Аннотации

- Аннотации - формализованный механизм включения в код информации, которую можно легко и удобно использовать в программе
- Аннотация для представления информации, которая необходима для полного описания программы
- Можно держать метаданные в исходном коде Позволяют сохранить доп инф о программе в формате, проверяемом компилятором избавляют от хлопот с написанием шаблонного кода
- Аннотации позволяют сохранит доп информацию о программе в формате, который может быть проверен
- Могут использоваться для генерирования файлов описаний и даже опр новых классов
- Содержит три аннотации общего назначения
- @Override - определение метода для переопр метода базового класса, @Deprecated - заставляет компилятор выдать предупр при использовании этого элемента, @SuppresWarnings - подавляет неподходящее предупреждение компилятора
- Использование и расширение инструментария и Api в сочетаниями с внешними библиотеками обработки байт кода - возможности анализа и обработки как исходного кода программы, так и байт-кода

- Базовый синтаксис
- @Test - для тестов, сама по себе ничего не делает
- Определение аннотаций
- Определение аннотаций во многом похожи на определения интерфейсов. Компилируются в файлы классов как и интерфейсы Java
- Содержат элементы, задающие некоторые значения
- Элементы похожи на методы интерфейсов
- Аннотация без элементов - маркерная аннотация
- Значения элементов аннотаций выражаются парами имя-значение @UseCase
- Мета-аннотации - предназначены для пометки аннотаций
- @Target – Constructor – объявление конструктора, Field - объявление поля, Local_Variable - объявление локальной переменной, Method – объявления метода. Package - объявление пакета, Parametr - объявление параметра, TYPE - объявления класса\перечисления
- @Retention Source – аннотации игнорируются компилятором, class - доступны в файле класса, игнориются виртуальной машиной, Runtime - сохраняются виртуальной машиной на стадии выполнения
- @Document - включена в Javadocs

- Написание обработчиков аннотаций
- Важная часть процесса работы с аннотациями - создание и использование обработчиков аннотаций

- средства чтения анотаций

Допустимые элементы аннотаций

Примитивы, Строки, класс, перечисление, аннотации, массивы

Ограничения значений по умолчанию

Ни один элемент не может иметь неопределенного значения. Либо элемент должен иметь значение по умолчанию, либо значения должны предоставляться классом, который использует аннотацию

Генерирование внешних файлов

Аннотации полезны при работе с фреймворками,

@Target - устанваливает ограничение, которое говорит о том,

@DBTable - элемент name(), чтобы аннотация могла предоставить имя таблицы бд, которая будет создана обработчиком

@Constrains - позволяет обработчику получить методанные, относящиеся к таблице базы данных

@interface - определяют типы SQL

- Чтобы создать вложенную аннотацию @Constrains у которой по умолчанию ограничение уникальности истинно, можно определить элемент следующим образом
- @DBTable - значение MEMBER< ,eltn bcgjkmpjdfyj d rfxtcndt bvtyb nf,kbws
- Аннотация @SQLString - получают значения
- @SQLString(30)
- @Constraint - долен быть установлен тип primaryKey
- @TableColumn - элемент-перечисление необходимость в объявлении @interface для каждого типа SQL,
- @interface - ключевое слово extends не может использоваться с @interface
- Аннотации не поддерживают наследование
- Реализация обработчика

Каждый класс проверяется на наличие аннотации @DBTable – getAnnotation SQLInteger или @SQLString

Большая часть фреймвореов использует аннотации @Constraint передается методу getConstraints

Apt - упрощает обработку аннотаций

При запуске apt - либо класс фобрика класс-фабрика При создании обработчика apt нельзя

- Паттерн "Посетитель" с apt
- Перебирает структуру данных или коллекцию объектов, выполняя операцию с каждым элементом. Операции отделяются от самих объектов -> новые операции могут добавляется без добавления методов в определения классов
- Класс Visitor содержит метод для обработки каждого типа объявлений

- Я прочитала главу по аннотациям тестирования, но удалила записи в блокноте

- Тестирование @Unit
- @Test - для методов тест
- Позволяет укоротить написанные код
- Тесты можно написать с помощью Junit, но там нет гибкости имен и ассерт
- Assert @Unit позволяет не прерывать процесс тестирования в случае не выполнения теста
- Объекты создаются конструктором по умлочанию. если нет конструктора по умолчанию -> @TestObjectCreate
- @TestObjectCreate - статический, возвращает объект тестируемого типа
- @TestProperty - может использоваться для пометки полей, используемые только для модульного тестирования
- + для пометки методов, используемые при тестировании, но сами тестами не являются
- @TestObjectCleanup - используется для статических методов, работающих после завершения работы с тестовым
- объектом(исп при необходимости завершающих действий)
- @TestObjectCreate - для создания объекта необходимого для тестирования \

- Использование @Unit с обобщениями
- Обобщенное тестирование невозможно - при тестировании должен быть конкретный параметр-тип\набор
- параметров. решение - создание тестового класса с помощью наследования от обобщенного класса
- @Unit упрощает работу по созданию тестов

- Реализация @Unit
- @Target, @Retention, @interface TestObjectCreate
- Экземпляр AtUnit может передаваться конструктору ProcessFiles
- автоматически находит тестируемые классы и методы, механизм семейств не нужен
- process()
- Если методы @Test будут найдены, программа выводит имя класса, после чего выполняется каждый тест
- @TestObjectCreate - используется после вызова метода createTestObject()
- Удаление тестового кода
- Можно оставить, но желательно удалить
- Для перебора файлов используется класс ProcessFiles
- удаление полей @TestProperty - болеее сложная задача, чем удаление методов

- Аннотации - механизм добавление методанных, который не загромождает код и не затрудняет его
- чтение.
- Можно создавать аннотации самим, только что сгенерированные файлы можно найти автоматически
- откомплировать с помощью аннотации Apt