
TÓPICOS AVANÇADOS DE POO – MAC 413

PADRÃO DE DESENHO ENUMERATOR

Jorge Francisco Del Teglia – nº USP 3.207.552
Rubens Sawaya Altimari – nº USP 3.286.287

II - IX - MMII

ENUMERATOR

Object Creational

Intenção

Definir um conjunto (estático) de constantes fortemente tipadas.

Motivação

Em alguns casos, é importante poder contar com um conjunto bem definido de elementos. Por exemplo, podemos ter um conjunto de dias da semana, contendo os elementos “Segunda-Feira”, “Terça-Feira”, e assim por diante. Nestas circunstâncias, é interessante que a representação utilizada denote precisamente o conjunto em questão.

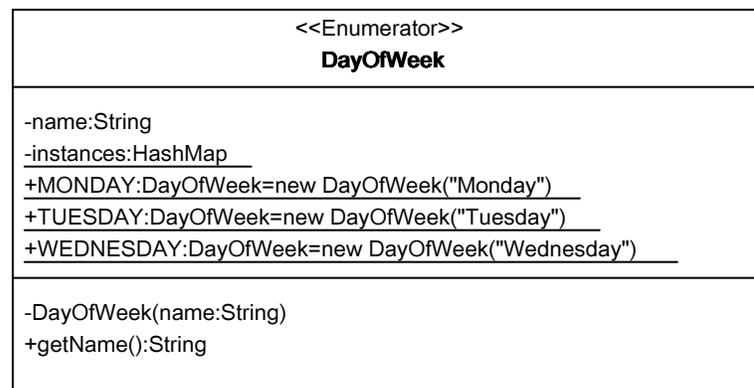
Em Java, por exemplo, não existe um mecanismo inerente à linguagem para isto, como o especificador de tipo **enum**, em C++. Uma implementação comum em Java seria definir constantes utilizando código similar ao seguinte:

```
public static final int SEGUNDA_FEIRA = 1;  
public static final int TERCA_FEIRA = 2;
```

Este esquema, porém, traz vários problemas. Em primeiro lugar, é preciso manter controle manual sobre os valores de cada constante, evitando que se repitam. Além disso, não se tratam de *tipos* específicos, e portanto o compilador não tem condições de garantir que *apenas* estas constantes sejam utilizadas em contextos que as esperam (ex.: chamadas de função, valores de variáveis, etc.). Podemos dizer também que não há maneiras de conhecer o conjunto completo de elementos, ou tratá-lo de forma coletiva. Finalmente, não é possível associar a cada constante nenhuma outra informação, além do próprio valor.

Mesmo em C++, embora haja a definição de um novo tipo, os possíveis valores de cada elemento devem se restringir a um tipo integral (**short**, **int**, **long**, etc.).

Uma solução alternativa e potencialmente melhor é definir um conjunto estático de constantes fortemente tipadas, pertencentes a uma classe própria, relacionando-as em termos de grupo e disponibilizando-as de uma forma global. Este é o padrão Enumerator.



O diagrama anterior ilustra uma forma simplificada do Enumerator. Note que o construtor da classe DayOfWeek é privado, o que significa que o cliente não pode instanciar nenhum objeto a partir dela. A única forma de criar objetos, então, é utilizar o construtor de dentro da própria classe, como é feito para a inicialização de cada um dos elementos (estáticos) que a compõem.

Com este artifício, garantimos que os *únicos* objetos criados são exatamente cada um dos elementos desejados, e a definição do conjunto de elementos é feita em tempo de compilação, não sendo possível, portanto, acrescentar posteriormente novos elementos.

Aplicabilidade

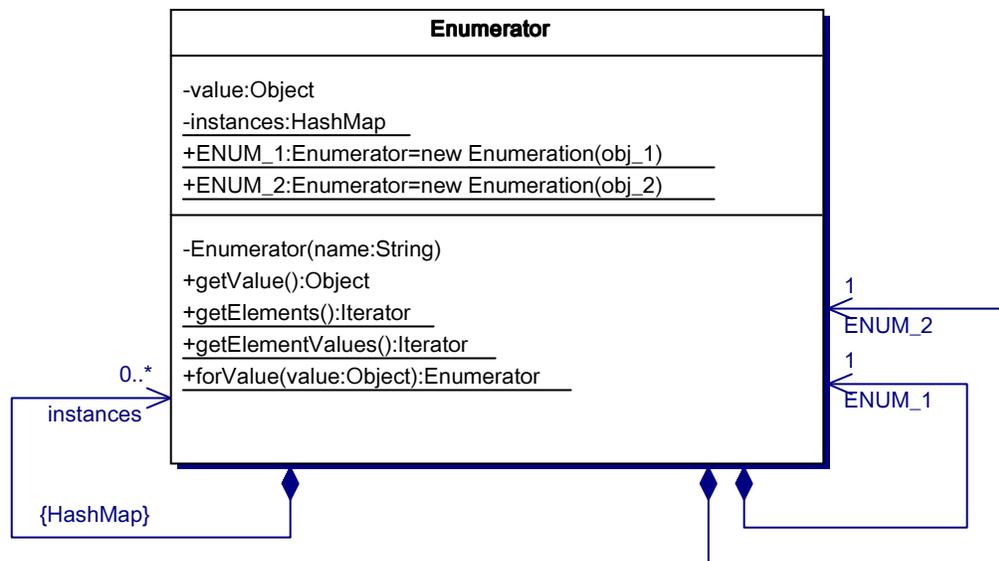
Use este padrão quando:

- você quer utilizar um conjunto de constantes relacionadas entre si, permitindo que o compilador faça uma checagem estática (em tempo de compilação) de tipos;
- você precisa relacionar a cada constante um objeto associado, sem recorrer a um mapeamento desvinculado da definição da constante.

Estrutura

Abaixo vemos uma representação do Enumerator e de alguns de seus possíveis recursos. O construtor (privado) da classe acrescenta cada novo objeto criado a um hash map (estático), para posterior referência. Com isto, ganha-se a possibilidade de ter métodos como *getElements()* e *getElementValues()*, para manipulação do conjunto como um todo.

O fato de o construtor ser privado garante que os únicos objetos criados são exatamente cada um dos elementos desejados, e a definição do conjunto de elementos é feita em tempo de compilação, não sendo possível, portanto, acrescentar posteriormente novos elementos.



O relacionamento da esquerda mantém as referências ao conjunto como um todo, enquanto que os relacionamentos da direita mantêm as referências a cada elemento individualmente.

Participantes

- Enumerator
 - define um conjunto de atributos estáticos para acessar os elementos do conjunto;
 - gerencia todas as instâncias mantendo uma referência a elas estaticamente, podendo, desta forma, oferecer serviços aplicáveis sobre todo o conjunto de elementos (como, por exemplo, saber o número de elementos).

Colaborações

- Os clientes acessam as instâncias do Enumerator unicamente através das constantes definidas na classe ou utilizando os serviços globais.

Conseqüências

O padrão Enumerator tem as seguintes conseqüências:

1. *Tipagem forte para os elementos do conjunto*: como cada elemento do conjunto pertence a uma classe, há um tipo bem definido a ser usado pelo compilador para checar a consistência de sua utilização. Pode-se assim evitar erros em chamadas de função, atribuições de variáveis e outras circunstâncias onde a falta de um tipo pode confundir o compilador.
2. *Acesso global*: provê mecanismo de acesso bem definido aos elementos do conjunto, sem poluir o espaço de nomes da aplicação.
3. *Conjunto fixo de instâncias*: a definição do conjunto de elementos é feita em tempo de compilação, não sendo possível, portanto, acrescentar posteriormente novos elementos.
4. *Associação de funcionalidade*: pode-se relacionar a cada constante um objeto associado, sem recorrer a um mapeamento desvinculado da definição da constante. Sem isto, seria preciso manter um registro à parte, usualmente implementado como um **switch**, uma tabela, ou mecanismo semelhante.
5. *Funcionalidade para tratar o conjunto como um todo*: pode-se dotar o Enumerator de métodos para acessar características gerais do conjunto, como ilustrado pelo método `getElements()`.

Implementação

Considere os seguintes tópicos ao implementar um Enumerator:

1. *Garantindo a imutabilidade das constantes*: pode ser desejável garantir que cada elemento do conjunto seja de fato uma constante, isto é, que seu valor seja único e mantido imutável ao longo da vida da aplicação. Para tanto, é preciso garantir que a classe que implementa o

padrão Enumerator não possua métodos capazes de modificar seu estado.

Um exemplo do uso contra-indicado acima pode ser visto abaixo:

```
public class DayOfWeek extends Object
{
    ...
    public void setValue(Object newObject) { ... }
    ...
}
```

2. *Enumerator como um mapa estático*: o Enumerator pode ser tratado como um mapeamento, de tamanho fixo em tempo de compilação, onde cada elemento, porém, pode ter seu valor modificado livremente em tempo de execução.
3. *Garantindo a unicidade do valor dos elementos*: uma vez que o construtor é responsável por acrescentar cada novo elemento ao hash map, é possível prevenir que elementos com mesmo valor sejam adicionados.
4. *Gerador de código*: é trivial criar uma aplicação capaz de criar um Enumerator a partir do nome da classe e do conjunto de constantes desejados.

Exemplo

Segue abaixo um exemplo de implementação do Enumerator, neste caso para um conjunto de dias da semana.

```
import java.util.*;

public class DayOfWeek
{
    //
    // Attributes
    //
    private String name;
    private static HashMap instances = new HashMap();

    public final static DayOfWeek Monday = new DayOfWeek( "Monday" );
    public final static DayOfWeek Tuesday = new DayOfWeek( "Tuesday" );
    public final static DayOfWeek Wednesday = new DayOfWeek( "Wednesday" );
    public final static DayOfWeek Thursday = new DayOfWeek( "Thursday" );
    public final static DayOfWeek Friday = new DayOfWeek( "Friday" );
    public final static DayOfWeek Saturday = new DayOfWeek( "Saturday" );
    public final static DayOfWeek Sunday = new DayOfWeek( "Sunday" );

    //
    // User methods
    //

    public String getName()
    {
        return this.name;
    }
}
```

```
public static Iterator getElements()
{
    return instances.values().iterator();
}

public static Iterator getElementNames()
{
    return instances.keySet().iterator();
}

public static int getSize()
{
    return instances.size();
}

private DayOfWeek( String name )
{
    this.name = name;
    instances.put( name, this );
}

public static DayOfWeek forName( String name )
{
    DayOfWeek o = (DayOfWeek) instances.get( name );
    if ( o == null )
        throw new IllegalArgumentException( "Não encontrado " + name );
    return o;
}

} // class DayOfWeek
```

Abaixo um exemplo muito simples de utilização do Enumerator:

```
// Definição de um método
class Calendar
{
    ...
    public void selectFirstDayOfWeek( DayOfWeek startDay ) { ... }
}

// Chamada do método acima
...
myCalendar.selectFirstDayOfWeek( DayOfWeek.MONDAY );
```

Casos Conhecidos

- Sistema de Vendas de Cursos via Web (SENAC):
www.sp.senac.br
- Sistema de Vendas de Domínios On-Line (WhiteHat Software):
www.dominiofacil.com.br
- Sistema de Simulação Dinâmica de Partículas (Asgard Software)

Padrões Relacionados

Há certa semelhança de técnicas com o padrão Singleton, na medida em que naquele padrão também se oculta o construtor para evitar instâncias indesejadas. A semelhança termina aí, uma vez que o Enumerator possui um conjunto de aplicações que não pode ser endereçado por um Singleton.