UML para Java

Christiano Vale

Este tutorial explica brevemente as características da Unified Modeling Language (UML) *fornecendo a notação convencional que aplicamos para ilustrar Projetos Orientados a Objetos.*

INTRODUÇÃO

Tendo em vista os rumores que giram em torno deste novo exame da Sun, SCJA – Sun Certified Java Associates, (que, aliás, eu fiz o exame.) onde o tema é cobrado e aproveitando a oportunidade de apresentar de maneira simplificada como ilustrar projetos de sistemas orientados a objetos resolvi escrever tutorial explica brevemente as características da *Unified Modeling Language(UML)* fornecendo a notação convencional que aplicamos para ilustrar este projetos. Embora ela não seja excessivamente complexa, podemos subestimar facilmente o poder das características que ela nos fornece. Para uma apresentação mais completa leia *The Unified Modeling Language User Guide* (Booch, Rumbaugh, and Jacobson 1999). Assimilando o uso das nomenclaturas e notações padrão, aprendemos a nos comunicar no nível de projetos, tornando-nos mais produtivos.

Classes

A figura abaixo aplica algumas características usadas para ilustrar classes:

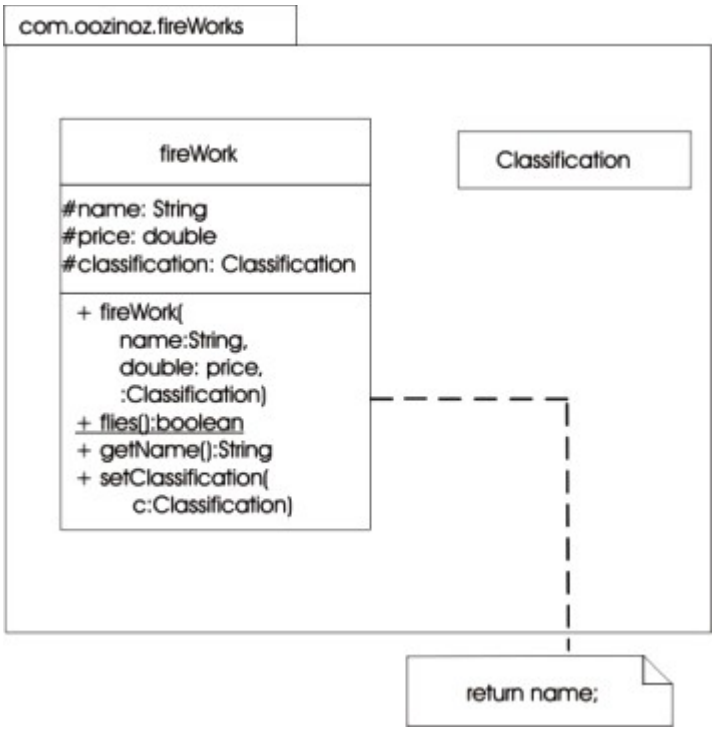

Figura 1 – O pacote *fireWorks* inclui as classes *fireWork* e *Classification*.

- Para indicar a existência de um pacote, coloque o nome dele em um retângulo alinhado à esquerda, com caixa maior, na qual podem conter: classes e interfaces. A figura 1 mostra o pacote *com.oozinoz.fireWorks*.
- A UML não exige que o diagrama mostre tudo a respeito do objeto retratado, como todo o conteúdo do pacote ou todos os métodos de uma classe.
- O nome da classe deve ser mostrado centralizado em um retângulo. (A figura 1 mostra 2 classes: *Classification* e *fireWork*).

- As variáveis de instância de juma classe devem ser mostradas em um retângulo abaixo do nome da classe. A classe *fireWork* possui as varáveis de instância *name, price, calssification*. Coloque dois-pontos depois do nome da variável e então o tipo da variável.
- Os métodos de uma classe podem ser mostrados em segundo retângulo, abaixo do nome dela. A classe *fireWork* possui um construtor, um método com o mesmo nome dela e possui pelo menos mais 3 métodos: *flies(), getName(), setClassification()*.
- Quando um método receber parâmetros, de forma geral devemos mostrá-los, com o método *setClassification()*.
- As variáveis nas assinaturas dos métodos de forma geral aparecem da seguinte forma: <nomeDaVariavel> : <tipoDaVariavel>. Podemos omitir ou abreviar o nome da variável se seu tipo implica o papel dela.
- Podemos indicar que um método ou uma variável de instância de uma classe estão protegidos (*protected*) colocando o sinal sustenido(#) na sua frente. O sinal mais(+) indica que o método ou a variável é do tipo *public* e o sinal menos(-) indica que o método ou a variável são do tipo *private*.
- Para indicar que uma variável de instância é estática – e tem escopo de classe – deixe-a sublinhada como o método *flies()*.
- Um retângulo com orelha pode ser usado para desenhar notas. O texto dessa nota pode conter: comentários, restrições, ou código. Uma linha tracejada é usada para anexar as notas a outros elementos do diagrama.

Relacionamentos entre classes

A figura abaixo apresenta algumas características usadas para ilustrar relacionamentos entre as classes:

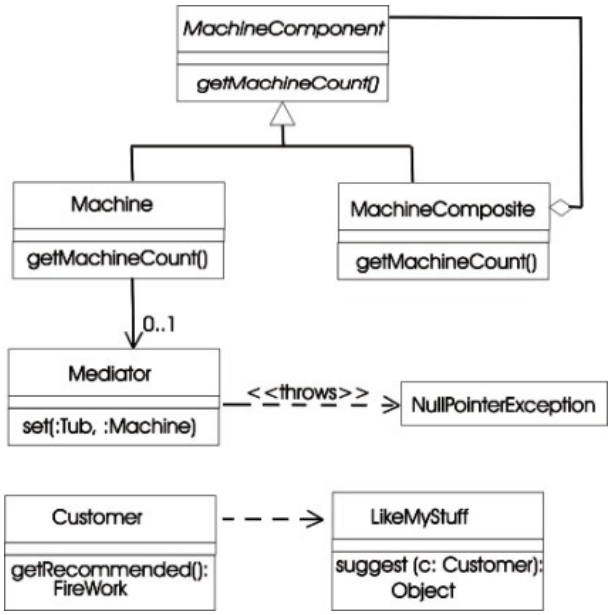

Figura 2 – Um objeto *MachineComposite* contém objetos *Machine* ou outros *composites*. A classe *Customer* depende da classe *LikeMyStuff* sem instanciá-la.

- Coloque o nome da classe ou método em itálico para indicar que a classe ou método são abstratos.

- Uma ponta de flecha fechada e oca é usada para apontar para a superclasse de uma classe. Podemos dizer que a classe *Machine* e *MachineComponent* herdam de *MachineComponent.*
- Utilize uma linha entre duas classes para indicar que as instâncias delas estão conectadas. Mais comumente, uma linha em um diagrama de classe significa que uma classe possui uma variável de instância que se refere à outra classe. A classe *MachineComposite*, por exemplo, utiliza uma variável *List* que contém referências a outros componentes de máquinas.
- Um losango é utilizado para mostrar que instâncias de uma classe contém uma coleção de instâncias de outra classe.
- Um ponta de flecha aberta indica navegabilidade. É utilizada para enfatizar que uma classe possui uma referência a outra que a classe apontada não possui uma referência de volta.
- Um indicador de multiplicidade , como 0..1, indica quantas conexões podem aparecer entre objetos. Um asterisco (*) é utilizado para indicar que 0 ou mais instâncias de um objeto de uma classe podem estar conectados a objetos de uma classe associada.
- Para mostrar que um método pode dispara uma exceção, utilize uma flecha tracejada apontado para a classe da exceção. Rotule esta flecha com <<*trows*>>.
- Utilize uma flecha tracejada entre classes para mostrar uma dependência que não usa referência de objeto. Por exemplo, a classe *Customer* utiliza um método estático do mecanismo de recomendação *likeMyStuff*.

Herança

A herança entre classe pode ser representada de 2 formas:

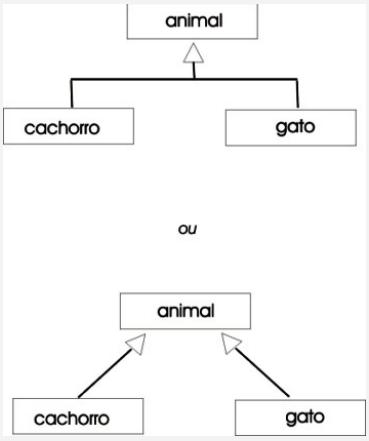

Figura 3 – Herança de classes

Os dois exemplos da figura 3 podem ser usados para representar herança. Isto representa:

```
public class cachorro extends animal
{
}


public class gato extends animal
{
}
```

Interfaces

A figura 4 mostra as características básicas para representar interfaces.

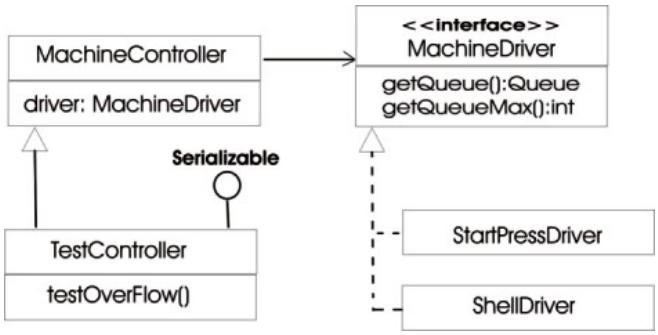

figura 4 – representação de interfaces

- Para desenhar uma interface colocamos o texto <<*interface*>> e o nome da interface em um retângulo.
- Podemos utilizar uma linha tracejada e com um ponta de flecha vazada para mostrar que uma classe implementa a interface.
- Outra forma de mostrar que uma classe implementa uma interface é mostrar uma linha com um círculo, como um "pirulito", e o nome dessa interface. (*TesteController implements Serializable)*.
- As interfaces e seus métodos sempre são abstratos em Java. Estranhamente, ambos não aprecem em itálico, como ocorre com as classes abstratas e os métodos abstratos em classes.

Objetos

Um diagrama de objetos ilustra instâncias especificas de classes, como mostra a figura 5.

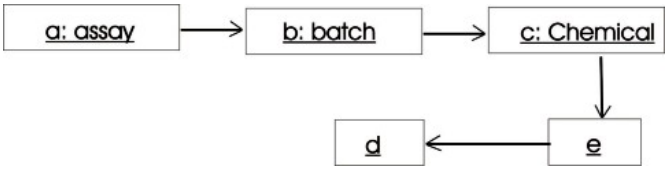

Figura 5 – representação de objetos

- Podemos mostrar um objeto fornecendo seu nome e tipo, separados por dois pontos. Opcionalmente, podemos mostrar apenas o nome, ou apenas o dois pontos e o tipo.Seu nome e/ou tipo devem ser sublinhados.
- Utilize uma linha entre dois objetos para mostrar que um objeto tem referencia ao outro. Podemos utilizar uma ponta de flecha aberta para enfatizar a direção da referência.

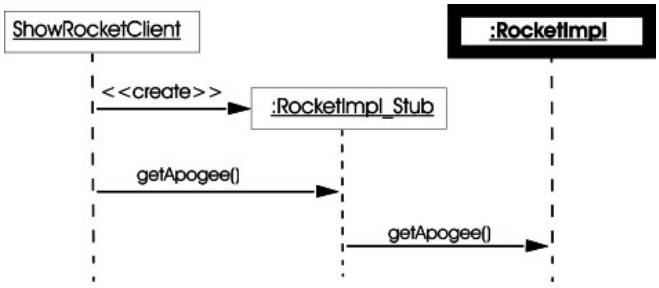

Figura 6 – Diagrama de seqüência que mostra uma sucessão de Chamadas de métodos.

- Podemos mostrar uma seqüência de objetos chamando métodos de outros objetos, como mostra a figura 6. A ordem das chamadas de métodos é de cima para baixo, e as linhas tracejadas indicam a existência do objeto em relação ao tempo.
- O estereótipo *<<create>>* é usado para mostrar que um objeto cria outro objeto.
- Uma moldura em negrito em volta de um objeto é usado para mostrar que ele está ativo em uma outra *thread*, processo ou outro computador. A figura 6 representa uma chamada remota.

Estados

Um diagrama de estados é mostra na figura 7.

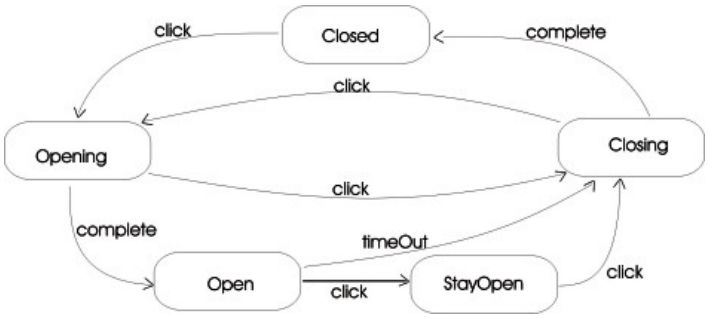

Figura 7 – Um diagrama de estados mostra as transições de estado para estado

- Mostre um estado em um retângulo com cantos arredondados.
- Mostre as transições de estados com flechas abertas.
- Um diagrama de estado não precisa mapear diretamente para um diagrama de classe ou de objeto, embora possamos arranjar uma transição direta.

Conclusão

Programadores sempre se deparam em situações onde os desafios possuem soluções em código, no entanto, muitos desses desafios solicitam que você faça um diagrama mostrando como as classes, pacotes e outros elementos se relacionam.

Christiano Vale (*christianovale@gmail.com*) é programador Java e trabalha na empresa LUCRA – CADASTROS E SERVIÇOS (Correspondente Bancário).

Referências Bibliográficas

METSKER, Steven John. 2004. Padrões de Projeto em Java. Bookman.
DEITEL, Deitel. 2003. Java – Como Programar, 4ª Edição. Bookman.