Java Pro 7.1

TEL-RAN
by Starta Institute

Рефлексия.

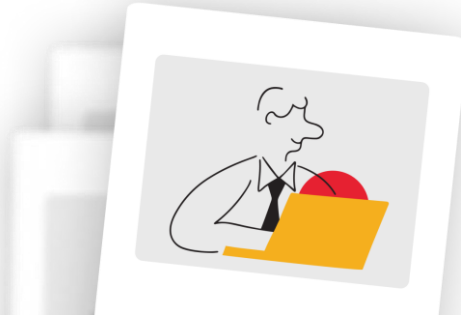
ПРЕПОДАВАТЕЛЬ

TEL-RAN
by Starta Institute

Юрий Костяной

Java/Kotlin backend-разработчик

- 3+ года опыта в коммерческой разработке
- 2+ года опыта в преподавании
- Проекты по интеграции сторонних платформ, CRM
- Проблемно-ориентированный подход в преподавании



ВАЖНО:

- Камера должна быть включена на протяжении всего занятия.
- Если у Вас возник вопрос в процессе занятия, пожалуйста, поднимите руку и дождитесь, пока преподаватель закончит мысль и спросит Вас, также можно задать вопрос в чате или когда преподаватель скажет, что начался блок вопросов.
- Организационные вопросы по обучению решаются с кураторами, а не на тематических занятиях.
- Вести себя уважительно и этично по отношению к остальным участникам занятия.
- Во время занятия будут интерактивные задания, будьте готовы включить камеру или демонстрацию экрана по просьбе преподавателя.

TEL-RAN
by Starta Institute

ПЛАН ЗАНЯТИЯ

1. Повторение
2. Основной блок
3. Вопросы по основному блоку
4. Домашняя работа

TEL-RAN
by Starta Institute

1 ОСНОВНОЙ БЛОК

Введение

TEL-RAN
by Starta Institute

- Смотримся в зеркало
- Делаем замечания

Проблема

TEL-RAN
by Starta Institute

Представьте, что после рождественских каникул Вы впервые заглянули в зеркало. По округлившемуся лицу становится понятно, что пора завязывать с рождественскими сладостями. А мешки под глазами говорят о том, что стоит высыпаться вместо поздних тусовок с друзьями. Наше отражение помогает понять, что можно улучшить в нашей жизни, чтобы выглядеть и чувствовать себя лучше.

А может ли программа «посмотреть в зеркало», чтобы лучше работать?

Смотримся в зеркало

Java Reflection API

TEL-RAN
by Starta Institute

Рефлексия (от лат. *reflexio* – обращение назад) – это механизм исследования данных о программе во время её выполнения.

Java Reflection API позволяет исследовать данные сборок, типов данных, интерфейсов, методов с их параметрами, полей, свойств и событий путем получения информации, описывающей их структуру класса, и, как следствие, создавать корректный динамический код.

Вы буквально можете жонглировать классами и их составляющими.

Смотримся в зеркало

Возможности Java Reflection API

- Узнать/определить класс объекта;
- Получить информацию о модификаторах класса, полях, методах, константах, конструкторах и суперклассах;
- Выяснить, какие методы принадлежат реализуемому интерфейсу/интерфейсам;
- Создать экземпляр класса, причем имя класса неизвестно до момента выполнения программы (*полиморфизм*);
- Получить и установить значение поля объекта по имени;
- Вызвать метод объекта по имени.

Во многих языках (Pascal, C, C++ и др.) рефлексия отсутствует, а, значит, и отсутствуют многие возможности. Почти все фреймворки Java (например, Spring, JUnit, Hibernate) и многие библиотеки (Jackson, JAXB) так или иначе используют возможности рефлексии.

Смотримся в зеркало
Класс Class

Пакет *java.lang.reflect* содержит всё необходимое для работы с рефлексией.

java.lang.Class – это основной класс для получения данных о членах класса.

Способы получения объектов типа Class приведены в примере:

```
import person.Person;

public class Main {
    public static void main(String[] args) {
        // Получить объект Class явно
        Class<Person> c1 = Person.class;
        System.out.println(c1.getName());

        // Получить объект Class по строке имени класса
        try {
            Class<Person> c2 =
                    (Class<Person>) Class.forName("person.Person");
            System.out.println(c2.getName());
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        }

        // Получить объект Class по типу данных
        Person person = new Person("Jim", 50);
        Class<Person> c3 = (Class<Person>) person.getClass();
        System.out.println(c3.getName());
    }
}
```

Смотримся в зеркало

Класс Class

Class позволяет получить необходимую информацию о самом классе. Например, имя класса в полном, простом или каноническом виде.

```
System.out.println("Name: " + c3.getName()); // полное имя с пакетом
System.out.println("Simple name: " + c3.getSimpleName()); // только имя класса
System.out.println("Canonical name: " + c3.getCanonicalName()); // имя в спецификации Java
System.out.println("Name of array: " + int[].class.getName());
System.out.println("Simple name of array: " + int[].class.getSimpleName());
System.out.println("Canonical name of array: " + int[].class.getCanonicalName());
```

Можно получить также информацию о пакете:

```
Package pkg = c3.getPackage();
System.out.println(pkg.getName());
```

```
Name: person.Person
Simple name: Person
Canonical name: person.Person

Name of array: [I
Simple name of array: int[]
Canonical name of array: int[]

person
```

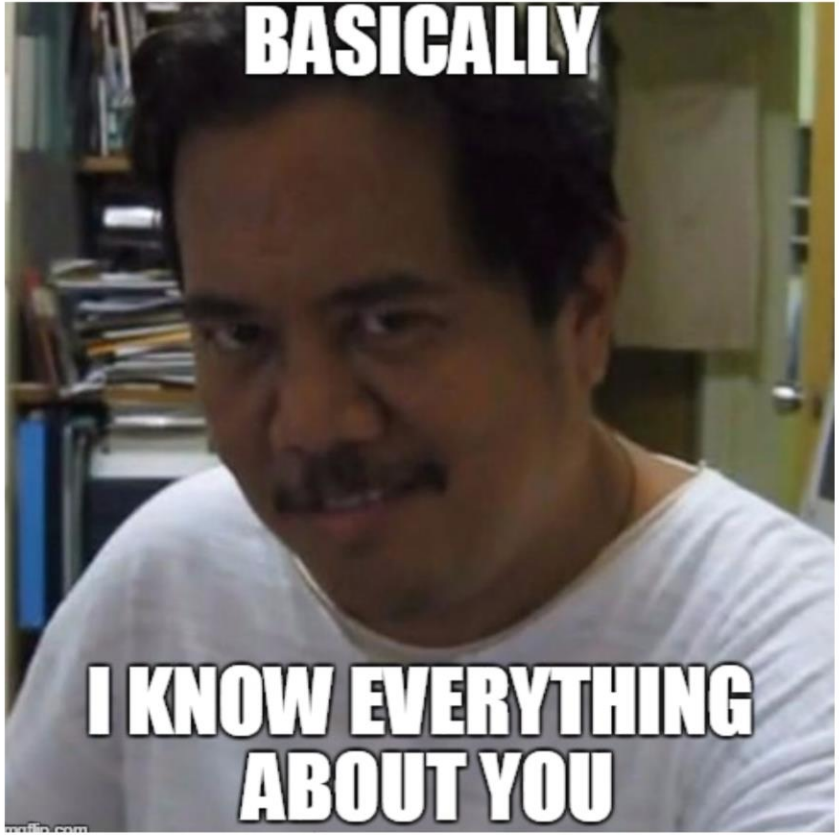
Смотримся в зеркало

Класс Class

У класса можно получить модификаторы указанные при объявлении. Метод *getModifiers()* возвращает число *int*, в котором закодирован состав всех модификаторов поля (в т.ч. *final*, *static* и т.д.). Для интерпретации числа используется класс утилитарный класс *Modifier*.

```
public class Person {...}
```

```
int mod = Person.class.getModifiers();
System.out.println(mod); // 1
System.out.println(Modifier.isPublic(mod)); // true
System.out.println(Modifier.isAbstract(mod)); // false
```



Смотримся в зеркало

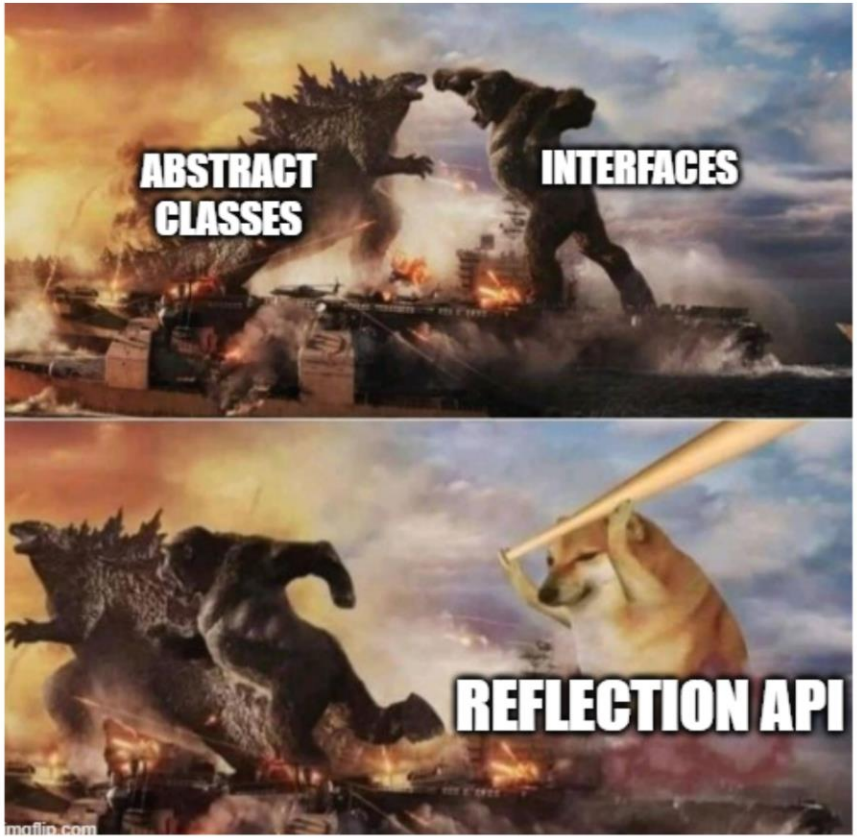
Класс Class

Чтобы получить предка текущего класса, применяется метод *getSuperclass()*.

Для получения данных о наследуемых интерфейсах класса используется метод *getInterfaces()*.

```
Class<?> persParent = Person.class.getSuperclass();
System.out.println(persParent.getSimpleName());

Class<?>[] personInterfaces =
person.getClass().getInterfaces();
System.out.println(Arrays.stream(personInterfaces)
   .map(Class::getSimpleName).toList());
```



Смотримся в зеркало

Класс Class

С помощью метода *isInstance()* можно выполнить проверку *instanceof* с помощью рефлексии без создания экземпляра класса.

Явное приведение можно выполнить, используя метод *cast()*.

```
public class MainInstance {
    private static class A { }
    private static class B extends A { }

    public static void main(String[] args) {
        try {
            Class<?> cls = A.class;
            boolean b1 = cls.isInstance(new Integer(37)); // false
            System.out.println(b1);
            boolean b2 = cls.isInstance(new A()); // true
            System.out.println(b2);
            boolean b3 = cls.isInstance(new B()); // true
            System.out.println(b3);
            A ab = new B();
            B bb = B.class.cast(ab); // эквивалентно (B)ab
        } catch (Exception e) {
            System.err.println(e);
        }
    }
}
```

Задание

TEL-RAN
by Starta Institute

Исследуйте класс String. Выведите в консоль

- имя полное, простое и каноническое имя класса
- в каком пакете находится
- какие модификаторы имеет
- наследником какого класса является
- какие интерфейсы наследует

Смотримся в зеркало

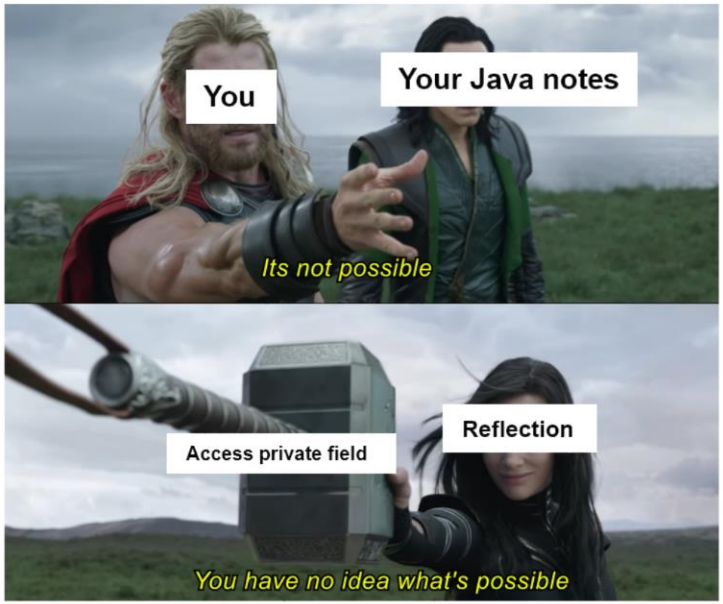
Рефлексия полей класса

TEL-RAN
by Starta Institute

Для получения всех полей, объявленных в классе, достаточно вызвать метод *getDeclaredFields()* у объекта типа *Class*. Метод вернёт все поля, включая приватные. Метод не может получить наследуемые поля. Для их получения нужно сначала получить объект *Class* класса-предка, и уже у него вызвать *getDeclaredFields()*.

Метод *getFields()* позволяет вернуть все поля объекта, включая наследуемые, но только публичные. Поэтому при получении полей методом *getDeclaredFields()* их приходится фильтровать по модификатору доступа (так же при помощи *Modifier*).

Одно поле можно получить с помощью метода *getField(fieldName)* и *getDeclaredField(fieldName)*.



Смотримся в зеркало

Рефлексия полей класса

TEL-RAN
by Starta Institute

У полей можно получить значения или установить их, используя объект класса *Field*.

Для этого используются методы *get()* и *set()* соответственно.

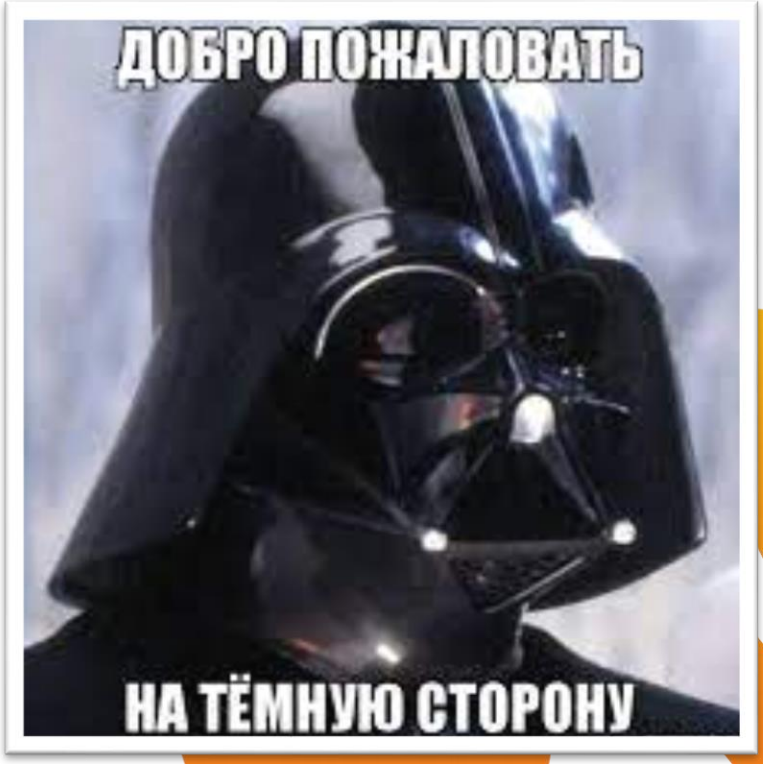
Однако если у поля установлен ограничивающий модификатор доступа, то будет выброшено исключение *IllegalAccessException*. Чтобы этого избежать, достаточно предварительно вызвать метод *setAccessible(true)*.

Смотримся в зеркало

Рефлексия полей класса

TEL-RAN
by Starta Institute

Работа с приватными членами класса через *Reflection API* является нарушением принципа инкапсуляции ООП и ведёт к усложнению логики работы программы. Код с рефлексией сложнее проверять, поэтому применение рефлексии должно быть обосновано отсутствием других вариантов решения проблемы.



Смотримся в зеркало

Пример рефлексии полей класса

TEL-RAN
by Starta Institute

ReflectionExample.zip

Задание

TEL-RAN
by Starta Institute

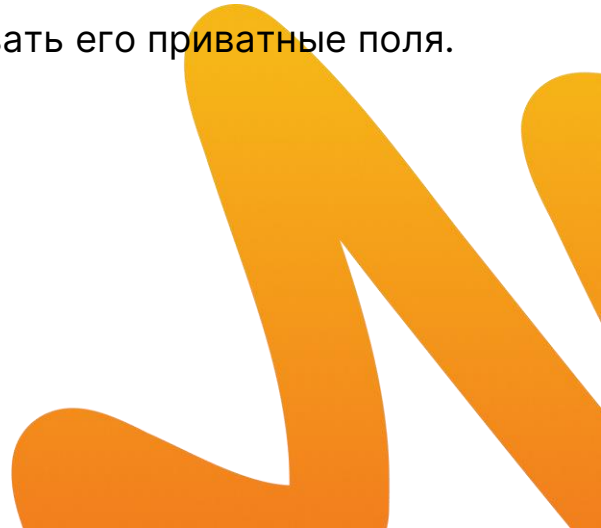
Создайте иерархию классов световых приборов с тремя поколениями. Например,

1. световой прибор (яркость свечения, цвет)
2. электрическая лампа (потребляемая мощность, тип цоколя)
3. умная лампа (перечень доступных функций, тип подключения к сети передачи данных)

Составьте мапу, в которой классу иерархии будут соответствовать его приватные поля.

Выведите мапу в консоль.

Всем строковым полям присвойте значение «abc».



Смотримся в зеркало

Рефлексия методов класса

Метод *getMethods()* получает все публичные методы класса.

Метод *getDeclaredMethods()* получает все методы класса, включая приватные.

Получить отдельный метод можно по его имени и типам аргументов:

getDeclaredMethod("methName", Class... args).

Класс *Method* предоставляет набор методов:

invoke(myObject, arg1, arg2...) – запуск метода с аргументами *arg1*, *arg2* и т.д.;

getParameterTypes() – возвращает массив типов аргументов (*Class[]*);

getExceptionTypes() – возвращает массив типов выбрасываемых методом исключений (*Class[]*);

getReturnType() – возвращает тип возвращаемого методом значения (*Class*);

Смотримся в зеркало

Пример рефлексии полей класса

TEL-RAN
by Starta Institute

ReflectionExample.zip

Задание

Создайте метод, который принимает переменную типа List. Выведите описание всех методов переданного списка. В main вызовите метод, передав в него

- экземпляр ArrayList
- экземпляр LinkedList
- экземпляр, полученный методом List.of().

Смотримся в зеркало

Рефлексия конструкторов класса

Метод *newInstance()* позволяет создать экземпляр класса, вызывая конструктор по умолчанию. Для вызва любого конструктора можно использовать метод *getConstructor(paramTypes).newInstance(argValues...)* Получить типы аргументов конструктора также можно с помощью метода *getParameterTypes()*.

```
public static void main(String[] args) {
    try {
        Class<?> clazz = Class.forName(Person.class.getName());
        // работает только с конструктором по умолчанию
        Person person = (Person) clazz.newInstance();
        System.out.println(person);
        // вызов конструктора с параметрами
        Class[] params = {String.class, int.class};
        person = (Person) clazz.getConstructor(params)
            .newInstance("Frankenstein's Monster", 100);
        System.out.println(person);
    } catch (ClassNotFoundException | InstantiationException |
IllegalAccessException | InvocationTargetException |
            NoSuchMethodException e) {
        e.printStackTrace();
    }
}
```

Смотримся в зеркало

Пример конструкторов полей класса

ReflectionExample.zip

Задание

Создайте экземпляр ArrayList с помощью рефлексии. Добавьте в список несколько элементов. Склонируйте список с помощью рефлексии.

TEL-RAN
by Starta Institute

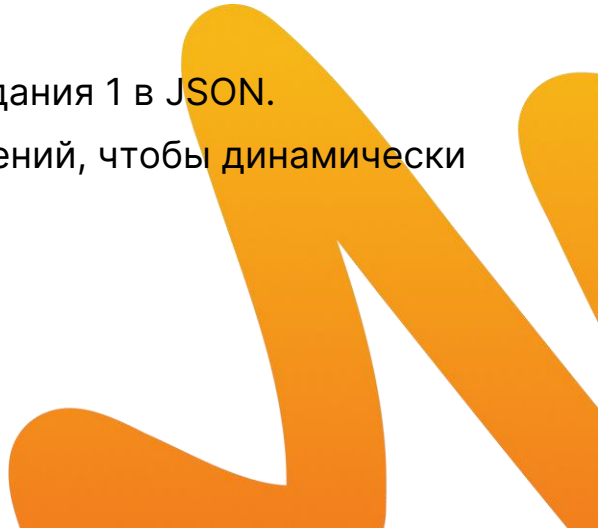
2 Домашнее задание

Домашнее задание

TEL-RAN
by Starta Institute

1 Создайте класс с 10 методами и 4 приватными полями. Геттеры и сеттеры добавлять не нужно. Создайте объект Вашего класса. Выведите в консоль информацию об объекте. Затем пользователь вводит имя поля или метода. Программа определяет, имя какого члена класса было введено. Если введено имя метода, то этот метод немедленно вызывается. Если введено имя поля, то программа запрашивает у пользователя новое значение и устанавливает его новым значением этого поля.

2 Создайте собственный класс для сериализации объекта из задания 1 в JSON. Используйте Reflection API для анализа полей объекта и их значений, чтобы динамически создавать JSON-представление объекта.



ЗАКЛЮЧЕНИЕ

TEL-RAN
by Starta Institute