Day 07 - Piscine Java

Reflection

Резюме: Сегодня вы разработаете собственные фреймворки, использующие механизм рефлексии

Contents

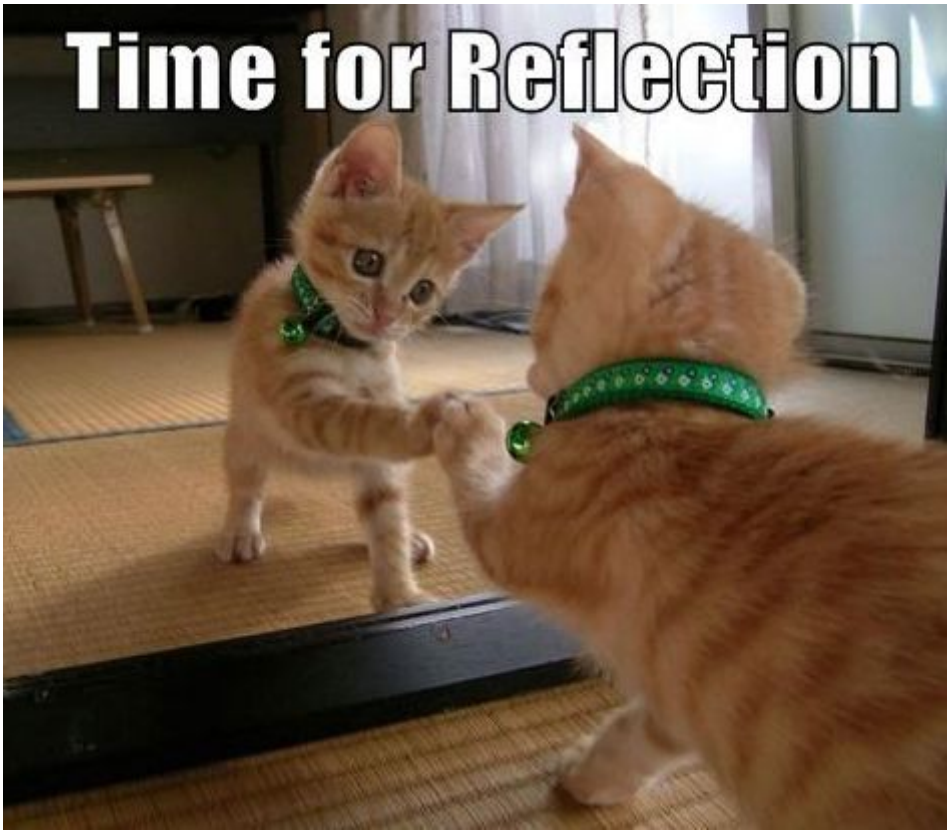
Chapter I
Preamble

Рефлексия - мощный механизм для обеспечения работы фреймворков (например Spring и Hibernate). Понимание принципов работы Java Reflection API гарантирует корректное использование различных технологий для реализации корпоративных систем.

Инструмент рефлексии позволяет гибко использовать информацию о классах в момент исполнения программы, а также динамически изменять состояние объектов без использования такой информации при написании исходного кода.

Так, одной из возможностей рефлексии является изменение значения приватных полей объекта извне. Возникает логичный вопрос, не нарушает ли это принцип инкапсуляции?

Нет :)



Chapter II
General Rules

- Use this page as the only reference. Do not listen to any rumors and speculations about how to prepare your solution.
- Сейчас для вас существует только одна версия Java - 1.8. Убедитесь, что на вашем компьютере установлен компилятор и интерпретатор данной версии.
- Не запрещено использовать IDE для написания исходного кода и его отладки.
- Код чаще читается, чем пишется. Внимательно изучите представленный документ с правилами оформления кода. В каждом задании обязательно придерживайтесь общепринятых стандартов Oracle - https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html
- Комментарии в исходном коде вашего решения запрещены. Они мешают восприятию.
- Pay attention to the permissions of your files and directories.
- To be assessed your solution must be in your GIT repository.
- Your solutions will be evaluated by your piscine mates.
- You should not leave in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid accidents.
- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.
- Have a question? Ask your neighbor on the right. Otherwise, try with your neighbor on the left.
- Your reference manual: mates / Internet / Google. И еще, для любых ваших вопросов существует ответ на Stackoverflow. Научитесь правильно их задавать.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- And may the Force be with you!
- Не откладывайте на завтра то, что можно было сделать вчера ;)

Chapter III
Exercise 00 - Work with classes

Exercise 00: Work with classes	
Turn-in directory	ex00
Files to turn-in	Reflection-folder

Сейчас вам необходимо реализовать maven-проект, взаимодействующий с классами вашего приложения. Необходимо создать не менее 2-х классов, каждый из которых будет иметь:

- приватные поля (допустимы типы `String`, `Integer`, `Double`, `Boolean`, `Long`)
- публичные методы
- пустой конструктор
- конструктор с параметром
- метод `toString()`

В данном задании не требуется реализация `get/set`-методов. Созданные классы необходимо разместить в отдельном пакете `classes` (данный пакет может находиться внутри других пакетов). Пусть приложение имеет классы `User` и `Car`. Описание класса `User` приведено ниже:

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

   public User() {
       this.firstName = "Default first name";
       this.lastName = "Default last name";
       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:

- Предоставлять информацию о каком-либо классе пакета `classes`.
- Позволять пользователю создавать объекты указанного класса с конкретными значениями полей.
- Выводить информацию о созданном объекте класса.
- Вызывать методы класса.

Пример работы приложения:

```
Classes:
    -  User
    -  Car
---------------------
Enter class name:
-> User
---------------------
fields:
       String firstName
       String lastName
       int height
methods:
       int grow(int)
---------------------
Let’s create an object.
firstName:
-> UserName
lastName:
-> UserSurname
height:
-> 185
Object created: User[firstName='UserName', lastName='UserSurname', height=185]
---------------------
Enter name of the field for changing:
-> firstName
Enter String value:
-> Name
Object updated: User[firstName='Name', lastName='UserSurname', height=185]
---------------------
Enter name of the method for call:
-> grow(int)
Enter int value:
-> 10
Method returned:
195
```

- Если метод содержит более одного параметра, необходимо задать значение каждого из них.
- Если метод имеет тип `void`, строка с информацией о возвращаемом значении не выводится
- В одну сессию работы программы можно взаимодействовать только с одним классом, изменять одно поле его объекта, вызывать ровно один метод.
- Разрешено использование оператора `throws`

Chapter IV
Exercise 01 - Annotations - SOURCE

Exercise 01: Annotations - SOURCE	
Turn-in directory	ex01
Files to turn-in	Annotations-folder

Аннотации позволяют хранить метаинформацию непосредственно в коде программы. Сейчас ваша задача - реализовать класс `HtmlProcessor` (потомок `AbstractProcessor`), обрабатывающий классы со специальными аннотациями `@HtmlForm` и `@Htmlnput` и генерирующий на их основе код HTML-формы внутри папки `target/classes` после выполнения команды `mvn clean compile`. Пусть 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;
}
```

Тогда на его основе должен быть сгенерирован 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 = "Send">
</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 плагина `maven-compiler-plugin` и зависимость `auto-service` от `com.google.auto.service`.

Chapter V
Exercise 02 - ORM

Exercise 02: ORM	
Turn-in directory	ex02
Files to turn-in	ORM-folder

Как было сказано ранее, ORM-фреймворк для работы с базами данных Hibernate базируется на использовании рефлексии. Концепция ORM позволяет отобразить реляционные связи в объектно-ориентированные автоматически. Такой подход делает приложение полностью независимым от СУБД. Вам необходимо реализовать тривиальную версию такого ORM-фреймворка.

Пусть есть некоторый набор классов-моделей. Каждый класс не содержит зависимостей на другие классы, а его поля могут иметь только следующие типы - `String`, `Integer`, `Double`, `Boolean`, `Long`. Укажем для самого класса и его членов определенный набор аннотаций, 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(name “age”)
	private Integer age;

	// setters/getters
}
```

Разработанный вами класс `OrmManager` при своей инициализации для всех классов, помеченных аннотацией `@OrmEntity`, должен генерировать и выполнять соответствующий SQL-код. Такой код будет содержать `CREATE TABLE`-команду для создания таблицы с указанным в аннотации названием. Каждое поле класса, помеченное аннотацией `@OrmColumn`, станет столбцом этой таблицы. Поле, помеченное аннотацией `@OrmColumnId`, указывает на необходимость создания автоинкрементного идентификатора. Также `OrmManager` должен поддерживать следующий набор операций (для каждой из которых 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 обеспечивать вывод генерируемого SQL в консоль при выполнении.
- При инициализации `OrmManager` должен удалять уже созданные таблицы.
- Метод `update` должен заменять значения столбцов, указанных в сущности, даже если значение поля объекта `null`.

CHECKLIST

https://docs.google.com/document/d/1WLBFZ4Au2dVG-yFp49GstqlWcGJMYYb4seYNIOjS1e4/edit?usp=sharingxNjg/edit?usp=sharing