① Design pattern description

Iterator c Behavioral pattern

What problems can the Iterator design pattern solve? [2]

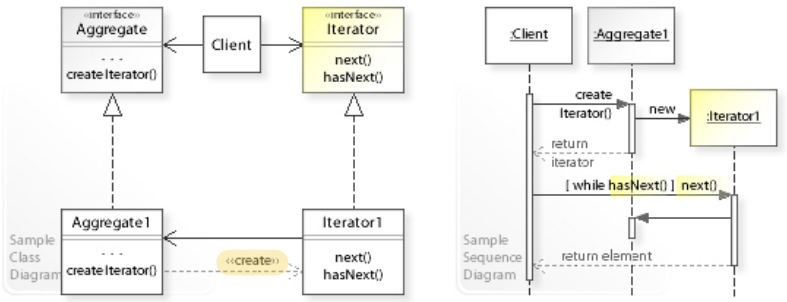
- The elements of an aggregate object should be accessed and traversed without exposing its representation (data structures).
- New traversal operations should be defined for an aggregate object without changing its interface.

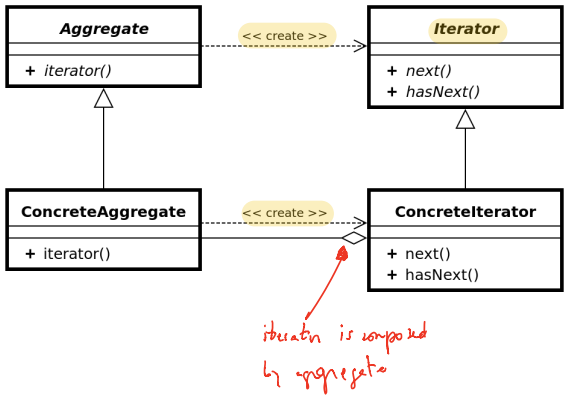
Defining access and traversal operations in the aggregate interface is inflexible because it commits the aggregate to particular access and traversal operations and makes it impossible to add new operations later without having to change the aggregate interface.

What solution does the Iterator design pattern describe?

- Define a separate (iterator) object that encapsulates accessing and traversing an aggregate object.
- Clients use an iterator to access and traverse an aggregate without knowing its representation (data structures).

Different iterators can be used to access and traverse an aggregate in different ways.
New access and traversal operations can be defined independently by defining new iterators.





Design pattern example

```php
<?php

// BookIterator.php

namespace DesignPatterns;

class BookIterator implements \Iterator
{
    private $i_position = 0;
    private $booksCollection;

    public function __construct(BookCollection $booksCollection)
    {
        $this->booksCollection = $booksCollection;
    }

    public function current()
    {
        return $this->booksCollection->getTitle($this->i_position);
    }

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

    public function next()
    {
        $this->i_position++;
    }

    public function rewind()
    {
        $this->i_position = 0;
    }

    public function valid()
    {
        return !is_null($this->booksCollection->getTitle($this->i_position));
    }
}
```

```php
<?php

// BookCollection.php

namespace DesignPatterns;

class BookCollection implements \IteratorAggregate
{
    private $a_titles = array();

    public function getIterator()
    {
        return new BookIterator($this);
    }

    public function addTitle($string)
    {
        $this->a_titles[] = $string;
    }

    public function getTitle($key)
    {
        if (isset($this->a_titles[$key])) {
            return $this->a_titles[$key];
        }
        return null;
    }

    public function is_empty()
    {
        return empty($this->$a_titles);
    }
}
```

```php
<?php

// index.php

require 'vendor/autoload.php';
use DesignPatterns\BookCollection;

$booksCollection = new BookCollection();
$booksCollection->addTitle('Design Patterns');
$booksCollection->addTitle('PHP7 is the best');
$booksCollection->addTitle('Laravel Rules');
$booksCollection->addTitle('DHH Rules');

foreach($booksCollection as $book){
    var_dump($book);
}
```

```
string(15) "Design Patterns"
string(16) "PHP7 is the best"
string(13) "Laravel Rules"
string(9) "DHH Rules"
```

iter = $booksCollection->getIterator()

if iter.valid()
iter.current()
iter.next()

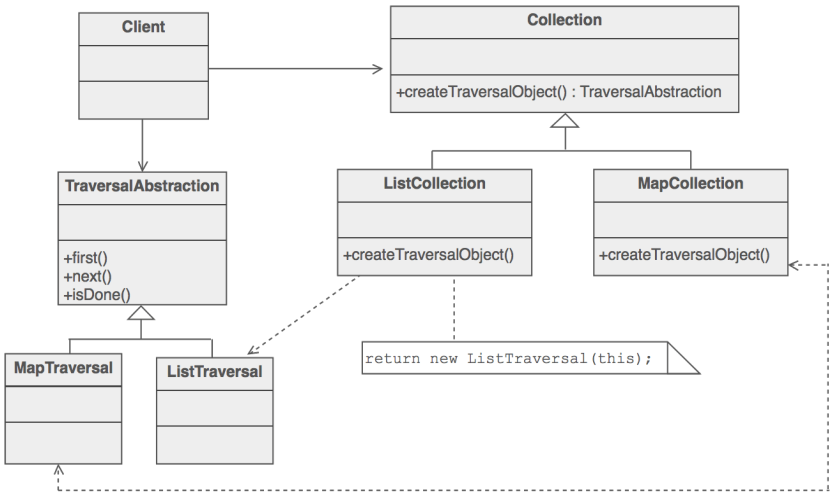
end

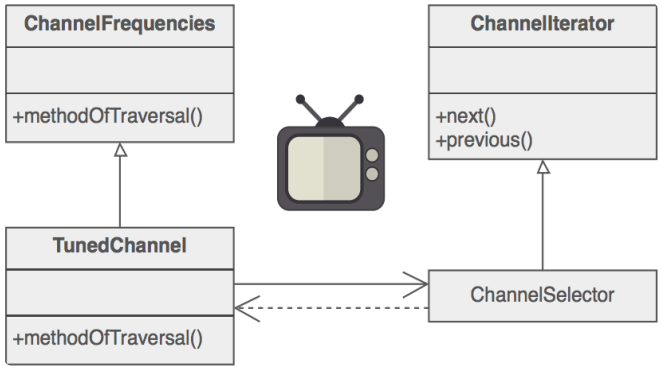
③ Existing Example in FEM-MAT-OO ?

Not explicitly, but almost

④ Design pattern proposal in FEM-MAT-OO ?

```
optimIteration = Optimizer.getIteration()
optimIteration.computeFirstIteration()
if optimizer.hasConverged()
    optimIteration.iterate()
    optimIteration.plot()
end
```





1. Add a `create_iterator()` method to the "collection" class, and grant the "iterator" class privileged access.
2. Design an "iterator" class that can encapsulate traversal of the "collection" class.
3. Clients ask the collection object to create an iterator object.
4. Clients use the `first()`, `is_done()`, `next()`, and `current_item()` protocol to access the elements of the collection class.

Rules of thumb

- The abstract syntax tree of Interpreter is a Composite (therefore Iterator and Visitor are also applicable).
- Iterator can traverse a Composite. Visitor can apply an operation over a Composite.
- Polymorphic Iterators rely on Factory Methods to instantiate the appropriate Iterator subclass.
- Memento is often used in conjunction with Iterator. An Iterator can use a Memento to capture the state of an iteration. The Iterator stores the Memento internally.