Les modèles de comportement

Le modèle de comportement simplifie l'organisation d'exécution des objets. Typiquement, une fonction est composée d'un ensemble d'actions qui parfois appartiennent à des domaines différents de la classe d'implémentation. On aimerait donc pouvoir "déléguer" certains traitements à d'autres classes.

D'une manière générale, un modèle de de comportement permet de réduire la complexité de gestion d'un objet ou d'un ensemble d'objet.

Iterator

L'Itérateur ou Iterator est le plus commun des modèles de comportement. L'idée étant de limiter la vision d'une collection par un utilisateur. Typiquement une collection contient un ensemble d'objets stocké par différentes méthodes (un tableau, un vecteur...), l'exploitant qui accède au contenu de la collection ne souhaite pas être concerné par cette manière de gérer les objets. La collection offre donc un point d'accès unique sous la forme d'une interface Iterator.

Exemple :

```
/** Classe de gestion d'un espace de dessin */
public class CanvasImpl implements GraphicsElement, Canvas {
     // Tableau pour stoker les éléments de la collection
     private GraphicsElement[] ge;
     ...
     /** Retourne un itérateur pour accéder aux objets de la
     collection */
     public Iterator getIterator() { return ArrayIterator( ge
     ); }
     }
     /** Interface pour toutes les collections d'objets de
     GraphicElement */
}
public interface Iterator {
     public GraphicElement getNextElement();
     }
     /** Iterateur parcourant un tableau pour retourner des
     objets de type GraphicElement */
     public class ArrayIterator implements Iterator {
     private GraphicElement[] ge;
     private int nge;
     /** Constructeur avec un tableau de données à parcourir */
     public ArrayIterator( GraphicElement[] ge ) {
     this.ge = ge;
     nge = 0;
     }
     /** Retourne chaque élément de la collection ou null */
     public GraphicElement getNextElement() {
     if ( nge >= ge.length ) return null;
```

```
        return ge[ nge++ ];
        }
}
```

Cet exemple comprend un itérateur pour la collection CanvasImpl qui contient des objets de type GraphicElement.

Ces objets sont stockés dans un tableau, un itérateur ArrayIterator parcourt le tableau et offre chaque élément contenu dans la collection par la méthode getNextElement. Si plus tard, on désire changer de méthode de stockage des objets pour des raisons de performances ou de coûts mémoire, il suffira de réaliser une nouvelle classe implémentant l'interface Iterator.

Exemple en PHP

```php
<?php

class AlphabeticalOrderIterator implements ¥Iterator
{
    private $collection;
    private $position = 0;

    private $reverse = false;

    public function __construct($collection, $reverse = false)
    {
        $this->collection = $collection;
        $this->reverse = $reverse;
    }

    public function rewind()
    {
        $this->position = $this->reverse ?
            count($this->collection->getItems()) - 1 : 0;
    }

    public function current()
    {
        return $this->collection->getItems()[$this->position];
    }

    public function key()
    {
        return $this->position;
    }

    public function next()
    {
        $this->position = $this->position + ($this->reverse ? -1 : 1);
    }

    public function valid()
    {
        return isset($this->collection->getItems()[$this->position]);
    }
}


class WordsCollection implements ¥IteratorAggregate
{
    private $items = [];

    public function getItems()
    {
        return $this->items;
    }
```

```
    public function addItem($item)
    {
        $this->items[] = $item;
    }

    public function getIterator(): Iterator
    {
        return new AlphabeticalOrderIterator($this);
    }

    public function getReverseIterator(): Iterator
    {
        return new AlphabeticalOrderIterator($this, true);
    }
}

$collection = new WordsCollection();
$collection->addItem("First");
$collection->addItem("Second");
$collection->addItem("Third");

echo "Straight traversal:¥n";
foreach ($collection->getIterator() as $item) {
    echo $item . "¥n";
}

echo "¥n";
echo "Reverse traversal:¥n";
foreach ($collection->getReverseIterator() as $item) {
    echo $item . "¥n";
}
```