Лекция 9.2: Продвинутые возможности / Аннотации.

1. Используем аннотации
….1.1 Ограничение: тип атрибута
….1.2 Ограничение: значения атрибутов – константы времени компиляции/загрузки JVM
2. Мета-аннотации: аннотации описывающие аннотации
….2.1 @Target, @Retention, @Documented, @Inherited, @Repeatable
3. Модифицируем поведение компилятора
….3.1 @Deprecated, @Override, @SafeVarargs, @SuppressWarnings, @FunctionalInterface
4. Вычитываем аннотации с помощью Reflection API
5. Расширенные возможности и нововведения в Java 8
….5.1 Annotation Processing Tool (APT)
….5.2 JSR 308: Type Annotations and Pluggable Type Systems
….5.3 Расширяем проверку типов с помощью Checker Framework: @NotNull, @GuardedBy, …
Тесты по всей лекции
Видео
Литература

Приветствуем, pafnat

Настройки профиля
Выйти

Ваша успеваемость

Java Core	14.34%
1. Основы Java	88.07%
2. Базовые алгоритмы	24.13%
3. Исключения	31.21%
4. Ввод/вывод	0%
5. Многопоточность	0%
6. Коллекции	0%
7. ООП: Синтаксис	0%
8. ООП: Шаблоны	0%
9. Продвинутые возможности	0%
10. Java 8	0%

1. Используем аннотации

Поехали!
Учебный пример: снабдить классы пользователя мета-информацией о “версии класса”.

Итерация #1:
Просто ставим @ перед interface.

```
public @interface Version {}
```

Итерация #2:
У аннотаций могут быть атрибуты.

```
public @interface Version {
    public int version();
}
```

Заполнятся при использовании

```
@Version(version = 42)
public class MyClass {}
```

Аннотация выше полностью эквивалентна следующей (без public). В этом аннотации схожи с интерфейсам: отсутствие модификатора области видимости автоматически означает public (а не package private как у классов).

```
public @interface Version {
    int version();
}
```

С protected и private – не компилируется

```
public @interface Version {
    protected int version();
}

>> COMPILATION ERROR: Modifier 'protected' not allowed here
```

Далее я буду использовать вариант без модификатора public

Итерация #3:
Если объявить атрибут с именем value, то его можно опускать при использовании

```
public @interface Version {
    public int value();
}
```

```
@Version(42)
public class MyClass {}
```

Хотя можно и по старинке

```
@Version(value = 42)
public class MyClass {}
```

Итерация #4:
Для атрибута можно объявить значения по умолчанию

```
public @interface Version {
    int value();
    String author() default "UNKNOWN";
}
```

Теперь у нас два варианта использования. Так

```
@Version(42)
public class MyClass {}
```

Или вот так

```
@Version(value = 42, author = "Jim Smith")
public class MyClass {}
```

Но не вот так (слушай, обидно, да)

```
@Version(42, author = "Jim Smith")
public class MyClass {}

>> COMPILATION ERROR: Annotation attribute must be of the form 'name=value'
```

Итерация #5:
Атрибуты могут иметь тип массива

```
public @interface Author {
    String[] value() default {};
}
```

```
@Author({"Anna", "Mike", "Sara"})
public class MyClass {}
```

Но только одномерного

```
public @interface Author2D {
    String[][] value() default {};
}

>> COMPILATION ERROR: Invalid type of annotation member
```

Итерация #6:
Возможен забавный трюк: аннотация – атрибут аннотации

```
public @interface Version {
    int value();
    String author() default "UNKNOWN";
}
```

```
public @interface History {
    Version[] value() default {};
}
```

Применяется вот так

```
@History({
        @Version(1),
        @Version(value = 2, author = "Jim Smith")
})
public class MyClass {}
```

У аннотаций много ограничений. Перечислим некоторые из них.

1.1 Ограничение: тип атрибута

1. Атрибуты могут иметь только следующие типы

- примитивы
- String
- Class или “any parameterized invocation of Class”
- enum
- annotation
- массив элементов любого из вышеперечисленных типов

Последний пункт надо понимать как то, что допустимы только одномерные массивы.

Ну что же, давайте действовать в рамках ограничений

Итерация #7:
В качестве типа атрибута нельзя использовать “обычные” классы Java (за исключением java.lang.String и java.lang.Class), скажем java.util.Date

```
import java.util.Date;

public @interface Version {
    Date date();
}

>> COMPILATION ERROR: Invalid type for annotation member
```

Но можно эмулировать записи/структуры на аннотациях

```
public @interface Date {
    int day();
    int month();
    int year();
}
```

```
public @interface Version {
    Date date();
}
```

```
@Date(year = 2001, month = 1, day = 1)
public class MyClass {}
```

Итерация #8:
Атрибутом аннотации может быть enum. Из приятного, его можно объявить в объявлении аннотации (как и в объявлении интерфейса тут может быть объявление enum, class, interface, annotation)

```
public @interface Colored {
    public enum Color {RED, GREEN, BLUE}
    Color value();
}
```

```
import static net.golovach.Colored.Color.RED;

@Colored(RED)
public class MyClass {}
```

Итерация #9:
Атрибутом аннотации может быть классовый литерал.
Аннотация версии включает ссылку на предыдущую версию класса.

```
public @interface Version {
    int value();
    Class<?> previous() default Void.class;
}
```

Первая версия класса

```
@Version(1)
public class ClassVer1 {}
```

Вторая версия со ссылкой на первую

```
@Version(value = 2, previous = ClassVer1.class)
public class ClassVer2 {}
```

// Да, я знаю, что нормальные люди не включают версию класса в имя класса. Но знаете как нудно придумывать примеры согласованные с реальной практикой?

Итерация #10:
Менее тривиальный пример с классовым литералом, где я не удержался и добавил generic-ов.
Интерфейс “сериализатора” – того, кто может записать экземпляр T в байтовый поток вывода

```
import java.io.IOException;
import java.io.OutputStream;

public interface Serializer<T> {
    void toStream(T obj, OutputStream out) throws IOException;
}
```

Конкретный “сериализатор” для класса MyClass

```
import java.io.IOException;
import java.io.OutputStream;

public class MySerializer implements Serializer<MyClass> {
    @Override
    public void toStream(MyClass obj, OutputStream out) throws IOException {
        throw new UnsupportedOperationException();
    }
}
```

Аннотация, при помощи которой мы “приклеиваем сериализатор” к конкретному классу

```
public @interface SerializedBy {
    Class<? extends Serializer> value();
}
```

Ну и сам класс MyClass отмеченный, как сериализуемый “своим сериализатором” MySerializer

```
@SerializedBy(MySerializer.class)
public class MyClass {}
```

Итерация #11:
Сложный пример

```
public enum JobTitle {
    JUNIOR, MIDDLE, SENIOR, LEAD,
    UNKNOWN
}
```

```
public @interface Author {
    String value();
    JobTitle title() default JobTitle.UNKNOWN;
}
```

```
public @interface Date {
    int day();
    int month();
    int year();
}
```

```
public @interface Version {
    int version();
    Date date();
    Author[] authors() default {};
    Class<?> previous() default Void.class;
}
```

Ну и наконец использование аннотации

```
import static net.golovach.JobTitle.*;

@History({
        @Version(
                version = 1,
                date = @Date(year = 2001, month = 1, day = 1)),
        @Version(
                version = 2,
                date = @Date(year = 2002, month = 2, day = 2),
                authors = {@Author(value = "Jim Smith", title = JUNIOR)},
                previous = MyClassVer1.class),
        @Version(
                version = 3,
                date = @Date(year = 2003, month = 3, day = 3),
                authors = {
                        @Author(value = "Jim Smith", title = MIDDLE),
                        @Author(value = "Anna Lea")},
                previous = MyClassVer2.class)
})
public class MyClassVer3 {}
```

1.2 Ограничение: значения атрибутов – константы времени компиляции/загрузки JVM

Должна быть возможность вычислить значения атрибутов аннотаций в момент компиляции или загрузки класса в JVM.

```
public @interface SomeAnnotation {
    int count();
    String name();
}
```

Пример

```
@SomeAnnotation(
        count = 1 + 2,
        name = MyClass.STR + "Hello"
)
public class MyClass {
    public static final String STR = "ABC";
}
```

Еще пример

```
@SomeAnnotation(
        count = (int) Math.PI,
        name = "" + Math.PI
)
public class MyClass {}
```

А вот вызовы методов – это уже runtime, это уже запрещено

```
@SomeAnnotation(
        count = (int) Math.sin(1),
        name = "Hello!".toUpperCase()
)
public class MyClass {}

>> COMPILATION ERROR: Attribute value must be constant
```

2. Мета-аннотации: аннотации описывающие аннотации

РАЗДЕЛ НАХОДИТСЯ В РАЗРАБОТКЕ

2.1 @Target, @Retention, @Documented, @Inherited, @Repeatable

РАЗДЕЛ НАХОДИТСЯ В РАЗРАБОТКЕ

3. Модифицируем поведение компилятора

РАЗДЕЛ НАХОДИТСЯ В РАЗРАБОТКЕ

3.1 @Deprecated, @Override, @SafeVarargs, @SuppressWarnings, @FunctionalInterface

РАЗДЕЛ НАХОДИТСЯ В РАЗРАБОТКЕ

4. Вычитываем аннотации с помощью Reflection API

РАЗДЕЛ НАХОДИТСЯ В РАЗРАБОТКЕ

5. Расширенные возможности и нововведения в Java 8

РАЗДЕЛ НАХОДИТСЯ В РАЗРАБОТКЕ

5.1 Annotation Processing Tool (APT)

РАЗДЕЛ НАХОДИТСЯ В РАЗРАБОТКЕ

5.2 JSR 308: Type Annotations and Pluggable Type Systems

РАЗДЕЛ НАХОДИТСЯ В РАЗРАБОТКЕ

5.3 Расширяем проверку типов с помощью Checker Framework: @NotNull, @GuardedBy, …

РАЗДЕЛ НАХОДИТСЯ В РАЗРАБОТКЕ

Тесты по всей лекции

Тест, состоящий из случайных вопросов тестов этой лекции

Some issues occured with quiz randomizer :(

Видео

Сложное НЕ мое видео о Type Annotations = нововведениях для аннотаций в Java 8
Александр Ильин, Type annotations in Java 8. И почему это хорошо

Литература

Шилдт. “Java. Полное руководство”. Издание 8
….Глава 12. Перечисления, автоупаковка и аннотации, стр 305-317

Хорстманн, Корнелл. “Java 2. Том II. Тонкости программирования”. Издание 7
….Глава 13. Аннотации, стр 1111-1147

oracle.com/JavaTutorial: “Annotations”

The Checker Framework

http://www.infoq.com/articles/Type-Annotations-in-Java-8