Аннотации в Java

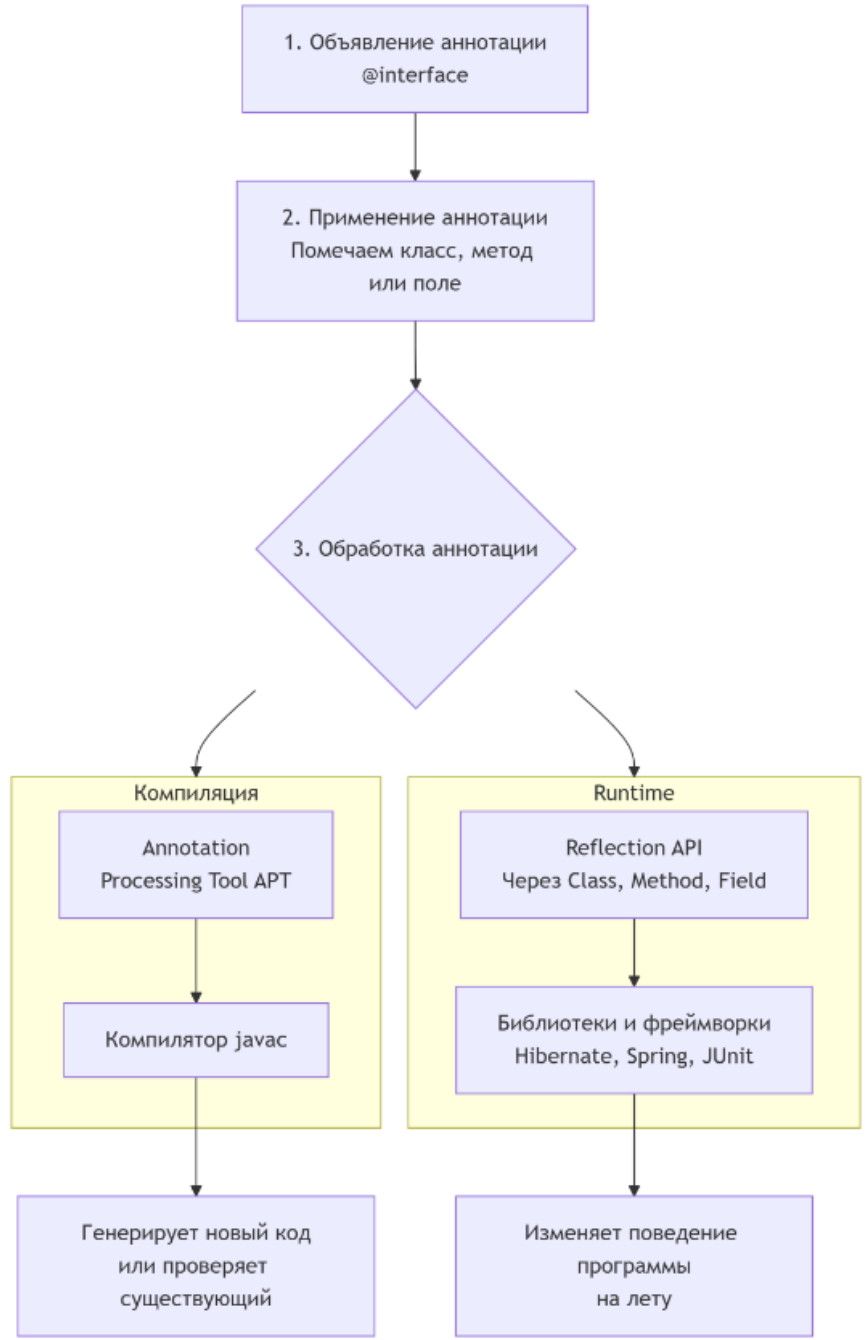
Докладчики: Коваль Егор, Мугтасимова Лиана

Аннотация — это форма метаданных, которая добавляется в исходный код Java. Она не меняет логику выполнения программы напрямую, но предоставляет информацию о программе для различных инструментов.

Виды аннотаций:

- Для компилятора
- Для времени выполнения
- Для других инструментов сборки и анализа

Простая аналогия: Это как стикер-напоминание, который вы клеите на код. Сам по себе стикер ничего не делает, но его читают вы (или ваши инструменты) и действуете соответственно.


1. Объявление аннотации
@interface
2. Применение аннотации
Помечаем класс, метод
или поле
3. Обработка аннотации
Компиляция
Annotation
Processing Tool APT
Компилятор javac
Генерирует новый код
или проверяет
существующий
Runtime
Reflection API
Через Class, Method, Field
Библиотеки и фреймворки
Hibernate, Spring, JUnit
Изменяет поведение
программы
на лету

Встроенные аннотации.

Java предоставляет набор встроенных аннотаций, самые важные из которых находятся в пакете `java.lang`.

`@Override` — Указывает, что метод переопределяет метод родительского класса.

Практическая польза: Компилятор проверяет, действительно ли такой метод существует в суперклассе. Помогает избежать опечаток.

```java
public class Employee {
    private String name;
    private double salary;

    public Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    // Базовая реализация расчета годового бонуса
    public double calculateAnnualBonus() {
        return salary * 0.1; // 10% от зарплаты
    }
}

public class Manager extends Employee {
    private double teamBonus;

    public Manager(String name, double salary, double teamBonus) {
        super(name, salary);
        this.teamBonus = teamBonus;
    }

    // Менеджер получает бонус по-другому: стандартный бонус + бонус за команду
    @Override
    public double calculateAnnualBonus() {
        return super.calculateAnnualBonus() + teamBonus;
    }
```

Встроенные аннотации.

@Deprecated — Пометка устаревшего кода, который не рекомендуется использовать.

Практическая польза: При использовании такого метода/класса компилятор и IDE покажут предупреждение.

Атрибуты since и forRemoval: Аннотация может содержать атрибуты since (указывающие на версию, в которой элемент был признан устаревшим) и forRemoval (логическое значение, указывающее, предназначен ли элемент для удаления в будущей версии).

```java
public class PaymentService {

    private final PaymentProcessor processor;

    public PaymentService(PaymentProcessor processor) {
        this.processor = processor;
    }

    /**
     * Старый метод оплаты через прямой ввод данных карты
     * Используйте processPayment(PaymentRequest)
     * из-за проблем безопасности. Новый метод поддерживает токенизацию.
     */
    @Deprecated(since = "2.1", forRemoval = true)
    public PaymentResult processCreditCard(String cardNumber, String expiryDate,
                                           String cvv, double amount) {
        CreditCard card = new CreditCard(cardNumber, expiryDate, cvv);
        return processor.processDirect(card, amount);
    }
    public PaymentResult processPayment(PaymentRequest request) {
        // Новая реализация - работа с токенами вместо реальных данных карты
        return processor.processSecure(request);
    }
}
```

Встроенные аннотации.

`@SuppressWarnings` — Подавляет предупреждения компилятора в указанном элементе.

Практическая польза: Используется, когда вы уверены в своей логике и хотите убрать "шум" от компилятора

Аннотация принимает в качестве параметра массив строк, где каждая строка представляет определенный тип предупреждения, подлежащего подавлению.

```java
public class LegacyIntegrationService {

    /**
     * Метод для работы со старой библиотекой, которая возвращает сырые типы
     * Мы знаем, что все элементы в списке - строки, но компилятор не может это проверить
     */
    @SuppressWarnings("rawtypes")
    public List<String> convertLegacyData(LegacyDataProvider provider) {
        // Старая библиотека возвращает сырой тип List
        List rawList = provider.getRawData();

        List<String> result = new ArrayList<>();
        for (Object item : rawList) {
            // Мы уверены, что все элементы - строки (документация библиотеки)
            result.add((String) item);
        }
        return result;
    }
```

Аннотации для управления жизненным циклом (Spring, JPA)

Аннотации широко используются в фреймворках для конфигурации приложений "на лету". Это основа таких технологий, как Spring и Jakarta Persistence (JPA).

Spring Framework: Управление зависимостями и бинами.

```java
public interface Validator {
    boolean validate(User user);
}

@Component
public class EmailValidator implements Validator {
    public boolean validate(User user) {
        return user.getEmail().contains("@");
    }
}

@Component
public class AgeValidator implements Validator {
    public boolean validate(User user) {
        return user.getAge() >= 18;
    }
}

@Service
public class UserRegistrationService {

    // Spring внедрит ВСЕ реализации Validator в список
    @Autowired
    private List<Validator> validators;

    public void registerUser(User user) {
        for (Validator validator : validators) {
            if (!validator.validate(user)) {
                throw new ValidationException("Validation failed");
            }
        }
        // Все валидации прошли
    }
}
```

Аннотации для управления жизненным циклом (Spring, JPA)

JPA (Hibernate): Отображение объектов на таблицы БД.

```java
@Entity //Сущность JPA
@Table(name = "customers") //Соответствует таблице customers в БД
public class Customer {
    @Id //Первичный ключ
    @GeneratedValue(strategy = GenerationType.IDENTITY) //Автоинкремент
    private Long id;

    @Column(name = "full_name", nullable = false) //Описание колонки
    private String name;

    @Column(unique = true, nullable = false)
    private String email;

    @OneToMany(mappedBy = "customer") //Связь один ко многим
    private List<Order> orders = new ArrayList<>();
}

@Entity
@Table(name = "orders")
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "customer_id") //Присоединение колонки из другой таблицы
    private Customer customer;

    @OneToMany(mappedBy = "order", cascade = CascadeType.ALL)
    private List<OrderItem> items = new ArrayList<>();

    @Enumerated(EnumType.STRING) //Работа с определённым набором значений
    private OrderStatus status;

    private LocalDateTime createdAt;
}
```

Создание собственной аннотации

Шаг 1: Объявление аннотации
Ключевое слово `@interface`.
Указываем, где (`@Target`) и когда (`@Retention`) будет работать наша аннотация.

```java
import java.lang.annotation.*;

// Определение нашей аннотации
@Retention(RetentionPolicy.RUNTIME) // Доступна во время выполнения
@Target(ElementType.METHOD) // Можно применять только к методам
public @interface MyAnnotation {
    String value() default "default value";
    String description() default "";
    int priority() default 1;
}
```

Создание собственной аннотации

Шаг 2: Обработка аннотации
Без обработчика аннотация — просто метка. Обработка происходит через Reflection API.

```java
import java.lang.reflect.Method;

public class AnnotationProcessor {

    public static void processAnnotations(Object obj) {
        Class<?> clazz = obj.getClass();
        // getDeclaredMethods() возвращает массив всех методов, объявленных в классе
        Method[] methods = clazz.getDeclaredMethods();
        for (Method method : methods) {
            // isAnnotationPresent() возвращает true, если аннотация есть на методе
            if (method.isAnnotationPresent(MyAnnotation.class)) {
                // getAnnotation() возвращает объект аннотации, из которого можно получить значения
                MyAnnotation annotation = method.getAnnotation(MyAnnotation.class);
                System.out.println("Найден метод с аннотацией: " + method.getName());
                System.out.println("Значение: " + annotation.value());
                System.out.println("Описание: " + annotation.description());
                System.out.println("Приоритет: " + annotation.priority());
                System.out.println("---");
                try {
                    // invoke() вызывает метод на указанном объекте
                    // obj - объект, на котором вызывается метод
                    method.invoke(obj);
                } catch (Exception e) {
                    // Обрабатываем возможные исключения:
                    e.printStackTrace();
                }
            }
        }
    }

    public static void main(String[] args) {
        // Создаем экземпляр класса, который содержит аннотированные методы
        MyClass myObject = new MyClass();

        // Запускаем обработку аннотаций для созданного объекта
        processAnnotations(myObject);
    }
```

Создание собственной аннотации

Шаг 3: Использование аннотации

```java
public class MyClass {

    @MyAnnotation(
        value = "important method",
        description = "Этот метод выполняет важную операцию",
        priority = 5
    )
    public void importantMethod() {
        System.out.println("Выполняется важный метод");
    }

    @MyAnnotation("обычный метод")
    public void regularMethod() {
        System.out.println("Выполняется обычный метод");
    }

    @MyAnnotation // Используем значения по умолчанию
    public void defaultMethod() {
        System.out.println("Выполняется метод с аннотацией по умолчанию");
    }
}
```

Итоги:

Аннотации — это мост между кодом и метаданными

- **Упрощают конфигурацию** — Позволяют описывать поведение прямо в коде, делая его более читаемым и понятным
- **Автоматизируют рутину** — Избавляют от шаблонного кода, делегируя работу компилятору и фреймворкам
- **Делают код безопаснее** — Помогают избежать ошибок на этапе компиляции (`@Override`, `@Deprecated`)
- **Аннотации сами по себе не выполняют код** — они лишь предоставляют информацию
- **Реальную работу делают обработчики аннотаций** (компилятор, фреймворки)
- **Правильное использование аннотаций делает код чище и профессиональнее**