Paginador

Intenção

Fornecer um mecanismo que permita o acesso a um conjunto de objetos por partes, definidas como páginas, mantendo o controle da navegação dos objetos na página corrente.

Motivação

Vamos supor uma aplicação para área hospitalar que, entre várias funcionalidades, salva em um meio persistente a informação de todos os medicamentos que foram consumidos. No final de cada mês um funcionário do departamento de suprimentos deve acessar uma listagem desses medicamentos consumidos e armazenar um histórico. Essa listagem deve apresentar o nome de todos os medicamentos movimentados de forma ordenada.

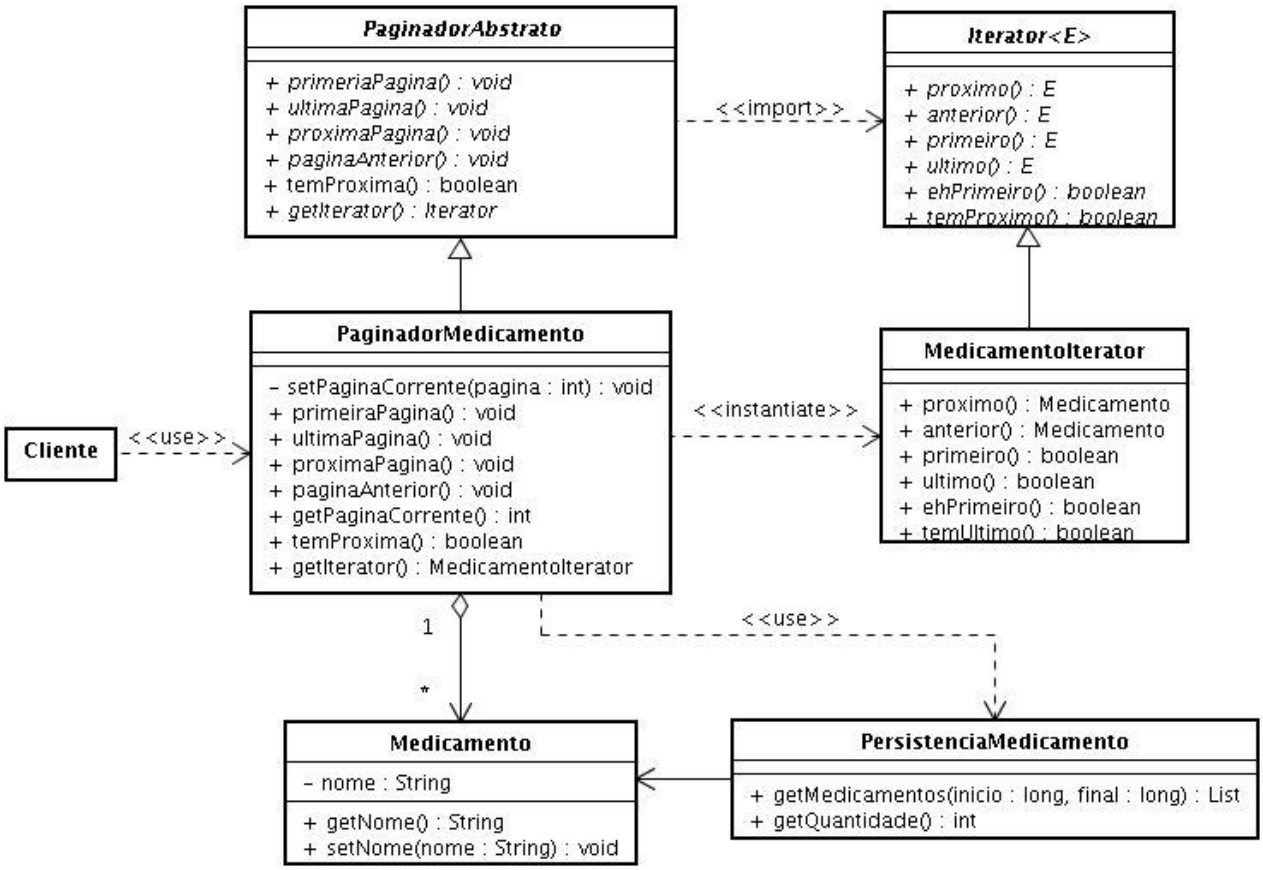
No cenário acima, o problema que surge é que o sistema pode manter uma quantidade muito grande de objetos armazenados no meio persistente e não é possível carregar todos esses objetos na memória para fazer a geração dessa listagem.

Para resolver o problema, podemos criar uma classe `PaginadorMedicamento` que será responsável por obter do meio persistente os objetos do tipo `Medicamento` em blocos ordenados. Isso quer dizer que quando alguma classe precisar processar a lista de `Medicamentos` ela deverá fazer o pedido para `PaginadorMedicamento` que acessará a persistência e retornará um bloco com uma quantidade pré-definida de objetos do tipo `Medicamento`, retornado esse bloco para o cliente. Depois de o cliente ter feito o processamento, ele poderá pedir para o `PaginadorMedicamento` o próximo bloco e assim sucessivamente até que todos os objetos requisitados tenham sido processados.

A classe `PaginadorMedicamento` deve gerenciar as operações de paginação mantendo informações a respeito do primeiro e último objetos que pertencem à página atual pois essas informações serão necessárias na hora fazer o acesso à próxima página ou página anterior.

Através do `PaginadorMedicamento` o cliente poderá acessar todo o conjunto de `Medicamentos` disponibilizados por parte, e dessa forma não será necessários manter todos os objetos na memória ao mesmo tempo.

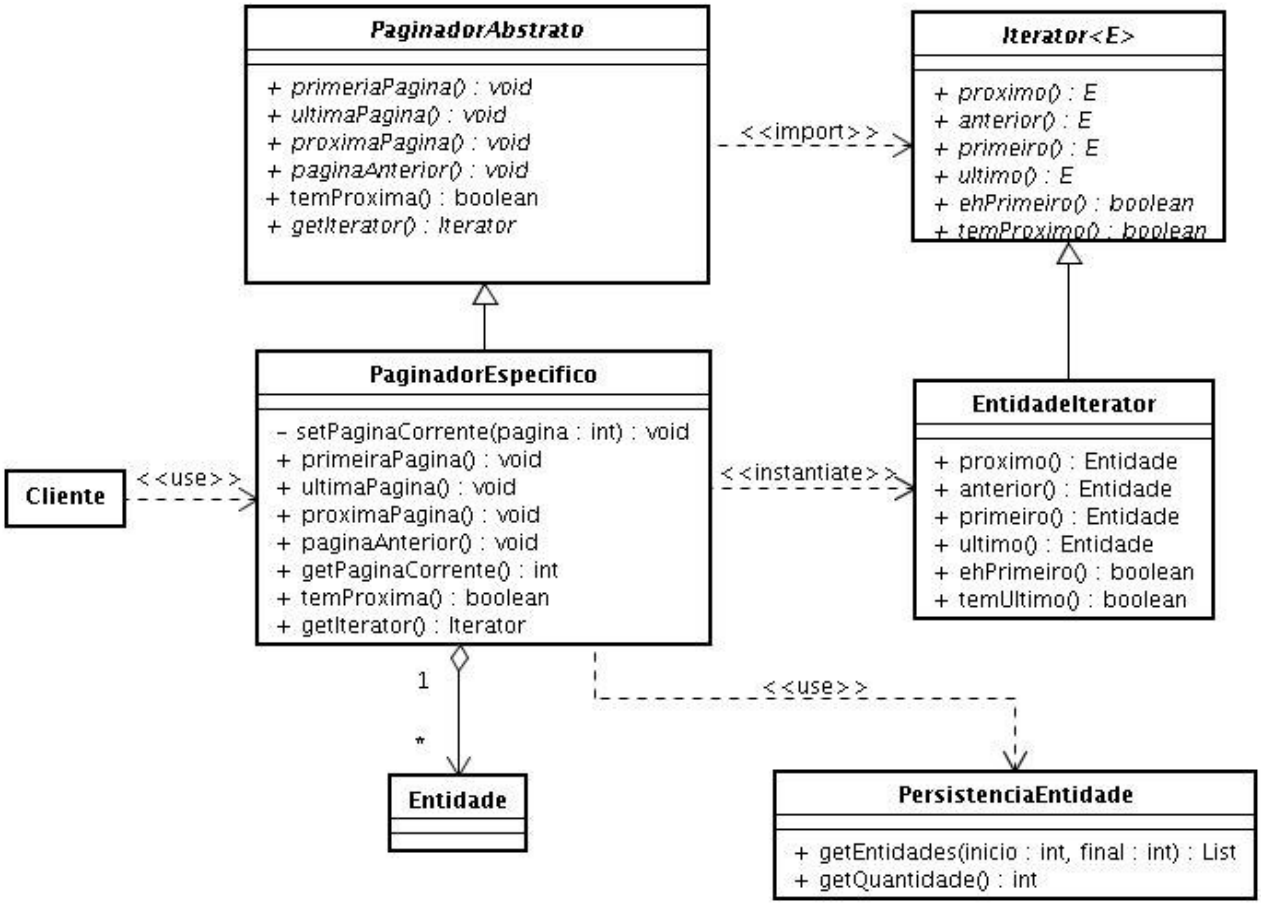
Para facilitar a navegação, `PaginadorMedicamento` retornará uma implementação de um Iterator [**GoF**] específico para permitir a navegação nos objetos da página corrente.



Aplicabilidade

- Um sistema necessita acessar uma quantidade muito grande de objetos, mas não pode carregá-los todos de uma vez na memória;
- Permitir a navegação em um conjunto de objetos encapsulando a forma de acesso a esses objetos e mantendo o estado atual da navegação.

Estrutura



Participantes

- `Cliente`: Qualquer classe que utilize os serviços do `PaginadorEspecifico`.
- `PaginadorAbstrato`: Interface básica que define o comportamento de Paginadores.
- `PaginadorEspecifico`: Paginado que contem o conhecimento a respeito a área de negócio e sabe como acessar o mecanismo persistente para recuperar os objetos além de implementar os mecanismos de navegação da melhor forma.
- `Entidade`: Entidade de negócio que o Paginador armazenará. Na prática pode ser qualquer tipo de objeto.
- `PersistenciaEntidade`: Classe responsável pela recuperação das entidades do meio persistente.
- `Iterator`: Interface responsável por definir os comportamentos do padrão Iterator.
- `EntidadeIterator`: Implementação concreta da interface Iterator para fazer tratamento específico de navegação dos objetos do tipo Entidades mantida pelo `PaginadorEspecifico`.

Colaborações

- O cliente envia uma mensagem para o `PaginadorEspecifico` pedindo que ele navegue para a primeira ou última pagina, bem como para a próxima ou anterior.
- O `PaginadorEspecifico` acessa um objeto de `PersistenciaEntidade` para recuperar a lista de objetos armazenados no meio persistente, obedecendo as informações

referente a página atual e o pedido de navegação (primeira ou última página, anterior ou próxima);
- `PaginadorEspecifico` cria uma instância da implementação concreta de `Iterator` e retorna ao cliente, para que esse cliente possa fazer a navegação nos objetos mantidos na página.

Conseqüências

1. O padrão Paginador permite que grandes quantidades de dados, representados como objetos possam ser acessados e manipulados sem que estes estejam todos carregados ao mesmo tempo na memória;
2. Tem a capacidade de prover o acesso por páginas (ou blocos) de objetos mantendo o controle da navegação sobre esse conjunto de objetos;
3. Serve como um controlador de navegação em um conjunto de objetos. Quando utilizado nesse sentido, a diferença entre o Paginador e o Iterator é o fato do Paginador manter informações a respeito das páginas na qual está sendo feita a navegação, bem como informações que permitam a recuperação dessas páginas.

Implementação

O padrão Paginador pode ser implementado de várias formas. Apresentaremos abaixo algumas dessas estratégias para implementação.

1. A implementação do `PaginadorAbstrato` ser feito como interface deixando toda a lógica de navegação por conta da implementação concreta, ou seja, do `PaginadorEspecifico`;
2. Outra alternativa é permitir que o `PaginadorAbstrato` seja definido utilizando o conceito de Template (mais especificamente Generics em Java) permitindo que todos os controles de navegação sejam encapsulado dentro dele, tornando as implementações das classes descendentes mais simples;
3. Uma outra abordagem é fazer com que `PaginadorAbstrato` seja uma implementação do próprio padrão Iterator, permitindo assim um nível ainda mais alto de abstração. A desvantagem dessa implementação é o fato de tornar as semânticas de uso dos iteradores no sistema mais complexo, pois teríamos iteradores de páginas e iteradores de objetos mantidos pela página;
4. A implementação do `PaginadorAbstrato`, `PaginadorEspecifico`, `Iterator` e `EntidadeIterator` pode ser feita de uma forma mais integrada permitindo que o `PaginadorAbstrato` e `PaginadorEspecifico` detectem quando o usuário tenta acessar um objeto que não está na página corrente (o primeiro objeto além do último da página corrente ou o último objeto da página anterior a página corrente) e fazer a carga automática dessa página. Dessa forma o Paginador toma uma característica de memória temporária de objetos (“*pool”)*
5. O `PaginadorAbstrato` e o `PaginadorEspecifico` podem ainda conter regras de ordenação permitindo que a coleção seja ordenada na memória. Esse tipo de solução pode ser complexa ou então adequar-se somente a algumas situações, pois uma página não contem todos os objetos da coleção completa e assim sendo, essa ordenação não é uma ordenação dos objetos da todos e sim uma ordenação de somente os objetos da página corrente.
6. Uma outra implementação mais sofisticadas é permitir que as páginas contenham objetos que possam ser removidos enquanto estão sendo iterados e essa alteração refletir na sua

persistência.

Exemplo de Código

Para demonstrar uma forma de implementação do padrão Paginador utilizaremos o primeiro exemplo apresentado para o `PaginadorMedicamento`.

A classe `PaginadorAbstrato` será definida como abstrata e terá como função definir o comportamento básico de todos os Paginadores. A vantagem de haver uma classe como esta é a possibilidade de utilizar fábricas de paginadores.

```
public abstract class PaginadorAbstrato {
     public abstract void primeiraPagina();
     public abstract void ultimaPagina();
     public abstract void proximaPagina();
     public abstract void paginaAnterior();
     public abstract boolean temProxima();
     public abstract Iterator getIterator();
}
```

A classe `PaginadorMedicamento` será responsável por gerenciar as páginas e fornecer um mecanismo de acesso aos objetos mantidos nessa página. Para manter o estado da navegação entre páginas é necessário que exista um conjunto de atributos com a finalidade de manter a página atual (`numeroPagina`), o tamanho da página (quantidade de elementos que a página pode conter - `tamanhoPagina`), o número total de páginas que está armazenado no meio persistente (`totalPaginas`), uma lista com os objetos da página corrente (medicamentos) e uma referência para o objeto responsável por recuperar os dados do meio persistente (`pm`).

```
private int numeroPagina = -1;
private int tamanhoPagina;
private int totalPaginas = 0;
private List<Medicamento> medicamentos =
     new ArrayList<Medicamento>();
private PersistenciaMedicamento pm =
     new PersistenciaMedicamento();
```

As funcionalidades de navegação nas páginas normalmente resultam no acesso a persistência para recuperação dos dados e dessa forma, definimos um método com a finalidade de passar para o objeto de acesso a persistência qual o intervalo de objetos que necessitamos. Existem várias formas de implementar esse mecanismo, mas nesse caso, optamos por um mecanismo simples baseado em um número seqüencial que é dado a cada objeto armazenado na persistência, assim será definido um método em `PaginadorMedicamento` que receberá o número da página desejada e ele se encarregará da carga da página:

```
private List<Medicamento> carregaPagina(int numeroPagina) {
     int i = tamanhoPagina * numeroPagina;
     int f = i + tamanhoPagina;
     return pm.getMedicamentos(i, f);
}
```

Nesse exemplo acima é feito o cálculo do intervalo de objetos que devem ser recuperados e é feita a

requisição ao objeto `pm` para buscar esses `Medicamentos`.

Para que o cálculo acima possa ser feito é necessário que seja conhecido o tamanho da página (quantidade de objetos por página). Essa informação é passada na criação do objeto, como um parâmetro para o construtor:

```
public PaginadorMedicamento(int tamanhoPagina) {
     this.tamanhoPagina = tamanhoPagina;
     this.totalPaginas = (int)Math.ceil(
          (double)pm.getQuantidade() / (double)tamanhoPagina);
}
```

Observe que no próprio construtor já é feito o cálculo do número total de páginas.

Os métodos de navegação entre páginas são semelhantes, portanto apresentaremos somente o método que tem como função requisitar ao Paginador que ele vá para a última página:

```
public void ultimaPagina() {
     if (this.getPaginaCorrente() != totalPaginas) {
          this.setPaginaCorrente(totalPaginas);
          medicamentos.clear();
          medicamentos.addAll(
               carregaPagina(this.getPaginaCorrente()));
     }
}
```

Nesse caso acima, podemos observar que a cada vez que é requisitada a navegação, fazemos a atualização da página corrente, apagamos a lista com os objetos anteriores e requisitamos para o objeto de controle de persistência que faça a carga da nova página. Em implementações mais robustas, deve haver o controles para verificação de erros na carga ou eventuais tentativas de navegações em páginas inválidas.

Uma vez que a página foi carregada, podemos disponibilizar o acesso aos objetos dessa página através de um `Iterator` como é mostrado logo abaixo:

```
public Iterator getIterator() {
     return new MedicamentoIterator(this.medicamentos);
}
```

Nesse trecho de código acima criamos um objeto personalizado do `Iterator` para `Medicamento` e retornamos para o cliente. Uma outra forma de implementar esse método é fazer com que a cada carga da página essa já crie um objeto `MedicamentoIterator` e mantenha-o válido enquanto a página não for alterada. Caso a página seja alterada, os Iterators anteriores devem ser invalidados, não permitindo que os clientes continuem fazendo uso.

As classes `Iterator` e `IteratorMedicamento` são implementações simples do padrão Iterator [**GoF**].

A classe de acesso à persistência pode variar de acordo com as necessidades do sistema, mas no nosso caso ela contem dois métodos importantes:

```
public List<Medicamento> getMedicamentos(int inicio, int fim)
public int getQuantidade()
```

`getMedicamentos` retorna do meio persistente uma lista de medicamentos onde as suas chaves estejam no intervalo começando em `inicio` (inclusive) e `fim` (exclusive). `getQuantidade` retorna a quantidade total de objetos armazenado no meio persistente permitindo. `getQuantidade` é utilizada para que o Paginador consiga definir a quantidade de páginas disponível.

Por fim, a classe `Cliente` é um cliente qualquer do Paginador.

```
PaginadorMedicamento paginadorMedicamento =
     new PaginadorMedicamento(10);

while (paginadorMedicamento.temProxima()) {
     paginadorMedicamento.proximaPagina();
     Iterator it = paginadorMedicamento.getIterator();
     while (it.temProximo()) {
          System.out.println(it.proximo());
     }
}
```

No código acima definimos `paginadorMedicamento` como sendo o Paginador e através do dois laços `while` iteramos sobre as páginas e para cada página iteramos sobre suas coleções de medicamentos carregados apresentamos os na tela.

Usos Conhecidos

- Vários sistemas comerciais, distribuídos têm a necessidade de apresentar um conjunto de dados para seus usuários. Devido a questões de memória e tráfego de rede, é interessante fazer esse tráfego por páginas e o usuário tem controle dessa navegação. Vários sítios de compras pela Internet apresentam esse comportamento. Por exemplo, os sítios das lojas Americanas, Submarino e Livraria Saraiva permitem que o usuário faça pesquisa de seus produtos. Nessas pesquisas podem ser retornados vários produtos e permitindo que o usuário navegue nesses conjuntos por página e opções de navegação como próxima página a página anterior.
- Os sistemas operacionais trabalham com o conceito de memória virtual. Essa memória virtual é na verdade uma cópia em disco de um espaço de memória que continha dados. Esse gerenciamento pode ser feito utilizando o padrão Paginador que recupera e salva a páginas em disco garantindo o controle sobre a sua ordem.
- SCORM é um conjunto de padrões técnicos desenvolvido pelo Departamento de Defesa Americano que possibilita sistemas de aprendizados baseados na teia que encontrem, importem, compartilhem, reutilizem e exportem conteúdos de aprendizado de uma forma padrão. SCORM define o uso de objetos de aprendizado que podem conter vários recursos como textos, imagens e sons além de uma regra de navegação e uso desses recursos da maneira a propiciar o aprendizado. Esses objetos de aprendizado são executados em Sistemas de Gerenciamento de Aprendizado ou Sistema de Gerenciamento de Conteúdo de Aprendizado (em inglês referem-se às siglas: LMS – Learning Management Systems e LCMS – Learning Content Management Systems) sendo que esses sistemas devem obedecer as regras de navegação definidas nos objetos de

aprendizagem. Para executar as tarefas de controle do fluxo de navegação esses sistemas utilizam um Paginador que carrega os recursos na memória de acordo com a necessidade e esse Paginador também deve ser dotado de uma inteligência adicional para permitir que além da navegação seja feito também uma análise respeito da evolução no aprendizado do usuário, mudando o caminho de aprendizagem e conseqüentemente o fluxo de paginações.

Padrões Relacionados

- Iterator [**GoF**]: O padrão Paginador pode ser considerado como uma versão mais sofisticada do Iterator, pois ele incorpora mecanismos mais sofisticados de acesso à dados além de permitir que algoritmos de controle sejam incorporados aos mecanismos de navegação inclusive para manter vários estados referentes a essa navegação.

Referências

- **SCORM**
  ADL – Advancement Distributed Learning: http://www.adlnet.gov/index.cfm
  Randall House Associates: http://www.rhassociates.com/scorm.htm

- **LMS e LCMS**
  http://en.wikipedia.org/wiki/Learning_Management_System

- **GoF**
  E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.