Instituto Federal de São Paulo - IFSP
***Campus* de Hortolândia**
Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas
Linguagem de Programação I – LOG A2

Hortolândia, 11 de setembro de 2013.

Aulas 25 e 26 – Encapsulamento e Comunicação entre as classes

Na presente aula serão apresentados conceitos sobre encapsulamento, sua utilidade e como influencia o processo de comunicação entre as classes de um projeto. Paralelamente diversos aspectos relacionados serão citados, como organização em pacotes, métodos set/get, entre outros.

I. Professores

Nome do Professor	Turma	E-mail
Arthur A. B. Buccioli	A	arthurbuccioli@gmail.com
Leandro C. Ledel	B	leandro.ledel@gmail.com ledel@ifsp.edu.br

II. Encapsulamento

O conceito de encapsulamento está intimamente ligado ao conceito de abstração. É também um dos pilares da Programação Orientada a Objetos, daí a importância em conhecer seu funcionamento.

De uma maneira geral, o encapsulamento é responsável por definir permissões de acesso (leitura e escrita) dos diversos elementos utilizados na programação orientada a objetos. Assim, será possível especificar para cada classe, atributo e método um nível individual de visibilidade em relação aos outros elementos. E esse controle da visiblidade pode ser útil em diversas situações:

- Esconder detalhes de implementação – Muitas vezes os usuários de classes prontas não precisam ou não querem conhecer intimamente os detalhes de implementação, como a quantidade de variáveis utilizadas, metodologia envolvida, laços de repetição/seleção, etc... Ao invés disso, é interessante que a classe forneça acesso somente às funcionalidades a que ela se propõe, através de uma interface de utilização e um material de utilização. Essa interface consistirá em métodos e atributos acessíveis aos usuários. Internamente a classe poderá ter muitos outros métodos e atributos invisíveis ao usuário, mas os necessários à sua utilização estarão disponíveis.

  Podemos fazer uma analogia com um televisor. Apesar de existirem centenas e até milhares de componentes eletrônicos dentro do televisor, o usuário tem acesso somente à interface de operação do mesmo, proporcionada pelo controle remoto ou botões no próprio aparelho. O televisor em si é uma caixa fechada, inacessível para o usuário comum. E essa falta de acesso não prejudica em absoluto a utilização do aparelho, pois o usuário não precisa de conhecimentos avançados em eletrônica, somente o manual de utilização do aparelho.

  Voltando à programação, pode-se dizer que o ocultamento dos detalhes de implementação de classes simplifica a sua utilização e proporciona maior aplicabilidade à classe, permitindo que a mesma seja utilizada por diversos tipos de aplicações.

- Proteger a integridade dos dados – Erros de programação são comuns a programadores iniciantes ou mesmo experientes. O encapsulamento, além de ocultar os dados, fornece uma maneira eficaz de controlar o seu acesso e manipulação, através de métodos set/get.

  Em vez de permitir a manipulação direta das variáveis internas dos objetos(atributos), são criados métodos intermediários para realizar esses acessos. Esses métodos fazem

parte da interface da classe, sendo a única maneira de acessar os dados nos objetos, tanto para leitura quanto para escrita.

Os métodos de escrita são comumente chamados de *setters* e sua função é validar os dados recebidos antes de efetivamente alterar as variáveis às quais estão associados. Por exemplo, um método setPeso, projetado para atualizar os dados do peso de uma pessoa, deve rejeitar valores inferiores a 0 e também superiores a 200 ou 250 kg, pois seriam inválidos para a faixa existente de peso de uma pessoa. Essa rejeição pode ser feita colocando os valores válidos mais próximos do recebido ou pode acontecer em forma de mensagens de erro para o usuário, obrigando-o a preencher novamente o valor desejado. Desta forma as variáveis sempre conterão valores válidos, evitando anomalias ou erros de preenchimento acidentais.

Os métodos de acesso são chamados de *getters* e sua função é recuperar os dados contidos nas variáveis dos objetos de diversas formas. A mais simples delas é simplesmente retornar o valor contido na variável. Porém, em alguns casos é interessante formatar a saída, pois nem sempre o armazenamento dentro dos objetos é feito no mesmo formato que deve ser visualizado, como por exemplo as datas que podem estar no formato numérico e devem ser recuperadas em formato com barras (20/01/01). Dessa forma os métodos get trazem flexibilidade na recuperação das informações, sendo possível inclusive a existência de 2 ou mais métodos get para um mesmo atributo, recuperando-o de formas distintas.

Além das funcionalidades citadas anteriormente, os métodos set/get tem ainda outra vantagem. Como são os únicos a acessarem os dados encapsulados, se houver qualquer problema nesses dados, os únicos responsáveis serão os set/get, facilitando assim a busca e resolução dos problemas, que de outra forma poderiam estar localizados em qualquer outra classe ou parte do programa principal.

- Facilitar a organização dos projetos de software. Como a utilização das classes encapsuladas prevê o não-conhecimento do funcionamento interno das mesmas, mas sim suas funcionalidades, o organização e compreensão dos projetos fica mais fácil principalmente podem ser feitas quaisquer modificaçõesse olharmos mais superficialmente. Em algumas linguagens é possível inclusive organizar grupos de classes interrelacionadas de um projeto em um mesmo espaço (pacotes,pastas,...), facilitando a divisão do sistema em módulos.

III. Encapsulamento em Java

Para implementar de fato o encapsulamento, o Java utiliza modificadores de acesso, que atuam sobre as classes e também sobre seus elementos individuais (atributos e métodos). Para entendermos melhor o funcionamento de alguns detalhes desta forma de encapsulamento, é interessante antes entendermos alguns conceitos auxiliares.

Em Java, temos uma estrutura denominada pacote, que serve para organizar e agrupar classes relacionadas em um mesmo local. É algo bastante semelhante a uma pasta do sistema operacional, inclusive é assim que as classes acabam sendo organizadas no disco quando são colocadas em um pacote.

A sintaxe para se declarar um pacote é bastante simples:

```
package nomeDoPacote;

public class...
```

Como pôde ser visto, a declaração do pacote é a primeira coisa a ser feita em uma classe, antes inclusive dos imports. Por falar nisso, os imports nada mais fazem que indicar ao compilador o caminho para uma determinada classe contida em pacotes e subpacotes. Por exemplo:

```
import java.awt.color.ColorSpace;
```

Nessa linha, indicamos que a classe ColorSpace está contida no pacote java, dentro dos subpacotes awt e color. Também podemos importar todas as classes de um determinado pacote através da sintaxe:

```
import pacote.*;
```

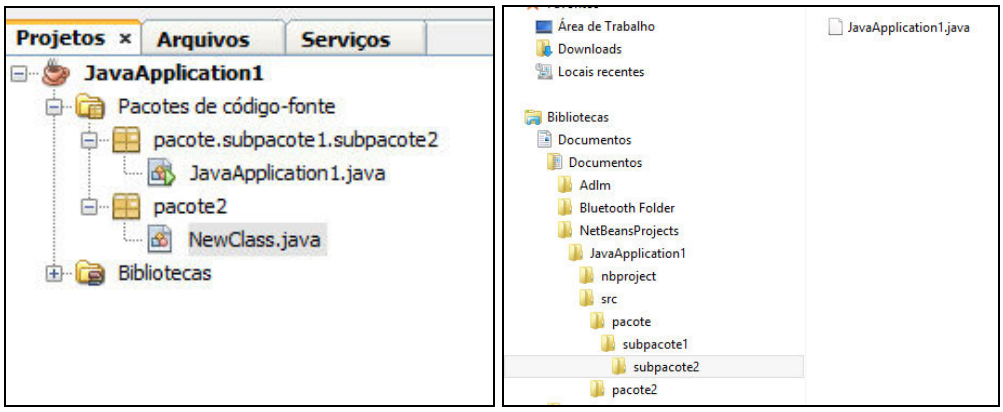
É importante ressaltar que se houver subpacotes, essa sintaxe não importará as classes deles, somente as que estiverem diretamente no pacote importado.

O nome do pacote deve ser sempre em letras minúsculas e seguir as mesmas regras de nomenclatura já especificadas anteriormente. Se quisermos declarar um subpacote, a sintaxe é a seguinte:

```
package pacote.nomeSubPacote;

public class...
```

Como foi falado, esta estrutura se reflete na maneira como o sistema organiza os arquivos do projeto. Veja o seguinte exemplo:



Assim, o projeto vai ficando organizado não só internamente mas também em sua estrutura de arquivos, permitindo um melhor controle sobre o mesmo. Não existe um limite definido de quantos subpacotes é possível criar, mas algumas versões de sistemas de arquivo podem ter problemas com estruturas muito profundas de subpastas.

Agora que já sabemos como funcionam os pacotes em Java. podemos continuar a falar sobre o encapsulamento, que inclusive utilizará o conceito de pacotes para ser implementado.

Em Java, modificadores de acesso determinam como outras classes podem acessar um determinado atributo ou método. Eles podem ser definidos em 2 níveis:

- Topo da Hierarquia(controle da classe) – modificadores *public* ou *package-private*(quando não houver definição explícita) – Quando uma classe é declarada com o modificador *public* ela pode ser acessada por qualquer classe existente. Quando uma classe é declarada sem modificador, automaticamente ela é reconhecida como *package-private*. Nesse modo, a classe só pode ser acessada por classes pertencentes ao seu pacote. O controle a nível de classe é interessante para um aspecto mais geral do projeto, determinando quais classes podem acessar outras, e com isso automaticamente também quais classes podem acessar as interfaces de outras (atributos e métodos).

- Por membro (controle em atributos e métodos) – modificadores public, private, protected e package private(quando não houver definição explícita) – O controle de acesso por membros funciona de maneira parecida ao controle por classe quando usando os mesmos modificadores, isto é, com modificador *public* os membros podem ser acessados por quaisquer classes e na ausência de modificador são considerados *package-private* podendo ser acessados dentro de suas classes e dentro de todas as classes em seus pacotes. Além desses dois modificadores existem também o *private* e o *protected* que estabelecem novas formas de controle de acesso. O modificador *private* permite o acesso dos membros apenas dentro da própria classe, sendo a restrição máxima existente. Esse tipo de restrição é o idealmente aplicado a todos atributos de uma classe, fornecendo oa métodos set/get para lidar com os seus valores ao invés de acesso direto. O modificador *protected* permite o acesso de forma similar ao *package-private* e, adicionalmente, a subclasses dentro de outros pacotes. esse tipo de encapsulamento em particular será melhor estudado quando falarmos sobre o assunto herança.

A tabela a seguir ilustra os níveis de acessibilidade dos modificadores de acesso.

Níveis de Acesso				
Modificador	**Classe**	**Pacote**	**Subclasse**	**Global**
Public	SIM	SIM	SIM	SIM
Protected	SIM	SIM	SIM	NÃO
Sem modificador(package-private)	SIM	SIM	NÃO	NÃO
Private	SIM	NÃO	NÃO	NÃO

Modificadores de acesso podem afetar suas classes de duas maneiras distintas:

- Quando você usa classes de terceiros em sua classe(como as classes da API Java) – Controlam a quais classes e membros internos sua classe poderá acessar.

- Quando você disponibiliza sua classe para terceiros – Você pode controlar de forma precisa a visibilidade da sua classe e seus membros internos, dando acesso somente aos itens que julgar necessários.

A sintaxe para inserir encapsulamento em uma classe é a seguinte:

```
encapsulamento NomeDaClasse{...}
```

Ex:

```
puclic Class Main{...}
```

Já para atributos a sintaxe é a seguinte:

```
encapsulamento tipo nome;
```

Ex:

```
private int contagem;
```

E Para métodos:

```
encapsulamento tipoDeRetorno nome(){}
```

Ex:

```
public int getContagem(){}
```

Finalizando, algumas dicas para boa utilização do encapsulamento e modificadores de acesso:

- Use sempre o nível de acesso mais restritivo possível para um determinado membro. Prefira o modificador *private* a menos que outros sejam necessários;

- Evite atributos com acesso *public*. Eles tendem a focar sua classe em uma implementação muito específica, limitando a sua flexibilidade em alterar o código posteriormente.