



## Модуль 07 - Piscine Java

### Отражение

Резюме: Сегодня вы будете разрабатывать собственные фреймворки, использующие механизм отражения.

# Содержание

I	Предисловие	2
II	Инструкции	3
III	Упражнение Работа с классами 00 :	5
IV	Упражнение Аннотации - ИСТОЧНИК 01 :	8
V	Упражнение ORM 02 :	10



# Глава I

## Предисловие

Отражение - это мощный механизм, обеспечивающий работу фреймворков (таких как Spring или Hibernate). Знание принципов работы Java Reflection API гарантирует правильное использование различных технологий для реализации корпоративных систем.

Инструмент Reflection позволяет гибко использовать информацию о классе во время выполнения программы, а также динамически изменять состояние объектов, не используя эту информацию при написании исходного кода.

Одной из возможностей отражения является изменение значений приватных полей извне. Тогда мы можем спросить, не противоречит ли это принципу инкапсуляции, и ответ будет отрицательным :)





## Глава II


### Инструкции

- Используйте эту страницу как единственную ссылку. Не слушайте никаких слухов и домыслов о том, как приготовить раствор.
- Теперь для вас существует только одна версия Java - 1.8. Убедитесь, что компилятор и интерпретатор этой версии установлены на вашей машине.
- Вы можете использовать IDE для написания и отладки исходного кода.
- Код чаще читают, чем пишут. Внимательно прочитайте [документ](#), в котором приведены правила форматирования кода. При выполнении каждой задачи убедитесь, что вы следуете общепринятым [стандартам Oracle](#)
- Комментарии не допускаются в исходном коде вашего решения. Они затрудняют чтение кода.
- Обратите внимание на разрешения ваших файлов и каталогов.
- Для оценки ваше решение должно находиться в вашем GIT-репозитории.
- Ваши решения будут оценивать ваши товарищи по аквариуму.
- Вы не должны оставлять в своем каталоге никаких других файлов, кроме тех, которые явно указаны в инструкциях к упражнению. Рекомендуется изменить свой .gitignore во избежание несчастных случаев.
- Когда вам нужно получить точный вывод в ваших программах, запрещено выводить предварительно рассчитанный вывод вместо правильного выполнения упражнения.
- У вас есть вопрос? Спросите своего соседа справа. В противном случае попробуйте поговорить с соседом слева.
- Ваше справочное пособие: товарищи / Интернет / Google. И еще кое-что. На любой ваш вопрос есть ответ на Stackoverflow. Научитесь правильно задавать вопросы.
- Внимательно прочитайте примеры. В них могут потребоваться вещи, которые не указаны в предмете.
- Используйте "System.out" для вывода

- И да пребудет с вами Сила!
- Никогда не оставляйте на завтра то, что вы можете сделать сегодня ;)

## Глава III

### Упражнение 00: Работа с классами

	Упражнение 00
	Работа с
Входящий каталог : <i>ex00/</i>	
Файлы для сдачи : Reflection-классами	
folder Разрешенные функции :	
Все	

Теперь вам нужно реализовать проект Maven, который взаимодействует с классами вашего приложения. Нам нужно создать как минимум два класса, каждый из которых имеет:

- частные поля (поддерживаемые типы: String, Integer, Double, Boolean, Long)
- публичные методы
- пустой конструктор
- конструктор с параметром
- toString() метод

В этом задании вам не нужно реализовывать методы get/set. Вновь созданные классы должны быть расположены в отдельном пакете classes (этот пакет может быть расположен в других пакетах). Предположим, что в приложении есть классы User и Car. Класс User описан ниже:

```
public class User {
    private String firstName;
    private String lastName;
    private int height;

    public User() {
        this . firstName = "Имя по
        умолчанию"; this . lastName =
        "Фамилия по умолчанию"; this . height
        = 0;
    }

    public User(String firstName, String lastName, int height) { this
        . firstName = firstName;
        this . lastName = lastName;
        this . height = height;
```





```

    }

    public int grow(int value) { this
        . height += value; return
        height;
    }

    @Override
    public String toString() {
        return new StringJoiner(", ", User.class .getSimpleName() + "[", "]")
            .add("firstName= '" + firstName + "' ")
            .add("lastName= '" + lastName + "' ")
            .add("height=" + height)
            .toString();
    }
}

```

Реализованное приложение должно работать следующим образом:

- Предоставляет информацию о классе в пакете classes.
- Позволяет пользователю создавать объекты указанного класса с определенными значениями полей.
- Отображение информации о созданном объекте класса.
- Вызов методов класса.

Пример работы программы:

```

Занятия:
Автом
обиль
пользо-----
вателя

Введите название класса:
-> Пользователь

поля :
    String firstName
    String lastName
    int height
методы:------
    int grow(int)

Давайте создадим
объект. firstName:
-> UserName
lastName:
-> Высота
UserSurname:
=> 185-----
Создан объект: User[firstName= 'UserName ', lastName= 'UserSurname ', height=185]

Введите название поля для изменения:
-> FirstName
Введите значение строки:
=> Имя-----
Объект обновлен: User[firstName= 'Name ', lastName= 'UserSurname ', height=185]

Введите имя метода для вызова:
-> grow(int)
Введите
значение int:
-> 10

Возвращенный
метод: 195

```


- Если метод содержит более одного параметра, необходимо задать значения для каждого из них



- Если метод имеет тип `void`, строка с информацией о возвращаемом значении не отображается
- В сеансе программы возможно взаимодействие только с одним классом; можно изменить только одно поле его объекта и вызвать только один метод
- Вы можете использовать оператор `throws`.

## Глава IV

### Упражнение 01 : Аннотации - ИСТОЧНИК

	Упражнение 01
Аннотации - ИСТОЧНИК	
Входящий каталог :	
ex01/	
Файлы для сдачи : Annotations-	
folder Разрешенные функции :	
Все	

Аннотации позволяют хранить метаданные непосредственно в коде программы. Теперь ваша задача - реализовать класс `HtmlProcessor` (производный от `AbstractProcessor`), который обрабатывает классы со специальными аннотациями `@HtmlForm` и `@HtmlInput` и генерирует код HTML-формы в папке `target/classes` после выполнения команды `mvn clean compile`. Предположим, что у нас есть класс `UserForm`:

```
@HtmlForm(fileName = "user_form.html", action = "/users", method = "post")
public class UserForm {
    @HtmlInput(type = "text", name = "first_name", placeholder = "Enter First
    Name") private String firstName;

    @HtmlInput(type = "text", name = "last_name", placeholder = "Enter Last Name")
    private String LastName;

    @HtmlInput(type = "password", name = "password", placeholder = "Enter Password")
    private String password;
}
```

Затем на его основе создается файл `"user_form.html"` со следующим содержимым:

```
<form action = "/users" method = "post">
<input type = "text" name = "first_name" placeholder = "Enter First Name">
<input type = "text" name = "last_name" placeholder = "Enter Last Name">
<input type = "password" name = "password" placeholder = "Enter Password">
<input type = "submit" value = "Отправить">
</form>
```


- Аннотации `@HtmlForm` и `@HtmlInput` должны быть доступны только во время компиляции.
- Структура проекта - на усмотрение разработчика.



- Для корректной работы с аннотациями рекомендуется использовать специальные настройки maven- compiler-plugin и зависимость auto-service от com.google.auto.service.

## Глава V

### Упражнение 02 : ORM

	Упражнение 02
Каталог для сдачи : <i>ex02/</i>	ORM
Файлы для сдачи : ORM-	
папка Разрешенные	
функции : Все	

Мы уже упоминали, что ORM-фреймворк Hibernate для баз данных основан на рефлексии. Концепция ORM позволяет автоматически отображать реляционные связи на объектно-ориентированные. Такой подход делает приложение полностью независимым от СУБД. Вам необходимо реализовать тривиальную версию такого ORM-фреймворка.

Предположим, что у нас есть набор классов модели. Каждый класс не содержит зависимостей от других классов, а его поля могут принимать только следующие типы значений: String, Integer, Double, Boolean, Long. Зададим определенный набор аннотаций для класса и его членов, например, для класса User:

```
@OrmEntity(table = "simple_user")
public class User {
    @OrmColumnId
    private Long id;
    @OrmColumn(name = "first_name", length = 10)
    private String firstName;
    @OrmColumn(name = "first_name", length = 10)
    private String LastName;
    @OrmColumn(имя "возраст")
    private Integer age;

    // задатчики/геттеры
}
```

Разработанный вами класс OrmManager должен генерировать и выполнять соответствующий SQL-код при инициализации всех классов, помеченных аннотацией @OrmEntity. Этот код будет содержать команду CREATE TABLE для создания таблицы с именем, указанным в аннотации. Каждое поле класса, помеченное аннотацией @OrmColumn, становится столбцом в этой таблице. Поле, помеченное аннотацией @OrmColumnId, указывает, что должен быть создан идентификатор с автоматическим увеличением. OrmManager также должен поддерживать следующий набор операций (для каждой из них также генерируется



соответствующий SQL-код в Runtime):

```
public void save(Object entity)
public void update(Object entity)
public <T> T findById(Long id, Class<T> aClass)
```

- OrmManager должен обеспечить вывод сгенерированного SQL на консоль во время выполнения.
- При инициализации OrmManager должен удалить созданные таблицы.
- Метод обновления должен заменить значения в столбцах, указанных в сущности, даже если значение поля объекта равно null.