VOLTAR

Threads em JAVA e o Ambiente Multithreading

Apresentar os Threads em JAVA e um exemplo de programa Mono e Multithreading

NESTE TÓPICO

Revisão de Conceitos sobre Orientação a Objetos

A orientação a objetos (OO) foi criada como um novo paradigma no desenvolvimento de sistemas de software, pois apresenta uma proposta de reuso de código, facilidade de manutenção e redução no tempo e custo de programação. A Figura 1 a seguir, mostra o histórico no surgimento das linguagens de programação orientadas a objeto.

A revisão dos conceitos sobre orientação a objetos tem como objetivo rever e alinhar o entendimento sobre o tema, pois a partir deste tópico serão discutidos alguns exemplos de programas que realizam tarefas nos modelos monotarefa, multitarefa, monothreading, mutithreading, entre outros. Para isso, utilizaremos a linguagem de programação JAVA para apresentar estes e outros exemplos. Assim, os conceitos básicos sobre OO serão discutidos a seguir:

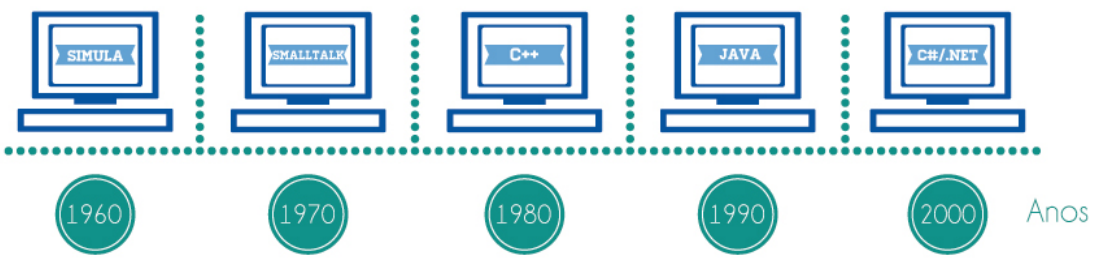

Figura 1 - Surgimento e a evolução das linguagens orientadas a objeto

Objetos

Quando fazemos uma análise ou programação orientada a objetos, partimos do princípio que os objetos representam melhor o mundo em que vivemos. Assim, o entendimento sobre o que está sendo projetado e construído no software ficará mais fácil de compreender e dar a manutenção no futuro. Assim como na vida real, os objetos contidos na análise ou programação orientada a objetos possuem características e comportamentos que os representam no contexto do software. Quando são instanciados, os objetos assumem as suas responsabilidades através de suas características e comportamentos. Este processo de definição de entidades do mundo real é denominado por Abstração.

Classes

As classes representam as estruturas dos objetos simulares e como estes objetos se relacionam dentro da lógica do software. Uma classe representa uma classificação ou categorização de um tema, e os objetos são elementos desta categoria. Os elementos principais que compõe uma classe são:

- **Nome**: O nome do objeto representa o nome dado ao objeto na análise ou instanciado durante a programação. Exemplos: Cliente, Contratos, Fornecedores, Produtos.
- **Atributos**: Os atributos do objeto representam as características ou as propriedades do objeto. Exemplo de atributos do objeto Cliente: nome_cliente, idade_cliente, sexo_cliente. Estes atributos possuem tipos de dados que definem o formato dos valores associados à eles, tais como inteiro (int), data (date), texto (char).
- **Métodos**: Os métodos do objeto, também conhecidos como operações, representam os comportamentos ou ações que o objeto executa durante o seu ciclo de vida.
    - Exemplos de métodos:
        - Criar_cliente()
        - Consultar_cliente()
        - Alterar_cliente()
        - Excluir_cliente()
    - Os métodos podem retornar valores ou não e possuírem valores de entrada ou não. Isso depende muito do tipo de implementação que está sendo feita. Exemplos:
        - void Consultar_cliente(int cpf)
        - bool Excluir_cliente(int cpf)
- **Visibilidade**: A visibilidade do atributo ou método, determina se a característica ou comportamento estará visível para outra classe ou não. Assim, certas formas de visibilidade não são permitidas dependendo do tipo de relacionamento que existir entre as classes. A visibilidade determina se o elemento da modelagem ao qual se refere pode ser visto fora de seu espaço de nome ou domínio, ou seja, a visibilidade de um elemento do modelo

define se ele poderá ser referenciado por um elemento fora de seu espaço de nome. A visibilidade do atributo ou método são definidas a seguir:

- Pública (*Public*) +: Qualquer classe externa com relacionamento com a classe que possui a característica, será capaz de usar.
- Privada (*Private*) -: Somente a classe dona da característica poderá visualizar.
- Protegida (*Protected*) #: Qualquer classe que seja descendente da classe cujo a característica esteja protegida, poderá visualizar.

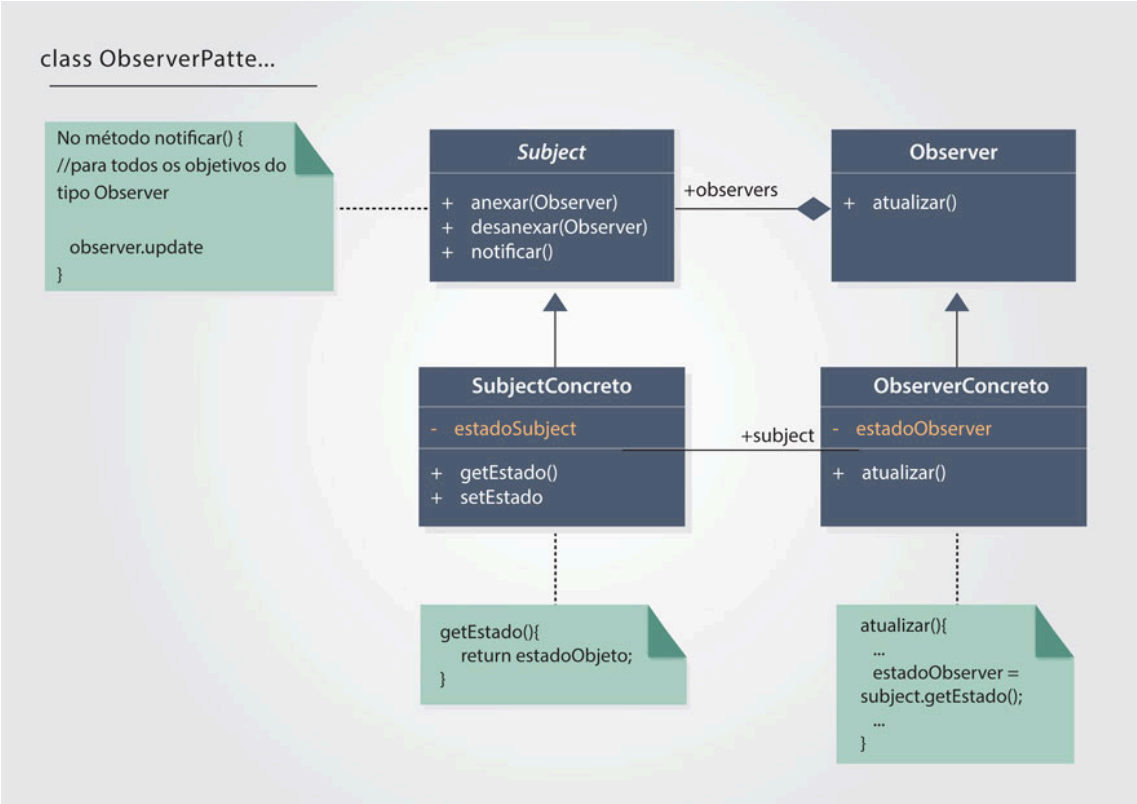

Figura 2 - Exemplo de diagrama de classes e seus relacionamentos

Encapsulamento

O encapsulamento é realizado pela visibilidade e utilizada através da abstração, ou seja, permite expor somente as características que são necessárias para a cumprimento das responsabilidades de um elemento de modelagem.

Instanciação

A instância ou ocorrência de um objeto representa a criação de um objeto que assume suas responsabilidades durante a execução do software. Em outras palavras, a instanciação é a criação de um novo item do conjunto representado por uma classe, assim como os demais itens (objetos) que possuem as mesmas características e comportamentos, e foram instanciados anteriormente. No momento da instanciação do objeto a ser criado, o programador informa um nome ao objeto e valores que definem as suas características ou atributos.

Herança

A herança é um comportamento que acontece na relação entre duas classes. Também conhecida por relacionamento entre a superclasse e a subclasse ou classe-mãe e classe-filha. Este relacionamento define que uma subclasse poderá herdar todas características e comportamentos da superclasse. De maneira mais didática, a herança nada mais é do que o reaproveitamento de atributos e dos métodos para agilizar o tempo e reduzir o custo de desenvolvimento e manutenção de software. No tópico **Especialização** logo a seguir, outros detalhes serão apresentados.

Abstração

Uma vez que um objeto é a representação de algo do mundo real (como vimos anteriormente), as suas características descritas no modelo determinam o nível de abstração que realizamos na análise. De maneira mais simples, representar uma entidade do mundo real em uma classe, significa que conseguimos abstrair corretamente as características ou propriedades necessárias para representar o objeto na análise. Por exemplo, podemos fazer a abstração de objetos do mundo real tais como carro, pessoa e livro da seguinte forma:

Objeto	Características (atributos)	Ações (métodos)
Carro	Cor, motorização, combustível, câmbio, tipo	Ligar, desligar, dar ré, virar à esquerda, virar à direita
Pessoa	Nome, idade, sexo, cor, altura, peso	Falar, andar, dormir, acordar, correr, sorrir
Livro Digital	Título, autor, tema, quantidade de páginas, editora, ISBN	Abrir, fechar, virar página, retroceder página, marcar página

Tabela 1 - Exemplo de abstração de objetos do mundo real

Polimorfismo

Antes de definir polimorfismo, é importante destacar que este conceito está intrinsecamente associado à herança. O polimorfismo é um princípio em orientação a objetos que permite que dois ou mais objetos diferentes e derivados da mesma superclasse, invoquem métodos com a mesma assinatura, mas que respondam ou tenham um comportamento completamente diferente. É importante lembrar que as subclasses podem implementar os métodos com a mesma nomenclatura (nome), mas com implementações diferentes. Dessa forma, o software deverá verificar se o método invocado pertence a subclasse ou herdado da superclasse.

De maneira mais didática, imagine uma superclasse Funcionário e as subclasses GerenteVendas e Vendedor. Na superclasse Funcionário existe um método denominado por Comissão. Como as subclasses GerenteVendas e Vendedor são derivadas da superclasse, elas podem invocar o método

Comissão por herança. Se as subclasses se comportarem de maneiras diferentes à mesma assinatura do método Comissão, temos um caso de polimorfismo.

Associação

A associação é um relacionamento estrutural entre os objetos de uma classe. Apenas representa a ligação entre duas classes, onde permite navegar de um objeto de classe A para o objeto ou objetos da classe B e vice-versa.

A linha sólida representa a associação simples entre as classes.

Multiplicidade

A multiplicidade é um tipo de relacionamento estrutural entre os objetos. Desta forma, é importante determinar a quantidade de objetos que podem se associar a instância de um ou mais objetos. Esta quantidade é denominada por multiplicidade e é expressada num intervalo de valores conforme abaixo:

- 0..* (zero ou muitos)
- 1..* (um ou muitos)
- 0..1 (zero ou um)
- 1 (um)
- * (muitos)

A linha sólida com os valores representa uma associação por multiplicidade entre as classes.

Inclusão

O relacionamento de inclusão (uso) ou *include (used)* representa que uma classe inclui a outra com objetivo de fatorar os comportamentos comuns ou sempre utilizados. É uma fatoração feita na programação e em sistemas operacionais (como o GNU-Linux) através de subprogramas. Quando existe este relacionamento, há uma indicação de obrigatoriedade, ou seja, quando um caso de uso A é executado, o caso de uso B deverá ser obrigatoriamente executado. Por exemplo, para ter acesso "Portal da Empresa", o usuário deverá "Fazer o Login" obrigatoriamente.

A linha tracejada contendo uma seta aberta e o estereótipo *include* representa um caso de inclusão.

Extensão

O relacionamento de extensão ou *extend* representa que uma classe estende a outra com objetivo de fatorar o comportamento incomum ou não-padrão. É um tipo de relacionamento com cenários opcionais ou quando uma condição específica for satisfeita. Isto é, trata-se de eventos que não

ocorrem com tanta frequência. Por exemplo, os "Relatórios Gerenciais" são rotineiramente solicitados pelo usuário, no entanto, os "Relatórios Gerenciais de BH" são pouco solicitados ou somente em casos específicos.

A linha tracejada contendo uma seta aberta e o estereótipo *extend* representa um caso de inclusão.

Agregação

O relacionamento de agregação é um tipo de associação entre duas classes. Neste relacionamento, uma classe representa o todo e outra a parte ou também como todo-parte, ou seja, a parte agrega o todo. Este relacionamento é apenas conceitual, logo seu propósito é representar o todo e a parte. Assim, a alteração de um objeto da classe não altera o tempo ou ciclo de vida do todo e suas partes.

A linha sólida é uma associação simples com um diamante sem cor na extremidade da classe que representa o todo.

Composição

O relacionamento de composição é um tipo de associação entre duas classes. Dessa forma, representa um relacionamento semântico entre os objetos com propriedade e tempo de vida bem claros. Assim, o relacionamento de composição com as multiplicidades bem definidas, as alterações das partes afetam o todo e a morte do todo causará automaticamente a morte das partes. Nada impede das partes serem removidas antes que o todo seja morto.

A linha sólida é uma associação simples com um diamante preto na extremidade da classe que representa o todo.

Especialização (Herança)

O relacionamento de especialização é um tipo de relacionamento de generalização e especialização, onde a classe-mãe representa a classe genérica e as classes-filhas, representam as classes especialistas. Assim, é importante lembrar que este relacionamento acontece quando o objetivo é que as classes-filhas herdam atributos (características) e operações (métodos) da classe-mãe.

Para não haver problemas com outras formas de representação, além de especificar este relacionamento por generalização/especialização e classe-mãe/classe-filha, também podemos encontrar como super-classe/sub-classe.

A linha sólida é uma associação de especialização com uma seta fechada na extremidade da super-classe.

Dependência

O relacionamento de dependência é um tipo de relacionamento de utilização determinando as modificações na especificação de um item. O relacionamento de dependência ocorre quando uma classe usa outra classe como argumento na assinatura de uma operação.

A dependência é representada graficamente com uma linha tracejada.

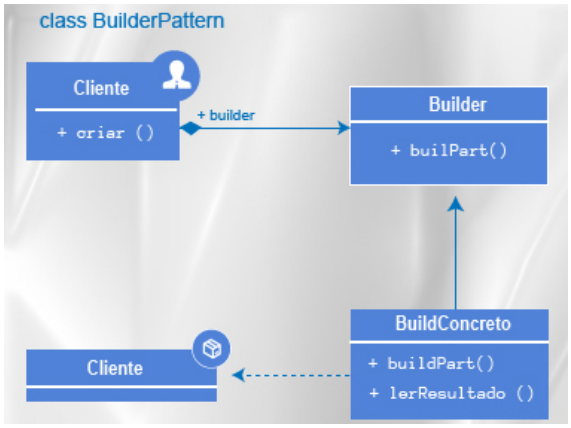

Figura 3 - Exemplo de diagrama de classes e outros relacionamentos

Aplicativo Monotarefa

O código-fonte na linguagem de programação JAVA a seguir representa um exemplo simples de um aplicativo monotarefa. Observe que a classe MinhaTarefa foi criada para conter a estrutura das tarefas que serão executadas e a classe Programa é o aplicativo que irá iniciar e instanciar a tafefa. Veja que a classe Programa somente instancia um objeto MinhaTarefa, logo ele é monotarefa.

```
1.  public class MinhaTarefa{
2.
3.          private String tarefa;
4.          public String nome;
5.
6.          public MinhaTarefa (String tarefa, String nome) {
7.                this.tarefa = tarefa;
8.                this.nome = nome;
9.          }
10.
11.        public void run(){
12.              System.out.println("O nome da tarefa é: " + nome);
13.              System.out.println("Tarefa executada: " + tarefa);
14.        }
15. }
16.
17.
18. public class Programa{
19.       public static void main (String[] args) {
20.                MinhaTarefa tarefa1 = new MinhaTarefa ("Listar","Listar Produtos");
21.                tarefa1.run();
22.       }
23. }
```

Quiz

Exercício

INICIAR

Aplicativo Multitarefa

O código-fonte na linguagem de programação JAVA a seguir representa um exemplo simples de um aplicativo Multitarefa. Assim como no exemplo anterior, a estrutura da classe MinhaTarefa é a mesma, a alteração está na classe Programa, pois esta instancia três tarefas denominadas por tarefa1, tarefa2 e tarefa3. Desta forma, o programa é considerado como multitarefa.

A questão é que, no multitarefa, cada tarefa será executada (tafefa1.run()) na sequência que foram colocadas no código. Veremos no próximo tópico que o comportamento de programas multithreading é bem diferente e mais adequado para modelos de sistemas distribuídos que utilizam processamento paralelo.

```
1.  public class MinhaTarefa{
2.
3.          private String tarefa;
4.          private String nome;
5.
6.          public MinhaTarefa (String tarefa, String nome) {
7.                this.tarefa = tarefa;
8.                this.nome = nome;
9.          }
10.
11.        public void run(){
12.              System.out.println("O nome da tarefa é: " + nome);
13.              System.out.println("Tarefa executada: " + tarefa);
14.        }
15. }
16.
17.
18. public class Programa{
19.       public static void main (String[] args) {
20.                MinhaTarefa tarefa1 = new MinhaTarefa ("Listar","Listar Produtos");
21.                MinhaTarefa tarefa2 = new MinhaTarefa ("Apagar","Apagar Produtos");
22.                  MinhaTarefa tarefa3 = new MinhaTarefa ("Atualizar","Atualizar Produto
    s");
23.
24.                tarefa1.run();
25.                tarefa2.run();
26.                tarefa3.run();
27.       }
28. }
```

Quiz

Exercício

Threads em JAVA e o Ambiente Multithreading

INICIAR

Referências

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML - Guia do Usuário: Tradução da Segunda Edição**. Elsevier Brasil, 2000.

TANENBAUM, Andrew S.; WOODHULL, Albert S. **Sistemas Operacionais: Projetjos e Implementação**. Bookman, 2008.

LEA, Douglas. **Concurrent programming in Java: design principles and patterns**. Addison-Wesley Professional, 2000.

Avalie este tópico

Portal Uninove
(http://www.uninove.br)
Mapa do Site