Modelos, Métodos e Técnicas de Engenharia de Software

Prof.: Sônia Aparecida Santana

una

Padrões de Projetos

- *Padrões são um repertório de soluções e princípios que ajudam os desenvolvedores a criar software e que são codificados em um formato estruturado consistindo de*
  - *Nome*
  - *Problema que soluciona*
  - *Solução do problema*
- *O objetivo dos padrões é codificar conhecimento (knowing) existente de uma forma que possa ser reaplicado em contextos diferentes*

Por que Aprender Padrões

- *Aprender com a experiência dos outros*
  - *Identificar problemas comuns em engenharia de software e utilizar soluções testadas e bem documentadas*
  - *Utilizar soluções que têm um nome: facilita a comunicação, compreensão e documentação*
- *Aprender a programar bem com orientação a objetos*
  - *Os 23 padrões de projeto "clássicos" utilizam as melhores práticas em OO para atingir os resultados desejados*
- *Desenvolver software de melhor qualidade*
  - *Os padrões utilizam eficientemente polimorfismo, herança, modularidade, composição, abstração para construir código reutilizável, eficiente, de alta coesão e baixo acoplamento*

Por que Aprender Padrões

- *Vocabulário comum*
  - *Faz o sistema ficar menos complexo ao permitir que se fale em um nível mais alto de abstração*
- *Ajuda na documentação e na aprendizagem*
  - *Conhecendo os padrões de projeto torna mais fácil a compreensão de sistemas existentes*
  - *"As pessoas que estão aprendendo POO freqüentemente reclamam que os sistemas com os quais trabalham usam herança de forma complexa e que é difícil de seguir o fluxo de controle. Geralmente a causa disto é que eles não entendem os padrões do sistema" [GoF]*
  - *Aprender os padrões ajudam um novato a agir mais como um especialista*

Elementos de um Padrão

- *Nome*
- *Problema*
  - *Quando aplicar o padrão, em que condições?*
- *Solução*
  - *Descrição abstrata de um problema e como usar os elementos disponíveis (classes e objetos) para solucioná-lo*
- *Conseqüências*
  - *Custos e benefícios de se aplicar o padrão*
  - *Impacto na flexibilidade, extensibilidade, portabilidade e eficiência do sistema*

Padrões Grasp

- Introduzidos por Craig Larman em seu livro "Applying UML and Patterns"
- GRASP: **G**eneral **R**esponsibility and **A**ssignment **S**oftware **P**atterns
  - Os padrões GRASP descrevem os princípios fundamentais da atribuição de responsabilidades a objetos, expressas na forma de padrões

Padrões Grasp

- Os diferentes padrões e princípios utilizados no GRASP são: controller (controlador), creator (criador), indirection (indireção), information expert (especialista na informação), alta coesão, baixo acoplamento, polimorfismo, pure fabrication (fabricação/invenção pura) e protected variations (variações protegidas).

Padrões Grasp

- Todos esses padrões respondem a algum problema, e esses problemas são comuns a quase todos os projetos de desenvolvimento de software.
- Essas técnicas não foram inventadas a fim de criar novas formas de trabalho, mas para melhor documentar e padronizar os antigos e amplamente praticados princípios de programação em padrões orientado a objetos.

Padrões Clássicos (GoF)

- *O livro "Design Patterns" (1994) de Erich Gamma, John Vlissides, Ralph Jonhson e Richard Helm, descreve 23 padrões de design [2]*
  - *São soluções genéricas para os problemas mais comuns do desenvolvimeto de software orientado a objetos*
  - *O livro tornou-se um clássico na literatura orientada a objeto e continua atual*
  - *Não são invenções. São documentação de soluções obtidas através da experiência. Foram coletados de experiências de sucesso na indústria de software, principalmente de projetos em C++ e SmallTalk*
  - *Os quatro autores, são conhecidos como "The Gang of Four", ou GoF*

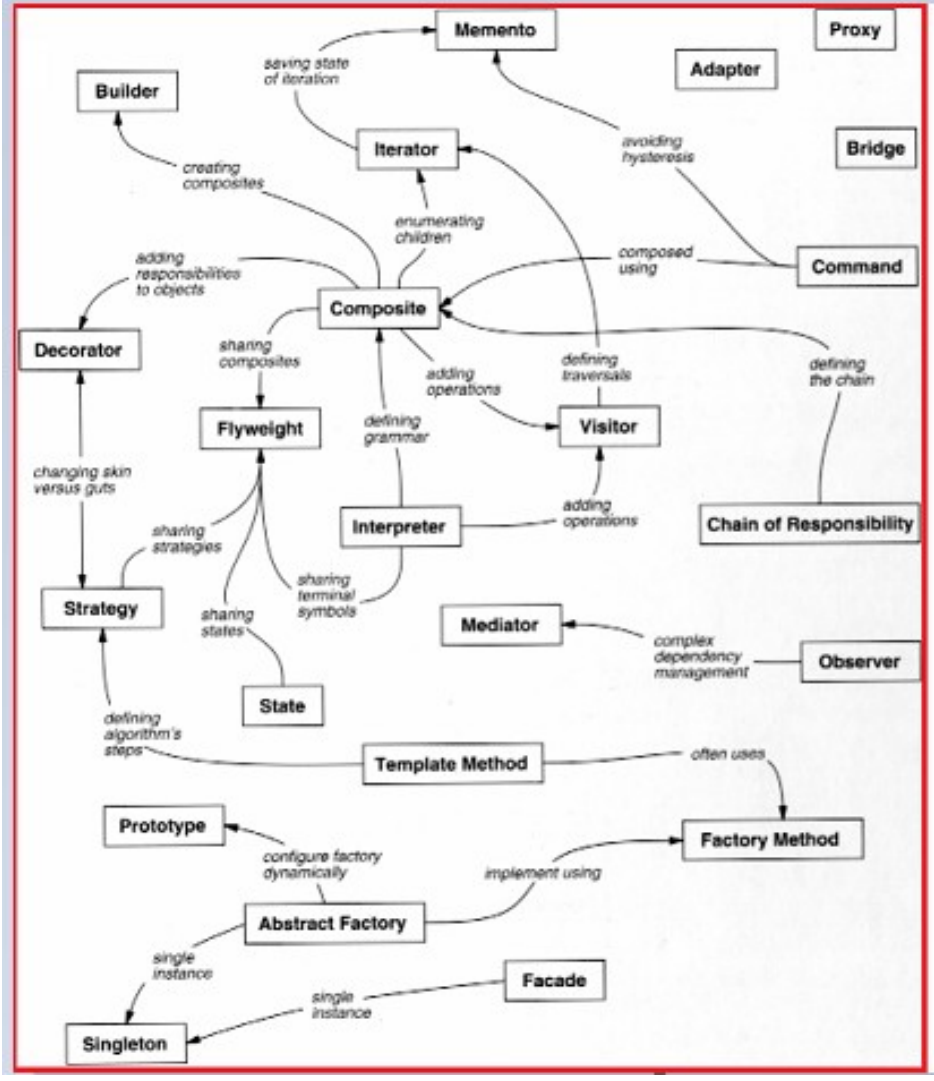
Padrões GoF (propósito)

		Propósito		
		1. Criação	2. Estrutura	3. Comportamento
Escopo	Classe	Factory Method	Class Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Object Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Padrões GoF (Intenção)

Intenção	Padrões
1. Interfaces	Adapter, Facade, Composite, Bridge
2. Responsabilidade	Singleton, Observer, Mediator, Proxy, Chain of Responsibility, Flyweight
3. Construção	Builder, Factory Method, Abstract Factory, Prototype, Memento
4. Operações	Template Method, State, Strategy, Command, Interpreter
5. Extensões	Decorator, Iterator, Visitor

- Relacionamento entre os 23 tipos de padrões de projeto



Atividade em Equipe

- Para cada padrão de projeto clássico mencionado:
  - Nome do Padrão de Projeto
  - Identificar o problema que ele resolve
  - Identificar a solução
    - Ilustrar a solução

Instruções

- Definição dos grupos (máximo de 3 alunos);
- Definição dos temas;
- Entregar material desenvolvido;
- Apresentar a solução para debate (10 minutos);

Temas

- Abstract Factory
- Builder
- Factory Method
- Singleton
- Prototype

Temas

- Proxy
- Adapter
- Facade
- Decorator

Temas

- Strategy
- Observer
- Template Method
- Iterator
- Visitor

Exemplo

- Padrão de Projeto *Singleton*

Singleton

- Problema que resolve:
  - É um padrão de projeto criacional que permite a você garantir que uma classe tenha apenas uma instância, enquanto provê um ponto de acesso global para essa instância.
  - Fornece um ponto de acesso global para aquela instância

Singleton

- Solução:
  - Fazer o construtor padrão privado, para prevenir que outros objetos usem o operador new com a classe singleton.
  - Criar um método estático de criação que age como um construtor. Esse método chama o construtor privado por debaixo dos panos para criar um objeto e o salva em um campo estático. Todas as chamadas seguintes para esse método retornam o objeto em cache.

Singleton: Exemplo

```
void f() {
  Logger log = new Logger();
  log.println("Executando f");
  ...
}
void g() {
  Logger log = new Logger();
  log.println("Executando g");
  ...
}
void h() {
  Logger log = new Logger();
  log.println("Executando h");
  ...
}
```

Construtor default privado

Atributo estático

```
class Logger {

  private Logger() {} // proíbe clientes chamar new Logger()

  private static Logger instance; // instância única

  public static Logger getInstance() {
    if (instance == null) // 1a vez que chama-se getInstance
      instance = new Logger();
    return instance;
  }

  public void println(String msg) {
    // registra msg no console, mas poderia ser em arquivo
    System.out.println(msg);
  }
}
```

Singleton: Exemplo

Construtor default privado

```
class Logger {

  private Logger() {} // proíbe clientes chamar new Logger()

  private static Logger instance; // instância única

  public static Logger getInstance() {
    if (instance == null) // 1a vez que chama-se getInstance
      instance = new Logger();
    return instance;
  }

  public void println(String msg) {
    // registra msg no console, mas poderia ser em arquivo
    System.out.println(msg);
  }
}
```

Atributo estático

```
void f() {
  Logger log = Logger.getInstance();
  log.println("Executando f");
  ...
}

void g() {
  Logger log = Logger.getInstance();
  log.println("Executando g");
  ...
}

void h() {
  Logger log = Logger.getInstance();
  log.println("Executando h");
  ...
}
```