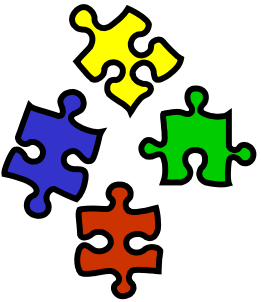
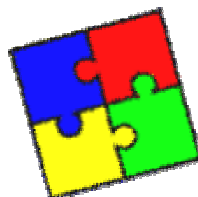
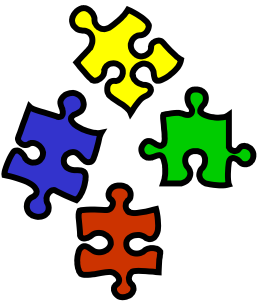


Categorias de Padrões

- Padrão Arquitetural ou Estilo Arquitetural
- Padrão de Design (Design Patterns)
- Idiomas



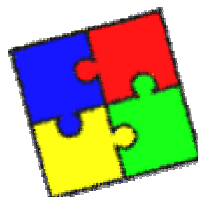


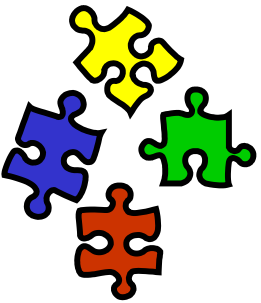
Categorias de Padrões

ESTILOS ARQUITETURAIS

PADRÕES DE DESIGN

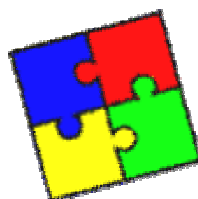
IDIOMAS

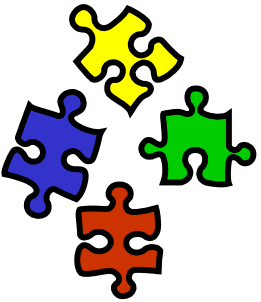




Padrões de Design

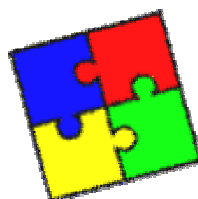
- Os subsistemas da arquitetura de software bem como os relacionamentos entre eles, consistem de várias unidades arquiteturais menores. Tais unidades são descritas usando *design patterns*.
- Oferece um esquema para refinar os subsistemas

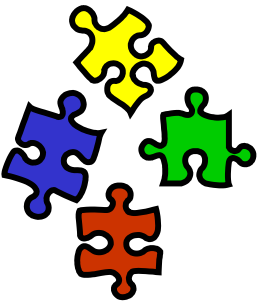




Padrões de Design

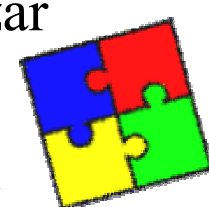
- São menores em escala que padrões arquiteturais mas estão em um nível mais alto que idiomas específicos de linguagens de programação
- Não tem efeito na estrutura fundamental do software (arquitetura) mas tem forte influência na arquitetura de um subsistema
- São independentes de um domínio particular da aplicação pois tratam com a estruturação da funcionalidade da aplicação e não com a implementação da funcionalidade

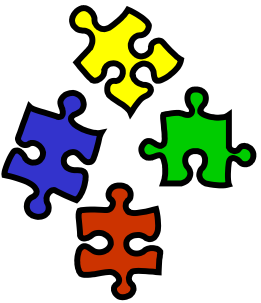




Padrões de Design

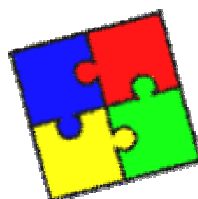
- Categorias:
  - **Decomposição Estrutural**: padrões que permitem uma decomposição adequada de subsistemas e componentes complexos em partes cooperantes
  - **Organização do Trabalho**: padrões que definem como componentes colaboram para resolver um problema complexo
  - **Controle de Acesso**: padrões que guardam e controlam o acesso a serviços ou componentes
  - **Gerenciamento:** padrões que gerenciam uma coleção homogênea de objetos, serviços e componentes
  - **Comunicação**: padrões que ajudam a organizar comunicação entre os componentes

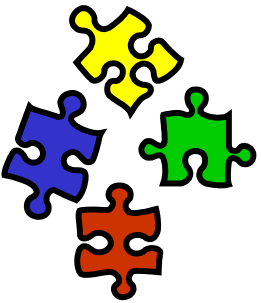




Padrões de Design
Decomposição Estrutural

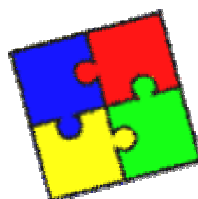
- Decompor sistemas em partes menores facilita a realização de mudanças e extensões e tornam os sistemas mais fáceis de compreender
  - Whole-Part (Todo-Parte) ajuda a agregar componentes para juntos formar uma unidade
  - Composite organiza objetos em estruturas de árvore que representa uma hierarquia

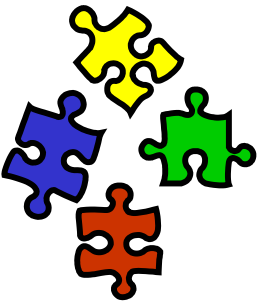




Padrões de Design
Decomposição Estrutural

- Whole-Part
  - Um componente agregado (o *whole*)
    - encapsula os componentes constituintes (o *part*)
    - organiza uma interface comum
  - Clientes enxergam o objeto agregado (whole) como um objeto atômico que não permite acesso direto às partes

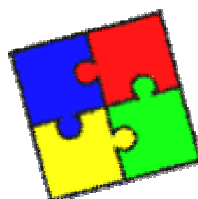


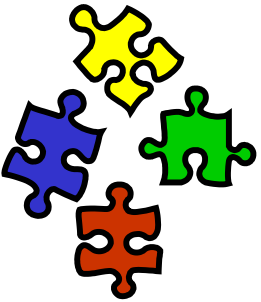


Padrões de Design
Decomposição Estrutural

- Whole-Part
  - Exemplo:

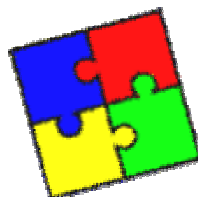
    Objetos gráficos como círculos, linhas, retângulos, triângulos, etc podem ser agrupados em uma classe base chamada **GraphicsObject** que define métodos comuns como *draw* e *rotate*.

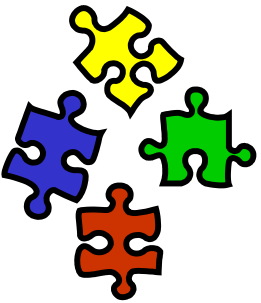




Padrões de Design
Organização do Trabalho

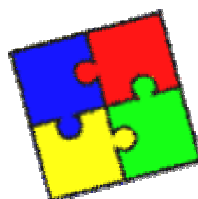
- Decompor sistemas complexos através da cooperação de componentes
  - Master-Slave (Mestre-Escravo) oferece suporte para computação paralela e tolerância a falhas. Aplica a idéia do princípio de "*Dividir e Conquistar*" onde o trabalho é dividido em subtarefas que são processadas independentemente

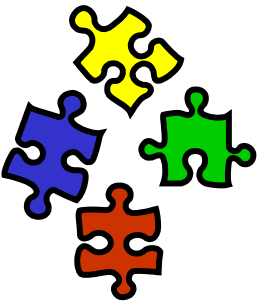




Padrões de Design
Organização do Trabalho

- Master-Slave
  - Um componente *master* (mestre) distribui trabalho a componentes *slaves* (escravos) que realizam a computação
    - o componente master divide o trabalho em sub-tarefas iguais e delega as sub-tarefas aos componentes escravos que são semanticamente idênticos



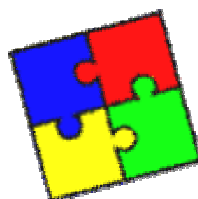


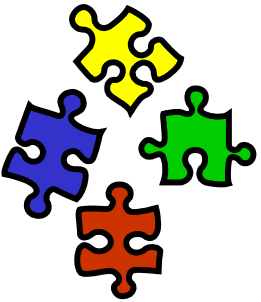
Padrões de Design
Organização do Trabalho

- Master-Slave
  - Exemplo:

    Problema do Caixeiro-Viajante (problema de teoria dos grafos) que deve encontrar o menor caminho entre duas cidades que visite cada cidade apenas 1 vez. (problema NP-completo)

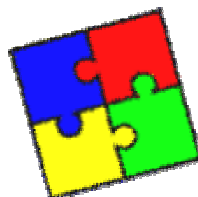
    O tamanho das diferentes rotas é calculado em paralelo pelos escravos.

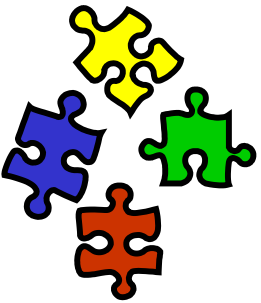




Padrões de Design
Controle de Acesso

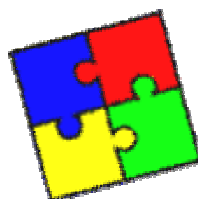
- Em algumas situações um servidor ou mesmo um sistema inteiro não deve estar acessível diretamente pelos clientes. Exemplo: nem todos os clientes estão autorizados a usar os serviços de um componente ou recuperar uma informação particular que o componente oferece
  - *Proxy* é um padrão de design que faz com que clientes de um componente comunique-se com um representante ao invés de comunicar-se com o próprio componente

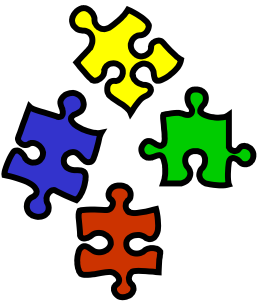




Padrões de Design
Controle de Acesso

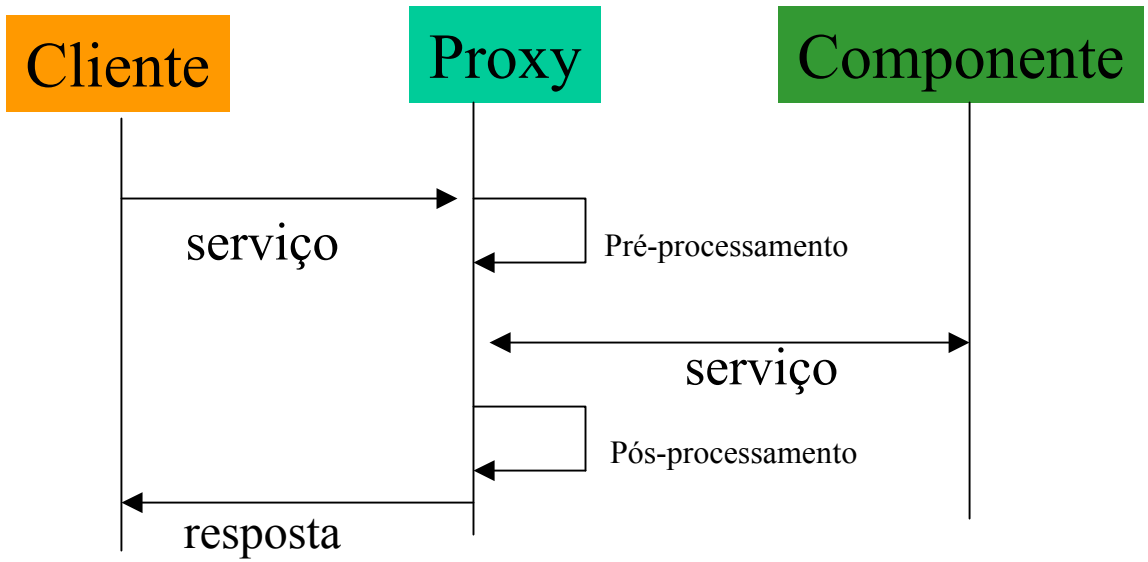
- Proxy
  - Introduz um representante (*proxy*) que serve para muitos propósitos incluindo facilidade de acesso e proteção contra usuários não autorizados
    - O proxy oferece uma interface para acesso ao componente e realiza processamento adicional (p.ex.: verificação de controle de acesso)
    - O proxy é associado com o componente original através de um *handle*, que pode ser um ponteiro, um endereço, um identificador, um socket.

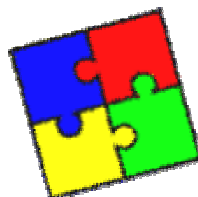


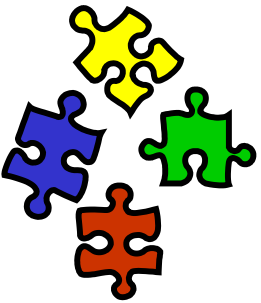


Padrões de Design
Controle de Acesso

- Proxy







Padrões de Design
Controle de Acesso

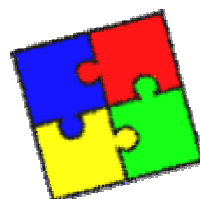
- Proxy
  - Exemplo:

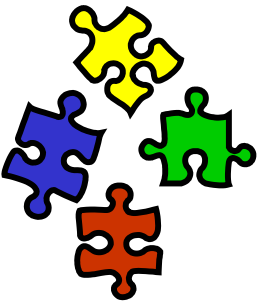
    Um proxy Firewall realiza a proteção contra potenciais acessos hostis ao ambiente.

    O proxy Firewall é, em geral, implementado como um processo daemon na máquina firewall que pode ser também chamado de "proxy server".

    Todos os clientes acessam os serviços de uma rede via o proxy.

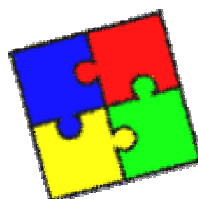
    O proxy rejeita solicitações que não obedecem a política de acesso da rede.

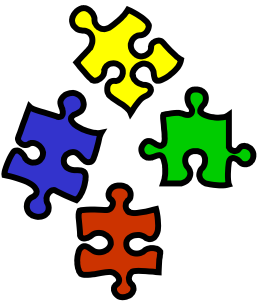




Padrões de Design
Comunicação

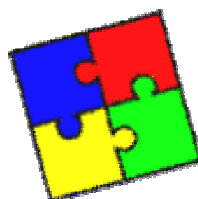
- Existem muitos mecanismos de comunicação entre processos: sockets, RPC, TLI, etc.
- O uso de um mecanismo de comunicação específico pode tornar difícil mudança na aplicação posteriormente (p.ex. em casos de migração de uma rede para outra)

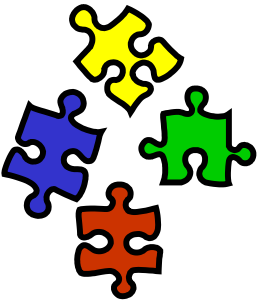




Padrões de Design Comunicação

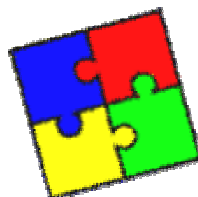
- Um padrão de design para comunicação pode permitir o desacoplamento entre componentes de um sistema distribuído e mecanismos usados para comunicação
- Dois aspectos importantes: encapsulamento do mecanismo de comunicação e transparência de localização

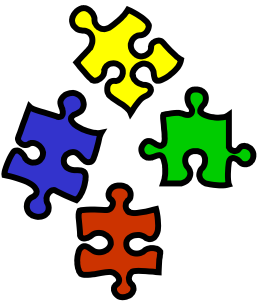




Padrões de Design
Comunicação

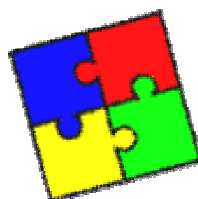
- Publisher-Subscriber
  - Mantém a sincronização entre os componentes cooperantes.
  - Um publisher notifica os subscribers sobre mudanças no seu estado.

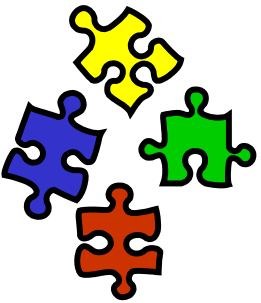




Idiomas

- É um padrão de baixo-nível específico para uma linguagem de programação
- Descreve como implementar aspectos particulares dos componentes ou relacionamentos entre eles usando as características de uma dada linguagem
- Podem endereçar a implementação de um padrão de design específico





Idiomas

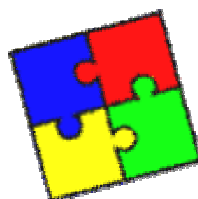
- Há muitas maneiras de se resolver um problema usando uma dada linguagem.
  - Exemplo: Implementação de cópia de string em C/C++

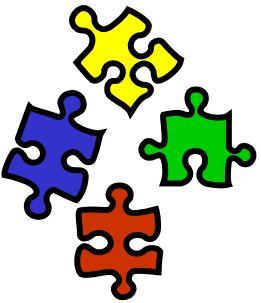
    Os programas copiam o string s para o string d até encontrar um caracter com valor 0.

```
void strcopyKR(char *d, const char *s) {
   while (*d++=*s++);
}

void strcopyPascal(char d[], const char s[]) {
   int i;
   for (i=0; s[i] != '\0'; i=i+1) {
     d[i] = s[i];  }
   d[i] = '\0';
}
```

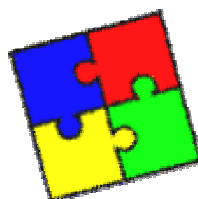
Ambas seguem seu próprio estilo

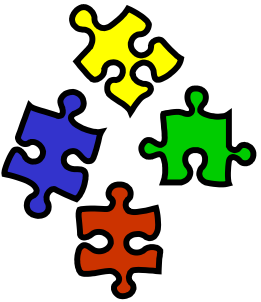




Idiomas

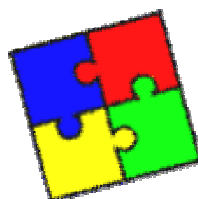
- Idiomas podem ensinar a usar as características de uma linguagem de programação
- Guias de estilos corporativos podem determinar um estilo consistente a ser seguido por times de desenvolvimento de software da corporação
  - Eles devem estabelecer como o programador deve resolver problemas recorrentes de codificação (como criação de objetos, nomeação de métodos, formato do código fonte para melhorar legibilidade, uso de biblioteca de componentes)

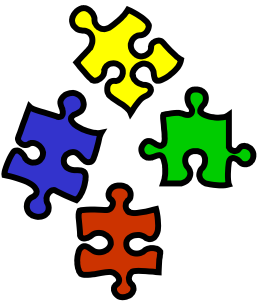




Idiomas

- Onde encontrar idiomas?
  - Alguns Padrões de Design que endereçam problemas de programação também oferecem idiomas.
  - Usando livros de linguagens de programação que falam sobre regras da linguagem crie seus próprios idiomas





Idiomas

- Exemplo: IMPLEMENTAÇÃO DE UM PADRÃO DE DESIGN *SINGLEINSTANCE* (que garante que uma classe tem apenas 1 instância)

  IDIOMA PARA SMALLTALK:

  Para os métodos *new* (construtores da classe), estabeleça uma exceção a ser emitida em casos de erro de criação do objeto.

  Implemente um método *getInstance* que retorna a instância da classe.

  A primeira vez que *getInstance* for chamado ele irá criar a única instância da classe