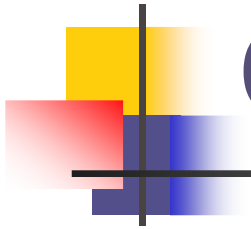
4-ação

Separação de especificação, implementação e configuração

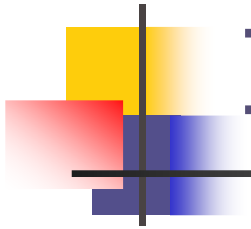
Emilio de Camargo Francesquini

emilio@ime.usp.br



Organização da apresentação

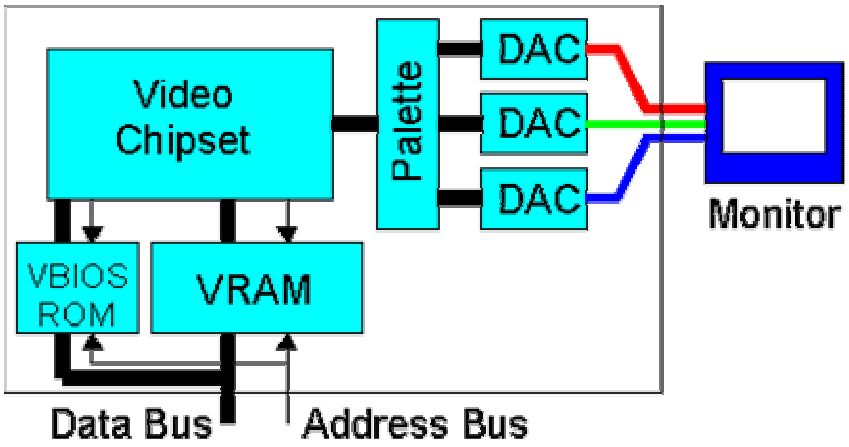
- Intenção
- Motivação
- Definir
  - Especificação
  - Implementação
  - Configuração
- 4-ação
- Exemplo de uso: JNDI
- Considerações finais



Intenção

- Fornecer um método de desenvolvimento de software onde o acoplamento entre os componentes do sistema seja suficientemente baixo de forma que seja possível a troca de componentes com pouquíssimo esforço e trabalho para adaptação do sistema.

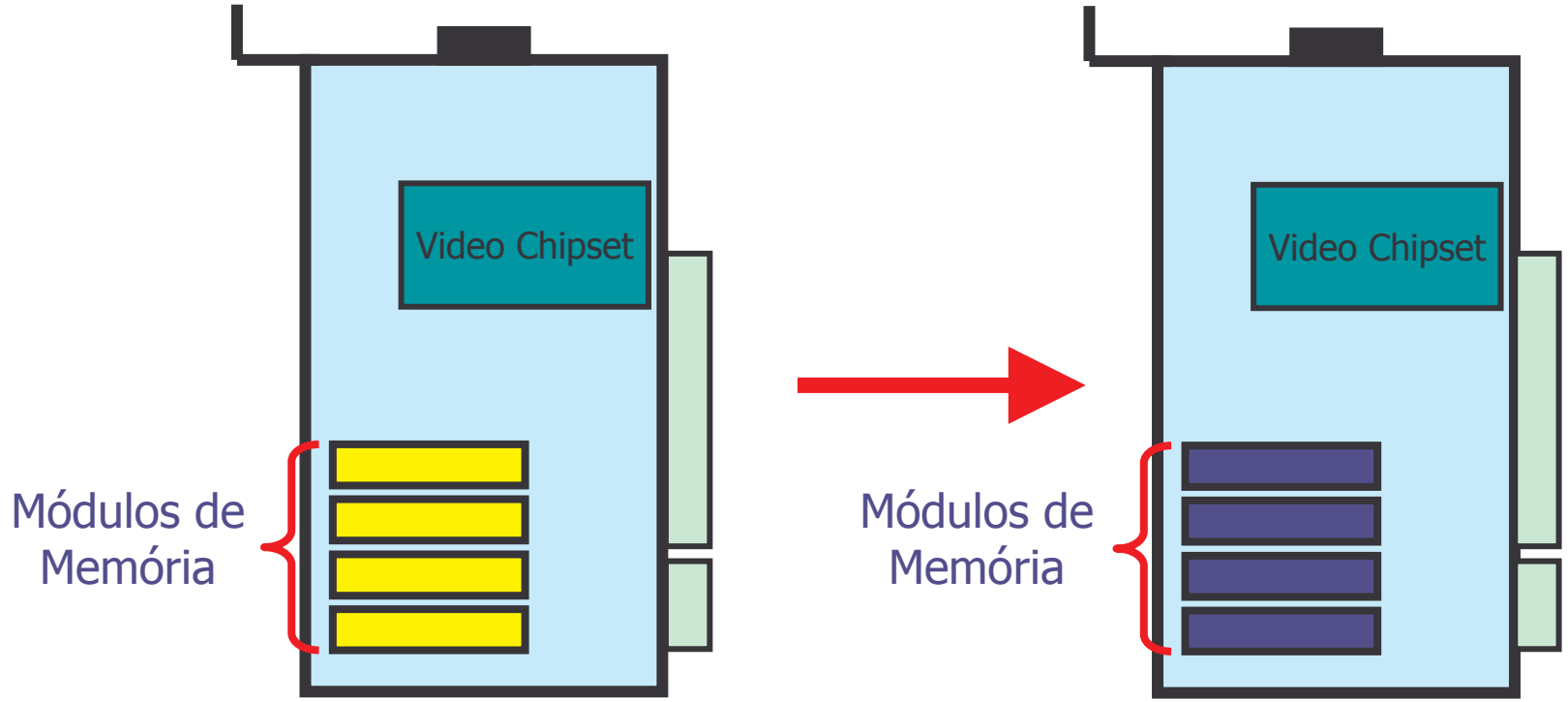
Motivação

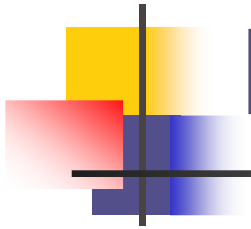


Fonte:
Notas de aula de John Lockwood - http://ipoint.vlsi.uiuc.edu/people/lockwood/lockwood.html
http://xnet.rrc.mb.ca/brianb/Installation/Video%20Install.htm

Motivação (cont.)

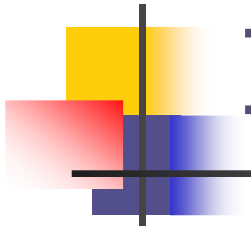
Duas marcas de memória:
- Amarela
- Azul





Especificação

- Para que as memórias das diferentes marcas sejam intercambiáveis precisamos que elas tenham a mesma especificação:
  - Tempo de resposta (10, 20, 50, ... ns)
  - Freqüência (100, 133, ... MHz)
  - Tensão (2.5, 3.3, ... Volts)
  - ...

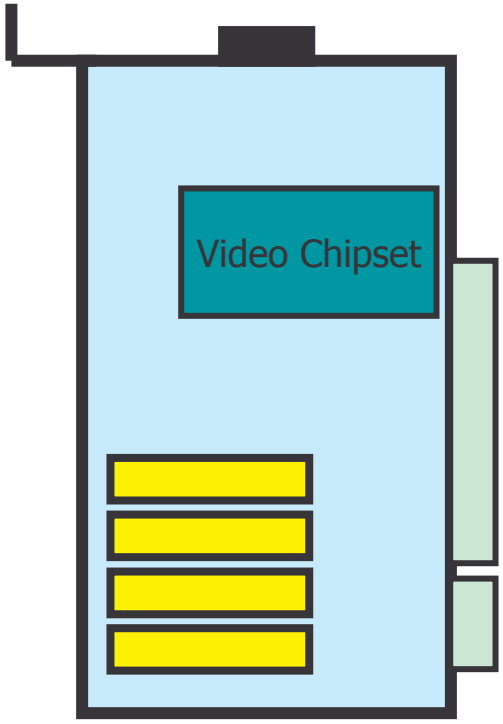


Implementação

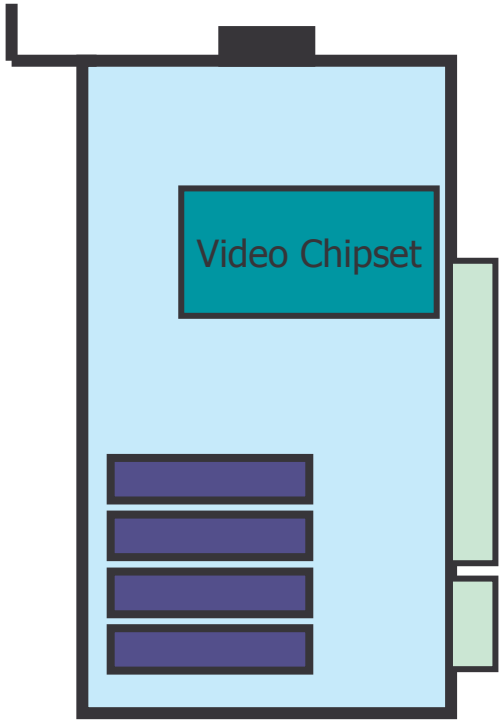
- Duas marcas: amarela e azul
- Ambas as marcas foram fabricadas utilizando-se a mesma especificação
- São duas implementações da mesma especificação
- Cada marca tem a liberdade de implementar os requisitos definidos na camada de especificação como quiser

Configuração

Configuração 1
Usa módulos da marca amarela

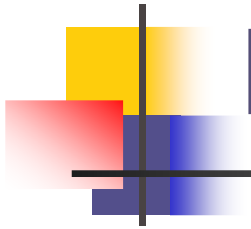


Configuração 2
Usa módulos da marca azul



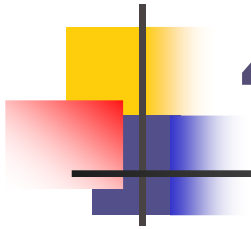
- Cada configuração pode ter um comportamento diferente ainda que os módulos sigam a mesma especificação

- Cada configuração é um sistema completo conforme descrito pela especificação



Motivação

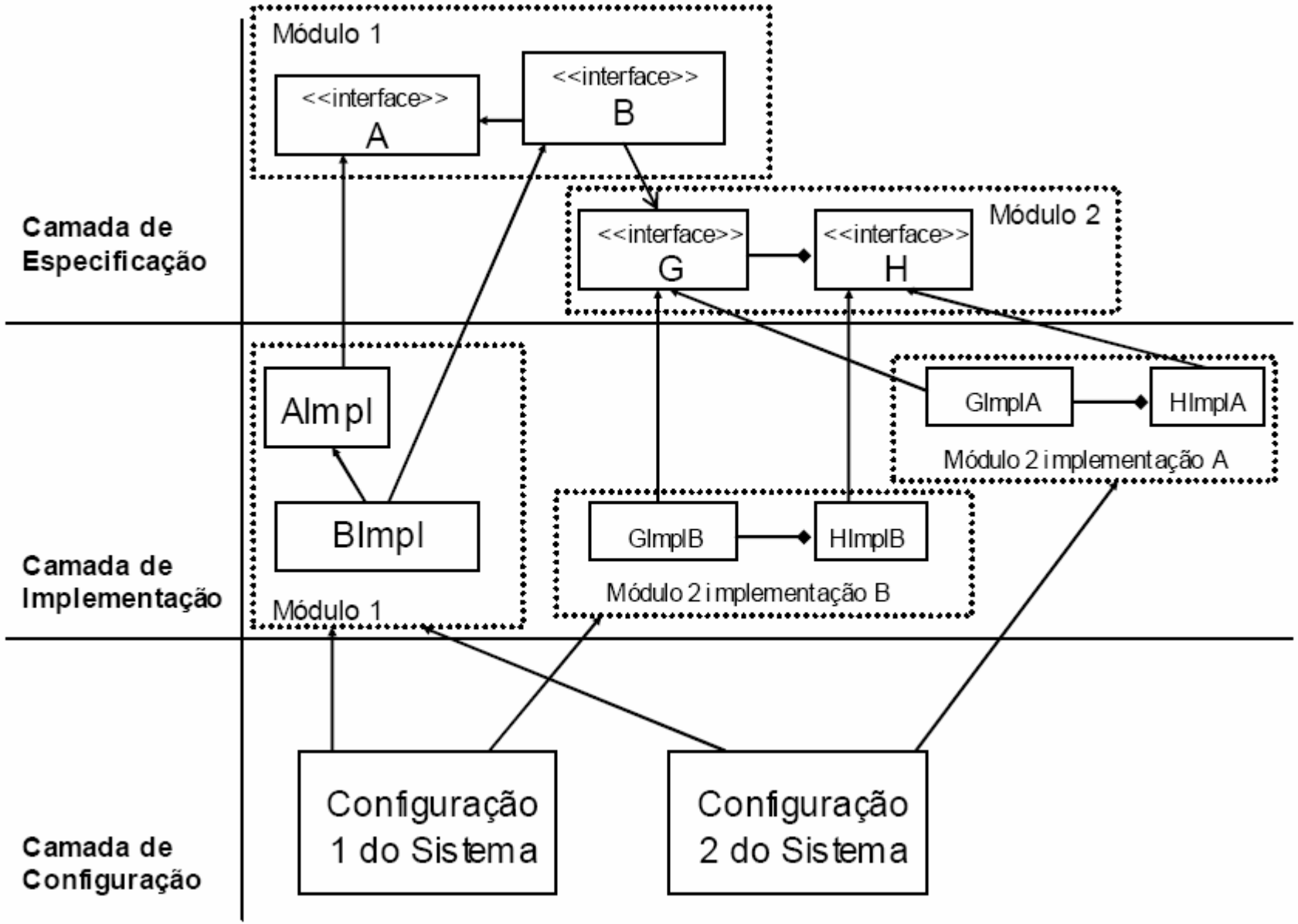
- Dependência entre implementações X dependência entre interfaces
  - Alto acoplamento
  - Sistema amarrado (alto custo para alteração de algum módulo)
  - Imagine se, no projeto da placa de vídeo, o chipset dependesse da implementação da memória

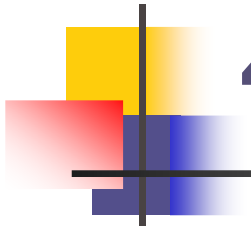


4-ação - Aplicabilidade

- Use 4-ação quando se deseja que:
  - os componentes do sistema possam ser facilmente trocados por outros que tenham a mesma interface
  - exista agilidade em tal troca
  - o código final seja legível, com módulos, interfaces e funções muito bem definidas
  - para que haja a possibilidade de que a troca de componentes seja feita dinamicamente.

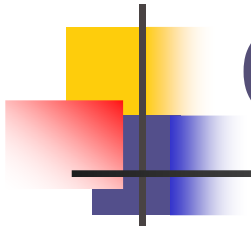
4-ação - Estrutura





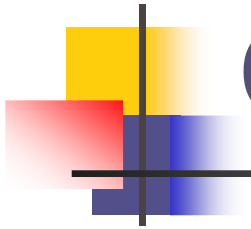
4-ação – Estrutura

- 3 camadas
  - Especificação
    - Contém todos os módulos e interfaces do sistema e como eles interagem entre si
  - Implementação
    - Contém todas as implementações dos módulos e classes concretas (que implementam as interfaces especificadas na camada de especificação)
  - Configuração
    - Define quais implementações o sistema deve utilizar para funcionar



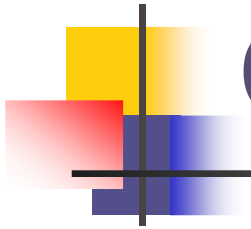
Camada de especificação

- Especifica todas as interfaces e módulos do sistema
- Especifica como os módulos e interfaces devem interagir
- Especifica o comportamento do sistema
- Classes concretas são proibidas nesta camada
- Define **o que** vai ser feito [2] e os contratos entre as classes.



Camada de implementação

- Contém as classes concretas que implementam as interfaces e módulos do sistema
- Define **como** as coisas são feitas [2]
- Referências para classes de outros módulos são **proibidas** (isso evita o acoplamento e a dependência entre módulos da camada de implementação deixando a dependência apenas na camada de especificação)
- Referências sempre devem ser feitas utilizando-se a interface do objeto (definida na camada de especificação) e nunca a classe concreta

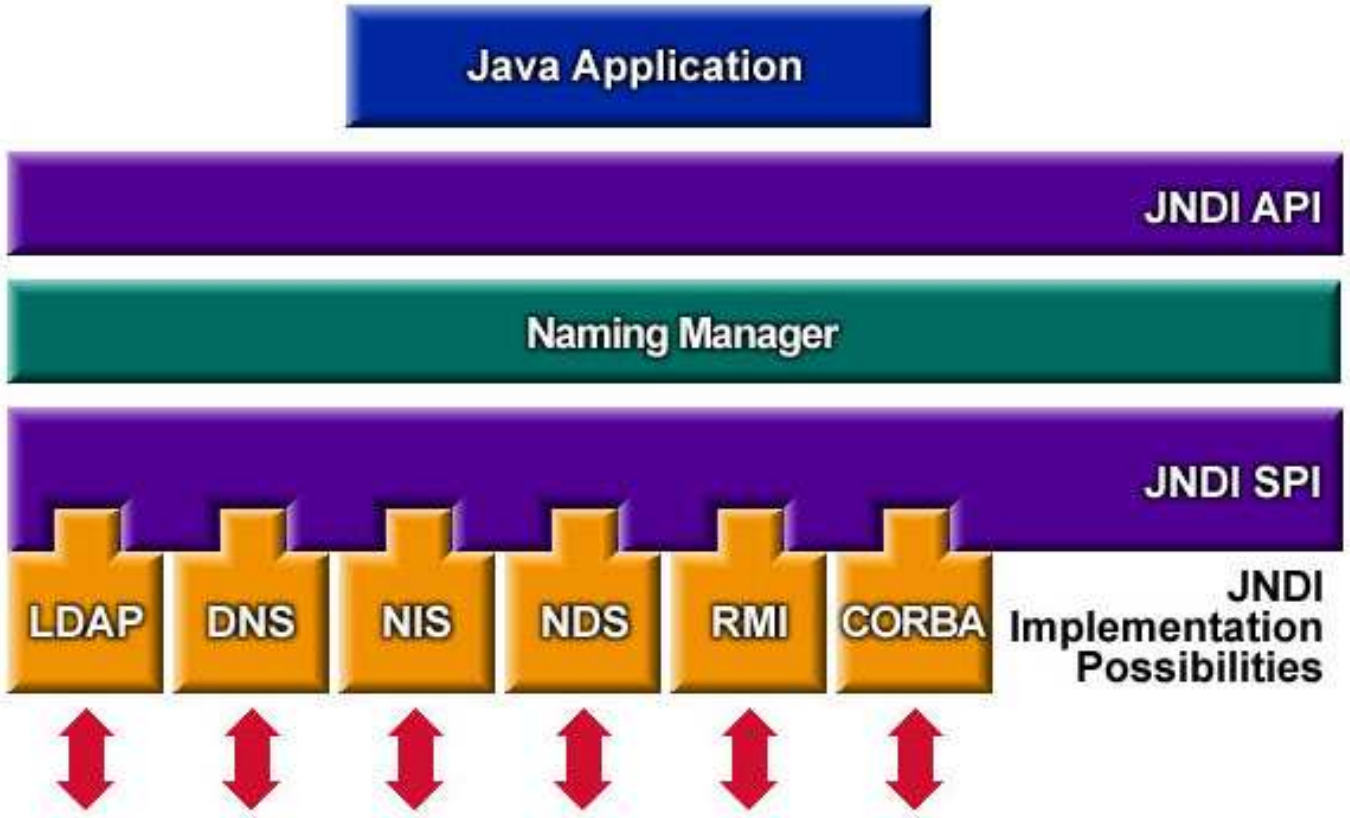


Camada de configuração

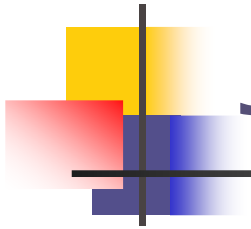
- Define quais das implementações contidas na camada de implementação serão usadas
- O ato de encaixar uma nova memória na placa de vídeo é análogo ao de fazer uma troca de configuração.
- Uma configuração pode depender de várias outras

Exemplo - JNDI

- Java Naming and Directory Interface
  - É uma interface única para acesso a diversos serviços de nomes e diretórios

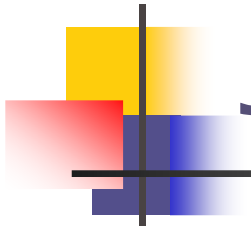


Fonte:
Java JNDI Tutorial - http://java.sun.com/products/jndi/tutorial/



JNDI

- Camada de especificação ~ API
- Camada de implementação ~ SPI
- Camada de configuração ~ jndi.properties, -D na linha de comando ou um parâmetro durante a inicialização:
  - java.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory
  - java.naming.provider.url=http://www.x.y.z/~user/NS_Ref

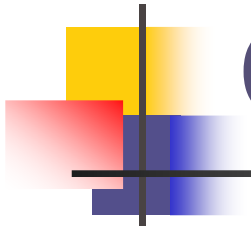


JNDI - Exemplo 1

```
/**
   * Aqui as variáveis de ambiente estão
   * ajustadas com valores “Desconhecidos”
   */
Context ctx = new InitialContext();
Object object = ctx.lookup(“NomeProcurado");
```

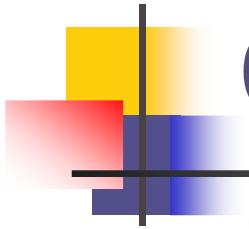
JNDI – Exemplo 2

```
class Lookup {
   public static void main(String[] args) {

        String name = args[0];

        try {
               Context ctx = new InitialContext();

               Object obj = ctx.lookup(name);

               System.out.println(name +
                      " is bound to: " + obj);

               ctx.close();
        } catch (NamingException e) {
               System.err.println("Problem looking up " +
                      name + ": " + e);
        }
   }
}
```



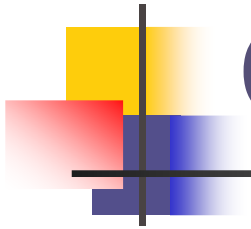
O desenvolvimento do sistema

- Fase 1 - Camada de especificação
  - Análise muito cuidadosa do comportamento que o sistema deve possuir e seu particionamento em módulos com função e comportamento muito bem definidos
  - Após o particionamento deve-se definir as interfaces que os módulos componentes do sistema utilizarão para interagir. Note que uma definição não muito boa nesta fase pode trazer sérias conseqüências durante a implementação das interfaces;
  - Definir precisamente **o que** o sistema faz.



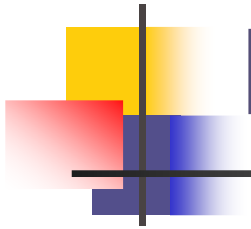
O desenvolvimento do sistema

- Fase 2 – Camada de implementação
  - Uma classe não pode de forma alguma depender diretamente de classes que constam nas implementações de outros módulos. Para corrigir isto deve-se apenas utilizar dependências para as interfaces de tais módulos
  - Aqui se deve focar na maneira que o sistema irá resolver um dado problema, ou seja, o aspecto algorítmico da solução
  - Talvez durante o desenvolvimento seja necessário, para testes, o desenvolvimento de *proxies* já que não se pode fazer referência alguma para implementações de outros módulos;
  - Em suma, nesta fase é definido um conjunto de maneiras, implementações, de **como** o sistema faz suas tarefas.



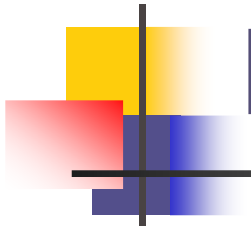
O desenvolvimento do sistema

- Fase 3 – Camada de configuração
  - Nesta fase são definidas que implementações serão utilizadas pelo sistema
  - Determinadas implementações são escolhidas baseadas em como elas efetuam o serviço definido pela camada de especificação
  - Em suma, é onde é feita a **"ligação"** entre as implementações dos módulos.



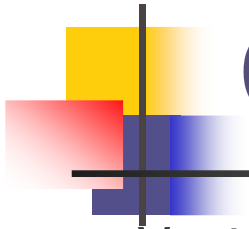
E os padrões de projeto?

- Eles acabam aparecendo automaticamente. Fica praticamente impossível trabalhar sem eles.
- Exemplo mais gritante é o da fábrica abstrata (Abstract Factory). Qualquer objeto do sistema deve ter uma fábrica que deve ser especificada, implementada e configurada.



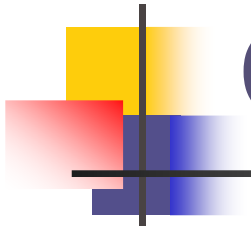
Padrões de projeto

- Estes são alguns dos padrões que aparecem durante o desenvolvimento do sistema
  - Abstract Factory, Factory Method, Builder – todo objeto deve ser criado por uma chamada a um método de instância de algum objeto definido na camada de configuração.
  - Adapter – uma nova implementação de alguma funcionalidade aparece no mercado. Crie um adaptador e troque a configuração.
  - Strategy – troca de implementações
  - Proxy – aparece durante a criação dos testes



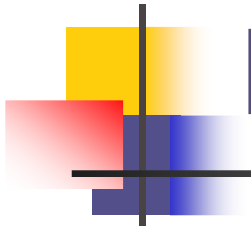
Considerações

- Vantagens
  - **Baixo** acoplamento entre módulos
  - **Fácil** reutilização de módulos entre sistemas diferentes, já que os módulos têm sua interface e funcionalidade muito bem definidas na camada de especificação
  - Troca entre diferentes implementações de módulos do sistema é realizada de forma **fácil**, bastando para isso uma única alteração na camada de configuração (tipicamente uma única classe ou arquivo)
  - O entendimento do sistema se dá de uma maneira mais **tranqüila**, em particular, o sistema como um todo pode ser compreendido apenas olhando para a camada de especificação
  - Avaliações de desempenho de diferentes implementações de uma dada especificação são feitas de maneira mais **rápida**
  - **Facilidade** de desenvolvimento distribuído, em diferentes equipes, uma em cada módulo da aplicação, por exemplo, após a especificação das interfaces
  - Possibilidade de troca de configuração **dinâmica**



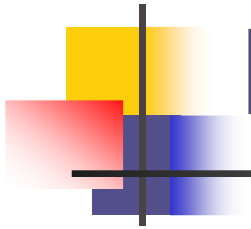
Considerações

- Desvantagens
  - O número de classes do sistema é mais do que duplicado. Para qualquer classe de implementação é necessária uma interface para representá-la, assim como uma outra interface para representar sua AbstractFactory e a implementação da mesma
  - Aumento no tempo de análise do software
  - Mudanças na camada de especificação causam mudanças gigantescas em todo o sistema
  - As regras são, em alguns momentos, incovenientes.



Bibliografia

- Gamma, et al., Padrões de Projeto (*Design Patterns*), 1995
- Conde, Francesquini, 4-ação, 2002
- Feofiloff, http://www.ime.usp.br/~pf/algoritmos/aulas/docu.htm



Perguntas

???