1. SW 재사용 방안들

1)Copy & Paste

-초보적인 재사용 방식으로 비슷한 예제를 다른 Source에서 복사해서 사용한다.

```
GregorianCalendar date = (GregorianCalendar)Calendar.getInstance();
SimpleDateFormat df = new SimpleDateFormat("yyyyMMdd");
String date = df.format(date);
```

-예를 들어 A라는 Class에서 Date type을 String type으로 변환하는 코딩을 하고, Class B에서 동일한 로직이 필요하여 복사했다고 가정한 경우,

-JDK버전이 바뀌어 동일한 기능을 제공하는 향상된 Interface가 나오면 위의 코드를 사용한 A, B Class를 모두 변경해야 한다.

2)Method(or Function)

-자주 사용되고, 유사한 기능들을 모아 메소드로 정의하여 재사용함.

```
public class DateUtility{
    public static String toStringToday(String format){
        GregorianCalendar date = (GregorianCalendar)Calendar.getInstance();
        SimpleDateFormat df = new SimpleDateFormat("yyyyMMdd");
        String date = df.format(date);
    }
}
String sdate = DateUtility.toStringToday("yyyyMMdd");
```

-JDK 버전이 바뀌거나 메소드의 내용이 수정되더라도 해당 Class를 모두 수정할 필요 없이 toStringToday() 메소드의 내용만 수정하면 된다.

-toStringToday() 메소드의 Signature를 변경하면 이 메소드를 사용하는 모든 Class에 영향을 준다.

-메소드 재사용방법은 '복사 & 붙이기'보다는 진보된 방식이지만, 작업 영역간의 결합도(Coupling) 문제는 여전히 존재한다.

3)Class(Inheritance)

```
public class Person{
    public String printBirthDate(String format){
        DateUtility.toStringToday(birthDate, format);
    }
}
```

-Person을 상속받은 모든 Class들은 자동적으로 변경된 printBirthDate() 메소드를 사용하게 된다.

-DateUtility class의 메소드가 변경되더라도 printBirthDate() 메소드의 인터페이스가 변하지 않으면 나머지 Class들은 영향을 받지 않는다.

-부모Class가 바뀌면 자식Class를 변경해야 한다.

4)AOP(Aspect Oriented Programming)

-관심의 분리(Seperation of Concerns)

-AOP는 OOP를 더욱 OOP 답게 만들어 줄 수 있다.

-AOP는 OOP 뿐만 아니라 기존의 절차적 프로그래밍에도 적용될 수 있다.

-AOP가 핵심관심 모듈의 코드를 직접 건드리지 않고 필요한 기능이 동작하도록 하는데, 위빙(Weaving)이라고 하는 특수한 작업이 필요하다.

-즉, AOP에서 위빙 작업을 통해 핵심모듈 사이 사이에 필요한 횡단 관심 코드가 동작하도록 엮어지게 만든다.

2. Design Pattern과 Framework의 관련성

1)Design Pattern 이란?

-프로그램 개발에서 자주 나타나는 과제를 해결하기 위한 방법 중 하나로, Software 개발과정에서 발견된 Know-How를 축적하여 이름을 붙여 이후에 재사용하기 좋은 형태로 특정 규약을 묶어서 정리한 것.

-이 용어를 Software 개발 영역에서 구체적으로 처음 제시한 곳은, GoF(Gang of Four)라 불리는 네 명의 컴퓨터 과학 연구자들이 쓴 서적 'Design Patterns : Elements of Reusable Object-Oriented Software'(재사용 가능한 객체지향 소프트웨어의 요소-Design Pattern)이다.

2)Design Pattern을 사용하는 이유

-요구사항은 수시로 변경 -> 요구사항 변경에 대한 Source Code 변경을 최소화

-여러 사람이 같이 하는 팀 프로젝트 진행 -> 범용적인 코딩 스타일을 적용

-상황에 따라 인수 인계하는 경우도 발생 -> 직관적인 코드를 사용

3)Framework 란?

-비 기능적(Non-Functional) 요구사항(성능, 보안, 확장성, 안정성 등)을 만족하는 구조와 구현된 기능을 안정적으로 실행하도록 제어해 주는 잘 만들어진 구조의 Library의 덩어리

-Framework는 Application들의 최소한의 공통점을 찾아 하부 구조를 제공함으로써 개발자들로 하여금

시스템의 하부 구조를 구현하는데 들어가는 노력을 절감하게 해 줌.

4)Framework를 사용하는 이유
-비기능적인 요소들을 초기 개발 단계마다 구현해야 하는 불합리함을 극복해준다.
-기능적인 요구사항에 집중할 수 있도록 해준다.
-Design Pattern과 마찬가지로 반복적으로 발견되는 문제를 해결하기 위한 특화된 solution을 제공한다. 예)한글 인코딩
-개발자는 각 개개인의 능력 차이가 큰 직종이고, 따라서 개발자의 구성에 따라 프로젝트의 결과 역시 차이가 크다.
-그래서 이런 상황을 극복하기 위한 코드의 결과물이 Framework이다.
-Framework를 이용한다는 의미는 프로그램의 기본 흐름이나 구조를 정하고, 모든 팀원이 이 구조에 자신의 코드를 추가하는 방식으로 개발하게 된다.
-Framework의 최대 장점은 개발에 필요한 구조를 이미 코드로 만들어 놓았기 때문에, 실력이 부족한 개발자라 하더라도 반쯤 완성한 상태에서 필요한 부분을 조립하는 형태의 개발이 가능하다는 점이다.
-회사의 입장에서 Framework를 사용하면 일정한 품질이 보장되는 결과물을 얻을 수 있고, 개발자의 입장에서는 완성된 구조에 자신에 만든 코드를 개발해서 넣어주는 형태이므로 개발시간을 단축시킬 수 있다.

5)Design Pattern과 Framework의 관련성
-Design Pattern을 Framework의 핵심적인 특징이고, Framework를 사용하는 Application에 그 패턴이 적용된다는 특징을 가지고 있다.
-하지만 Framework는 Design Pattern이 아니다.
-Design Pattern은 Application을 설계할 때 필요한 구조적인 가이드라인이 되어 줄 수는 있지만 구체적으로 구현된 기반코드를 제공하지 않는다.
-Framework는 Design Pattern과 함께 패턴이 적용된 기반 클래스 Library를 제공해서 Framework를 사용하는 구조적인 틀과 구현코드를 함께 제공한다.

6)결론
-개발자는 Framework의 기반코드를 확장하여 사용하면서 자연스럽게 그 Framework에서 사용된 패턴을 적용할 수 있게 된다.

## 3. Framework의 구성요소와 종류

1)IoC(Inversion of Control)
-'제어의 역전' 즉, 인스턴스 생성부터 소멸까지의 인스턴스 생명주기 관리를 개발자가 아닌 Container가 대신 해준다는 뜻임.
-Container 역할을 해주는 Framework에게 제어하는 권한을 넘겨서 개발자의 코드가 신경 써야 할 것을 줄이는 전략이다.
-Framework의 동작원리를 제어흐름이 일반적인 프로그램 흐름과 반대로 동작하므로 IoC라고 설명함.
-Spring Container는 IoC를 지원하며, 메타데이터(XML설정)를 통해 beans를 관리하고 Application의 중요부분을 형성함.
-Spring Container는 관리되는 bean들을 의존성 주입(Dependency Injection)을 통해 IoC를 지원함.

2)Class Library
-Framework는 특정 부분의 기술적인 구현을 Library 형태로 제공한다.
-Class library라는 구성요소는 Framework의 정의 중 하나인 'Semi Complete(반제품)'이다. 라고 해석하게 만들었다.

| 특징 | Framework | Library |
|---|---|---|
| 유저 코드의 작성 | Framework 클래스를 서브 클래싱 해서 작성 | 독립적으로 작성 |
| 호출 흐름 | Framework코드가 유저코드를 호출 | 유저코드가 Library를 호출 |
| 실행흐름 | Framework가 제어 | 유저코드가 제어 |
| 객체의 연동 | 구조 Framework가 정의 | 독자적으로 정의 |

3)결론
-Framework와 Library를 구분하는 방법은 실행제어가 어디서 일어나는가에 달려있다.
-Library는 개발자가 만든 클래스에서 직접 호출하여 사용하므로 실행의 흐름에 대한 제어를 개발자의 코드가 관장하고 있다.
-Framework는 반대로 Framework에서 개발자가 만든 클래스를 호출하여 실행의 흐름에 대한 제어를 담당한다.

4)Design Pattern
-Design Pattern + Library = Framework
-Framework는 Design Pattern과 그것이 적용된 기반 Library의 결합으로 이해할 수 있다.
-Framework의 Library를 살펴볼 때도 적용된 패턴을 주목해서 살펴 본다면 그 구성을 이해하기 쉽다.
-특히 Framework를 확장하거나 커스터마이징 할 때는 Framework에 적용된 패턴에 대한 이해가 꼭 필요하다.

5)Framework 종류
-아키텍처 결정 = 사용하는 Framework의 종류 + 사용전략
-Web(MVC) : Spring MVC, Struts2, Webwork, PlayFramework
-OR(Object-Relational) Mapping : MyBatis, Hibernate, JPA, Spring JDBC
-AOP(Aspect Oriented Programming) : Spring AOP, AspectJ, JBoss AOP
-DI(Dependency Injection) : Spring DI, Google Guice
-Build와 Library의 관리 : Ant + Ivy, Maven, Gradle
-단위 테스트 : jUnit, TestNG, Cactus
-JavaScript : jQuery, AngularJS, Node.js

4. Lab을 위한 환경설정
1)JDK 1.8.x
2)STS : http://spring.io/tools/sts
3)Spring Source Site : http://spring.io/projects
4)Spring Documentation : https://docs.spring.io/spring/docs/
5)Spring Framework API : 4.3.12(https://docs.spring.io/spring/docs/4.3.12.RELEASE/javadoc-api/)
6)Tomcat 8.x
7)DB : Oracle, MySQL(MariaDB), H2(http://h2database.com/html/main.html)

5. Spring의 주요 모듈
1)Spring Framework
-Spring을 이용해서 어플리케이션을 개발할 때 기반이 되는 Framework.
-Spring의 핵심 기능인 DI와 AOP 기능을 제공.
-Web Application을 개발할 때 사용하는 Spring MVC, Spring ORM 등의 기능도 포함

2)Spring Data
-데이터 연동을 위한 단일 API 제공
-JPA, MongoDB, Neo4j, Redis 등 RDBMS와 NOSQL 과의 연동을 적은 양의 코드로 처리할 수 있도록 해준다.

3)Spring Security
-인증과 허가에 대한 기반 프레임워크 및 관련 모듈을 제공
-웹 어플리케이션을 위한 보안을 간단한 설정과 약간의 코드 구현으로 처리할 수 있다.

4)Spring Batch
-배치 처리를 위한 기반 프레임워크를 제공
-데이터 처리, 흐름 제어, 실패 재처리 등 배치 처리 어플리케이션이 필요로 하는 기능을 기본으로 제공

5)Spring Integration
-시스템 간의 연동을 위한 메시징 프레임워크를 제공

6)Spring Social
-Twitter, Facebook 등 소셜 네트워크 연동을 위한 기능을 제공

6. Spring Framework 이란?
-Enterprise application 에서 필요로 하는 기능을 제공하는 Framework 이다.
-J2EE를 대체하는 Framework이다.
-Java Enterprise 개발을 편하게 해주는 오픈소스 경량급 Application Framework이다.
-Application Framework : 특정 계층이나 기술, 업무, 분야에 국한되지 않고 Application의 전 영역을 포괄하는 범용적인 Framework를 말한다.
-경량급 Framework : 단순한 Web Container(예.tomcat)에서도 엔터프라이즈 개발의 고급기술을 대부분 사용할 수 있다.
-엔터프라이즈 개발 용이 : 개발자가 복잡하고 실수하기 쉬운 low level(보안, 인증, 트랜잭션)에 많이 신경쓰지 않으면서 Business logic 개발에 전념할 수 있도록 해준다.
-오픈소스 : Spring은 오픈소스의 장점을 충분히 취하면서 동시에 오픈소스 제품의 단점과 한계를 잘 극복한다.
-과거에 나왔던 다른 Framework들과 Spring과의 차별점
--복잡함에 반기를 들어서 만들어진 Framework
---Spring은 그 태생 자체가 엔터프라이즈급의 시스템이 실패하는 이유를 복잡성으로 보고, 복잡성을 해결하기 위해서 나온 경량화된 Framework이다.
---일반적인 Java의 클래스와 인터페이스를 이용하는 구조를 사용하기 때문에 진입 장벽이 높지 않았고, EJB에 비해 가볍기 때문에 빠른 시간에 엔터프라이즈급의 시스템을 작성할 수 있었다.
--프로젝트의 전체 구조를 설계할 때 유용한 Framework

---다른 Framework들은 Web 영역이나 데이터베이스 영역등의 전문적인 영역에 대해서만 지원하는 경우가 많았고, 비즈니스 로직을 처리하는 부분에 대한 설계는 개발자의 역량에 맡기는 경우가 많았다.
---반면 Spring은 어느 한 분야에 집중하지 않고, 전체를 설계하는 용도로 사용될 수 있었다.
--다른 Framework들의 포용
---Spring은 전체 구조에 집중했기 때문에 특정한 영역의 Framework와 공존하는 방식을 사용할 수 있었다.
---다른 Framework들은 특정 Framework를 채택하면 해당 영역 전체를 수정해야 하는 고질적인 문제를 가지고 있었지만, Spring은 다른 Framework들과의 통합을 지원했기 때문에 최소한의 수정이 가능했다.
---Spring의 최대 장점은 기본 뼈대를 흔들지 않고, 여러 종류의 Framework를 혼용해서 사용할 수 있다는 점이다.
--개발 생산성과 개발 도구의 지원
---Spring의 경우 이론적으로는 개발자가 제대로 이해애햐 하는 부분이 많지만, 결과적으로 코드의 양은 확실히 줄어들 수 있었고, 유지보수에 있어서도 XML의 설정 등을 이용했기 때문에 환영받을 수 있었다.
---STS나 Eclipse, IntelliJ등의 Plug-in의 지원 역시 다른 Framework들에 비해서 빠른 업데이트가 되었기 때문에 별도의 새로운 개발 도구에 대한 적응 없이도 개발이 가능했다.

## 7. Spring의 개요와 특징

1) EJB 기반으로 개발하지 않는다. POJO 기반으로 개발한다.
2) 가볍다.
3) 제어가 가능하다.
4) 상호 관련이 적다.
5) AOP(Aspect Oriented Programming) 를 지원한다.
- 트랙잭션이나 로깅, 보안과 같이 여러 모듈에서 공통으로 필요로 하지만 실제 모듈의 핵심은 아닌 기능들을 분리해서 각 모듈에 적용할 수 있다.
6) Container를 통한 라이프사이클을 관리하고, Container로부터 필요한 객체를 가져와 사용할 수 있다.
7) XML 기반으로 컴포넌트를 개발할 수 있도록 지원해 준다.
8) 객체의 라이프사이클을 관리하기 위하여 DI 를 사용하는 경량 Container이다.
9) DI(Dependency Injection) 패턴을 지원한다.
- Spring 은 설정 파일을 통해서 객체 간의 의존 관계를 설정할 수 있도록 한다.
- 따라서 객체는 직접 의존하고 있는 객체를 생성하거나 검색할 필요가 없다.
10) 영속성과 관련된 다양한 API 를 지원한다.
- JDBC, iBatis, Hibernate, JPA ...

## 8. Spring Version Up

1)Spring 2.5 : Annotation을 활용하는 설정을 도입하면서 편리한 설정과 개발이 가능하도록 함.
2)Spring 3.0 : 별도의 설정 없이도 Java Class만으로 설정 파일을 대신할 수 있게 함.
3)Spring 4.0 : Mobile 환경과 Web 환경에서 많이 사용되는 REST 방식의 Controller 지원됨.

## 9. Spring Framework 전략

1)Spring 삼각형
2)엔터프라이즈 개발의 복잡함을 상대하는 Spring의 전략
3)Portable Serivce Abstraction, DI, AOP(삼각형의 3면), POJO(삼각형의 중심)

## 10. Portable Serivce Abstraction

-트랜잭션 추상화, OXM 추상화, 데이터 액세스의 Exception 변환기능 등 기술적인 복잡함은 추상화를 통해 low level의 기술 구현부분과 기술을 사용하는 인터페이스로 분리한다.

## 11. 객체지향과 DI

-Spring은 객체지향에 충실한 설계가 가능하도록 단순한 객체 형태로 개발할 수 있고, DI는 유연하게 확장 가능한 객체를 만들어 두고 그 관계는 외부에서 다이나믹하게 설정해 준다.

## 12. AOP

-Application 로직을 담당하는 코드에 남아있는 기술 관련 코드를 분리해서 별도의 모듈로 관리하게 해 주는 강력한 기술이다.

## 13. POJO

-객체지향 원리에 충실하면서 특정 환경이나 규약에 종속되지 않고 필요에 따라 재활용될 수 있는 방식으로 설계된 객체이다.

14. Spring 구조
1)Spring Framework를 구성하는 기능 요소
https://www.slideshare.net/pmanvi/spring-framework-overview-ppt-3068245 중 12번째 슬라이드 참조
-Spring Core
--Spring Framework의 기본 기능을 제공
--이 모듈에 있는 BeanFactory는 Spring의 기본 Container이면서 Spring DI의 기반이다.
-Spring AOP
--이 모듈을 통해 Aspect 지향 프로그래밍을 지원
--이 모듈은 Spring Application에서 Aspect를 개발할 수 있도록 지원
-Spring ORM
--MyBatis, Hibernate, JPA 등 널리 사용되는 ORM Framework와의 연결고리를 지원
--ORM 제품들을 Spring의 기능과 조합해서 사용할 수 있도록 해준다.
-Spring DAO
--JDBC에 대한 추상화 계층으로 JDBC 코딩이나 예외처리 하는 부분을 간편화시켰으며, AOP 모듈을 이용해 트랜잭션 관리 서비스도 제공
-Spring Web
--일반적인 WebApplication 개발에 필요한 기본기능을 제공한다.
--WebWork나 Struts와 같은 다른 Web Application Framework와의 통합을 지원
-Spring Context
--BeanFactory의 개념을 확장한 것으롤  국제화(i18N) 메시지, Application 생명주기 이벤트, 유효성 검증 등을 지원
-Spring Web MVC
--사용자 인터페이스가 Application 로직과 분리되는 Web Application을 만드는 경우에 일반적으로 사용되는 패러다임이다.

15. Maven이란?
1)http://maven.apachde.org Library 관리 + 빌드 툴
-프로젝트의 전체적인 라이프 사이클을 관리하는 도구이며, 많은 편리함과 이점이 있어 널리 사용되고 있다.
-기존에는 Ant가 많이 사용되었지만 Maven이 Ant를 넘어서 더 많은 개발자들이 사용하게 되었고 비교적 최근에는 Gradle이 새롭게 나와 사용되고 있다(대표적으로는 Android Studio).
-프로젝트 구조와 내용을 기술하는 선언적 접근방식의 오픈소스 빌드 툴 이다.
-컴파일과 동시에 빌드를 수행할 수 있으며 테스트를 병행하거나 서버측 디플로이 자원을 관리할 수 있는 환경을 제공한다.
-하지만 아무래도 개발자들에게 가장 큰 장점은 프로젝트의 종속 라이브러리들과 그 라이브러리에 영향을 미치는 Dependency 자원까지 관리 할 수 있다는 점일 것 같다.
-즉, jar 파일을 다운받아 프로젝트에 추가할 경우 그것과 연관된 다른 종속 라이브러리 또한 다 찾아야 하는 불편함을 Maven을 통해서 일관성 있는 라이브러리간의 의존관계 (의존성) 관리를 할 수 있다는 점이다.
-이는 단순히 라이브러리 뿐 아니라 프로젝트별 모듈의 의존성 또한 관리가 된다는 뜻이기도 하다.
-메이븐은 프로젝트 전반의 리소스 관리와 Configuration 파일, Doc 생성 및 이와 관련한 표준 디렉터리 구조를 처음부터 일관된 형태로 구성하여 진행하기 때문에 프로젝트 관리 및 배포 역할을 하는 다른 툴들과의 연계에서도 뛰어난 유연성을 보여준다.
-단점은, 버전별로 이클립스에서 구동되는 방식이 약간 호환성이 떨어진다는 점인데, 사용된 플러그인의 문제인지 메이븐 자체의 하위 호환성 문제인지는 모르겠다. 가끔 저장소 접근에 관한 문제도 발생한다고 알려져 있다.
-메이븐을 학습하기 위해서는 메이븐을 설치하고 POM (Project Object Model) 을 작성한 후 각종 빌드 스크립트 혹은 명령어를 통해 배워나가야 하지만 여기서는 기본적인 사용법과 더불어 STS를 통해 어플리케이션을 작성 한 후 어떻게 이클립스상에서 메이븐을 활용하는지에 초점을 맞춰 진행하도록 하겠다.

2)Maven을 사용하는 이유
-편리한 Dependent Library 관리 - Dependency Management
-의존성(dependency)
--라이브러리 다운로드 자동화
---더 이상 필요한 (의존성 있는) 라이브러리를 하나씩 다운로드 받을 필요가 없다. 필요하다고 선언만 하면 메이븐이 자동으로 다운로드 받아준다.
--메이븐은 선언적 (명령식이 아니다) :
---사용되는 jar 파일들을 어디서 다운로드 받고, 어느 릴리즈(버전)인지 명시하면, 코딩을 하지 않아도 메이븐이 알아서 관리한다. (재 다운로드, 최신 버전 설치 등)
--메이븐이 관리한다.
---라이브러리 (lib) 디렉터리를 생성할 필요가 없다.
---이클립스 내에서 라이브러리, 클래스패스 환경 설정을 할 필요도 없다
----여러 프로젝트에서 프로젝트 정보나 jar파일들을 공유하기 쉬움
----모든 프로젝트의 빌드 프로세스를 일관되게 가져갈 수 있음.

3)Maven 이전의 Library 관리 방법

-Library 사이트 접속 -> Library 다운로드 -> 압축 해제 -> 프로젝트에 Library 복사 -> 클래스패스에 추가 -> 다시 Library 사이트 접속

4)Maven의 Library 관리 방법
-pom.xml 파일 수정 -> 빌드 -> pom.xml 파일 수정

5)pom.xml
-Maven 프로젝트를 생성하면 pom.xml 파일이 생성
-pom.xml 파일은 Project Object Model 정보를 담고 있다.
--프로젝트 정보 : 프로젝트의 이름, 개발자 목록, 라이센스 등
--빌드 설정 : 소스, 리소스, 라이프 사이클별 실행한 플러그인(goal)등 빌드와 관련된 설정
--빌드 환경 : 사용자 환경 별로 달라질 수 있는 프로파일 정보
--POM 연관 정보 : 의존 프로젝트(모듈), 상위 프로젝트, 포함하고 있는 하위 모듈 등
-프로젝트 당 하나의 pom.xml
--각각의 프로젝트는 pom.xml 파일을 하나씩 가진다.
--pom은 프로젝트 자체와 의존성에 대한 설정 및 정보를 포함한다.
--메이븐을 pom.xml 을 읽어, 프로젝트를 가공하는 방법을 이해한다.
-3 가지 "coordinates"를 이용해 자원을 식별한다.
--그룹 ID (Group ID) :
--아티펙트 ID (Artifact ID) :
--버전 (Version) :
-중요한 속성들
--<artifactId/>
---아티팩트의 명칭 (Artifact's name), groupId 범위 내에서 유일해야 한다.
--<groupId/>
---일반적으로 프로젝트의 패키지 명칭
--<version/>
---아티팩트(artifact)의 현재 버전
--<name/>
---어플리케이션의 명칭
--<packaging/>
---아티팩트(artifact) 패키징 유형
---POM, jar, WAR, EAR, EJB, bundle, ... 중에서 선택 가능
--<distributionManagement/>
---아티팩트(artifact)가 배포될 저장소 정보와 설정
--<parent/>
---프로젝트의 계층 정보
--<version/>
---아티팩트(artifact)의 현재 버전
--<scm/>
---소스 코드 관리 시스템 정보
--<dependencyManagement/>
---의존성 처리에 대한 기본 설정 영역
--<dependencies/>
---의존성 정의 및 설정 영역

6)Maven 설치
-http://maven.apache.org/
-Refer to : http://javacan.tistory.com/entry/MavenBasic
-Download
--http://maven.apache.org/download.cgi
--Binary zip archive : apache-maven-3.5.2-bin.zip
-OS에 M2_HOME(C:\Program Files\apache-maven-3.5.0)과 PATH(...\bin) 추가
-In Command Window, set M2_HOME, set PATH

-설치 확인

```
$ mvn -version
Apache Maven 3.5.2 (138edd61fd100ec658bfa2d307c43b76940a5d7d;
2017-10-18T16:58:13+09:00)
Maven home: C:\Program Files\apache-maven-3.5.2\bin\..
Java version: 1.8.0_152, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.8.0_152\jre
Default locale: en_US, platform encoding: MS949
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

-Maven 프로젝트 생성하기

--설치가 끝났다면 Maven 프로젝트를 생성해 보자. 명령 프롬프트에서 아래 명령어를 실행하면 된다. (아래 명령어를 처음 실행할 경우 꽤 오랜 시간이 걸리는데, 그 이유는 Maven이 필요한 플러그인과 모듈을 다운로드 받기 때문이다. Maven 배포판은 최초로 Maven을 사용하는 데 필요한 모듈만 포함하고 있고, 그 외에 archetype 플러그인, compiler 플러그인 등 Maven을 사용하는 데 필요한 모듈은 포함하고 있지 않다. 이들 모듈은 실제로 필요할 때 Maven 중앙 리포지토리에서 로딩된다.)
--먼저 C:\SpringHome 폴더를 생성한 후, 폴더로 이동한다.

--$ mvn archetype:generate
...
Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): 1085: 엔터
Choose org.apache.maven.archetypes:maven-archetype-quickstart version:
1: 1.0-alpha-1
2: 1.0-alpha-2
3: 1.0-alpha-3
4: 1.0-alpha-4
5: 1.0
6: 1.1
Choose a number: 6: 엔터
...
Define value for property 'groupId': com.javasoft
Define value for property 'artifactId': demo
Define value for property 'version' 1.0-SNAPSHOT: : 엔터
Define value for property 'package' com.javasoft: : 엔터
Confirm properties configuration:
groupId: com.javasoft
artifactId: demo
version: 1.0-SNAPSHOT
package: com.javasoft
Y: : 엔터
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.1
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: basedir, Value: C:\SpringHome
[INFO] Parameter: package, Value: com.javasoft
[INFO] Parameter: groupId, Value: com.javasoft
[INFO] Parameter: artifactId, Value: demo
[INFO] Parameter: packageName, Value: com.javasoft
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\SpringHome\demo
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 06:55 min
[INFO] Finished at: 2017-11-26T08:27:10+09:00
[INFO] Final Memory: 15M/132M
[INFO] ------------------------------------------------------------------------

위 과정에서 실제로 입력하는 값은 다음과 같다.
--groupId - 프로젝트 속하는 그룹 식별 값. 회사, 본부, 또는 단체를 의미하는 값이 오며, 패키지 형식으로 계층을 표현한다. 위에서는 com.javasoft를 groupId로 이용하였다.
--artifactId - 프로젝트 결과물의 식별 값. 프로젝트나 모듈을 의미하는 값이 온다. 위에서는 demo을 artifactId로 이용하였다.
--version - 결과물의 버전을 입력한다. 위에서는 기본 값인 1.0-SNAPSHOT을 사용하였다.
--package - 기본적으로 생성할 패키지를 입력한다. 별도로 입력하지 않을 경우 groupId와 동일한 구조의 패키지를 생성한다.

-Maven 프로젝트의 기본 디렉토리 구조
--archetype:generate이 성공적으로 실행되면, artifactId에 입력한 값과 동일한 이름의 디렉터리가 생성된다.
--archetype:generate 명령어는 미리 정의된 템플릿을 이용해서 메이븐 프로젝트를 생성하는 기능으로 지금 사용한 Maven Project Template는 archetype-quickstart라는 템플릿이다.
--템플릿을 사용하지 않고 직접 디렉토리를 생성하고 pom.xml 파일을 작성해 주어도 동일한 메이븐 프로젝트가 생성된다.
--위 경우에는 현재 디렉터리에 demo 이라는 하위 디렉터리가 생성된다.
--위 과정에서 선택한 archetype은 maven-archetype-quickstart 인데, 이 archetype을 선택했을 때

생성되는 디렉터리 구조는 다음과 같다.
src/main/java - 자바 소스 파일이 위치한다.
src/main/resources - 프로퍼티나 XML 등 리소스 파일이 위치한다. 클래스패스에 포함된다.(생성예정)
src/main/webapp - 웹 어플리케이션 관련 파일이 위치한다. (WEB-INF 디렉터리, JSP 파일 등, 생성예정)
src/test/java - 테스트 자바 소스 파일이 위치한다.
src/test/resources - 테스트 과정에서 사용되는 리소스 파일이 위치한다. 테스트 시에 사용되는 클래스패스에 포함된다.
--기본적으로 생성되지 않은 디렉터리라 하더라도 직접 생성해주면 된다. 예를 들어, src/main 디렉터리에 resources 디렉터리를 생성해주면 Maven은 리소스 디렉터리로 인식한다.

-컴파일 해보기/테스트 실행 해보기/패키지 해보기
--이제 간단하게 컴파일과 테스트를 실행해보자.
--소스 코드를 컴파일 하려면 다음과 같은 명령어를 실행해주면 된다.
--demo폴더로 이동 후

--$ mvn compile
...
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\SpringHome\demo\target\classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 49.782 s
[INFO] Finished at: 2017-11-26T08:44:43+09:00
[INFO] Final Memory: 16M/133M
[INFO] ------------------------------------------------------------------------

--컴파일 된 결과는 target/classes 디렉터리에 생성된다.
--테스트 클래스를 실행해보고 싶다면, 다음과 같은 명령어를 사용하면 된다.

--$ mvn test

-------------------------------------------------------
T E S T S
-------------------------------------------------------
Running com.javasoft.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.006 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 13.608 s
[INFO] Finished at: 2017-11-26T08:46:57+09:00
[INFO] Final Memory: 18M/166M
[INFO] ------------------------------------------------------------------------

--그러면, 테스트 코드를 컴파일한 뒤 테스트 코드를 실행한다.
--그리고 테스트 성공 실패 여부를 화면에 출력한다.
--컴파일 된 테스트 클래스들은 target/test-classes 디렉터리에 생성되고, 테스트 결과 리포트는 target/surefire-reports 디렉터리에 저장된다.
--(아무것도 한게 없으니 당연하지만) 모든 코드가 정상적으로 만들어지고 테스트도 통과했으니, 이제 배포 가능한 jar 파일을 만들어보자.
--아래 명령어를 실행하면 프로젝트를 패키징해서 결과물을 생성한다.

--$ mvn package
[INFO] Building jar: C:\SpringHome\demo\target\demo-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 8.334 s
[INFO] Finished at: 2017-11-26T08:48:37+09:00
[INFO] Final Memory: 14M/169M

[INFO] ------------------------------------------------------------------------

--mvn package가 성공적으로 실행되면, target 디렉터리에 프로젝트 이름과 버전에 따라 알맞은 이름을 갖는 jar 파일이 생성된다.
--위 예제의 경우에는 demo-1.0-SNAPSHOT.jar 파일이 생성된 것을 확인할 수 있다.

-POM 파일 기본
--Maven 프로젝트를 생성하면 pom.xml 파일이 프로젝트 루트 디렉터리에 생성된다.
--이 pom.xml 파일은 Project Object Model 정보를 담고 있는 파일로서, 이 파일에서 다루는 주요 설정 정보는 다음과 같다.
---프로젝트 정보 - 프로젝트의 이름, 개발자 목록, 라이센스 등의 정보를 기술
---빌드 설정 - 소스, 리소스, 라이프 사이클 별 실행할 플러그인 등 빌드와 관련된 설정을 기술
---빌드 환경 - 사용자 환경 별로 달라질 수 있는 프로파일 정보를 기술
---POM 연관 정보 - 의존 프로젝트(모듈), 상위 프로젝트, 포함하고 있는 하위 모듈 등을 기술
--archetype:create 골 실행시 maven-archetype-quickstart Archetype을 선택한 경우 생성되는 pom.xml 파일은 다음과 같다.

[기본으로 생성되는 pom.xml 파일]

```
<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.javasoft</groupId>
  <artifactId>demo</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>demo</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

--위 POM 파일에서 프로젝트 정보를 기술하는 태그는 다음과 같다.
<name> - 프로젝트 이름
<url> - 프로젝트 사이트 URL
--POM 연관 정보는 프로젝트간 연관 정보를 기술하는데, 관련 태그는 다음과 같다.
<groupId> - 프로젝트의 그룹 ID 설정
<artifactId> - 프로젝트의 Artifact ID 설정
<version> - 버전 설정
<packaging> - 패키징 타입 설정. 위 코드의 경우 프로젝트의 결과 Artifact가 jar 파일로 생성됨을 의미한다. jar 뿐만 아니라 웹 어플리케이션을 위한 war나 JEE를 위한 ear 등의 패키징 타입이 존재한다.
<dependencies> - 이 프로젝트에서 의존하는 다른 프로젝트 정보를 기술한다.
<dependency> - 의존하는 프로젝트 POM 정보를 기술
<groupId> - 의존하는 프로젝트의 그룹 ID
<artifactId> - 의존하는 프로젝트의 artifact ID
<version> - 의존하는 프로젝트의 버전
<scope> - 의존하는 범위를 설정

-의존설정
--<dependency> 부분의 설정에 대해서 좀 더 살펴보도록 하자.
--Maven을 사용하지 않을 경우 개발자들은 코드에서 필요로 하는 라이브러리를 각각 다운로드 받아야 한다.

--예를 들어, 아파치 commons DBCP 라이브러리를 사용하기 위해서는 DBCP 뿐만 아니라 common pool 라이브러리도 다운로드 받아야 한다.
--물론, commons logging을 비롯한 라이브러리도 모두 추가로 다운로드 받아 설치해 주어야 한다.
--즉, 코드에서 필요로 하는 라이브러리 뿐만 아니라 그 라이브러리가 필요로 하는 또 다른 라이브러리도 직접 찾아서 설치해 주어야 한다.
--하지만, Maven을 사용할 경우에는 코드에서 직접적으로 사용하는 모듈에 대한 의존만 추가해주면 된다.
--예를 들어, commons-dbcp 모듈을 사용하고 싶은 경우 다음과 같은 <dependency> 코드만 추가해주면 된다.

```
<dependency>
    <groupId>commons-dbcp</groupId>
    <artifactId>commons-dbcp</artifactId>
    <version>1.2.1</version>
</dependency>
```

--그러면, Maven은 commons-dbcp 뿐만 아니라 commons-dbcp가 의존하는 라이브러리를 자동으로 처리해준다.
--Maven은 commons-dbcp 모듈을 다운로드 받을 때 관련 POM 파일도 함께 다운로드 받는다.
--실제로 1.2.1 버전의 commons-dbcp 모듈의 pom.xml 파일을 보면 의존 부분이 다음과 같이 설정되어 확인할 수 있다.

```
<dependencies>
    <dependency>
      <groupId>commons-collections</groupId>
      <artifactId>commons-collections</artifactId>
      <version>2.1</version>
    </dependency>
    <dependency>
      <groupId>commons-pool</groupId>
      <artifactId>commons-pool</artifactId>
      <version>1.2</version>
    </dependency>
    <dependency>
      <groupId>javax.sql</groupId>
      <artifactId>jdbc-stdext</artifactId>
      <version>2.0</version>
      <optional>true</optional>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>xml-apis</groupId>
      <artifactId>xml-apis</artifactId>
      <version>2.0.2</version>
    </dependency>
    <dependency>
      <groupId>xerces</groupId>
      <artifactId>xerces</artifactId>
      <version>2.0.2</version>
    </dependency>
  </dependencies>
```

--Maven은 commons-dbcp 모듈을 다운로드 받을 때 관련 POM 파일도 함께 다운로드 받는다.
--그리고 POM 파일에 명시한 의존 모듈을 함께 다운로드 받는다.
--즉, commons-dbcp 1.2.1 버전의 경우 commons-collections 2.1 버전과 commons-pool 1.2 버전 등을 함께 다운로드 받는다.
--이런 식으로 반복해서 다운로드 받은 모듈이 필요로 하는 모듈을 다운로드 받고 이들 모듈을 현재 프로젝트에서 사용할 클래스패스에 추가해준다.
--따라서, 개발자는 일일이 필요한 모듈을 다운로드 받을 필요가 없으며, 현재 코드에서 직접적으로 필요로 하는 모듈에 대해서만 <dependency>로 추가해주면 된다.
--나머지 의존은 모두 Maven이 알맞게 처리해준다.

-의존의 scope: compile, runtime, provided, test

--앞의 pom.xml 파일에서 <dependency> 부분을 보면 <scope>를 포함하고 있는 것과 그렇지 않은 것이 존재한다는 것을 알 수 있다.
--<scope>는 의존하는 모듈이 언제 사용되는 지를 설정할 때 사용되며, <scope>에 올 수 있는 값은 다음의 네 가지가 존재한다.
---compile - 컴파일 할 때 필요. 테스트 및 런타임에도 클래스패스에 포함된다. <scope>를 설정하지 않을 경우 기본 값은 compile 이다.
---runtime - 런타임에 필요. JDBC 드라이버 등이 예가 된다. 프로젝트의 코드를 컴파일 할 때는 필요하지 않지만, 실행할 때 필요하다는 것을 의미한다. 배포시 포함된다.
---provided - 컴파일 할 때 필요하지만, 실제 런타임 때에는 컨테이너 같은 것에서 기본으로 제공되는 모듈임을 의미한다. 예를 들어, 서블릿이나 JSP API 등이 이에 해당한다. 배포시 제외된다.
---test - 테스트 코드를 컴파일 할 때 필요. Mock 테스트를 위한 모듈이 예이다. 테스트 시에 클래스패스에 포함되며, 배포시 제외된다.

-원격 리포지토리와 로컬 리포지토리
--Maven은 컴파일이나 패키징 등 작업을 실행할 때 필요한 플러그인이나 pom.xml 파일의 <dependency> 등에 설정한 모듈을 Maven 중앙 리포지토리에서 다운로드 받는다. 현재 중앙 리포지토리의 주소는 http://mvnrepository.com/ 이다.
--원격 리포지토리에서 다운로드 받은 모듈은 로컬 리포지토리에 저장된다.
--로컬 리포지토리는 [USER_HOME]/.m2/repository 디렉터리에 생성되며, 로컬 리포지토리에는 다음과 같은 형식의 디렉터리를 생성한 뒤 다운로드 받은 모듈을 저장한다.
[groupId]/[artifactId]/[version]
--예를 들어, commons-dbcp 1.2.1 버전의 경우, 모듈 및 관련 POM 파일이 저장되는 디렉터리는 다음과 같다.
[USER_HOME]/.m2/repository/commons-dbcp/commons-dbcp/1.2.1
--위 디렉터리에 저장되는 파일은 패키징 된 모듈 파일, pom 파일, 그리고 소스 코드 다운로드 옵션을 실행한 경우에는 소스 코드를 포함한 jar 파일이 포함된다.
--일단 원격 리포지토리로부터 파일을 다운로드해서 로컬 리포지토리에 저장하면, 그 뒤로는 로컬 리포지토리에 저장된 파일을 사용하게 된다.

-Maven 라이프사이클(Lifecycle)과 플러그인 실행
--본 글의 서두에 Maven은 프로젝트의 라이프사이클 기반 프레임워크를 제공한다고 했다.
--앞서 프로젝트를 생성한 뒤 컴파일하고(mvn compile), 테스트 하고(mvn test), 패키징 하는(mvn package) 과정을 정해진 명령어를 이용해서 실행했는데, 이때 compile, test, package는 모두 빌드 라이프사이클에 속하는 단계이다.
--Maven은 clean, build (default), site의 세 가지 라이프사이클을 제공하고 있다.
--각 라이프사이클은 순서를 갖는 단계(phase)로 구성된다.
--또한, 각 단계별로 기본적으로 실행되는 플러그인(plugin) 골(goal)이 정의되어 있어서 각 단계마다 알맞은 작업이 실행된다.
--아래 표는 디폴트 라이프사이클을 구성하고 있는 주요 실행 단계를 순서대로 정리한 것이다.

[표] 디폴트 라이프사이클의 주요 단계(phase)

| 단계 | 설명 | 단계에 묶인 플러그인 실행 |
|---|---|---|
| generate-sources | 컴파일 과정에 포함될 소스를 생성한다. 예를 들어, DB 테이블과 매핑되는 자바 코드를 생성해주는 작업이 이 단계에서 실행된다. | |
| process-sources | 필터와 같은 작업을 소스 코드에 처리한다. | |
| generate-resources | 패키지에 포함될 자원을 생성한다. | |
| process-resources | 필터와 같은 작업을 자원 파일에 처리하고, 자원 파일을 클래스 출력 디렉토리에 복사한다. | resources:resources |
| compile | 소스 코드를 컴파일해서 클래스 출력 디렉터리에 클래스를 생성한다. | compiler:compile |
| generate-test-sources | 테스트 소스 코드를 생성한다. 예를 들어, 특정 클래스에서 자동으로 테스트 케이스를 만드는 작업이 이 단계에서 실행된다. | |
| process-test-sources | 필터와 같은 작업을 테스트 소스 코드에 처리한다. | resources:testResources |
| generate-test-resources | 테스트를 위한 자원 파일을 생성한다. | |
| process-test-resources | 필터와 같은 작업을 테스트 자원 파일에 처리하고, 테스트 자원 파일을 테스트 클래스 출력 디렉터리에 복사한다. | |
| test-compile | 테스트 소스 코드를 컴파일해서 테스트 클래스 추력 디렉터리에 클래스를 생성한다. | compiler:testCompile |
| test | 테스트를 실행한다. | surefire:test |
| package | 컴파일 된 코드와 자원 파일들을 jar, war와 같은 배포 형식으로 | |

패키징한다. 패키징에 따라 다름

jar - jar:jar

war - war:war

pom - site:attach-descriptor

ejb - ejb:ejb

install 로컬 리포지토리에 패키지를 복사한다. install:install

deploy 생성된 패키지 파일을 원격 리포지토리에 등록하여, 다른 프로젝트에서 사용할 수 있도록 한다. deploy:deploy

--라이프사이클의 특정 단계를 실행하려면 다음과 같이 mvn [단계이름] 명령어를 실행하면 된다.

mvn test

mvn deploy

--라이프사이클의 특정 단계를 실행하면 그 단계의 앞에 위치한 모든 단계가 실행된다.

--예를 들어, test 단계를 실행하면 test 단계를 실행하기에 앞서 'generate-sources' 단계부터 'test-compile' 단계까지 각 단계를 순서대로 실행한다.

--각 단계가 실행될 때는 각 단계에 묶인 골(goal)이 실행된다.

--플러그인을 직접 실행할 수도 있다. mvn 명령어에 단계 대신 실행할 플러그인을 지정하면 된다.

mvn surefire:test

--단, 플러그인 골을 직접 명시한 경우에는 해당 플러그인만 실행되기 때문에 라이프사이클의 단계가 실행되지는 않는다.

-플러그인 골(Plugin Goal)

--Maven에서 플러그인을 실행할 때에는 '플러그인이름:플러그인지원골'의 형식으로 실행할 기능을 선택한다.

--예를 들어, compiler:compile은 'compiler'는 플러그인에서 'compile' 기능(goal)을 실행한다는 것을 뜻한다.

-맺음말

--이번 글에서는 Maven의 기본 사용법을 살펴봤다.

--Maven이 제공하는 의존 관리는 개발자를 jar 지옥(?)에서 구해준다는 것을 알 수 있었다.

--또한, Maven은 표준화된 라이프사이클을 제공하고 있기 때문에 개발자가 컴파일-테스트-패키징 등의 과정을 손으로 정의하지 않아도 되며, 개발자는 Maven이 제공하는 단계 중 필요한 단계만 실행하면 된다.

--그럼, 나머지 작업(컴파일, 테스트 실행, jar 파일 생성)은 모두 Maven이 처리해준다.

-관련자료

--Maven 홈 페이지: http://maven.apache.org

--Maven: The Definitive Guide (Sonatype, Oreilly)

--Maven 컴파일러 버전 설정

출처: http://javacan.tistory.com/entry/MavenBasic [자바캔(Java Can Do IT)]

7)수동으로 maven project 생성 후 해야할 작업

-src/main 디렉토리에 resources 디렉토리를 직접 수동으로 추가

-메이븐의 src/main/resources 디렉토리는 클래스패스(classpath)로 사용되는 디렉토리로서, 이 디렉토리에는 XML이나 properties 파일과 같은 자원 파일 중에서 클래스패스에 위치해야 하는 파일들을 넣는다.

-그리고 src/test/java 디렉토리에 생성된 AppTest.java와 src/main/java 디렉토리에 생성된 App.java 파일을 필요하지 않으므로 삭제한다.

-메이븐 프로젝트가 자바 코드를 컴파일할 때 UTF-8 charset과 Java 1.8 버전을 사용하도록 설정이 필요하다.

-이를 위해 pom.xml에 아래 코드를 추가한다.

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
```

```
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                   <source>1.8</source>
                   <target>1.8</target>
                   <encoding>${project.build.sourceEncoding}</encoding>
                </configuration>
             </plugin>
          </plugins>
       </build>
    </project>

8)Eclipse에서 Maven Project Import 하기
   -File > Import > Maven > Existing Maven Projects > Browse
   -[Root Directory] : C:\SpringHome\demo
   -그러면 자동으로 pom.xml 파일을 기준으로 프로젝트를 찾는다. > Finish

9)pom.xml 의존관계(dependency) 추가
   -https://projects.spring.io/spring-framework/에서 springframework 버전 선택
   -Maven에 Spring Context 추가하기
      --In pom.xml
         <dependencies>
            <dependency>
               <groupId>junit</groupId>
               <artifactId>junit</artifactId>
               <version>3.8.1</version>
               <scope>test</scope>
            </dependency>
            <!--아래 코드 추가-->
            <dependency>
               <groupId>org.springframework</groupId>
               <artifactId>spring-context</artifactId>
               <version>4.3.12.RELEASE</version>
            </dependency>
         </dependencies>
   -pom.xml > right-click > Run As > Maven install
   -Eclipse가 spring-context 모듈을 포함해서 필요한 jar파일들을 다운로드받기 시작한다.
   -필요한 jar 파일들을 모두 다운로드 받으면, Eclipse 프로젝트에 다운로드 받은 jar 파일들이 Maven 의존
   목록(Maven Dependencies)에 표시된다.

10)Maven Repositories View 추가하기
   -Window 메뉴 > Show View > Other > Maven > Maven Repository
   -Local Repository -> C:\Users\...\.m2\repository 에 저장

11)Eclipse에서 직접 Maven Project 생성하기
   -In Eclipse EE,
     --New > Other > Maven > Maven Project > Next
    Next > org.apache.maven.archetypes | maven-archetype-quickstart | 1.1 >
    ---Group Id : com.javasoft
    ---Artifact Id : demo
    ---Version : 0.0.1-SNAPSHOT
    ---Package : com.javasoft.demo
    Finish

   -App.class 실행
      --src/main/java > package com.javasoft.demo > App.java
            package com.javasoft.demo;

         /**
          * Hello world!
          *
          */
         public class App
         {
               public static void main( String[] args )
               {
                     System.out.println( "Hello World!" );
               }
```

```
        }

    --Run As > Java Application
    --Delete App.java
    --Greeter.java 생성
      ---src/main/java > package com.javasoft.demo > New > Create Class Greeter.java

        package com.javasoft.demo;

        public class Greeter {
          private String format;

          public String greet(String guest){
            return String.format(format, guest);
          }

          public void setFormat(String format){
            this.format = format;
          }
        }

-'demo' 프로젝트 > src > main > resources 폴더 Build path 추가하기
    --Click on [Build Path]
    --Click on [Congifure Build Path]
    --Click on [Source] Tab
    --Click on [Add Folder]
    --Select 'main' folder
    --Click [Create New Folder]
  ---Folder name : resources
       ---Finish
    --OK

-demo > src > main > resources > right-click > New > Other > XML > xml file
    file name : applicationContext.xml

      <?xml version="1.0" encoding="UTF-8"?>
      <!--아래 코드는 google에서 'spring conrtext xml'로 검색할 것 -->
      <beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">
        <bean id="greeter" class="com.javasoft.demo.Greeter">
          <property name="format" value="%s, Hello, World!" />
        </bean>
      </beans>

-Maven에 Spring Context 설치하기
    --In pom.xml
      <dependencies>
        <dependency>
          <groupId>junit</groupId>
          <artifactId>junit</artifactId>
          <version>3.8.1</version>
          <scope>test</scope>
        </dependency>
        <!--아래 코드 추가-->
        <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-context</artifactId>
          <version>4.3.12.RELEASE</version>
        </dependency>
      </dependencies>

-pom.xml > Run As > Maven install

-src/main/java > com.javasoft.demo > Main class 생성
```

```
798         package info.javaexpert.demo;
799
800         import org.springframework.context.support.GenericXmlApplicationContext;
801
802         public class Main {
803             public static void main(String [] args){
804                 GenericXmlApplicationContext ctx = new
                    GenericXmlApplicationContext("classpath:applicationContext.xml");
805                 Greeter g = ctx.getBean("greeter", Greeter.class);
806                 String msg = g.greet("Spring");
807                 System.out.println(msg);
808                 ctx.close();
809             }
810         }
811
812       -Main.java 실행하기
```