. Naturalmente, conforme a implementação avançava e a complexidade aumentava, tornava-se mais difícil integrar as diversas partes do sistema e os resultados tornavam-se menos aparente, até mesmo porque exigiam uma sutileza maior da codificação e um trato fino de exceções e mensagens de erro.

Conseguimos entregar uma versão quase completamente funcional no final de novembro, faltando apenas implementar a segurança na autenticação do sistema. Até a conclusão dessa monografia, não foi possível colocar o *software* em fase experimental. Nas próximas semanas, haverá uma estagiária da EEFE alimentando o sistema e redigindo a documentação para o usuário final. Será um período de depuração e refinamento, e talvez surja a necessidade de mais requisitos. O *feedback* do usuário é fundamental.

6.1 Trabalhos futuros

Uma vez o programa em funcionamento, deseja-se ter uma estimativa de quantos funcionários realmente interrompem o trabalho e realizam a ginástica laboral. A implementação da geração desses dados fica para o ano que vem.

Até o presente momento, a contratação do *designer* para a identidade visual do sistema está em bom andamento. Já temos um rascunho da nova interface *web* que em breve será implementada.

Da última reunião com o cliente, cogitou-se a possibilidade de, no ano que vem, complementar os materiais de exercícios com *podcast* para meditação. Os áudios seriam gerados pelos profissionais da Educação Física e disponibilizados para os usuários cadastrados. Trata-se, entretanto, de um recurso de menor prioridade, dado que o pleno funcionamento do sistema é primordial.

7 Parte subjetiva

Embora eu goste muito de matemática e computação teórica, e disciplinas como Análise de Algoritmos e Otimização Combinatória estejam entre minhas favoritas do BCC, gostei muito de ter escolhido esse projeto como trabalho de formatura, pois percebia que havia uma falta de disciplinas de produção de *software* no meu currículo, e não tenho a intenção de me tornar acadêmico. O contato com diversas ferramentas amplamente utilizadas no mercado de trabalho (como Struts, JSP, Hibernate) atuou de maneira a complementar minha formação: sendo assim, adquiri toda formação fortemente acadêmica que o currículo de computação do IME-USP oferece e ainda pude obter conhecimentos pragmáticos que são muitas vezes requisitos para diversas vagas de emprego na área. Dessa forma, concluo o meu bacharelado com satisfação, tendo podido unir os dois aspectos complementares da Ciência da Computação.

7.1 Disciplinas do BCC decisivas para esse projeto

É claro que, de uma forma geral, disciplinas fundamentais da graduação como *Princípio de desenvolvimento de algoritmos*, *Estrutura de dados*, *Análise de algoritmos* e *Conceitos fundamentais de linguagens de programação* estão presentes em qualquer atividade relacionada à computação, ainda que indiretamente. No entanto, o conteúdo com que lidei de perto foi o das disciplinas:

- **MAC0332 - Engenharia de Software**
  Primeira disciplina em que tive contato com os problemas e as técnicas para desenvolver um *software* grande que seja de fácil manutenção. Aprendi noções de análise de requisitos, especificação e testes unitários.

- **MAC0340 - Laboratório de Engenharia de Software**
  Pude aplicar de forma melhor os conceitos aprendidos na disciplina de engenharia. A matéria aborda a questão do desenvolvimento de uma maneira oposta à da metodologia XP. Contribuiu para o meu amadurecimento. Atualmente, penso que não existe metodologia melhor ou pior do que a outra, mas sim mais ou menos adequada. Conforme o sistema de Ginástica Laboral foi crescendo, foram aparecendo dificuldades que a metodologia XP não solucionava, e me pareceu que a utilização de algumas técnicas da engenharia clássica antes da implementação do aplicativo teria facilitado nosso trabalho.

- **MAC0342 - Laboratório de Programação eXtrema**
  Foi a disciplina onde o desenvolvimento do sistema começou, sugerindo uma nova metodologia de engenharia de *software*: um método ágil em espiral. Pude apreciar suas estratégias baseadas em resultado, no entanto acredito que a redução da documentação do *software* pode dificultar a manutenção do sistema. Embora XP possa transpassar uma idéia de relaxamento dos fundamentos da engenharia de *software*, na verdade ele exige ainda mais disciplina do que os outros métodos, caso contrário o sistema demora para avançar e pode tornar-se um grande elefante branco.

- **MAC0426 - Sistemas de Banco de Dados**
  Essa disciplina me foi familiar desde o começo, pois eu costumava escrever sistemas de

banco de dados em Visual Basic quando cursava a oitava série. Foi muito útil buscar maior esclarecimento e aprender teorias que fundamentam as técnicas adotadas, até mesmo porque eu nunca havia conseguido terminar meus sistemas em Visual Basic em virtude de problemas que não sabia solucionar até então. Ginástica Laboral foi o meu primeiro projeto de banco de dados com sucesso.

- **MAC0441 - Programação Orientada a Objetos**
  As noções de orientação a objetos estão presentes no BCC desde a primeira disciplina de introdução à programação. No entanto, alguns conceitos mais aprofundados dessa disciplina foram úteis para uma melhor compreensão dos fundamentos de XP, como exemplo a refatoração.

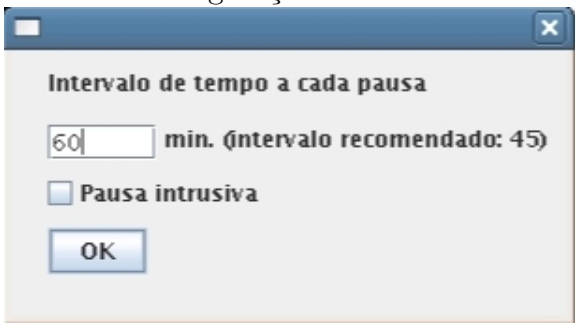
- **PCS0210 - Redes de Computadores**
  Essa disciplina serviu como base para que eu não me sentisse totalmente perdido ao trabalhar com os protocolos de rede pela primeira vez num projeto de computação. A disciplina tem um caráter muito mais conceitual do que prático, então lembrar-se de seu conteúdo era mais uma forma de elucidação técnica do que de solução direta de problemas.

8 Apêndice

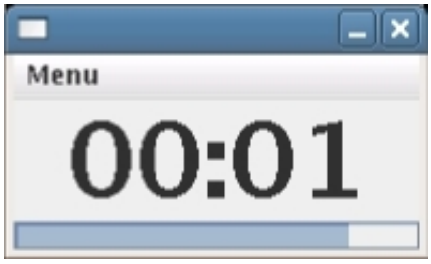
8.1 *Screenshots*

Sessão de exercícios em andamento:

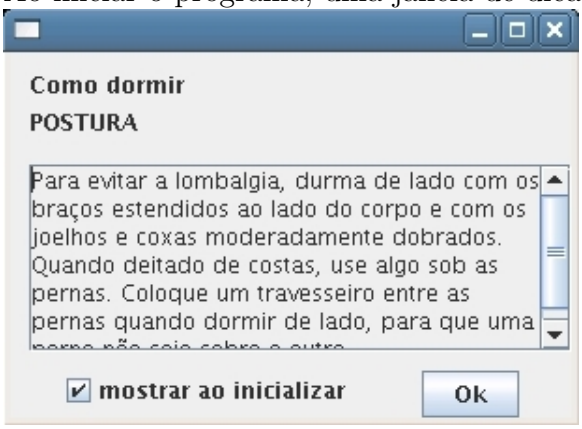
Janela de configuração do usuário:



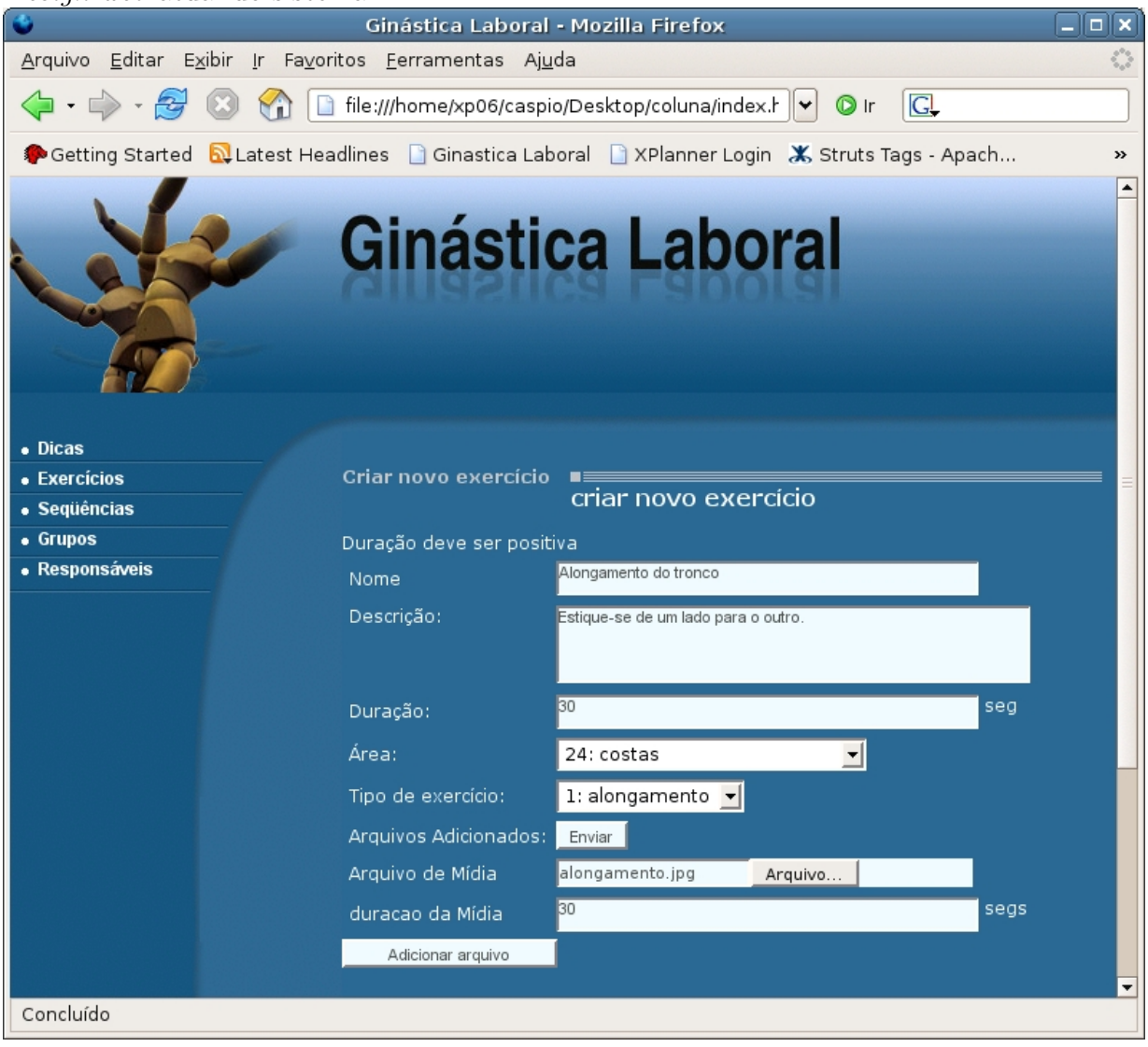
Janela principal do programa, indicando quanto tempo falta para a próxima sessão de ginástica laboral:



Ao iniciar o programa, uma janela de dicas é exibida:



Design web atual do sistema:



8.2 Links e bibliografia

- Ginástica Laboral: http://www.usp.br/espacoaberto/arquivo/2005/espaco57jul/print/ptcapa.htm
- HUSTED, Ted, DUMOULIN, Cedric, FRANCISCUS, George, WNTERFELDT, David **Struts in Action** Greenwich: Manning Publications, 2004.
- BECK, Kent *Programação Extrema Explicada* Bookman, 2004
- Hibernate Reference Documentation Version 3.0.5: http://www.hibernate.org