MAC 413 – Tópicos de Programação Orientada a Objetos

Atividade - Padrão de Projeto

Regis de Abreu Barbosa
Nº USP: 3135701

Rodrigo Mendes Leme
Nº USP: 3151151

Nome

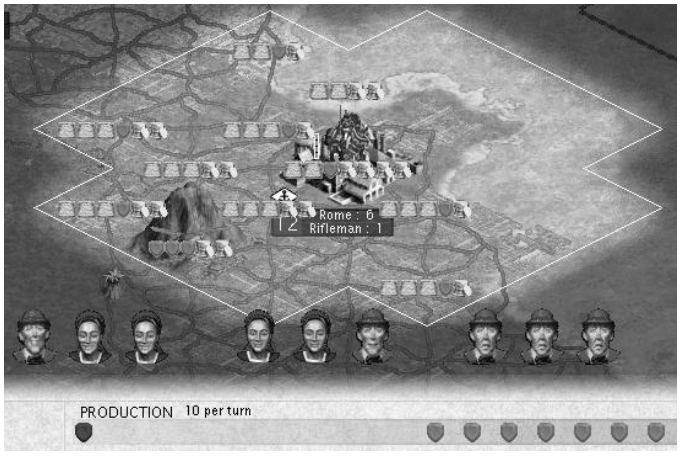
Controlador de Pessoas

Objetivo

Prover uma estrutura para representar localidades (cidades, colônias, planetas, bases, etc.) e sua relação com pessoas (soldados, diplomatas, colonos, cientistas, etc) em jogos de estratégia para computador.

Motivação

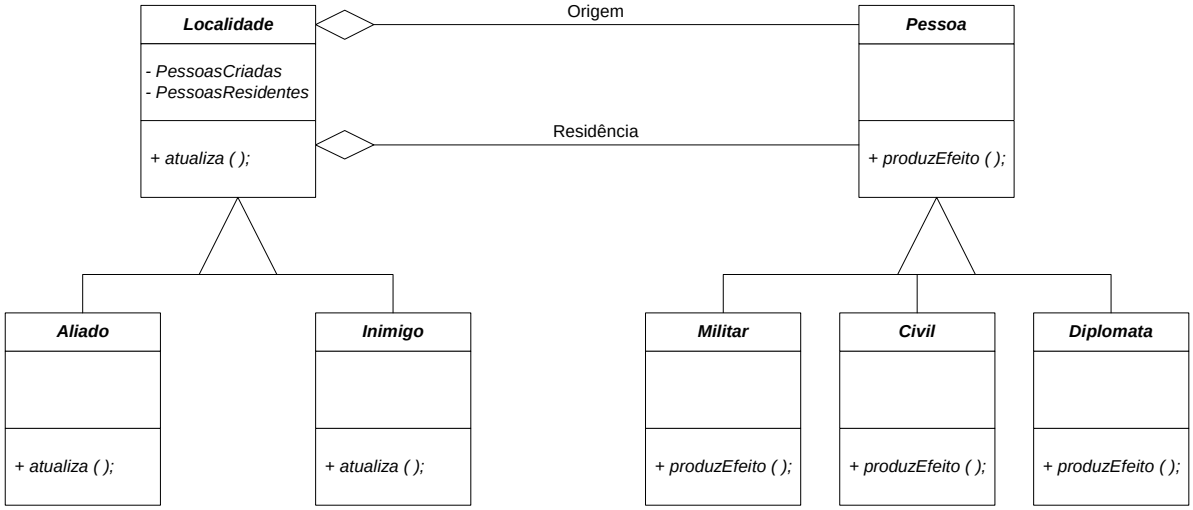
Em diversos jogos de estratégia, existem cidades ou colônias onde pessoas podem ser alocadas ou criadas, existindo, portanto, uma relação de dependência entre ambos. Um exemplo é o jogo *Civilization 3*, conforme as figuras:



Na figura à esquerda, pode-se ver os cidadãos criados por aquela cidade e como os mesmos influenciam a produção, taxa de desperdício, corrupção, capacidade defensiva, entre outros atributos. Na figura da direita, um mapa geral de jogo, pode-se ver pessoas criadas em uma cidade deslocando-se para outras, onde produzirão alguns efeitos como diminuição da infelicidade do povo daquela cidade, aumento da capacidade defensiva, etc. É importante frisar que isso pode ser feito tanto pelo jogador e seus aliados quanto por potências inimigas.

Em aplicações como esta, temos que as ações e comportamento das pessoas produzem efeitos em si mesmas e alteram, também, o estado das localidades relacionadas. Para que as pessoas possam alterar o estado de uma localidade, seria necessário que elas conhecessem e levassem em consideração os efeitos produzidos pelas demais pessoas relacionadas à localidade. Além disso, cada pessoa poderia determinar o estado de localidade de acordo com o seu interesse, tornando inconsistente a definição deste estado, pois diversas pessoas poderiam gerar estados diferentes para a mesma localidade.

A solução mais apropriada não deve, portanto, permitir que as pessoas alterem o estado de uma localidade. A própria localidade deve ser responsável por verificar o seu estado a partir de suas propriedades e dos efeitos produzidos pelas ações das pessoas.

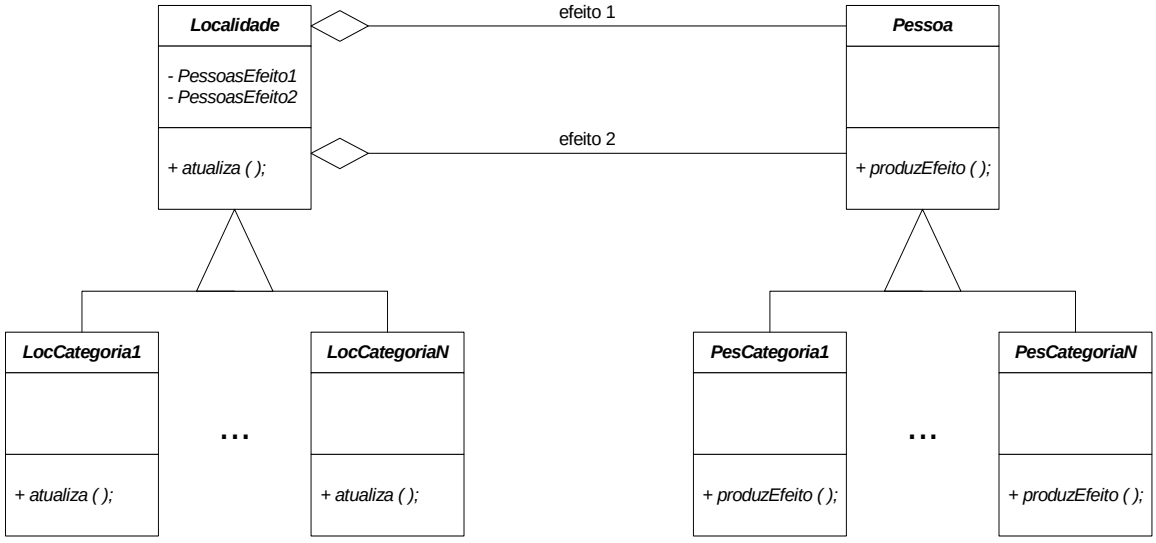


Aplicabilidade

Use o padrão quando:

- houver uma relação de dependência entre pessoas e localidades;
- o comportamento das pessoas afetam o estado das localidades relacionadas a estas pessoas;
- as pessoas não podem alterar o estado da localidade diretamente, a não ser pelos efeitos de suas próprias ações.

Estrutura



Participantes

- Localidade: define o comportamento básico de qualquer tipo de localidade e mantém referências para objetos da classe Pessoa.
- LocCategoria's: especializações de Localidade que refinam comportamentos específicos de cada categoria de localidade.
- Pessoa: define o comportamento básico de qualquer tipo de pessoa.
- PesCategoria: especializações de Pessoa que refinam comportamentos específicos de cada categoria de pessoa.

Colaborações

Através de suas referências para objetos Pessoa, uma instância de Localidade tem seu estado alterado de acordo com os efeitos produzidos pelos objetos Pessoa.

Consequências

O padrão:

1. **Aumenta o encapsulamento do *design***: a classe Pessoa, por não ter referências para Localidade, tem seus efeitos restritos a ela própria, não podendo alterar o estado de localidade diretamente. A própria classe Localidade é responsável por alterar seu estado, bastando consultar suas referências para Pessoas.
2. **Controle mais refinado sobre o efeito das Pessoas**: como Localidade tem dois tipos de referências para instâncias de Pessoa, um para Pessoas criadas por ela e outro para as Pessoas presentes nela no momento, o efeito que cada objeto Pessoa tem sobre Localidade é mais facilmente controlado.
3. **Facilidade para adicionar efeitos**: caso queira-se acrescentar um novo tipo de efeito da Pessoa sobre Localidade, basta apenas fazer uma nova agregação entre ambos. Por exemplo: em um jogo, pode-se querer criar o efeito de uma Pessoa numa Localidade inimiga (sabotagem, terrorismo). Criando-se um novo tipo de agregação, este novo efeito é facilmente acrescentado ao *design*.

Implementação

Considere os seguintes tópicos ao implementar o padrão:

1. **Instanciação de classes**: no contexto de aplicabilidade deste padrão, as classes Localidade e Pessoa devem ser abstratas. Ambas podem conter alguns métodos com implementação, mas outros só podem ser implementados pelas subclasses. Por isso, apenas as subclasses devem ser instanciadas.
2. **Definindo os efeitos**: em Localidade, cria-se um tipo de referência para cada efeito que os objetos Pessoa podem ter sobre ela.

Exemplo de código

Tomando como exemplo o caso citado na Motivação (Civilization 3), no qual temos duas localidades: uma aliada e outra inimiga, e temos diversos tipos de pessoas, entre elas os Militares.

O seguinte trecho de código mostra como instâncias das subclasses de Pessoa afetam o estado das subclasses de Localidade. Para tal, vejamos a classe Localidade. Ela define os dois tipos de efeito que as pessoas podem ter nas localidades desse jogo:

```
import java.util.*;

abstract public class Localidade
{
  private ArrayList pessoasCriadas;
  private ArrayList pessoasResidentes;

  abstract public void atualiza();
  //...
}
```

Imaginemos, agora, uma subclasse de Localidade, onde os efeitos serão aplicados:

```
import java.util.*;

public class Aliado extends Localidade
{
  public void atualiza()
  {
    for (Iterator i = pessoasCriadas.iterator(); i.hasNext();)
      ((Pessoa) i.next()).produzEfeito();
    for (Iterator i = pessoasResidentes.iterator(); i.hasNext();)
      ((Pessoa) i.next()).produzEfeito();
  }
  //...
}
```

Podemos ver que a classe Aliado delega para os seus objetos das subclasses de Pessoa a responsabilidade de produzir os efeitos em sua localidade. Estes efeitos podem ser então manipulados pelo Aliado de acordo com a classificação das pessoas que produziram o efeito.

A subclasse Inimigo seria análoga à subclasse Aliado.

Em seguida, temos a classe Pessoa, que define o método produzEfeito.

```
abstract public class Pessoa
{
  public void produzEfeito();
  //...
}
```

E, então, podemos estender a classe Pessoa para os diversos tipos de pessoa do jogo, cada um implementando o método produzEfeito de acordo com a sua atuação. Vejamos, por exemplo, a classe Militar:

```
public class Militar extends Pessoa
{
  public void produzEfeito()
  {
    //...
  }
}
```

O Militar, numa das possíveis possibilidades permitidas pelo jogo, poderia aumentar a capacidade defensiva da localidade em questão. Também poderia gerar um custo para sua manutenção.

Usos conhecidos

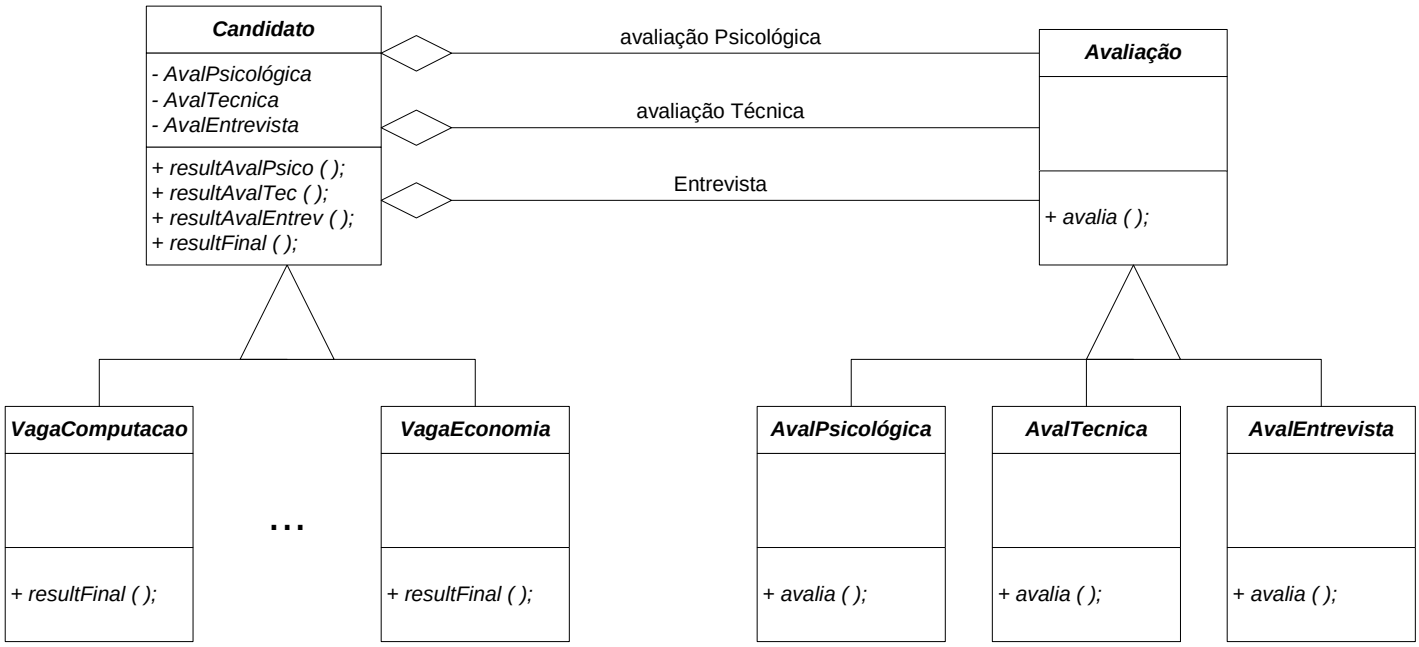
1. Jogos de estratégia, como a série Civilization, Colonization, Command & Conquer, Dune, Age of Empires, etc.

2. Sistema de avaliação de candidatos em um processo seletivo:

(2) Mostra que este padrão pode ser aplicado num domínio completamente diferente: um sistema composto de candidatos a um processo seletivo e diversas etapas de avaliação. Os candidatos poderiam ser avaliados sob diferentes aspectos: psicológico, conhecimento técnico, formação acadêmica, etc. Os candidatos podem, também, estar concorrendo a vagas de diferentes departamentos, o que altera os seus requisitos quanto à avaliação.

Um avaliador não pode determinar o resultado final da avaliação de um candidato. Este resultado é produzido a partir da combinação de avaliações dos diversos avaliadores do processo sob diferentes aspectos.

O diagrama UML seria o seguinte:



Para fazer uma avaliação psicológica, a partir do objeto Candidato correspondente chama-se o método *resultAval*, que irá delegar esta tarefa para um objeto AvalPsicológica. Este objeto será incorporado aos atributos do candidato, e terá os resultados obtidos na avaliação psicológica (pontuação, comentários, etc.). Analogamente, faz-se às demais avaliações.

Feita todas as avaliações, o método *resultFinal* será responsável por produzir um resultado final a partir dos resultados produzidos pelas demais avaliações. Ganha-se em flexibilidade.