Annotation

* JEE5부터 새롭게 추가된 문법 요소
* 기존에는 선언적인 프로그래밍 방식을 사용하였다. 선언적이란 프로그램의 전체 및 각 레이어별 구성과 설정값들을 외부의 XML 설정파일에 명시하는 방식을 의미한다.
* 비즈니스 로직에는 영향을 주지 않지만 해당 타겟의 연결방법이나 소스코드의 구조를 변경할 수 있음

메타데이터

* 데이터에 관한 구조화된 데이터(Structured Data About Data)
* 웹 자료나 다른 것들에 관해 기계가 이해할 수 있는 정보
* 필수적인 데이터 이외의 부차적인 내용이나 특성

AOP(Aspect Oriented Programming)

* 기능을 핵심 비즈니스 로직과 공통 모듈로 구분하고, 핵심 로직에 영향을 미치지 않고 사이사이에 공통 모듈을 효과적으로 잘 끼워 넣도록 하는 개발 방법
* 공통 모듈: 보안 인증, 로깅… etc
* 메소드 성능검사
* 트랜잭션 처리
* 예외 반환
* 아키텍쳐 검증
* 멀티쓰레드 Safety 관리

UML(Unified Modeling Language)

정의

* 프로그램 설계를 표현하기 위해 사용하는 주로 그림으로 된 표기법을 의미한다.
* 객체지향 언어와 밀접한 관련이 있기에 객체지향 모델링 언어라고도 불린다.
* 소프트웨어 시스템, 업무 모델링, 시스템의 산출물을 규정하고 시각화하며 문서화하는 언어

종류

* 클래스 다이어그램(Class Diagram)  
  클래스 명세와 클래스 간의 관계를 표현
* 복합 구조 다이어그램(Composite Structure Diagram)  
  전체-부분 구조를 가진 클래스를 실행할 때의 구조를 표현
* 컴포넌트 다이어그램(Component Diagram)  
  파일과 데이터베이스, 프로게스와 스레드 등의 소프트웨어 구조를 표현
* 디플로이먼트 다이어그램(Deployment Diagram)  
  하드웨어와 네트워크 등 시스템의 물리 구조를 표현
* 객체 다이어그램(Object Diagram)  
  인스턴스 간의 연관 관계를 표현
* 패키지 다이어그램(Package Diagram)  
  패키지 간의 연간 관계를 표현
* 액티비티 다이어그램(Activity Diagram)  
  일련의 처리에 있어 제어의 흐름을 표현
* 시퀀스 다이어그램(Sequence Diagram)  
  인스턴스 간의 상호 작용을 시계열로 표현
* 커뮤니케이션 다이어그램(Communication Diagram)  
  인스턴스간의 상호 작용을 중심으로 표현
* 인터액션 오버뷰 다이어그램(Interaction Overview Diagram)  
  조건에 따라 다르게 동작하는 시퀀스 다이어그램을 액티비티 다이어그램 안에 포함하여 표현
* 타이밍 다이어그램(Timing Diagram)  
  인스턴스 간의 상태 전이와 상호 작용을 시간 제약으로 표현
* 유스케이스 다이어그램(UseCase Diagram)  
  시스템이 제공하는 기능과 이용자의 관계를 표현

유스케이스 다이어그램

정의

* 행위자 중심의 시스템 구상
* 소프트웨어 시스템의 기능적요구사항에 대한 베이스라인
* 사용자의 시각에서 소프트웨어 시스템의 범위와 기능을 정의한 모델
* 행위자가 어떤 기능을 사용할 수 있는지 보여줌

1) 행위자(Actor)

- 시스템이 외부에 존재하면서 시스템과 교류 혹은 상호작용 하는 것

(1) 사용자 액터

- 기능을 요구하는 대상이나 시스템의 수행결과를 통보 받는 사용자

(2) 시스템 액터

- 사용자 액터가 사용한 유스케이스를 처리해주는 외부의 시스템

시퀀스 다이어그램

목적

* 어떤 결과를 만들어내는 이벤트 시퀀스를 정의하는 것

방식

* 수직적 방식(Top Down)  
  메시지/호출이 발생한 시간 순서
* 수평적 방식  
  왼쪽에서 오른쪽으로 메시지가 보내진 객체 인스턴스를 보여준다.

LifeLines

* LifeLine 표기법 앨리먼트는 다이어그램 상단에 놓인다. LifeLine은 모델링 되는 시퀀스에 개입된 역할 또는 개개 인스턴스를 나타낸다.
* 해당 클래스에서 만들어진 객체의 존재 ㅈ선

추상화(Abstraction)란

* 객체지향 프로그래밍(OOP) 추상화
* 스프링 추상화

SimpleAsyncTaskExecutor

* 어떤 스레드도 재사용하지 않고 호출마다 새로운 스레드를 시작한다.

SyncTaskExecutor

* 호출을 비동기적으로 실행하지 않고, 각 호출이 호출 스레드에 추가된다.

ConcurrentTaskExecutor

* Java 5 java.util.concurrent.Executor의 래퍼. ThreadPoolTaskExecutor가 원하는 만큼 안정적이지 않다면 대신 사용 가능

SimpleThreadPoolTaskExecutor

* Quartz SimpleThreadPool의 하위 클래스로 스프링의 생명주기 콜백을 받는다. Quartz와 Quartz가 아닌 컴포넌트간에 공유해야 하는 스레드 풀이 있는 경우에 보통 사용

TimerTaskExecutor

* 지원 구현체로 단일 TimerTask를 사용한다.

WorkManagerTaskExecutor

* CommonJ WorkManager를 지원 구현체로 사용하고 스프링 컨텍스트에서 참조를 구성하는 핵심 클래스

테스크 네임스페이스

Scheduler 엘리먼트

<task:scheduler id="scheduler" pool-size="10"/>

* 지정한 크기의 스레드풀을 가진 ThreadPoolTaskScheduler 인스턴스를 생성한다.
* Id속성으로 제공한 값은 풀(pool)에서 스레드 이름의 접두사로 사용한다.

Executor 엘리먼트

<task:executor id="executor" pool-size="10"/>

* ThreadPoolTaskExecutor 인스턴스를 생성한다.
* id속성은 풀(pool)에서 슷레드 이름의 접두사로 이용한다.

Java Executor

1. 개요

Java의 쓰레드 구동 방식은 다양하지만 Executor는 단지 하나의 방식으로 간단하게 쓰레드를 구동할 수 있다.

* 태스크를 정의한 클래스의 생성
* Executor Service에 태스크 객체 제공

Executor 프레임웍은 다음과 같은 특징을 갖는다

* 쓰레드 풀을 사용
* 무거운 쓰레드는 미리 할당 가능
* 태스크와 쓰레드를 생성하고 관리하는 것을 분리
* 쓰레드 풀안의 쓰레드는 한번에 하나씩 여러 태스크를 실행
* 태스크 큐를 이용해 태스크를 관리
* Executor Service를 더 이상 필요 없으면 중지
* Executor Service가 멈추면 모든 쓰레드도 중지

2. 주요 클래스와 인터페이스

Executor 인터페이스: 태스크와 쓰레드를 분리하고 실행을 담당

Void execute(Runnable task);

ExecutorService 인터페이스: Executor 인터페이스를 확장하며 라이프 사이클을 제어

ValueOf vs toString

ValueOf vs parseInt

ValueOf(String): new Integer()으로 객체를 반환

ParseInt(String)은 int 기본 자료형을 반환

Integer.valueOf(int)