

# Reflection API Аннотации

Наставник:  
Полковников Дмитрий



# Введение

**Рефлексия** — мощный инструмент Java, который позволяет творить “магию”, но которым нужно уметь пользоваться.

Рефлексия и аннотации повсеместно используется при разработке веб-приложений с использованием фреймворка Spring



# План

- Рефлексия: определение и возможности Reflection API.
- Примеры использования рефлексии:
  - Получение списка полей и методов.
  - Получение доступа к `private` полям и методам.
  - Вызов конструкторов.
- Аннотации: определение и причины их использования.
- Аннотации: базовый синтаксис.
- Демо
- Итог



# Рефлексия: определение и

**возможности** — механизм получения и модификации программы во время ее выполнения.

- Получение информации о классах и объектах.
- Получение доступа к полям и методам классов, в том числе приватным.
- Доступ к конструкторам классов.
- Вызов методов объектов по имени.



# Получение списка полей, методов и

// Класс с которым будем работать

```
public class Test {  
    private String value;  
  
    public String getValue() { return value; }  
}
```

// Получение информации об объекте

```
public static void getClassInfo(Object obj) {  
    Class<?> aClass = obj.getClass();  
    Method[] methods = aClass.getDeclaredMethods();  
    Field[] fields = aClass.getFields();  
    Field[] declaredFields = aClass.getDeclaredFields();  
    Constructor<?>[] constructors = aClass.getConstructors();  
}
```

// class Test  
// [public java.lang.String Test.getValue()]  
// []  
// [private java.lang.String Test.value]  
// [public Test()]



# Изменение private поля

// Метода для изменения значение private поля

```
public static void setPrivateValue(Object obj, String fieldName, String newValue) {  
    Class<?> aClass = obj.getClass();  
    try {  
        Field field = aClass.getDeclaredField(fieldName);    // Получение необходимого поля по имени  
        field.setAccessible(true);                            // Изменение прав доступа к полю  
        field.set(obj, newValue);                             // Установка нового значения поля для объекта  
    } catch (NoSuchFieldException | IllegalAccessException e) {  
        e.printStackTrace();  
    }  
}
```

```
public static void main(String[] args) {  
    var test = new Test();  
    System.out.println(test);                                // Test{value='null'}  
    setPrivateValue(test, "value", "newValue");  
    System.out.println(test);                                // Test{value='newValue'}  
}
```



# Вызов метода по имени

```
// Метод для вызова метода по имени
public static void callPrivateMethod(Object obj, String methodName, String parameter) {
    Class<?> aClass = obj.getClass();
    try {
        Method method = aClass.getDeclaredMethod(methodName, String.class); // Получение необходимого метода
        method.setAccessible(true); // Изменение прав доступа к метода
        method.invoke(obj, parameter); // Вызов метода
    } catch (IllegalAccessException | NoSuchMethodException | InvocationTargetException e) {
        e.printStackTrace();
    }
}

public static void main(String[] args) {
    var test = new Test();
    System.out.println(test); // Test{value='null'}
    callPrivateMethod(test, "setValue", "newValue");
    System.out.println(test); // Test{value='newValue'}
}
```



# Создание объекта класса

```
public static Test createTestObjectWithValue(String value) {  
    try {  
        var constructor = Test.class.getDeclaredConstructor(String.class); // Получение конструктора класса  
        constructor.setAccessible(true); // Изменение прав доступа к метода  
        return constructor.newInstance(value); // Создание нового объекта  
    } catch (Exception e) {  
        throw new RuntimeException(e);  
    }  
}  
  
public static void main(String[] args) {  
    var test = createTestObjectWithValue("newValue");  
    System.out.println(test); // Test{value='newValue'}  
}
```





# Рефлексия: определение и

**возможности**  
Аннотация — специальная синтаксическая конструкция, позволяющая описывать метаданные классов/полей/методов и пр.

Аннотации используются для хранения дополнительной информации и использования ее на различных этапах работы с кодом.



# Аннотации: синтаксис

```
@Retention(RetentionPolicy.RUNTIME)    // Видимость аннотации: SOURCE, CLASS, RUNTIME
@Target(ElementType.METHOD)           // Типы аннотируемых элементов: TYPE, FIELD, METHOD, PARAMETER и пр.
public @interface Cmd {
    String name();                       // Параметры аннотаций
    String description();
}

// Пример использования
public class Commands {
    @Cmd(name = "Exit", description = "ExitApplication")
    public void exit() {
        System.out.println("exit");
    }
}
```



# Демо

- Среда разработки: IntelliJ Idea 2022.3
- SDK: OpenJDK 17.0.5



# Итог

- Рассмотрели возможности, которые предоставляет Reflection API
  - Доступ к private элементам объектов.
  - Вызов методов по имени.
  - Создание объектов по имени класса.
- Узнали что такое аннотации, как они разрабатываются и для чего могут пригодиться



# Спасибо за внимание

