Conceitos finais de OO e Requisitos

Pilares da Orientação a Objeto

Agregação: Um jeito de agregar objetos a outros, as informações de um objeto são usadas para completar outro objeto de outra classe.

Composição: Uma maneira de se combinar objetos simples para formar objetos mais complexos. Implica em uma relação de ter (ou é composto), ao invés de uma relação de ser (obtida via herança)

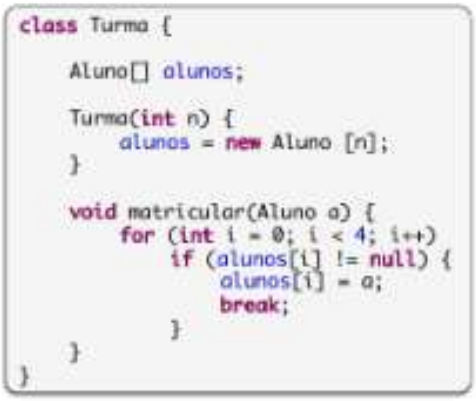
Um objeto (A) pode ter uma relação com outro objeto (B) de diversas formas (multiplicidade):

- 0...1: A tem no máximo um B
- 1...1: A tem um e somente um B
- 0...*: A tem muitos B's*
- *1...*: A tem um ou mais B's
- 3...5: A tem de três a cinco B's (valores específicos)

A multiplicidade é implementada usando objetos. Um objeto pode ter um objeto como atributo:
Um relógio possui data e hora.

Outro exemplo é que um objeto pode ter uma coleção de objetos como atributo:

Uma turma possui vários alunos (implementado como arrays, coleções ou outros).



```
class Turma {

    Aluno[] alunos;

    Turma(int n) {
        alunos = new Aluno [n];
    }

    void matricular(Aluno a) {
        for (int i = 0; i < 4; i++)
            if (alunos[i] != null) {
                alunos[i] = a;
                break;
            }
    }
}
```

Ou

```
import java.util.*;
class Turma {

    List<Aluno> alunos = new
        ArrayList<Aluno>();

    void matricular(Aluno a) {
        alunos.add(a);
    }
}
```

Herança vs Composição

Herança e composição são dois mecanismos para a reutilização de funcionalidades.

Herança: estende atributos e métodos de uma classe.

Composição: estende por meio de delegação.

A herança sempre foi considerada como a ferramenta básica para extensão e reuso, mas a composição é muito superior na maioria dos casos. A herança define um relacionamento estático, enquanto a composição define um relacionamento dinâmico. Composição e herança não são mutuamente exclusivas; essas técnicas podem ser combinadas para obter melhores resultados. Mas quando devemos usar herança e quando devemos usar composição?

Quando usar Herança? Somente deve ser usada quando estiver construindo uma família de tipos (relacionados entre si). Somente se puder comparar um objeto A com outro objeto B dizendo que, A "É UM tipo de..." B

Quando usar Composição? – Em todos os outros casos! ☺

Análise e Projetos Orientados a Objetos (APOO): serve para compreender o domínio do problema e a partir disso fazer um modelo das atividades necessárias para tal, ou seja, modela o que deve ser feito para solucionar um problema. Além disso, é através da análise que a informação produzida para o cliente discutir e

aprovar. O projeto em si, possui as atividades resultantes em informação necessária apenas ao programador.

A Análise e Projetos Orientados a Objetos é uma atividade crucial em um processo de desenvolvimento de software. Portando, seus principais objetivos são identificar objetos, atributos dos objetos e as ações que eles devem sofrer na hora do processamento. Os atributos são considerados as características ou propriedades dos objetos, as ações são métodos que atuam sobre os objetos e alteram seu comportamento.

Requisitos: são as necessidades que definem a realização do negócio e atenda ao usuário. A ideia principal de requisitos é identificar e documentar o que um sistema deve fazer, ajudando a comunicação de todos os envolvidos no projeto de forma clara e não-equívoca de maneira que os riscos sejam identificados a priori e não acontecer imprevistos.

Tipos de requisitos: Funcionais, não funcionais, domínio

Funcionais: o que o sistema deve fazer.

Não funcionais: são restrições sobre os serviços ou funções oferecidas pelo sistema.

Domínio: São regras de negócios

Permanentes: requisitos que não mudam.

Voláteis:

Mutáveis: se modificam por causa do ambiente do sistema.

Emergentes: surgem a medida que a compreensão do cliente do sistema se desenvolve.

Consequentes: resultam da introdução do sistema no ambiente do usuário.

Compatibilidade: são os requisitos que dependem de outro equipamento ou processo. Se os equipamentos são alterados os requisitos também.

Usuário: são declarações, em uma linguagem natural com diagramas para o entendimento de qualquer pessoa.

De sistemas: são descrições mais detalhadas das funções, serviços e restrições operacionais do sistema de software.

Problemas da Análise de Requisitos: entender o que o cliente deseja realmente fazer, nem sempre ele detalha o suficiente. Mudanças de requisitos no meio do processo, fatores políticos e organizacionais.

Validação dos Requisitos: Verifica se realmente o que está descrito é o que o cliente/usuário deseja.

Atividade Extra

Para saber mais leia a apresentação "Requisitos de Software", disponível em https://edisciplinas.usp.br/pluginfile.php/3142953/mod_resource/content/2/Aula09-Requisitos.pdf.

Referências Bibliográficas

BOOCH, Grady. **Uml - Guia do Usuário**. Rio de Janeiro: Editora Campus, 2018.

GUEDES, Gilleanes T. A. **UML 2 - Uma Abordagem Prática**. São Paulo: NovaTec, 2018.

PRESSMAN, Roger; MAXIM, Bruce. **Engenharia de Software.** Porto Alegre: Bookman, 2010.

SOMMERVILLE, Ian. **Engenharia de software.** São Paulo: Pearson, 2015.

Ir para exercício