PROJETO ARQUITETURAL
PARTE II: PADRÕES DE PROJETO

Projeto de Programas – PPR0001

QUALIDADE DO PROJETO

Qualidade do Projeto de Software

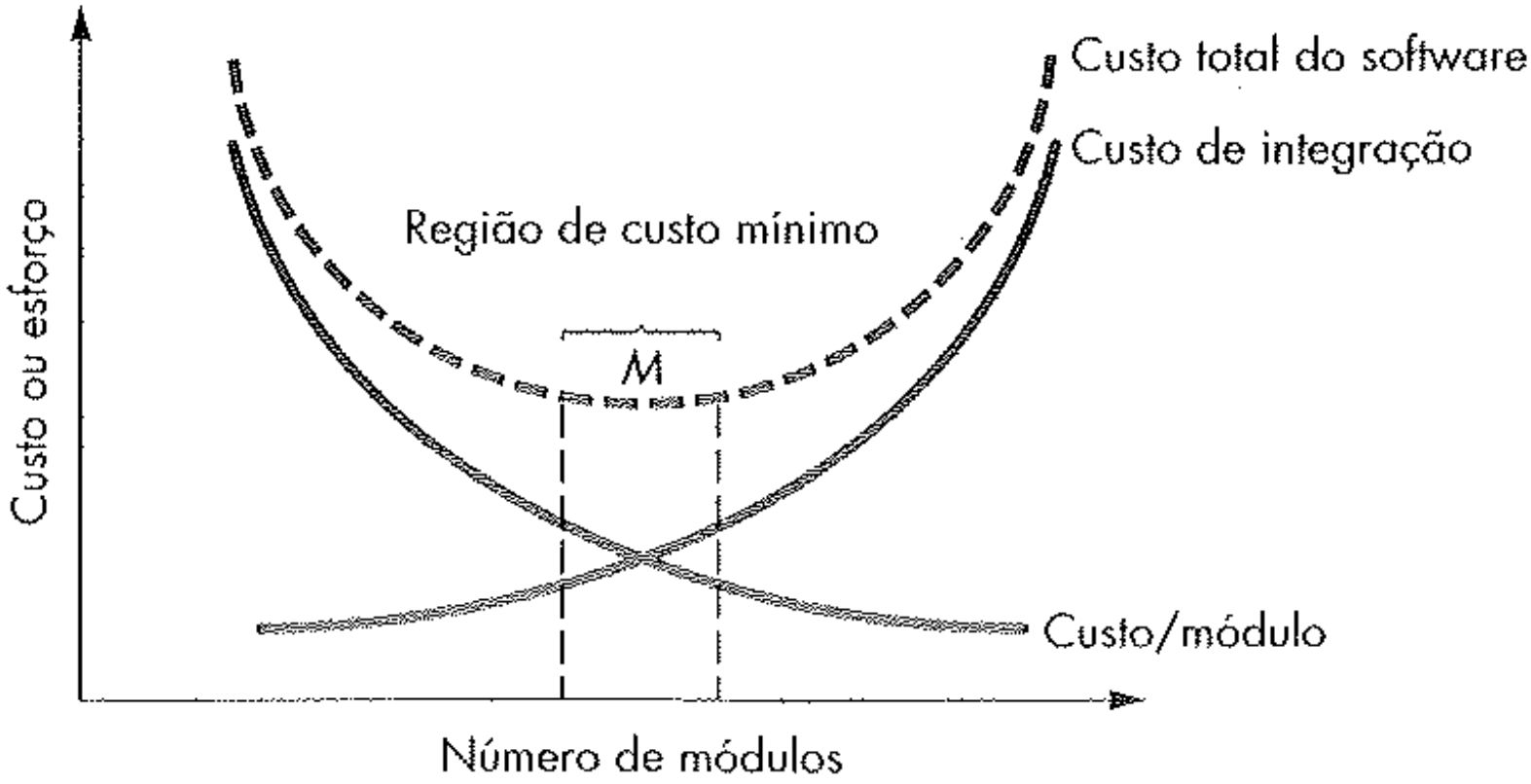
Modularidade: gerar particionamento em elementos que executam funções específicas;

- Características:
  - **Coesão:** medida da identidade funcional de um módulo;
    (uma funcionalidade somente? Mais de uma?)
  - **Acoplamento:** Mede o grau em que um módulo está "conectado" a outros elementos;

- Desafios:
  - Qual o número "correto" de módulos para um projeto?
  - Qual o tamanho "correto" dos módulos?

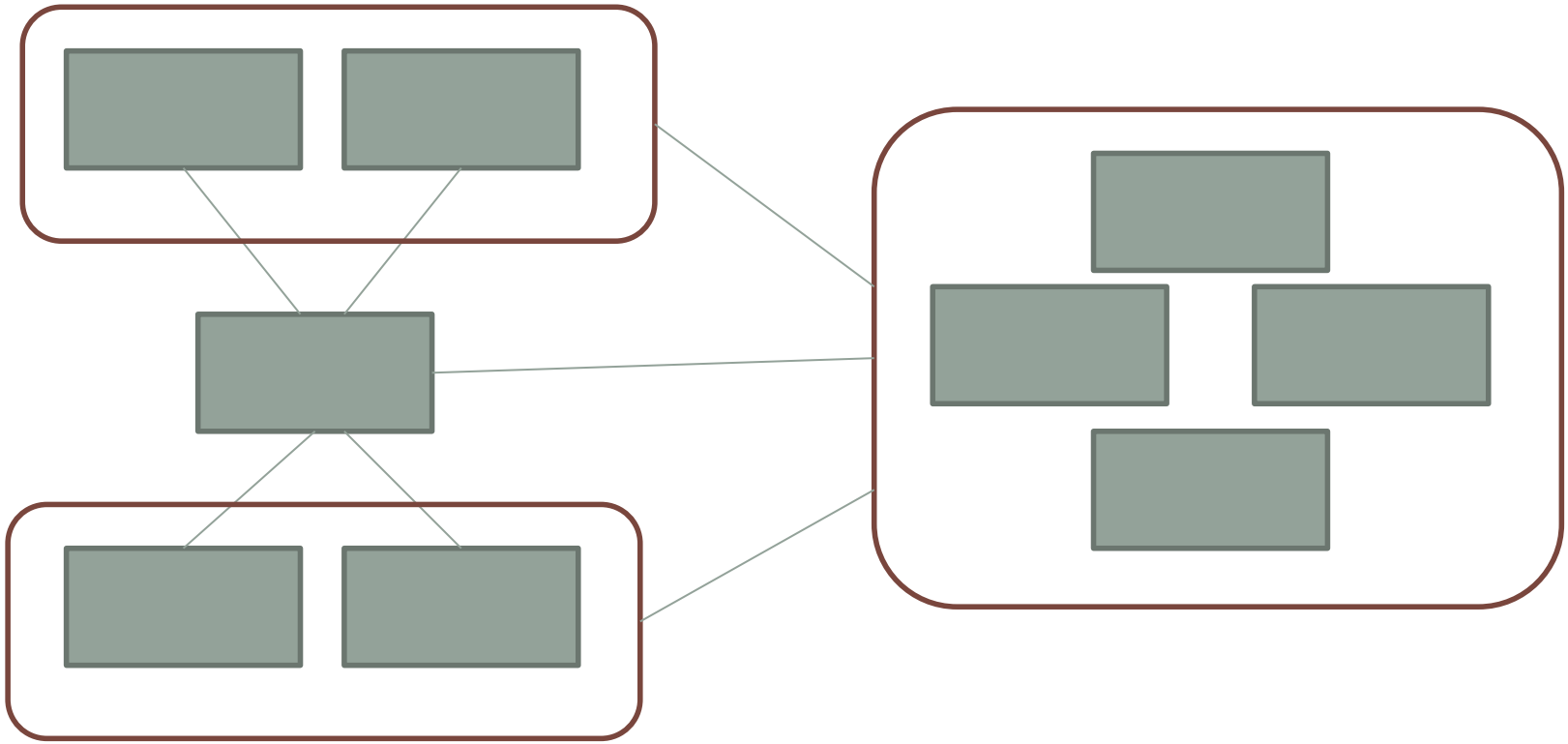
Qualidade do Projeto de Software

Modularidade:



Qualidade do Projeto de Software

Organização Hierárquica: manter uma boa organização entre os elementos / módulos de software;



Princípios de Projeto

- Minimizar a distância intelectual entre o software e o problema do mundo real: Evitar associações abstratas para o usuário quando possível
- Não reinventar a roda - Utilize padrões
- Usar bem o **encapsulamento** de modo que as informações de um módulo não sejam acessíveis aos módulos que não precisam delas
  - Reduz a ocorrência de efeitos colaterais
  - Questão: *public* *vs.* *getters* & *setters*
  - Erros não se propagam pelo software
- Definir regras de estilo e projeto para a equipe do projeto;
- Ter cuidado e definir bem as interfaces do sistema

PADRÕES DE PROJETO

Introdução

- Você utiliza um padrão para programar? Qual?
  - Programação orientada a gambiarra?!

Algoritmo KickSort / BoboSort / EstouComSort [desciclopedia - adaptado]

```
void kickSort(int v[10]){
    int aux1, aux2, aux3;
    srand( time(NULL) );
    inicio:
        if( v[0]<=v[1] && v[1]<=v[2] && v[2]<=v[3] &&
            v[3]<=v[4] && v[4]<=v[5] && v[5]<=v[6] &&
            v[6]<=v[7] && v[7]<=v[8] && v[8]<=v[9])
                return;
    aux1 = rand()%10;
    aux2 = rand()%10;
    aux3 = v[aux1];
    v[aux1] = v[aux2];
    v[aux2] = v[aux3];
    goto inicio;
}
```

Padrões de nomenclatura

- Você utiliza um padrão para programar? Qual?
  - Padrão utilizado na nomenclatura de variáveis e métodos?

Padrões de nomenclatura

- O padrão mais utilizado em nomenclatura é:
  - Nomes de variáveis e métodos condizentes
  - Variáveis: letras minúsculas, onde nomes compostos são separados por *underline* *(ex.: linha_atual, salario_novo)*
  - Métodos: letras minúsculas, onde as iniciais dos nomes compostos ficam em caixa alta *(ex.: abrirArquivo(), fecharArquivo())*
  - Valores constantes: letras maiúsculas
  - Classes: palavra capitalizada (inclusive primeira)

Padrão de estrutura

- Você utiliza um padrão para programar? Qual?
  - ~~Padrão utilizado na nomenclatura de variáveis e métodos?~~
  - Padrão de estrutura para o código?

Padrão de estrutura

- Padrão de estrutura para o código?
  - Operações sempre implementadas em funções
  - **Classe de Definições, Linguagem, Operações Genéricas e Sistema**
  - **Padrão *Singleton***
  - **Padrão Façade (fachada)**

Padrão de estrutura

- **Classe de Definições**
  - Definições do sistema (volume, linguagem atual, dificuldade, constantes, ...)
- **Classe de Linguagem**
  - Métodos que retornam os textos para cada label da tela tomando como base a linguagem atual definida
- **Classe de Operações Genéricas (Toolbox)**
  - Classe com métodos genéricos que podem ser utilizados em qualquer parte do código, ex: conversão de graus para radianos, conversão de um numero de telefone em inteiro para string e vice-versa.
- **Classe de Sistema**
  - Classe principal do sistema que possui os registros salvos ou que serão resgatados, ex: clientes, produtos.
    Pode ser agrupada com a classe de definições.
    Deve-se garantir que exista apenas uma única instância dessa classe.

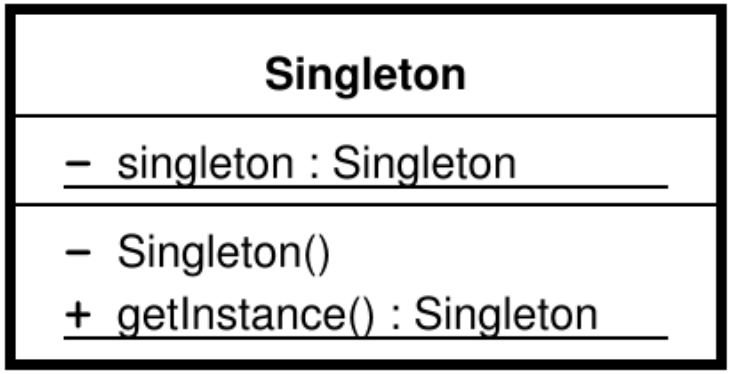
Padrão de estrutura

- **Padrão Singleton**
  - Garante uma única instância de um dado objeto.

  - **Forma simples**: uso de atributos staticos e públicos
  - **Forma encapsulada**: uso de atributo statico e privado + construtor privado + método para recuperar instância.

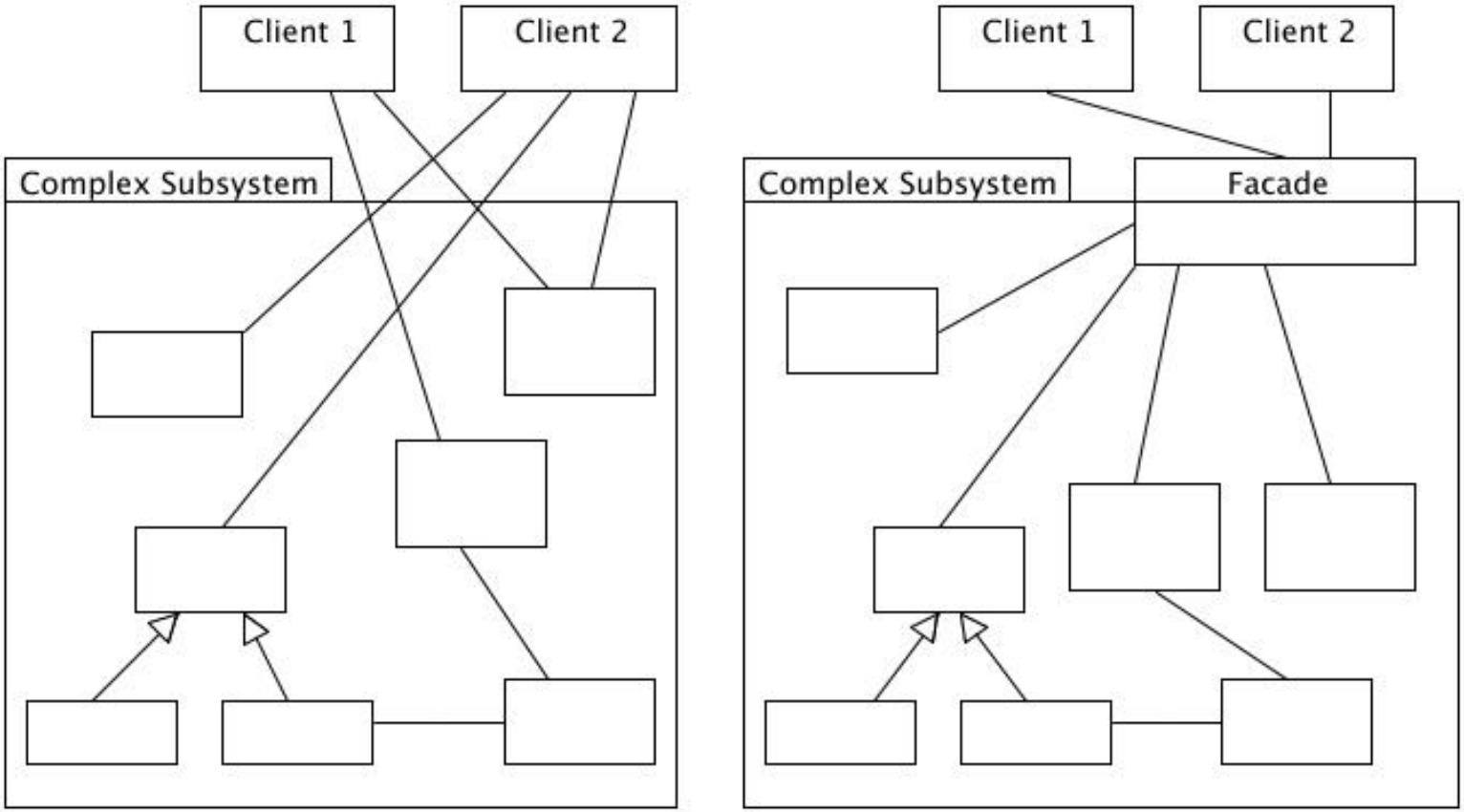
    Se a instância do atributo não existe: cria uma nova e retorna-a;

    Se a instância do atributo já existe: apenas retorna o atributo/objeto;



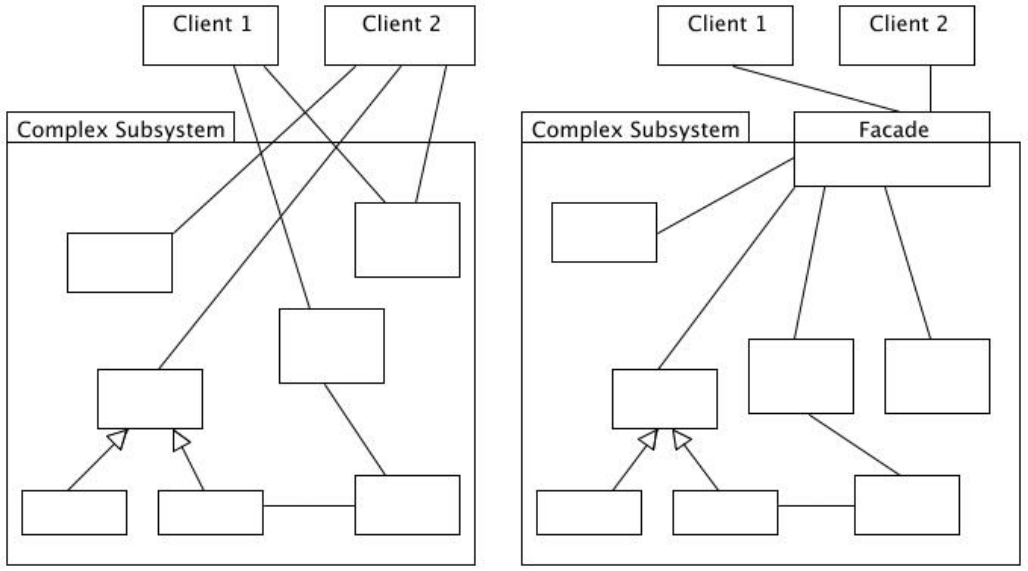
Padrão de estrutura

- **Padrão Façade (fachada) – classe de acesso/distribuidora**



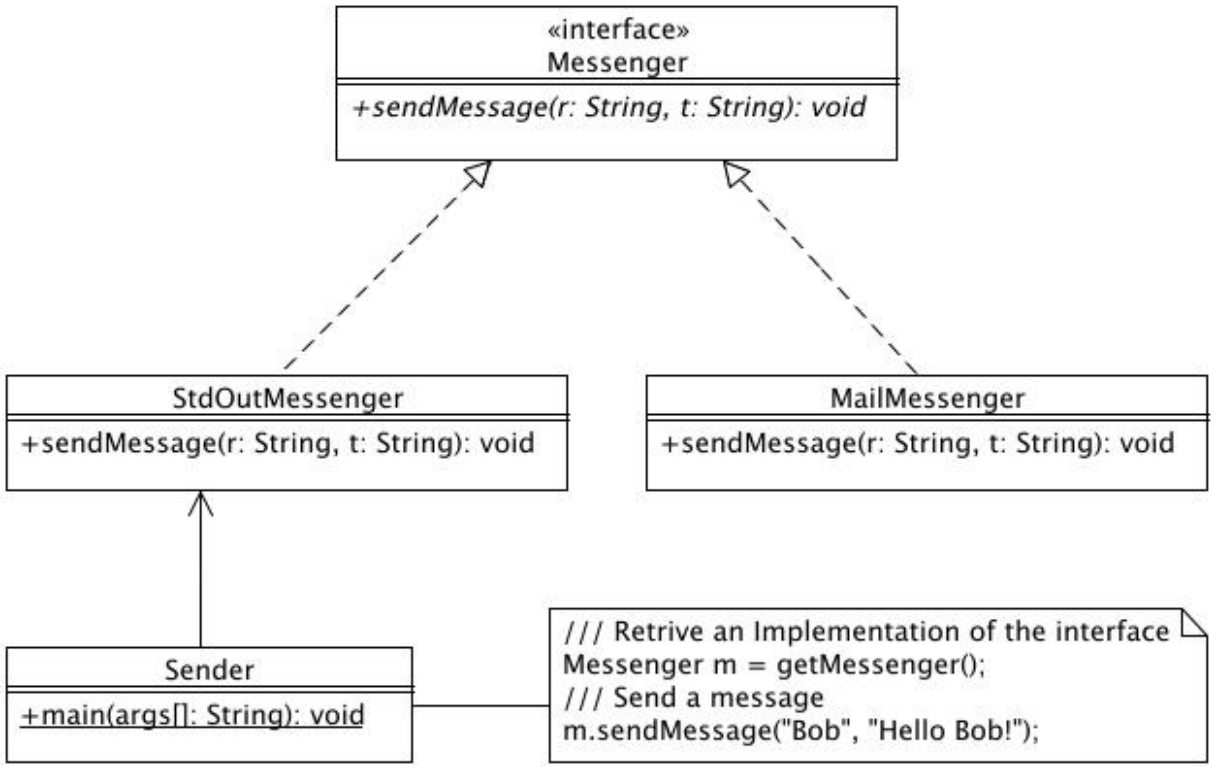
Padrão de estrutura

- **Padrão Façade (fachada) – classe de acesso/distribuidora**
  - Objetivo: centralizar e simplificar invocação de métodos
  - Implementação: todas as classes do subsistema (pacote) não são públicas com exceção da fachada. Como a fachada é uma classe do pacote, pode direcionar as chamadas para os demais métodos



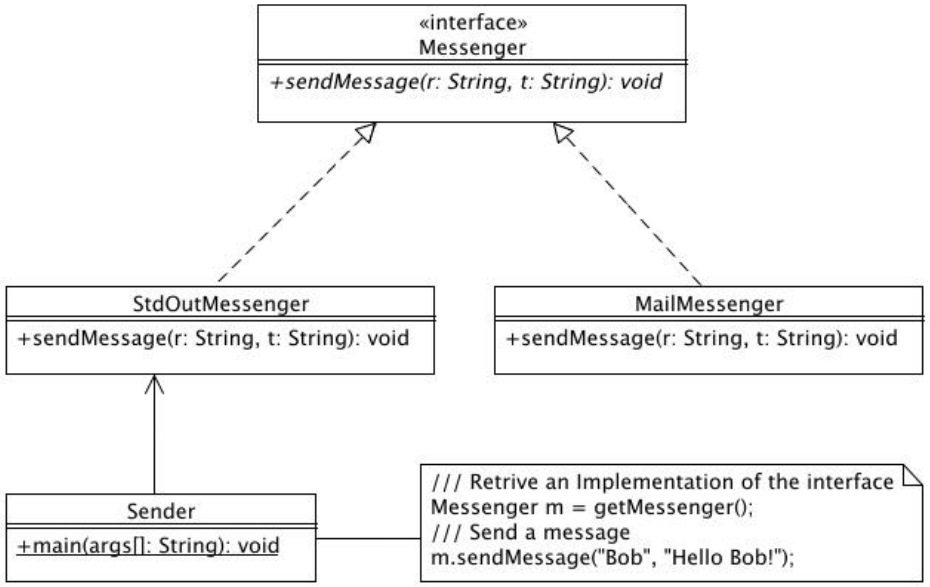
Padrão de estrutura

- **Padrão Façade (fachada) – interface de acesso/implementação**



Padrão de estrutura

- **Padrão Façade (fachada) – interface de acesso/implementação**
  - Objetivo: facilitar reuso e realização de alterações
  - Implementação: a classe de fachada é uma interface que descreve todos os métodos que devem ser implementados. As classes que implementam a interface são obrigadas a implementar todos os métodos da fachada.
  - Facilita muito a alteração de tecnologia ou método aplicado.



Padrão de arquitetura

- Você utiliza um padrão para programar? Qual?
  - ~~Padrão utilizado na nomenclatura de variáveis e métodos?~~
  - ~~Padrão de estrutura para o código?~~
  - Padrão da estrutura para o software [padrão arquitetural?]

Padrão de arquitetura

- Os **estilos arquiteturais** irão estabelecer uma **estrutura padronizada** para os componentes do sistema

- Estilo descrevem:
  - Conjunto de componentes
  - Conjunto de conectores
  - Restrições sobre a interação dos componentes
  - Modelos semânticos que permitem que o projetista entenda as propriedades gerais do sistema

Padrão de arquitetura

- Padrão da estrutura para o software [padrão arquitetural?]
  - Centrado em dados
  - Fluxo de dados
  - Chamada e retorno
  - **MVC (*Model View Controller*)**
  - **Camadas**
  - Componentes independentes
  - Máquina Virtual

Arquitetura MVC

- **Conceito**:
  - Separação de conceitos: interação vs. representação de informação
  - Reusabilidade de código

- Utilização de três tipos de componentes:
  - **M**odel: dados da aplicação, regras de negócio, lógica e funções
  - **V**iew: qualquer saída de representação dos dados
    (ex: tabela, diagrama)
  - **C**ontroller: controla a aplicação convertendo entradas dos usuários
    em chamadas para as classes de modelo e visão.

Arquitetura em camadas

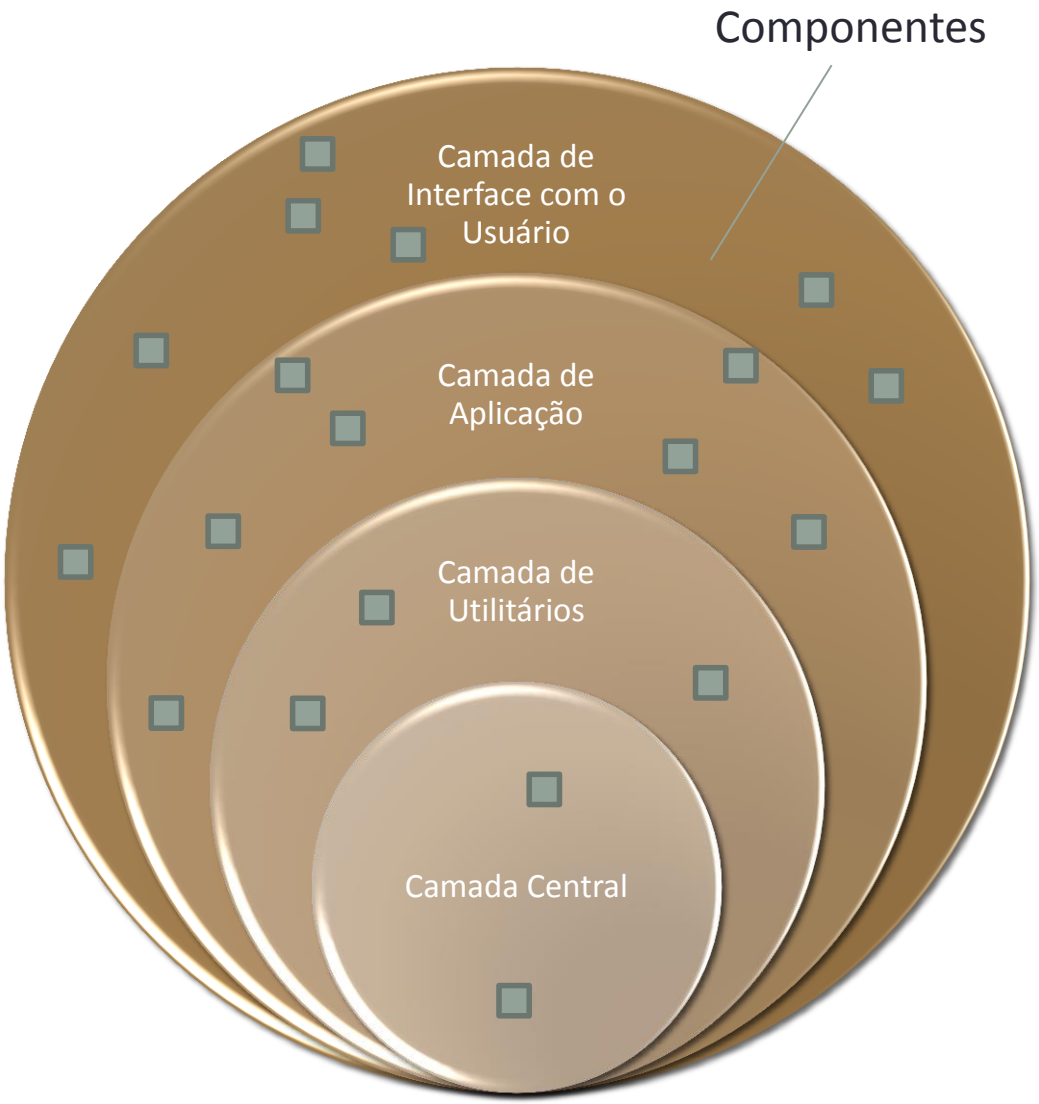
- **Conceito**:
  - Separação de conceitos: interação, regras de negócio, abstração, armazenamento dos dados
  - Reusabilidade de código
- O projeto dispõem os componentes em camadas
- A comunicação entre componentes só é permitida pelas camadas vizinhas (hierarquia)

Arquitetura em camadas

Cada camada tem um nível de aproximação:

Camadas mais altas (externas) estão relacionadas aos componentes de interação com o usuário

Camadas mais baixas (internas) representam os componentes que realizam a interface com o sistema operacional



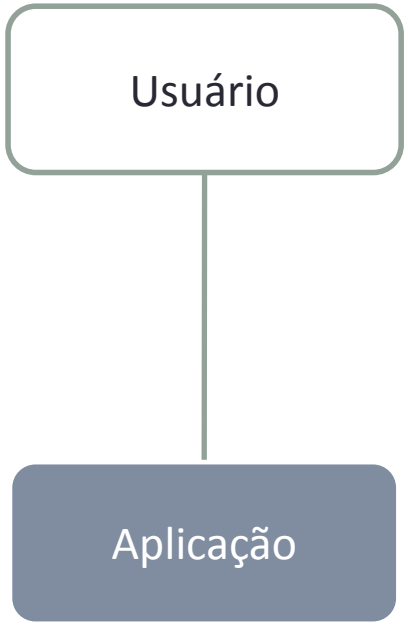
Variações do sistema em camadas

- As variações estão relacionadas ao nível de detalhamento e a especificação das funções:
  - Sistemas em duas camadas
  - Sistemas em três camadas
  - Sistemas em N camadas

Sistema de uma camada

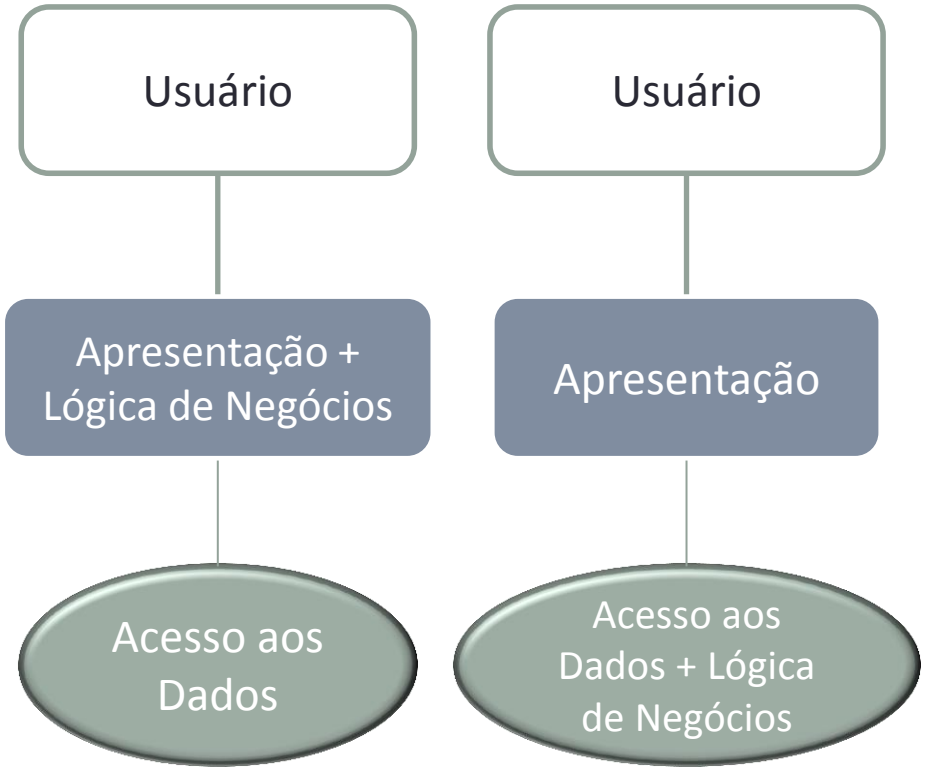
- Todas as partes constituintes estão fortemente agregadas

- Desvantagens:
  - Difícil manutenção
    - Não há separação entre lógica de negócios, dados e apresentação
  - Qualquer alteração em uma parte da aplicação pode causar efeitos colaterais em outras
  - Atualizações requerem reengenharia completa do sistema



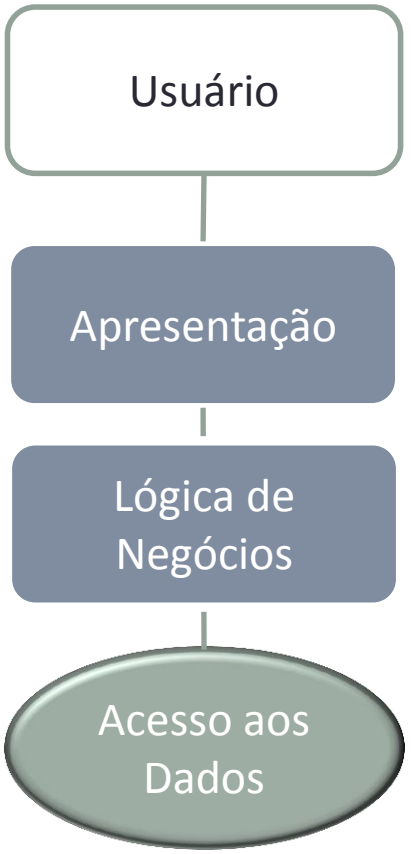
Sistemas de duas camadas

- Existe a separação da lógica de negócios e dos dados
  - Apresentação + lógica de negócios <-> Dados
  - Apresentação <-> Lógica de negócios + Dados
- Vantagem:
  - Se a lógica de negócios não for muito complexa ela pode ser agregada a outra camada
- Desvantagem
  - Existe forte relação entre as duas camadas, prejudica a manutenção

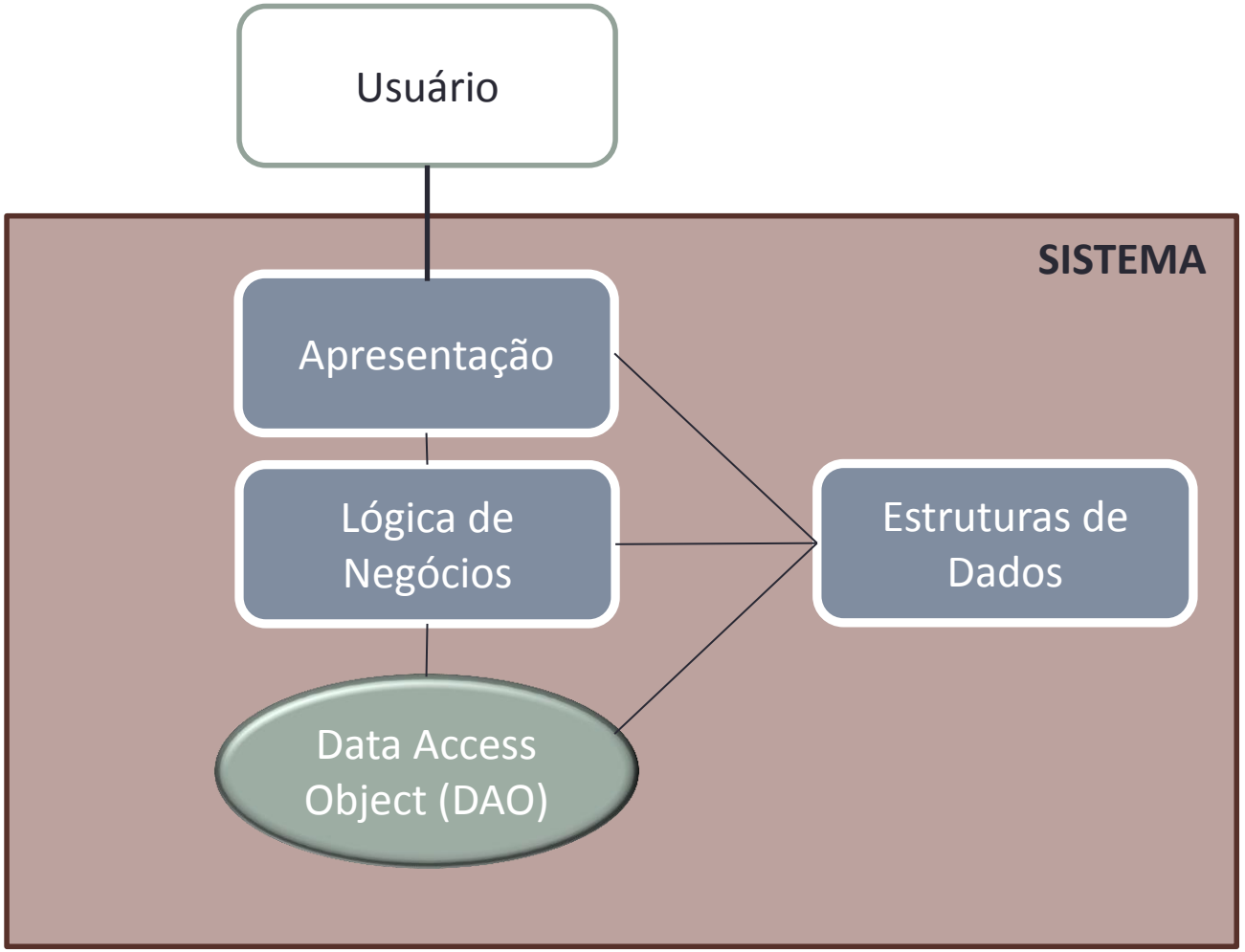


Sistema de três camadas

- Separação clara das três camadas principais do sistema
  - **Apresentação:** Interface com o usuário
  - **Lógica de Negócios:** Componentes que trabalham para codificar o processo de negócios
  - **Dados:** Provê e mantém os dados utilizados
- Vantagem:
  - Maior flexibilidade para alterar os componentes de cada camada, não há preocupação com os efeitos colaterais das alterações



Sistemas de N camadas



Exemplo

Levantar os requisitos necessários para um sistema acadêmico que permite o **controle** e **gerenciamento** de **matricula**, **frequência** e **desempenho dos discentes** e a **organização das disciplinas ofertadas**. O sistema acadêmico deverá permitir que os **acadêmicos realizem suas matrículas nas turmas de disciplinas disponíveis**, considerando restrições de pré-requisitos, número máximo de créditos (9) e limite de alunos por turma. Deverá permitir **que chefes de departamento incluam novas disciplinas e novos professores, abram novas turmas** para as disciplinas existentes com sala, horário, lotação máxima e professor definidos. As disciplinas só poderão ser ofertadas entre 7:30 e 12:00, e, 13:30 e 21:40, em blocos de 50 minutos por aula (hora-aula). Também deverá ser possível que **professores acessem suas turmas e registrem frequência e notas** para seus alunos.

Exemplo

O sistema deverá ter uma **opção para finalizar o semestre**, possibilitando a **inclusão das notas de exame**. Um aluno deverá ter frequência superior a 75% e deverá ter uma média superior a 3 para realizar exame. Caso sua nota seja maior ou igual a 7 está aprovado (desde que tenha a frequência necessária). Após a digitação das notas de exame o **professor deverá finalizar a turma** e o **sistema mostrará o resultado final**. O sistema deverá funcionar nos sistemas operacionais Windows e Linux e deverá ter seu **acesso controlado por login e senha**.

Atividade

Agora é a sua vez!

Altere seu diagrama de classes criando uma estrutura em camadas para o sistema descrito na página da disciplina.

Adicione as classes para as janelas que imagina ser necessárias ao sistema + uma classe de fachada distribuidora no pacote de negócio + uma interface de fachada no pacote DAO que é implementado por uma classe Memoria e outra classe Arquivo.

(tente colocar ao menos duas classes no pacote de negócio, além da classe de fachada)

Bibliografia

- **Básica:**

  BEZERRA, E. Princípios de Análise e Projetos de Sistemas com UML. Rio de Janeiro: Campus, 2003.

  PRESSMAN, R.S. Engenharia de Software. São Paulo: Makron Books, 2002.

  SOMMERVILLE, I. Engenharia de Software. São Paulo: Addison Wesley, 2003.

- **Complementar:**

  WARNIER, J. Lógica de Construção de Programas. Rio de Janeiro: Campus, 1984.

  JACKSON, M. Princípios de Projeto de Programas. Rio de Janeiro: Campus, 1988.

  PAGE-JONES, M. Projeto Estruturado de Sistemas. São Paulo: McGraw-Hill, 1988.