Hanbit eBook

Realtime 83

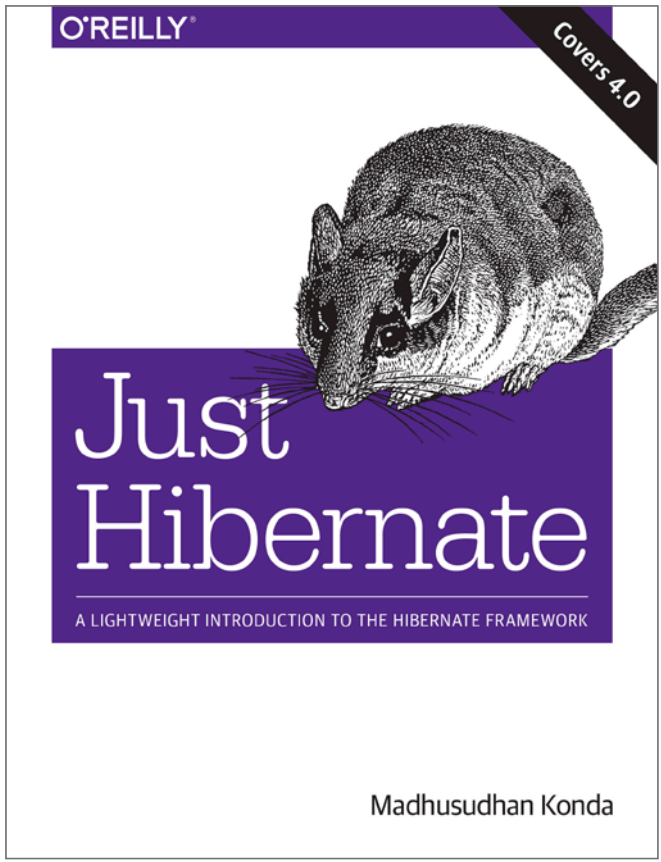
일주일 만에 끝내는

하이버네이트

Just Hibernate

마드후수단 콘다 지음 / 송기용 옮김

O'REILLY® 한빛미디어 Hanbit Media, Inc.



이 도서는 O'REILLY의
Just Hibernate
번역서입니다.

일주일 만에 끝내는 **하이버네이트**

초판발행 2014년 11월 05일

지은이 마드후수단 콘다 / **옮긴이** 송기용 / **펴낸이** 김태헌
펴낸곳 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부
전화 02-325-5544 / **팩스** 02-336-7124
등록 1999년 6월 24일 제10-1779호
ISBN 978-89-6848-722-4 15000 / **정가** 12,000원

총괄 배용석 / **책임편집** 김창수 / **기획·편집** 정지연
디자인 표지 여동일, 내지 스튜디오 [밈], 조판 최송실
영업 김형진, 김진불, 조유미 / **마케팅** 박상용, 김옥현

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주십시오.
한빛미디어 홈페이지 www.hanbit.co.kr / **이메일** ask@hanbit.co.kr

지금 하지 않으면 할 수 없는 일이 있습니다.
책으로 펴내고 싶은 아이디어나 원고를 메일(ebookwriter@hanbit.co.kr)로 보내주세요.
한빛미디어(주)는 여러분의 소중한 경험과 지식을 기다리고 있습니다.

저자 소개

저자_ **마드후수단 콘다**Madhusudhan Konda

마드후수단 콘다는 런던의 투자은행과 금융기관에서 경력직 자바 컨설턴트로 일하고 있다. 배포, 멀티 스레드, 확장 가능한 N-티어 아키텍처에 관심이 있으며, 금융 관련 고빈도high-frequency와 저지연low-latency 애플리케이션 아키텍트다. 집필에 남다른 애정이 있으며 멘토링에도 관심이 많다.

역자 소개

역자_ **송기용**

오픈소스 프레임워크를 사용한 웹 애플리케이션을 개발하고 있으며, 최근에는 함수형 프로그래밍에 관심이 많다. 현재 온라인 음원 서비스 회사에 재직 중이다.

저자의 말

저의 책들에 대한 긍정적이거나 부정적인 피드백, 리뷰, 의견, 제안, 글을 더 쓸 수 있는 힘이 되는 칭찬도 받았습니다. 저는 이 모두를 건설적으로 받아들였습니다. 여러분의 기대에 미치지 못한 글일지라도 지적해 주신다면 다음에는 최선을 다할 것입니다. "좋은 사람은 포기하지 않아요, 아빠."라고 아들 조슈아가 말한 것처럼 포기하지 않을 겁니다. 제가 즐겁게 글을 쓴 만큼 이 책을 즐기셨으면 좋겠습니다.

저나 제 글이 마음에 들지 않더라도 연락 주세요. 여러분의 의견은 항상 제게 큰 힘이 됩니다. 런던이나 주변에 계신다면 메시지나 메일을 주세요. 커피나 케이크를 함께 할 수 있을지도 모릅니다.

다음 Just 시리즈로 『Just Java 8』이 나올 예정입니다. 관심을 가져 주세요.

마드후수단 콘다

www.madhusudhan.com
m.konda@outlook.com
@mkonda007

역자 서문

처음 이 책의 번역을 맡았을 때 경험이 없어 생기는 걱정보다는 기쁜 마음이 더 컸습니다. 동료들과 IT 관련 기술이나 정보를 나누면서 느꼈던 즐거움을 좀 더 많은 분과 나눌 수 있지 않을까 하는 기대가 컸기 때문입니다.

하이버네이트나 JPA 관련 서적은 다른 기술 서적보다 찾기가 어려워 용어 선택이나 자연스럽지 못한 표현들 때문에 고민을 많이 했습니다.

이 책은 하이버네이트의 기초와 실무에서 사용할 수 있는 내용을 모두 담고 있습니다. 구글링을 통해 필요한 지식을 단편적으로 얻는 방법도 있지만, 책을 통해 하이버네이트 프레임워크의 시작부터 중고급 주제까지 살펴보는 것도 좋은 방법의 하나라고 생각합니다. 저 역시 실무에서 JPA 명세로는 한계가 있었으니까요.

이 책을 통해 개발자 여러분이 하이버네이트를 올바르게 사용하고자 하는 마음이 생긴다면 더할 나위 없이 기쁠 것 같습니다. 항상 독자에게 도움이 되는 책을 만들기 위해 개발자 입장에서 고민하시는 김창수 팀장님과 초보 역자인 저에게 하나부터 열까지 맞춰주느라 애써주신 정지연 과장님께 감사드립니다.

그리고 마지막으로 번역의 즐거움과 보람을 알게 해 준 아내 민정과 언제나 응원해 주시는 부모님께 감사의 마음을 전합니다.

일러두기

이 책의 대상 독자

이 책은 하이버네이트의 기초부터 중급까지 다룹니다. 예제 중심의 빠르고 쉽고 간단한 하이버네이트 입문서를 찾고 있다면, 이 책이 바로 그 책입니다. 하이버네이트 관련 프로젝트를 시작하기 전에 이 프레임워크를 배울 시간이 일주일밖에 없다면, 제대로 고르신 겁니다. 또한, 하이버네이트를 알고 있지만 세부 내용에 자신이 없다면 이 책을 통해 자신감을 얻을 수 있습니다.

이 책은 기술을 빠르게 습득하고자 하는 사람들을 위한 책이므로 하이버네이트에 능숙한 개발자에게는 적합하지 않습니다. 그러나 읽다 보면 흥미로운 부분을 찾을 수도 있으니 한번 훑어봐도 무방합니다.

다만 한 가지, 이 책은 하이버네이트 프레임워크에 대한 바이블이 아닙니다. 하이버네이트에 대한 내용을 적은 분량 안에 집약해서 넣었으므로 고급 기술의 책을 찾고 있다면 이 책을 추천하지 않습니다. 또한, 난관에 빠진 프로젝트의 솔루션을 찾으려고 이 책을 선택했다면 해결책을 찾기 힘들지도 모릅니다.

이 책을 집필한 이유

두 코스로 제공되는 식사처럼 배움에도 두 단계가 있다고 생각합니다. 첫 번째 코스는 간단하고 쉽지만 입맛을 돋우며 두 번째 코스에 대해 기대를 하게 만듭니다. 또한, 허기를 조금 채워줄 뿐만 아니라 두 번째 코스로 무엇이 나올지 맛보기를 제공합니다.

많은 책은 두 코스의 음식을 한꺼번에 제공합니다. 이러한 방식이 좋을 수도 있지

만, 사람들 대부분은 깊이 있는 기술을 배울만한 열정이나 시간 또는 공간이 부족합니다. 또한, 다른 레스토랑에서 파는 동일한 메뉴들이 우리를 늘 혼란스럽게 합니다. 메뉴가 계속 바뀌는 건 말할 것도 없고요.

첫 번째 코스는 손님에게 다음 코스를 위해 식당에 계속 머물러야겠다는 확신을 줄 수 있는, 흥미롭고 식욕을 돋우는 간단한 코스여야 한다고 생각합니다. 이런 코스를 제공하는 것은 어렵고 때로는 벅찬 일입니다.

일곱 살 난 아들 조슈아의 숙제를 함께하면서 기본과 원리를 쉽고 간단한 방법으로 가르치는 것이 얼마나 중요하고 어려운지 새삼 깨달았습니다. 아들의 수준에 맞게 쉬운 용어와 예제로 설명하면 아이는 아무리 어려운 주제라도 쉽게 이해했습니다.

기본에서 헤매는 많은 프로그래머와 개발자를 보았습니다. 이들은 동료에게 도움을 청하는 걸 때로는 부끄럽게 생각합니다. 그리고 신기술을 훈련하거나 학습할 시간 없이 정해진 단기간 내에 결과를 만들어야 하는 현실에 등 떠밀려 프로젝트를 진행하는 몇몇 사람도 본 적이 있습니다.

신기술을 배우는 데 관심은 있지만, 방대한 문서나 매뉴얼 때문에 배움을 미뤄두는 사람들도 있습니다. 이런 사람들은 열정적이고 새로운 업무를 바로 시작하고 싶어 하지만 많은 양의 책을 읽고 이해할 만한 인내심이나 시간이 없고, 간단한 책을 읽고 기술을 익혀서 실전에 바로 적용하기를 원합니다. 일단 그 기술을 이해하고 나면 좀 더 깊이 있는 지식을 원하게 됩니다.

저는 새로운 무언가를 배울 때 기본 코드를 실행해 보고 좀 더 나은 코드를 따라 해 보기도 합니다. 즉, 하이버네이트 프레임워크가 어떻게 동작하는지 이해하기 시작

하면 다른 경로를 통해 배움의 갈증을 해소합니다. 이때부터 난도가 높고 깊이 있는 매뉴얼과 명세서 그리고 두꺼운 책을 찾기 시작합니다.

Just 시리즈는 독자들이 간단하더라도 재미있고 빠르게 읽을 수 있도록 썼습니다. 기술을 실용적인 예제 중심으로 핵심을 집어내서 전달하려 했습니다. 물론 쉽게 썼고요. Just 시리즈는 여러분에게 충분한 지식과 실무 프로젝트에서 일을 시작할 수 있는 자신감을 줄 것입니다.

이 책의 구성

이 책은 총 8장으로 구성되어 있습니다. 각 장은 한 개 또는 두 개의 특정 주제를 다루며, 소스 코드를 제공합니다.

각 장의 내용은 다음과 같습니다.

1장 기초

하이버네이트 사용 환경을 설정합니다. 문제 범위를 정의하고, JDBC를 이용한 해결법을 연습해 보고, 하이버네이트를 도입해서 문제점을 개선하겠습니다. 여기서 하이버네이트를 살짝 맛보게 됩니다.

2장 기본 개념

하이버네이트가 해결하려는 문제에 대해 알아봅니다. 하이버네이트 프레임워크의 실행과 동작을 살펴보기 위해 프레임워크에 대해 좀 더 깊이 있게, 하이버네이트의 핵심 부분을 자세히 알아보겠습니다.

3장 어노테이션

어노테이션을 이용하여서 하이버네이트 애플리케이션을 생성하는 것에 집중할 것입니다. 또한, 뒷장에서 다룰 하이버네이트 어노테이션에 대한 준비 작업을 위해 어노테이션의 기본을 다지게 됩니다.

4장 콜렉션 영속화

영속화된 콜렉션은 개발자에게 어려운 과제입니다. 이 장은 콜렉션을 가지고 동작하는 방법과 영속성의 메커니즘을 이해하는 데 도움을 주기 위해 할애하였습니다.

5장 연관 관계

하이버네이트에서 제공하는 연관 관계와 관계를 알아봅니다. '일대일', '다대다' 같은 기본 연관 관계에 관련된 예제들을 다룹니다. 이 장은 주제에서는 살짝 벗어나지만 가능하면 간단하게 설명하려고 노력했습니다. 매우 중요한 장으로, 연관 관계를 제대로 이해하는 것만으로도 반은 성공입니다.

6장 고급 개념

캐싱, 상속 전략, 타입, 필터와 같은 고급 주제를 다룹니다. 프레임워크와 프레임워크가 상속, 캐싱, 그 밖의 기능과 관련하여 제공하는 것에 대해 좀 더 깊이 알 수 있을 것입니다.

7장 하이버네이트 질의어

하이버네이트는 SQL과 유사하게 자체 질의어가 있습니다. 하이버네이트 질의어(HQL)에 대해 소개하고 예제와 함께 API에 대해 알아봅니다.

8장 자바 퍼시스턴스 API

하이버네이트 관점에서 자바의 영속성 세계의 표준인 자바 퍼시스턴스 API(JPA)에 대해 살펴봅니다. 또한, JPA 표준 구현을 위한 하이버네이트의 지원과 애플리케이션에서 JPA 표준을 어떻게 활용할 수 있는지 살펴보겠습니다.

참고사항

이 책에서 사용하는 소스 코드는 https://github.com/madhusudhankonda/jh.git에서 내려받을 수 있습니다.

이 아이콘은 제안이나 팁을 의미한다.

이 아이콘은 일반적인 주석을 의미한다.

이 아이콘은 경고나 주의를 의미한다.

한빛 eBook 리얼타임

한빛 eBook 리얼타임은 IT 개발자를 위한 eBook입니다.

요즘 IT 업계에는 하루가 멀다 하고 수많은 기술이 나타나고 사라져 갑니다. 인터넷을 아무리 뒤져도 조금이나마 정리된 정보를 찾는 것도 쉽지 않습니다. 또한 잘 정리되어 책으로 나오기까지는 오랜 시간이 걸립니다. 어떻게 하면 조금이라도 더 유용한 정보를 빠르게 얻을 수 있을까요? 어떻게 하면 남보다 조금 더 빨리 경험하고 습득한 지식을 공유하고 발전시켜 나갈 수 있을까요? 세상에는 수많은 종이책이 있습니다. 그리고 그 종이책을 그대로 옮긴 전자책도 많습니다. 전자책에는 전자책에 적합한 콘텐츠와 전자책의 특성을 살린 형식이 있다고 생각합니다.

한빛이 지금 생각하고 추구하는, 개발자를 위한 리얼타임 전자책은 이렇습니다.

1. **eBook Only - 빠르게 변화하는 IT 기술에 대해 핵심적인 정보를 신속하게 제공합니다.**

   500페이지 가까운 분량의 잘 정리된 도서(종이책)가 아니라, 핵심적인 내용을 빠르게 전달하기 위해 조금은 거칠지만 100페이지 내외의 전자책 전용으로 개발한 서비스입니다. 독자에게는 새로운 정보를 빨리 얻을 수 있는 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내는 선배, 전문가, 고수 분에게는 보다 쉽게 집필할 수 있는 기회가 될 수 있으리라 생각합니다. 또한 새로운 정보와 지식을 빠르게 전달하기 위해 O'Reilly의 전자책 번역 서비스도 하고 있습니다.

2. **무료로 업데이트되는, 전자책 전용 서비스입니다.**

   종이책으로는 기술의 변화 속도를 따라잡기가 쉽지 않습니다. 책이 일정 분량 이상으로 집필되고 정리되어 나오는 동안 기술은 이미 변해 있습니다. 전자책으로 출간된 이후에도 버전 업을 통해 중요한 기술적 변화가 있거나 저자(역자)와 독자가 소통하면서 보완하여 발전된 노하우가 정리되면 구매하신 분께 무료로 업데이트해 드립니다.

3. 독자의 편의를 위하여 DRM-Free로 제공합니다.

구매한 전자책을 다양한 IT 기기에서 자유롭게 활용할 수 있도록 DRM-Free PDF 포맷으로 제공합니다. 이는 독자 여러분과 한빛이 생각하고 추구하는 전자책을 만들어 나가기 위해 독자 여러분이 언제 어디서 어떤 기기를 사용하더라도 편리하게 전자책을 볼 수 있도록 하기 위함입니다.

4. 전자책 환경을 고려한 최적의 형태와 디자인에 담고자 노력했습니다.

종이책을 그대로 옮겨 놓아 가독성이 떨어지고 읽기 힘든 전자책이 아니라, 전자책의 환경에 가능한 한 최적화하여 쾌적한 경험을 드리고자 합니다. 링크 등의 기능을 적극적으로 이용할 수 있음은 물론이고 글자 크기나 행간, 여백 등을 전자책에 가장 최적화된 형태로 새롭게 디자인하였습니다.

앞으로도 독자 여러분의 충고에 귀 기울이며 지속해서 발전시켜 나가도록 하겠습니다.

차례

1 | 기초

두 개의 서로 다른 소프트웨어 세계가 있는데, 하나는 객체로 알려진 자바 세계고 다른 하나는 데이터가 왕인 관계형 데이터베이스 세계다.

자바 개발자는 작업할 때 언제나 객체를 다룬다. 객체는 현실 세계를 모델링하는 상태state와 행동behavior을 나타낸다. 자바 애플리케이션에서 객체의 영속성object persistence은 필요조건이다. 상태는 내구성 있는 스토리지durable storage에 영속화되도록 모델링되어 있기 때문에 영구적이다.

하지만 데이터를 보관할 때는 관계형 데이터베이스에 의존하게 된다. 관계형 데이터베이스에서 데이터는 전통적으로 로우row와 컬럼column 형태로 데이터 간의 관계relationship나 연관 관계association를 나타낸다. 자바 개발자에게 자바 객체를 관계형 세상으로 가져온다는 것은 항상 어렵고 복잡한 일이다. 이런 과정을 객체-관계 연결ORM, Object-Relational Mapping이라 한다.

이 장에서는 객체 영속성 문제를 살펴봄으로써 하이버네이트에 대한 논의를 시작하겠다. 이런 과제에 직면한 우리를 도와줄 JDBC와 하이버네이트Hibernate 같은 여러 기술과 툴에 대해 알아보고, 이 기술들을 비교 대조해 볼 것이다. 그리고 하이버네이트에서 어떻게 쉽고 편하게 영속화된 객체 관계형 모델을 얻는지 살펴본다.

1.1 하이버네이트의 탄생

온라인 뱅킹 시스템을 설계한다고 생각해 보자. 은행이 계좌 정보, 개인 정보, 거래 내역 등의 데이터를 안전하게 보관할 거라고 기대할 것이다. 이는 애플리케이션 정보가 오랫동안 파손되지 않고 영구적인 저장 공간에 있어야 한다는 걸 의미한다. 뱅킹 애플리케이션의 고객, 주소, 계정, 기타 도메인 객체는 설계대로 영속화된 데

이터로 사용될 것이다. 이 애플리케이션을 통해 영속화된 데이터들은 애플리케이션보다 더 오래 남아 있을 것이다. 다시 말해, 온라인 뱅킹을 하다가 폰뱅킹으로 바꾼다 하더라도 뱅킹 애플리케이션에서 생성된 데이터들은 원할 때 볼 수 있고 사용할 수 있어야 한다.

이제 영속화된 객체(객체의 상태는 영속화할 데이터)는 현실 세계 대부분의 애플리케이션을 위한 필요조건이라는 것을 알게 되었다. 그리고 데이터를 저장하기 위해서 데이터베이스라는 내구성 있는 저장 공간이 필요하다. 다양한 부가기능을 제공하는 데이터베이스 벤더들(오라클, MySQL, DB2, JavaDB 등)은 많다.

어떻게 하면 객체 간 관계를 데이터베이스로 영속화할 수 있을까?

기업에서는 프로그래밍 플랫폼으로 객체지향 언어(Java 등)를, 데이터베이스로는 관계형 데이터베이스(Oracle, MySQL, Sybase 등)를 도입하고 있다. 소위 객체 관계형 임피던스object-relational impedance 불일치에도 불구하고 이 두 소프트웨어 기술은 현실 세계 대부분의 애플리케이션에 꼭 필요하다.

다음 장에서는 이 불일치에 대해 자세하게 살펴보고 여기서는 주요 사항을 다음과 같이 간단히 소개한다.

- 상속은 객체지향 프로그래밍의 기본 원리다(상속 개념 없이는 객체 연관 관계를 설계할 수 없으며, 데이터베이스는 상속 개념을 알지 못한다).
- 일대일, 일대다, 다대다와 같은 다양한 객체 관계의 구현을 해도, 데이터베이스에서는 모든 관계를 지원하지 않으므로 상속은 결국 실패로 끝난다.
- 마지막으로 동일성identity의 불일치가 있다. 객체는 동일성과 동등성equality 둘 다를 가지고 있지만, 데이터베이스 레코드는 컬럼 값으로만 식별된다.

개발자는 직접 작성한 프레임워크, 기술적 솔루션, 전략을 통해 객체-관계 간 차이에서 오는 불편함을 줄일 수 있다.

자바에는 데이터베이스 접근을 위한 표준 도구가 있는데, 이를 JDBC Java Database Connectivity API Application Programming Interface라고 한다. 최근까지도 JDBC API는 자바 애플리케이션에서 잘 활용되고 있다. 이 API는 작은 규모의 프로젝트에 적합하지만, 도메인 모델이 복잡해질수록 프로젝트는 매우 비효율적으로 변한다(그리고 때때로 통제가 불가능하다). 또한, 반복되는 많은 코드와 노동력이 필요할 뿐만 아니라 객체-관계 모델 매핑을 다루는 것 역시 필요 이상의 큰 비용이 필요하다.

이는 개발자에게 고통스러운 부분이다. 큰 문제 없이 데이터를 영속화할 수 있는 간단한 도구가 생기길 모두가 기다렸다. 하이버네이트 팀은 ORM 매핑 공간에서 발생하는 차이점을 발견했고 개발자의 일을 좀 더 편하게 해주는 간단한 프레임워크를 만들게 되었다.

이것이 하이버네이트의 시작이다. 하이버네이트는 나오자마자 큰 성공을 거두었고 단순함과 강력함을 특징으로 단숨에 ORM 툴 분야에서 가장 인기 있는 오픈소스로 자리매김하였다.

1.2 문제 범위

하이버네이트에 대해 자세히 알아보기 전에 하이버네이트가 나오게 된 이유를 예를 통해 알아보자.

사람들은 영화 보는 것을 좋아한다. 하지만 영화가 개봉할 때마다 영화를 다 볼 수 있을 만큼 시간이 많지 않다. 그래서 보고 싶은 영화를 '위시 리스트'로 만든다. 그러다 어느 날 문득 'JustMovies'라는 애플리케이션을 만들기로 마음먹는다. 'JustMovies'는 웹 기반 애플리케이션으로, 사용자는 개인 계정을 만들고 자신만의 영화 위시 리스트를 작성할 수 있다. 또한, 언제든지 웹사이트에서 위시 리스트에 영화를 추가하거나 수정 또는 삭제할 수 있다.

이때, 사용자마다 위시 리스트를 저장해야 하므로 반드시 데이터베이스 같은 내구성 있는 스토리지에 각 리스트가 저장되어야 한다. 자, 우선 데이터베이스에 저장하고 조회해서 가져오는 간단한 자바 애플리케이션을 만들어보자.

1.2.1 MovieManager 애플리케이션

데이터베이스에 영화 정보를 저장하고, 조회하고, 찾아서 가져오는 일을 주로 하는 자바 애플리케이션인 'MovieManager'가 있다. 자바로 구현된 애플리케이션이고 영화 정보를 저장해야 하므로 데이터베이스 테이블이 필요하다. 영화와 관련된 데이터는 [표 1-1]과 같이 MOVIES 테이블에 로우 형태로 저장한다.

[표 1-1] MOVIES 테이블

ID	TITLE	DIRECTOR	SYNOPSIS
1	Top Gun	Tony Scott	Maverick is a hot pilot…
2	Jaws	Steven Spielberg	A tale of a white shark!

각 컬럼은 `VanillaMovieManager` 애플리케이션에서 `Movie` 인스턴스를 나타낸다. 하이버네이트 없는 세상에 살고 있다고 가정해 보자. 필요한 것들을 구현할 수 있기를 바라면서 JDBC를 사용하여 몇 줄의 예제 코드를 작성하게 될 것이다.

JDBC 사용하기

어떤 데이터베이스 애플리케이션이든 시작은 데이터베이스를 연결하고 유지하는 일이다. 커넥션connection은 데이터베이스와 연결된 게이트웨이다. 자바 애플리케이션에서 데이터 작업을 수행하는 JDBC는 데이터베이스 속성에 기반을 둔 커넥션을 생성하는 커넥션 API를 제공한다. 데이터베이스 제공자는 일반적으로 데이터베이스 연결 메커니즘을 가지고 있는 클래스를 구현하고 있다. 예를 들어, MySQL에서는 `com.mysql.jdbc.Driver`이고, 더비derby(JavaDB)에서는 `org.apache.derby.jdbc.EmbeddedDriver`다.

이 책에서는 MySQL을 다룬다. 프로젝트와 데이터베이스 설정에 대한 자세한 설명은 1.7 하이버네이트 설정하기를 참고하기 바란다.

다음 소스 코드의 createConnection은 데이터베이스 연결을 생성하는 과정을 보여준다. 이 부분에서 드라이버 클래스를 초기화하고 DriverManager를 이용하여 커넥션을 얻는다.

```
public class VanillaMovieManager {
     private Connection connection = null;

     //데이터베이스 속성
     private String url = "jdbc:mysql://localhost:3307/JH";
     private String driverClass = "com.mysql.jdbc.Driver";
     private String username = "mkonda";
     private String password = "mypass";
     …
     private Connection getConnection() {
         try {
             Class.forName(driverClass).newInstance();
             connection = DriverManager.getConnection(url, username, password);
         } catch (Exception ex) {
             System.err.println("Exception:"+ ex.getMessage());
         }
         return connection;
     }
}
```

데이터베이스로 연결되고 유지되면, 다음 단계로 영속화를 위한 메소드를 작성하고 movie 엔티티Entity를 조회한다. JDBC를 다뤄봤다면 작성된 메소드 코드가 익숙할 것이다.

그다음, 영화 정보를 데이터베이스에 저장하는 메소드와 조회해서 가져오는 메소드를 추가한다. 이 메소드를 각각 persistMovie와 queryMovies라 하자. 다음 코드는 이들 메소드의 구현부다.

```
public class VanillaMovieManager {
     private String insertSql = "INSERT INTO MOVIES VALUES (?,?,?,?)";
     private String selectSql = "SELECT * FROM MOVIES";
     …

     private void persistMovie() {
         try {
             PreparedStatement pst = getConnection().prepareStatement(insertSql);
             pst.setInt(1, 1001);
             pst.setString(2, "Top Gun");
             pst.setString(3, "Action Film");
             pst.setString(4, "Tony Scott");

         // statement 실행하기
             pst.execute();
             System.out.println("Movie persisted successfully!");

         } catch (SQLException ex) {
             System.err.println(ex.getMessage());
             ex.printStackTrace();
         }
     }
     private void queryMovies() {
         try {
             Statement st = getConnection().createStatement();
             ResultSet rs = st.executeQuery("SELECT * FROM MOVIES");
             while (rs.next()) {
                 System.out.println("Movie Found: "
                           + rs.getInt("ID")
```

```
                        + ", Title:"
                        + rs.getString("TITLE"));
                }
            } catch (SQLException ex) {
                System.err.println(ex.getMessage());
            }
        }
    }
```

예제 코드의 내용은 다음과 같다.

1. 커넥션과 `insertSql` 문자열 변수를 통해 `PreparedStatement`를 생성한다.
2. `Preparedstatement` 객체에 영화 정보(컬럼 번호에 맞는 컬럼 값: 1은 ID, 2는 제목 등)를 넣는다.
3. `PreparedStatement` 객체를 실행하면 `Movies` 테이블에 영화 정보가 추가된다.
4. 데이터베이스의 모든 영화를 대상으로 조회 쿼리를 실행하고, 콘솔에 리스트를 출력한다.

각 단계는 추가 설명 없이도 쉽게 이해될 것이다. `PreparedStatement`를 생성하고, 각 컬럼에 맞는 데이터를 넣은 후 실행한다. 실행이 잘 되었는지 확인하려면 모든 영화 정보의 조회와 콘솔 출력을 위한 `SELECT` 문 쿼리를 실행하면 된다.

그러나 몇 가지 주의 사항이 있다.

- 컬럼 값을 입력(또는 조회)하기 위해 미리 정의된 SQL문을 사용한다.
- 컬럼 값은 컬럼 순번(또는 컬럼 이름)으로 하나씩 설정한다.
- SQL문이 제대로 실행되지 않으면 `SQLException`이 발생한다.

간단한 프로그램에서는 값과 함께 코드를 생성하고 실행하는 방법으로도 충분하다. 그러나 현실 세계의 프로그램은 훨씬 더 복잡하다. 비즈니스 로직과 상관없는

엄청난 양의 코드를 기꺼이 기쁜 마음으로 작성하고 관리할 수 있다면 JDBC를 사용해도 된다. 객체 간 복잡한 관계나 많은 테이블을 다뤄야 한다면 JDBC 도입은 문제가 된다. 이렇게 데이터를 다루는 것은 객체지향 원리를 사용하는 어떠한 종류나 형태에도 해당하지 않는다.

1.2.2 Movie 애플리케이션 구현

Movie 객체를 바로 영속화하기 위해 유틸리티 클래스에서 제공하는 persist() 같은 메소드를 호출하면 좋지 않을까? 객체지향 프로그래머가 이런 편리한 기능을 바라는 것이 죄가 되겠는가!

이를 위해 영화 한 편을 나타내는 POJO[Plain Old Java Object]를 만들겠다. 개봉했거나 아직 개봉하지 않은 모든 필름 영화를 위해 새로운 Movie 객체를 생성한다. 다음은 Movie POJO의 선언부다.

```
public class Movie {
     private int id = 0;
     private String title = null;
     private String synopsis = null;
     private String director = null;
     …
     // Setters와 getters 생략
}
```

따라서 이제 POJO 객체를 데이터베이스 테이블 MOVIES로 영속화하는 기능이 필요하다. 본질적으로 객체 모델(Movie 객체)을 관계형 모델(테이블의 컬럼)로 변환하는 것이다.

변환 작업을 위해 MoviePersistor 클래스를 만들어보자.

```
// Pseudocode
public class MoviePersistor {
     public void persist(Movie movie) {
        // 이 부분에서 Movie 객체가 영속된다
     }
     public void fetch(String title) {
        // 이 부분에서 title을 매개변수로 movie 객체를 가져온다
     }
     …
}
```

아직 persist와 fetch 기능을 구현하지 않았다(이것은 프로그램의 주제가 아니다). 자, MoviePersistor 유틸리티 클래스를 이용하여 Movie를 영속화하는 것이 얼마나 쉬운지 살펴보자.

```
//Pseudocode
MoviePersistor moviePersistor = new MoviePersistor();
Movie movie = new Movie();
movie.setId(1);
movie.setTitle("Jaws");
movie.setDirector("Steven Spielberg");
movie.setSynopsis("Story of a great white shark!");

moviePersistor.persist(movie);
```

멋지지 않은가? 한 편의 필름 영화에 해당하는 하나의 POJO는 유틸리티 클래스를 통해 객체 모델에서 관계형 모델로, 데이터베이스 테이블 하나의 레코드로 영속화된다.

Persist와 fetch 메소드를 실제로 구현하지 않을 것을 제외하고는 모든 것이 좋다. 이 기능을 구현하려면 데이터베이스 연결 기능뿐만 아니라 객체를 로우로 변환

하는 메커니즘도 필요하다(객체 속성을 데이터베이스 컬럼으로 매핑한 것과 같다).

이러한 변환의 핵심 부분과 영속화 메커니즘을 감추는 클래스를 통해 자신만의 프레임워크를 만들 수 있다(화면 뒤에 오래된 JDBC 구문을 숨기는 데 좋은 방법이 될 것이다). 비록 이 프레임워크를 구축하는 것이 복잡한 일은 아니지만, 구현하는 데 많은 시간과 노력이 필요하다.

시간이 지나면 조직의 영속성 관련 요구사항이 변하거나 심지어 데이터베이스를 오라클에서 MySQL로 이관해야 할 수도 있다. 이는 프레임워크가 아주 포괄적이고 기능적이며 기술적인 요구사항에 대해 많은 부분을 제공해야 한다는 의미이기도 하다.

필자의 경험에 비춰보면 개인적인 프레임워크는 다루기 힘들고 유연성과 확장성이 부족하며 때로는 구닥다리가 될 수도 있다. 조직이 원하는 요구사항이 말도 안 되는 것이 아니라면(회사에서 화성에 데이터를 남기길 원할지도 모른다) 인터넷 검색으로 여기서 언급한 사항을 가장 만족하는 프레임워크를 선택할 것을 추천한다.

그런데 이미 관계형 데이터베이스로 객체를 영속화하는 기능을 가진 훌륭한 프레임워크가 존재한다. 그것이 바로 하이버네이트다!

자 그럼, 하이버네이트로 기존의 동일 메소드를 어떻게 리팩토링하는지 살펴보자.

```
public class BasicMovieManager {
     private void persistMovie(Movie movie) {
         Session session = sessionFactory.getCurrentSession();
         …
         session.save(movie);
     }
     …
}
```

한 줄의 코드 session.save(movie)의 실행으로 Movie 인스턴스를 데이터베이스로 저장하는 것을 보았는가? 앞에서 원했던 것이 아닌가? 객체지향 방식으로 간단히 영속화 객체를 저장하는 클래스인 Hibernate API 클래스는 쉽고 편하게 자바 객체를 다루기 위한 메소드들을 제공한다. 이는 곧, 다량의 카페인을 섭취하고 고심하면서 프레임워크를 작성하거나 JDBC를 사용해서 많은 양의 코드를 작성하지 않아도 된다는 의미다.

하이버네이트는 객체 영속화 기능을 제공하지만, 일회성 설정과 사용 목적에 맞는 매핑 정보는 직접 제공해야 한다. 다음 절에서 이 내용을 자세히 살펴보겠다.

1.3 하이버네이트 사용하기

하이버네이트 애플리케이션을 생성하기 위한 일반적인 단계는 다음과 같다.

1. 데이터베이스 연결 설정하기
2. 매핑 정보 생성하기
3. 클래스 영속화하기

다음은 MovieManager 애플리케이션을 자바-하이버네이트 버전으로 개발하는 일반적인 단계다.

1. Movie 도메인Domain 객체 생성하기(데이터 테이블에 상응하는 도메인 모델 POJO)
2. 하이버네이트 속성properties 파일과 매핑 파일 같은 설정 파일 생성하기
3. Movie 객체를 관리(입력, 변경, 삭제, 조회)할 테스트 클라이언트 생성하기

앞에서 Movie POJO를 이미 준비했으므로 다시 만들 필요는 없다.

하이버네이트 애플리케이션의 핵심은 설정configuration에 있다. 두 개의 설정 파일이 필요한데, 하나는 데이터베이스 연결을 생성하고, 다른 하나는 객체와 테이블을 매

핑하는 역할을 한다. JDBC를 살펴보면 애플리케이션에 데이터베이스 정보를 제공하고, 데이터를 다루기 위해 커넥션을 연결한다. 매핑 설정은 테이블의 컬럼에 매핑된 객체 속성을 정의한다. 이번 장의 목표는 빨리 시작해 보는 것이므로 상세한 부분까지는 들어가지 않겠다.

다음 절에서 하이버네이트 애플리케이션을 만드는 과정을 자세히 살펴보자.

1.4 데이터베이스 연결 설정하기

데이터베이스를 연결하려면 하이버네이트에 데이터베이스, 테이블, 클래스, 기타 장비에 대한 상세 정보를 제공해야 한다. 상세 정보는 XML 파일 형태(일반적인 형태는 hibernate.cfg.xml)로 제공하거나 '이름/값' 형태의 텍스트 파일(일반적인 형태는 hibernate.properties)로 제공한다. 여기서서는 XML 형태를 사용하겠다. 파일명은 hibernate.cfg.xml로 하고, 이 설정 파일은 하이버네이트에서 자동으로 로드된다.

다음은 xml 설정 파일에 대한 설명이다. MySQL 데이터베이스를 사용하므로 MySQL 데이터베이스를 위한 연결 상세 정보는 다음 hibernate.cfg.xml 파일에 선언되어 있다.

```
<hibernate-configuration>
    <session-factory>
        <property name="connection.url">
            jdbc:mysql://localhost:3307/JH
        </property>
        <property name="connection.driver_class">
            com.mysql.jdbc.Driver
        </property>
        <property name="connection.username">
            mkonda
        </property>
```

```
        <property name="connection.password">
            password
        </property>
        <property name="dialect">
            org.hibernate.dialect.MySQL5Dialect
        </property>
        <mapping resource="Movie.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

이 파일은 MySQL 데이터베이스에 실시간 연결을 위한 정보를 갖고 있다.

앞선 속성들을 '이름/값' 형태로 표현할 수도 있다. 예를 들면, hibernate.properties 텍스트 파일에서 '이름/값' 형태와 같이 동일 정보를 나타낼 수 있다.

```
hibernate.connection.driver_class = com.mysql.jdbc.Driver
hibernate.connection.url = jdbc:mysql://localhost:3307/JH hibernate.dialect =
org.hibernate.dialect.MySQL5Dialect
```

`connection.url`은 어느 URL에 연결해야 할지를 가리키고, `driver_class`는 커넥션 생성에 맞는 `Driver` 클래스를 보여준다. 그리고 'dialect'에는 사용하는 데이터베이스의 dialect를 알려준다(여기서는 MySQL).

hibernate.properties 파일을 따른다면 모든 속성명은 hibernate.* properties와 같은 패턴으로 'hibernate' 접두사와 함께 정의해야 한다. 설정 파일을 제공하고 나면 매핑 파일과 설정 파일의 위치 정보도 제공해야 한다. 앞서 언급했듯이 매핑 파일은 객체 속성과 맞는 로우와 컬럼 값에 대한 객체 속성을 가지고 있다. 매핑 정보는 각 파일에 선언되었는데, 일반적으로 '.hbm.xml' 접미사를 사용한다. 다음과 같이 앞에서 본 설정 파일의 `mapping` 요소를 포함하는 매핑 설정 파일에 대한 정보를 제공해야 한다.

```
<hibernate-configuration>
    <session-factory>
        ...
        <mapping resource="Movie.hbm.xml" />
        <mapping resource="Account.hbm.xml" />
        <mapping resource="Trade.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

resource 속성은 하이버네이트에서 로드할 매핑 리소스명을 가리킨다. 예를 들어, Movie.hbm.xml은 Movie 테이블과 Movie 객체 간 매핑 방법에 대한 상세 정보를 가진 매핑 파일이다. Account.hbm.xml과 Trade.hbm.xml에서도 마찬가지로 매핑 정보를 볼 수 있다. 이 매핑 파일에 대해서 잠시 살펴보겠다.

하이버네이트는 설정 파일로 무엇을 할까?

하이버네이트 프레임워크는 SessionFactory를 생성하기 위해 설정 파일을 로드한다. SessionFactory는 Session 생성을 위한 Thread-safe한 전역 클래스다. 여기서는 하나의 SessionFactory를 생성하고 애플리케이션 전반에 걸쳐 공유하여 사용한다.

SessionFactory는 오직 하나의 데이터베이스를 위해 정의된 클래스라는 걸 기억하자. 예를 들어, MySQL에서 또 다른 데이터베이스를 사용한다고 하자. 해당 데이터베이스를 위한 별개의 SessionFactory를 생성하려면 hibernate.hbm.xml 파일에 적합한 설정을 정의해야 한다.

SessionFactory의 역할은 Session 객체를 생성하는 것이다. Session은 데이터베이스로 연결된 하나의 게이트웨이다. Session의 임무는 테이블에 레코드를 저장하고, 로딩하며, 조회해서 가져오는 데이터베이스의 모든 동작을 처리하는 것이다.

또한, 애플리케이션 간 트랜잭션을 유지하기도 한다. 데이터베이스 접속을 포함한 이런 동작은 '트랜잭션transaction'이라 불리는 하나의 처리 단위에 포함된다. 그래서 트랜잭션 안의 모든 작업은 성공적으로 완료되거나 롤백될 수 있다.

설정 파일은 SessionFactory 인스턴스를 통해 Session을 생성하기 위해 사용된다는 것을 기억하자. Session 객체는 Thread-safe하지 않으므로 다른 클래스 사이에서 공유되어서는 안 된다는 점도 기억하자. 앞으로 진행 과정을 통해 Session을 어떻게 사용해야 할지를 자세히 알아보겠다.

1.5 매핑 정의 만들기

데이터베이스 연결 설정이 준비되면, 다음 단계는 객체 테이블 간 매핑 정의가 된 Movie.hbm.xml 파일을 만드는 것이다. 다음 XML 코드는 MOVIES 테이블에 맞는 Movie 객체의 매핑 정보를 정의하고 있다.

```
<hibernate-mapping>
    <class name="com.madhusudhan.jh.domain.Movie" table="MOVIES">
        <id name="id" column="ID">
            <generator class="native"/>
        </id>
        <property name="title" column="TITLE"/>
        <property name="director" column="DIRECTOR"/>
        <property name="synopsis" column="SYNOPSIS"/>
    </class>
</hibernate-mapping>
```

이 매핑 파일에서 살펴볼 것이 많다. hibernate-mapping 요소는 객체-테이블 간 매핑 정보를 모두 가지고 있다. 객체 각각의 매핑 정보는 이 class 요소 하위에 선언되고, class 태그의 name 속성은 POJO 도메인 클래스 com.madhusudhan.

jh.domain.Movie를 참조한다. 하지만 table 속성은 객체를 영속화할 MOVIES 테이블을 참조한다.

나머지 속성은 객체 변수와 테이블 컬럼 간의 매핑 정보를 나타낸다(예를 들어, id는 ID, title은 TITLE, director는 DIRECTOR). 각 객체는 테이블의 기본키Primary Key와 같은 고유한 식별자를 가져야 한다. 네이티브native 전략을 사용한 id 태그를 구현함으로써 식별자를 설정할 수 있다. 아직 id 태그나 생성 전략generation strategy에 관심을 가질 필요는 없다. 이는 다음 장에서 자세히 살펴보겠다.

1.6 객체 영속화하기

지금까지 하이버네이트 설정을 알아보았다. 이제 하이버네이트로 객체를 영속화하는 클라이언트를 만들어보자.

Session 객체를 생성할 SessionFactory 인스턴스가 필요하다. 다음은 SessionFactory 클래스 생성을 위한 초기 설정 부분이다.

```
public class BasicMovieManager {
     private SessionFactory sessionFactory = null;

     // 하이버네이트 4.2 버전을 이용해서 SessionFactory 생성
     private void initSessionFactory(){
         Configuration config = new Configuration().configure();
         // Properties 설정파일을 가지고 Registry 객체 생성
         ServiceRegistry serviceRegistry = new ServiceRegistryBuilder().applyS
ettings(config.getProperties()).buildServiceRegistry();

         // SessionFactory 생성
         sessionFactory = config.buildSessionFactory(serviceRegistry);
     }
     …
}
```

하이버네이트를 시작할 때 클래스패스classpath에서 hibenate.cfg.xml이나 hibernate.properties와 같은 디폴트 파일명을 찾아서 로드하므로 속성과 설정, 매핑 파일을 꼭 명시할 필요는 없다.

SessionFactory 클래스 초기화를 위한 설정은 글을 쓰고 있을 당시 최신 버전인 4.2에 해당된다. 하이버네이트 4.x 버전부터 서비스 레지스트리service registries가 소개되었고, 이는 뒷부분에서 다루겠다. 3.x 대 버전에서는 Configuration 객체를 생성하기 위해 configure 메소드는 클래스패스 내 hibernate.cfg.xml(또는 hibernate.properties) 파일을 찾는다. 그리고 생성된 Configuration 객체는 SessionFactory 인스턴스를 만든다. 하이버네이트 4.x 이전 버전에서는 SessionFactory를 초기화하기 위해 다음 코드를 사용한다.

```
// 하이버네이트 3.x 버전을 이용해서 SessionFactory 생성
private void init3x(){
   sessionFactory =
       new Configuration().configure().buildSessionFactory();
}
```

4.x 버전에서는 Configuration 객체로부터 속성 매핑 정보를 얻는 서비스 레지스트리의 도입으로 약간 변경된 부분이 있다.어느 버전을 선택하더라도 이렇게 생성된 SessionFactory는 동일한 역할을 하고 Session 또한 마찬가지다.

1.6.1 Persist 메소드 만들기

이제 BasicMovieManager 클래스를 만들어보자. 영화 정보를 영속화하기 위해 Session의 save 메소드를 이용하여 클래스 내에는 persist 메소드를 선언한다.

```
public class BasicMovieManager {
     private void persistMovie(Movie movie) {
```

```
        Session session = sessionFactory.getCurrentSession();
        session.beginTransaction();
        session.save(movie);
        session.getTransaction().commit();
    }
}
```

간단해 보이지 않는가? 객체를 저장하는 비즈니스 기능에 집중하는 것 말고 불필요하거나 반복적인 코드는 전혀 작성할 필요가 없다.

SessionFactory로부터 Session을 얻는다. 그러고 나서 트랜잭션 객체를 생성하고(트랜잭션에 관해서는 다음 장에서 자세히 살펴볼 예정이다). Session.save 메소드를 통해 매개변수로 들어온 Movie 객체를 영속화한다. 마지막으로 트랜잭션 커밋Commit을 실행해서 데이터베이스에 Movie 객체를 영구적으로 저장한다.

1.6.2 영속화된 데이터 테스트

영속화된 데이터를 테스트하는 방법은 두 가지가 있다. 데이터베이스에서 네이티브 SQL 쿼리를 실행하는 방법과 테스트 클라이언트 프로그램을 만드는 방법이다.

SELECT * FROM MOVIES와 같은 SQL 쿼리를 데이터베이스에서 실행하면 애플리케이션을 통해 저장된 레코드 전체를 조회한다. select 쿼리문의 결과는 [표 1-2]와 같이 출력된다.

[표 1-2] MOVIES

ID	TITLE	DIRECTOR	SYNOPSIS
1	Top Gun	Tony Scott	Maverick is a hot pilot…
2	Jaws	Steven Spielberg	A tale of a white shark!

다른 방법은 테스트 클라이언트(findMovie라고 하자)의 새 메소드를 만드는 것이다.

이 메소드는 Session에서 제공하는 load 메소드를 이용하여서 데이터베이스의 레코드를 조회한다. 영화 정보를 조회하기 위해 영화 ID를 인수로 넘겨서 findMovie를 호출한다.

```
public class BasicMovieManager {
      …
      private void findMovie(int movieId) {
          Session session = sessionFactory.getCurrentSession();
          session.beginTransaction();

          Movie movie = (Movie)session.load(Movie.class, movieId);

          System.out.println("Movie:"+movie);
          session.getTransaction().commit();
      }
}
```

Session API에서 제공하는 load 메소드는 주어진 식별자로 해당하는 Movie 객체를 조회한다. 하이버네이트의 보이지 않는 부분에서 SELECT 문을 사용할 거라 생각한다면, 맞다!

테이블의 모든 영화 정보를 조회하고 싶다면, "from Movie"라는 간단한 쿼리문과 함께 Query 객체를 생성하고 실행하면 된다. Query 객체(session.createQuery로 생성된)의 list 메소드는 다음과 같이 영화 리스트를 반환한다.

```
public class BasicMovieManager {
      // 모든 영화 조회하기
      private void findAll() {
          Session session = sessionFactory.getCurrentSession();
          session.beginTransaction();
          List<Movie> movies = session.createQuery("from Movie").list();*
```

```
        session.getTransaction().commit();
        System.out.println("All Movies:"+movies);
    }
    …
}
```

1.7 하이버네이트 설정하기

하이버네이트 프로젝트 설정은 쉬운 편이다. 여기서는 넷빈즈NetBeans IDE에서 개발된 메이븐Maven 기반의 프로젝트 소스를 사용한다. 개발환경 설정에 관해 자세히 설명하지는 않지만, 다음 단계를 따라 진행하면 도움이 될 것이다. 비록 넷빈즈 IDE에서 개발된 코드지만, 기호에 맞는 다른 IDE를 사용해도 괜찮다. 또한, 더비나 HyperSQL과 같은 인-메모리In-Memory 데이터베이스로 바꿔 사용해도 괜찮다.

우선 JDK 5.0+, 넷빈즈 IDE, 메이븐, MySQL 데이터베이스로 구성된 필수 개발환경을 준비한다. 여기서는 JDK 6, 넷빈즈 7.3, 메이븐 3.2, MySQL 5.2, Hibernate 4.2.3.Final을 사용했다.

개발환경이 준비되었다면, 다음 단계는 넷빈즈(또는 이클립스) IDE에서 새 메이븐 프로젝트를 생성하는 것이다. 다음과 같이 pom.xml 파일에 적당한 하이버네이트 라이브러리 의존성dependencies을 추가하자.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
          http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.madhusudhan</groupId>
<artifactId>just-hibernate</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

```
<dependencies>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>4.2.3.Final</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.18</version>
    </dependency>
    ...
</project>
```

메이븐은 프로젝트를 빌드할 때 관련 라이브러리 의존성을 해결해 준다. 제공하는 소스코드를 내려받고, 각자 주로 사용하는 IDE로 프로젝트를 가져오길 바란다.

다음 단계로 데이터베이스를 준비한다. 데이터베이스가 이미 있다면 이 단계는 넘어간다. 여기서는 MySQL 데이터베이스를 사용하므로 최신 MySQL 패키지를 설치한다. MySQL(또는 다른 데이터베이스)이 설치되었다면 다음과 같이 'JH'를 스키마 명으로 생성한다.

```
CREATE SCHEMA JH;
```

프로젝트에서 테이블 대부분은 하이버네이트에 의해 자동으로 생성된다. 하이버네이트는 상황에 따라(테이블이 없다면) 설정 파일을 읽고 테이블을 생성한다. 테이블 자동생성을 위해 하이버네이트 설정 파일(hibernate.cfg.xml)의 속성(hbm2ddl.auto)에 다음과 같이 값을 정의한다.

```
<property name="hbm2ddl.auto">update</property>
```

이 속성값을 설정할 때 테이블이 없으면 자동으로 테이블이 생성되고, 테이블 스키마의 변경이 생기면 자동으로 반영된다.

`hbm2ddl.auto` 속성을 상용화된 서비스에 사용해서는 안 되며 모든 테이블 정의를 통해 스키마를 생성해야 한다.

여기까지 하이버네이트의 기초에 대해 알아보았다.

다루기 힘든 JDBC 구문과 커넥션의 핵심 부분을 숨기는 메커니즘을 기다려왔고 번거로운 컬럼 설정setting/getting 없이 POJO 객체를 데이터베이스로 바로 저장할 수 있는 기능을 만들고 싶었다. 하이버네이트를 도입함으로써 그동안 바라던 것을 이루게 되었다. 궁금한 것이 많겠지만 하이버네이트와 함께 하는 여정을 통해서 이런 궁금증이 해결될 것이니 계속 함께 가자.

1.8 요약

이 장에서는 예제를 통해 객체-관계-모델object-relational-model 문제를 살펴보았다. 데이터 접근을 위해 JDBC를 사용할 수 있지만, 많은 매핑 수작업과 불필요하게 반복되는 코드가 요구된다는 것을 알게 되었다. 객체-관계object-to-relational의 데이터 영속화 문제를 풀기 위해 몇 단계를 거쳤고, 하이버네이트를 소개했다. `SessionFactory`와 `Session`에 관한 하이버네이트의 고급 개념도 살펴보았다. 하이버네이트 프레임워크로 JDBC 예제를 리팩토링했다. 그리고 예상과 같이 POJO를 영속화하고 조회하는 데 성공했다.

다음 장에서는 하이버네이트의 기본 개념에 대해 좀 더 자세히 살펴보겠다.