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.

Civilization 3 é um exemplo: trata-se de um jogo de estratégia baseado em turnos, ou seja, cada jogador executa suas jogadas em seqüência ao jogador anterior (não existem jogadas de dois jogadores ocorrendo simultaneamente).

O objetivo de *Civilization 3* é criar e desenvolver uma civilização ao longo da História. Quando alguém deseja jogar, deve fazer algumas escolhas iniciais: qual será sua civilização (romanos, gregos, alemães, japoneses, etc), o mapa territorial do jogo e quantas outras civilizações haverá (estas outras são controladas pelo computador; o número máximo de civilizações permitidas é 8).

O jogo começa com o jogador humano controlando um grupo de pessoas, no ano 4000 AC. Com esse grupo, ele pode fundar sua primeira cidade. A partir daí, ele deverá desenvolver sua civilização, através de expansão territorial (fundação de novas cidades ou conquista militar), exploração, desenvolvimento interno (construção de estradas, áreas agrícolas/mineradoras e benfeitorias nas cidades), diplomacia com outras civilizações, pesquisa científica e tecnológica, comércio exterior, construção de unidades militares, entre outros. Todos esses fatores são controlados por regras bem definidas em *Civilization 3*. É importante dizer que as outras civilizações também procurarão se desenvolver, num momento cooperando com o jogador humano, num outro momento em conflito com ele.

O território do jogo é outro ponto importante de *Civilization 3*. Esse território representa o mundo virtual onde as civilizações desenvolvem-se. Ele é implementado como uma matriz bidimensional de terrenos, dos quais existem diversos tipos: pradaria, planície, floresta, selva, oceano, tundra, etc. Cada um possui propriedades diferentes, que podem ser exploradas pelos jogadores.

Para realizar essa exploração, o jogador precisa de pessoas que o façam. As pessoas são outra característica fundamental de *Civilization 3*. Existem diversas categorias, dentre as quais destacamos:

- Civil: é o tipo de pessoa mais comum, a base de todas as civilizações do jogo. São os civis os responsáveis pela produção de alimentos, de bens e de arrecadação fiscal, através da alocação dos mesmos em terrenos. Mas eles não possuem efeitos apenas positivos. Eles também geram uma taxa de desperdício e outra de corrupção, taxas estas que obedecem às regras de *Civilization 3* (desperdício está

Figura 1: território do jogo.

relacionado à produção de bens, uma porcentagem que é perdida; corrupção está relacionada à arrecadação fiscal, uma parte dela que é desviada). Os civis sempre estão vinculados à cidade de origem.

- Militar: os militares são os responsáveis pela defesa e expansão territorial de sua civilização. Podem atacar militares, matar civis de outras civilizações e saquear cidades inimigas. Cada militar possui três atributos importantes: capacidade ofensiva, capacidade defensiva e mobilidade. Os dois primeiros são utilizados em combate; o último, quando o militar está em movimento. Um efeito interessante que os militares possuem é o de diminuir as taxas de desperdício e de corrupção de uma localidade de sua civilização, caso estejam nessa localidade. Outro efeito é que cada militar possui é uma taxa de manutenção, cobrada da cidade que criou o militar. A próxima figura mostra exemplos de militares.

- Coletor de impostos: uma pessoa diferente de todas as outras do jogo, ela, diretamente, não produz nada. Seu efeito é provocar o aumento de arrecadação fiscal numa cidade. Logo, também está sempre vinculada à cidade de origem.

Civis e coletores de impostos, conforme dito, atuam nas cidades em que foram criados (ou localidades, em *Civilization 3* os termos são intercambiáveis). As cidades são os lugares mais importantes das civilizações, pois são elas (e seus entornos) que formam o território de cada civilização.

As cidades são estabelecidas em qualquer terreno do mundo virtual (exceto oceanos e montanhas). Além disso, cada uma possui sua zona de influência (praticamente uma matriz 5 x 5 do território em torno dela), onde os civis podem ser alocados para começarem a produzir seus efeitos. Coletores de impostos e militares (quando estes estão em cidades) permanecem nos núcleos urbanos, não sendo necessário (e, pelas regras do jogo, nem possível) alocá-los em terrenos para gerarem seus efeitos.

Figura 3: uma cidade.

Cada cidade possui alguns atributos:

- Capacidade defensiva: é a soma da capacidade defensiva de todos os militares nela presentes. Quanto maior esse atributo, mais difícil da cidade ser conquistada ou destruída por militares inimigos.
- Produção: soma de toda a produção de bens gerada por seus civis.
- Arrecadação fiscal: soma de toda a arrecadação fiscal gerada por seus civis.
- Quantidade de alimentos produzidos: soma de todos os alimentos produzidos por seus civis. Esse atributo determina o tamanho da cidade, pois cada civil ou coletor de impostos consome duas unidades de alimentação, de forma que a produção de alimentos deve ser, no mínimo, o dobro do número de habitantes.
- Taxa de manutenção: valor cobrado para manter todos os militares criados numa cidade.
- Desperdício: quantidade de produção de bens que é perdida. Calculado conforme regras do jogo.
- Corrupção: quantidade de arrecadação fiscal que é desviada. Calculada conforme regras do jogo.

Uma outra forma de se entender esses atributos é que eles são justamente os efeitos que as pessoas geram nas cidades.

É importante frisar que uma cidade não pode ter infinitas pessoas. Embora seja possível teoricamente, na prática o número de civis e coletores de impostos é restringido pela quantidade de alimentos produzidos (o limite máximo, que é uma função dos terrenos da cidade e da sua zona de influência, é 42). O mesmo ocorre

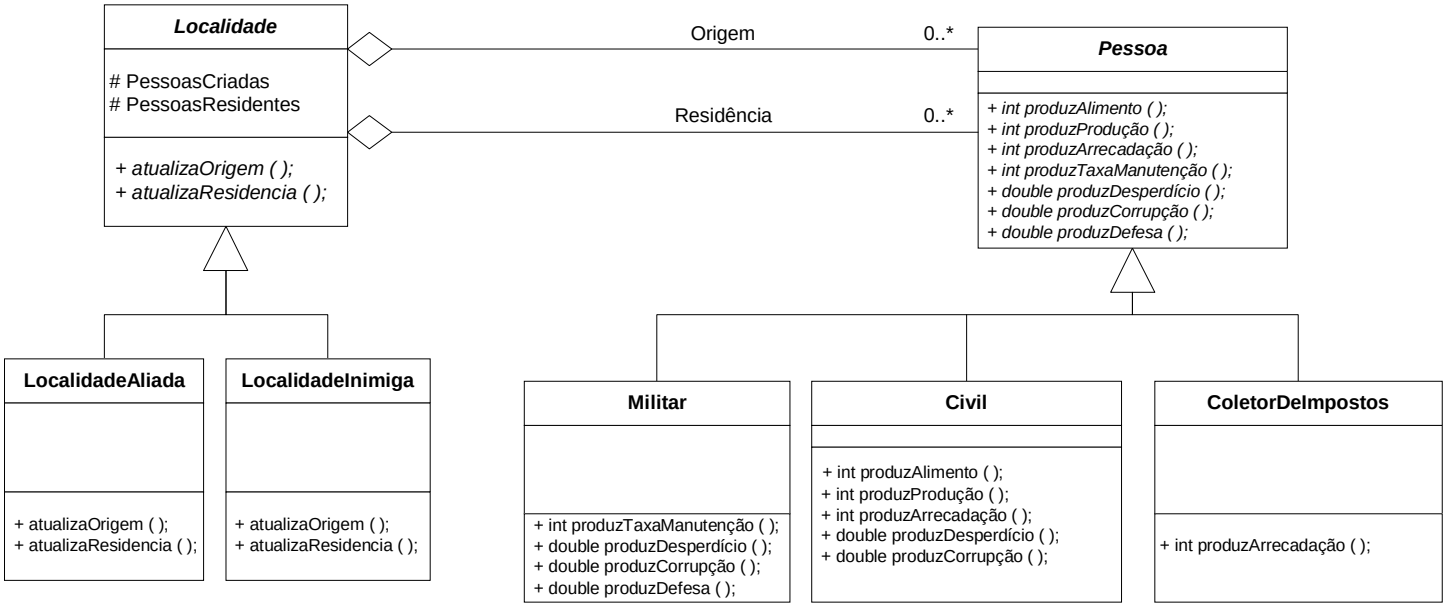
com militares, mas, nesse caso, a restrição é uma função da arrecadação fiscal menos a taxa de manutenção: quando essa diferença chega a zero, a cidade não consegue manter mais militares além dos já existentes.

Em termos de funcionamento interno, *Civilization 3* possui aquilo que, na comunidade de jogos de computador, é conhecido como *engine*. É a espinha-dorsal do software do jogo, e a parte responsável por sua dinâmica.

O *engine* controla as civilizações do jogo não pertencentes ao jogador humano; mantém uma lista com todas as cidades do jogo; é responsável pela estrutura de turnos, ou seja, a cada turno do jogo, ele percorre sua lista com as cidades de todas as civilizações, aplicando os efeitos das pessoas nas cidades; quando uma pessoa muda de categoria, o *engine* é quem notifica a cidade; controla a passagem do tempo; mantém estatísticas sobre cada civilização; enfim, o *engine* é parte essencial não apenas de *Civilization 3*, mas de todos os jogos de estratégia para computador.

Portanto, 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 da localidade de acordo com o seu interesse, tornando inconsistente a definição desse 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.

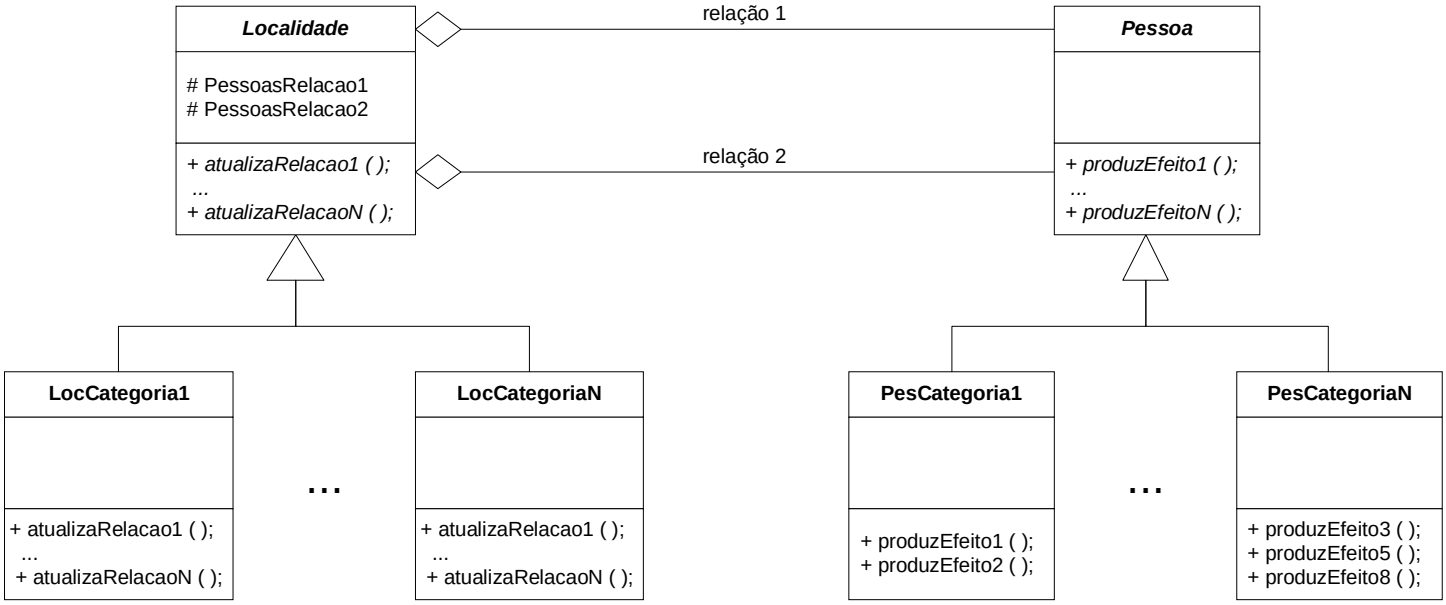


Aplicabilidade

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.

Conseqüências

O padrão:

1. **Aumenta o encapsulamento**: 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 relações**: caso queira-se acrescentar um novo tipo de relação da Pessoa sobre Localidade, basta apenas fazer uma nova agregação entre ambos. O método *atualiza ()* será, então, responsável por interpretar os efeitos produzidos pelas pessoas desta nova relação. Por exemplo: em um jogo, pode-se querer criar a relação de uma Pessoa numa Localidade inimiga (sabotagem, terrorismo). Criando-se um novo tipo de agregação, esta nova relação é facilmente acrescentada 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 as relações**: em Localidade, cria-se uma referência para cada tipo de relação que os objetos de Pessoa podem ter sobre ela.

Exemplo de código

Tomemos 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.

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 relações entre pessoas e localidades no jogo, bem como os efeitos que as primeiras geram nas segundas:

```
import java.util.*;

abstract public class Localidade
{
  protected ArrayList pessoasCriadas;         // Relação
  protected ArrayList pessoasResidentes;      // Relação
  protected int alimentacao;                  // Efeito
  protected int producao;                     // Efeito
  protected int arrecadacao;                  // Efeito
  protected int taxa_manutencao;              // Efeito
  protected int desperdicio;                  // Efeito
  protected int corrupcao;                    // Efeito
  protected int defesa;                       // Efeito

  abstract public void atualizaOrigem();
  abstract public void atualizaResidencia();
  //...
}
```

Imaginemos, agora, uma subclasse de Localidade, onde os efeitos serão aplicados. Conforme explicado na Motivação, o *engine* do jogo chamará o método a*tualiza* a cada turno do jogo:

```
import java.util.*;

public class LocalidadeAliada extends Localidade
{
```

```
  public void atualiza()
  {
    alimentacao = 0;
    producao = 0;
    arrecadacao = 0;
    taxa_manutencao = 0;
    desperdicio = 0;
    corrupcao = 0;
    defesa = 0;
    atualizaOrigem();
    atualizaResidencia();
  }

  public void atualizaOrigem()
  {
    double desp_temp = 0,
           corrup_temp = 0;

    for (Iterator i = pessoasCriadas.iterator(); i.hasNext();)
    {
      alimentacao += ((Pessoa) i.next()).produzAlimento();
      producao += ((Pessoa) i.next()).produzProducao();
      arrecadacao += ((Pessoa) i.next()).produzArrecadacao();
      taxa_manutencao += ((Pessoa) i.next()).produzTaxaManutencao();
      desp_temp += ((Pessoa) i.next()).produzDesperdicio();
      corrup_temp += ((Pessoa) i.next()).produzCorrupcao();
    }
    desperdicio = (int) java.lang.Math.floor(desp_temp);
    corrupcao = (int) java.lang.Math.floor(corrup_temp);
  }

  public void atualizaResidencia()
  {
    double def_temp = 0;

    for (Iterator i = pessoasResidentes.iterator(); i.hasNext();)
    {
      desperdicio += ((Pessoa) i.next()).produzDesperdicio();
      corrupcao += ((Pessoa) i.next()).produzCorrupcao();
      def_temp += ((Pessoa) i.next()).produzDefesa();
    }
    defesa = (int) java.lang.Math.ceil(def_temp);
  }
  //...
}
```

Podemos ver que a classe LocalidadeAliada delega para os seus objetos das subclasses de Pessoa a responsabilidade de produzir os efeitos em sua localidade. Estes efeitos podem ser então manipulados pela LocalidadeAliada conforme as regras de *Civilization 3*.

A subclasse LocalidadeInimiga seria análoga à subclasse LocalidadeAliada.

Em seguida, temos a classe Pessoa, que define os métodos que produzem os efeitos.

```
abstract public class Pessoa
{
  abstract public int produzAlimento();
```

```
  abstract public int produzProducao();
  abstract public int produzArrecadacao();
  abstract public int produzTaxaManutencao();
  abstract public double produzDesperdicio();
  abstract public double produzCorrupcao();
  abstract public double produzDefesa();
  //...
}
```

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

```
public class Militar extends Pessoa
{
  private int brutalidade;
  private int capac_ofensiva;
  private int capac_defensiva;
  private int mobilidade;
  private String status;

  public int produzAlimento()
  {
    return 0;
  }

  public int produzProducao()
  {
    return 0;
  }

  public int produzArrecadacao()
  {
    return 0;
  }

  public int produzTaxaManutencao()
  {
    if (status.equals("convocado"))
      return 1;
    else
      return 2;
  }

  public double produzDesperdicio()
  {
    return -0.1 * brutalidade;
  }

  public double produzCorrupcao()
  {
    return -0.15 * brutalidade;
  }

  public double produzDefesa()
```

```
  {
    if (status.equals("convocado"))
      return 2 * capac_defensiva;
    if (status.equals("regular"))
      return 3 * capac_defensiva;
    if (status.equals("veterano"))
      return 4 * capac_defensiva;
    if (status.equals("elite"))
      return 5 * capac_defensiva;
    return 0;
  }
  //...
}
```

Agora a classe Civil:

```
public class Civil extends Pessoa
{
  private int produtividade;
  private Terreno terreno;

  public int produzAlimento()
  {
    return terreno.getAlimentacao() * produtividade;
  }

  public int produzProducao()
  {
    return terreno.getProducao() * produtividade;
  }

  public int produzArrecadacao()
  {
    return terreno.getArrecadacao() * produtividade;
  }

  public int produzTaxaManutencao()
  {
    return 0;
  }

  public double produzDesperdicio()
  {
    return 0.1 * CivilizacaoRomana.grauDeDesperdicio();
  }

  public double produzCorrupcao()
  {
    return 0.1 * CivilizacaoRomana.grauDeCorrupcao();
  }

  public double produzDefesa()
  {
    return 0;
  }
```

```
  //...
}
```

Por fim, a classe ColetorDeImpostos:

```
public class ColetorDeImpostos extends Pessoa
{
  public int produzAlimento()
  {
    return 0;
  }

  public int produzProducao()
  {
    return 0;
  }

  public int produzArrecadacao()
  {
    return 1 * CivilizacaoRomana.grauDeFiscalizacao();
  }

  public int produzTaxaManutencao()
  {
    return 0;
  }

  public double produzDesperdicio()
  {
    return 0;
  }

  public double produzCorrupcao()
  {
    return 0;
  }

  public double produzDefesa()
  {
    return 0;
  }
  //...
}
```

Usos conhecidos

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

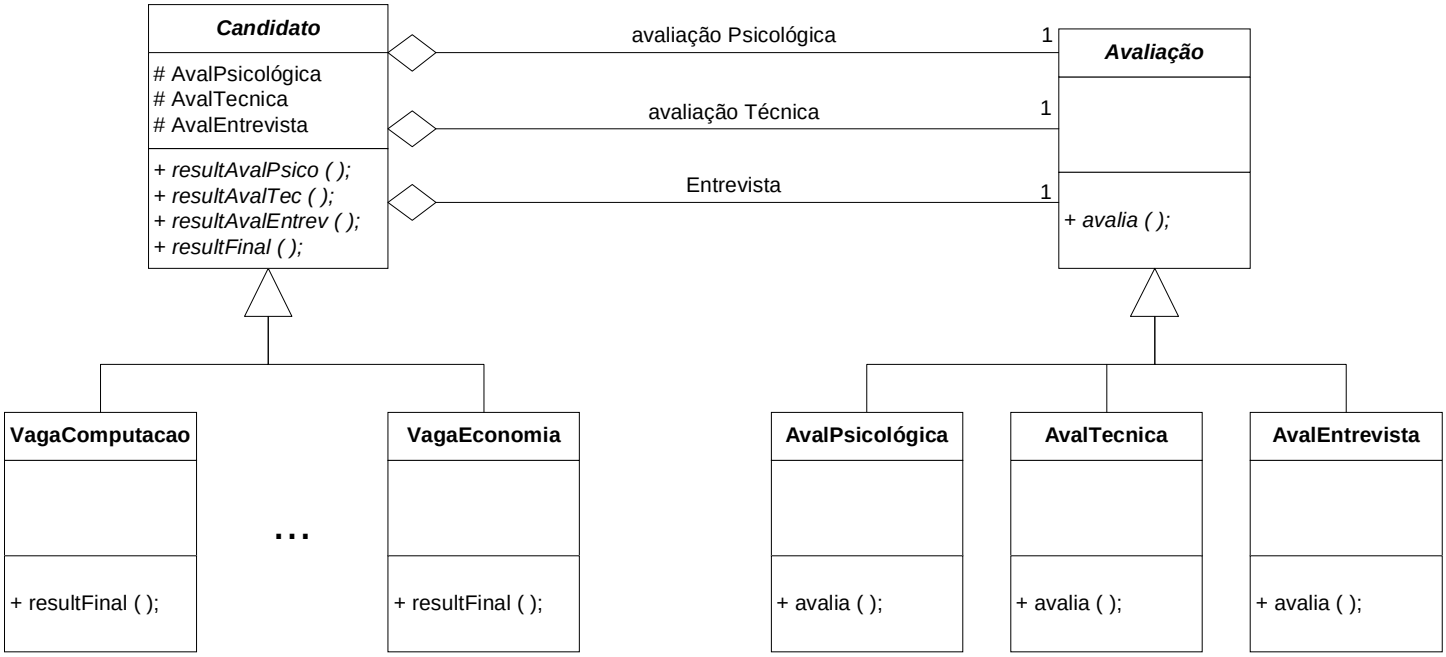
2. Sistema de avaliação de candidatos em um processo seletivo:

(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.