WebQuest

Aula Semana 09

Mais Sobre Padrões de Projeto Básicos:

Static Factory Method, Null Object,

Hook Methods e Hook Classes

**Introdução**

O objetivo deste WebQuest é consolidar o entendimento e implementação dos seguintes padrões básicos: Static Factory Method, Null Object, Hook Methods e Hook Classes.

Um padrão é básico se ele é usado isoladamente ou como parte de outros padrões de projeto do livro GoF [ Recurso Secundário 1].

Recomendo comprar o livro do Prof. Guerra [Recurso Secundário 2].

**Tarefa**

Conhecer, ver exemplos e exercitar o uso dos padrões de projeto básicos Static Factory Method, Null Object, Hook Methods e Hook Classes.

**Processo**

1. [Com seu colega do lado/da frente/de trás]
   1. [05min] [Recurso Primário 1] Definir o que é e para que serve o padrão básico Static Factory Method, nomes alternativos e estrutura.

**RESP:** Static Factory Method são métodos estáticos que retornam instâncias da classe, em geral para substituir o construtor, que passa a ser private/protected. Um exemplo de estrutura abaixo:

public class Classe {

private Classe() {}

public static Classe criar(par a, par b) {

return new Classe(par a, par b, par c)

}

}

O parâmetro c pode ser tratado como um valor default ou combinação de a e b.

* 1. [10min] Dada a classe RandonIntGenerator, que gera números aleatórios entre um mínimo e um máximo, implemente-a passo-a-passo:

**public** **class** RandonIntGenerator {

**public** **int** next() {**...**}

**private** **final** **int** min;

**private** **final** **int** max;

}

Como os valores min e max são final, eles devem ser inicializados na declaração ou via construtor. Vamos inicializar por meio de um construtor!

**public** RandonIntGenerator(**int** min, **int** max) {

**this**.min = min;

**this**.max = max;

}

Crie um novo construtor, supondo que o valor min é fornecido e o valor max é o maior valor inteiro do Java (Integer.MAX\_VALUE)!

**public** RandonIntGenerator(**int** min) {

**this**.min = min;

**this**.max = Integer.*MAX\_VALUE*;

}

Crie um novo construtor, supondo que o valor max é fornecido e o valor min é o menor valor inteiro do Java (Integer.MIN\_VALUE)!

**public** RandonIntGenerator(**int** max) {

**this**.min = Integer.*MIN\_VALUE*;

**this**.max = max;

}

Como resolver este problema?

**RESP:** Basta utilizar o padrão Static Factory Method, assim podemos criar outros construtores com os parâmetros default Integer.MAX\_VALUE e Integer.MIN\_VALUE

public class RandomIntGenerator {

private final int min;

private final int max;

private RandomIntGenerator(int min, int max) {

this.min = min;

this.max = max;

}

public static RandomIntGenerator between(int max, int min) {

return new RandomIntGenerator(min, max);

}

public static RandomIntGenerator biggerThan(int min) {

return new RandomIntGenerator(min, Integer.MAX\_VALUE);

}

public static RandomIntGenerator smallerThan(int max) {

return new RandomIntGenerator(Integer.MIN\_VALUE, max);

}

public int next() {...}

}

* 1. [05min] Melhore a legibilidade do código abaixo:

public class Foo{

public Foo(boolean withBar){

//...

}

}

//...

// What exactly does this mean?

**Foo foo = new Foo(true);**

// You have to lookup the documentation to be sure.

// Even if you remember that the boolean has something to do with a // Bar, you might not remember whether it specified withBar or

// withoutBar.

Solução:

Public class Foo{

private Foo(boolean withBar) {

…

}

public static Foo createWithBar() {

Return new Foo(true);

}

public static Foo createWithoutBar() {

Return new Foo(false);

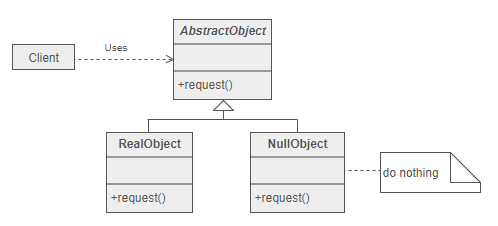
}

}

* 1. [Exercício para Casa] Em [Recurso Primário 1], estende-se o gerador de inteiro do item b) para suportar inteiro, Double, Long e String. Mostrar uma implementação com static factory methods que resolva essa situação

1. [Com outro colega do lado/da frente/de trás][Mudar de local, se for preciso]
   1. [05min] Definir o que é e para que serve o padrão básico Null Object, nomes alternativos e estrutura.

**RESP:** Null Object existe para encapsular a inexistência de um objeto providenciando um substituto que oferece valores default que encaixem no comportamento de Null. Também chamado de Special Case no catálogo “EEA” do Fowler. Estrutura:



* 1. [10min] Dada a classe RealCustomer abaixo, projetar e implementar um exemplo de aplicação simples, mostrando o antes (sem o padrão) e o depois (com o padrão) quando alguns clientes reais existem no repositório de clientes e outros ainda não fazem parte dele! Simular tudo o que for necessário para exemplificar a necessidade do uso do Null Object, inclusive o repositório de clientes!

public class RealCustomer {

public RealCustomer(String name) {

this.name = name;

}

@Override

public String getName() {

return name;

}

@Override

public boolean isNil() {

return false;

}

}

**RESP:**

**public** **abstract** **class** RealCustomer {

**protected** String name;

**public** **abstract** String getName();

**public** **abstract** **boolean** isNil();

}

**public** **class** NullRealCustomer **extends** RealCustomer{

@Override

**public** String getName() {

**return** **“Não existe esse nome.”**;

}

@Override

**public** **boolean** isNil() {

**return** **true**;

}

}

**public** **class** NonNullRealCustomer **extends** RealCustomer {

**public** NonNullRealCustomer(String name) {

**this**.name = name;

}

@Override

**public** String getName() {

**return** name;

}

@Override

**public** **boolean** isNil() {

**return** **false**;

}

}

**public** **class** Exemplo {

**public** **static** **final** String[] ***names*** = {"Dick", "Lindo", "Bulldogs"};

**public** **static** RealCustomer getCostumer(String name) {

**for** (**int** i=0; i < ***names***.length; i++) {

**if**(***names***[i].equalsIgnoreCase(name)) {

**return** **new** NonNullRealCustomer(name);

}

}

**return** **new** NullRealCustomer();

}

}

**public** **class** Demo {

**public** **static** **void** main(String[] args) {

RealCustomer rc1 = Exemplo.*getCostumer*("Dick");

RealCustomer rc2 = Exemplo.*getCostumer*("Papagaio");

System.***out***.println(rc1.getName());

System.***out***.println(rc2.getName());

}

}

O Output obtido através da simulação do arquivo Demo.java:

Dick

Não existe esse nome.

1. [Com outro colega do lado/da frente/de trás][Mudar de local, se for preciso]
   1. [05min] Definir o que é e para que serve o padrão básico Hook Method, nomes alternativos e estrutura. [Recursos Primários 3 e 4]

**RESP:** HookMethods são utilizados para mudar o comportamento de um método principal de uma superclasse nas suas subclasses. O método principal chama um HookMethod() durante sua execução, para cada subclasse, o HookMethod() é implementado de acordo com a necessidade, dessa forma mudando o comportamento de MetodoPrincipal().

* 1. [10min] Pesquisar no [Recursos Primários 3 e 4] ou em qualquer outra fonte e projetar e implementar um exemplo de aplicação simples, mostrando o antes (sem o padrão) e o depois (com o padrão)!

**RESP:**

**public** **class** Algorithm {

**public** **void** templateMethod() {

:

.

hookMethod();

.

:

}

**public** **void** hookMethod() {

// default implementation

}

}

**public** **class** RefinedAlgorithm **extends** Algorithm {

**public** **void** hookMethod() {

// refined implementation

}

}

1. [Com outro colega do lado/da frente/de trás][Mudar de local, se for preciso]
   1. [07min] Diferencie hook method de hook class, começando com um exemplo não operacional em Java que implementa um hook method e transforme-o em hook class.

O exemplo do item 3b. poderia ser estendido para que inúmeras classes RefinedAlgorithm1, RefinedAlgorithm2 .. herdassem de Algorithm. Para isso serve a aplicação de HookClass, pois apenas uma HookClass seria herdeira de Algorithm e as classes RefinedAlgorithm1 .. seriam herdeiras de HookClass.

**RESP:**

**public** **class** Algorithm {

**public** **void** templateMethod() {

:

.

component.hookMethod();

.

:

}

}

**public** **class** HookClass **extends** Algorithm {

**public** **void** hookMethod() {

// default implementation

}

}

**public** **class** RefinedAlgorithm1 **extends** HookClass {

**public** **void** hookMethod() {

// refined implementation

}

}

**Recursos Primários**

1. [Static Factory Method] <http://jlordiales.me/2012/12/26/static-factory-methods-vs-traditional-constructors/>

(former link: [http://jlordiales.wordpress.com](http://jlordiales.wordpress.com/2012/12/26/static-factory-methods-vs-traditional-constructors/))

1. [Null Object] https://sourcemaking.com/design\_patterns/null\_object
2. PDF com arquivo do link desativado <https://www.cs.oberlin.edu/~jwalker/nullObjPattern/> [TIDIA – Semana 09]
3. [Hook Methods 1] Hook Methods—Livro Guerra [TIDIA – Semana 09]
4. [Hook Methods 2] <http://c2.com/cgi/wiki?HookMethod>
5. [Hook Classes] Hook Classes—Livro Guerra [TIDIA – Semana 09]

**Recursos Secundários**

1. Gamma, Erich; Richard Helm, Ralph Johnson, and John Vlissides (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. [“Gang of Four” or GoF]
2. Eduardo Guerra. Design Patterns com Java: Projeto Orientado a Objetos Guiado por Padrões. São Paulo: Casa do Código, 2013. [ISBN 978-85-66250-11-4][e-Book R$ 29,90]
3. Null Object apresentado como refatoração: <http://www.refactoring.com/catalog/introduceNullObject.html>
4. Null Object é chamdo de “Special Case” no catalogo “EAA” do Fowler: <http://martinfowler.com/eaaCatalog/specialCase.html>