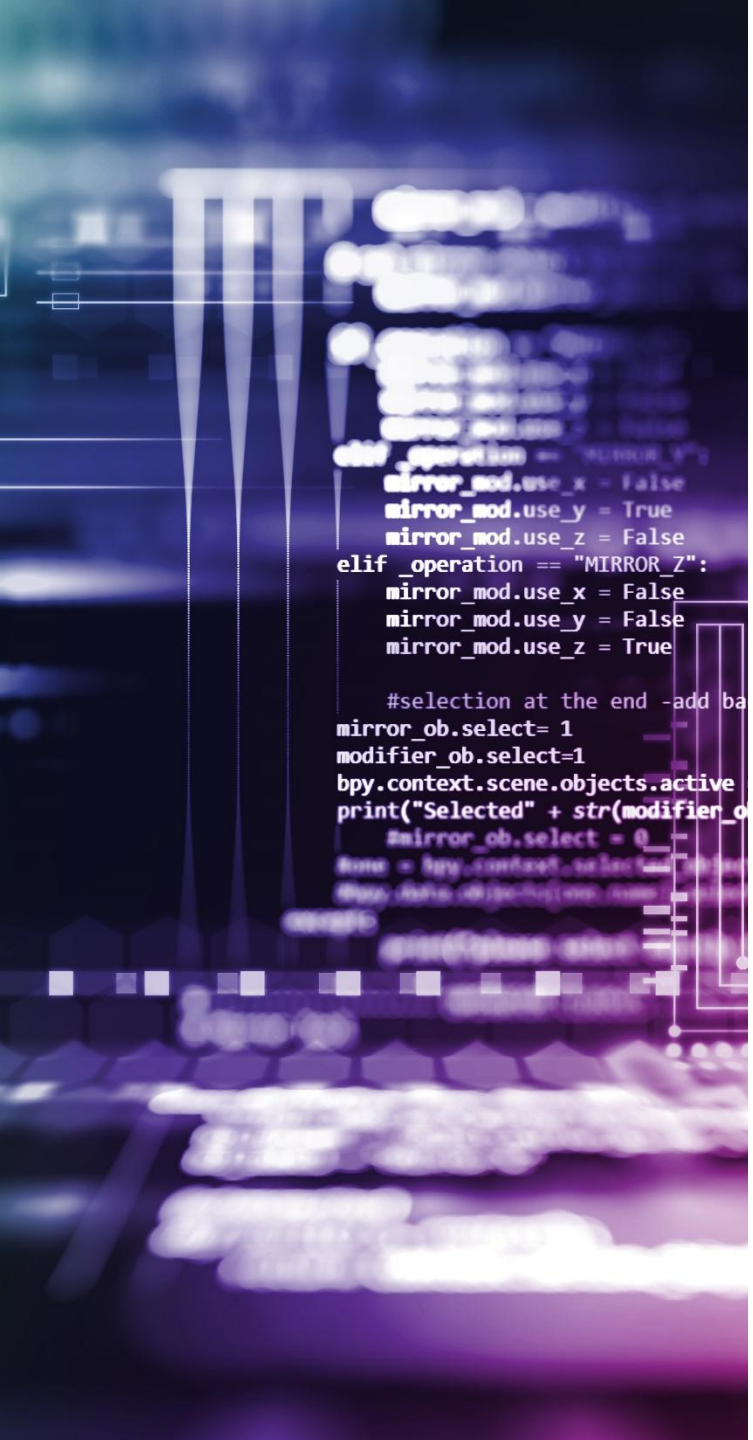
КРОК

Java:
Annotations & Reflection

Александр Головин

Ведущий инженер-разработчик

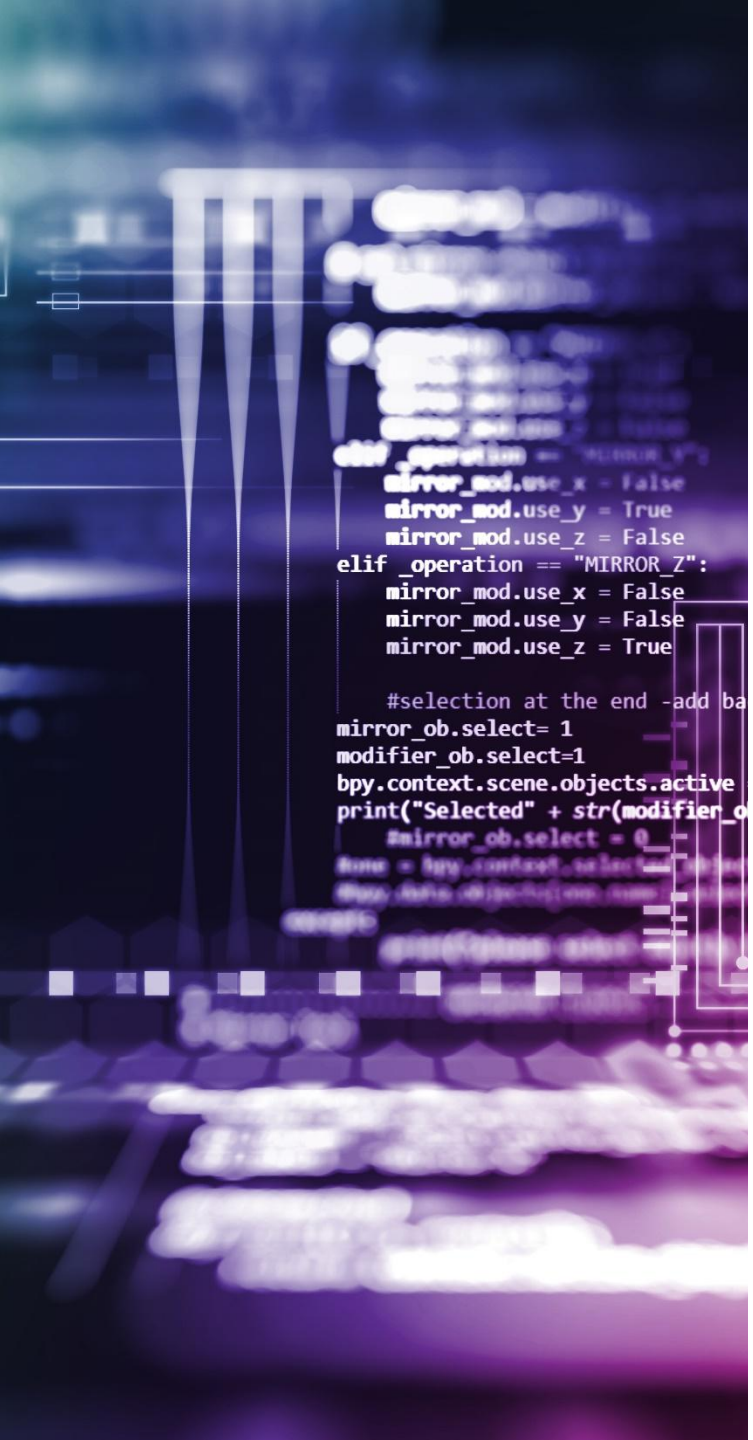


Annotations

Метаданные предоставляющие дополнительную информацию об элементах программы.

Могут использоваться:

- **Компилятором**
- **IDE**
- **Процессором аннотаций**
- **Анализатором байткода**
- **Runtime**



Annotations

```
@Documented
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.SOURCE)
public @interface MarkedClass {
    // nothing
}

@MarkedClass
class MyClass {

}
```

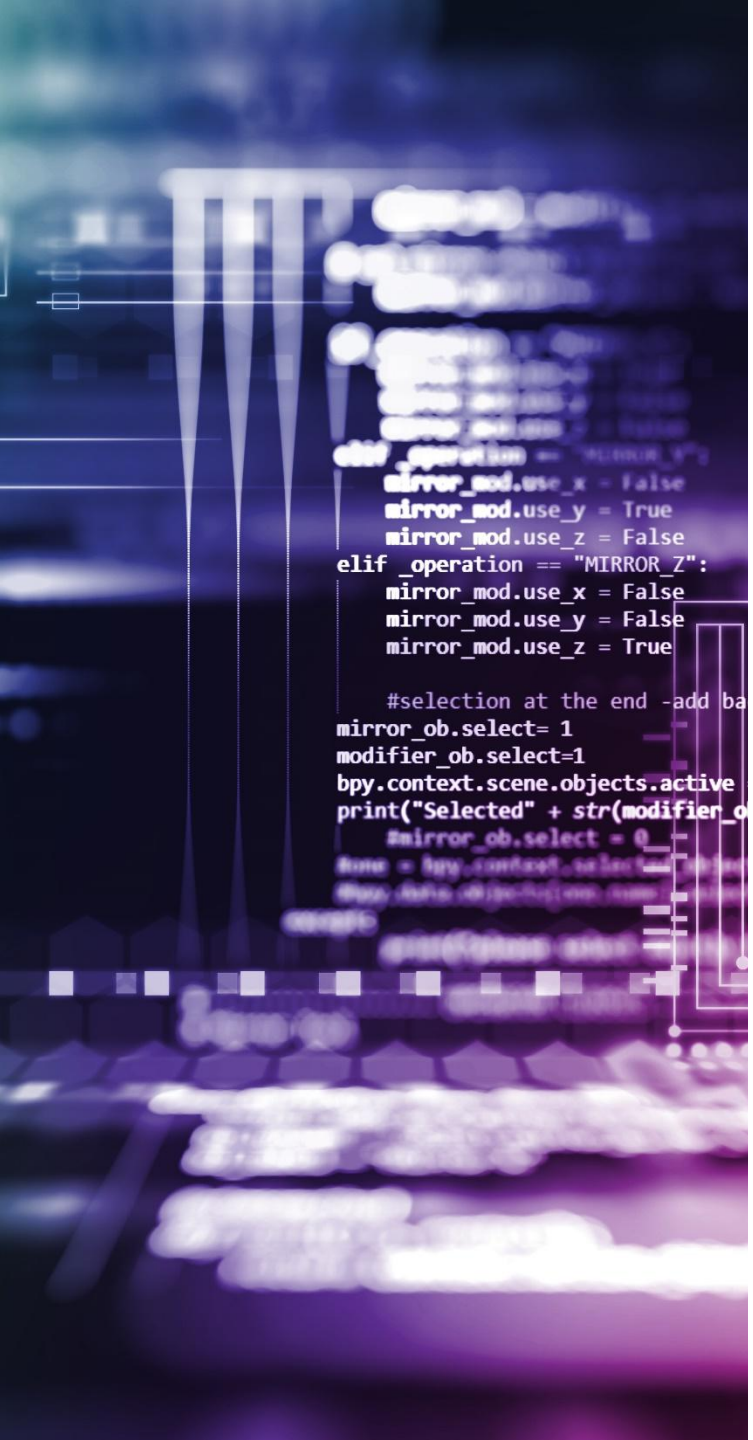
Retention(доступность)

```
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.ANNOTATION_TYPE)
public @interface Retention {
    /**
     * Returns the retention policy.
     * @return the retention policy
     */
    RetentionPolicy value();
}
```

- SOURCE - исходный код
- CLASS - исходный код и байткод
- RUNTIME - исходный код, байткод и runtime

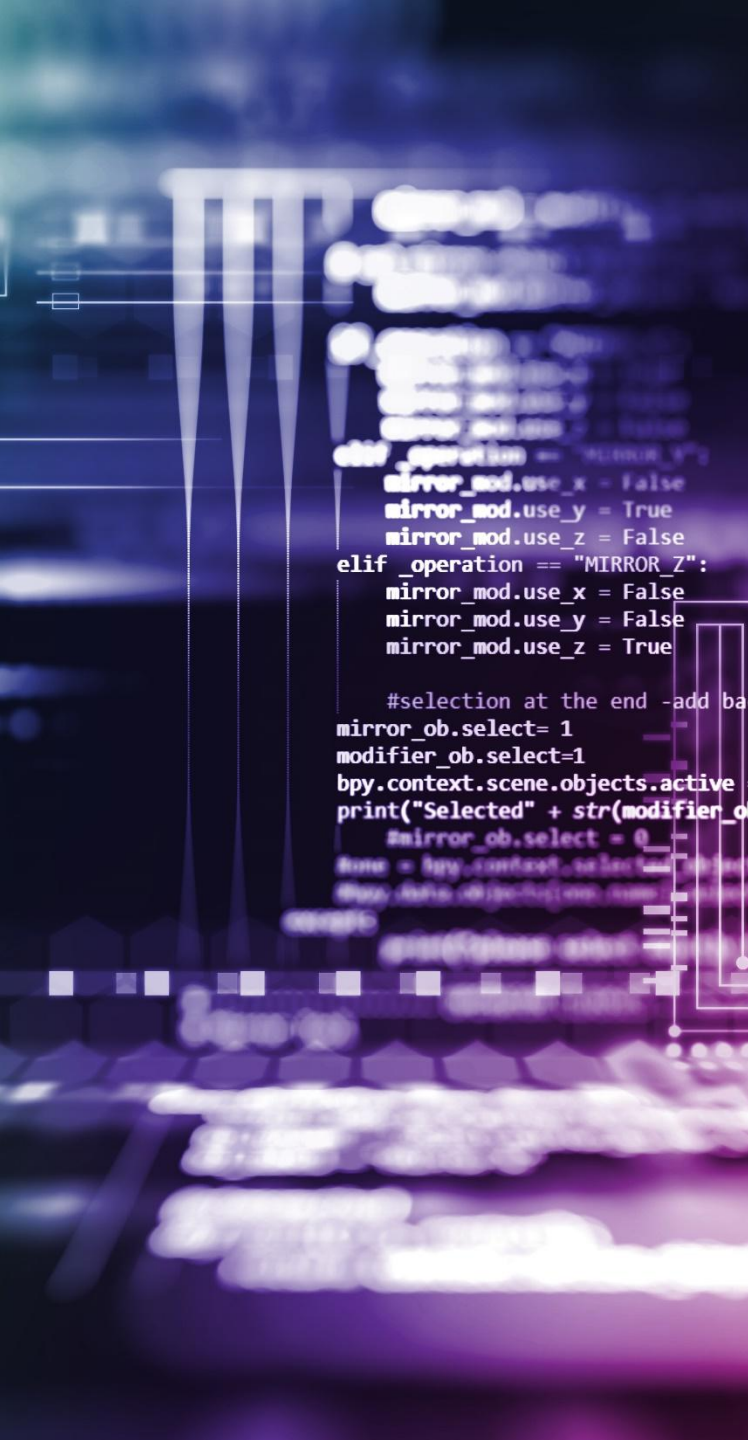
Target(цель применения)

```
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.ANNOTATION_TYPE)
public @interface Target {
    /**
     * Returns an array of the kinds of elements an annotation type
     * can be applied to.
     * @return an array of the kinds of elements an annotation type
     * can be applied to
     */
    ElementType[] value();
}
```



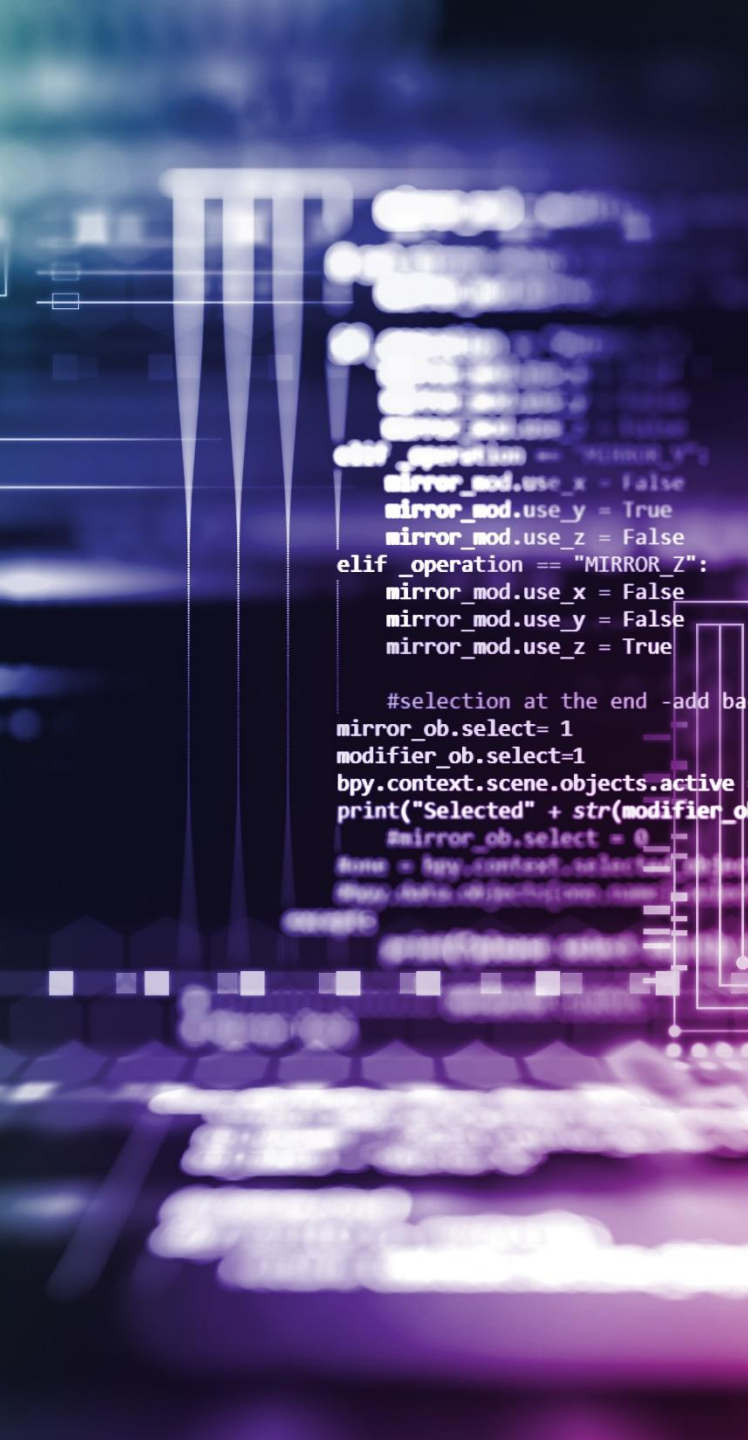
Target(цель применения)

- TYPE
- FIELD
- METHOD
- PARAMETER
- CONSTRUCTOR
- LOCAL_VARIABLE
- ANNOTATION_TYPE
- PACKAGE
- TYPE_PARAMETER*(java 8)*
- TYPE_USE*(java 8)*
- MODULE*(java 9)*
- RECORD_COMPONENT*(java 14 Preview Feature)*



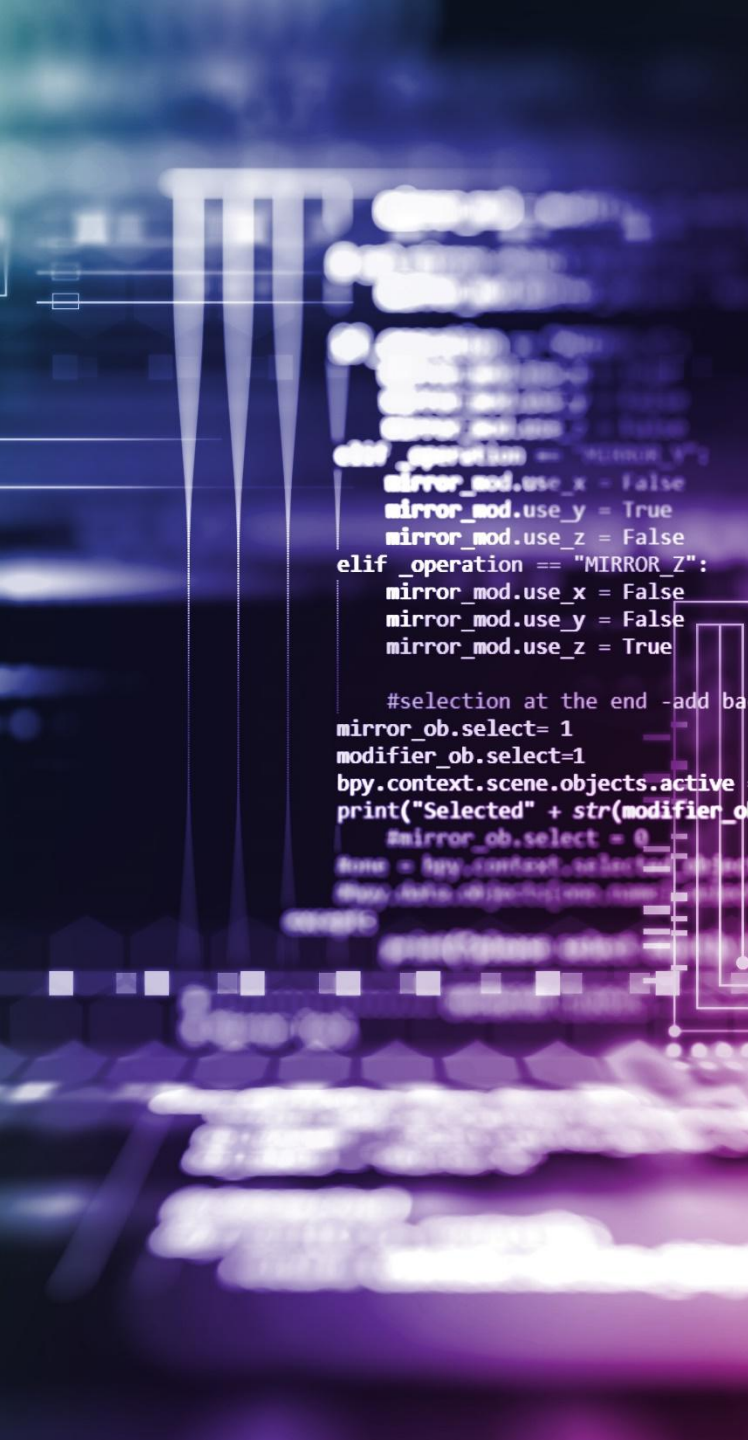
Пример

```
@Target(ElementType.TYPE) @interface MarkedType {}
@Target(ElementType.FIELD) @interface MarkedField {}
@Target(ElementType.METHOD) @interface MarkedMethod {}
@Target(ElementType.PARAMETER) @interface MarkedParameter {}
@Target(ElementType.CONSTRUCTOR) @interface MarkedConstructor {}
@Target(ElementType.LOCAL_VARIABLE) @interface MarkedLocalVariable {}
@Target(ElementType.ANNOTATION_TYPE) @interface MarkedAnnotationType {}
@Target(ElementType.PACKAGE) @interface MarkedPackage {}
@Target(ElementType.TYPE_PARAMETER) @interface MarkedTypeParameter {}
@Target(ElementType.TYPE_USE) @interface MarkedTypeUse {}
@Target(ElementType.MODULE) @interface MarkedModule {}
@Target(ElementType.RECORD_COMPONENT) @interface MarkedRecordComponent {}
```



Пример

```
@MarkedType
public class SuperAnnotationMarkedClass<@MarkedTypeParameter T> {
    @MarkedField
    private int count = 0;

    @MarkedConstructor
    public SuperAnnotationMarkedClass( @MarkedParameter int  count) {
        this.count = count;
    }

    @MarkedMethod
    public @MarkedTypeUse List<String> f(
            @MarkedParameter String param) {
        @MarkedLocalVariable String value = "" + count;
        return Collections.singletonList(value);
    }
}
```



Допустимые типы параметров

- примитивные типы
- строки
- Class, Class<?>
- перечисления
- аннотация(без циклических зависимостей)
- одномерные массивы
- значения известные в момент компиляции

Допустимые типы параметров

```
@Documented
@Inherited
@Retention(RUNTIME)
@Target({TYPE, FIELD, METHOD})
public @interface SuperAnnotation {
    int number() default 0;
    boolean is() default false;
    String name();
    Class<? extends CharSequence> strings() default String.class;
    E enumm() default E.B;
    double[] array() default {1.0, 2.6, 3.9};
    String object() default " " + 74;
}

enum E { A, B, C }
```

Аннотация с параметрами

```
@SuperAnnotation("ss")
class ClassWithSuperAnnotationA {}

@SuperAnnotation(value = "ss")
class ClassWithSuperAnnotationB {}

@SuperAnnotation(
        value = "ss",
        array = {5.5},
        strings = CharSequence.class,
        number = 9,
        object = "text"
)
class ClassWithSuperAnnotationC {}
```

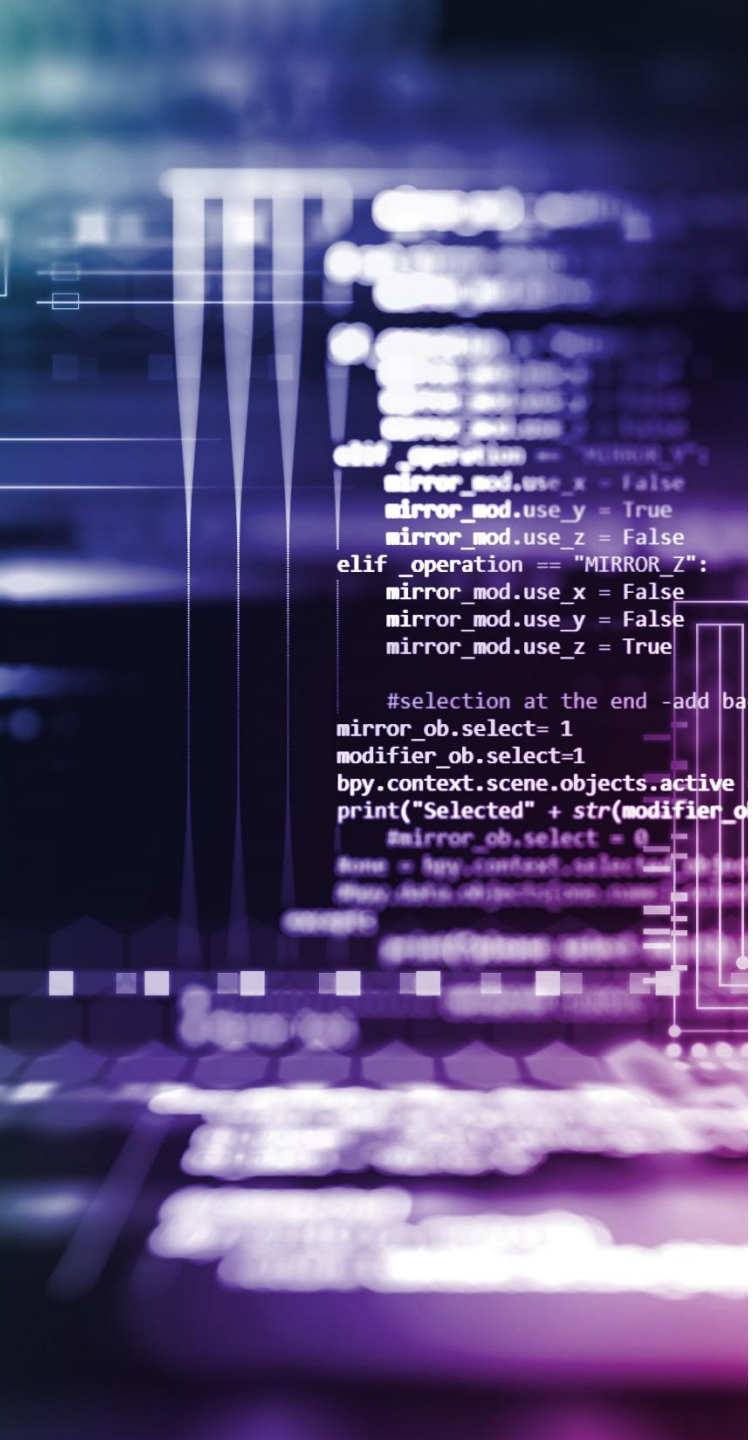
Data Binding

- Сериализация Java Object -> xml
- Десериализация xml -> Java Object
- Связывание полей класса и элементов xml

- JAXB
- FasterXML/jackson-dataformat-xml

```
class Book {
    private String author;
    private String title;
}
```

```
<Book>
    <author>Дорофеев Максим</author>
    <title>Джедайские техники</title>
</Book>
```



JAXB

- @XmlRootElement - корневой элемент
- @XmlElement - элемент
- @XmlAttribute - атрибут
- @XmlTransient - игнорирование
- @XmlElementWrapper - поместить в List/Map
- @XmlProperty - namespace

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.9.10</version>
</dependency>
<dependency>
  <groupId>com.fasterxml.jackson.module</groupId>
  <artifactId>jackson-module-jaxb-annotations</artifactId>
  <version>2.9.10</version>
</dependency>
<dependency>
  <groupId>jakarta.xml.bind</groupId>
  <artifactId>jakarta.xml.bind-api</artifactId>
  <version>2.3.3</version>
</dependency>
```

JAXB

```
@XmlRootElement
public class Book {
    @XmlElement
    private String author;
    @XmlElement
    private String title;
    @XmlAttribute
    private String number;
```

```
<Book number="#34">
    <author>Дорофеев Максим</author>
    <title>Джедайские техники</title>
</Book>
```

JAXB (почти, но нет)

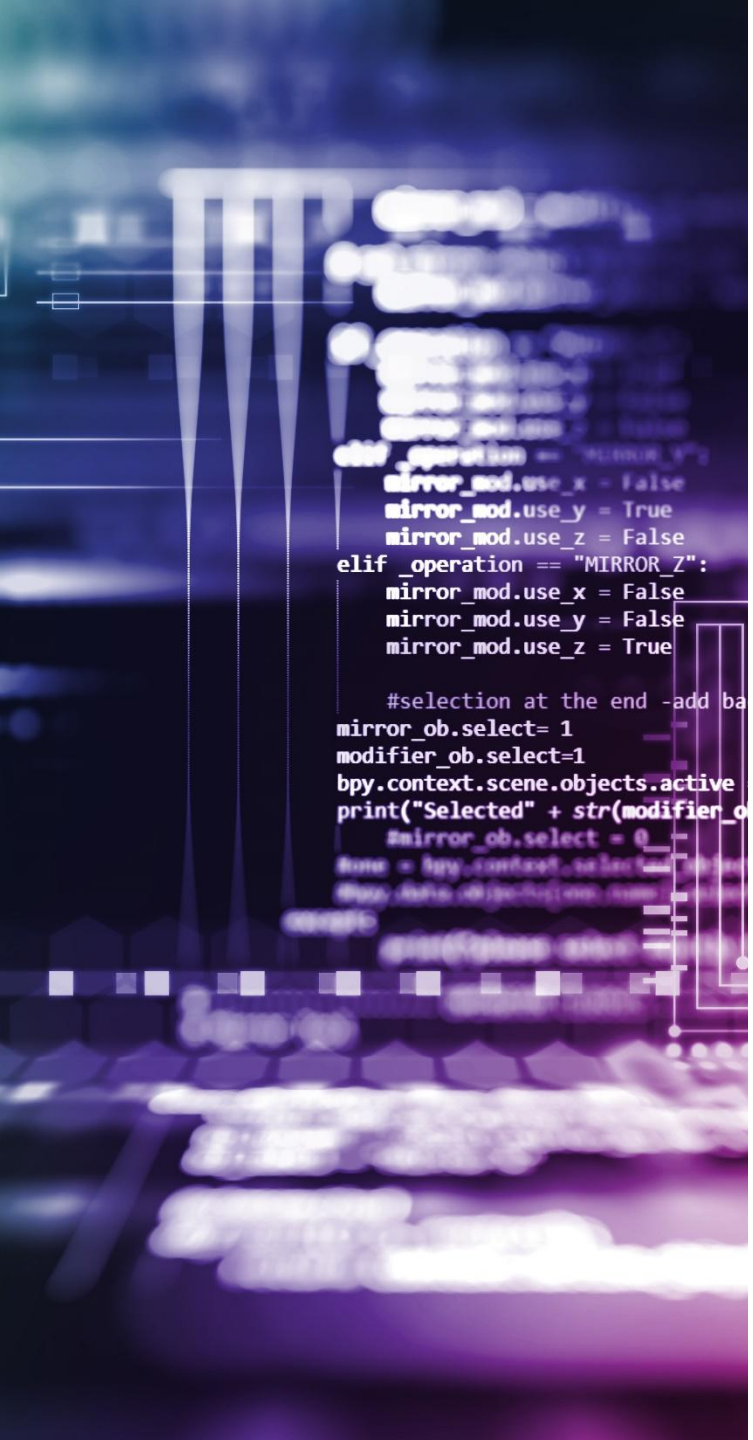
```java
public class JaxbConverter {
    public <T> T fromXml(String xml, Class<T> type) throws IOException {
        XmlMapper mapper = createXmlMapper();
        return mapper.readValue(xml, type);
    }

    public String toXml(Object obj) throws JsonProcessingException {
        XmlMapper mapper = createXmlMapper();
        return mapper.writeValueAsString(obj);
    }

    private XmlMapper createXmlMapper() {
        final XmlMapper mapper = new XmlMapper();
        JaxbAnnotationModule module = new JaxbAnnotationModule();
        mapper.registerModule(module);
        mapper.enable(SerializationFeature.INDENT_OUTPUT);
        return mapper;
    }
}
```

JAXB

```java
public class Author {
    @XmlAttribute(name = "ФИО", required = true)
    private String name;
    @XmlAttribute(name = "Возраст")
    private int age;
    @XmlElement(name = "Биография")
    private String biography;
```

```java
@XmlRootElement(name = "Книга")
public class Book {
    @XmlElementWrapper(name = "Авторы")
    @XmlElement(name = "Автор")
    private List<Author> authors;
    @XmlElement(name = "Название")
    private String title;
    @XmlAttribute(name = "Номер")
    private String number;
```

```xml
<Книга Номер="#0000">
    <Авторы>
        <Автор ФИО="Автор 1" Возраст="34">
            <Биография>text1 ...</Биография>
        </Автор>
        <Автор ФИО="Автор 2" Возраст="90">
            <Биография>text2 ...</Биография>
        </Автор>
        <Автор ФИО="Автор 3" Возраст="45">
            <Биография>text3 ...</Биография>
        </Автор>
    </Авторы>
    <Название>Джедайские техники</Название>
</Книга>
```



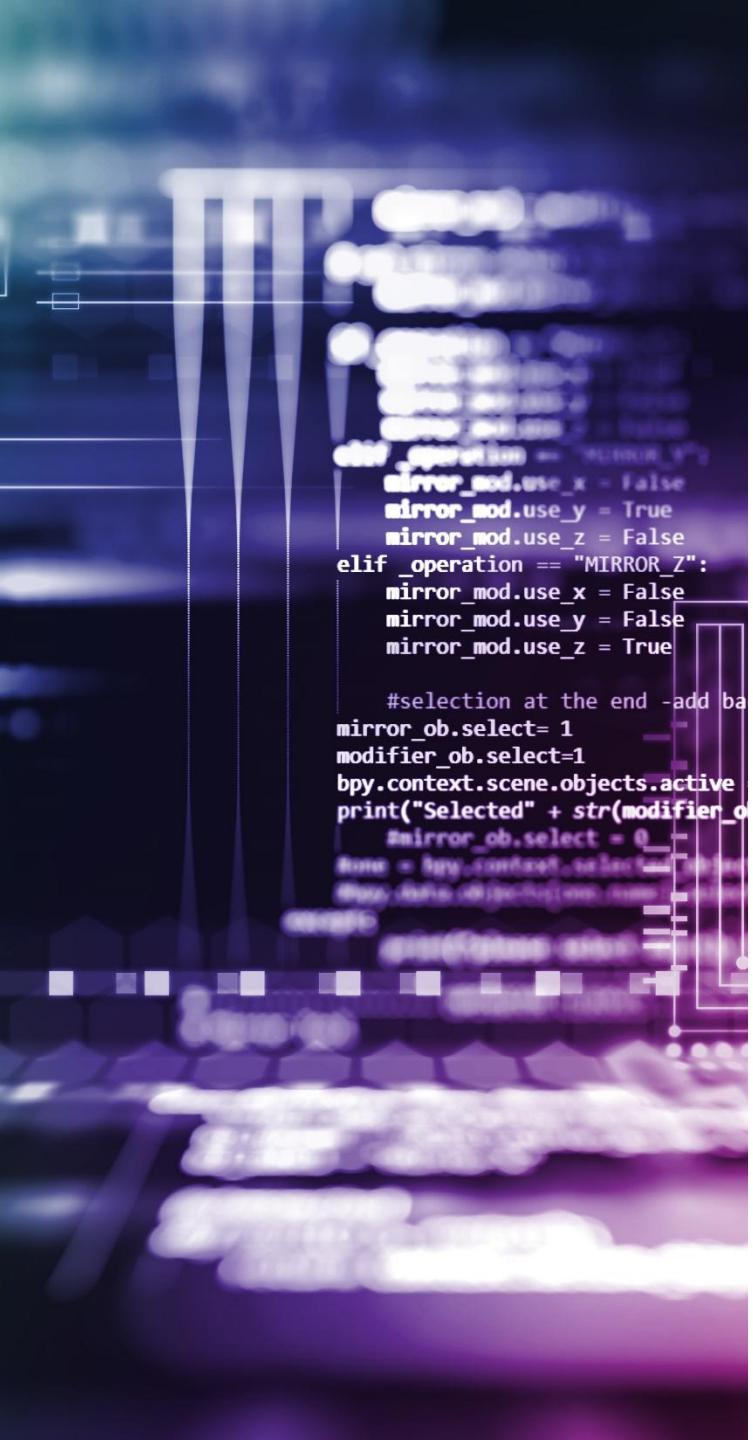
FasterXML/jackson-dataformat-xml

This projects contains Jackson extension component for reading and writing XML encoded data.

Further, the goal is to emulate how JAXB data-binding works with "Code-first" approach (that is, no support is added for "Schema-first" approach). Support for JAXB annotations is provided by JAXB annotation module; this module provides low-level abstractions (JsonParser, JsonGenerator, JsonFactory) as well as small number of higher level overrides needed to make data-binding work.

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.9.10</version>
</dependency>
```

https://github.com/FasterXML/jackson-dataformat-xml



Reflection - интроспекция

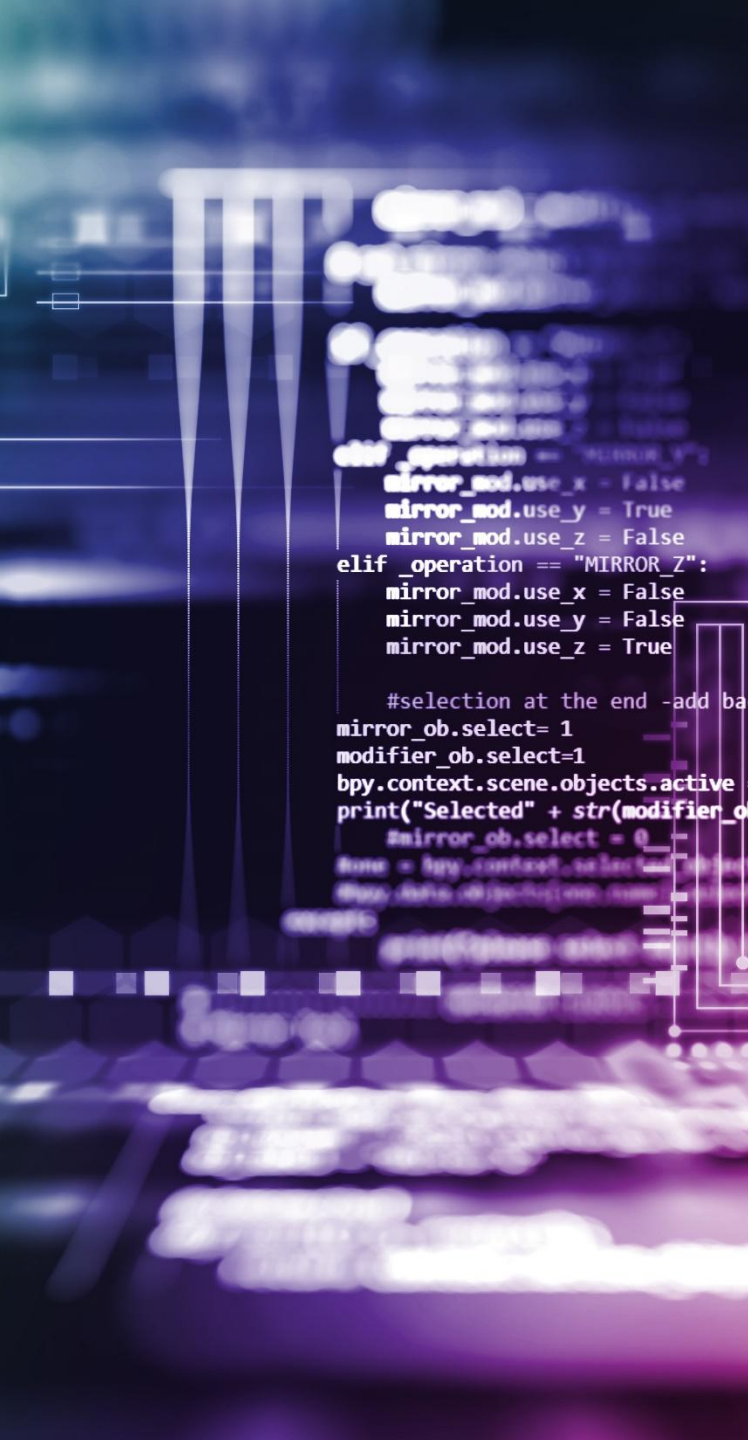
Reflection is a feature in the Java programming language. It allows an executing Java program to examine or "introspect" upon itself, and manipulate internal properties of the program. For example, it's possible for a Java class to obtain the names of all its members and display them.

Вызываем private метод

```
public class RunPrivateMethodTest {

    class A {
        private String f() {
            return "777";
        }
    }

    @Test
    public void test() throws Exception {
        A a = new A();
        RunPrivateMethod runner = new RunPrivateMethod();
        Assertions.assertEquals(a.f(), runner.run(a, methodName: "f"));
    }
}
```

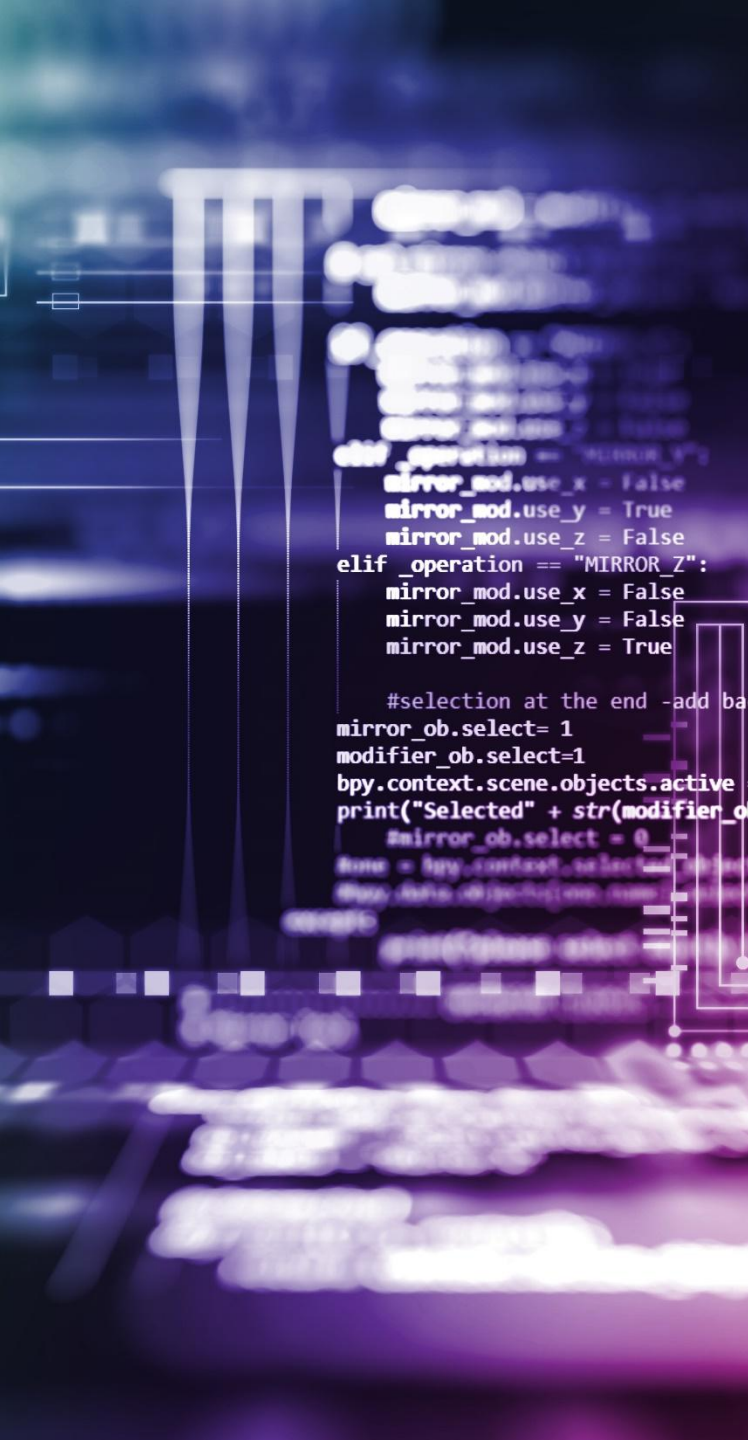
Вызываем private метод

```
public class RunPrivateMethod {
    public Object run(Object obj, String methodName) throws NoSuchMethodException,
            InvocationTargetException, IllegalAccessException {
        final Method method = obj.getClass().getDeclaredMethod(methodName);
        method.setAccessible(true);
        return method.invoke(obj);
    }
}
```



Применение

- Динамическая загрузка классов
- Доступ к аннотациям
- Dependency injection
- Сериализация/десериализация
- Анализ классов в библиотеках
- Доступ к приватным элементам класса(это плохо!)



Reflection для Generic

Можно узнать тип из исходников
Нельзя узнать тип стертый в runtime

Задача

Написать сериализатор в json используя собственные аннотации.

КРОК

ИНТЕГРИРУЕМ
БУДУЩЕЕ