

1. Настройки и возможности.

Все настройки, в т.ч. таблица маршрутизации, находятся в файле `framework\app\config\config.xml`.

Примечание: работу тестового задания можно увидеть на <http://testfrmmy.kl.com.ua/web/index.php>

Для запуска приложения на другом хосте необходимо задать:

наименование хоста: `<host>localhost</host>`;

путь к `index.php` (без имени хоста): `<baseUrl>/TestFramework/web/index.php</baseUrl>`;

путь к БД: `<connDB>../framework/app/data/students.db</connDB>`;

путь к странице для описания сгенерированных исключений (без имени хоста):
`<excepPage>/TestFramework/framework/exceptions/defExcep.php</excepPage>`.

В `<startRoute>home</startRoute>` - задается маршрут, который будет вызван при запуске приложения.

В `<default></default>` - задается действие и отображение для default-контролера (класс `DefaultController`), который вызовет соответствующее отображение:

```
<default>
    <action>index</action>
    <view>defaultpage</view>
</default>
```

Каждый маршрут задается в теге `route`:

1. `<route name="editfaculty">`
2. `<controller name="FacultyController"/>`
3. `<action name="update">`
4. `<param limitationrule="/^[0-9]+$/">id</param>`
5. `<param defaultvalue="null">faculty_name</param>`
6. `</action>`
7. `<view>faculty</view>`
8. `<status value="RES_ACT_OK">`
9. `<redirect>listfaculty</redirect>`
10. `</status>`
11. `</route>`

В нашем примере в 1-ой строке вносится наименование маршрута;

во 2-ой - имя класса контролера (контролеры расположены в `framework\controllers*.*`);

в 3-ей строке - имя действия (метода) контролера;

в 4-ой и 5-ой – параметры передаваемые в действие. Для параметров могут быть заданы два атрибута `limitationrule` и `defaultvalue`. «`limitationrule`» - задает ограничение (регулярное выражение) на передаваемое значение параметра, если клиент передаст неверное значение то фреймворк сгенерирует собственное исключение и отправит страницу с описанием произошедшей ошибки. «`defaultvalue`» - задает значение параметра по умолчанию, оно используется лишь в случае отсутствия данного параметра в пришедшем запросе. Оба атрибута не являются обязательными и могут не задаваться.

в 7-ой строке - имя представления для данного маршрута, в нашем случае `faculty` – соответствует шаблону `framework\views\faculty.html`. Следует отметить, что в шаблонах `*.html` для подстановки данных используются маркеры (`{mark}`), соответствующие классы `Template View` (расположены в `src\ViewHelper*.php`) готовят данные, самописный

шаблонизатор (framework\engine\PrototypeTemplate.php) заменяет маркеры этими данными, а Response – отправляет полученный html пользователю;

в 8-ой строке - задается значение результата выполнения действия контролера, при котором фреймворк самостоятельно выполнит форвард на маршрут, указанный в строке 9 (<redirect>listfaculty</redirect>). Данные значения определены в родительском классе всех контролеров – ControllerBase ('RES_ACT_DEF', 'RES_ACT_OK', 'RES_ACT_ERR', 'RES_ACT_NO_DATA').

Т.е. если в действии update контролера FacultyController:

```
class FacultyController extends ControllerBase
```

```
{ ...
```

```
    public function update( $id, $faculty_name )
```

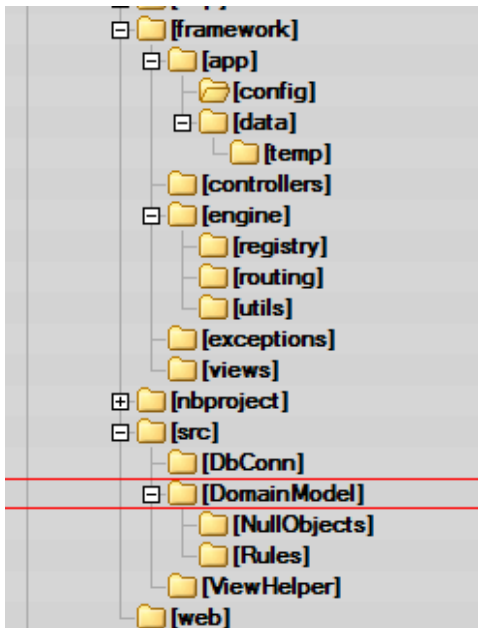
```
    { ... $this->status = self::$enumResultAction['RES_ACT_OK'];
```

```
    ...
```

будет задано значение 'RES_ACT_OK', то по окончании выполнения метода-действия фреймворк автоматически вызовет маршрут listfaculty (строка 9), если же в таблице маршрутизации (config.xml) будет указано несколько значений результата выполнения действия ('RES_ACT_DEF', 'RES_ACT_OK', 'RES_ACT_ERR', 'RES_ACT_NO_DATA') с соответствующими маршрутами для редиректа, то в зависимости от внесенного в код метода контролера значения, фреймворком будет выполнен редирект на соответствующий внесенному значению указанный маршрут. Если значение в коде метода не внести, то редиректа не будет. Следует отметить, что то же самое справедливо и для представлений (строка 7), т.е. если указать:

<view value="RES_ACT_OK">faculty</view>, то представление автоматически будет возвращено, в случае если в коде метода-действия будет внесено: “\$this->status = self::\$enumResultAction['RES_ACT_OK'] ”. Если значения не совпадут – представление не отобразится. Можно указать несколько представлений для соответствующих результатов выполнения и автоматически будет показано представление для заданного в коде результата выполнения действия контролера. Редирект на другие маршруты возможен не только через таблицу маршрутизации, но и с помощью метода programmRedirectToRoute () класса Routing.

2. Структура проекта



framework\app\config\config.xml – таблица маршрутизации и пути;
framework\app\data\temp*. – временные файлы с кэшируемыми данными;
framework\app\data\students.db – база данных со студентами;
framework\controllers*. – родительский контролер, default- контролер и контролеры для работы со студентами;
framework\engine\registry*. – реестры для кэширования данных;
framework\engine\routing*. – Routing и непосредственно связанные с ним классы;
framework\engine\utils\BaseService.php – служебный класс с универсальными методами, которые используются остальными классами;
framework\engine*. – FrontController, шаблонизатор PrototypeTemplate, Request и Response;
framework\exceptions*. – классы собственных исключений фреймворка;
framework\views*.html – шаблоны представлений;
framework\ClassLoader.php – автозагрузчик;
src\DbConn\DbPdo.php – класс соединения с базой данных;
src\DomainModel\NullObjects*. – классы, реализующие решение Null Object для Student, Group, Faculty;
src\DomainModel\Rules*. – классы правил проверки бизнес-логики для данных о студентах, группах, факультетах (эти классы вызываются в классах BaseObject, Student, Group, Faculty);
src\DomainModel*. – классы реализующие решение Active Record для студентов, групп и факультетов;
src\ViewHelper*. – классы реализующие решение Template View (преобразовывают данные в HTML формат);
web\index.php – с него все начинается;

3. Порядок работы

3.1 web\index.php

Вот его код:

```
require_once( "../framework/ClassLoader.php");

use framework\engine\FrontController;

framework\ClassLoader::onAutoLoad();
FrontController::main();
```

Подключаем автозагрузчик классов и запускаем единственный метод FrontController-a - main().

3.2 framework\engine\FrontController.php

FrontController организует и распределяет выполнение запросов.

```
namespace framework\engine;

use framework\engine\registry\FrameworkRegistry;
use framework\engine\routing\RouteMapLoader;
use framework\engine\Request;
use framework\exceptions\FrameworkException;

final class FrontController {

    private function __construct() {}

    public static function main()
    {
        try {
            RouteMapLoader::getInstance()->initialize();
            while( $controller = FrameworkRegistry::getRouting()->
                getController( Request::getInstance() ) ) {
                $controller->run();
            }
        } catch (FrameworkException $frmExcep) {
            $frmExcep->redirectToExcepPage();
        }
    }
}
```

“RouteMapLoader::getInstance()->initialize();” - считываем таблицу маршрутизации и другие настройки, сохраняем их в реестре FrameworkRegistry. В дальнейшем за config-данными обращаемся напрямую к FrameworkRegistry.

“while(\$controller = FrameworkRegistry::getRouting()->getController(Request::getInstance()))” - обращаемся к FrameworkRegistry за Routing, в случае если Routing отсутствует, создаем и сохраняем его в FrameworkRegistry. При создании экземпляра Routing в его конструктор передается таблица маршрутизации (класс

RouteMapManager). “->getController (Request::getInstance())” – создаем и возвращаем экземпляр контролера, соответствующий текущему запросу (Request).

“\$controller->run()” – вызываем метод run (реализован в родителе всех контролеров - ControllerBase), run – вызывает необходимый метод-действие в созданном выше экземпляре контролера (через вызов метода executeAction() класса Routing). В действии контролера выполняем введенный код, формируем данные для представления и вызываем его. Response вызывает метод generateView() шаблонизатора PrototypeTemplate, который подставляет в *.html-шаблон на место маркеров данные из соответствующих Template View. В результате Response возвращает клиенту сформированный *.html.

“catch (FrameworkException \$frmExcep)” – перехватываем сгенерированные исключения, в случае возникновения ошибок, “\$frmExcep->redirectToExcepPage();” - через Response возвращаем страницу с описанием исключения и возможными способами решения.

3.3 framework\engine\routing\Routing.php

Routing - обеспечивает создание экземпляра необходимого контролера и выполнение соответствующего действия. Реализует шаблон проектирования Proxu, перехватывая все обращения к RouteMapManager (таблице маршрутизации), для этого реализуется интерфейс IRouteMapManager (его также реализует RouteMapManager), через его методы общаемся с одноименными методами RouteMapManager. Функционал класса не следует расширять и менять, поэтому – final.

final class Routing implements IRouteMapManager

Рассмотрим наиболее важные методы:

Запрос на получение созданного экземпляра контролера:

```
public function getController( Request $req )
{
    ...
    //возвращаем Default контролер
    if ( !$controllerName ) {
        return $this->getDefaultController();
    }
    ...

    // получаем экземпляр контролера по его имени $controllerName:
    $controllerObj = $this->createController( $controllerName );
    ...
    return $controllerObj;
}

private function createController( $nameController )
{
    ...
    $className = NamespaceController::class . "\\$nameController";
    ...
    $controllerClass = new \ReflectionClass($className);
    ...
    // создание экземпляра контролера по имени $nameController:
    return $controllerClass->newInstance($this);
    ...
}
```

public function executeAction() - обеспечивает выполнение действия данного контролера

public function getViewName() – возвращает имя представления для данного маршрута

public function getActionName(\$route = ") – возвращаем наименование действия-метода для данного маршрута

public function getParametersForAction() - возвращение параметров для действия контролера

public function programmRedirectToRoute(\$nameRedirectRoute, \$arrParams) – иницируем программный редирект на другой маршрут. Следует отметить, что фреймворк позволяет переключаться на другой маршрут двумя способами, как с помощью настройки таблицы маршрутизации: тег <redirect>listfaculty</redirect>, так и с помощью метода programmRedirectToRoute(...).

3.4 framework\engine\routing\ RouteMapManager.php

RouteMapManager – содержит в своих массивах всю информацию о таблице маршрутизации.

3.5 framework\engine\routing\ ParametersRoute.php

ParametersRoute - проверяет, обрабатывает и подготавливает параметры для заданного действия контролера. Параметры из Request сверяются с соответствующими параметрами из RouteMapManager - таблицы маршрутизации (config.xml). В результате получаем массив параметров для передачи в соответствующее действие контролера.

3.6 framework\engine\Request.php

Request - обеспечивает работу с запросом.

private function filterReq() – один из главных методов, разбираем запрос, выделяет и сохраняет маршрут и параметры.

3.7 framework\engine\Response.php

Response - обеспечивает формирование и отправку клиенту ответа.

Основные методы:

Редирект на стартовую страницу приложения:

```
public function redirectToStartPage()
{
    ...
    $this->redirect($baseUrl . FrameworkRegistry::getStartRoute());
}
```

Редирект на страницу с описанием сгенерированного исключения:

```
public function redirectToExcepPage()
{
    $this->redirect(FrameworkRegistry::getExcepPage());
}
```

Формирование контента для отправки ответа клиенту. Шаблонизатор PrototypeTemplate на основе пути к файлу шаблона и массива с данными возвращает HTML представления, заполненного необходимыми данными:

```
public function preparePage( $nameTemplateView, $arrViewData = array() )
{
    $this->setContent((new PrototypeTemplate($nameTemplateView, $arrViewData))-
>generateView());
    return $this;
}
```

3.8 framework\controllers\ControllerBase.php

ControllerBase - Родительский класс для пользовательских контролеров. Из экземпляра Routing получает данные о контролере, действии (методе), аргументах действия, соотв. представлении, возможном маршруте для редиректа из данного действия.

Вызывает заданное действие (метод) контролера:

```
public final function run( )
{
    ...
    $this->routing->executeAction();
    ...
}
```

Иницилируем программный редирект на другой маршрут:

```
public final function programmRedirectToRoute( $nameRoute, $arrParameters = array() )
{
    ...
    $this->routing->programmRedirectToRoute( $nameRoute, $arrParameters );
}
```

Вызов представления. Представление можно вызвать не задавая имени шаблона \$nameTemplateView, тогда имя будет взято из таблицы маршрутизации (тег <view>facultylist</view>), в коде обеспечивается вызовом метода getView(\$arrViewData = array()). А можно явно задать имя шаблона и массив данных для подстановки (\$arrViewData):

```
protected final function getViewByNameTemplate( $nameTemplateView, $arrViewData )
{
    ...
    (new Response())->preparePage($nameTemplateView, $arrViewData)->sendPage();
}
```

3.9 framework\engine\PrototypeTemplate.php

PrototypeTemplate- Прототип шаблонизатора. Заменяет маркеры (заключаются в {}) в файлах-шаблонах соответствующими значениями из передаваемого в конструктор массива \$arrViewData. Шаблонизатор также может включать в main-шаблон другие файлы. В серьезных приложениях конечно лучше воспользоваться smarty, twig... Хотя с учетом студентов PrototypeTemplate вполне справился.

3.10 framework\engine\utils\BaseService.php

BaseService - служебный класс, определяющий ряд универсальных методов, используемых остальными классами.

Например:

static public function strStartsWith(\$source, \$needle) - метод проверяет начинается ли данная строка (\$source) с заданных символов (\$needle).

static public function strEndsWith(\$source, \$needle) - метод проверяет заканчивается ли данная строка (\$source) заданными символами (\$needle).

static public function isId (\$id) - метод проверяет может ли считаться передаваемый параметр целым положительным числом (идентификатором объекта).

static public function isParamCfg (\$str, \$minLength = 2, \$maxLength = 48) - метод проверяет может ли передаваемый параметр считаться корректным значением из таблицы маршрутизации.

и другие...

3.11 framework\exceptions\FrameworkException.php

FrameworkException - родительский класс для собственных классов исключений.

Метод suggestedSolutions () – возвращает описание возможных путей решения ошибок, вызвавших данное исключение. Каждый потомок задает свое сообщение, переопределяя данный метод.

private function getErrorMessage() – возвращает содержимое с описанием исключения для страницы, на которую выполняется редирект во время обработки исключения, за это отвечает метод:

```
public final function redirectToExcepPage ()
{
    SessionRegistry::getInstance()->putCache('msgExcep', $this->getErrorMessage());
    $response = new Response();
    $response->redirectToExcepPage();
}
```

Исключения перехватываются в методе FrontController:: main() и в действиях контролеров.

Реализованы следующие исключения: CreateObjectException, DbException, FileFormatException, FileNotFoundException, InvalidArgumentException, NullReferenceException, RunTimeException, UnhandledException.

3.12 framework\engine\registry*Registry.php

Классы для кэширования данных.

framework\engine\registry\Registry.php – родительский класс.

framework\engine\registry\FrameworkRegistry.php – класс для сохранения служебных данных фреймворка, в т.ч. таблицы маршрутизации, реализует шаблон Singleton.

framework\engine\registry\SessionRegistry.php - реестр для хранения данных клиентского кода фреймворка. В BaseObject – родителе Faculty, Group и Student, есть методы для работы с классом SessionRegistry:

```
public final static function putCache ($key, $value)
{
    SessionRegistry::getInstance()->putCache($key, $value);
}
```

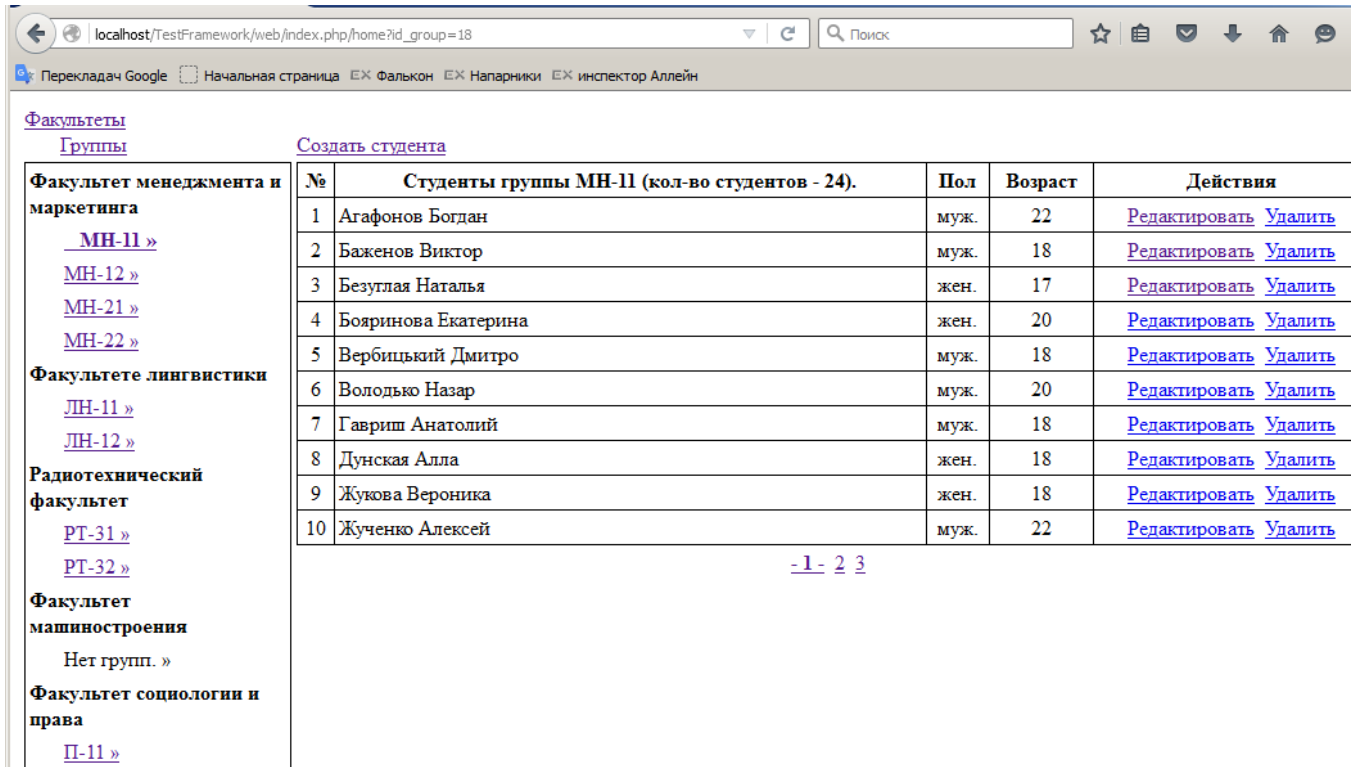
```
public final static function getCache ($key)
{
    return SessionRegistry::getInstance()->getCache($key);
}
```

```
public final static function deleteFromCache ($key)
{
    SessionRegistry::getInstance()->deleteByKey($key);
}
```

SessionRegistry также реализован как Singleton.

4. Задание учет студентов

С использованием тестового фреймворка было выполнено задание по учету студентов.



Факультеты	Создать студента
Группы	
Факультет менеджмента и маркетинга	
МН-11 »	
МН-12 »	
МН-21 »	
МН-22 »	
Факультете лингвистики	
ЛН-11 »	
ЛН-12 »	
Радиотехнический факультет	
РТ-31 »	
РТ-32 »	
Факультет машиностроения	
Нет групп. »	
Факультет социологии и права	
П-11 »	

№	Студенты группы МН-11 (кол-во студентов - 24).	Пол	Возраст	Действия
1	Агафонов Богдан	муж.	22	Редактировать Удалить
2	Баженов Виктор	муж.	18	Редактировать Удалить
3	Безуглая Наталья	жен.	17	Редактировать Удалить
4	Бояринова Екатерина	жен.	20	Редактировать Удалить
5	Вербицкий Дмитро	муж.	18	Редактировать Удалить
6	Володько Назар	муж.	20	Редактировать Удалить
7	Гавриш Анатолий	муж.	18	Редактировать Удалить
8	Дунская Алла	жен.	18	Редактировать Удалить
9	Жукова Вероника	жен.	18	Редактировать Удалить
10	Жученко Алексей	муж.	22	Редактировать Удалить

- 1 - 2 3

В качестве СУБД – sqlite.

В базе (framework\app\data\students.db) создано три таблицы:

Факультеты:

```
CREATE TABLE `faculty` (  
    `id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,  
    `name` TEXT NOT NULL  
);
```

Группы:

```
CREATE TABLE `group_acad` (  
    `id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,  
    `name` TEXT NOT NULL,  
    `id_faculty` INTEGER NOT NULL  
);
```

Студенты:

```
CREATE TABLE `student` (  
    `id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,  
    `surname` TEXT NOT NULL,  
    `firstname` TEXT NOT NULL,  
    `sex` INTEGER NOT NULL,  
    `age` INTEGER NOT NULL,  
    `id_group` INTEGER NOT NULL  
);
```

Класс DbPdo (src\DbConn\DbPdo.php)

В статическом методе возвращает соединение с БД. Класс реализован как Singleton.

Класс BaseObject (src\DomainModel\BaseObject.php)

Родительский класс для Faculty, Group, Student.

Включает в себя общие для всех потомков данные, с помощью SQL-запросов работает с базой данных. Также BaseObject реализует интерфейс IRule, тем самым обеспечивая возможность проверки правил бизнес-логики для всех своих потомков.

Другими словами BaseObject взаимодействует с базой данных и содержит логику домена. Такое поведение подходит под определение решения Active Record. Но, строго говоря, структура данных класса, реализующего Active Record, должна в точности соответствовать записи в соответствующей таблице базы данных, т.е. каждое поле объекта должно соответствовать одному столбцу таблицы или представлению базы данных. Поэтому при строгом подходе к реализации Active Record следовало бы создать лишь три класса - Faculty, Group и Student, поля каждого из которых полностью соответствовали бы полям заданной таблицы БД, связать их SQL-запросами и "наполнить" классы логикой. Логика домена в нашем случае - это правила, накладываемые на данные и их структуру, например, не стоит разрешать удалять группу, если в ней существуют студенты, не следует разрешать удалять факультет, если в нем существуют группы... Есть правила, очень похожие для всех трех классов (Faculty, Group и Student): например, хорошо было бы, чтобы пользователь не внес несколько одинаковых названий факультетов или групп.

Разумеется мы создадим классы реализующие правила проверки (вместе с их родителем - RuleBase).

А в Faculty, Group и Student будем их вызывать, подставляя разные параметры, затем обработаем результаты проверки.

Кроме этого во всех трех таблицах существуют еще и одинаковые поля, например id и name, некоторые запросы у всех трех классов будут напоминать друг друга: 'SELECT * FROM NAME_TABLE WHERE id = :id' или 'DELETE FROM NAME_TABLE WHERE id = :id'. И вероятней всего, что на этой почве мы получим дублирование кода в Faculty, Group и Student... Чтобы избежать дублирования можно(нужно) общее поведение вынести в общего родителя - BaseObject. Но ценой одного из принципов Active Record...

//включаем проверку правил:

```
public function OnCheckRules() { $this->codeModeRules = self::ON_RULES; }
```

//отключаем проверку правил:

```
public function OffCheckRules() { $this->codeModeRules = self::OFF_RULES; }
```

Содержит методы getAllItems() и getSelfById (\$id), которые реализуют шаблон Template method: вызываемые внутри них методы getNameTable(), doSortArrObj(\$arrObj), doGetSelfById(\$resObj) переопределяются во всех потомках Faculty, Group и Student.

В BaseObject находится вся реализация проверки правил:

```
/**
```

```
 * Вызываем проверку правил.
 * Каждый потомок у себя заполнит $this->persistenceRules нужными
 * ему правилами. RuleBase::collectBrokenRules($this->persistenceRules, $brokenRules) -
 * проверит заданные правила и в $brokenRules вернет нарушенные правила, а
 * в RuleBase::getTotalMessage() - описание нарушенных правил.
 * Так как метод универсален и перегружать его в потомках не стоит,
 * пусть будет final, равно как и isValid() и getBrokenRulesTotalMessage().
 *
 * @return array массив нарушенных правил.
 */
```

```

public final function brokenRules()
{
    $brokenRules = array();
    if ($this->getCheckRules() == self::OFF_RULES){
        return $brokenRules;
    }
    RuleBase::collectBrokenRules($this->persistenceRules, $brokenRules);
    $this->checkedRuleMessage = RuleBase::getTotalMessage();
    return $brokenRules;
}

```

/**

```

* Иницируем проверку правил.
* Если $this->brokenRules() вернет массив с нарушенным(и)
* правилом(и) то и isValid() вернет false, а если все
* проверки прошли успешно, то $this->brokenRules() - пуст
* и isValid() возвращает true.
*
* @return bool true - проверка правил прошла успешно, иначе - false.
*/

```

```

public final function isValid()
{
    return count($this->brokenRules()) == 0;
}

```

/**

```

* Возвращает описание нарушенных правил.
*
* @return string описание нарушенных правил.
*/

```

```

public final function getBrokenRulesTotalMessage()
{
    return $this->checkedRuleMessage;
}

```

Класс Faculty (src\DomainModel\ Faculty.php)

Обеспечивает работу с факультетом. Перегружает необходимые методы BaseObject.

public function getGroups() - Возвращает группы данного факультета или array(new NullGroup()) - если факультет не содержит групп.

В методе save() задаем правила, которым должен соответствовать факультет. В случае если правило(а) будут нарушены, сохранение отменяется, а пользователь получит сообщение с описанием своей ошибки:

```

public function save()
{
    $result = false;
    if($this->getCheckRules() == self::ON_RULES){
        $minLengthName = 2; // @var int мин длина наименования факультета
        $maxLengthName = 52; // @var int max длина наименования факультета
        // подключаем правила для проверки:
        $this->persistenceRules = array(new IsIdValidRule($this),
            new IsNameValidRule($this->getName(), $minLengthName, $maxLengthName,
                IsNameValidRule::NAME_DEPARTMENT),
            new CheckAlreadyExistsNameRule(self::getAllItems()->getArrayCopy(), $this));
    }
}

```

```
// проверяем результат проверки:
if (!$this->isValid()){
    return $result;
}
...
```

Результат нарушения правила IsNameValidRule:

Факультет: Внимание: "#\$%" - недопустимое имя.

Результат нарушения правила HasFacultyGroupsRule:

[« На главную страницу](#) [Создать факультет](#)

№	Наименования факультетов	Действия
1	Факультет менеджмента и маркетинга	Редактировать Удалить
2	Факультете лингвистики	Редактировать Удалить
3	Радиотехнический факультет	Редактировать Удалить
4	Факультет машиностроения	Редактировать Удалить
5	Факультет социологии и права	Редактировать Удалить

Внимание, удаление факультета невозможно: В факультете " Факультете лингвистики" есть группы (2). Факультет можно будет удалить после удаления из него всех групп.

Класс Group (src\DomainModel\ Group.php)

Обеспечивает работу с группой. Перегружает необходимые методы BaseObject.

```
private $idFaculty = 0;    //@var int идентификатор факультета группы
private $faculty = null;   //@var object факультет группы
//возвращаем факультет группы:
public function getFaculty()
{
    return ($this->faculty) ? $this->faculty : Faculty::getSelfById($this->idFaculty);
}
public function setFaculty(Faculty $val)
{
    return $this->faculty = $val;
}
```

public function getStudents() - возвращает студентов данной группы или array(new NullStudent()) - если группа не содержит студентов.

В методе save() задаем правила аналогично Faculty, только набор правил будет другим.

Результат нарушения правила CheckAlreadyExistsNameRule:

Факультет:

Группа:

Внимание: "МН-11" - уже существует в БД.

Класс Student (src\DomainModel\Student.php)

Обеспечивает работу со студентом. К фамилии и имени студента добавлены данные о поле и возрасте.

Переопределенный метод getName() возвращает фамилию и имя:

```
public function getName()
{
    return $this->getSurname() . ' ' . $this->getFirstname();
}
```

```
private $idGroup = 0; //@var int идентификатор группы, в которой учится студент
private $group = null; //@var object группа, в которой учится студент
```

```
public function getGroup()
{
    //возвращаем группу:
    return ($this->group) ? $this->group : Group::getSelfById($this->idGroup);
}
public function setGroup($val) { $this->group = $val; }
```

```
/**
 * Возвращаем факультет на котором учится студент.
 *
```

```
* @return object факультет
*/
```

```
public function getFaculty()
{
    return $this->getGroup()->getFaculty();
}
```

В остальном класс аналогичен Group и Faculty. Комментарии в классах есть, поэтому не хотелось бы останавливаться на мелких подробностях.

Класс RuleBase и его потомки (src\ DomainModel\Rules\RuleBase.php)

Абстракция правила бизнес-логики. Все классы реализующие конкретные правила проверки наследуют данный класс.

```
abstract class RuleBase
{
    protected $brokenRuleMessage;    //@var string сообщение о нарушенном правиле
    private static $brokenTotalMessage; //@var string суммарное сообщение о ВСЕХ нарушенных
    правилах

    abstract public function isValid();

    /**
     * Метод вызывает проверку для каждого экземпляра правила ($eachRulesToCheck->isValid())
     * и в случае если правило нарушено копирует его в массив $brokenRules.
     * $brokenRules - ссылочный аргумент, который и вернет клиентскому коду все
     * нарушенные им правила.
     *
     * @param array $rulesToCheck массив с правилами.
     * @param array $brokenRules массив, в который копируются экземпляры нарушенных правил
     (isValid() == false) из $rulesToCheck.
     */
    public static function collectBrokenRules(array $rulesToCheck, array &$brokenRules)
    {
        foreach ($rulesToCheck as $eachRulesToCheck) {
            if (!$eachRulesToCheck->isValid()){
                $brokenRules[] = $eachRulesToCheck;
                self::$brokenTotalMessage .= $eachRulesToCheck->brokenRuleMessage . ' ' . PHP_EOL;
            }
        }
    }

    /**
     * Возвращает суммарное сообщение о нарушенных правилах
     */
    public static function getTotalMessage()
    {
        return self::$brokenTotalMessage;
    }
}
```

Правило проверяющее корректность возраста студента:

```
class CorrectAgeRule extends RuleBase
{
    private $age; //@var int возраст

    public function __construct( $age )
    {
        $this->age = $age;
        $this->brokenRuleMessage = "";
    }
}
```

```

/**
 * Метод проверяет допустимость применения значения $age в качестве возраста.
 *
 * @return bool true - возраст верен, иначе false.
 */
public function isValid()
{
    $result = (BaseService::isId($this->age) && (16 <= $this->age && $this->age <= 50));
    if (!$result){
        $this->brokenRuleMessage = "" . $this->age . " - недопустимый возраст.";
    }
    return $result;
}

```

Всего реализовано 8 правил, среди них правило запрещающее ввод нескольких одинаковых названий, правило запрещающие удаление факультета, имеющего группы, правило запрещающие удаление группы, имеющей студентов ...

Все они находятся в src\ DomainModel\Rules*. * и документированы.

Классы NullFaculty, NullGroup и NullStudent (src\ DomainModel\NullObjects*.*)

Классы реализуют решение Null Object и являются потомками Faculty, Group и Student. Это позволяет избавиться от постоянных проверок на null. В некритических случаях, когда можно обойтись без исключений, возвращаются NullFaculty, NullGroup или NullStudent и пользователь просто видит, благодаря перегруженным методам: «Нет студентов», «Нет групп»...

The screenshot shows a web application interface. At the top, there is a navigation bar with links: "Факультеты", "Группы", and "Создать студента". Below the navigation bar, there is a table with columns: "№", "Студенты группы РТ-31 (кол-во студентов - 0).", and "Пол". The table contains one row with the value "Нет студентов." in the "Студенты" column and "0" in the "Пол" column. To the left of the table, there is a sidebar with a list of faculties and their groups. The faculties listed are: "Факультет менеджмента и маркетинга" (with groups МН-11, МН-12, МН-21, МН-22), "Факультете лингвистики" (with groups ЛН-11, ЛН-12), "Радиотехнический факультет" (with groups РТ-31, РТ-32), "Факультет машиностроения" (with group "Нет групп."), and "Факультет социологии и права" (with group П-11).

Классы FacultyHelper, GroupHelper и StudentHelper (src \ViewHelper *.*)

Классы реализуют решение Template View для отображения данных в шаблонах *.html. Статические методы этих классов преобразуют данные из классов Faculty, Group, Student в формат HTML.

Классы FacultyController, GroupController и StudentController (framework\controllers*.*)

Рассмотрим GroupController, остальные контролеры аналогичны и документированы. Порядок документирования маршрута описан в пункте 1. Таблица маршрутизации и настройки путей (на стр. 1).

Контролер для работы с группами.

```
class GroupController extends ControllerBase
{
/**
 * Вот маршрут для listGroup():
 * <route name="listgroup">
 *   <controller name="GroupController"/>
 *   <action name="listGroup"/>
 *   <view>grouplist</view>
 * </route>
 *
 * Вызов представления (grouplist) для отображения групп
 */
    public function listGroup()
    {
        $this->getView( array('tblgroups' =>GroupHelper::getGroupsWithActions()) );
    }
/**
 * Вот маршрут для insert($id_facul, $group_name, $submit_val):
 * <route name="addgroup">
 *   <controller name="GroupController"/>
 *   <action name="insert">
 *     <param limitationrule="/^[0-9]+$/" defaultvalue="0">id_facul</param>
 *     <param defaultvalue="null">group_name</param>
 *     <param limitationrule="/(start|Сохранить|Отмена)$/"
 * defaultvalue="start">submit_val</param>
 *   </action>
 *   <view>group</view>
 *   <status value="RES_ACT_OK">
 *     <redirect>listgroup</redirect>
 *   </status>
 * </route>
 *
 * Создание группы
 *
 * @param int $id_facul - id факультета
 * @param string $group_name - наименование группы
 * @param string $submit_val - режим выполнения
 */
}
```

```

public function insert($id_facul, $group_name, $submit_val)
{
    if ( ! BaseService::isId($id_facul) ){
        $id_facul = 0;
    }
    switch ($submit_val) {
        case 'start':
            // вызов заданного в табл. маршрутизации представления (group) для ввода данных:
            $this->getView(array('cmb_faq' => FacultyHelper::getComboBox(),
                                'val_group_name' => "",
                                'msg_validate' => ""));
            break;
        case 'Сохранить':
            try {
                $group = new Group ($group_name, $id_facul);
                if ($group->save()){
                    //Группа успешно сохранена.
                    //Задаем статус 'RES_ACT_OK' результата выполнения действия,
                    //для автоматического редиректа на маршрут listgroup
                    //(<redirect>listgroup</redirect>),
                    //который вызовет свое представление с табл. групп:
                    $this->status = self::$enumResultAction['RES_ACT_OK'];
                } else {
                    // имя группы не прошло проверку правил,
                    // поэтому вызываем заданное в табл. маршрутизации
                    // представление (group) для ввода данных,
                    // с описанием нарушенного правила:
                    $this->getView(array('cmb_faq' => FacultyHelper::getComboBox($id_facul),
                                        'val_group_name' => "",
                                        'msg_validate' => 'Внимание: ' . $group-
>getBrokenRulesTotalMessage()));
                }
            } catch (FrameworkException $frmExcep) {
                $this->status = self::$enumResultAction['RES_ACT_ERR'];
                $frmExcep->redirectToExcepPage();
            }
            break;
        case 'Отмена':
            //задаем статус 'RES_ACT_OK' результата выполнения действия,
            //для автоматического редиректа на маршрут listgroup,
            //который вызовет представление с табл. групп:
            $this->status = self::$enumResultAction['RES_ACT_OK'];
            break;
        default:
            throw new FrmworkExcep\RunTimeException('Аргумент $submit_val содержит
недопустимое значение - "' . $submit_val . '".',
                GroupController::class, __METHOD__, 'Клиент изменил значение
аргумента на ошибочное.');
```

...