

РНР. Уровень 4. Проектирование и разработка сложных веб - проектов

Ведущий курса: Тарасов Алексей Владимирович

Темы курса

- Шаблоны проектирования
- Standard PHP Library (SPL)
- PHP Data Objects (PDO)
- Reflection
- cURL
- Регулярные выражения
- Пространства имен
- Модульное тестирование
- Шаблон проектирования MVC
- Создание REST-сервиса



Центр компьютерного [®]
(ОБУЧЕНИЯ)
«СПЕЦИАЛИСТ»
при МГТУ им. Н.Э.Баумана

Шаблоны проектирования

Ведущий курса: Тарасов Алексей Владимирович

Темы модуля

- Обзор UML
- Диаграмма классов
- Введение в шаблоны проектирования
- Шаблоны проектирования
 - Singleton Pattern
 - Factory Pattern
 - Strategy Pattern
 - Decorator Pattern
 - Adapter Pattern
 - Другие шаблоны

Обзор UML

- UML – язык унифицированного моделирования
- Используется для описания ООП моделей
- В июне 2015 была выпущена UML 2.5
- <http://www.omg.org/spec/UML/2.5/>
- Большое количество диаграмм
- Есть сертификация

Диаграмма классов

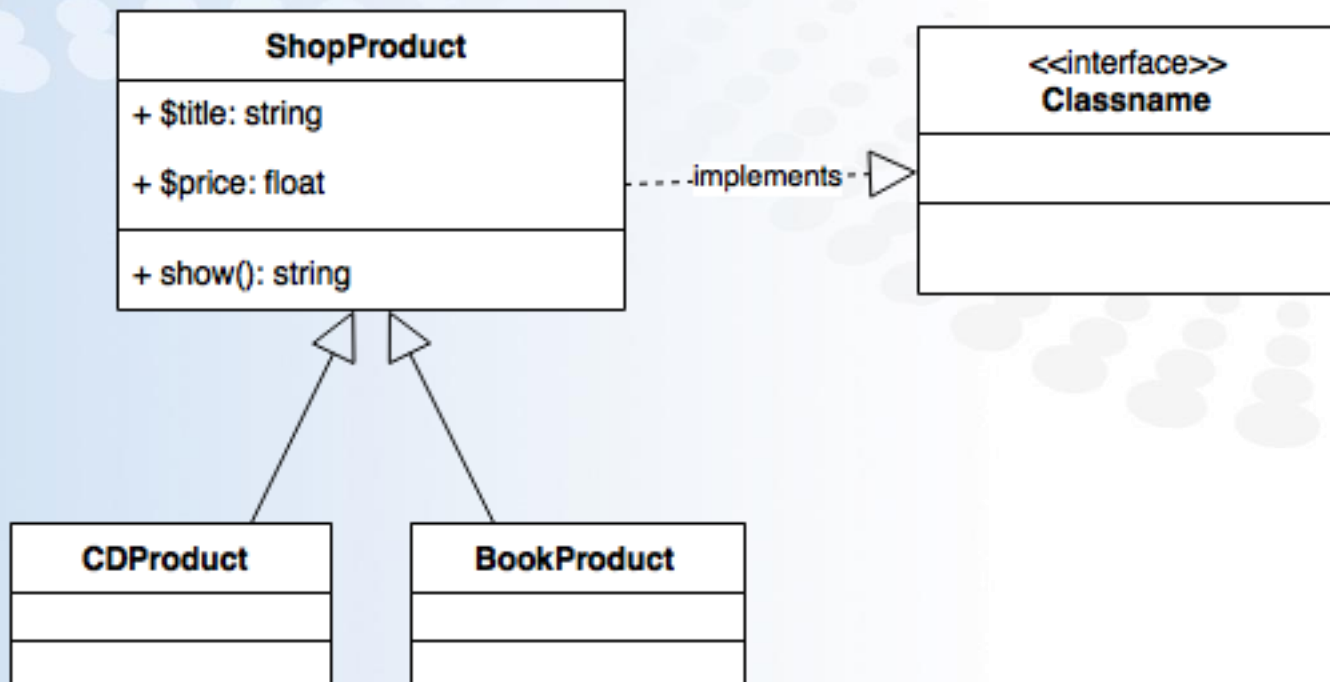
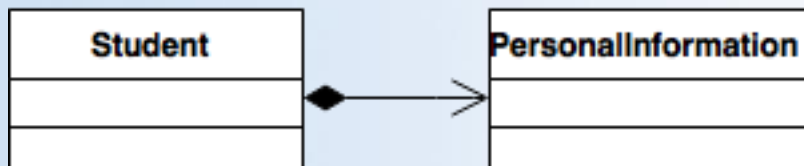
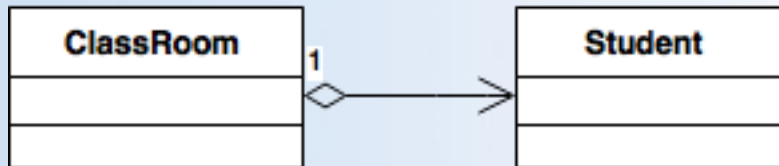
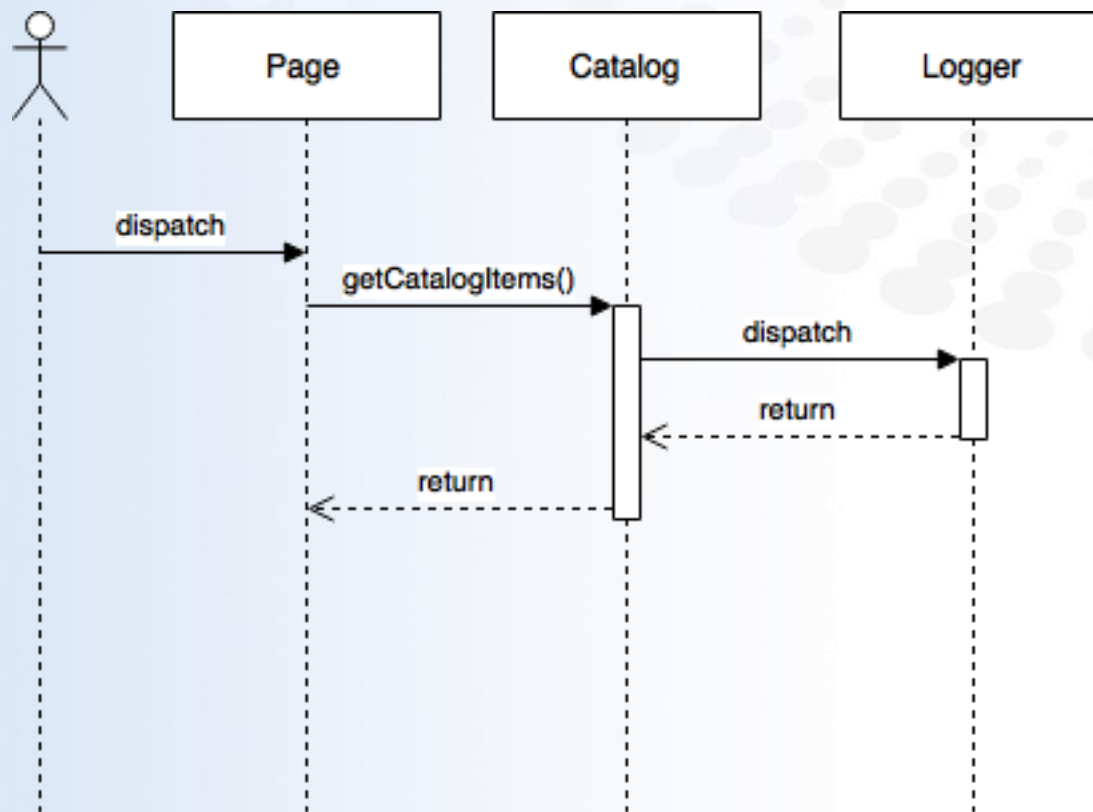


Диаграмма классов: ассоциации, агрегирование и композиция



- Ассоциация: свойство класса содержит ссылку на экземпляр другого класса
- Агрегирование: объекты составляют основную часть объекта-контейнера (но могут быть и в других объектах)
- Композиция: на содержащийся объект ссылается только объект-контейнер

Диаграмма последовательности



Введение в шаблоны проектирования

- Шаблоны описывают задачу, которая возникает снова и снова, а затем описывает суть решения данной задачи, так что вы можете использовать это решение миллион раз, каждый раз делая это по-разному. A Pattern Language. Christopher Alexander
- Структура шаблона: Название – Формулировка задачи – Решение – Результаты
- Определяют задачи
- Определяют решения
- Не зависят от языка
- Определяют терминологию
- Хороший стиль работы

Шаблоны проектирования

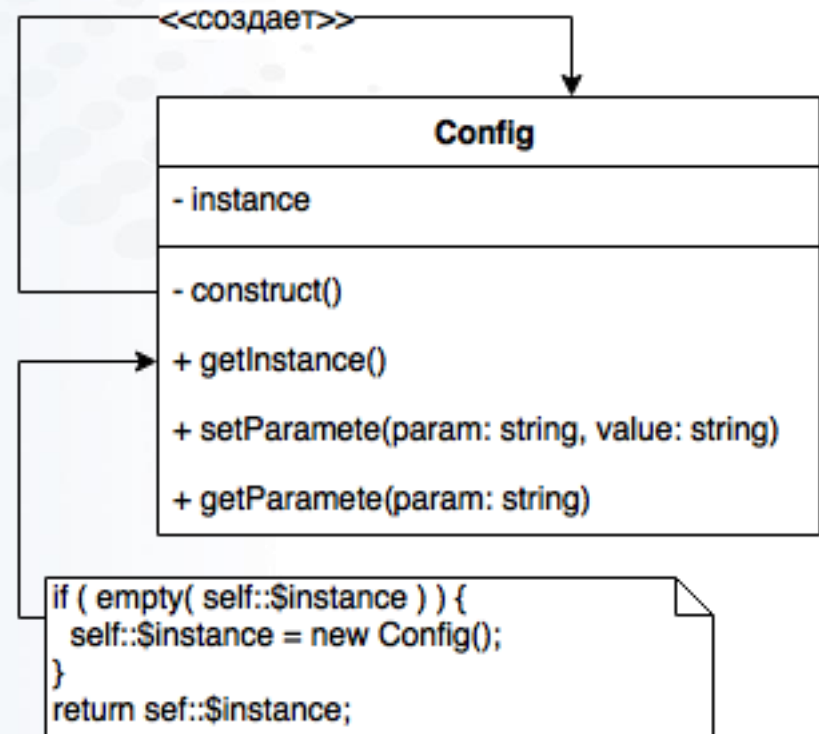
- Singleton Pattern
- Factory Pattern
- Strategy Pattern
- Decorator Pattern
- Adapter Pattern
- Другие шаблоны

Пример работы без Singleton Pattern

```
1. class Config {  
2.     private $parameters = array();  
3.     public function setParameter( $param, $value ){  
4.         $this->parameters[$param] = $value;  
5.     }  
6.  
7.     public function getParameter( $param ){  
8.         return $this->parameters[$param];  
9.     }  
10. }
```

Singleton Pattern

- Проблема: использование глобальных переменных
- Решение: создание класса, экземпляр которого нельзя создать за его пределами

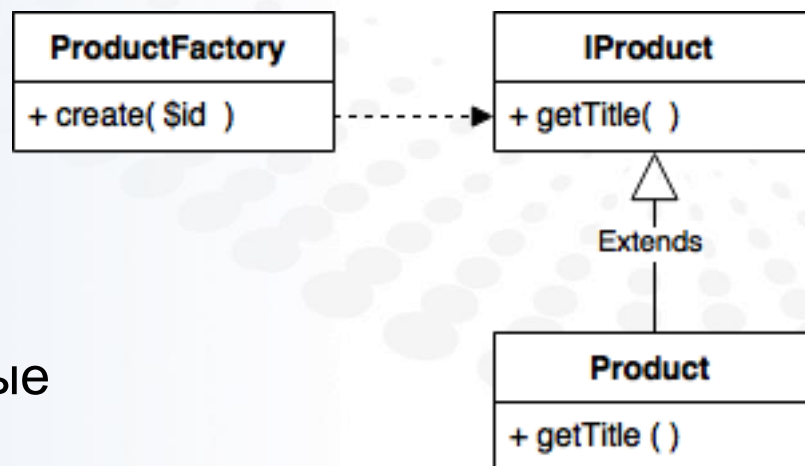


Пример работы без Factory Pattern

```
1. class Config {  
2.     private $parameters = array();  
3.     public function setParameter( $param, $value ){  
4.         $this->parameters[$param] = $value;  
5.     }  
6.  
7.     public function getParameter( $param ){  
8.         return $this->parameters[$param];  
9.     }  
10. }
```

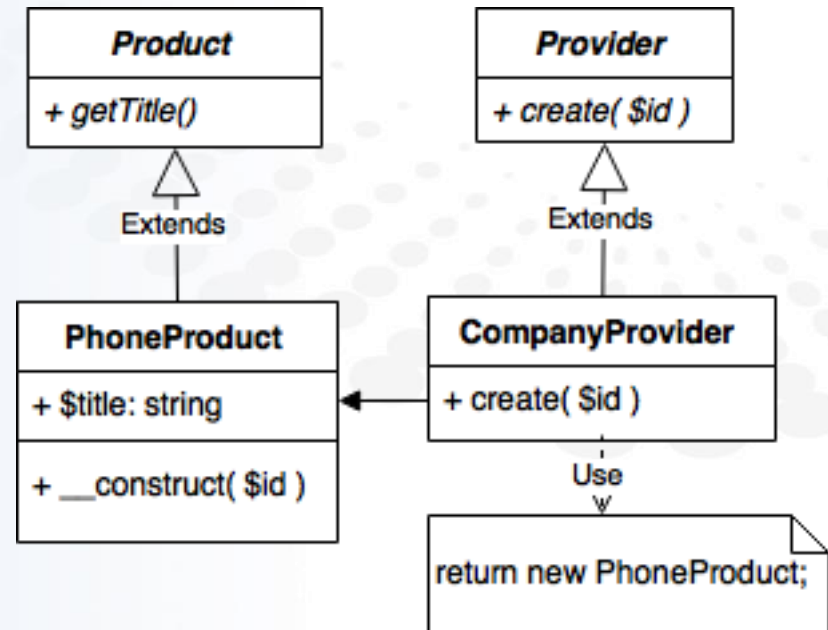
Factory Pattern

- Проблема: условные операторы и расширение объектной модели только за счет наследования
- Решение: классы создатели отделены от объектов, которые они должны генерировать. Класс создатель – класс фабрики, в котором определен метод генерации объекта продукта



Factory Method / Virtual Constructor

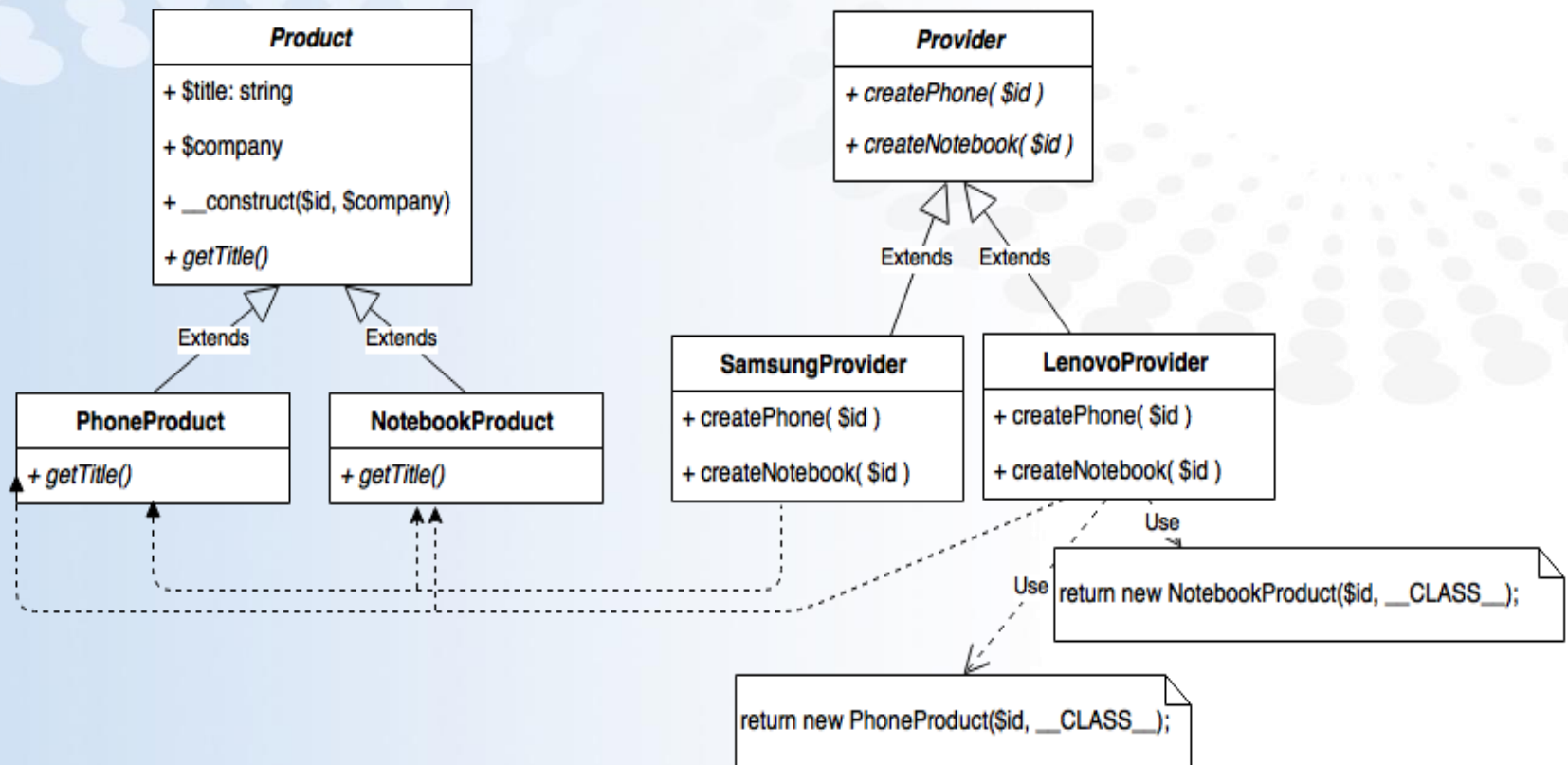
- Случаи использования:
- Классу неизвестно, объекты каких классов ему нужно создавать
- Класс спроектирован так, чтобы создаваемые им объекты специфицировались классами
- Класс передает свои обязанности одному из нескольких вспомог. подклассов и мы определяем кто принимает обязанности



Abstract Factory

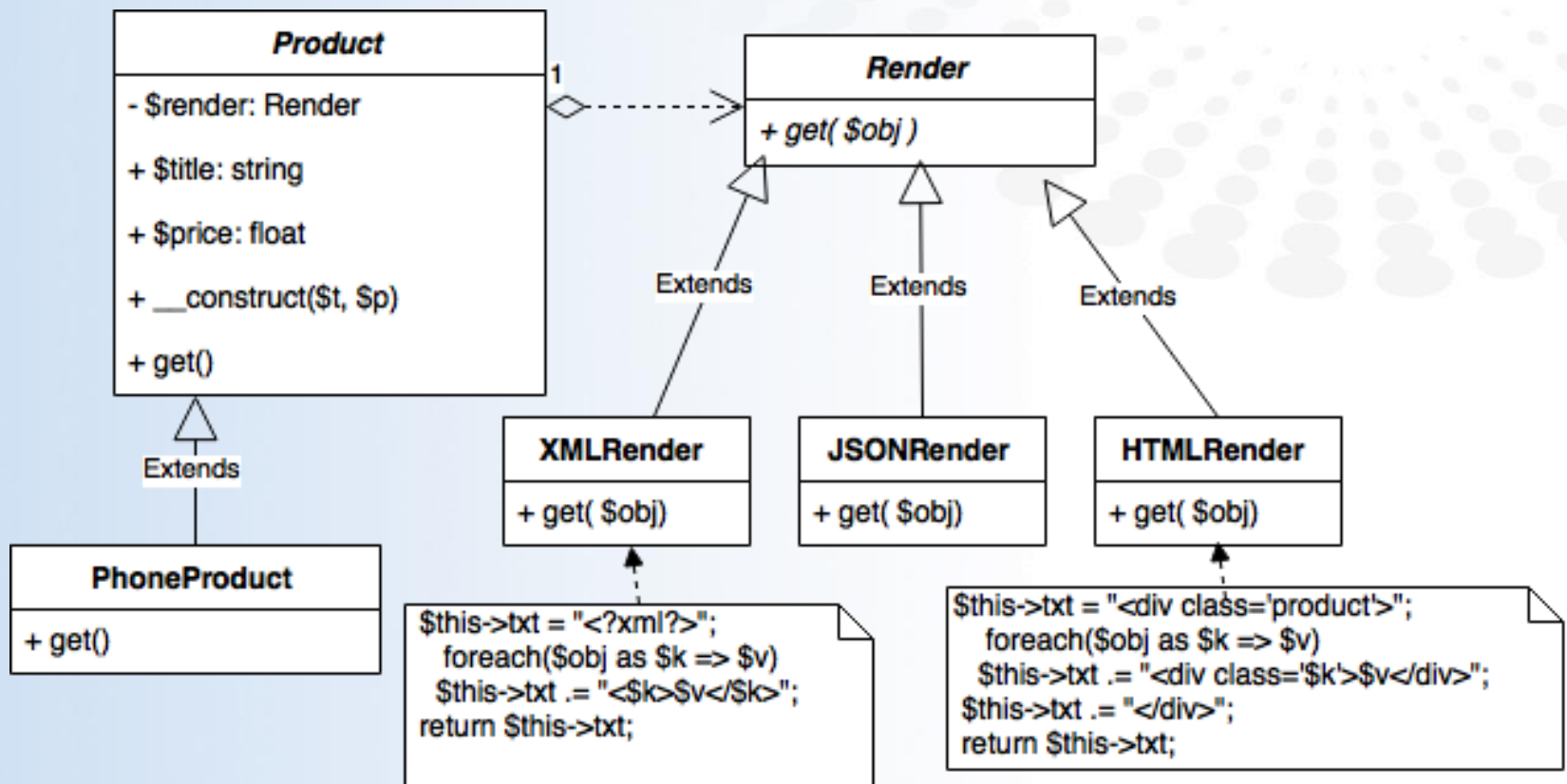
- Необходимо представить решение для создания семейств связанных объектов
- Система не должна зависеть от того, как создаются, компонуются и представляются входящие в нее объекты
- Система должна конфигурироваться одним из семейств составляющих ее объектов
- Нужно предоставить библиотеку, раскрывая их интерфейсы и не показывая реализацию

Abstract Factory

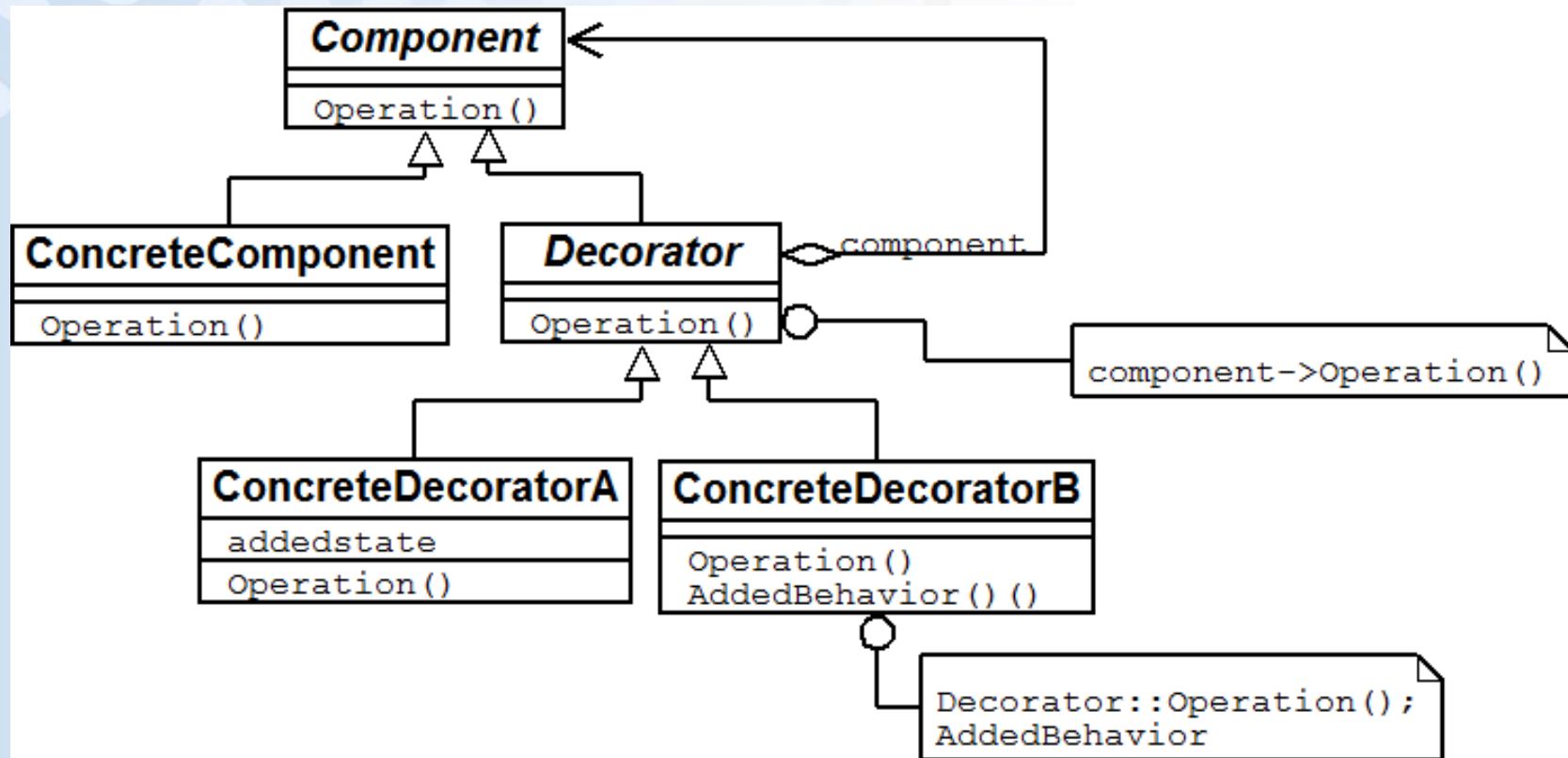


Strategy Pattern

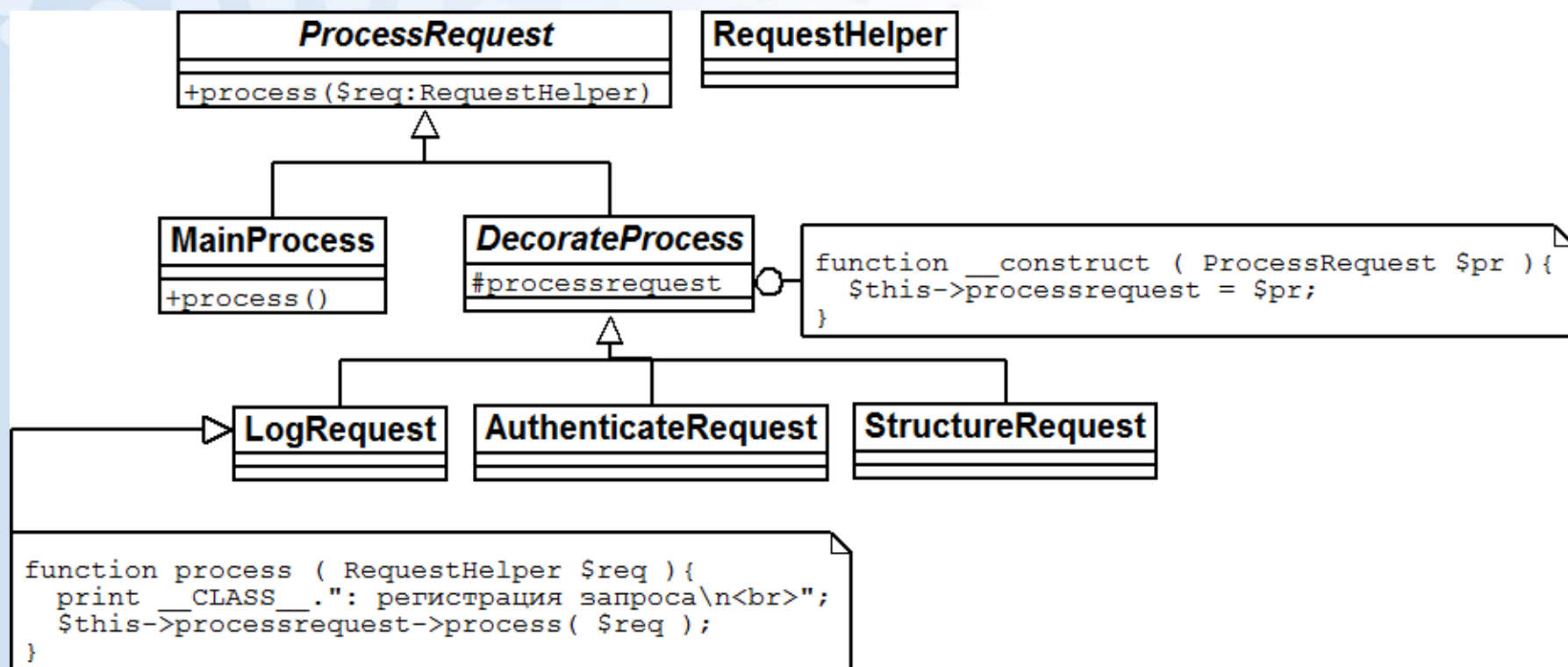
- Избавление классов от лишнего функционала



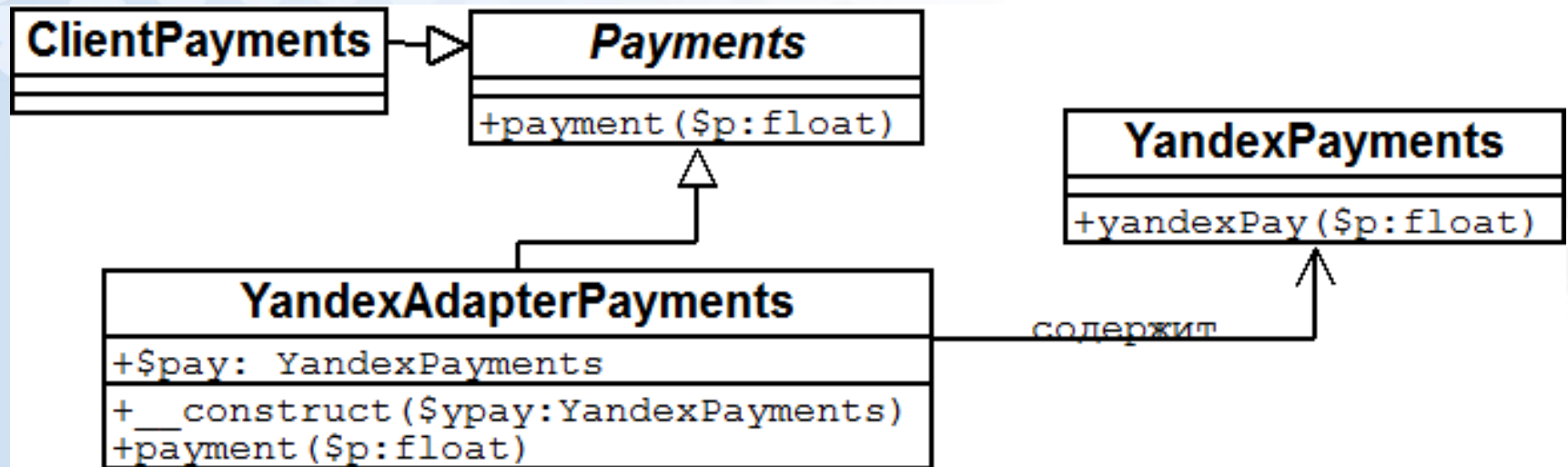
Decorator Pattern



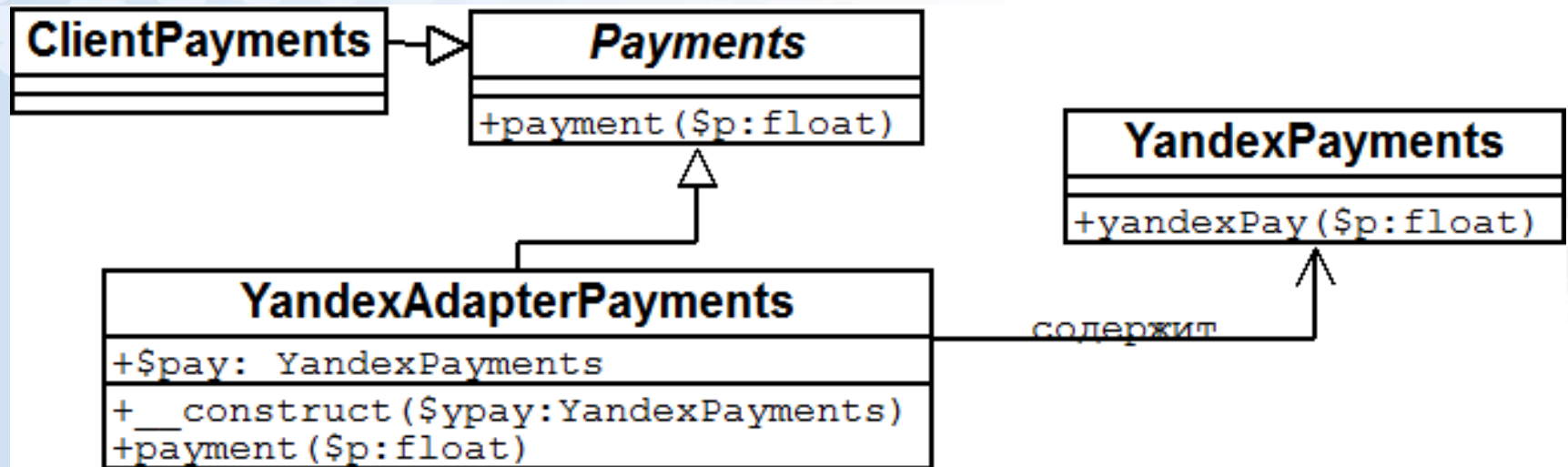
Decorator Pattern



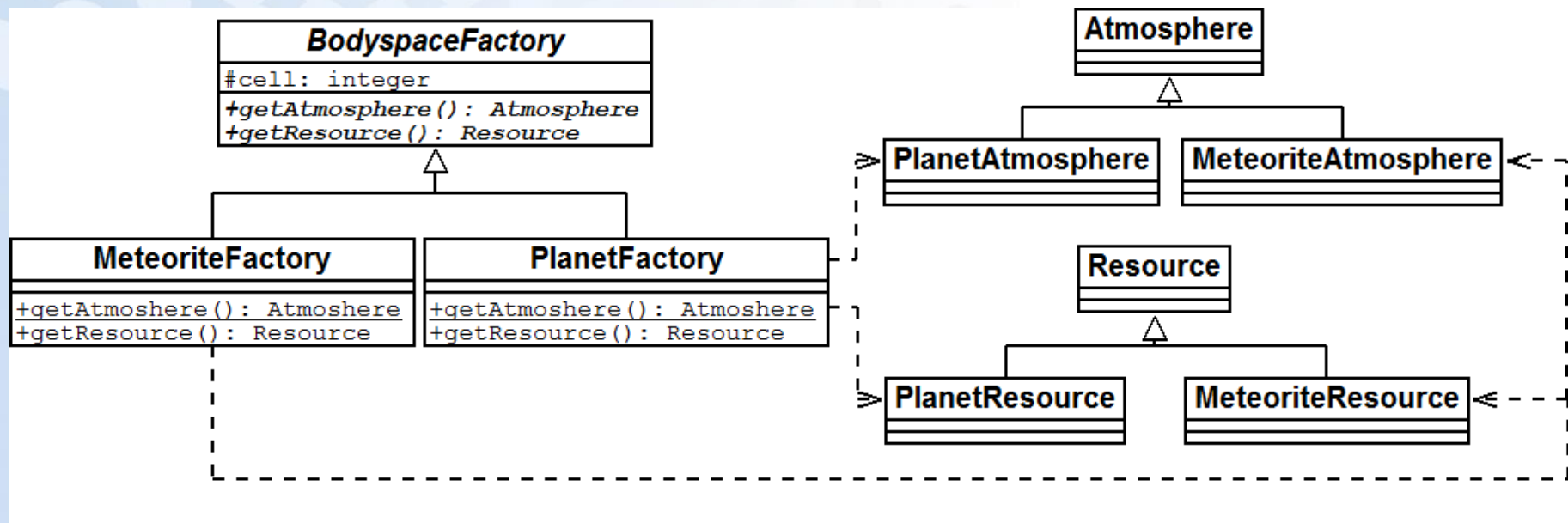
Adapter Pattern



Adapter Pattern



Prototype Pattern



Другие шаблоны

Порождающ ие	Структуриру ющие	Выполнения задач и результата	Корпоратив ных приложений	Шаблоны баз данных
Singleton Factory Method Abstract Factory Prototype	Composite Decorator Façade	Interpreter Strategy Observer Visitor Command	Registry Front Controller Application Controller Page Controller Template View Transaction Script Domain Model	Data Mapper Identity Map Unit of work Lazy Load Domain Object Factory Identity Object

Лабораторная работа

Итоги

- Обзор UML
- Диаграмма классов
- Введение в шаблоны проектирования
- Шаблоны проектирования
 - Singleton Pattern
 - Factory Pattern
 - Strategy Pattern
 - Decorator Pattern
 - Adapter Pattern
 - Другие шаблоны



Центр компьютерного [®]
(ОБУЧЕНИЯ)
«СПЕЦИАЛИСТ»
при МГТУ им. Н.Э.Баумана

Standard PHP Library (SPL)

Ведущий курса: Тарасов Алексей Владимирович

Темы модуля

- Встроенные интерфейсы и классы
- Замыкания
- Генераторы
- SPL – Standard PHP Library
- Общие принципы
- Интерфейсы
- Итераторы
- Классы
- Структуры данных
- Функции
- Лабораторные работы

Встроенные интерфейсы и классы

- Closure
- Generator
- Traversable
- Iterator
- IteratorAggregate
- ArrayAccess
- Serializable

Замыкания

- Closure
- Класс используется для создания анонимных функций
- Доступны с 5.3
- Псевдо-тип callback использовался в этой документации до того, как был введен тип callable в PHP 5.4. Он означает в точности то же самое.

Пример анонимной функции

- Анонимная функция

- ```
$greet = function($name) {
 printf("Hello %s\r\n", $name);
};
$greet('World');
$greet('PHP');
```

- Наследование переменных из родительской области видимости

- ```
$example = function () use ($message) {  
    var_dump($message);  
};  
echo $example();
```

Интерфейс `ArrayAccess`

```
ArrayAccess {  
    /* Методы */  
    abstract public boolean offsetExists ( mixed $offset )  
    abstract public mixed offsetGet ( mixed $offset )  
    abstract public void offsetSet (mixed $offset, mixed $value )  
    abstract public void offsetUnset ( mixed $offset )  
}
```


Интерфейс Serializable

```
Serializable {  
    /* Методы */  
    abstract public string serialize ( void )  
    abstract public void unserialize ( string $serialized )  
}
```

- Классы реализующие интерфейс не поддерживают `__sleep()` и `__wakeup()`
- Метод `unserialize()` вызывается как конструктор вместо вызова `__construct()`

Интерфейс Traversable

```
Traversable {  
}
```

- Интерфейс определяющий обходимый класс при помощи foreach (traversable)
- Не может быть реализован сам по себе
- Методов нет

Интерфейс Iterator/IteratorAggregate

```
Iterator extends Traversable {  
    /* Методы */  
    abstract public mixed current ( void )  
    abstract public scalar key ( void )  
    abstract public void next ( void )  
    abstract public void rewind ( void )  
    abstract public boolean valid ( void )  
}  
  
IteratorAggregate extends Traversable {  
    /* Методы */  
    abstract public Traversable getIterator ( void )  
}
```

- Специальный интерфейс для внешних итераторов или объектов

Генераторы

```
Generator implements Iterator {  
    /* Методы */  
    public mixed current ( void )  
    public mixed key ( void )  
    public void next ( void )  
    public void rewind ( void )  
    public mixed send ( mixed $value )  
    public mixed throw ( Exception $exception )  
    public bool valid ( void )  
    public void _0_wakeup ( void )  
}
```

- Генераторы создают возможность для реализации итераторов

SPL – Standard PHP Library

- Стандартная библиотека для решения задач в PHP
- Начиная с 5.0.0 доступно, а с 5.3.0 нельзя отключить!
- Нет настроек в php.ini

- Типы
- Структуры данных
- Итераторы
- Исключения
- SPL функции
- Обработка файлов
- Классы и интерфейсы

Типы данных

- Нужна библиотека PECL
- SplInt
- SplFloat
- SplEnum
- SplBool
- SplString

Структуры данных

- SplDoublyLinkedList класс двусвязного списка
 - SplStack
 - SplQueue
- SplHeap класс кучи
 - SplMaxHeap
 - SplMinHeap
- SplPriorityQueue
- SplFixedArray класс фиксированного массива
- SplObjectStorage класс карта

Класс SplDoublyLinkedList

```
SplDoublyLinkedList implements Iterator , ArrayAccess , Countable {  
    /* Методы */ , __construct , add ( mixed $index , mixed $newval ) , mixed  
    bottom , int count , mixed current , int getIteratorMode , bool isEmpty  
    , mixed key , next , bool offsetExists ( mixed $index ) , mixed offsetGet  
    ( mixed $index ) , offsetSet ( mixed $index , mixed $newval ) , offsetUnset ( mixed $index ) ,  
    mixed pop , prev , push ( mixed $value ) , rewind , string serialize , setIteratorMode ( int $mode ) ,  
    mixed shift , mixed top , unserialize ( string $serialized ) , unshift ( mixed $value ) , bool valid  
}
```


Классы SplStack, SplQueue

```
SplStack extends SplDoublyLinkedList implements Iterator , ArrayAccess  
, Countable {  
    __construct ( void )  
    void setIteratorMode ( int $mode )  
    ...  
}
```

```
SplQueue extends SplDoublyLinkedList implements Iterator , ArrayAccess  
, Countable {__construct ( void )  
    mixed dequeue ( void )  
    void enqueue ( mixed $value )  
    void setIteratorMode ( int $mode )  
    ...  
}
```

Класс SplHeap

```
abstract SplHeap implements Iterator , Countable {  
    public __construct ( void )  
    abstract protected int compare ( mixed $value1 , mixed $value2 )  
    public int count ( void )  
    public mixed current ( void )  
    public mixed extract ( void )  
    public void insert ( mixed $value )  
    public bool isEmpty ( void )  
    public mixed key ( void )  
    public void next ( void )  
    public void recoverFromCorruption ( void )  
    public void rewind ( void )  
    public mixed top ( void )  
    public bool valid ( void )  
}
```

Класс SplMaxHeap

- Класс SplMaxHeap предоставляет основные функциональные возможности кучи, сохраняя максимальный элемент наверху
- SplMaxHeap::compare — Сравнивает элементы, чтобы во время сортировки корректно разместить их в куче

```
SplMaxHeap extends SplHeap implements Iterator ,  
Countable {  
  
...  
}
```

Массив фиксированной длины

- SplFixedArray
- Реализует функциональность обычных массивов
- Индексы только целочисленные
- Более быстрая обработка массива

Класс SplObjectStorage

- Обеспечивает соответствие объекты - данные

Итераторы

- AppendIterator
- ArrayIterator
- CachingIterator
- CallbackFilterIterator
- DirectoryIterator
- EmptyIterator
- FilesystemIterator
- FilterIterator
- GlobIterator
- InfiniteIterator
- IteratorIterator
- LimitIterator
- ...

Рекурсивная итерация

- `$arr = array('Zero', 'name'=>'Adil', 'address' => array('city'=>'Dubai', 'tel' => array('int' => 971, 'tel'=>12345487)), '' => 'nothing');`
- `$iterator = new \RecursiveIteratorIterator(new \RecursiveArrayIterator($arr));`
- `var_dump(iterator_to_array($iterator,true));`

Фильтрационный итератор

- Этот абстрактный итератор фильтрует нежелательные значения. Этот класс следует расширить для реализации пользовательских фильтров итератора. Метод `FilterIterator::accept()` должен быть реализован в подклассе.

Ограничительный итератор

- `$obj = new stdClass();`
- `$obj->rock = "камень";`
- `$obj->scissors = "ножницы";`
- `$obj->paper = "бумага";`

- `$infinite = new InfiniteIterator(new
ArrayIterator($obj));`
- `foreach (new LimitIterator($infinite, 2, 5) as
$value) {`
- `echo $value, "
";`
- `}`

Бесконечная итерация

- `$infinite = new InfiniteIterator(new ArrayIterator($obj));`
- `$i = 0;`
- `foreach ($infinite as $value) {`
- `print($value . "
");`
- `$i++;`
- `if($i > 10) break;`
- `}`

Объединение итераторов

- `$it = new AppendIterator;`
- `$it->append($it1);`
- `$it->append($it2);`

Обработка файлов

```
$dir = new FilesystemIterator(__DIR__);  
// Filter large files ( > 100MB)  
function is_large_file($current)  
{  
    return $current->isFile() && $current->getSize()  
> 0;  
}  
  
$large_files = new CallbackFilterIterator($dir,  
'is_large_file');
```

Итератор директории

```
foreach (new DirectoryInfo(__DIR__) as
$fileInfo) {
    if($fileInfo->isDot()) continue;
    echo $fileInfo->getFilename() . "<br>\n";
}
```

Работа с массивом как с объектом

- `ArrayObject`

Исключения

- **BadFunctionCallException**
- **BadMethodCallException**
- **DomainException**
- **InvalidArgumentException**
- **LengthException**
- **LogicException**
- **OutOfBoundsException**
- **OutOfRangeException**
- **OverflowException**
- **RangeException**
- **RuntimeException**
- **UnderflowException**
- **UnexpectedValueException**

Функции

- **class_implements** — Возвращает список интерфейсов, реализованных в заданном классе или интерфейсе
- **class_parents** — Возвращает список родительских классов заданного класса
- **class_uses** — Возвращает список трейтов, используемых заданным классом
- **iterator_apply** — Вызывает функцию для каждого элемента в итераторе
- **iterator_count** — Подсчитывает количество элементов в итераторе
- **iterator_to_array** — Копирует итератор в массив
- **spl_classes** — Возвращает доступные классы SPL
- **spl_object_hash** — Возвращает хеш-идентификатор для объекта

Классы

- `print_r(spl_classes());`

Лабораторные работы

Автозагрузка

- **spl_autoload_call** — Попытка загрузить описание класса всеми зарегистрированными методами `__autoload()`
- **spl_autoload_extensions** — Регистрация и вывод расширений файлов для `spl_autoload`
- **spl_autoload_functions** — Получение списка всех зарегистрированных функций `__autoload()`
- **spl_autoload_register** — Регистрирует заданную функцию в качестве реализации метода `__autoload()`
- **spl_autoload_unregister** — Отмена регистрации функции в качестве реализации метода `__autoload()`
- **spl_autoload** — Реализация по умолчанию метода `__autoload()`

Итоги

- Встроенные интерфейсы и классы
- Замыкания
- Генераторы
- SPL – Standard PHP Library
- Общие принципы
- Интерфейсы
- Итераторы
- Классы
- Структуры данных
- Функции
- Лабораторные работы