

Instituto Federal de Educação, Ciência e Tecnologia do Espírito Santo – Campus  
Serra  
Bacharel em Sistemas de Informação  
Italo Lourenço Trindade

## TRABALHO 3 POO2

Italo Lourenço Trindade

## TRABALHO 3 POO2

Trabalho apresentado ao Instituto Federal do Espírito Santo – Campus Serra como parte a das exigências da disciplina Programação Orientada a Objetos 2 do Curso de Bacharelado em Sistemas de Informação sob a orientação do professor Paulo como avaliação parcial para aprovação.

Serra / 2015

## Sumário

Mini Mundo.....	4
Explicação do Uso dos Padrões:.....	8
MVC:.....	12
Diagrama de Classe:.....	13
Discussão sobre o uso dos Padrões:.....	14

## **Desenvolvimento**

### **Mini Mundo**

Dois estudantes do IFES foram contratados por jogadores profissionais para desenvolver um simulador do jogo “Age of Empires” , com isso o simulador iria simular uma partida, uma guerra entre as civilizações.

O jogo basicamente consiste em colocar duas nações uma contra a outra, onde cada nação monta um conjunto de guerreiros para representa-la.

Os guerreiros são divididos em dois grupos os de defesa que são os defensores e os guerreiros de ataque que são os ofensores.

Para esse simulador do jogo, existem 3 possíveis nações a serem escolhidas:

- China: suas unidades de ataque, em geral, não são muito poderosas. As unidades de defesa têm propriedades bem interessantes.
- Japão: suas unidades de ataque são poderosas as de defesa nem tanto.
- Índia: suas unidades de ataque são medianas, mas possuem ótima defesa.

Basicamente serão feitas 4 filas de duelos:

- 1 fila de defesa e 1 de ataque para a nação1
- 1 fila de defesa e 1 de ataque para a nação2

A fila de ataque da nação1 ataca a fila de defesa da nação2.

A fila de ataque da nação2 ataca a fila de defesa da nação1.

A escolha da nação que ataca primeiro é feita por sorteio. Um ataque de uma nação implica em cada ofensor atacar 1 vez. Os ataques são feitos na fila de defensores da nação adversária.

Os ofensores atacam sucessivamente um defensor até que ele seja eliminado, uma vez que isso ocorra o próximo defensor da fila de defesa entrará em sua vez.

Depois que atacam os ofensores voltam para o final da fila de ofensores.

O jogo acaba se uma nação não tem mais ofensores ou defensores, ou seja, se um fila não tem mais guerreiros a nação perde.

### **Desenvolvimento**

Primeiramente é necessário definir o que é um Guerreiro. Um Guerreiro é alguém que luta, podendo ser ofensor ou defensor e possui obrigatoriamente:

- **Energia:** que deve ser inicializada em 100 no momento da criação do guerreiro.

Guerreiros morrem quando sua energia fica menor ou igual a 0.

A habilidade de atacar é definida no ofensor, mas o ofensor não sabe como atacar (sempre será um tipo de ofensor que terá essa habilidade).

A seguir apresentaremos os Guerreiros de cada nação:

### **Chineses:**

#### **Ofensores:**

- 1) **Chun Ku:** os Chun Ku são arqueiros chineses. Retiram 5 pontos de qualquer defensor indiano e 10 pontos de qualquer defensor japonês.
- 2) **Gun Te:** os Gun te são guerreiros de grandes espadas. São especialmente bons contra a defesa japonesa, retirando 20 pontos de qualquer defensor. Quando atacam defensores indianos tiram 1 ponto mas morrem em seguida.

#### **Defensores:**

- 1) **Mangal de defesa:** é um boneco mecânico de defesa automática. Os mangais de defesa tiram 2 pontos de qualquer atacante.
- 2) **Montor do escudo:** os montores são guerreiros de grandes escudos cuja energia inicial é de 150 (é o único guerreiro que redefine esse valor). Quando os montores morrem eles levam consigo (matam) o guerreiro ofensor que os atacou.
- 3) **Mirk o conversor:** se atacados por Samurais os convertem em guerreiros Gun Te e colocam na fila de atacantes. Não sofrem qualquer dano de Samurais.

### **Japoneses:**

#### **Ofensores:**

- 1) **Samurai:** guerreiros lendários japoneses. Qualquer defensor atacado perde 50 pontos, exceto Mirk o conversor.

2) **Ninja**: guerreiros sorrateiros japoneses. Qualquer defensor atacado perde 20 pontos.

**Defensores:**

- 1) **Tan tan**: os tan tan são guerreiros com escudos fixos nos braços. Quando morrem se transformam em ninjas.

**Indianos:**

**Ofensores:**

- 1) **Seak**: os seak são flexíveis unidades de ataque indianas, possuindo espada e arco. Os Seak retiram 25 pontos de qualquer guerreiro defensor atacado.

**Defensores:**

- 1) **Monge Leaf**: quando atacados por Ninjas ou Chun Kus recebem um escudo de ouro que os tornam inatacáveis por esses tipos de guerreiros ofensores, ou seja, ficam invulneráveis a Ninjas e Chun Kus.
- 2) **Monge Bomb**: quando atacados morrem, mas deixam o ofensor atacante com energia em 1 unidade.

O programa deverá ler 2 arquivos (nacao1.txt eacao2.txt) e montar as filas de ofensores e defensores de cada nação.

**Entrada de dados:**

A entrada de dados de um arquivo de nação deverá ter o seguinte formato:

<nome da nação>

Ofensores:

<tipo do ofensor>

...

<tipo do ofensor n>

Defensores:

<tipo do defensor>

...

<tipo do defensor n>

**Exemplo:**

Japão

Ofensores:

Samurai

Samurai

Samurai

Samurai

Defensores:

Tantan

Tantan

**Saída de dados:**

A nação vencedora foi: <nome da nação> (Japão, Índia ou China)

A nação perdedora foi: <nome da nação> (Japão, Índia ou China)

Acabaram os guerreiros <categoria> (Ofensores ou defensores).

**Exemplo:**

A nação vencedora foi: Japão

A nação perdedora foi: Índia

Acabaram os guerreiros Defensores.

## Explicação do Uso dos Padrões

### Adapter

A intenção do padrão Adapter é converter a interface de uma classe em outra interface, esperada por um determinado cliente. O Adapter permite que classes com interfaces incompatíveis trabalhem em conjunto. Logo o uso do Adapter é recomendável quando você deseja utilizar uma classe existente, mas a sua interface não corresponde a interface que necessita ou quando precisamos usar várias subclasses mas for impossível adaptar essas interfaces criando subclasses para cada uma. Sendo assim, não foi necessário o uso do padrão Adapter no trabalho pois nenhuma classe precisou ser adaptada para outra funcionalidade, o uso de subclasses foi o suficiente para resolver nossas necessidades.

### Decorator

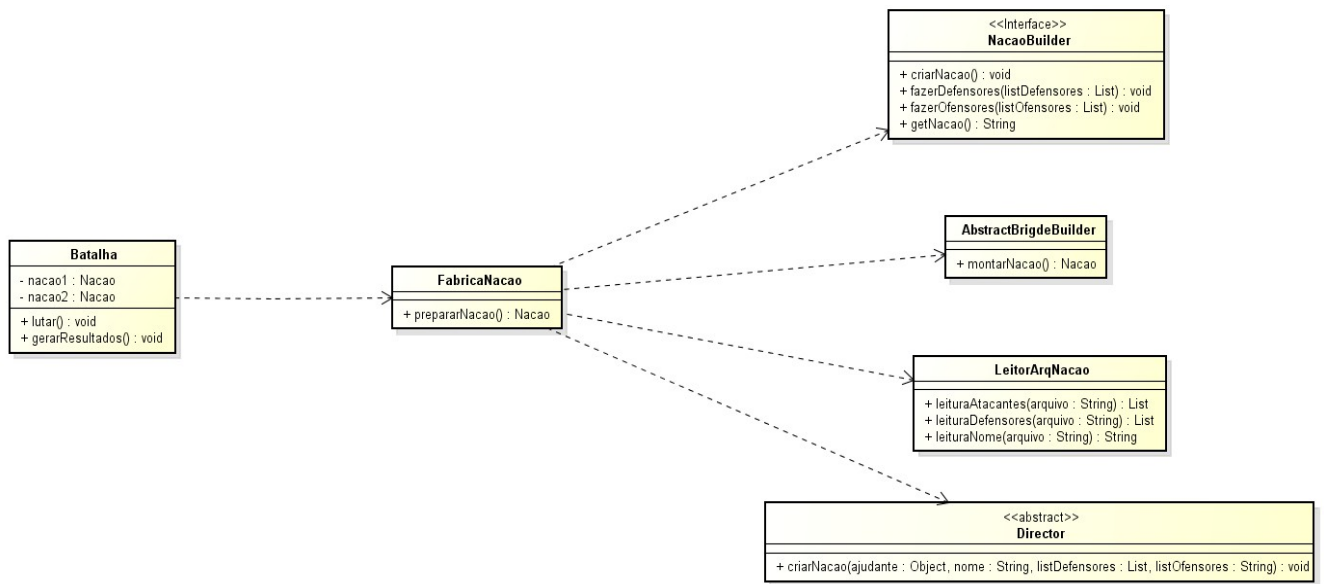
Não foi possível implementar o padrão decorator no trabalho sem que fizéssemos alterações no minimundo. O padrão decorator é aplicado quando temos que acrescentar responsabilidades a objetos individuais de forma dinâmica e transparente mas não podemos realizar essa extensão através de uso de subclasses, pois corre o risco de acontecer uma explosão de subclasses para suportar cada combinação. Para implementar o padrão, poderíamos ter feito a seguinte alteração no minimundo: Definir um herói para cada nação e equipamentos para ele, assim quando um equipamento fosse colocado em um herói ele receberia novos atributos e habilidades especiais. Mas como o prazo de entrega é curto não daria tempo de fazer uma alteração desse porte.

### Fachada

O padrão fachada fornece uma interface unificada para um conjunto de interfaces, manipulando interfaces de outros subsistemas, criando assim uma interface de alto nível tornando o uso dos subsistemas mais fácil. Logo, a fachada pode fornecer uma visão simples do sistema que é boa o suficiente para a maioria dos clientes. No trabalho por exemplo, o padrão fachada pode ser observado na classe “FabricaNacao” pois quando um cliente necessitar criar uma nação ele

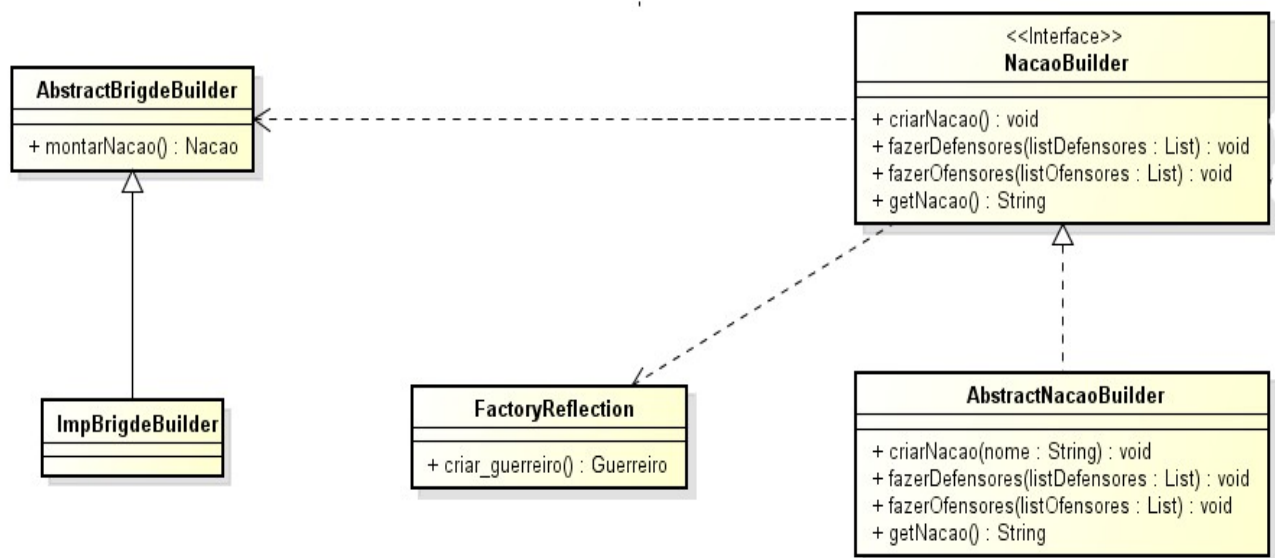


utilizará o método “prepararNacao” que implementa todo o processo de criação de uma nação (leitura de arquivo com os dados da nação, instanciar um objeto da classe Deus para manipular o builder de nação, instanciar o builder e a classe onde esta a sua implementação), assim a classe FabricaNacao torna a criação de uma nação mais simples.



## Brigde

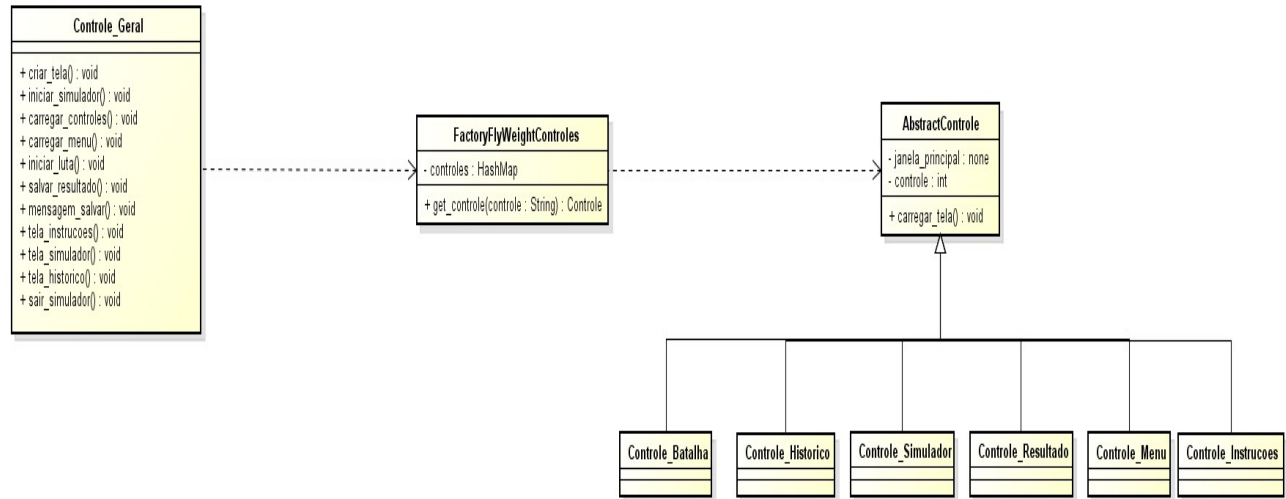
A intenção do padrão Bridge é desacoplar uma abstração da sua implementação, de modo que as duas possam variar independentemente. No trabalho usamos o Bridge para separar a abstração da classe Builder da sua implementação, pensando em atualizações futuras, poderiam existir novos passos para criar uma nação ou nações como uma maneira de criação diferente. Isso poderá ser feito com facilidade já que o padrão oferece uma boa extensibilidade.



## Flyweight

O flyweight tem o objetivo permitir a representação de um grande número de objetos de forma eficiente e econômica. Ele é aplicado onde existem diversas instâncias da mesma classe em memória. A solução então é reutilizar a mesma instância em todos os locais onde objetos semelhantes precisam ser utilizados. No trabalho o padrão flyweight foi aplicado nos controladores da interface gráfica. Sempre que o usuário solicita a troca entre telhas o programa faria a instanciação do controlador da página requerida para então abrir a página, gerando um desperdício de memória. Com o flyweight todos os controladores já estão instanciados em um dicionário que é atributo da classe

**“FactoryFlyweightControles”** é sempre que o usuário precisa trocar de tela, a classe **“Controle\_Geral”** solicita através do método **“get\_controle( String controle)”** o controlador necessário para atender o pedido do usuário, fazendo assim uma grande economia de memória.



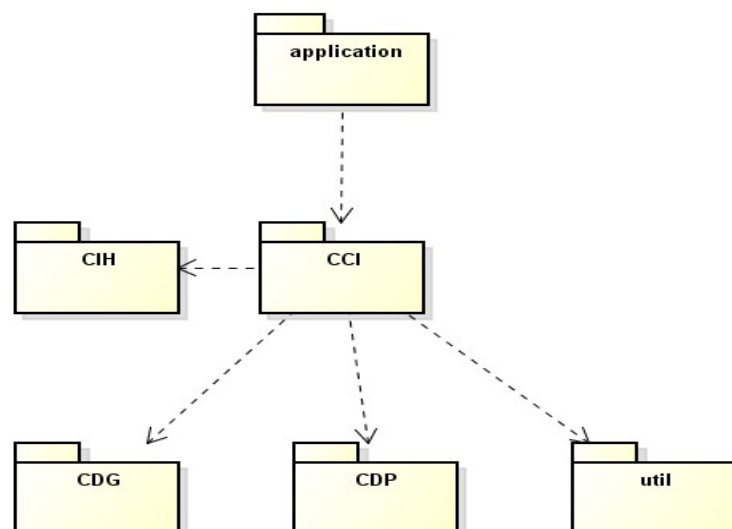
## Descrição da Utilização do MVC



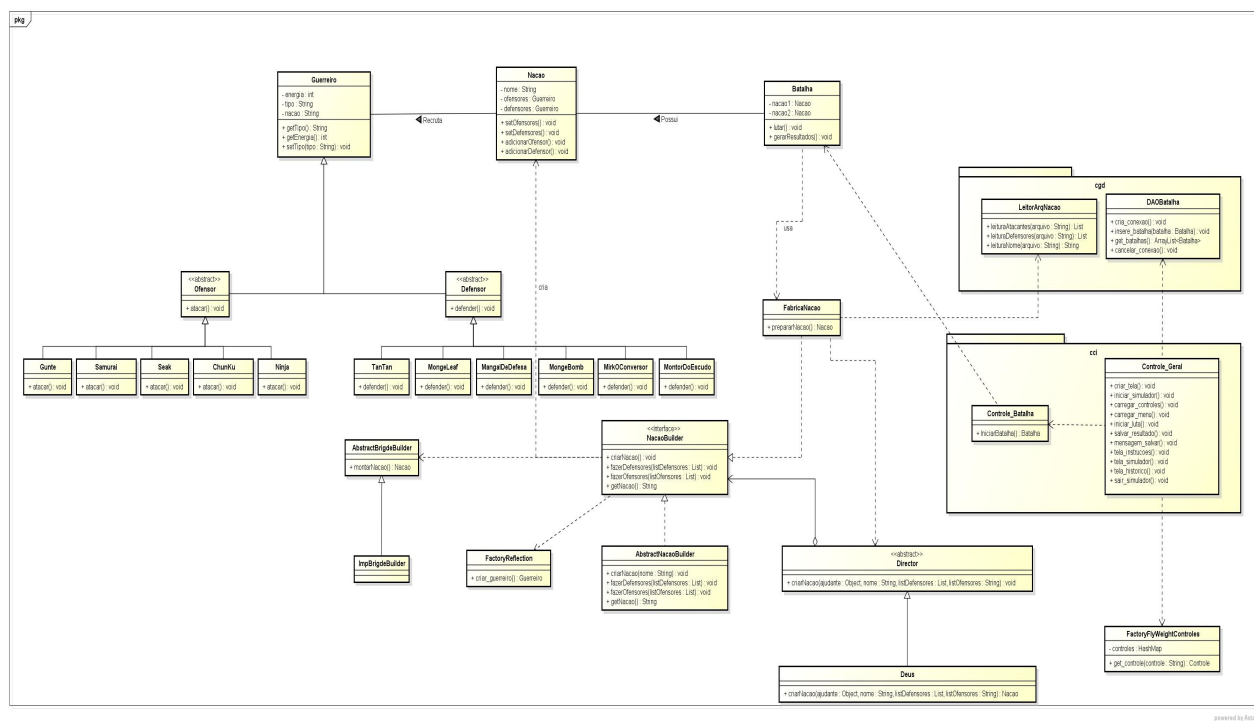
Figura - MVC

A abordagem MVC é composta por três tipos de objetos. O Modelo é o objeto de aplicação, a Visão é a apresentação na tela e o Controlador é o que define a maneira como a interface do usuário reage às entradas do mesmo. Em nosso trabalho o MVC foi decomposto em 5 camadas, o pacote “CCI” ficou responsável pelo controle entre a interface e a lógica de negócio, o “CIH” é onde estão as classes que são as telas para interação com o usuário, por exemplo “Tela\_Menu” e “Tela\_Resultado”. O “CGD” é onde ficam as classes de manipulação com dados, como: “DAOBatalha” e “LeitorArqNacao”, o “CDP” é onde estão as classes que definem o escopo da lógica de negócio.

A figura abaixo mostra como é feita a comunicação entre as camadas



## Diagrama de classe







Observação: Como a imagem ficou pequena no relatório, a imagem e o arquivo astah estão na repositório do trabalho no Github.

## Discussão sobre o uso de padrões






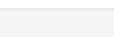
Os padrões estruturais se preocupam com a forma como classes e objetos são compostos para formar estruturas maiores. Os padrões estruturais de classes utilizam a herança para compor interfaces ou implementações. Já os padrões estruturais de objetos descrevem maneiras de compor objetos para obter novas funcionalidades. A flexibilidade obtida pela composição de objetos provém da capacidade de mudar a composição em tempo de execução.

Logo os padrões estruturais fornecem meios de melhorar o código em vários aspectos como: economia de memória com o padrão flyweight, utilizar várias interfaces de subsistemas através de uma única interface com o padrão Fachada, compor e dar responsabilidades de maneira dinâmica com o Decorator, entre outros.

## Analisando o resultado do Sonar

	04/10/2015	22/11/2015	
		1.0	
Complexity	149	184	
Complexity /function	2,1	1,5	
Complexity /class	7,0	4,2	
Complexity /file	5,3	3,5	

Podemos observar que a complexidade das funções, classes e arquivos caíram mais a complexidade geral aumentou (Não sabemos explicar esse fato).

Lines of code	453	756	
Lines	705	1.175	
Statements	451	755	
Files	28	52	
Classes	21	43	
Functions	70	119	
Accessors			

Mesmo com o aumento de classes a complexidade das classes diminuíram, além do fato de estarmos utilizando o padrão Reflection para instanciar as classes, os padrões estruturais sem dúvidas estão contribuindo para essa diminuição aconteça.