Enterprise Java Beans #1

작성자 : 김성박(삼성 SDS 멀티캠퍼스 전임강사)

email : urstory@nownuri.net

homepage : http://sunny.sarang.net

- 잘못된 부분을 발견하였거나, 추가할 사항이 있다면 작성자에게 메일 보내주신다면 감사하겠습니다.
- 해당 문서는 JAVA 기본, JSP, Servlet, RMI 등의 기술에 대하여 알고 있는 분을 대상으로 합니다.

1. EJB란 무엇인가?

EJB는 Sun Microsystem사(이하 SUN)에서 제안한, 서버 컴퓨터에서 실행되는 자바 컴포넌트 모델이며, EJB는 ORACLE, IBM 등의 여러회사와 수 많은 개발자들이 참여하여 만드는 공통된 표준규칙을 말합니다.

EJB를 공부하고자 하는 분들에게는 이렇게 말합니다. EJB는 자바의 종합예술과 같은 분야이다. 그러므로 무턱대고 EJB에 도전하다가는 좌절할 수 밖에 없다고 말합니다. 그 이유는 EJB에는 수많은 자바 기술이 들어가 있기 때문입니다.

보통 우리가 말하는 EJB프로그래밍이라는 것은 사실, EJB 프로그래밍만을 의미하지 않습니다. "무슨소리지요?" 라고 물어볼 분을 위하여 설명하자면, SUN에서는 기업용 에플리케이션을 개발하기 위한 API의 묶음인 J2EE를 발표합니다.

J2EE를 구성하고 있는 것들을 보면, 핵심으로 JSP, 서블릿, EJB(Enterprise Java Beans)등이 있으며, 데이터베이스를 위한 JDBC, 디렉토리를 위한 JNDI(Java Naming and Directory Interface), 트랜잭션을 위한 JTA(Java Transaction API), 메시징을 위한 JMS(Java Message Service), 이 메일 시스템을 위한 JavaMail 그리고 CORBA접속을 위한 Java IDL(Interface Definition Language)로 구성되는 것을 알 수 있습니다.

이렇게 여러가지 API로 구성된 J2EE의 가장 중요한 기능은 비즈니스 로직을 담당하고 있는 EJB에 있다고 말해도 과언이 아닙니다. 이렇게 EJB 기능을 포함하고 있는 J2EE와 같은 API의 묶음을 EJB 컨테이너라고도 말을 합니다.

J2EE 자체만 보아도 훌륭한 EJB 컨테이너지만, 썬사에서는 J2EE를 상업적 용도로 사용못하도록 라이센스에 표시해 놓았습니다. 그러므로, 실제로 상용으로 이용 할려면 J2EE가 아닌 다른 EJB 컨테이너를 이용해야 합니다.

EJB는 JAVA로 작성되어지기 때문에, J2EE에서 작성된 프로그램은 수정하지 않고 다른 EJB 컨테이너에서 동작할 수 있습니다. 그렇게 될 수 있는 이유는 여러 종류의 EJB컨테이너가 모두 EJB 스펙을 준수하고 있기 때문입니다.

J2EE 를 구성하고 있는 기술들은 각각 하나씩 보아도 초보자들 입장으로는 어려울 것입니다. 또한 J2EE의 스펙 자체가 복잡하기 때문에, J2EE를 어렵다고 말을 합니다. 하지만 반대로 이러한 스펙을 잘 알고 있는 사람이라면, 그 스펙대로 프로그래밍을 작성하면 쉽게 프로그래밍을 짤 수도 있다는 말이기도 합니다. 그러므로 J2EE를 쉽게 익히기 위하여는 스펙이 어떻게 되는지 정확하게 이해햐여야 하는 것입니다.

그런데, 여기에서 스펙을 이해할 수 있으려면 자바의 기본지식이 필요합니다. 자바의 기본지식으로는 JAVA 기초, JSP, Servlet, RMI등에 대하여 알고, 객체지향에 대한 확실한 이해가 있을 경우 J2EE를 이해할 수 있을 것입니다. 특히 JAVA기초부분에서 Interface의 기능과 특징에 대하여 자유자재로 쓸 수 있어야만 됩니다. 그러한 내용은 본 문서에서 다루지 않겠습니다.

참고 : EJB 와 JAVA Bean은 모두 class이긴 하지만 개념자체는 틀립니다. JAVA Beans는 보통 비쥬얼한 개발환경에서 재사용되기 위한 컴포넌트 모델이며, EJB는 미션 크리티컬한 서버쪽에서 기업용 애플리케이션에서 사용될 수 있는 분산객체를 말합니다. (좀더 여러가지로 비교할 수 있지만 생략합니다.)

2. EJB를 왜 사용하는가?

EJB를 사용하는 이유는 여러가지 장점이 있기 때문일 것입니다. EJB는 작은 작업을 처리할 때 사용하지 않습니다. 작은 작업을 웹상으로 구현 할려면 jsp나 servlet 만 이용하고도 쉽게 작성할 수 있습니다. EJB는 중요한 기능을 처리하는 기업에서 사용되도록 작성되는 것입니다.

a. 비즈니스 관점으로 본 장점

- 빠른 개발 속도 : EJB가 동작하는 EJB컨테이너는 실제로 개발자가 비즈니스 관련 프로그램을 작성하는데 가장 중요한 트랜잭션, 보안, 패일오버등을 자동으로 처리해 주기 때문에 오류가 발생할 가능성이 적어집니다. 그러므로 빠른 시간안에 개발될 수 있습니다
- 작업할 내용의 감소 : 앞에서 말한 장점과 같은 의미로, 개발자가 직접 작성해야 될 내용이 이미 구현되어 있으므로, 직접 모두 구현하는 것보다 작업할 내용이 감소 될 수 있습니다.
- 코드 재사용성 증가 : EJB 자체가 비즈니스 단위로 재사용 되도록 만들어지기 때문에 코드 자체의 재사용률이 높아집니다.
- 전문가의 필요성 감소 : 실제 가장 작성하기 어려운 부분인 트랜잭션, 보안, 패일오보등의 기능을 자동으로 처리해주기 때문에 스펙을 확실하게 이해하고 있다면 전문개발자가 아니라도 쉽게 개발할 수 있습니다.

b. 개발자 관점의 장점

A. 컴포넌트 아키텍처 도입
B. 트랜잭션 보장
C. 이식성
D. 확장성
E. 유연성

위와 같은 장점이 있으나 단점은 EJB 컨테이너를 구입해야 하는 비용적인 부분과 스펙 자체가 어렵다는 것이 있습니다.

3. EJB 아키텍처의 목표

EJB는 다음과 같은 목표로 만들어 졌습니다. EJB가 어떤 목표를 가지고 있는지 확실하게 파악하고 있어야, 알맞게 EJB를 이용할 수 있을 것입니다.

- JAVA 언어로 작성되는 분산 객체 형태의 비즈니스 로직을 구현하는 애플리케이션을 개발하기 위한 표준 컴포넌트이다.
- 한가지 툴에 종속적인 것이 아니라, 여러가지 툴들을 이용할 수 있으며 툴들을 통하여 컴포넌트를 조합하여 애플리케이션을 작성할 수 있다.
- 트랜잭션, 보안, 패일오버, 멀티쓰레드, 커넥션풀링 등과 같은 복잡하고 어려운 기능들을 개발자가 몰라도 쉽게 작성할 수 있다.
- EJB는 한번 작성된 후 재 컴파일 없이 재사용 할 수 있다.

- EJB 컴포넌트를 개발하는 단계와 EJB컨테이너에 배치하고 실행하는 단계는 명확하게 구분된다.
- EJB 애플리케이션 기존의 JAVA API등을 이용할 수 있다.
- 다른 언어로 개발된 컴포넌트와 함께 동작할 수 있다.
- CORBA 프로토콜을 수용하며 이용할 수 있다.

위와 같은 기능들은 EJB 아키텍처가 지켜야 할 목적입니다. 반대로 프로젝트에 EJB를 사용할 것인가? 라고 결정할 때 개발자들이 염두에 둘 부분이기도 합니다.

4. EJB의 종류

EJB는 세션 빈(Session Bean)과 엔티티 빈(Entity Bean)으로 구분됩니다. 각각의 사용용도는 아래와 같습니다.

세션 빈 – 실제 로직을 담당하는 빈입니다. 예를 들자면, 물품검색, 물품주문, 물품 결재 등의 처리를 해주는 기능을 컴포넌트화 한 것을 의미합니다.

- 실제로 세션빈은 상태를 유지할 수 있는 상태유지 세션빈과 무상태 세션빈으로 나뉩니다.

엔티티 빈 – 일반적으로 데이터베이스의 테이블을 의미합니다. 즉 자료를 나타내는 집합을 나타내는 객체입니다.

- 엔티티 빈은 EJB 컨테이너가 자동으로 데이터 베이스를 관리하는 CMP(Container Managed Persistence)엔티티 빈과 빈 자체가 데이터 베이스를 관리하는 BMP(Bean Managed Persistence)엔티티 빈으로 나뉘어 집니다.

5. EJB 개발 순서

a. 홈 인터페이스와 리모트 인터페이스의 작성

- 홈 인터페이스(Home Interface) : 엔터프라이즈 빈을 클라이언트가 사용할 수 있도록 생성하고 찾아주는 기능
- 리모트 인터페이스(Remote Interface) : 엔터프라이즈 빈이 클라이언트에게 제공하는 서비스를 메소드들로 선언한 인터페이스

b. Bean Class의 작성

- 엔터프라이즈 빈이 실제로 처리하는 작업을 내부코드로 구체적으로 작성하는 클래스입니다. Remote Interface에서 선언된 비즈니스 메소드를 실제로 구현해 줘야 합니다. 개발자 입장으로 보면 가장 할 일이 많은 작업이기도 합니다.
- Bean Class를 작성할 때는 Remote Interface에서 정의된 메소드를 실제로 구현해 주는 것 외에도 EJB 컨테이너의 규약 메소드를 정의해야 합니다. 규약 메

소드는 EJB 컨테이너가 특정한 순간에 호출하는 메소드들로 아주 중요한 기능을 처리합니다.

c. **디플로이먼트 디스크립터(Deployment Descriptor) 작성**
   - 디플로이먼트 디스크립터는 XML파일로 엔터프라이즈 빈의 이름, 트랜잭션 처리방법, 보안, 자원관리 방법 등의 정보를 작성한 파일

d. **엔터프라이즈 빈과 관련된 모든 클래스와 디플로이먼트 디스크립터의 패키지화(jar로 묶음)**

참고 : 위에서 c. d. 를 합쳐서 "디플로이먼트" 라고 합니다.

e. **패키지를 EJB 컨테이너에 설치함**
   - 참고로 EJB 컨테이너안에 EJB패키지를 설치하는 개발자를 배치자(deployer)라고 호칭합니다.
   - 컨테이너에 설치하는 과정을 '플러그인' 이라고 말하며, 플러그인 되는 과정에서 리모트 인터페이스를 구현한 클래스, 홈 인터페이스를 구현한 클래스, 각각의 RMI 스텁, 스켈레톤 클래스가 자동으로 생성됩니다.

6. J2EE의 설치

SUN사에서 제공하여 주는 EJB 컨테이너인 J2EE를 설치하도록 하겠습니다.

a. http://www.javasoft.com/j2ee/ **에서 J2EE 소프트웨어를 다운로드 받습니다**. 윈도우용의 경우 파일명은 j2sdkee-1_3_01-win.exe 입니다.

b. **j2sdkee-1_3_01-win.exe를 실행합니다.** 아래의 그림과 같은 방법으로 설치합니다. 보통 next만 누르면 됩니다.

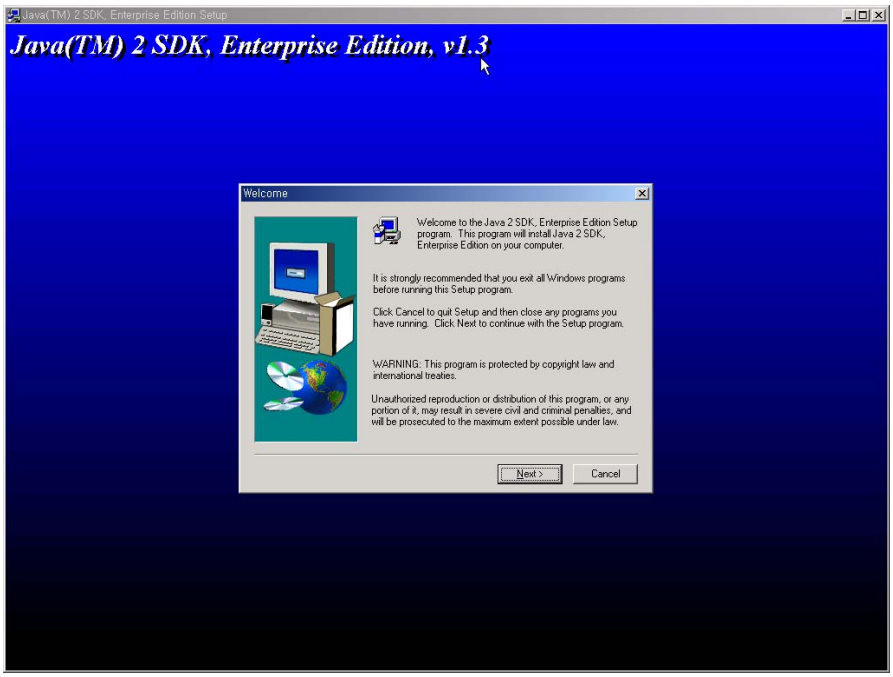

그림 1 환영메시지

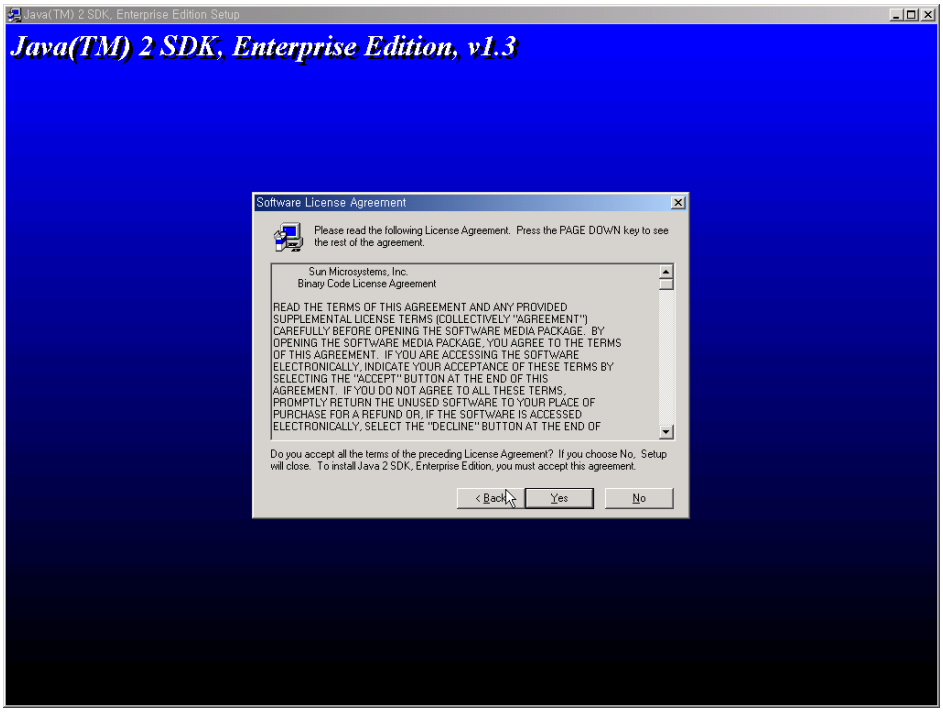

그림 2 라이센스 동의

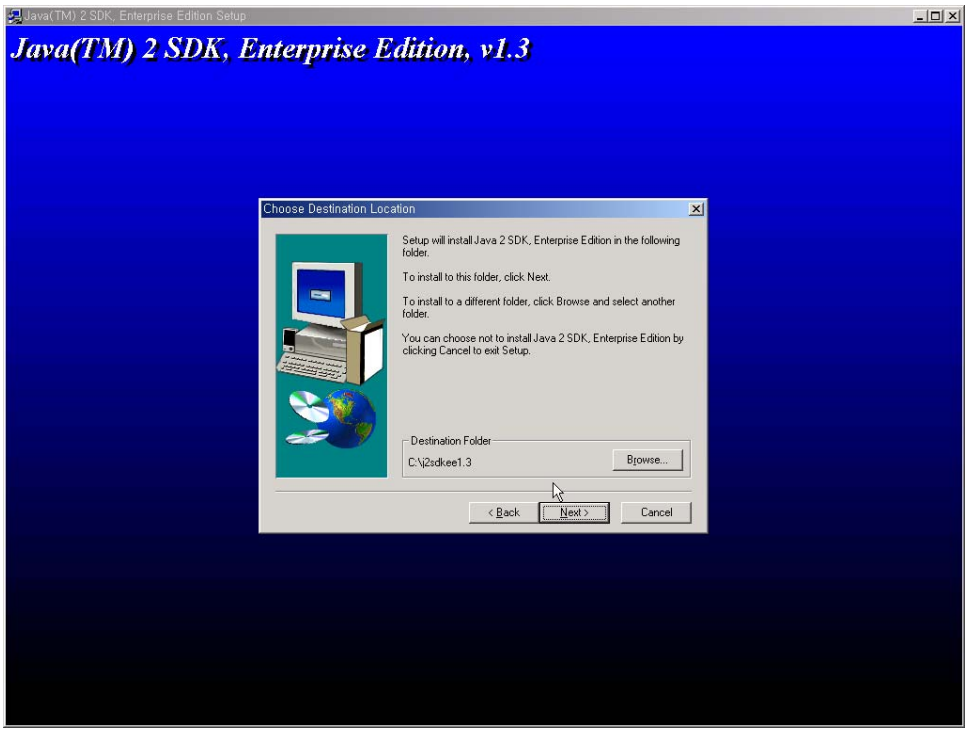

그림 3 설치디렉토리 지정 : 기본은 c:₩j2sdkee1.3₩

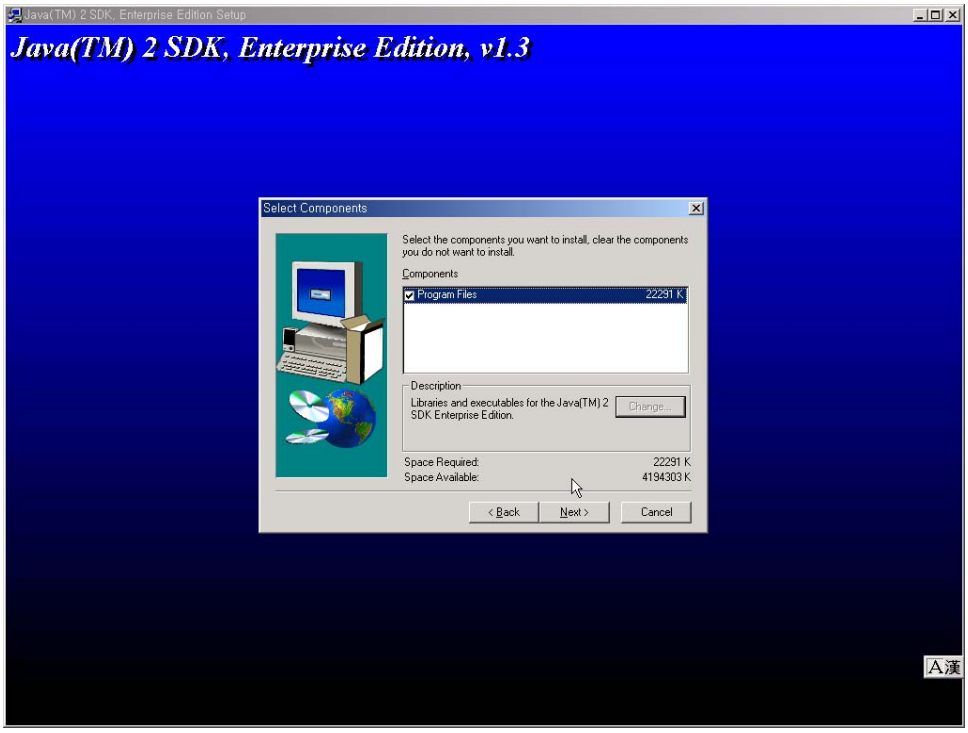

그림 4 설치프로그램 선택

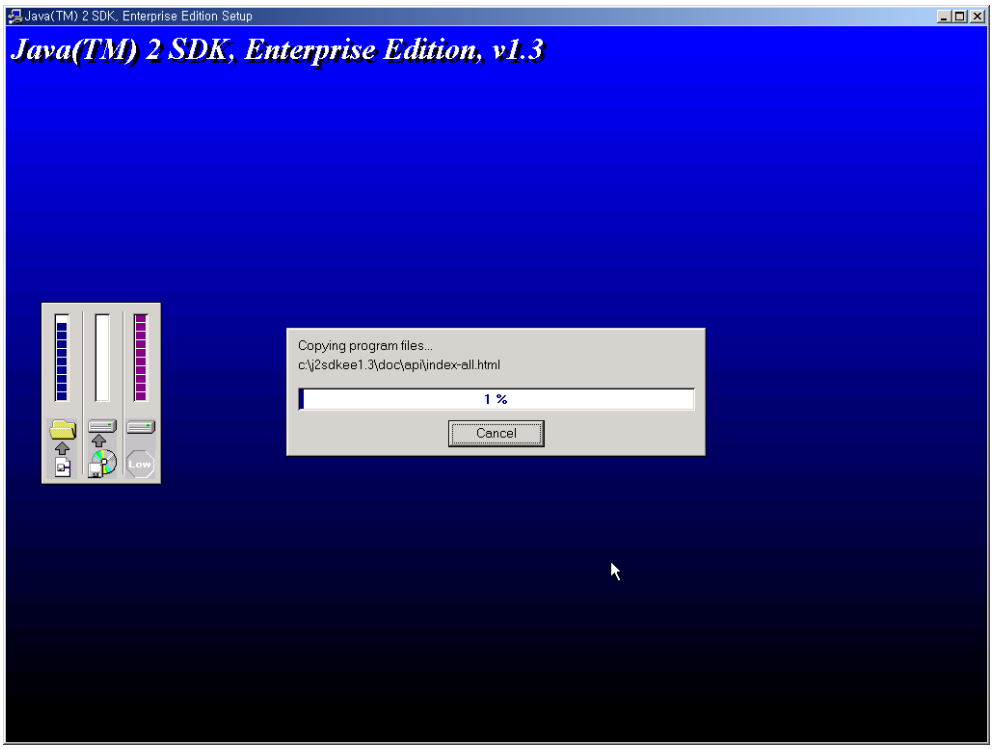

그림 5 설치진행화면

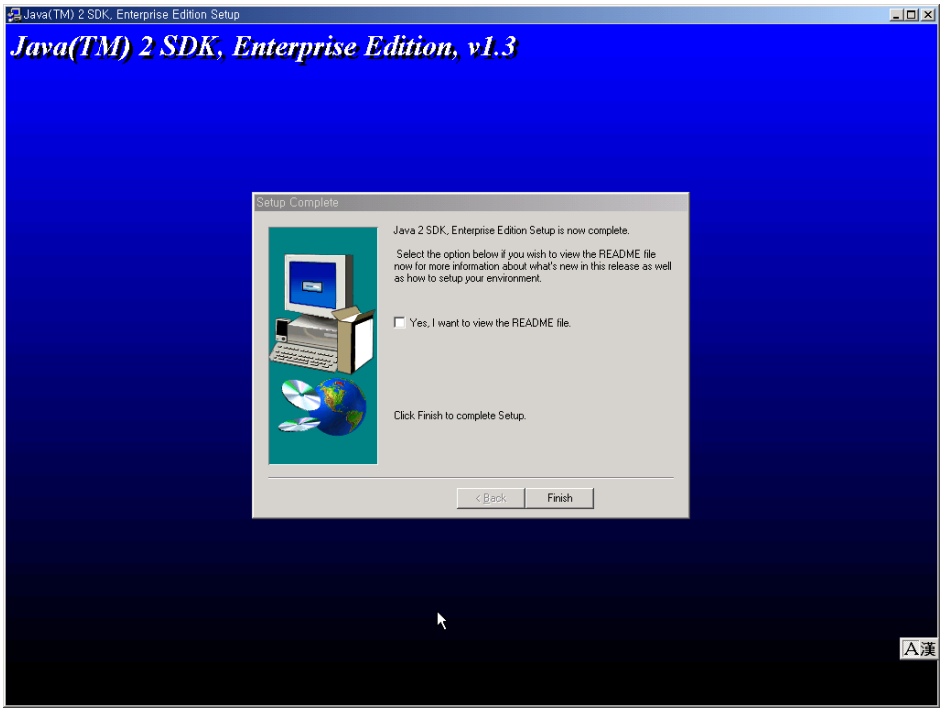

그림 6 설치종료화면

c. **설치가 끝나면 제어판의 시스템등록정보에서 환경변수를 아래와 같이 설정합니다.**

- JAVA_HOME : jdk 설치된 디렉토리입니다. 예) c:₩jdk1.3
- J2EE_HOME : J2EE가 설치된 디렉토리입니다. 예) c:₩j2sdkee1.3
- PATH : jdk와 J2EE의 프로그램을 실행할 수 있도록 path에 추가합니다. 예) PATH=.;c:₩jdk1.3₩bin;c:₩j2sdkee1.3₩bin
- CLASSPATH : J2EE의 클래스를 추가합니다. 예) set CLASSPATH=.;c:₩j2sdkee1.3₩lib₩j2ee.jar;

참고. J2EE SDK의 중요한 도구들

a. J2EE 서버

중요옵션들

-verbose : J2EE서버를 구동한 후 로그출력

-version : 현재 J2EE 버전 출력

-stop : J2EE 서버 중지

-singleVM : 하나의 프로세스로 EJB컨테이너 실행, 디플로이

-multiVM : 배치된 각각의 애플리케이션에 대하 추가로 가상 머신이 실행된다. 속도는 빨라지지만 메모리 자원을 많이 사용하게 됨

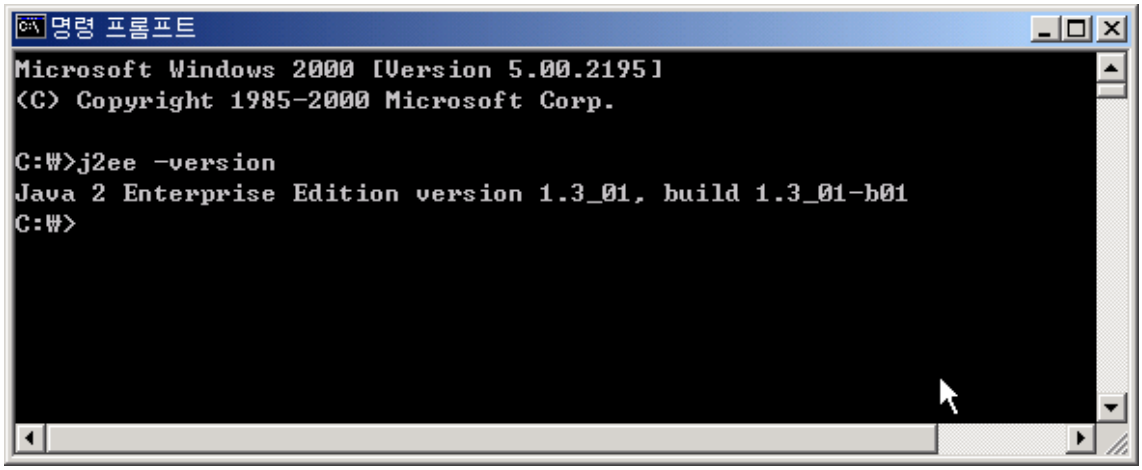

그림 7 j2ee를 command창에서 실행한 모습 버전이 1.3_01이라는 것을 알 수 있다.

b. Deploytool

애플리케이션 배치툴로 J2EE의 컴포넌트나 애플리케이션을 작성, 배치할 때 사용됩니다. 실행은 deploytool.exe 파일을 실행하면 됩니다

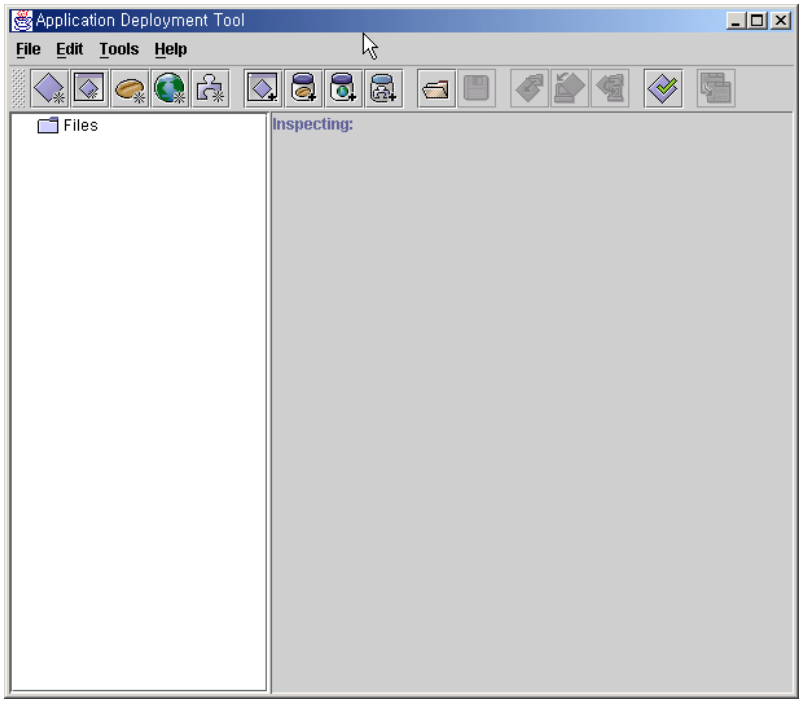

그림 8 deploytool 을 실행한 모습으로 기본적으로 GUI를 지원한다.

c. CloudScape

- J2EE에 기본적으로 포함되어 있는 데이터베이스 서버

d. KeyTool

- 공개 키와 개인 키를 생성하여 X.509 self-signed 인증서를 생성하여 주는 도구입니다.

e. Packager

- J2EE의 컴포넌트를 패키지화 할 때 이용됩니다. 해당 파일을 이용하여 엔터프라이즈 빈 file, 웹 애플리케이션 WAR file, 애플리케이션 클라이언트 JAR file, J2EE 애플리케이션 EAR file, 자원 어댑터(adapter) RAR file 등을 만들 수 있습니다.

f. Realm Tool

- J2EE 사용자, 파일을 추가 삭제할 수 있는 툴

g. RunClient

- 애플리케이션 클라이언트를 실행하기 위한 툴

h. verifier
- 컴포넌트의 유효성을 검사하는 툴

i. CleanUp
- J2EE 서버에 배치된 모든 애플리케이션을 삭제하는 툴

7. 무상태 세션빈 만들기

실제로 간단한 무상태 세션빈을 만들어 보도록 하겠습니다. "5. EJB 개발 순서" 를 참고하시기 바랍니다. 어떤 프로그램이든지 hello 를 찍지 않고서야..... 넘어갈 수가 없군요.

a. 홈 인터페이스의 작성

홈인터페이스는 기본적으로 java.rmi 패키지와 javax.ejb 패키지를 임포트 하여야 하며 javax.ejb.EJBHome은 상속받아 구현해 줍니다. 인터페이스의 이름은 홈 인터페이스인 것을 쉽게 알아보도록 하여 주기 위하여 Home이라는 이름을 보통 붙여서 작성합니다. 또한 create메소드를 작성하여 주는데 무상태 세션빈의 경우 인자가 없는 메소드를 선언합니다. create메소드가 리턴하여 주는 값은 리모트 인터페이스가 됩니다. EJB컨테이너가 트랜잭션, 패일오버등을 자동으로 처리해 주기 위하여는 직접 객체를 생성하면 안되기 때문에 create 메소드를 이용하여 interface를 리턴하는 것입니다. create 메소드는 CreateException과 RemoteException을 throws 할 수 있도록 작성되야 합니다.

HelloHome.java 시작 ----------

```
import java.rmi.*;
import javax.ejb.*;

public interface HelloHome extends EJBHome{
    public Hello create() throws CreateException, RemoteException;
}
```

HelloHome.java 끝 ----------

b. 리모트 인터페이스의 작성

리모트 인터페이스는 클라이언트가 세션빈의 메소드를 이용할 수 있도록 제공되는 것입니다. 리모트 인터페이스와 마찬가지로 java.rmi 패키지와 javax.ejb 패키지를 구현해 주었습니다. 파일명은 리모트 인터페이스에서 create 메소드가 리턴하는 이름과 같아야 합니다. 여기에서는 간단히 문자열을 지정한 후, 지정된 값을 리턴할 수 있는 기능을 가진 두개의 메소드를 선언하였습니다. 또한 각각의 메소드는 RemoteException 을

throws 해야 합니다.

Hello.java 시작 ----------

```
import java.rmi.*;
import javax.ejb.*;

public interface Hello extends EJBObject{
  public void setHello(String txt) throws RemoteException;
  public String getHello() throws RemoteException;
}
```

Hello.java 끝 ----------

c. 빈 클래스 작성

빈 클래스는 실제로 리모트 인터페이스를 구현해주는 클래스입니다. 개발자에게는 가장 중요한 부분이며, 세션 빈일 경우 javax.ejb.SessionBean을 구현해줘야 합니다. 또한 EJB컨테이너 규약 메소드도 구현해줘야 하는데, HelloEJB.java에서는 선언만 해주었고 실제로 구현하지는 않았습니다.

HelloEJB.java 시작 ----------

```
import java.util.*;
import javax.ejb.*;
public class HelloEJB implements SessionBean{
    public String txt;

    public void setHello(String txt){
            this.txt = txt;
    }

    public String getHello(){
            return txt;
    }

    public HelloEJB(){}
    public void ejbCreate(){}
    public void ejbRemove(){}
    public void ejbActivate(){}
    public void ejbPassivate(){}
```

```
    public void setSessionContext(SessionContext sc){}
}
HelloEJB.java 끝 ----------------------------------
```

d. **홈 인터페이스, 리모트 인터페이스 그리고 빈 클래스를 컴파일 합니다.** J2EE의 설치와 환경변수를 제대로 설정하였다면 아래와 같이 도스창에서 실행합니다.(물론 파일이 있는 디렉토리여야 겠지요?)

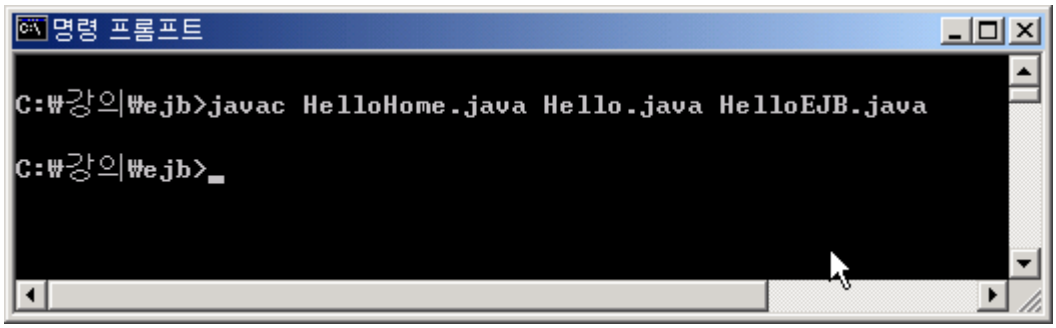

그림 9 홈,리모트 인터페이스, 빈클래스의 컴파일

e. **디플로이먼트를 합니다.**

먼저 J2EE의 서버를 실행한 후 deploytool을 실행합니다. 그림을 보고 차례대로 따라하도록 합시다.

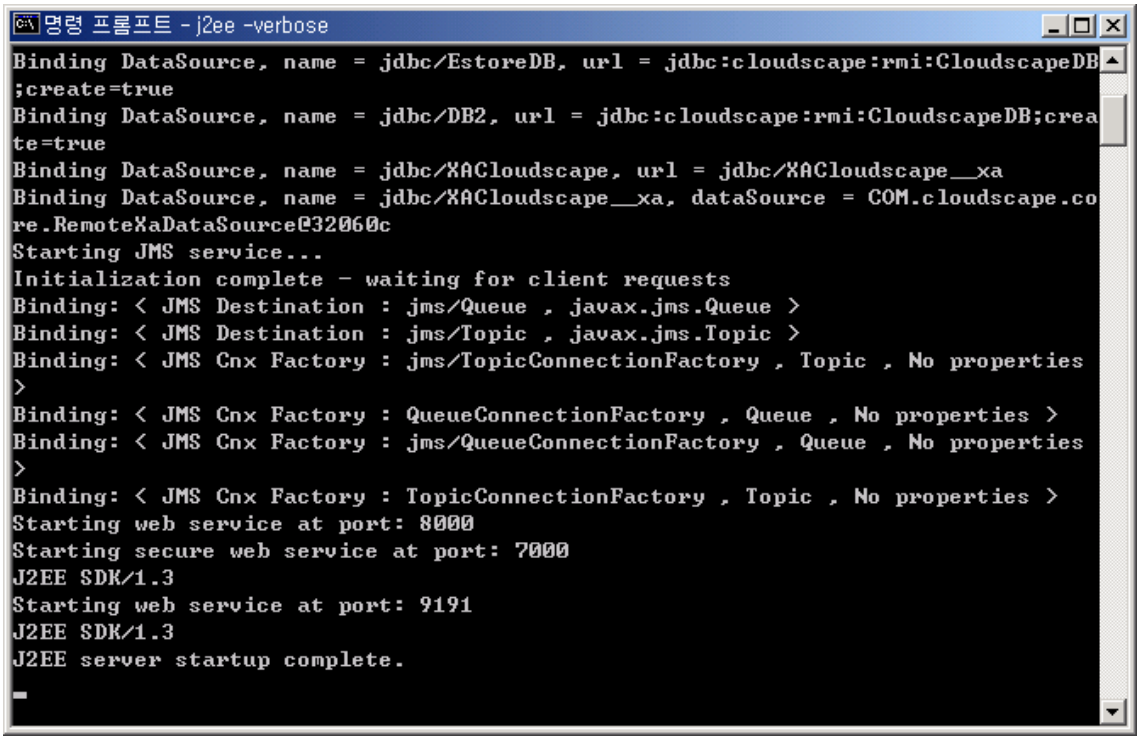

그림 10 j2ee 서버를 –verbose 옵션을 주어 실행

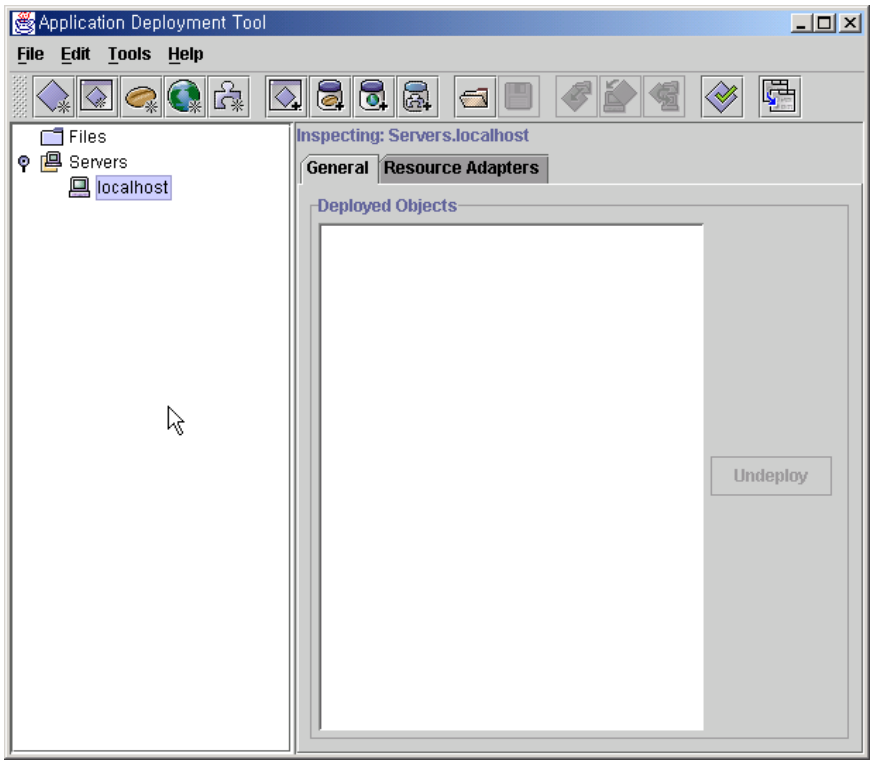

그림 11 deploytool을 실행한 모습

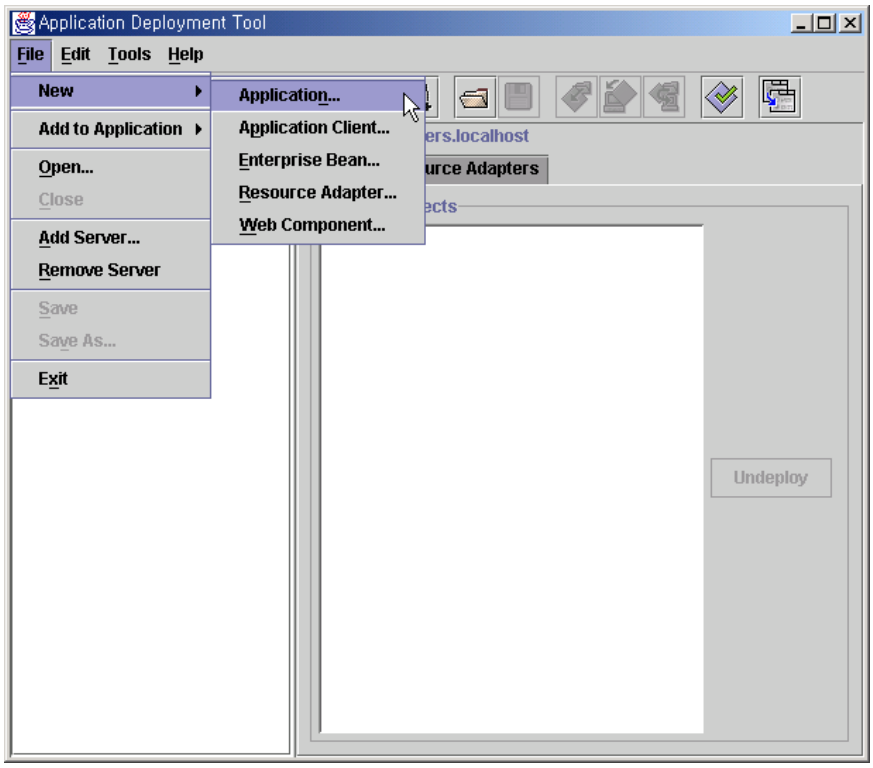

그림 12 File메뉴에서 Application을 선택합니다.

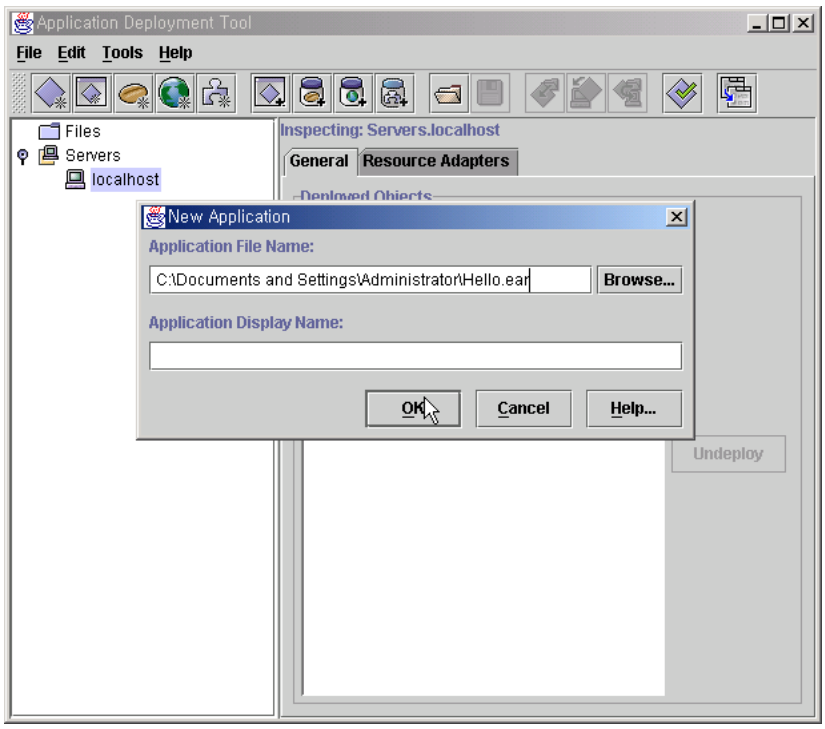

그림 13 다이얼로그 창이 뜨면 저장될 어플리케이션 파일 이름을 지정합니다. 예제에서는 Hello.ear 라고 주었습니다.

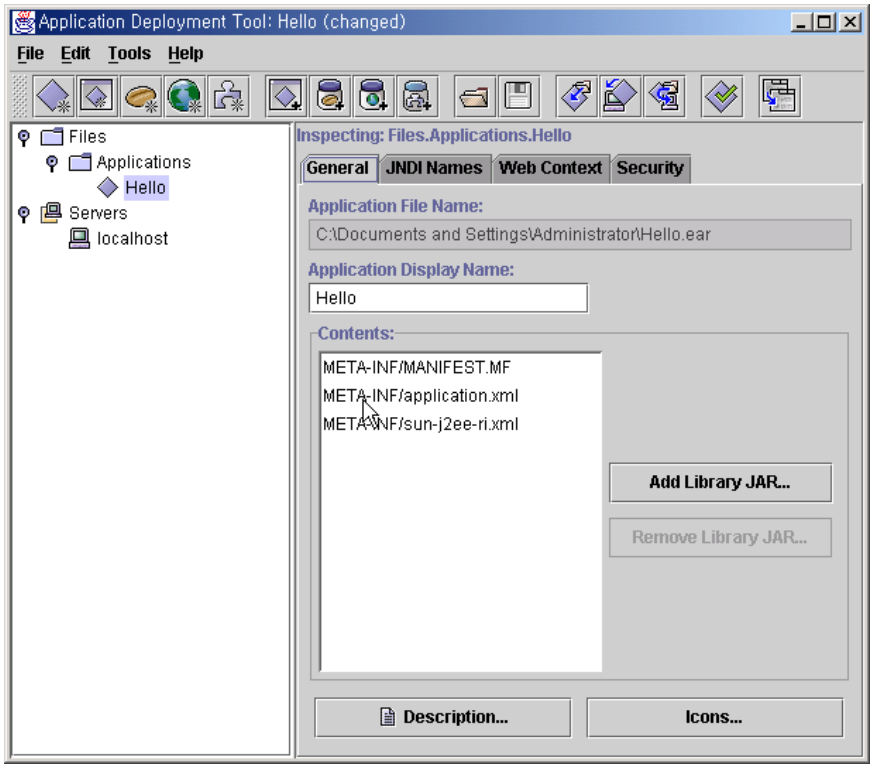

그림 14 새로운 어플리케이션이 작성이 되면 왼쪽메뉴에 Hello 라는 이름의 어플리케이션이 추가된 것을 알 수 있습니다.

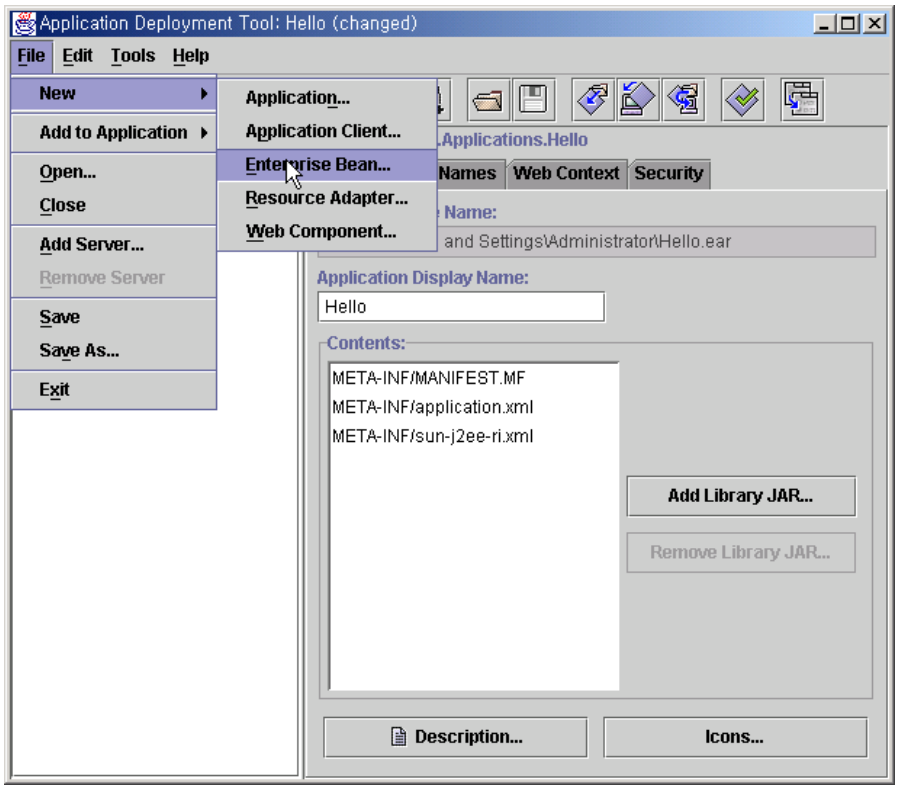

그림 15 엔터프라이즈 빈을 새롭게 생성합니다..

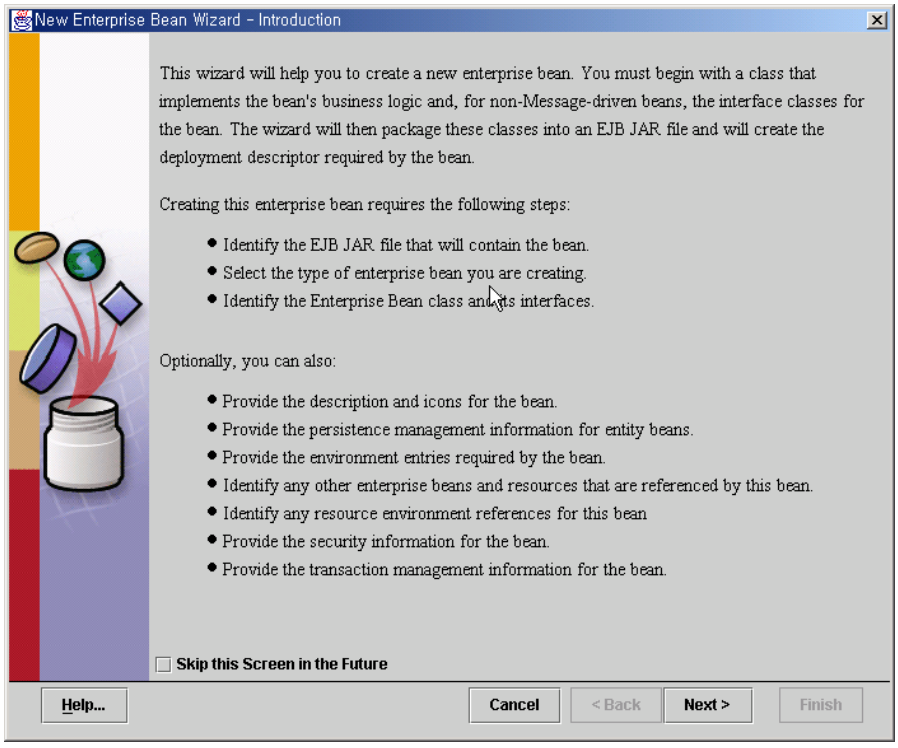

그림 16 엔터프라이즈생성을 누르면 정보가 나옵니다. 잘 읽고나서 Next를 누릅니다.

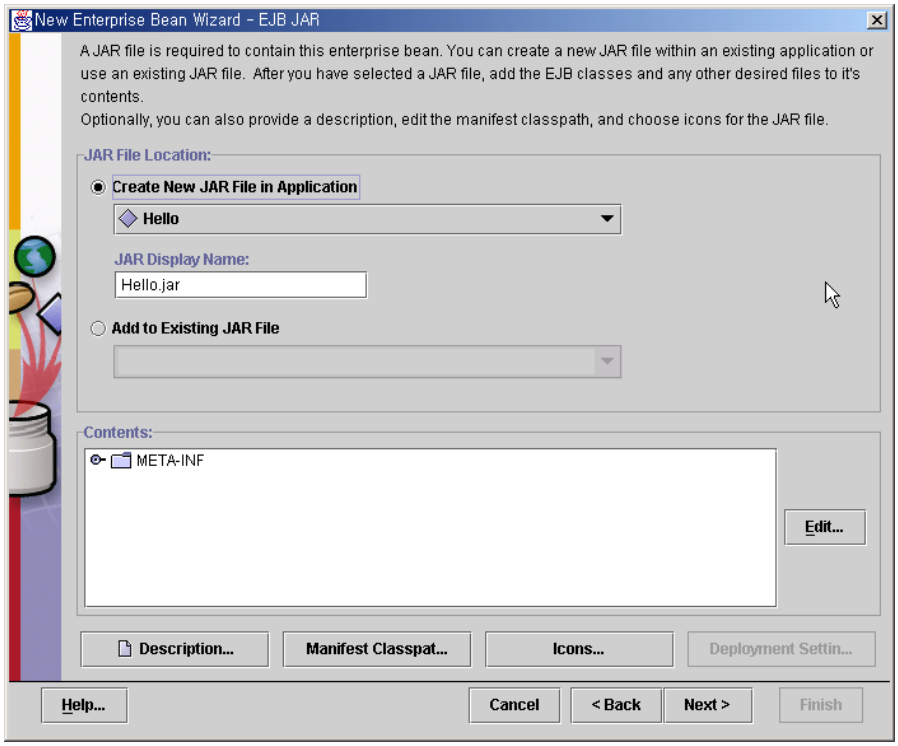

그림 17 그림과 같이 이름을 지정합니다. 그 후에 Edit 버튼을 클릭합니다.

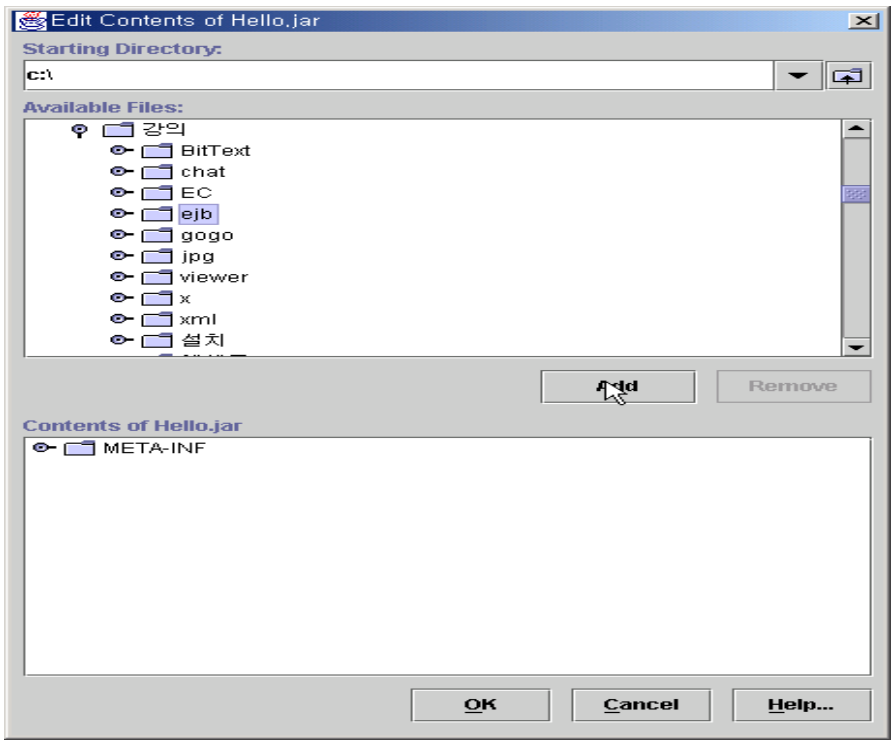

그림 18 Hello.jar 파일안에 포함될 class가 담긴 디렉토리를 상단 트리에서 선택한 후 add 버튼을 클릭합니다..

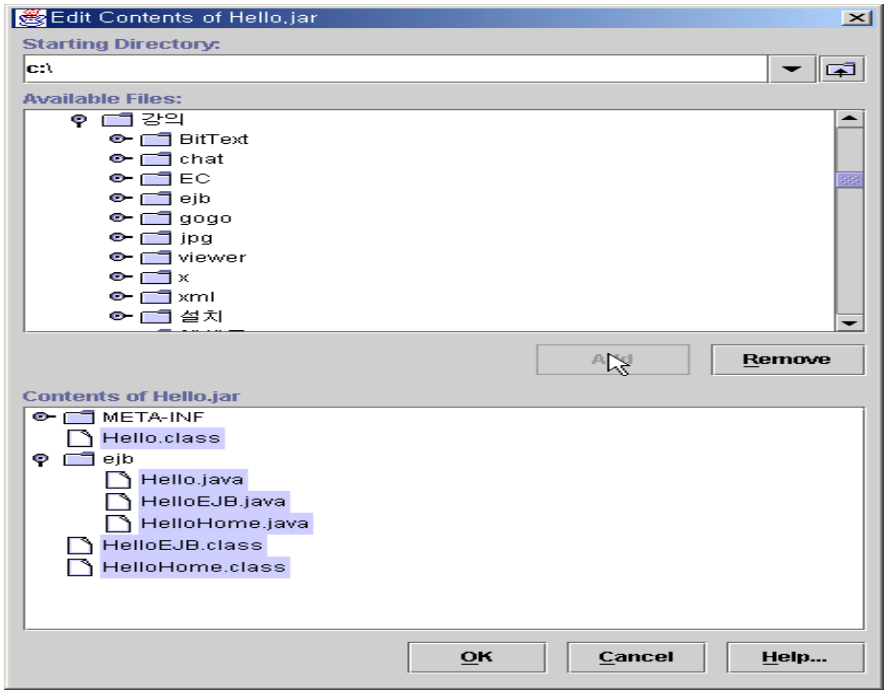

그림 19 디렉토리 안에 포함된 내용이 jar 파일에 추가되는 것을 비쥬얼 하게 알 수 있습니다. 이때 디렉토리 안까지 들어가 HelloEjb.class HelloHome.class Hello.class 만 추가해도 됩니다. ok 버튼을 클릭합니다.

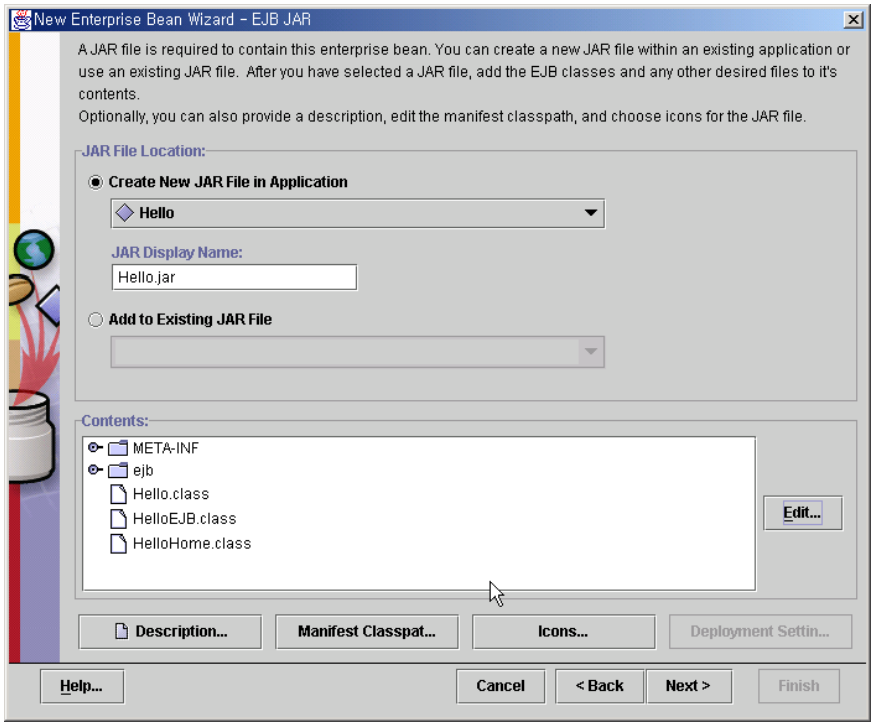

그림 20 화면 상단에 보면 jar 파일안에 들어갈 자료 목록이 보입니다. Next버튼을 클릭합니다.

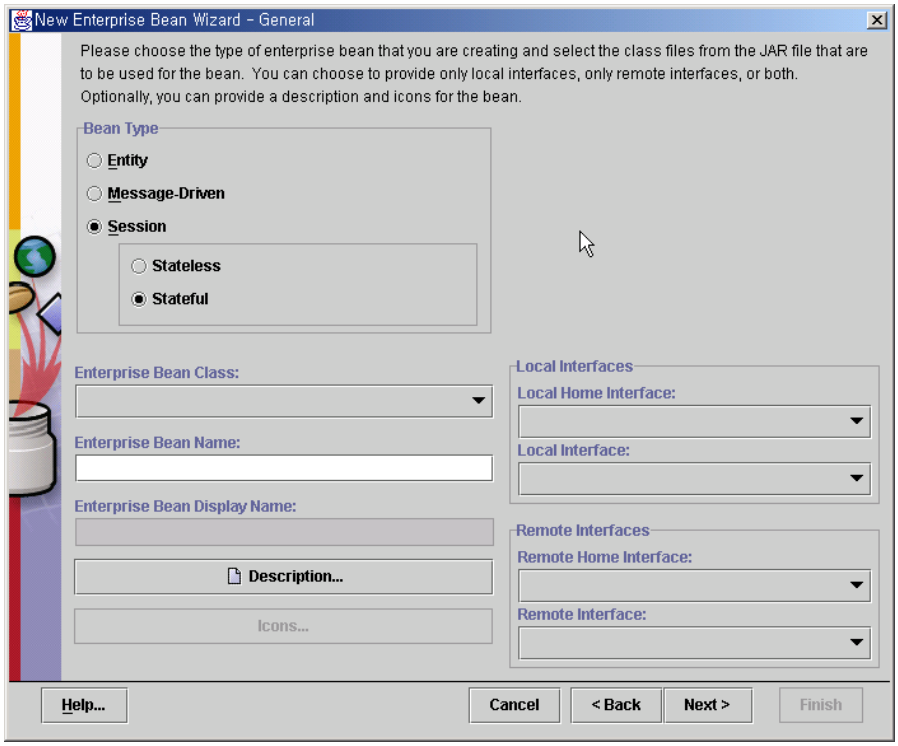

그림 21 파일들에 대한 설정하는 창입니다. 그림22)와 같이 설정하여 줍니다.

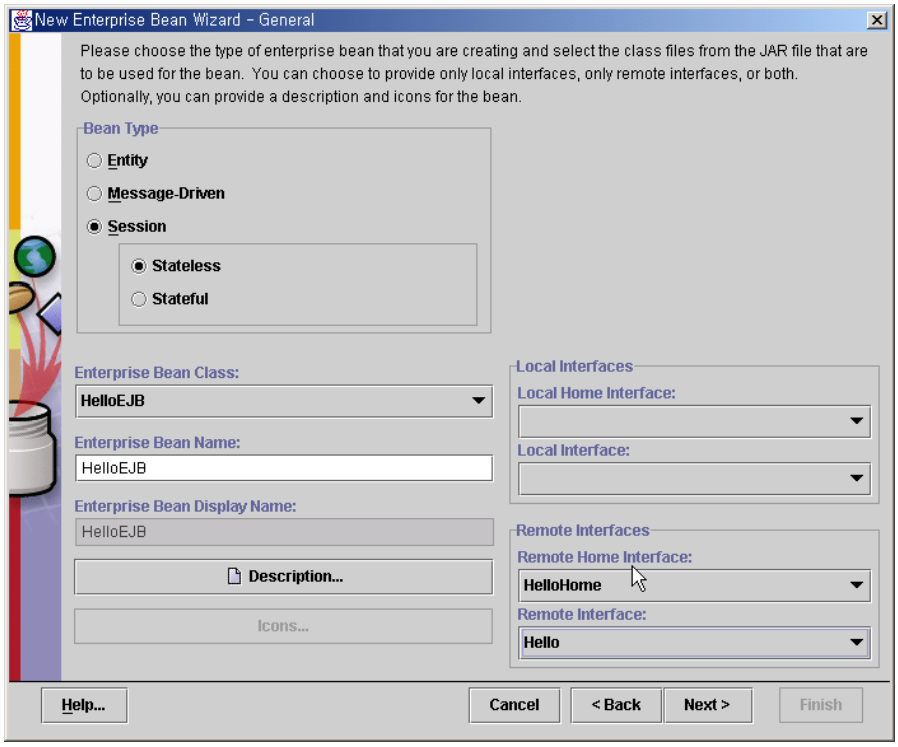

그림 22 무상태 새션빈이므로 Stateless 를 선택하였으며 빈 클래스, 홈 인터페이스, 리모트 인터페이스를 지정합니다. Next 버튼을 클릭합니다.

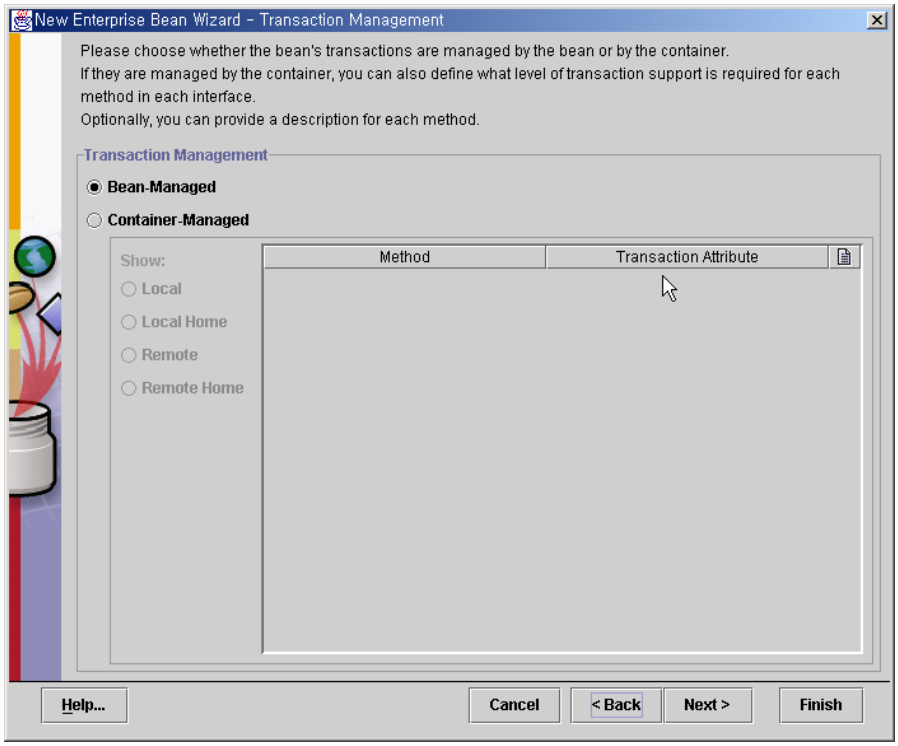

그림 23 무상태 세션빈에서는 보통 이용하지 않는 부분입니다. Finish버튼을 클릭하여 작업을 종료합니다.

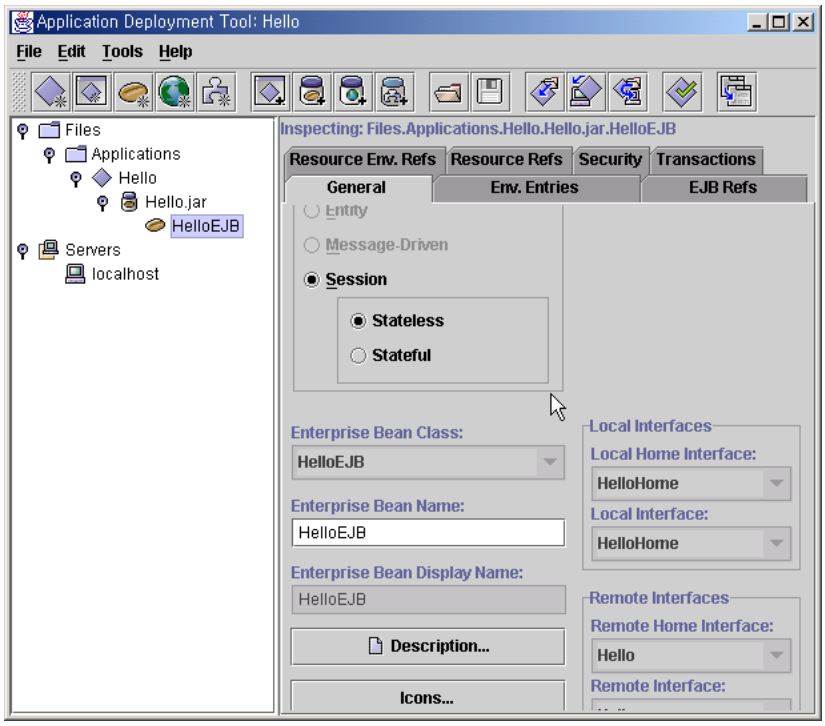

그림 24 Hello Application에 Hello.jar 가 포함되어 있고 EJB 빈이 추가된 것을 알 수 있습니다.

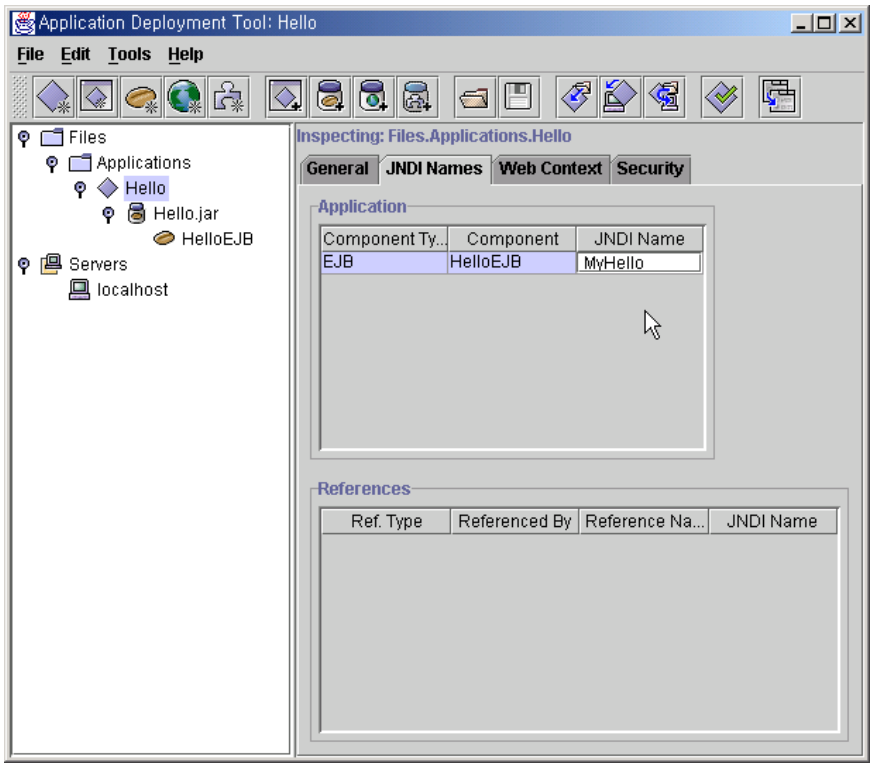

그림 25 Hello Application을 선택한 후 JNDI Names 항목에서 JNDI name을 설정합니다. 해당 이름은 JNDI를 이용하여 빈을 찾을 때 사용되는 중요한 이름입니다.

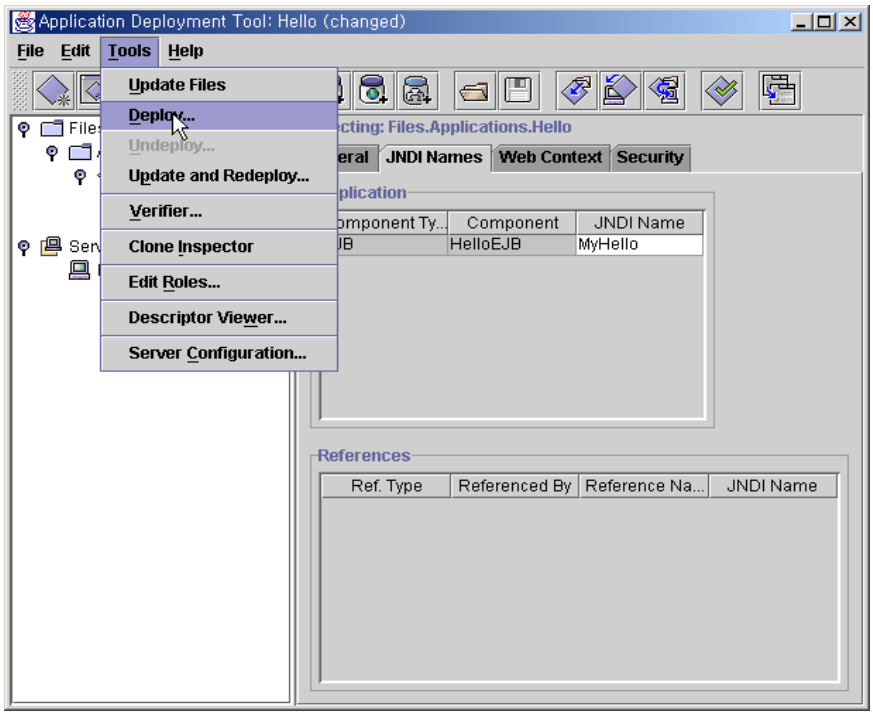

그림 26 이제 모든 설정이 끝났습니다. plugin 을 하기 위하여 tools 메뉴의 Deploy를 선택합니다.

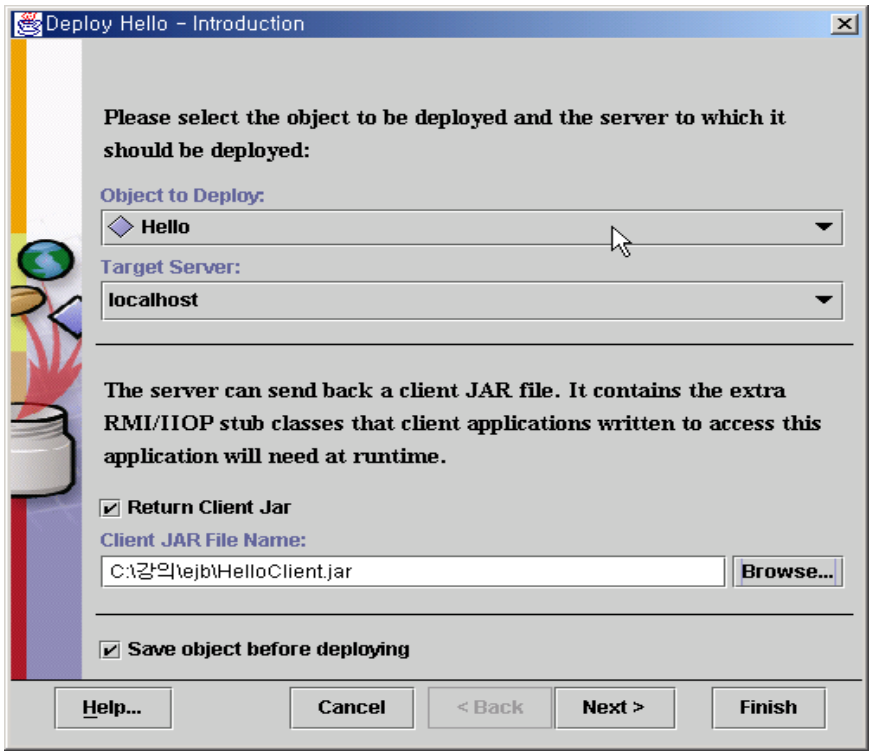

그림 27 Object to Deploy에서는 Hello를 선택한 후 Return Client Jar 옵션을 체크합니다. 해당 jar 파일은 client를 실행할 때 필요한 내용이 될 것 입니다. next 버튼을 클릭합니다.

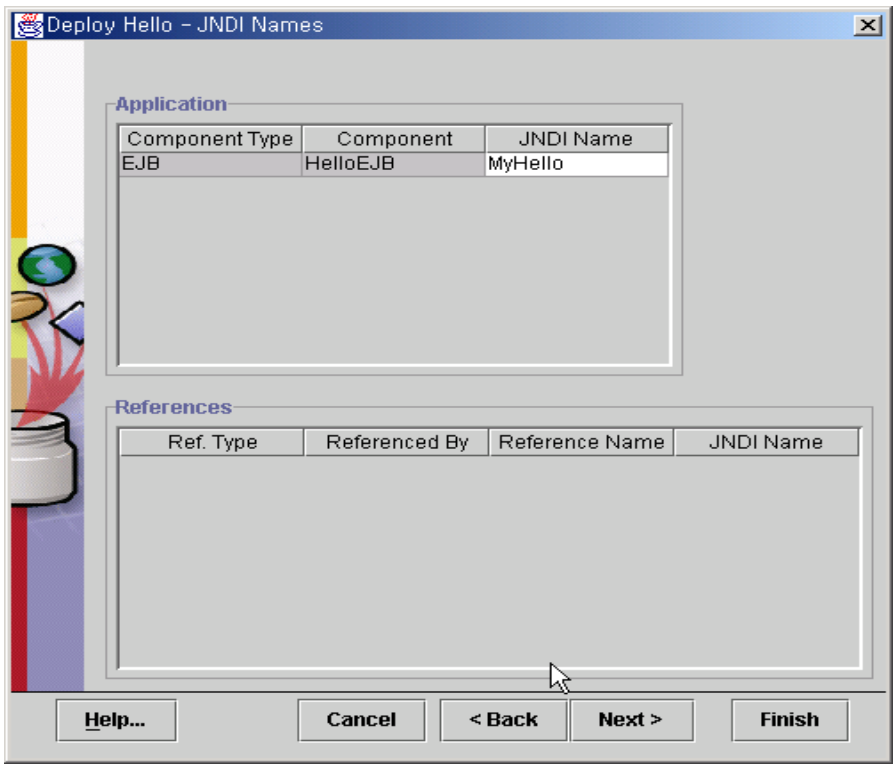

그림 28 JNDI Name을 지정하는 항목입니다. 이미 지정하였으니 Next버튼을 클릭합니다.

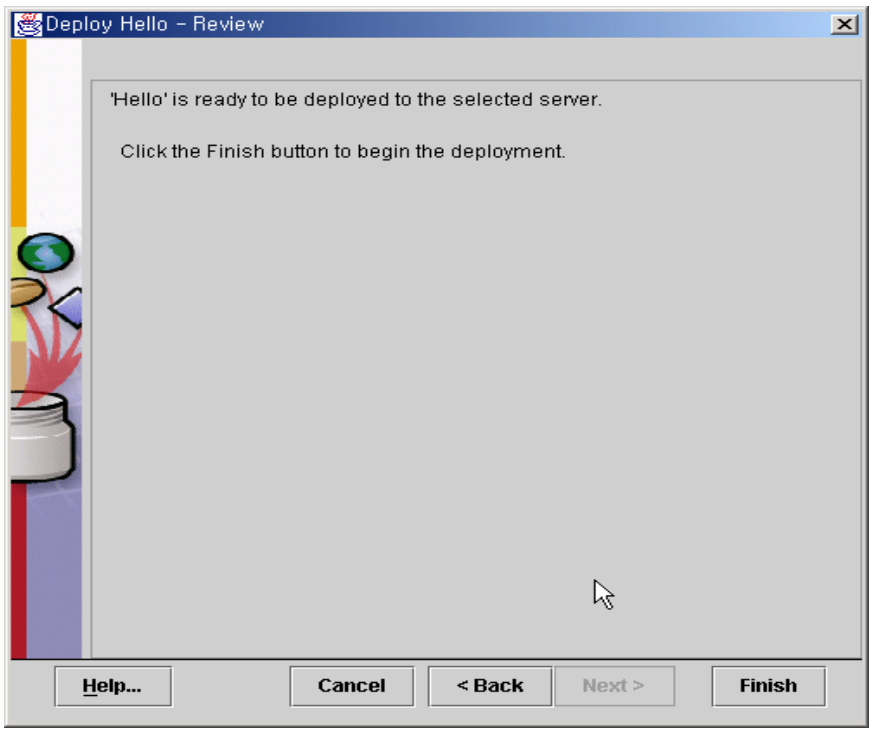

그림 29 Deploy될 준비가 끝났습니다. Finish버튼을 클릭합니다.

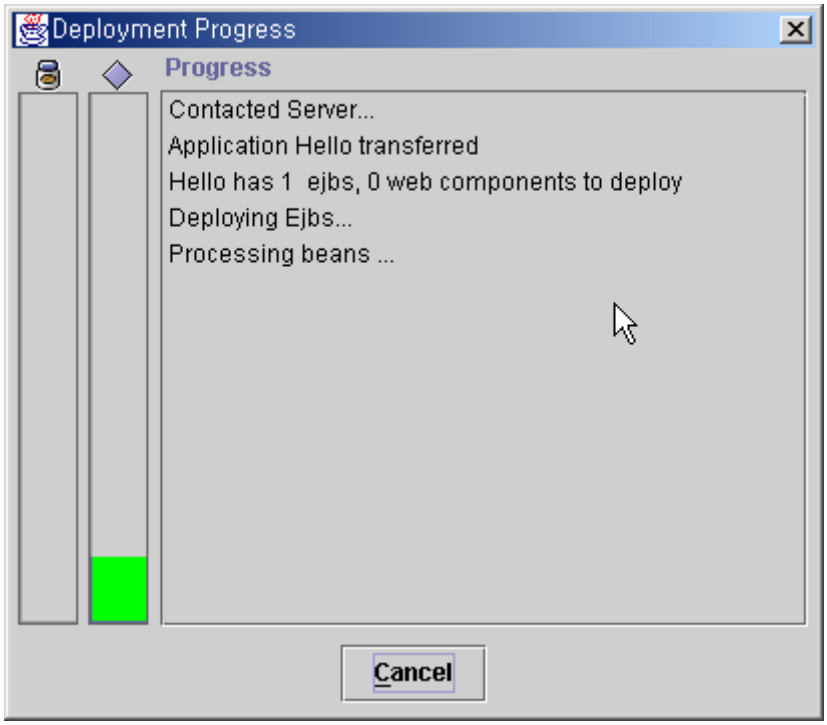

그림 30 디플로이먼트 작업이 진행됩니다. 이 부분에서 실패하는 경우가 많습니다. 물론 잘못 작성된 경우겠지요? ☺

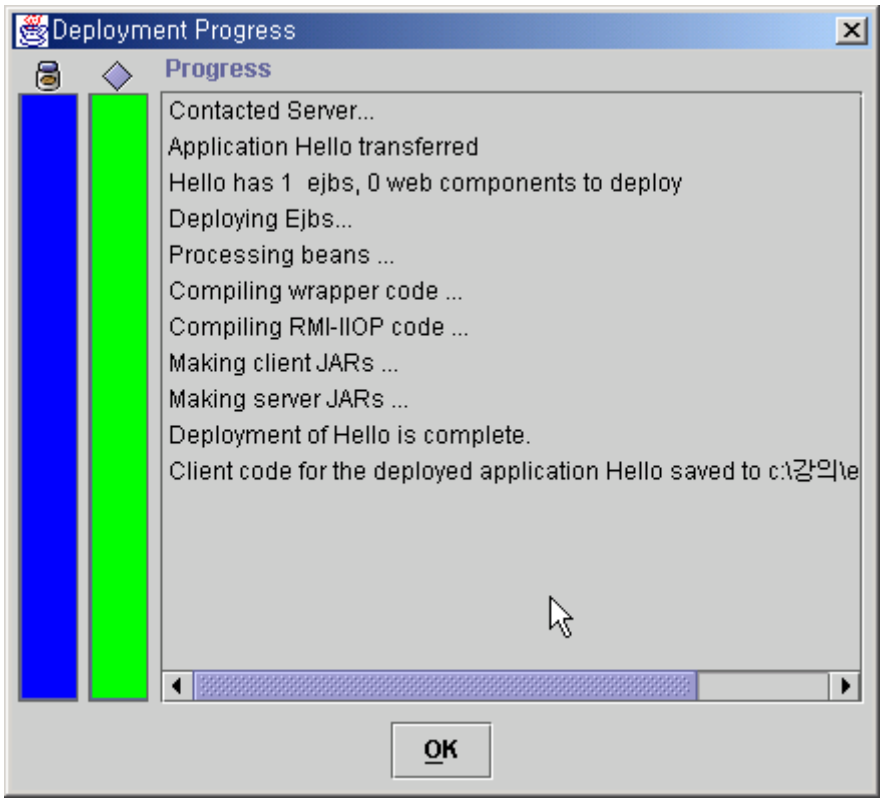

그림 31 디플로이 작업이 끝난 화면입니다. ok 버튼을 클릭합니다.

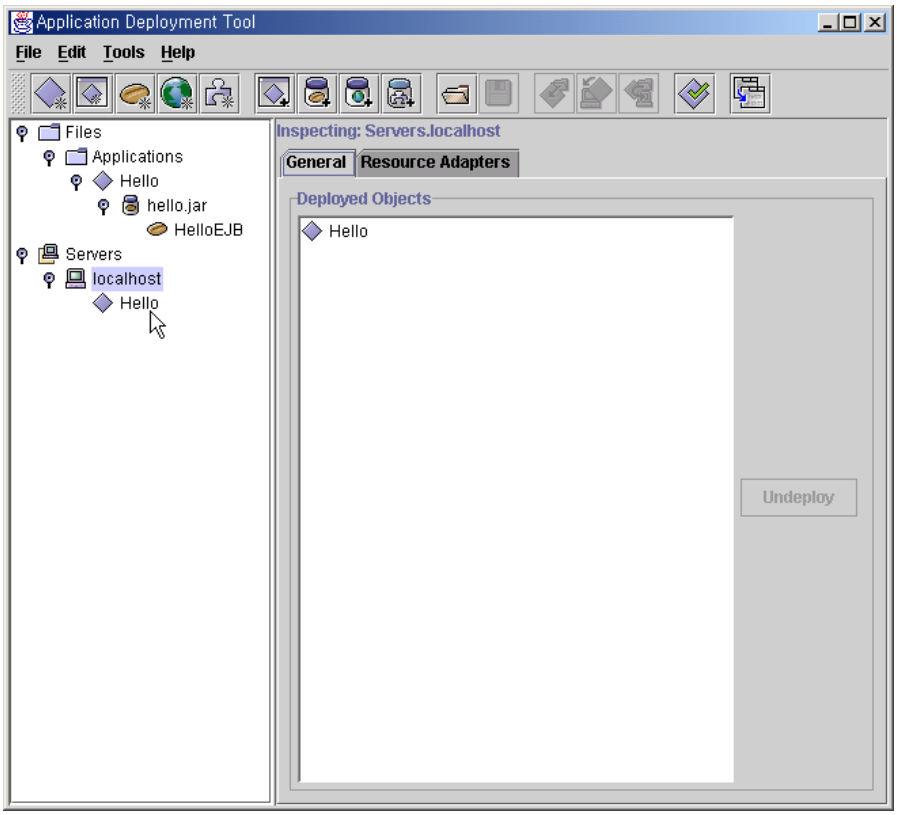

그림 32 왼쪽 트리메뉴에 보면 Hello라는 내용이 deploy 된 것을 알 수 있습니다.

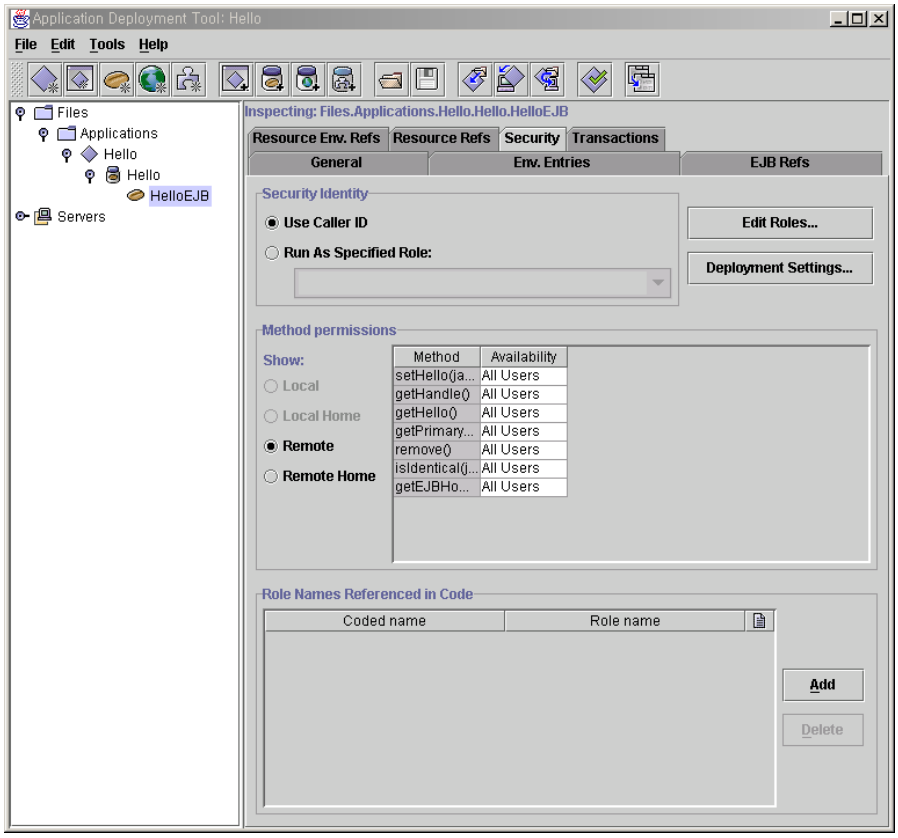

그림 33 그림32)까지 작업을 하고 클라이언트에서 엔터프라이즈 빈을 이용할려고 하면 Permission Exception이 발생할 것 입니다. 그림33)처럼 HelloEJB 엔터프라이즈 빈을 선택한 후 Security탭에서 Deployment Settings 를 클릭합니다.

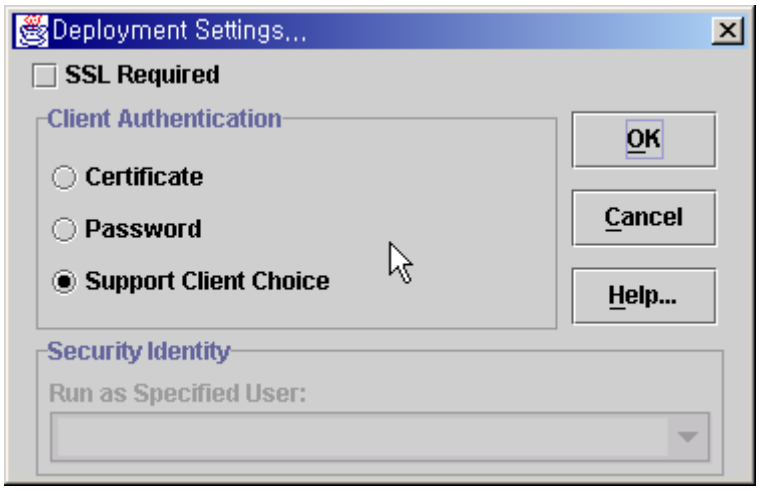

그림 34 원래는 Password로 지정되어 있을 것입니다만, Support Client Choice로 바꿉니다. Deploy 되어있는 EJB 를 해제하고 다시 Deploy시킵니다.

f. EJB 클라이언트의 작성

실제로 EJB컨테이너에 있는 HelloEJB객체를 이용하는 간단한 어플리케이션입니다. 설명은 소스의 주석문으로 대신합니다.

HelloClient.java 시작 ----------

```
import javax.naming.*;
import javax.rmi.PortableRemoteObject;

public class HelloClient{
    public static void main(String args[]){
        try{
            Context initial = new InitialContext();
            // 세션빈의 객체 레퍼런스를 가지고 옵니다.
            Object obj = initial.lookup("MyHello");

            // 홈 객체 레퍼런스를 가지고 옵니다.
            HelloHome home =
(HelloHome)PortableRemoteObject.narrow(obj, HelloHome.class);
            if(home == null) System.out.println("null -_-;");

            // EJB 객체를 생성합니다.
            Hello h1 = home.create();
            Hello h2 = home.create();

            // 비지니스 메소드 호출
            h1.setHello("안녕하세요. EJB의 세계에 온것을 환영해요.");
            System.out.println("메아리 :" + h1.getHello());

            h2.setHello("무상태 세션빈을 처음 연습합니다.");
            System.out.println("메아리2 :" + h2.getHello());
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

HelloClient.java 끝 ----------

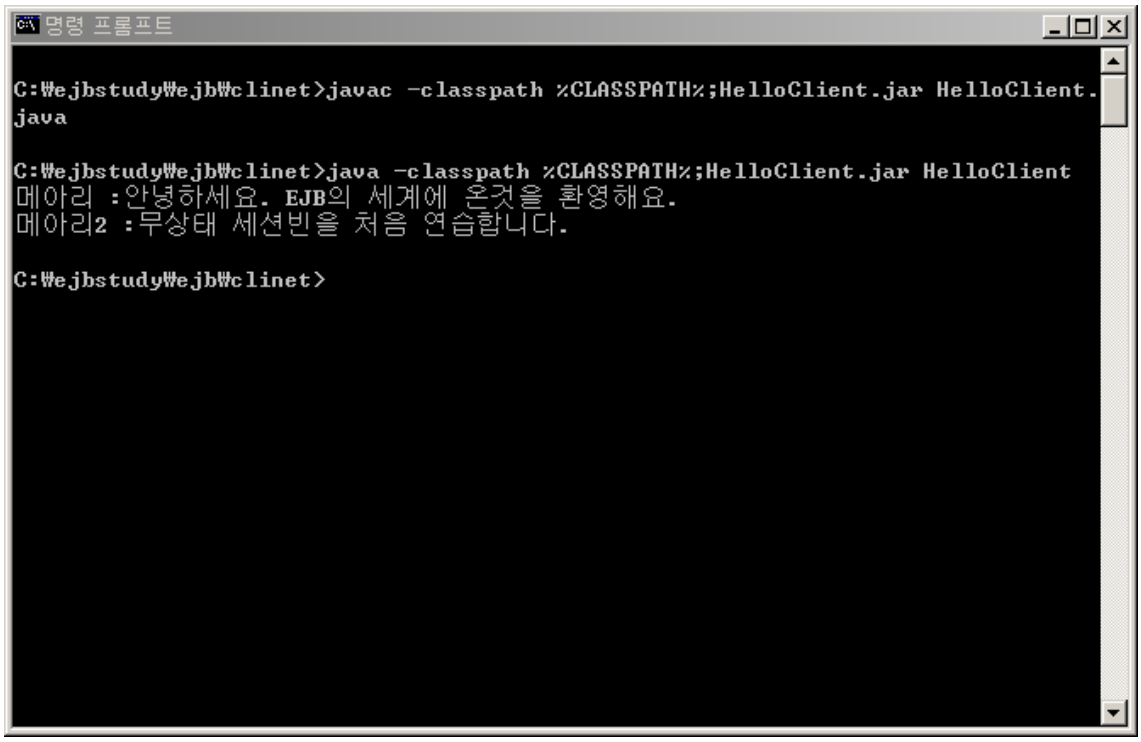

그림 35 첫번째 문장은 컴파일 하는 방법입니다. classpath 옵션을 잘 보면, DeployTool이 자동으로 만들어준 HelloClient.jar 를 지정한 것을 알 수 있습니다. 두 번째 문장은 HelloClient 를 실제로 실행하는 것입니다. 결과가 여러분들이 예상하는 것처럼 나왔는지요?

HelloClient.java 에서 다음과 같이 한 줄을 추가한 후에 결과를 확인 하면 어떤 값이 출력될까요?

```
h2.setHello("무상태 세션빈을 처음 연습합니다.");
System.out.println("메아리2 :" + h2.getHello());
System.out.println("메아리2 :" + h1.getHello());
```

진한 글씨로 된 부분을 추가한 후 다시 컴파일 하여 실행하여 봅시다. 무상태 빈의 특징에 대하여 확실히 알 수 있을 것입니다.(궁금한 분은 결과를 실행해 보시기 바랍니다.)

8. 무상태 세션빈과 JSP

이번에는 간단한 덧셈을 계산하여 주는 EJB 와 jsp 의 연동에 대하여 알아보도록 하겠습니다. 차례대로 따라하면서 익혀보시기 바랍니다.

a. Home Interface의 작성

```
AddHome.java 시작 ----------------------------------------
import java.rmi.*;
import javax.ejb.*;

public interface AddHome extends EJBHome{
  public Add create() throws CreateException, RemoteException;
}
AddHome.java 끝 ------------------------------------------
```

b. Remote Interface의 작성

```
Add.java 시작-------------------------------------------------
import java.rmi.*;
import javax.ejb.*;

public interface Add extends EJBObject{
  public int getAdd(int num1, int num2) throws RemoteException;
}
Add.java 끝 ---------------------------------------------------
```

c. Bean class의 작성

```
AddEJB.java 시작 -----------------------------------------
import java.util.*;
import javax.ejb.*;
public class AddEJB implements SessionBean{

  public int getAdd(int num1, int num2){
          return num1 + num2;
  }

  public AddEJB(){}
  public void ejbCreate(){}
```

```
    public void ejbRemove(){}
    public void ejbActivate(){}
    public void ejbPassivate(){}
    public void setSessionContext(SessionContext sc){}
}
AddEJB.java 끝 ------------------------------------------------
```

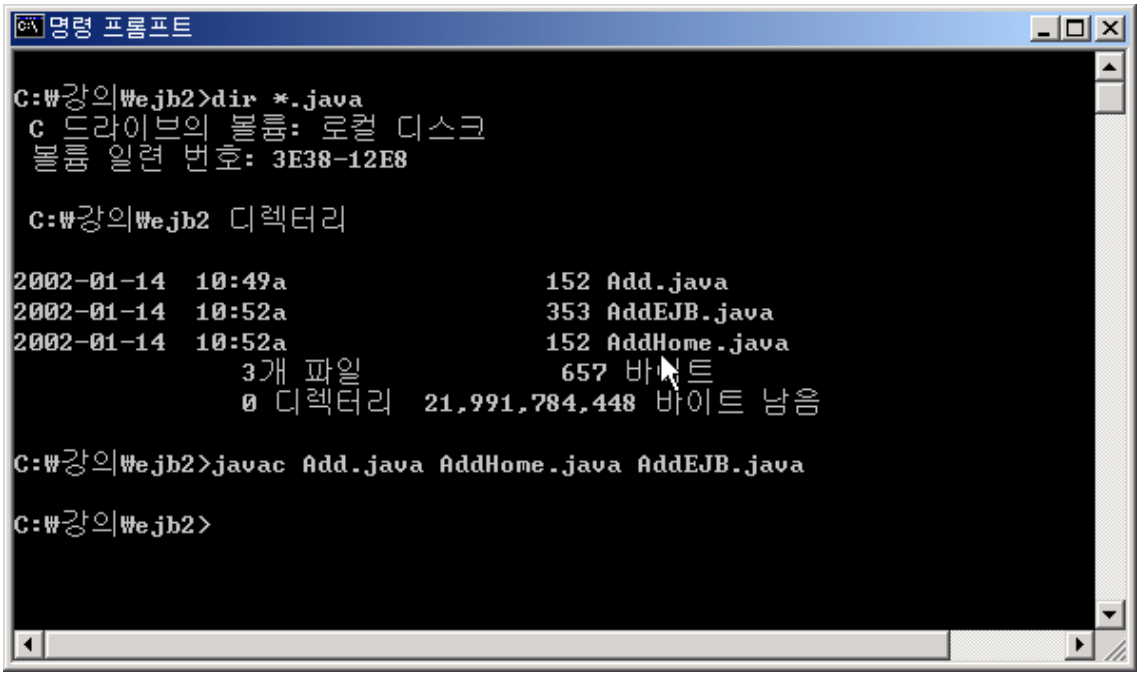

그림 36 홈인터페이스, 리모트인터페이스, 빈클래스를 함께 컴파일 합니다.

d. 디플로이먼트 하기

앞의 예제와 같은 방법으로 현재 만들어진 class들을 디플로이먼트 합니다. 그림을 잘 보고 따라 하시기 바랍니다.

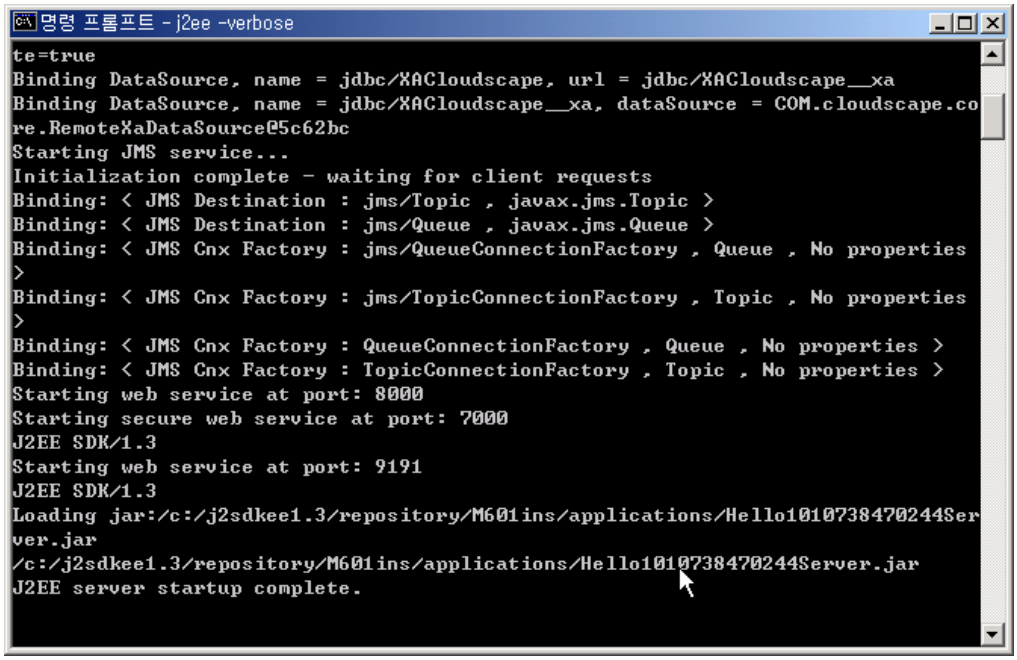

그림 37 j2ee 서버를 실행합니다.

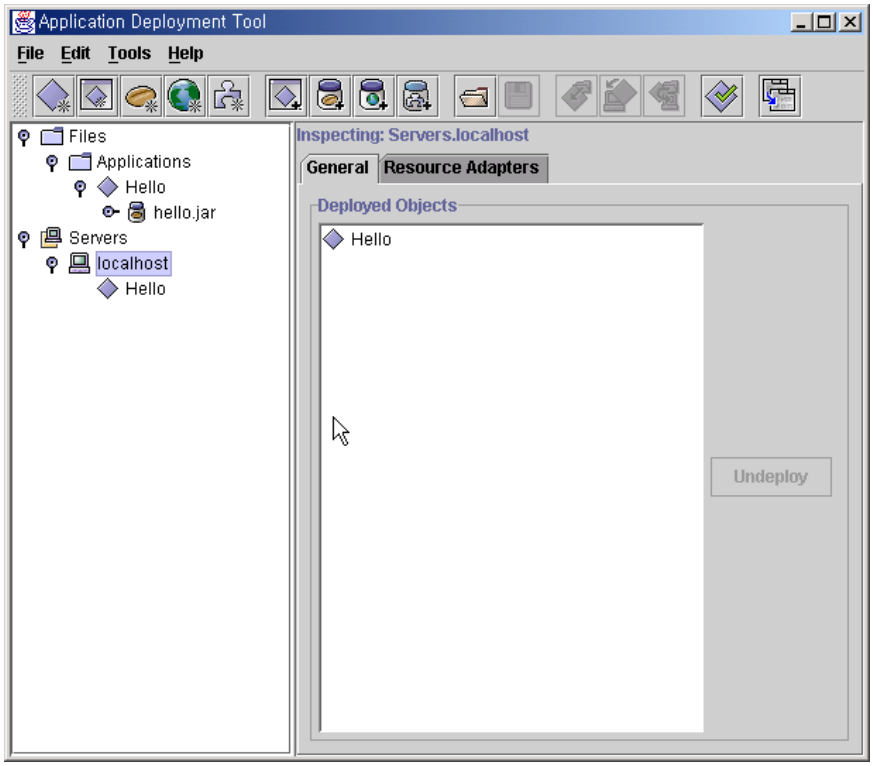

그림 38 deploytool 실행한 후 File – new – Application 을 선택한 후 add.ear 파일을 생성합니다.

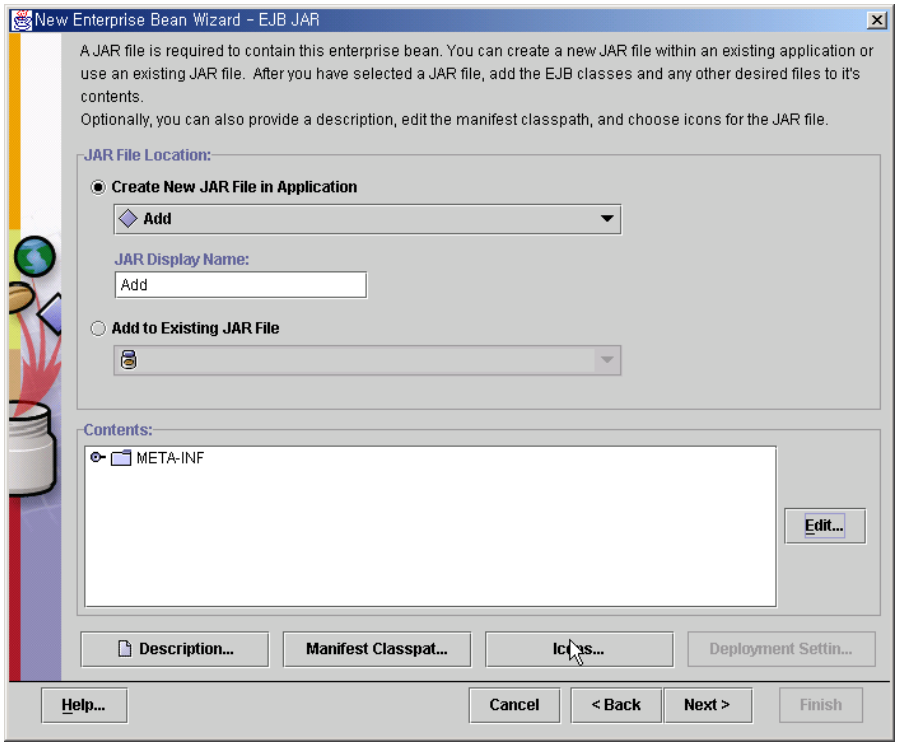

그림 39 File-new-Enterprise Bean 항목을 선택한 후, Add Application엔터프라이즈 빈을 추가할 내용을 선택합니다.

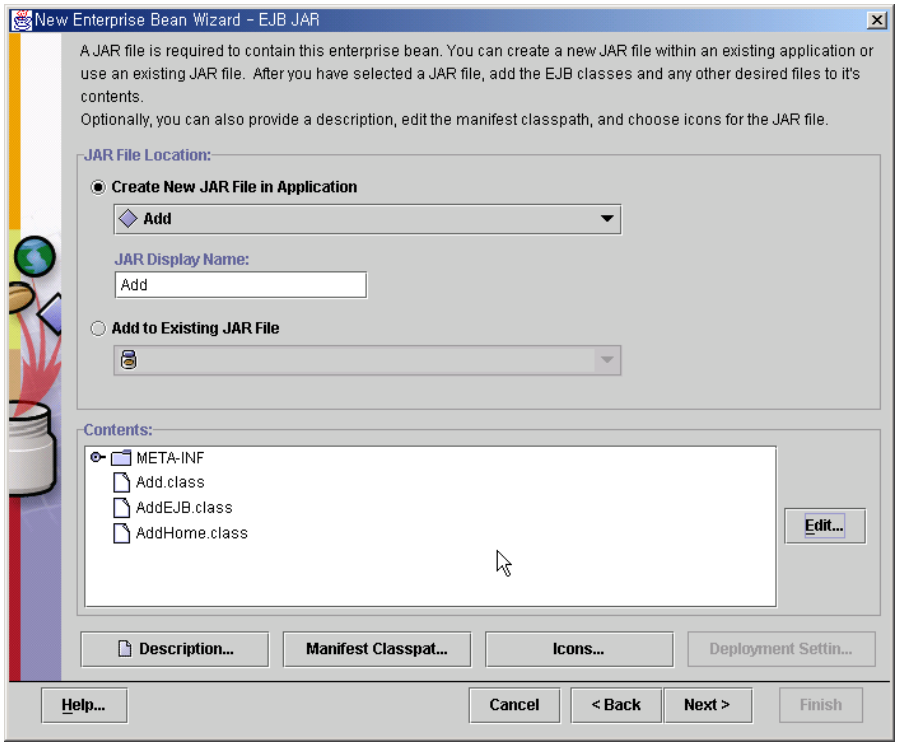

그림 40 컴파일된 Add, AddEJB, AddHome 클래스를 선택한 화면입니다.

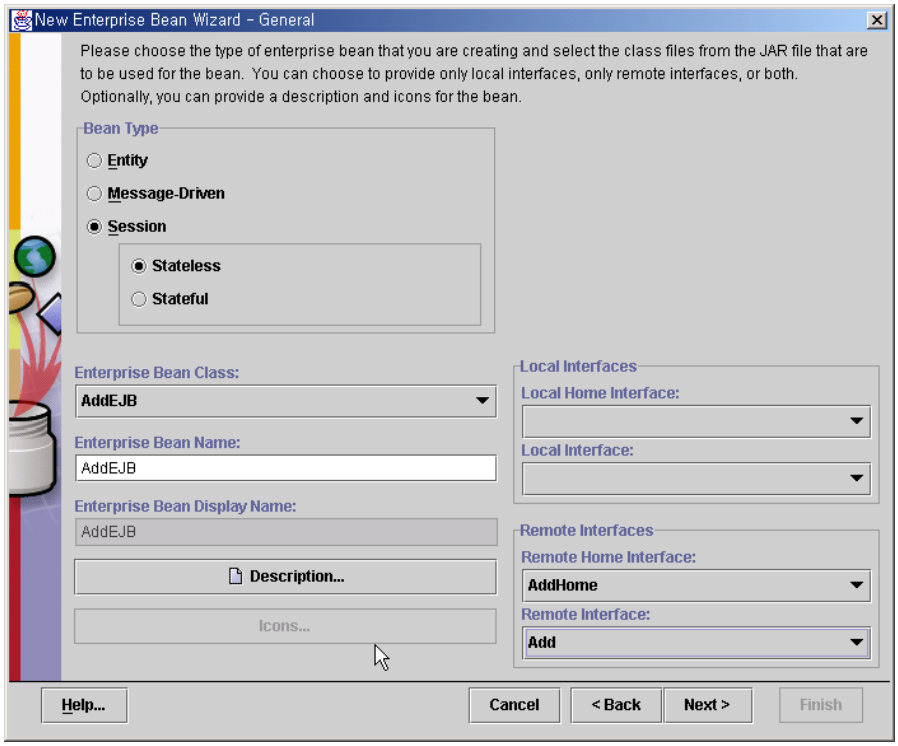

그림 41 무상태 세션빈이므로 Stateless 를 선택한 후 각각의 값을 그림과 같이 지정합니다.

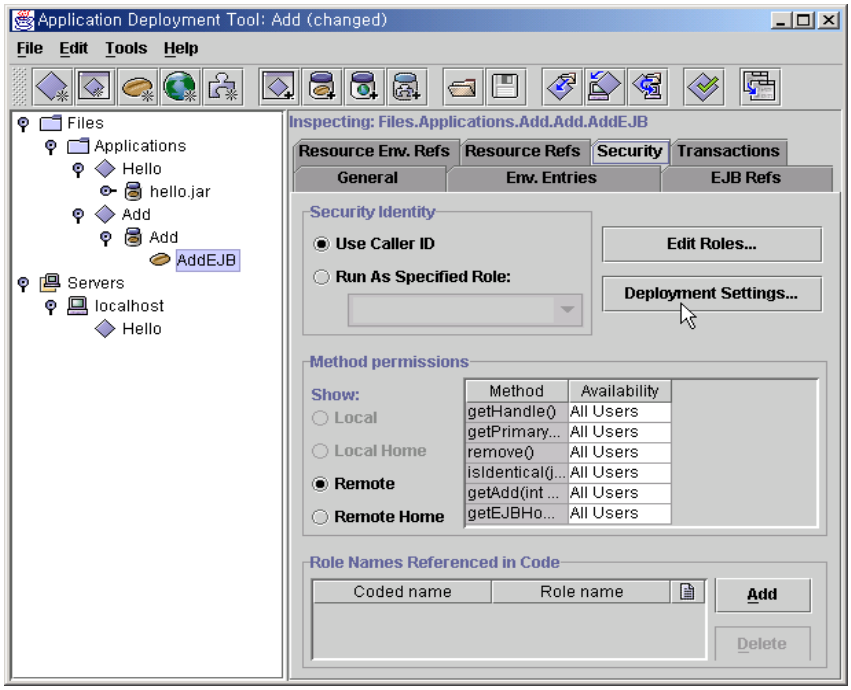

그림 42 엔터프라이즈 빈이 추가가 된 것을 확인하고 Security 탭을 그림과 같이 선택합니다.

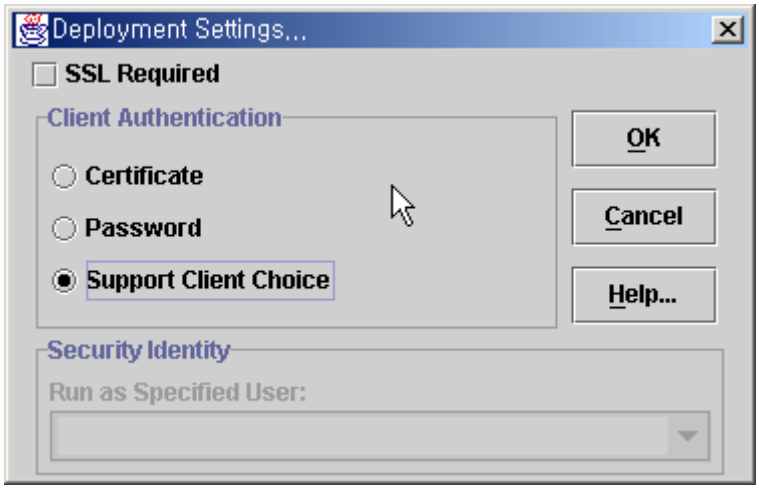

그림 43 보안 설정을 "Support Client Choice" 로 변경합니다.

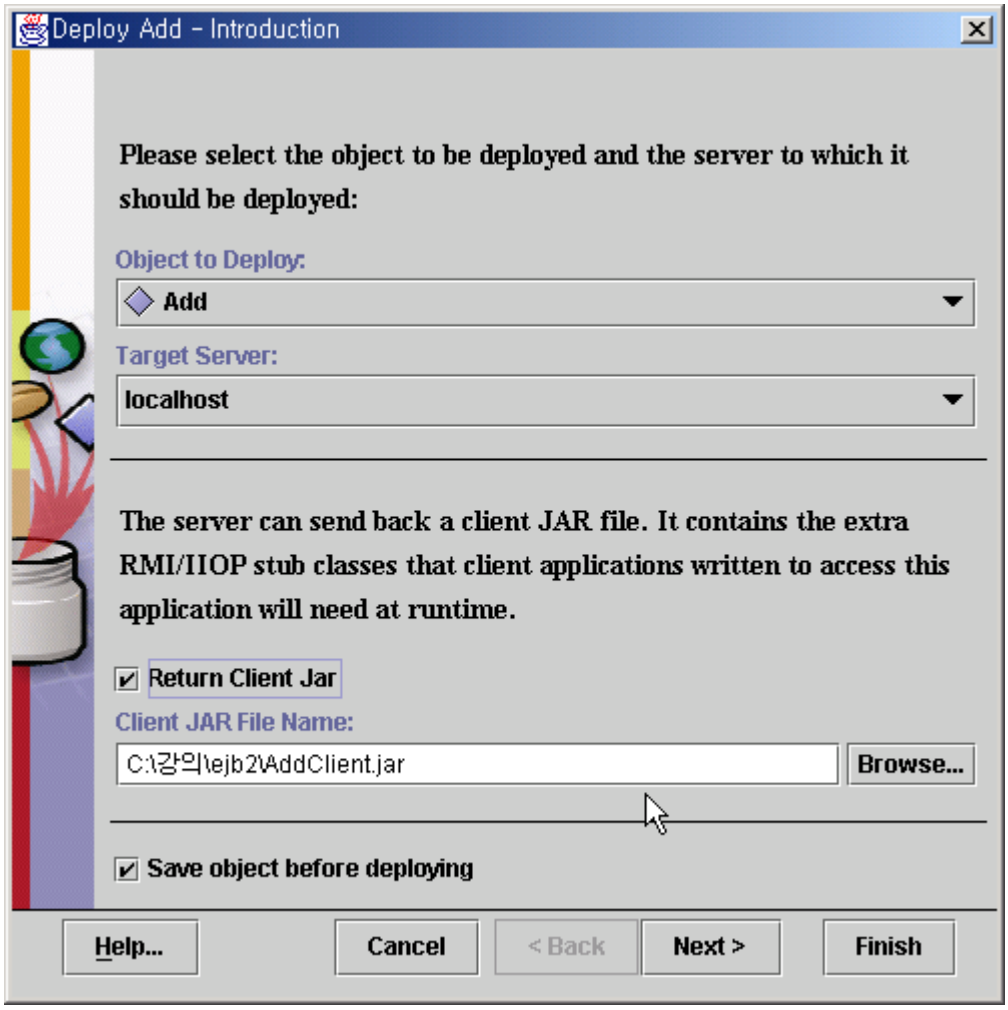

그림 44 Tools- deploy 를 선택한 후에 그림과 같이 셋팅합니다.

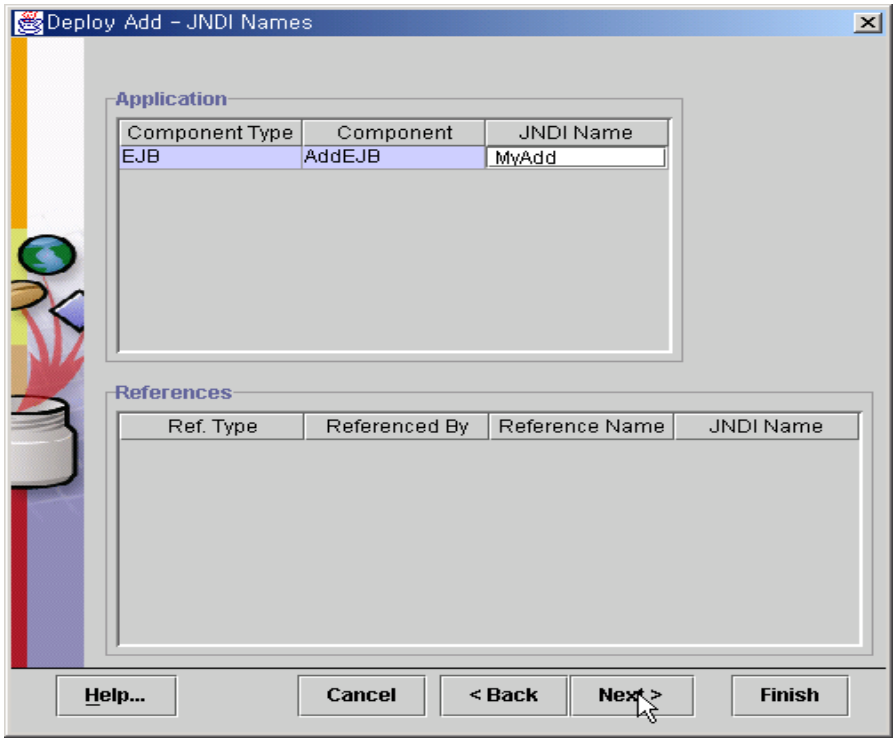

그림 45 JNDI Name을 MyAdd 로 선택합니다.

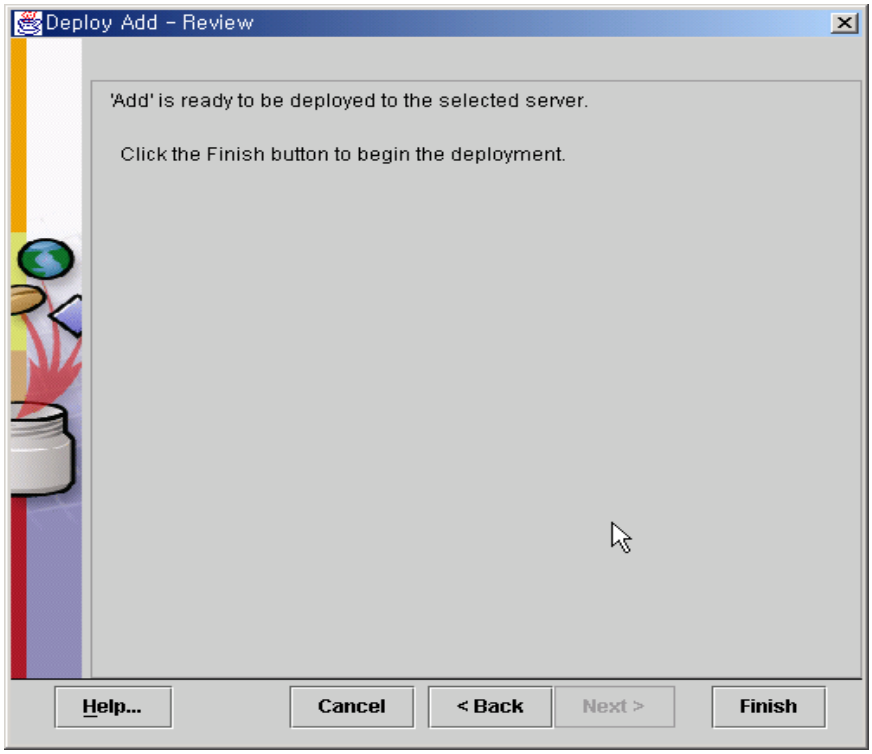

그림 46 finish 버튼을 누릅니다.

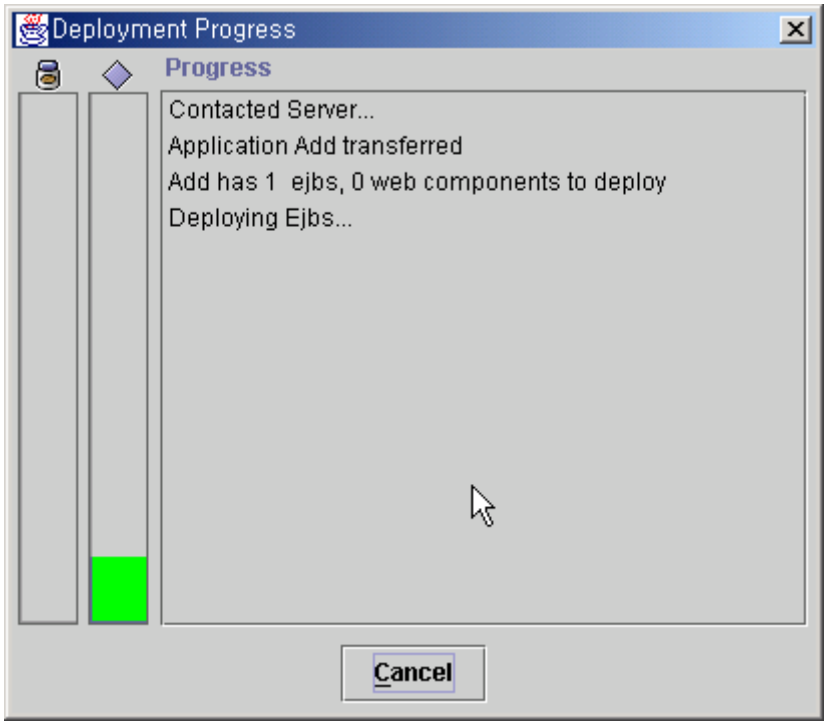

그림 47 deploy 되는 과정이 그래프와 함께 표시됩니다.

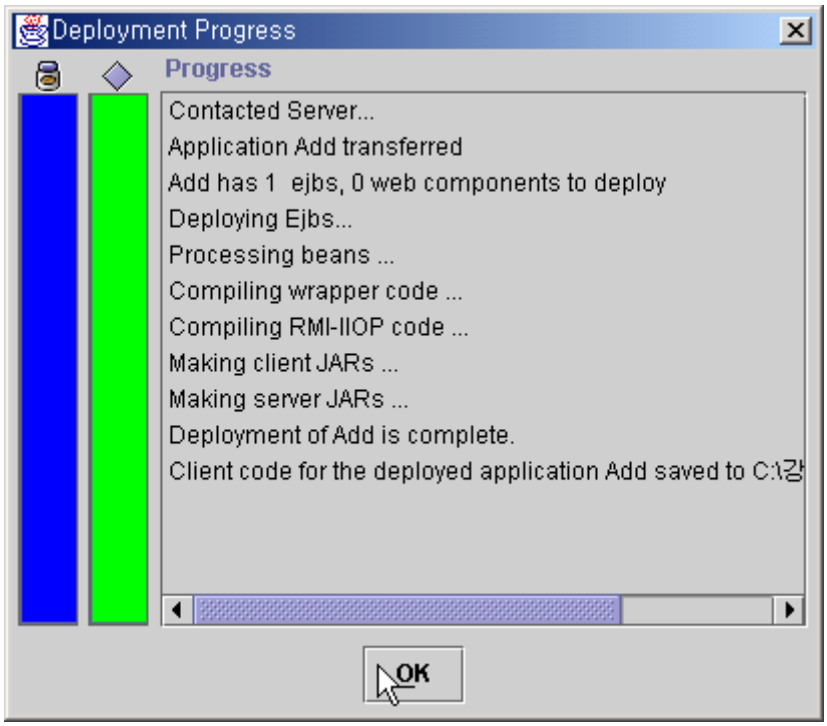

그림 48 Deploy 가 성공한 화면입니다. 이제 Add EJB를 사용할 수 있게 되었습니다.

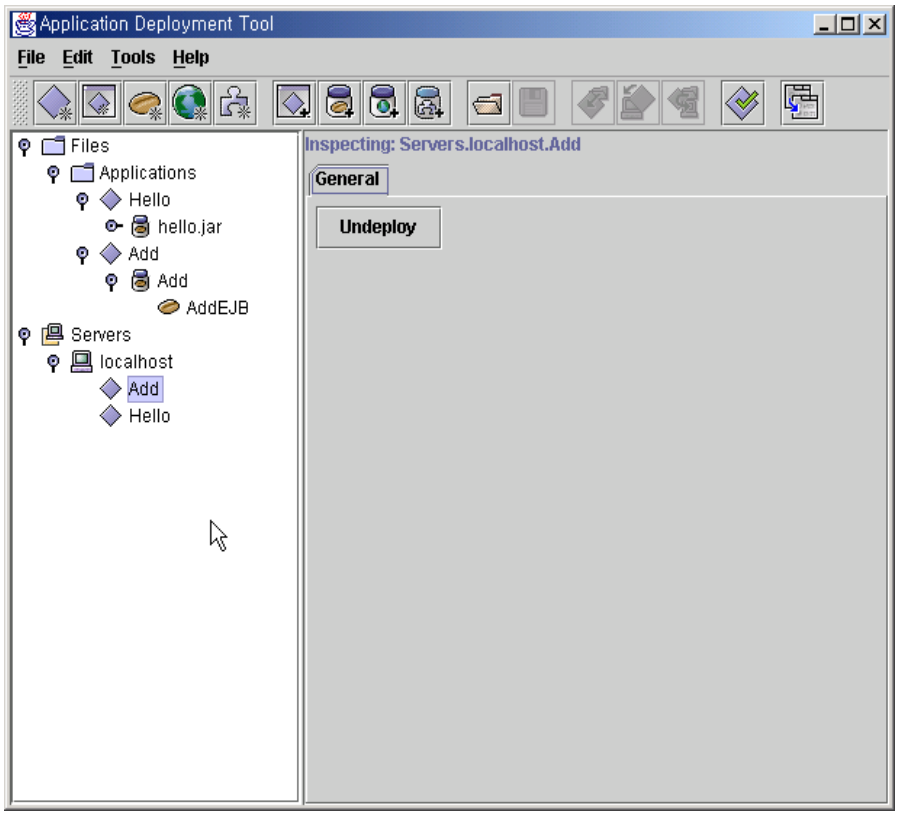

그림 49 서버에 설치된 Add객체입니다.

e. HTML파일과 JSP 파일의 작성

실제로 EJB객체를 이용하는 HTML과 JSP파일을 작성합니다. 설명은 주석문으로 대신합니다.

addform.html 시작 ----------

```
<html>
<head>
<title>EJB 와 덧셈</title>
</head>
<body>
<!—
  post 방식으로 add.jsp 를 호출합니다. add.jsp 는 EJB객체를 이용하는 jsp 파일입니다.
—>
<form method="post" action="add.jsp">
값 1: <input type="text" name="num1"><br>
값 2: <input type="text" name="num2"><br>
```

```
<input type="submit" value="계산">
</form>

</body>
</html>
```

addform.html 끝 ----------

add.jsp 에서 중요한 점은 홈인터페이스와 리모트 인터페이스를 import 하엿다는 것입니다. 패키지로 컴파일 하지 않았지만 반드시 import 하여야 jsp 파일이 서블릿으로 변환되면서 오류를 발생시키지 않습니다.

add.jsp 시작 ----------

```
<%@ page contentType = "text/html; charset=euc-kr" %>
<%@page import="javax.naming.*" %>
<%@page import="javax.rmi.PortableRemoteObject" %>
<%@page import="AddHome" %>
<%@page import="Add" %>
<html>
<head>
<title>EJB 와 덧셈</title>
</head>
<body>

<%
  /* addform.html로부터 2개의 문자열로 구성된 숫자값을 전달받아 Integer 클래스를 이용하여 정수화 하는 부분입니다. */
  int num1 = 0;
  int num2 = 0;
  try{
          num1 = Integer.parseInt(request.getParameter("num1"));
  }catch(Exception e){
          num1 = 0;
  }

  try{
          num2 = Integer.parseInt(request.getParameter("num2"));
  }catch(Exception e){
```

```
            num2 = 0;
        }

        Context initial = new InitialContext();
        // 세션빈의 객체 레퍼런스를 가지고 옵니다.
        Object obj = initial.lookup("MyAdd");

        // 홈 객체 레퍼런스를 가지고 옵니다.
        AddHome home = (AddHome)PortableRemoteObject.narrow(obj,
    AddHome.class);

        // EJB 객체를 생성합니다.
        Add a1 = home.create();
        out.println("결과값 :" + a1.getAdd(num1, num2));
    %>
    </body>
    add.jsp 끝 ------------------------------------------------
```

f. Web Componenet 만들기 (WAR 파일 생성)

EJB를 실제로 이용하는 JSP파일과 값을 입력 받는 HTML을 포함하는 Web Componenet 를 작성하도록 하겠습니다. 그림을 보고 잘 따라하세요.

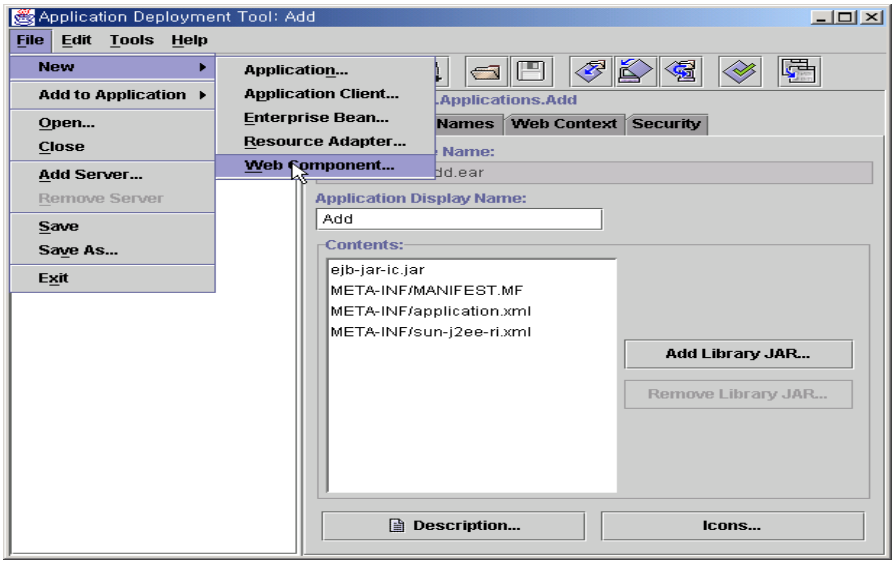

그림 50 File – New – Web Component... 를 선택합니다.

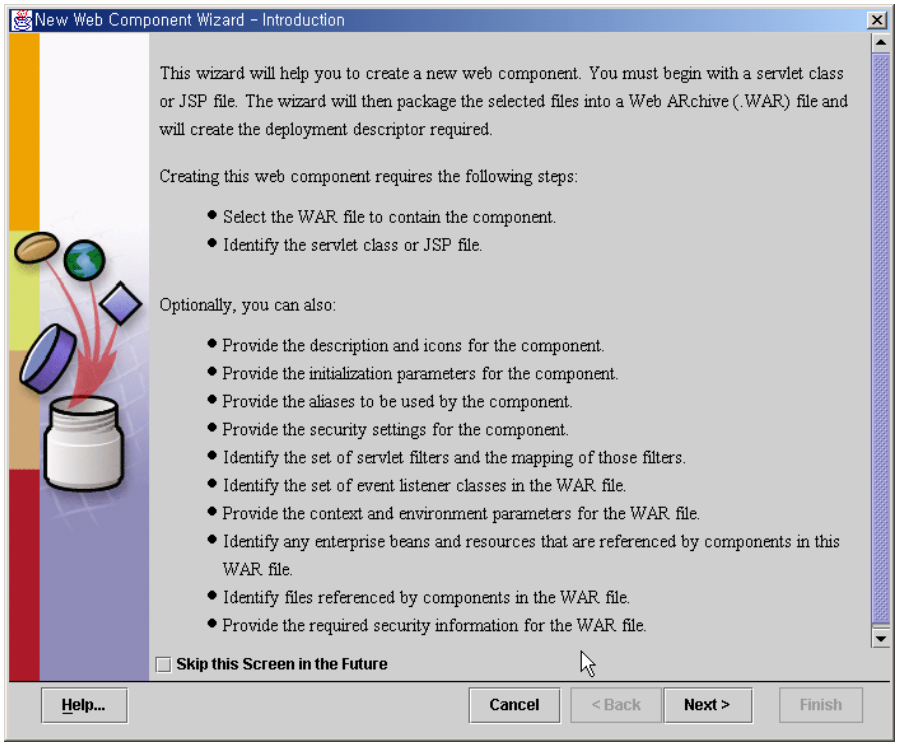

그림 51 설명을 잘 읽은 후 Next 버튼을 클릭합니다.

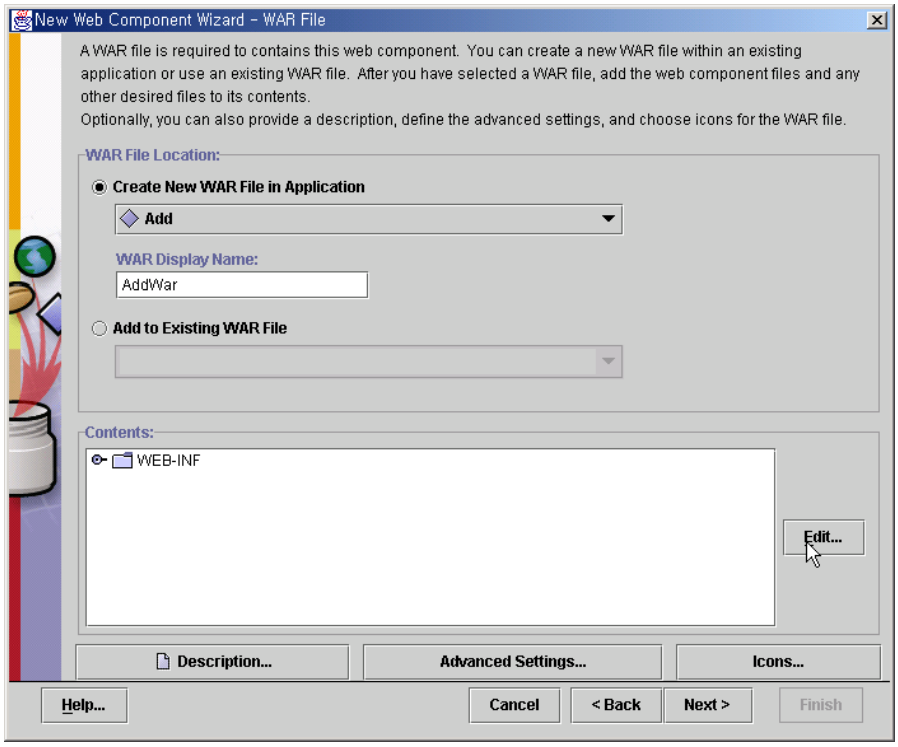

그림 52 그림과 같이 설정한 후 Edit 버튼을 클릭합니다.

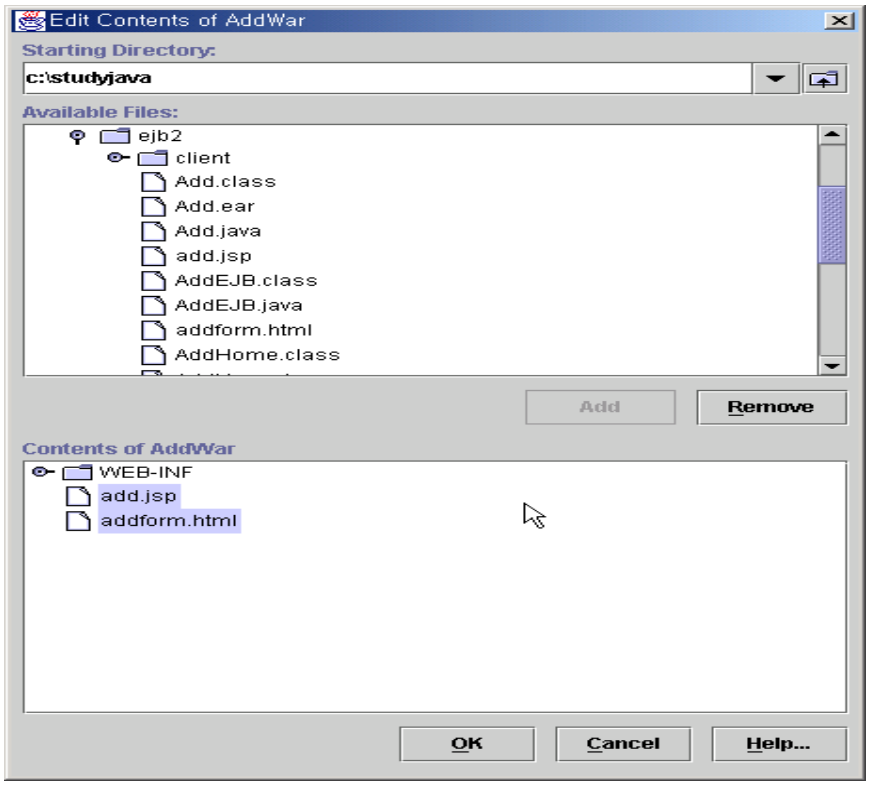

그림 53 추가할 html과 jsp를 선택한 후 Add 버튼을 클릭합니다.

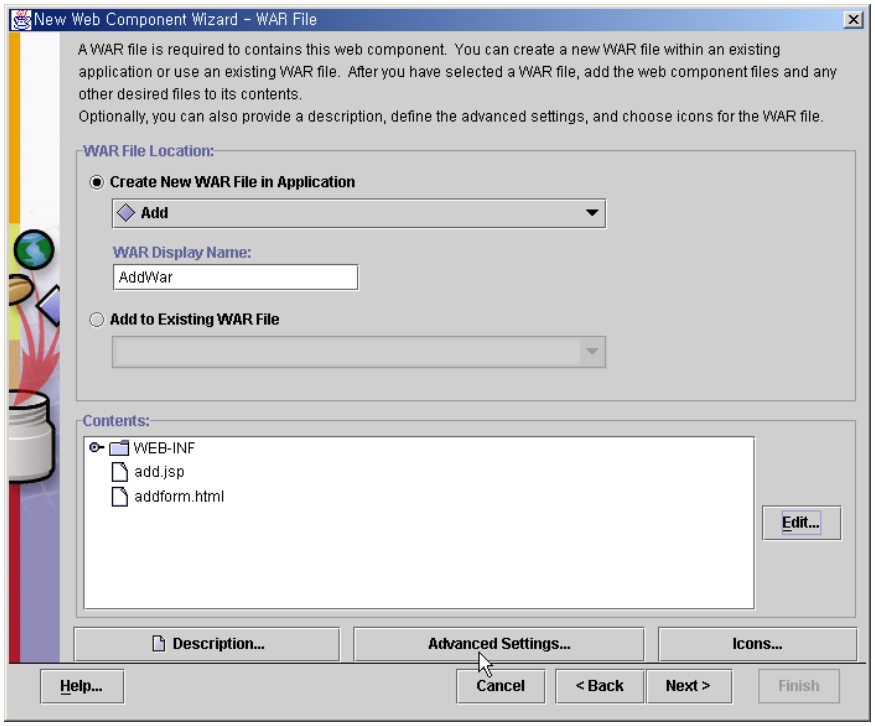

그림 54 확인 후 Next 버튼을 클릭합니다.

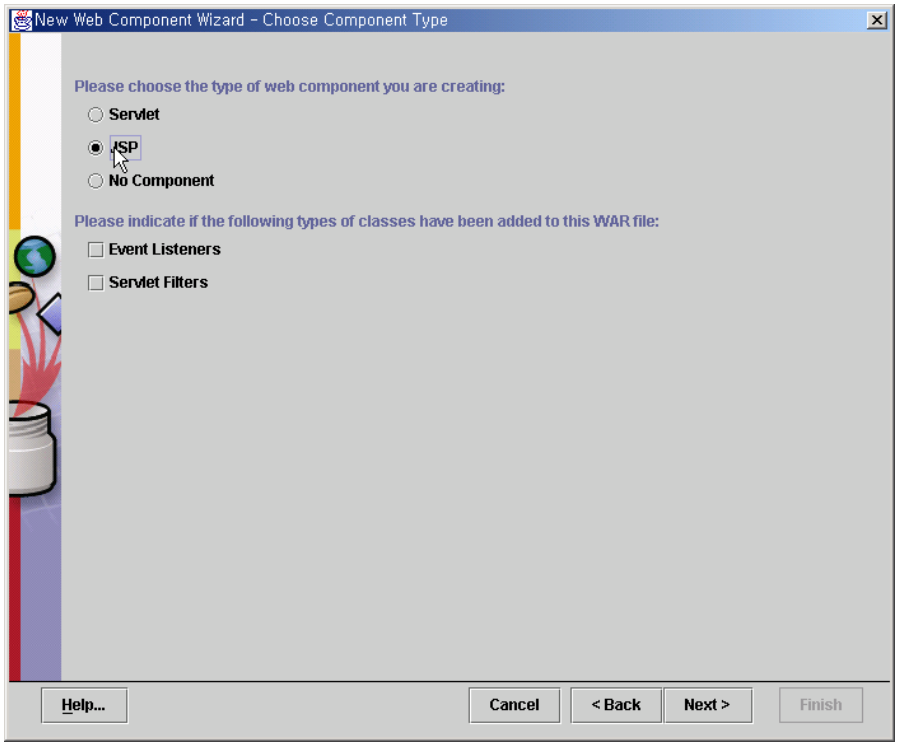

그림 55 jsp 파일로 구성되어있으므로 jsp 를 선택한 후 Next버튼을 클릭합니다.

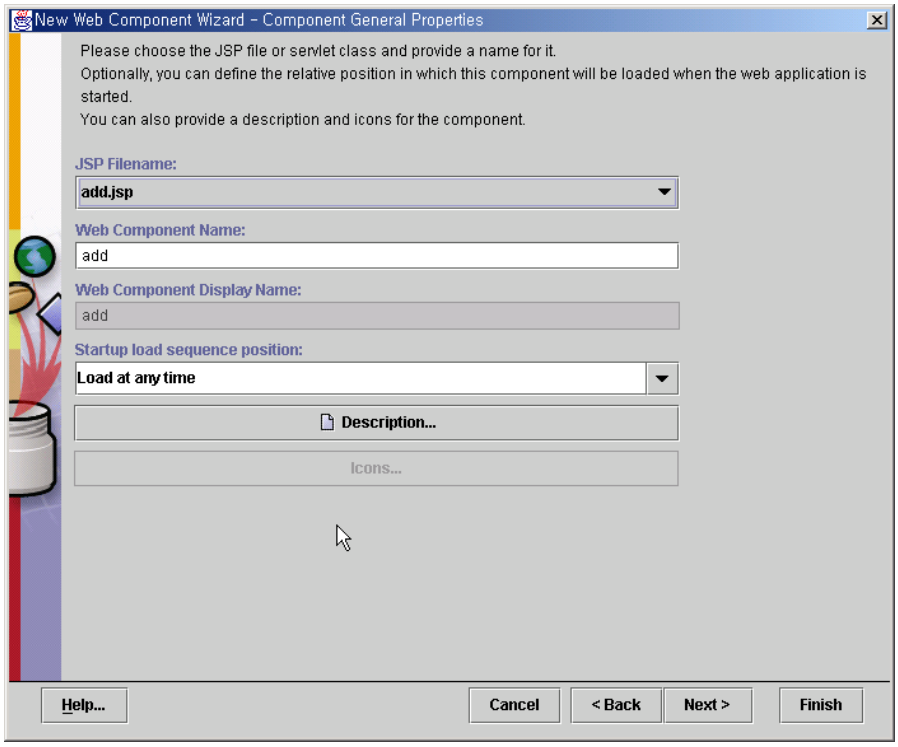

그림 56 그림과 같이 설정한 후 Next 버튼을 클릭합니다.

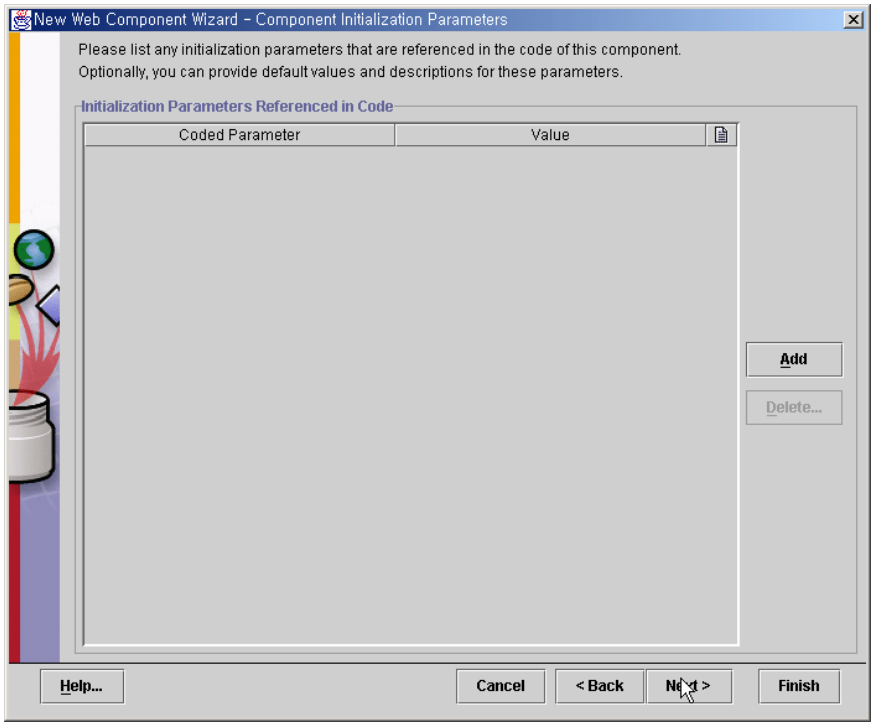

그림 57 Next 버튼 클릭.

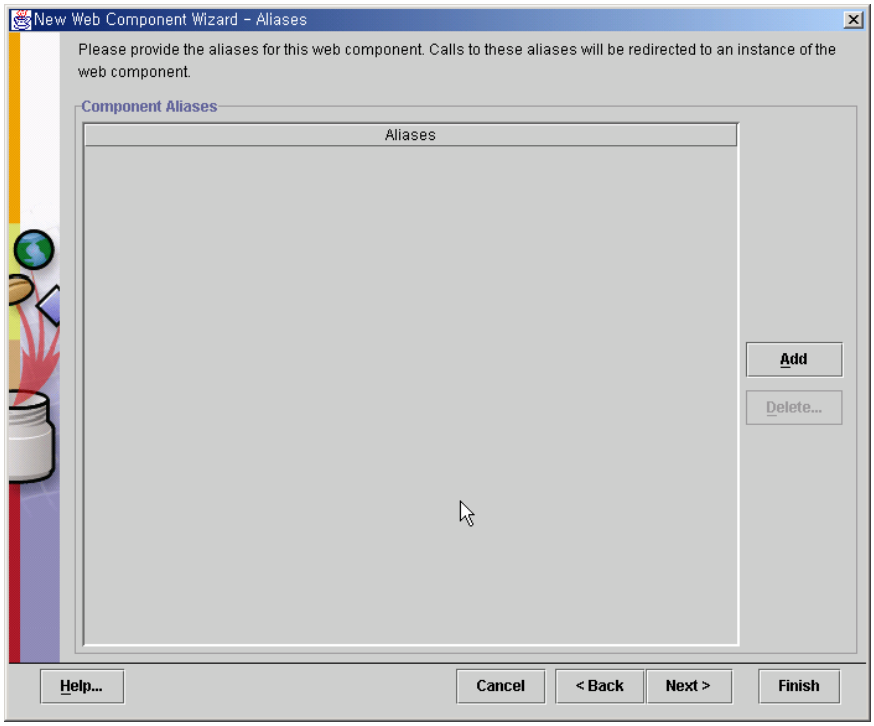

그림 58 Next 버튼 클릭

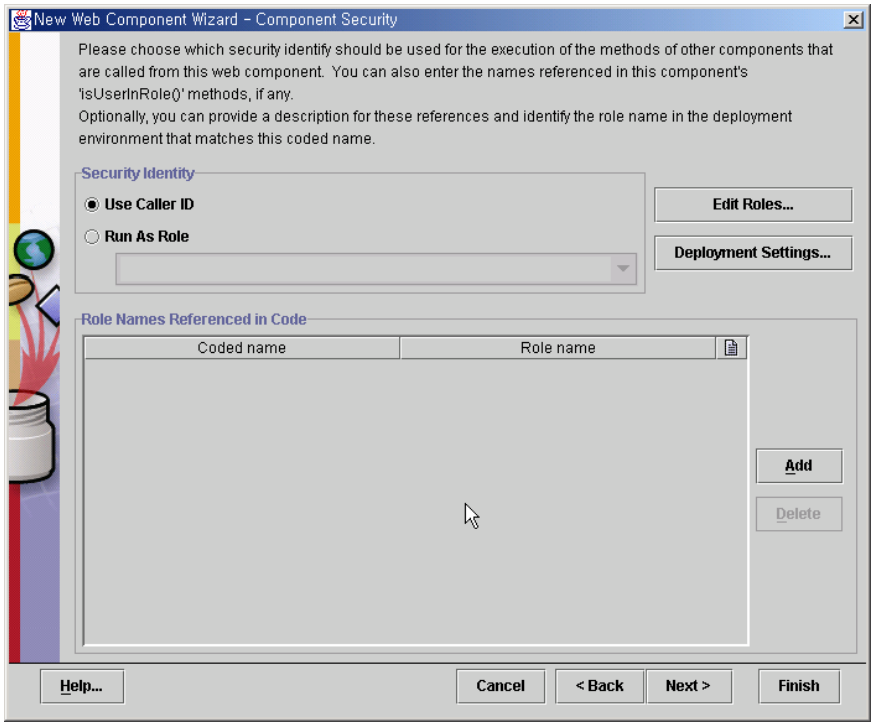

그림 59 Next 버튼 클릭

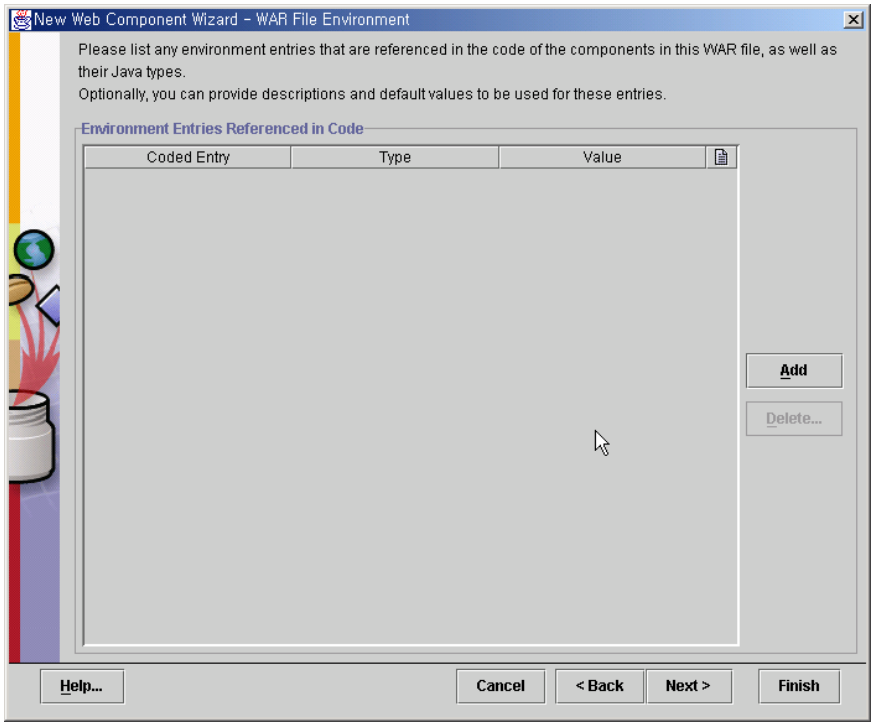

그림 60 Next 버튼 클릭

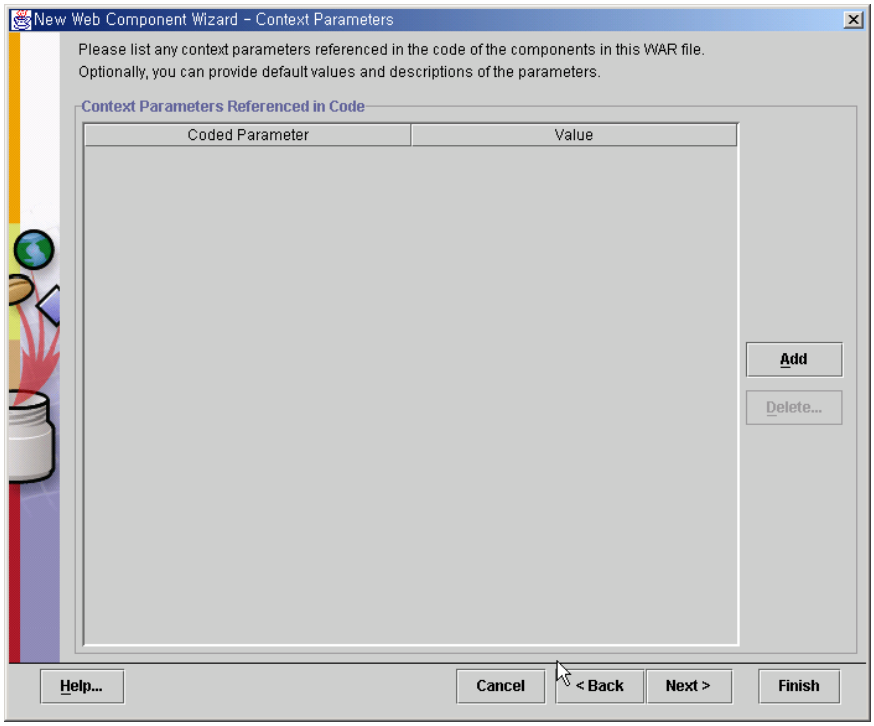

그림 61 Next버튼 클릭

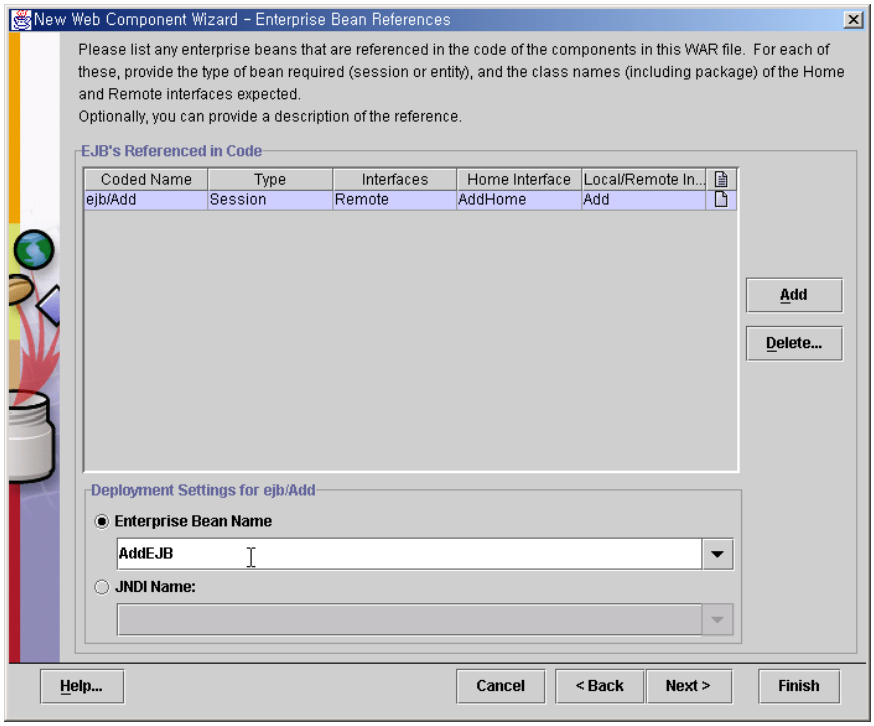

그림 62 그림과 같이 설정 후 Next버튼을 클릭합니다.

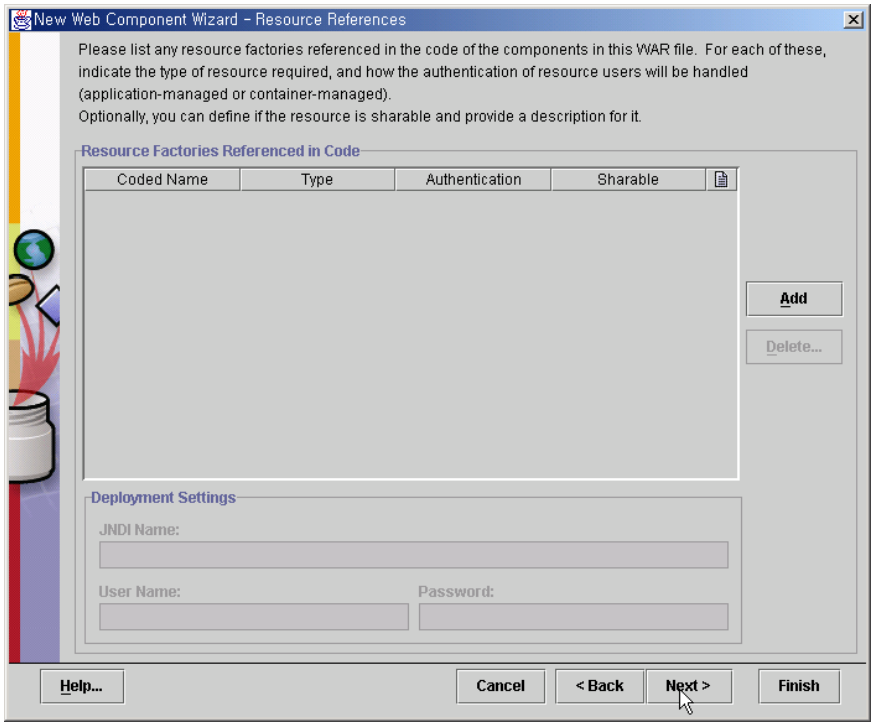

그림 63 Next버튼을 클릭합니다.

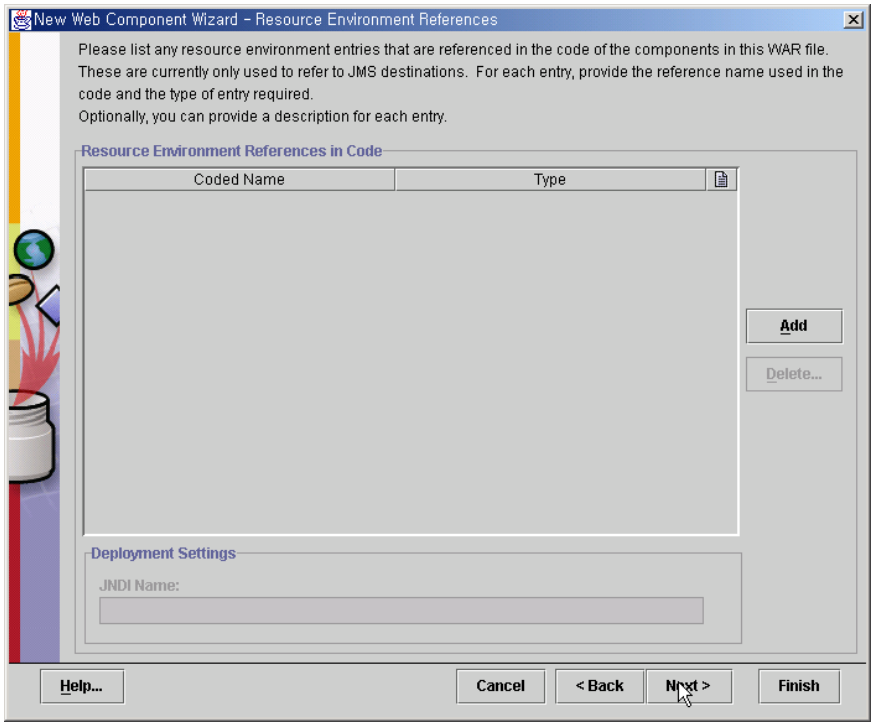

그림 64 Next 버튼을 클릭

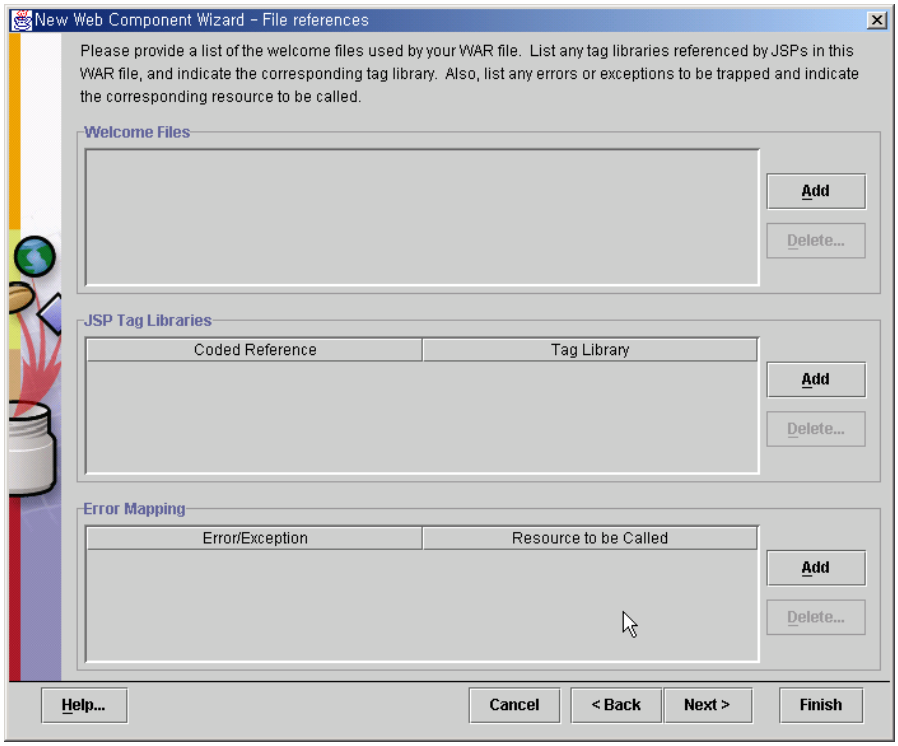

그림 65 Next 버튼을 클릭

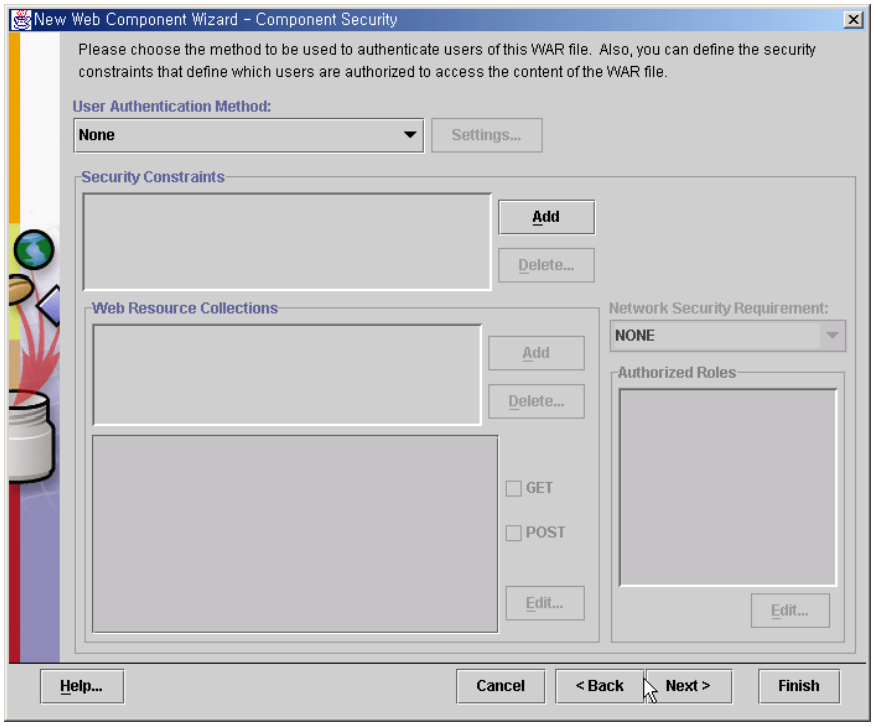

그림 66 Next버튼을 클릭

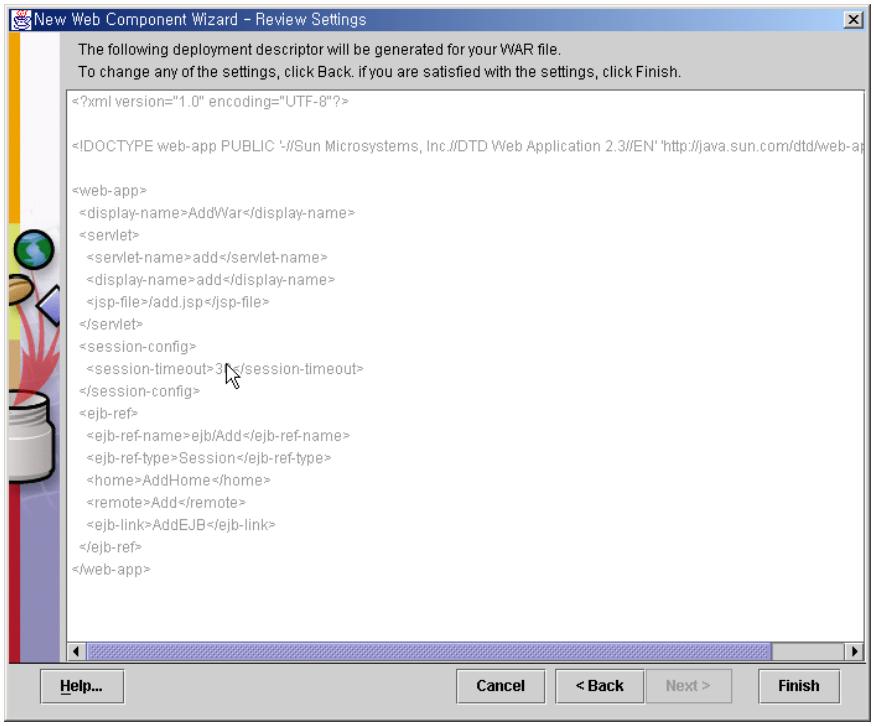

그림 67 Next버튼을 클릭하면 WAR 파일이 애플리케이션에 추가가 됩니다. 이제 다시 deploy를 실행하여 갱신을 합니다.

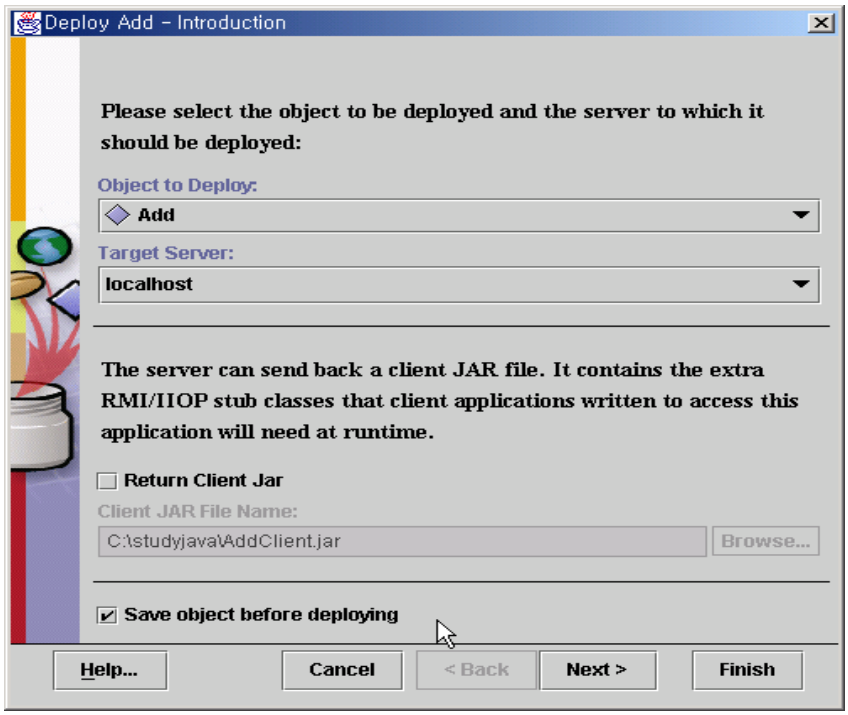

그림 68 Tool-Deploy를 선택한 후 그림과 같이 설정한 후 Next버튼을 클릭합니다.

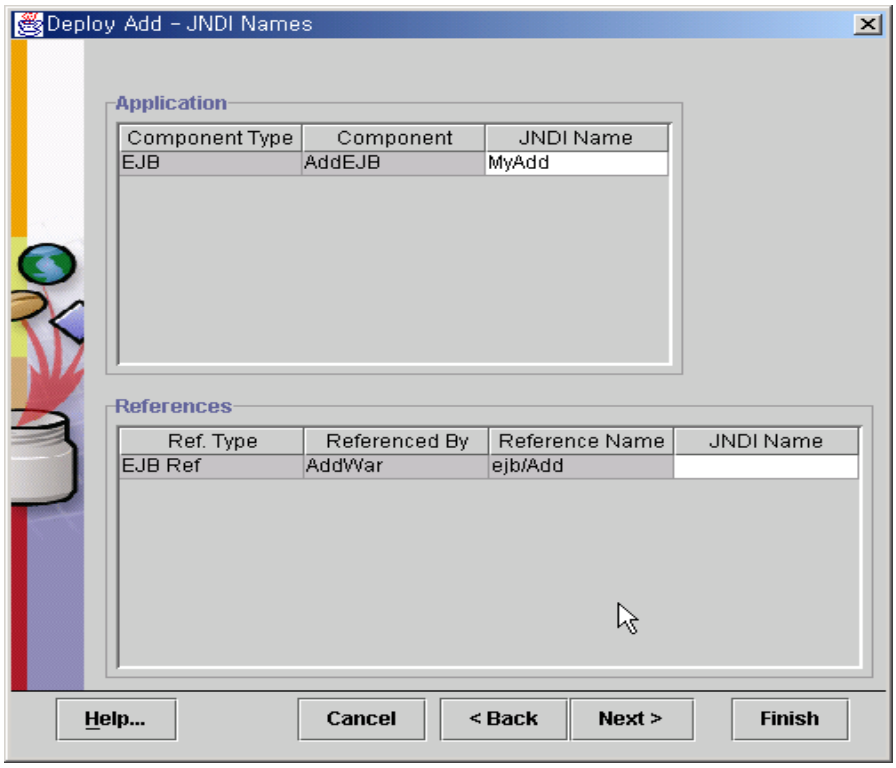

그림 69 Next버튼을 클릭합니다.

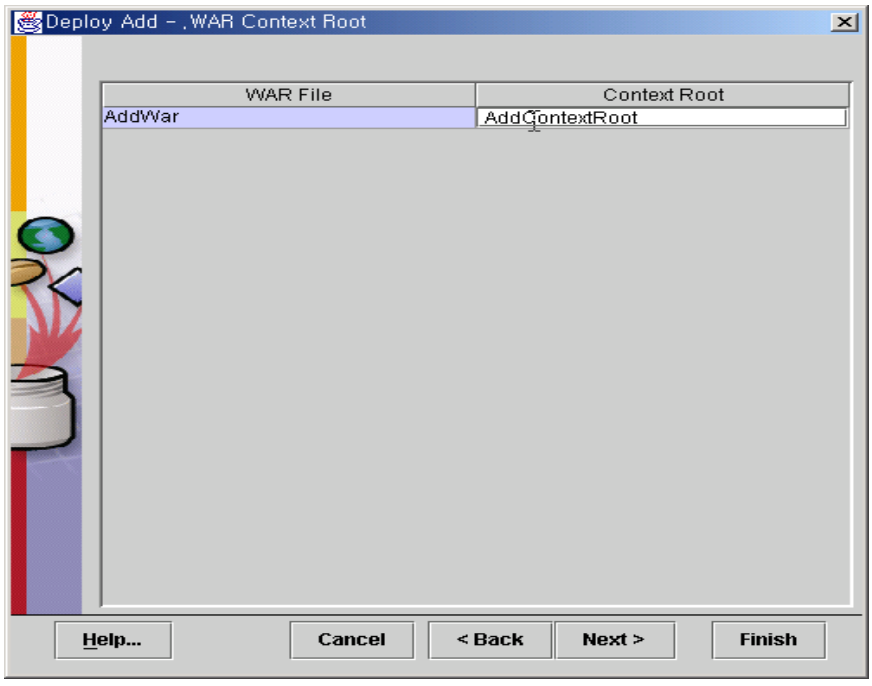

그림 70 Context Root를 지정합니다.

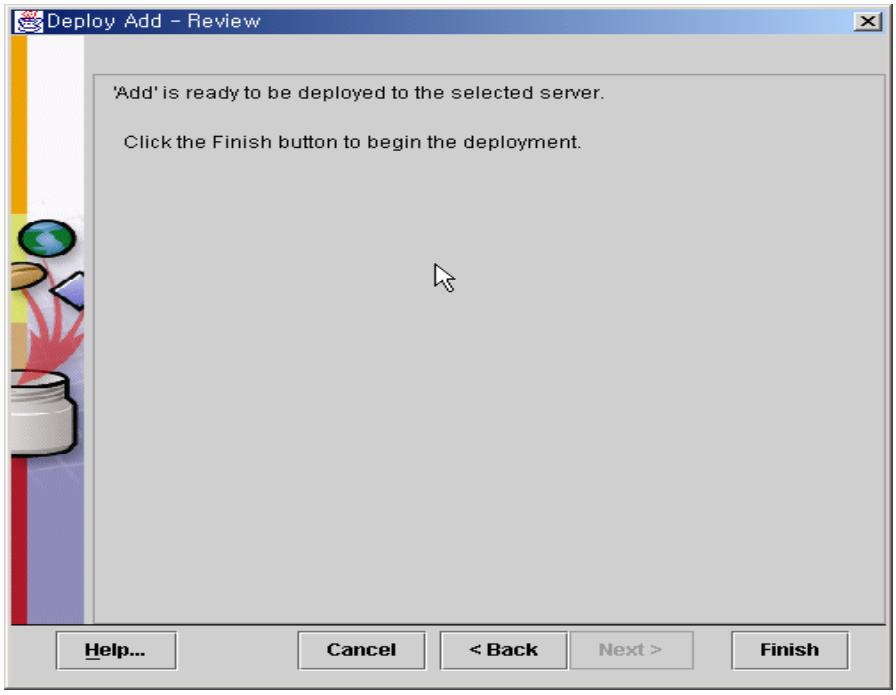

그림 71 Finish버튼을 누릅니다.

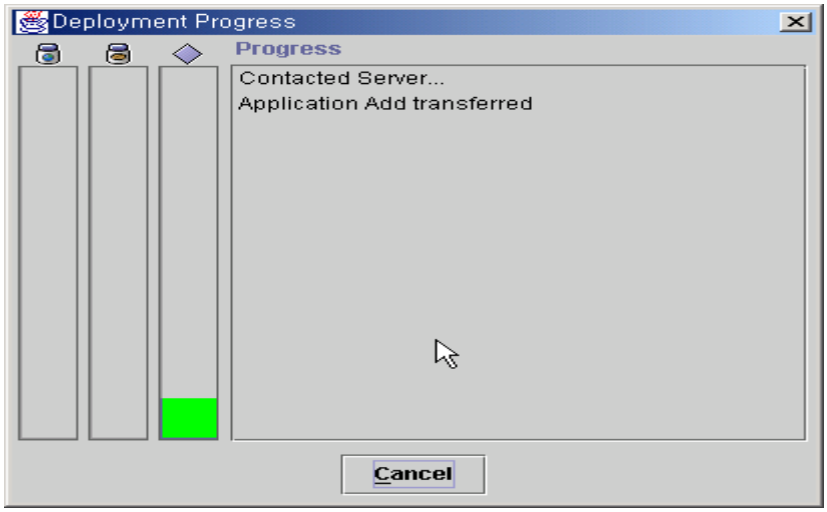

그림 72 Deploy되는 모습이 그래프와 함께 보여집니다.

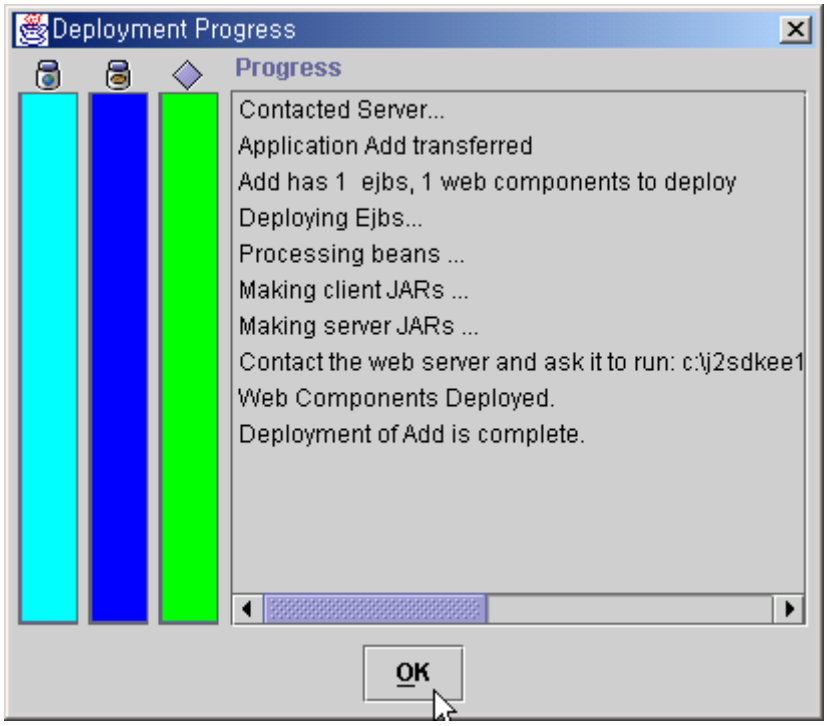

그림 73 Deploy가 끝났습니다. 이제 JSP파일로 테스트를 합니다.

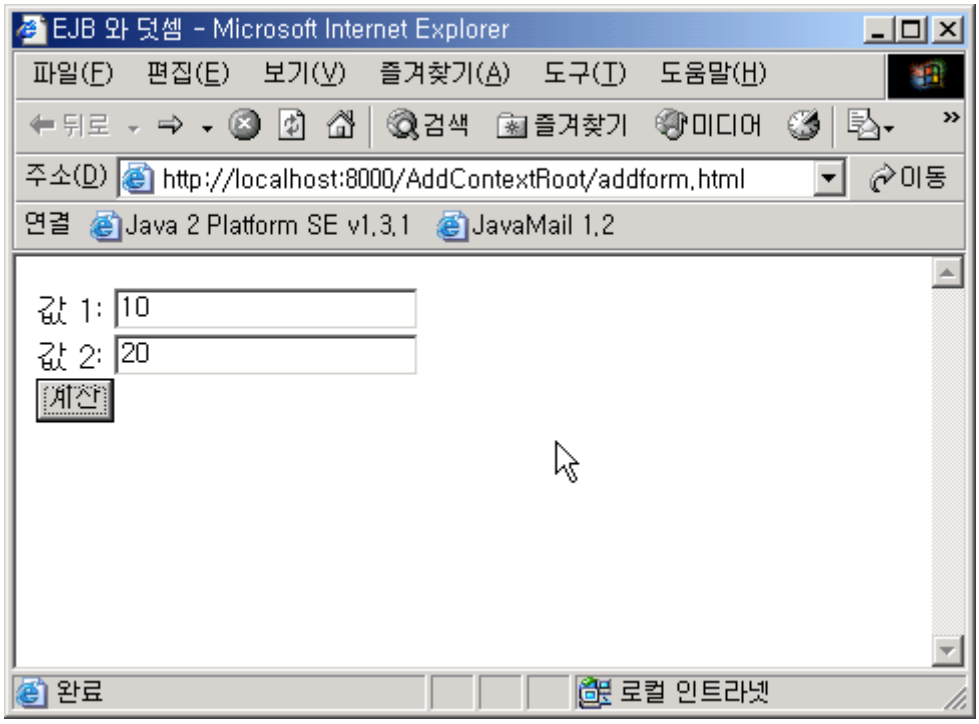

그림 74 기본적으로 J2EE의 JSP컨테이너는 8000번 포트를 이용합니다. URL을 보면 ContextRoot가 적혀있는 것을 볼 수 있습니다.

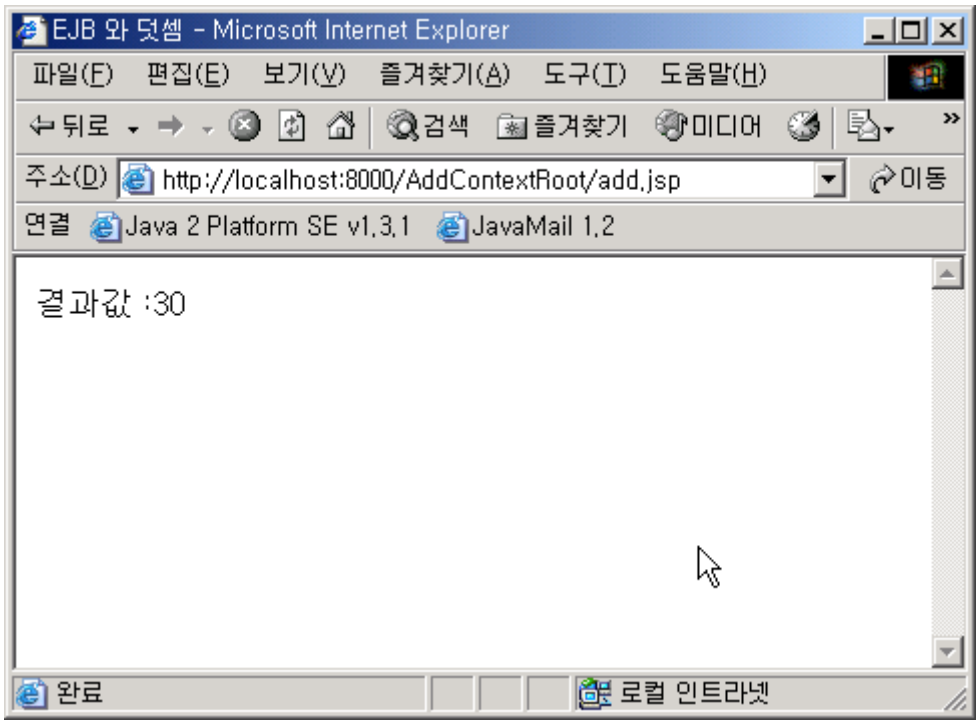

그림 75 입력된 값을 전달받아 계산 결과를 출력합니다. JSP파일에서 EJB를 이용하였습니다.

서블릿에서 EJB 객체를 이용 할려면 어떻게 해야 할까요?

9. 상태유지 세션빈

상태 유지 세션빈은 무상태 세션빈과는 다르게 한 클라이언트에 종속적인 특징을 가집니다. 그렇기 때문에 상태 유지 세션빈은 클라이언트의 생명주기와 같은 생명주기를 가집니다. 또한 상태 유지 세션빈은 활성화와 비활성화라는 상태를 가질 수가 있습니다. 상태유지 세션빈은 클라이언트와 생명주기가 같기 때문에 잘못 사용하게 되면 자원을 계속 하여 점유하고 있을 수가 있습니다. 그렇기 때문에 사용되지 않을 때에는 하드디스크 등의 저장장치에 저장되어진 상태로 요청이 올 때까지 비활성화 상태로 유지될 수 있습니다. 물론, 요청이 있다면 다시 사용 가능한 상태인 활성화 상태로 바뀌게 됩니다.

이번 단락에서는 간단한 계산기능을 가지고 있는 상태유지 세션빈을 작성하는 방법에 대하여 알아보도록 하겠습니다.

a. Home Interface의 작성

상태유지 세션빈은 정보를 유지할 수 있기 때문에 create 메소드에서 인자를 지정할 수 있습니다.

CalculatorHome.java 시작----------

```
import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.*;

public interface CalculatorHome extends EJBHome{
        Calculator create() throws RemoteException, CreateException;
        Calculator create(int number) throws RemoteException, CreateException;
}
```

CalculatorHome.java 끝 ----------

b. Remote Interface의 작성

상태유지 세션빈의 리모트 인터페이스는 무상태 세션빈의 리모트 인터페이스와 작성하는 방법이 동일합니다.

Calculator.java 시작 ----------

```
import javax.ejb.*;
import java.rmi.RemoteException;

public interface Calculator extends EJBObject{
        // 값을 추가하는 메소드
        void add(int number) throws RemoteException;
```

```
        void multiply(int number) throws RemoteException;
        void minus(int number) throws RemoteException;
        void divide(int number) throws RemoteException;
        int getNum() throws RemoteException;
}
```

Calculator.java 끝 ----------

c. 빈클래스의 작성

무상태 세션빈에서 작성하지 않았던 ejbCreate() 메소드를 구현합니다. 또한 활성화 메소드 ejbActivate() 메소드와 비활성화 메소드 ejbPassivate()메소드를 구현합니다.

CalculatorEJB.java 시작 ----------

```
import java.util.*;
import javax.ejb.*;

public class ENameEJB implements EntityBean{
        public String id;
        public String txt;
        private EntityContext context;

        public void setName(String txt){
                this.txt = txt;
        }

        public String getName(){
                return txt;
        }

        public String ejbCreate(String id,String txt) throws CreateException{
                if(id == null){
                        throw new CreateException("id is required.");
                }
                this.id = id;
                this.txt = txt;
                return null;
        }
```

```
        public void setEntityContext(EntityContext context){
                this.context = context;
        }

        public void ejbActivate(){
                id = (String)context.getPrimaryKey();
        }

        public void ejbPassivate(){
                id = null;
                txt = null;
        }

        public void ejbRemove(){}
        public void ejbLoad(){}
        public void ejbStore(){}
        public void unsetEntityContext(){}
        public void ejbPostCreate(String id, String txt){}
}
```

CalculatorEJB.java 끝 ----------

d. 디플로이먼트

그림을 보며 잘 따라하도록 합시다.

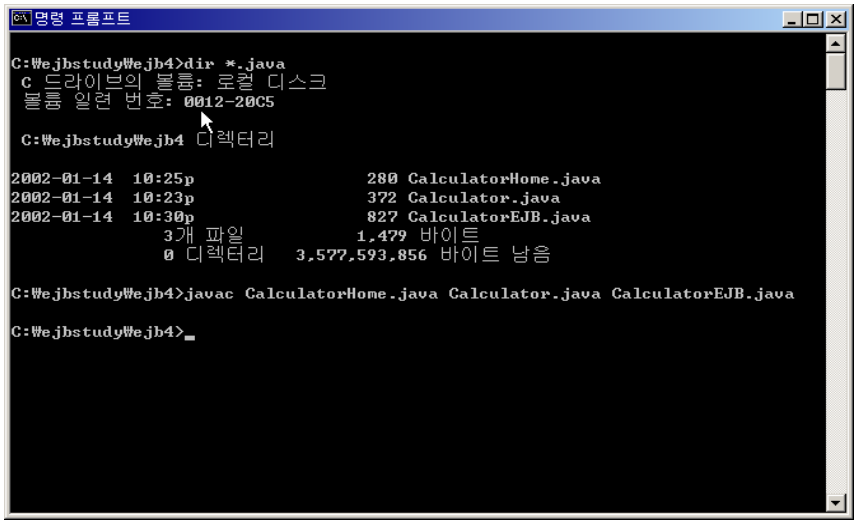

그림 76 홈 인터페이스, 리모트 인터페이스, 빈 클래스를 함께 컴파일 합니다.

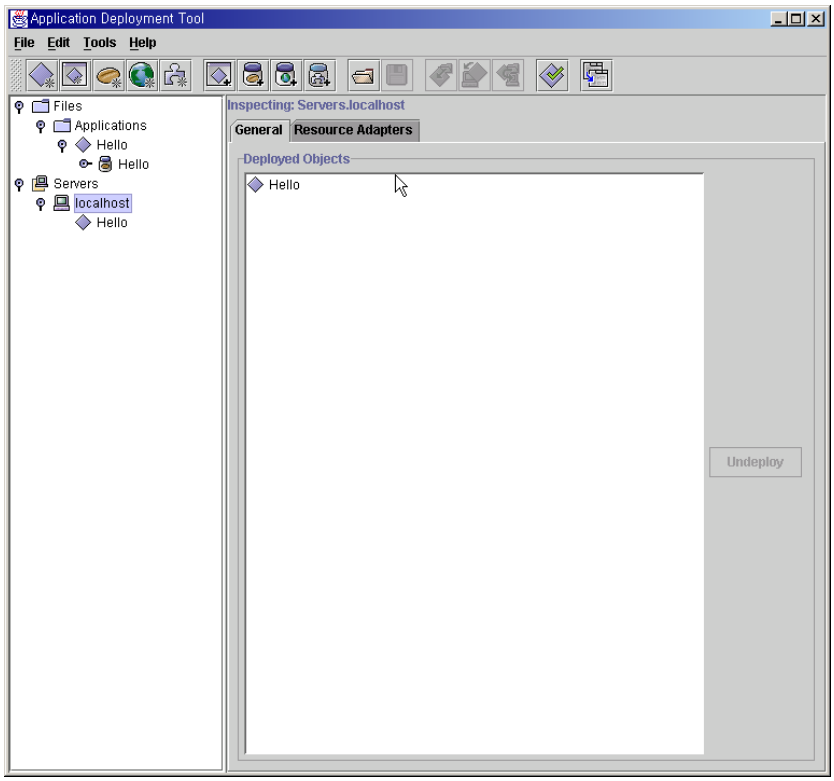

그림 77 deploytool 을 실행합니다.

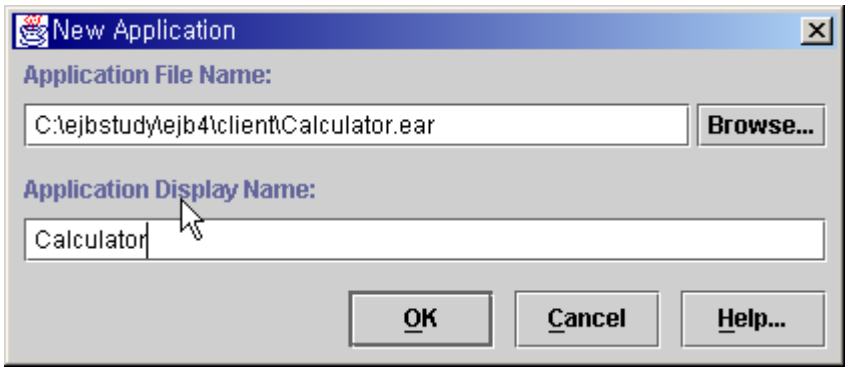

그림 78 File-New-Application 을 선택한 후 그림과 같이 셋팅합니다.

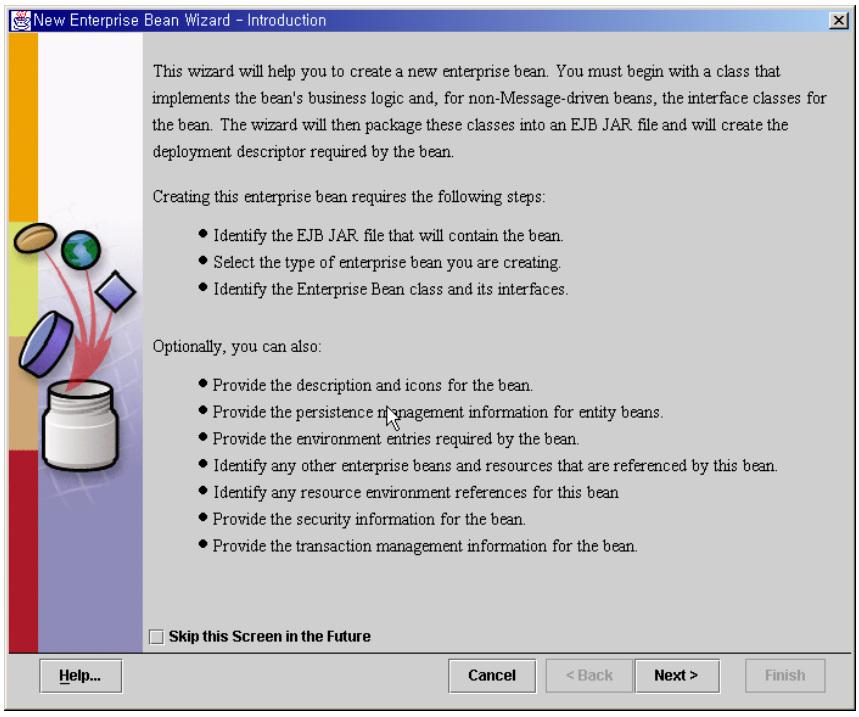

그림 79 File – new –EnterpriseBean 을 선택한 후 Next 를 누릅니다.

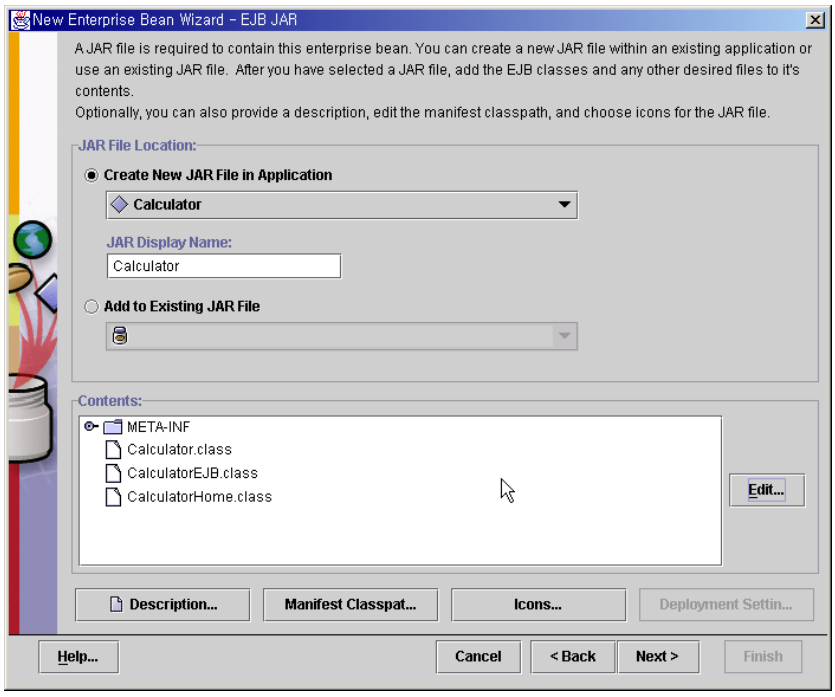

그림 80 그림과 같이 셋팅한 후 Next 버튼을 클릭합니다.

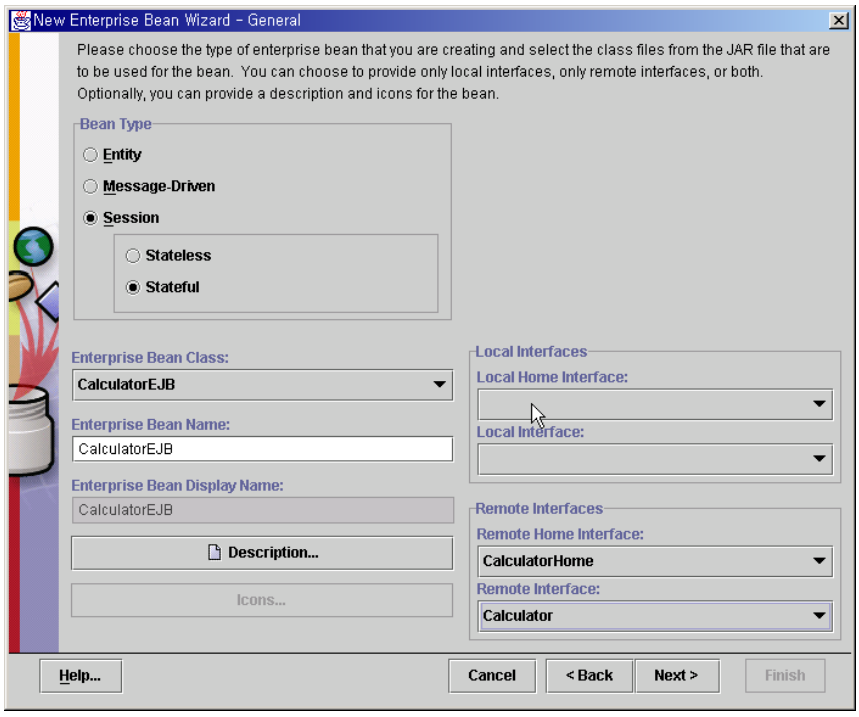

그림 81 상태유지 빈이므로 stateful 을 선택한 후 그림과 같이 셋팅합니다.

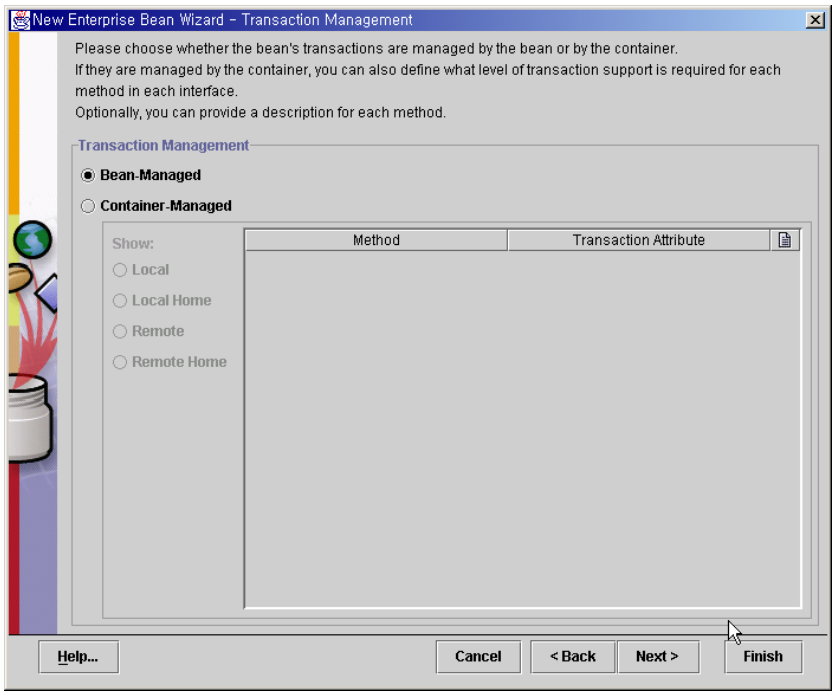

그림 82 Finish 버튼을 클릭합니다.

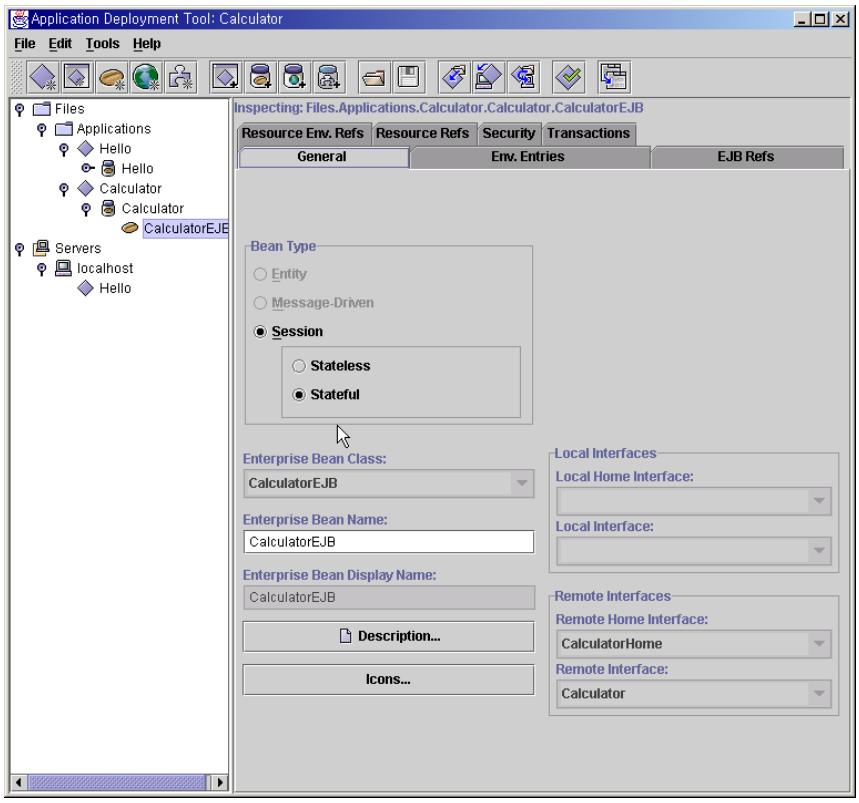

그림 83 그림과 같이 Application과 Enterprise Bean이 등록된 것을 알 수 있습니다.

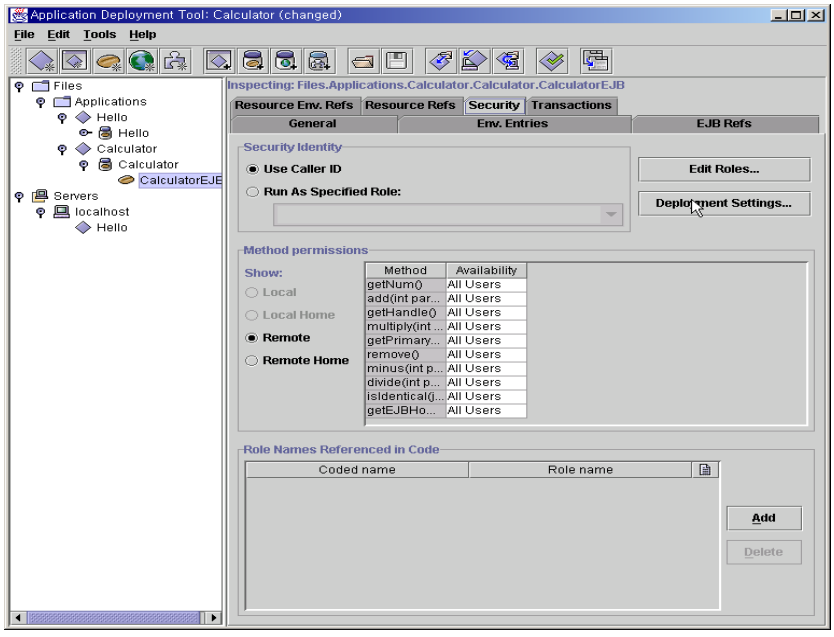

그림 84 Enterprise Bean의 Security Tab에서 Deployment Setting 을 설택합니다.

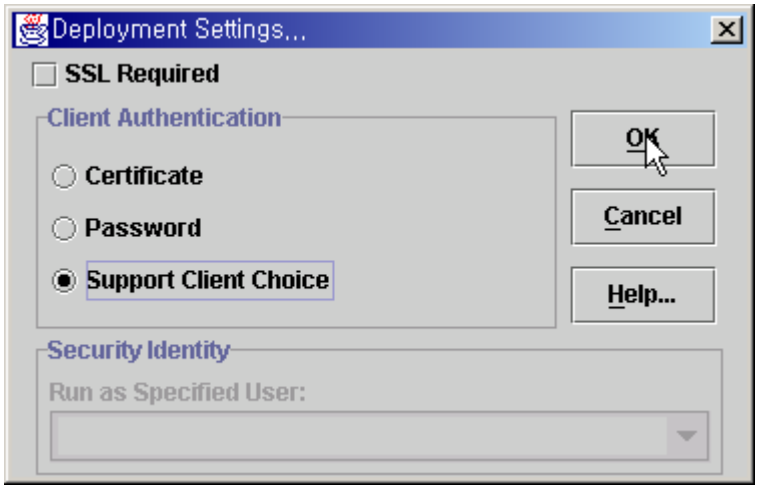

그림 85 Client Authentication 을 Support Client Choice로 바꿉니다.

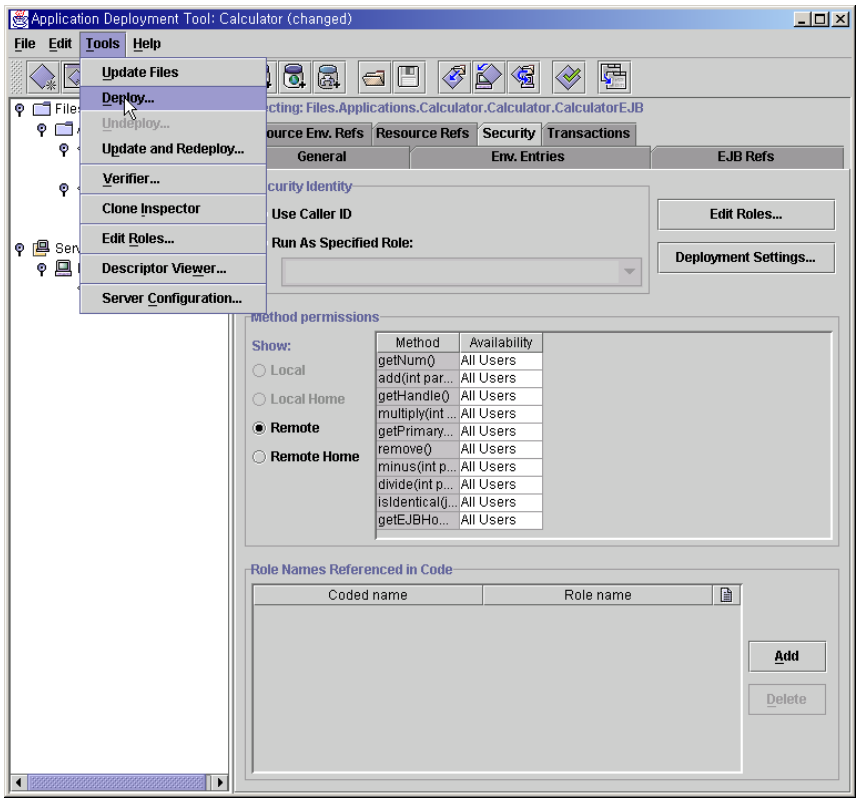

그림 86 Tools – Deploy를 선택합니다.

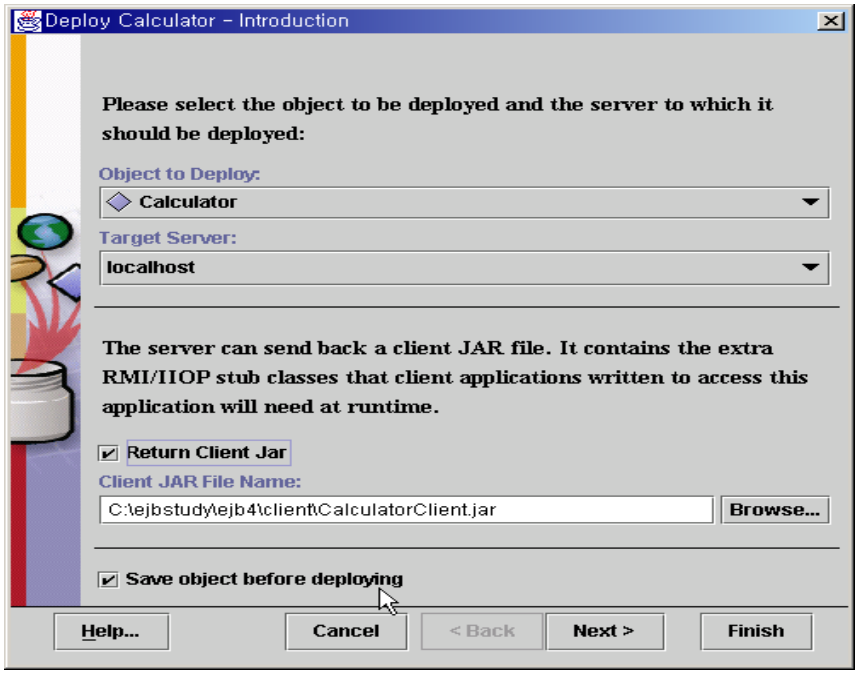

그림 87 그림과 같이 셋팅한 후 Next버튼을 클릭합니다.

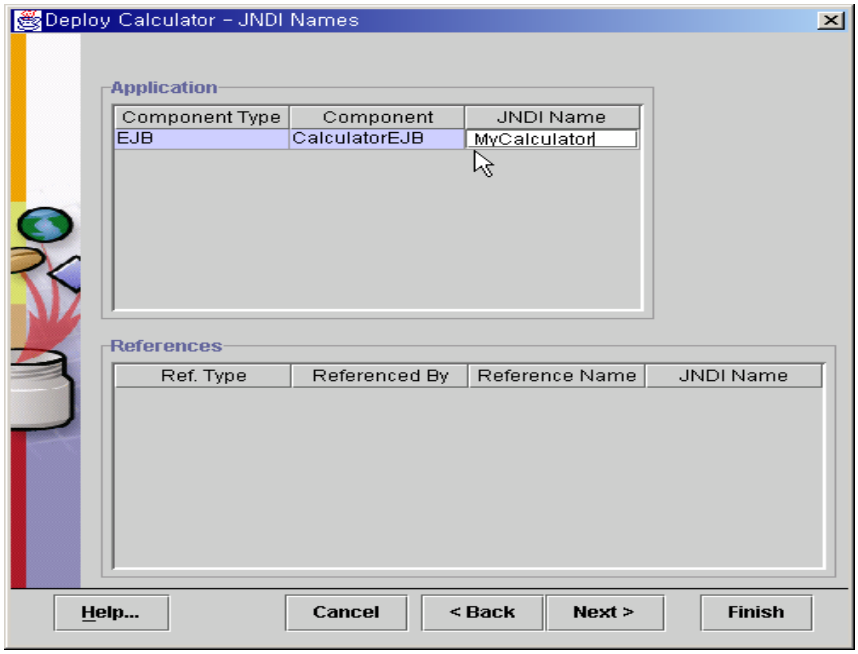

그림 88 JNDI Name을 그림과 같이 셋팅한 후 Next버튼을 클릭합니다.

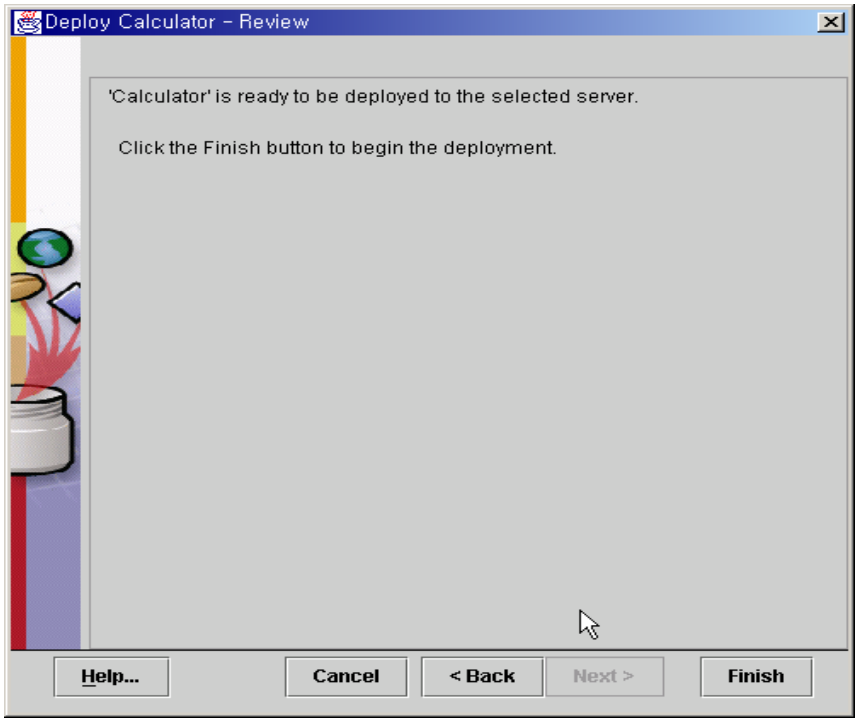

그림 89 Finish 버튼을 클릭합니다.

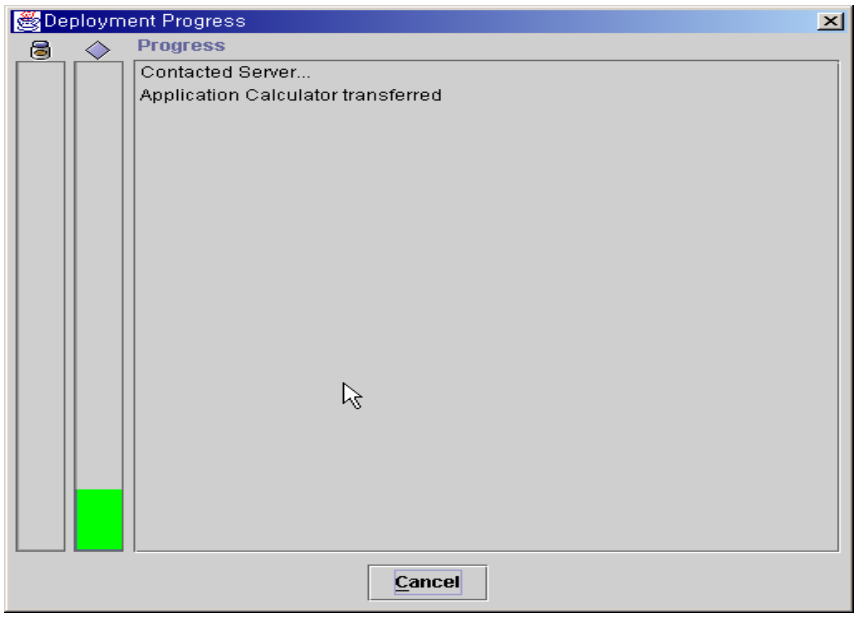

그림 90 Deploy되는 과정이 그래프와 함께 보여집니다.

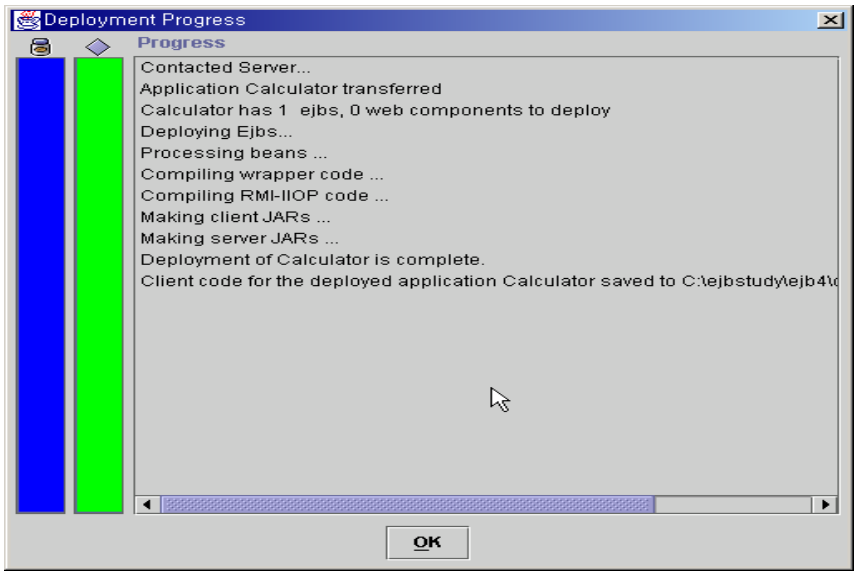

그림 91 디플로이가 종료한 화면입니다.

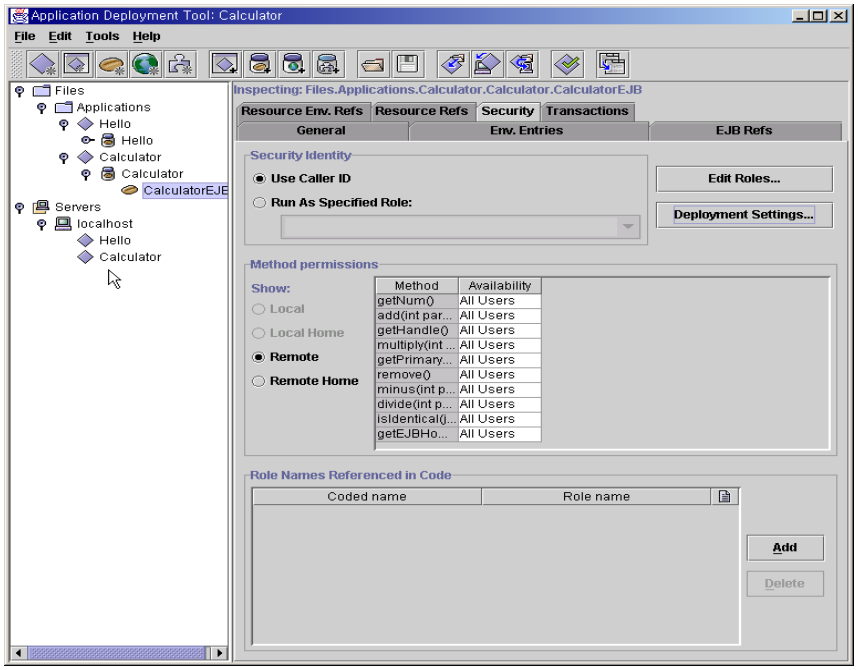

그림 92 Server에 Calculator 가 등록된 것을 알 수 있습니다.

e. 상태유지 세션빈을 이용하는 서블릿 프로그래밍의 작성과 WAR 작성

상태유지 세션빈은 클라이언트와 함께 운명을 같이 합니다. 그렇기 때문에 servlet 의 init 에서 한번 초기해준 후 맴버 변수로 지정을 하여 놓았습니다.

CalculatorServlet.java 시작 ----------

```
import java.io.*;
```

```
import javax.servlet.*;
import javax.servlet.http.*;
import javax.rmi.*;
import javax.naming.*;

import CalculatorHome;
import Calculator;

public class CalculatorServlet extends HttpServlet{
	Calculator calc;

	public void init() throws ServletException{
		try{
			// Initialcontext 생성
			InitialContext initial = new InitialContext();
			// 세션빈의 레퍼런스 획득
			Object obj = initial.lookup("MyCalculator");
			CalculatorHome home =
(CalculatorHome)PortableRemoteObject.narrow(obj, CalculatorHome.class);
			// EJB 객체생성
			calc = home.create(0);
		}catch(Exception e){
			e.printStackTrace();
		}
	} // end init

	public void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException{
		String snum = req.getParameter("num");
		String symbol = req.getParameter("symbol");
		if(snum == null) snum = "0";
		if(symbol == null) symbol ="+ ";
		int inum = 0;
		try{
			inum = Integer.parseInt(snum);
		}catch(Exception e){
```

```
                inum = 0;
            }

            if(symbol.equals("*")){
                calc.multiply(inum);
            }else if(symbol.equals("+ ")){
                calc.add(inum);
            }else if(symbol.equals("-")){
                calc.minus(inum);
            }else if(symbol.equals("/")){
                calc.divide(inum);
            }

            int total = calc.getNum();

            res.setContentType("text/html; charset=KSC5601");
            PrintWriter out = res.getWriter();
            out.println("<html><head><title>계산기</title></head><body>");
            out.println("<h1>계산기</h1>");
            out.println("<form method='get' action='CalculatorServlet'>");
            out.println("덧셈<input type='radio' name='symbol' value='+'
checked> ");
            out.println("뺄셈<input type='radio' name='symbol' value='-'> ");

            out.println("곱셈<input type='radio' name='symbol' value='*'> ");

            out.println("나눗셈<input type='radio' name='symbol'
value='/'> ");
            out.println("값 : <input type='text' name='num'>");
            out.println("<input type='submit' value='계산하기'><br>");
            out.println("계산결과 : " + total);
            out.println("</form></body></html>");
        } // end doGet
}
CalculatorServlet.java 끝 ----------------------------------------
```

아래의 과정은 해당 서블릿을 WAR파일로 만드는 과정입니다. 그림을 잘 보고 따라하도록 합시다.

환경변수로 J2EE_CLASSPATH 를 만들고 오라클 JDBC드라이버인 classes12.zip 파일을 지정하여 줍니다.

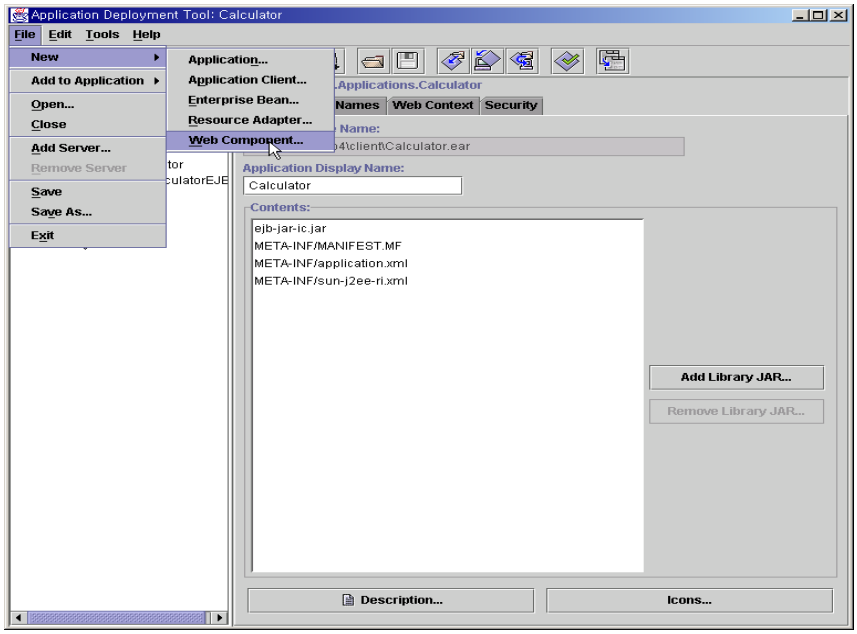

그림 93 File – New – Web Component 를 선택합니다.

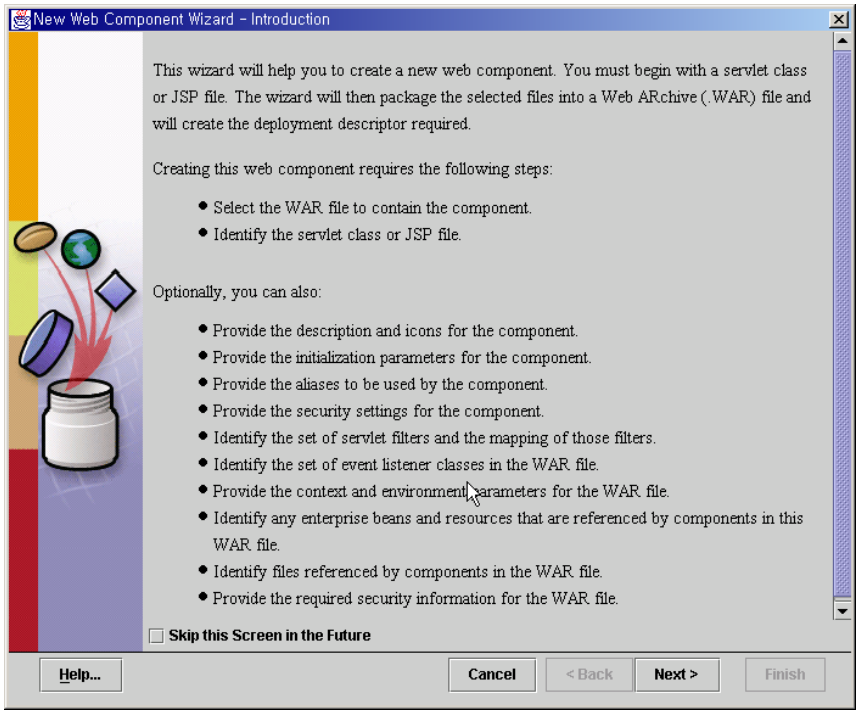

그림 94 설명문을 잘 읽은 후 Next 버튼을 클릭합니다.

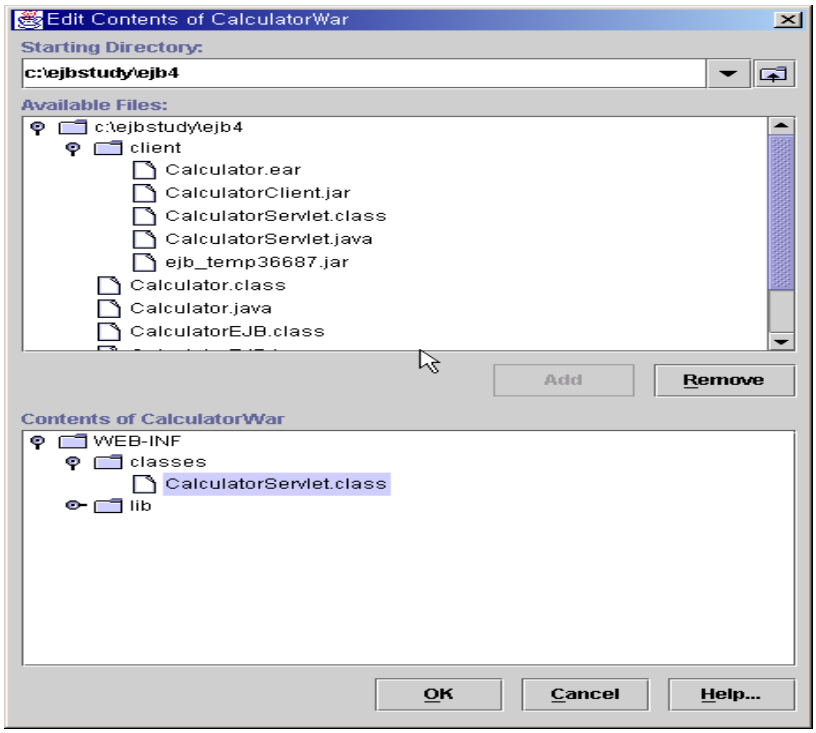

그림 95 WAR파일에 추가할 Servlet파일을 선택합니다.

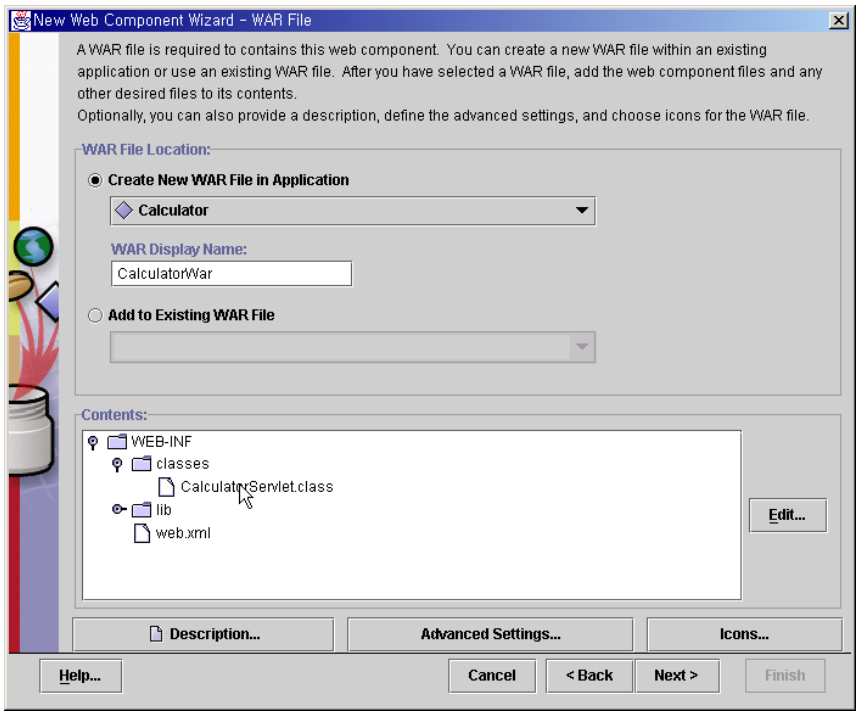

그림 96 Servlet파일이 추가된 모습입니다. 그림과 같이 셋팅 한 후 Next버튼을 클릭합니다.

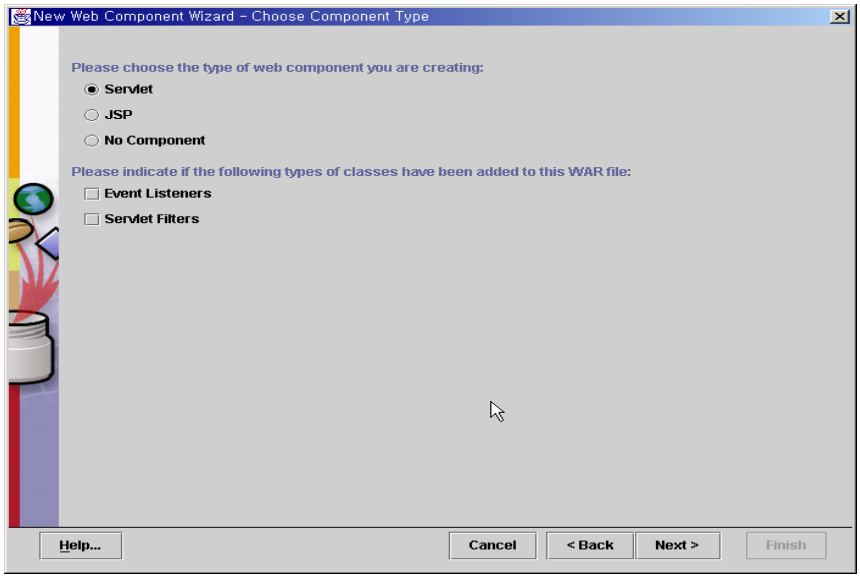

그림 97 추가할 내용이 Servlet이므로 Servlet을 선택 Next버튼을 클릭합니다.

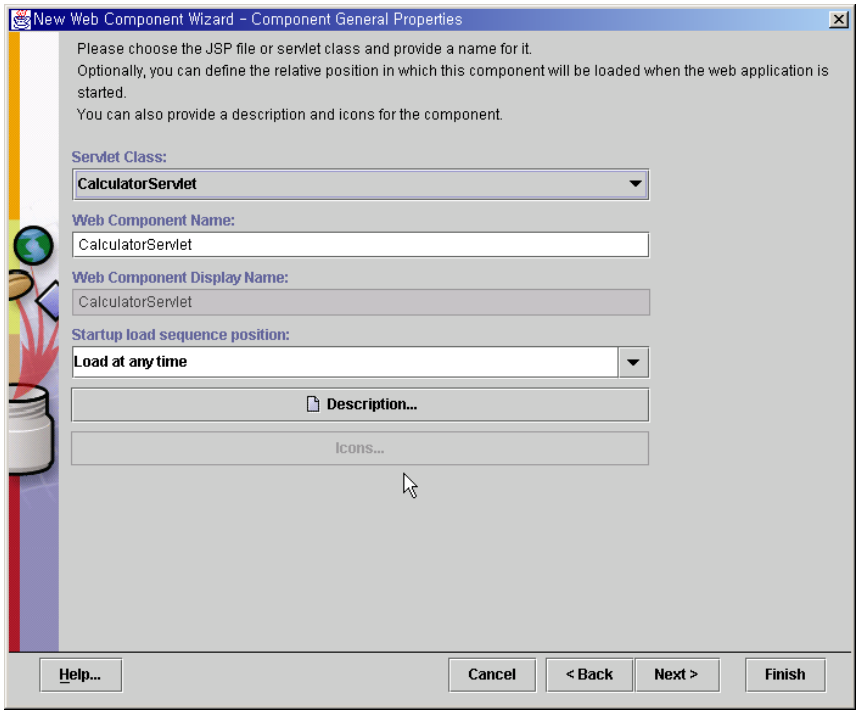

그림 98 그림과 같이 셋팅 Next 버튼을 클릭합니다.

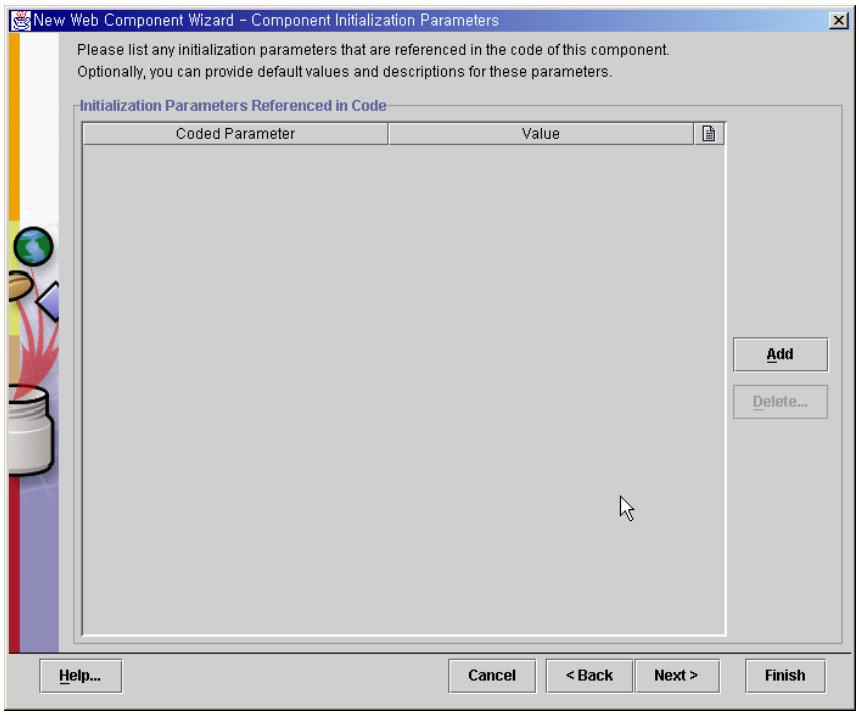

그림 99 Next버튼을 클릭합니다.

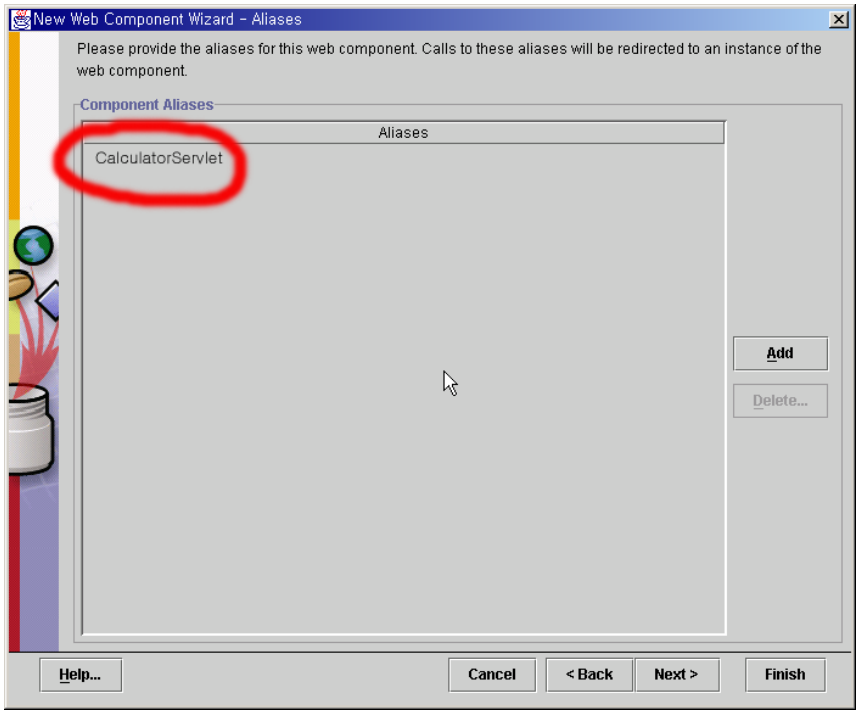

그림 100 Aliases에서 별명을 등록합니다.

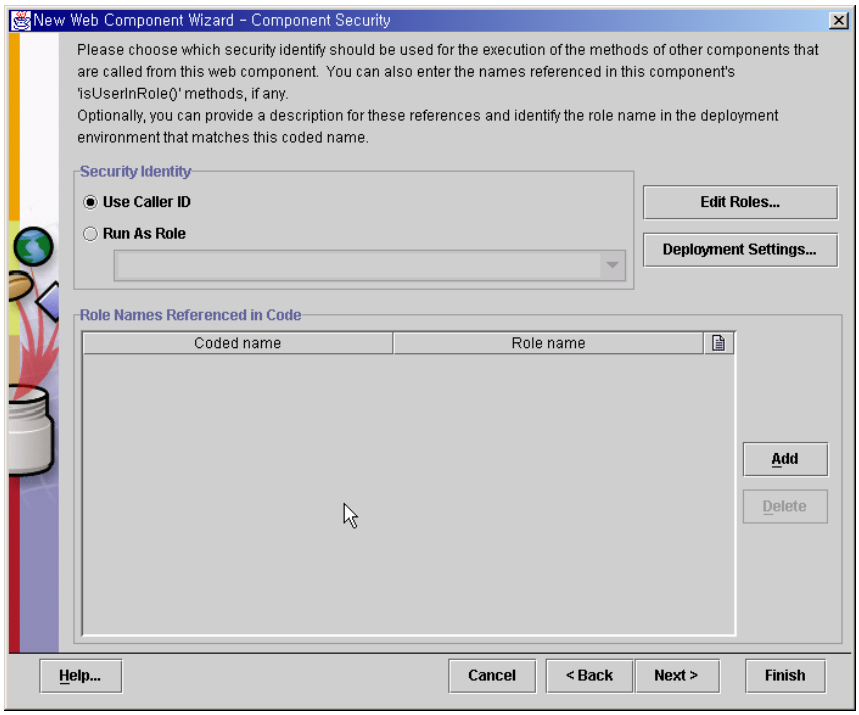

그림 101 Next 버튼을 클릭합니다.

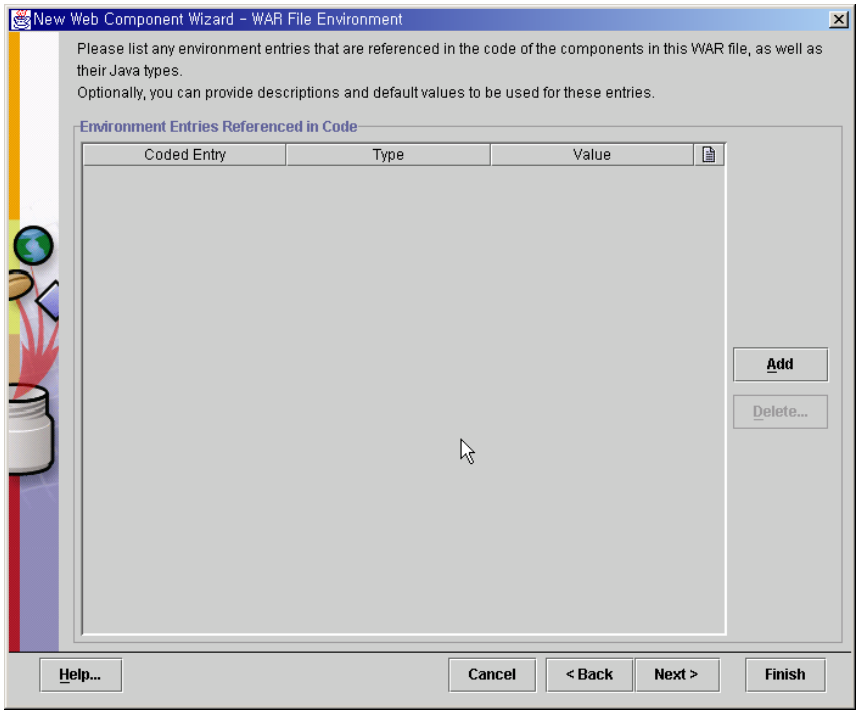

그림 102 Next버튼을 클릭합니다.

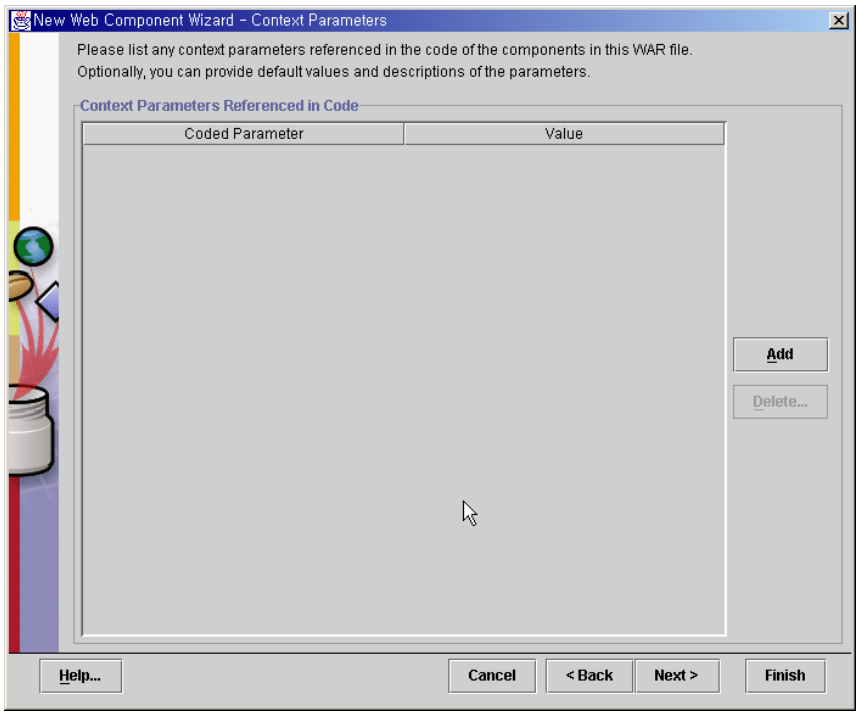

그림 103 Next버튼을 클릭합니다.

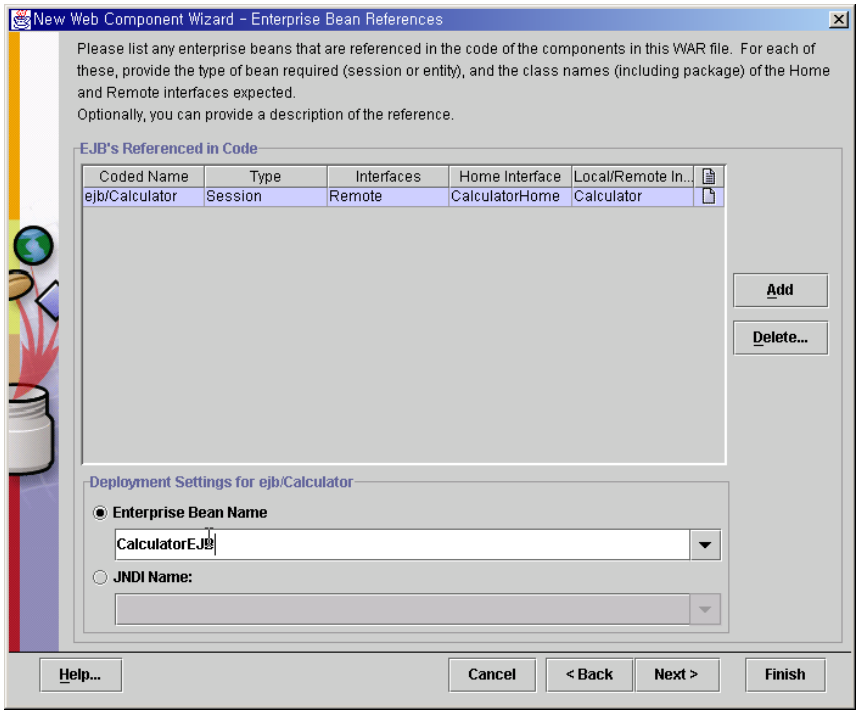

그림 104 그림과 같이 셋팅합니다.

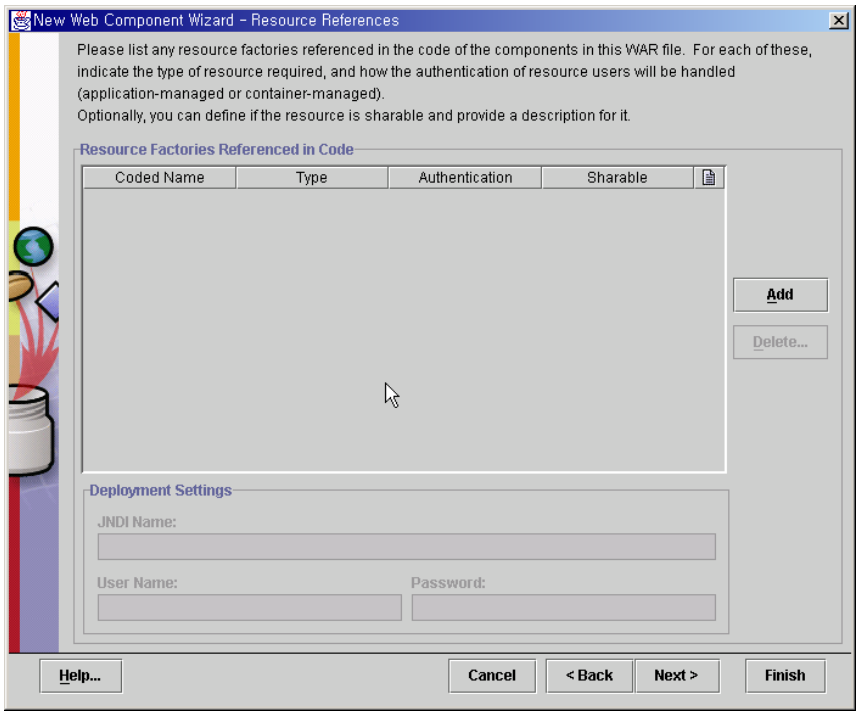

그림 105 Next 버튼을 클릭합니다.

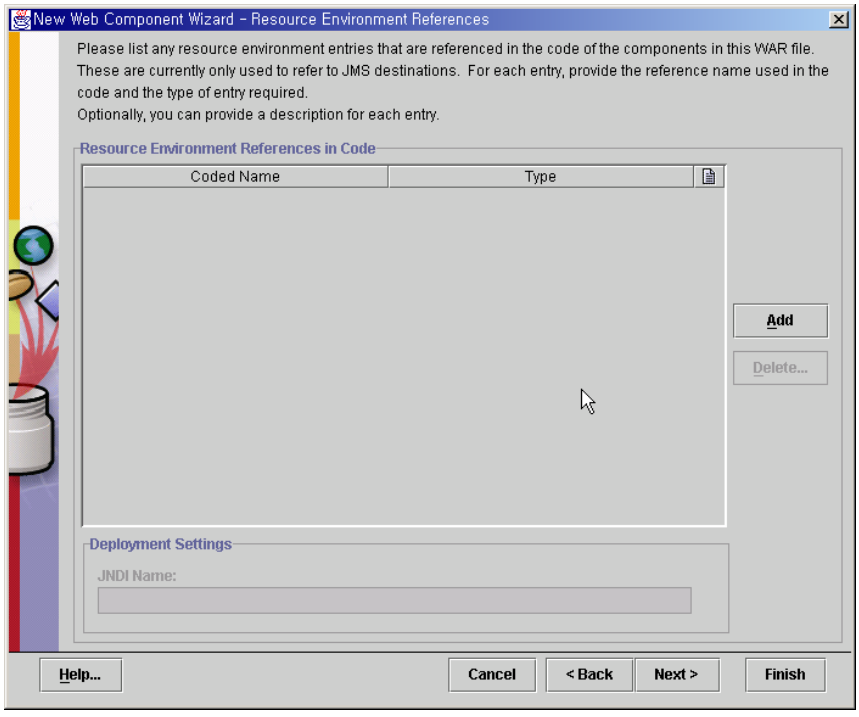

그림 106 Next버튼을 클릭합니다.

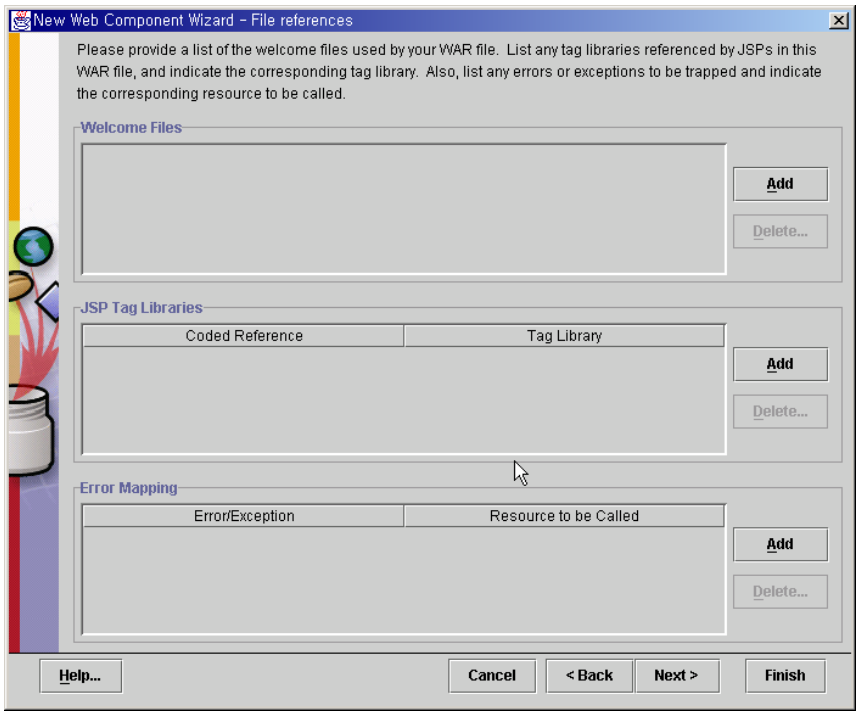

그림 107 Next버튼을 클릭합니다.

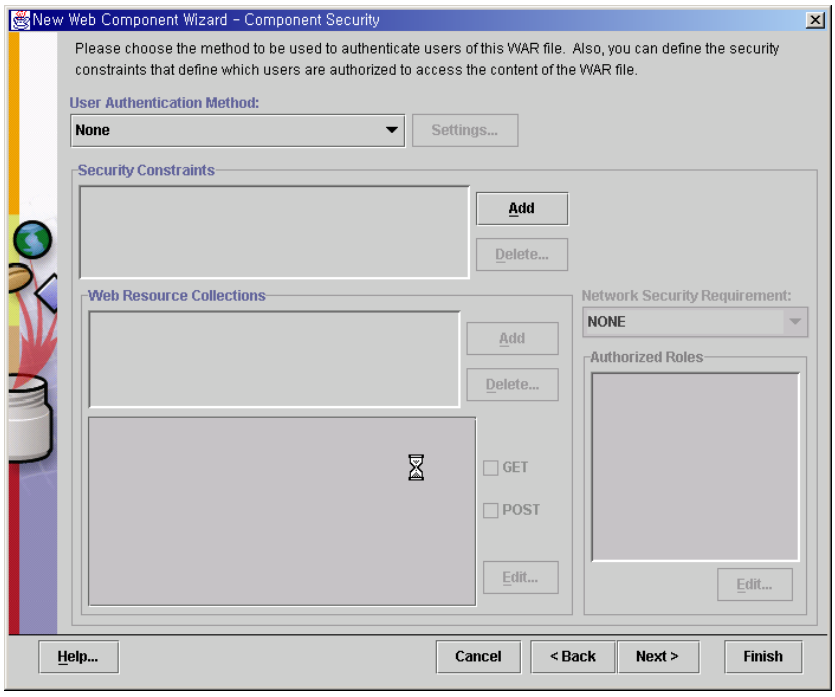

그림 108 Next버튼을 클릭합니다.

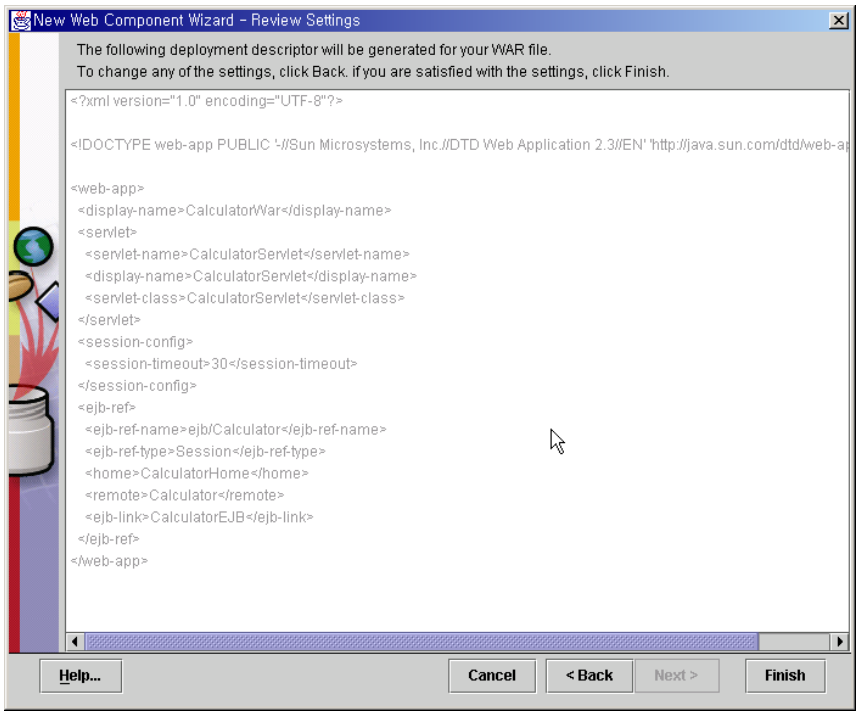

그림 109 Finish 버튼을 클릭합니다.

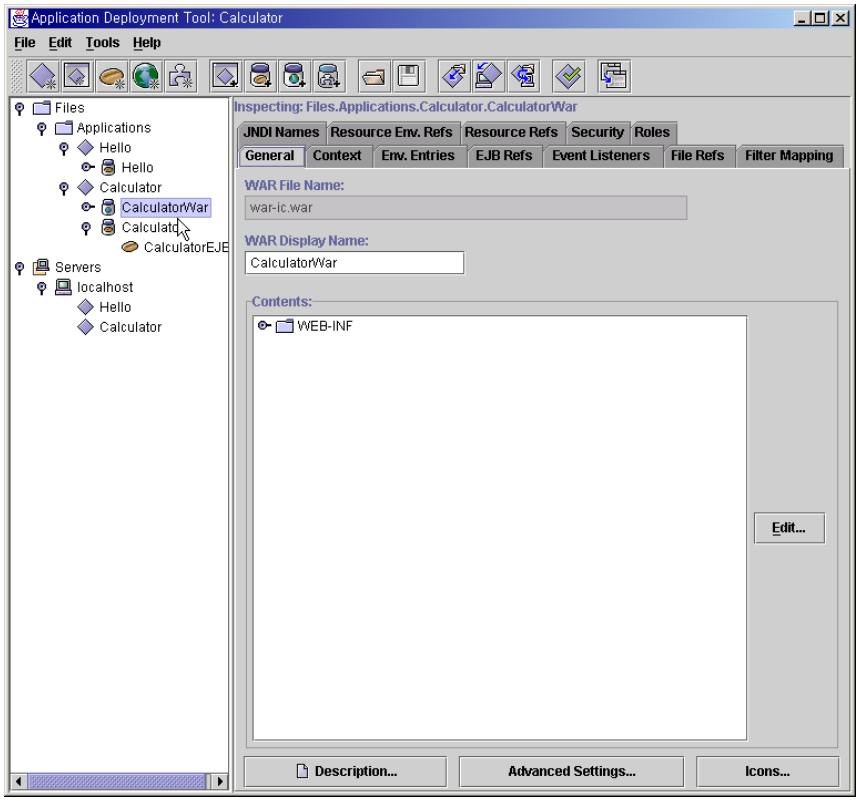

그림 110 그림과 같이 WAR파일이 추가가 된것을 알 수 있습니다.

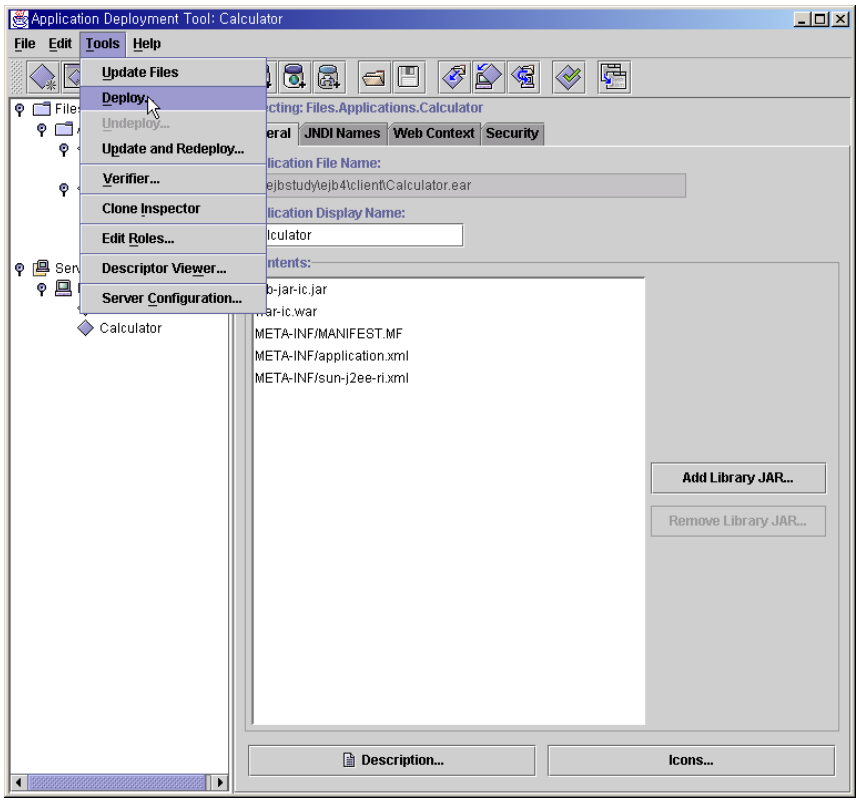

그림 111 추가된 WAR파일과 함께 deploy를 합니다.

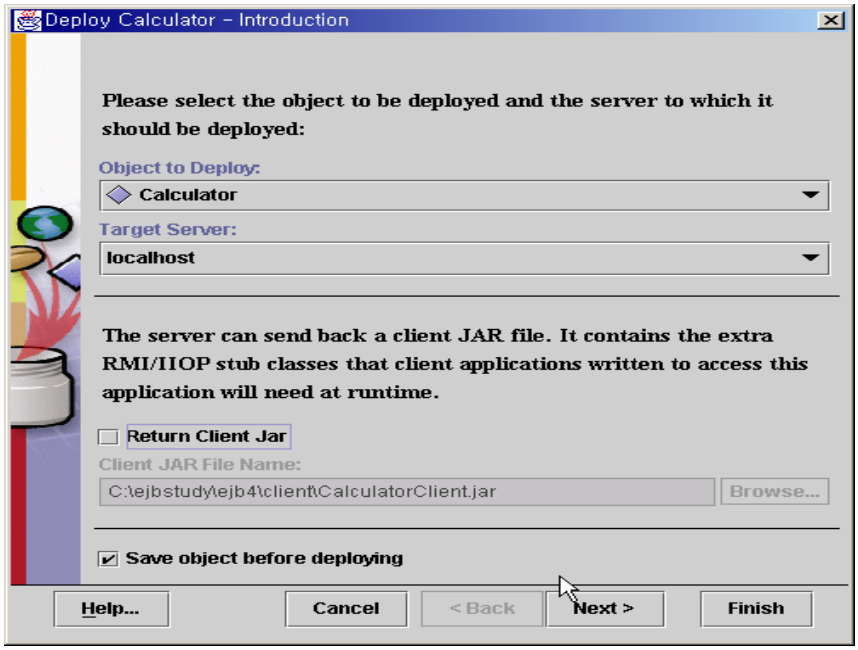

그림 112 그림과 같이 셋팅한 후 Next버튼을 클릭합니다.

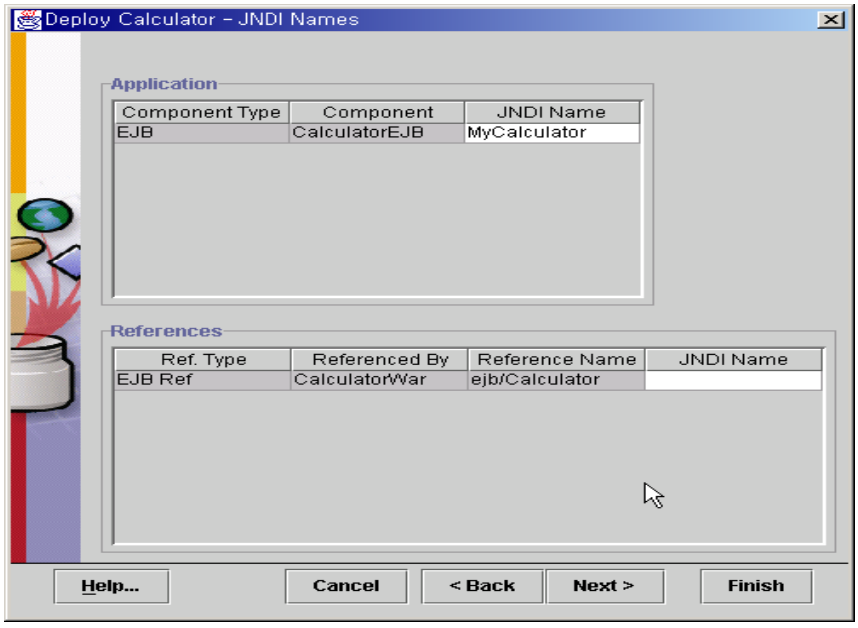

그림 113 EJB 객체에 대한 JNDI name을 지정합니다.

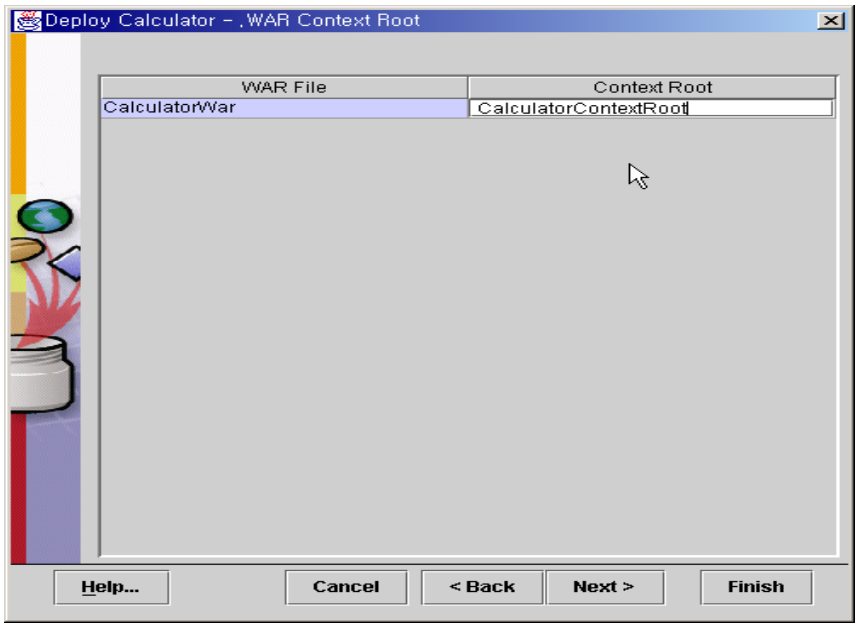

그림 114 ContextRoot를 그림과 같이 지정합니다.

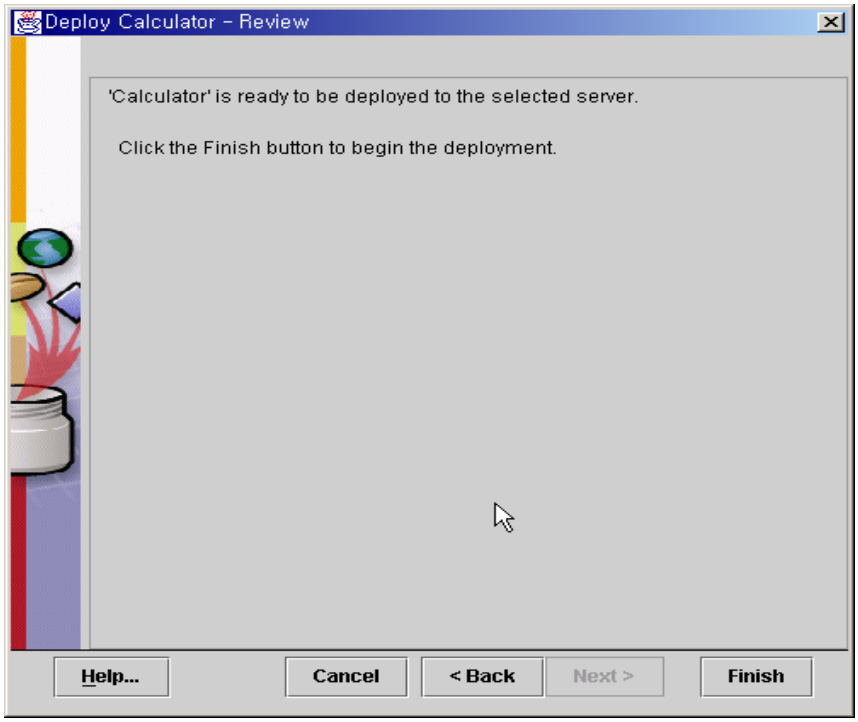

그림 115 Finish 버튼을 클릭합니다.

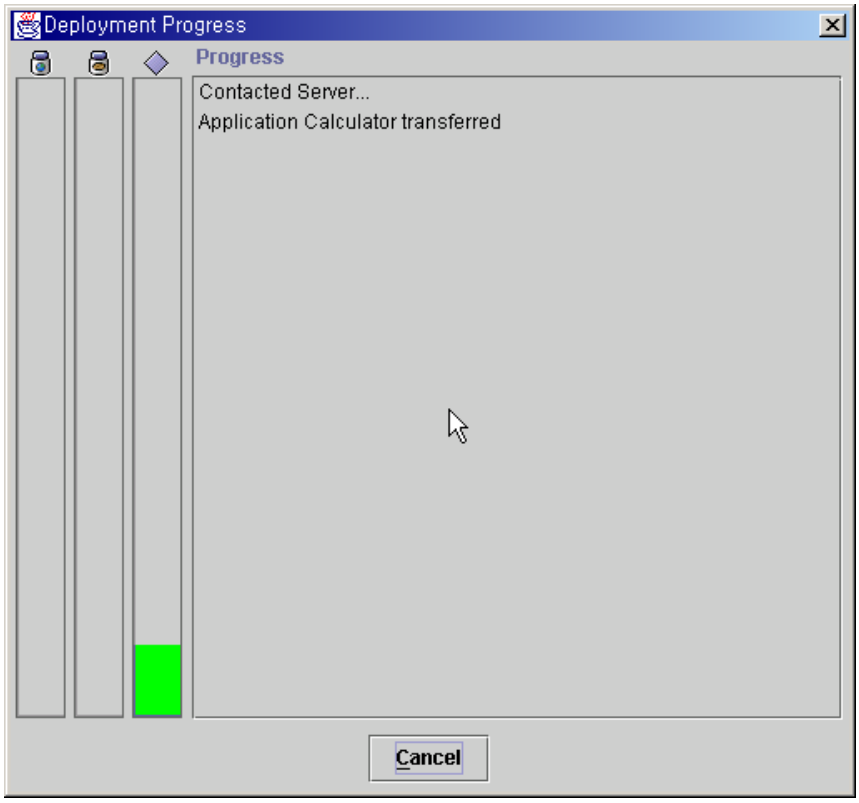

그림 116 디플로이 과정이 화면에 보여집니다.

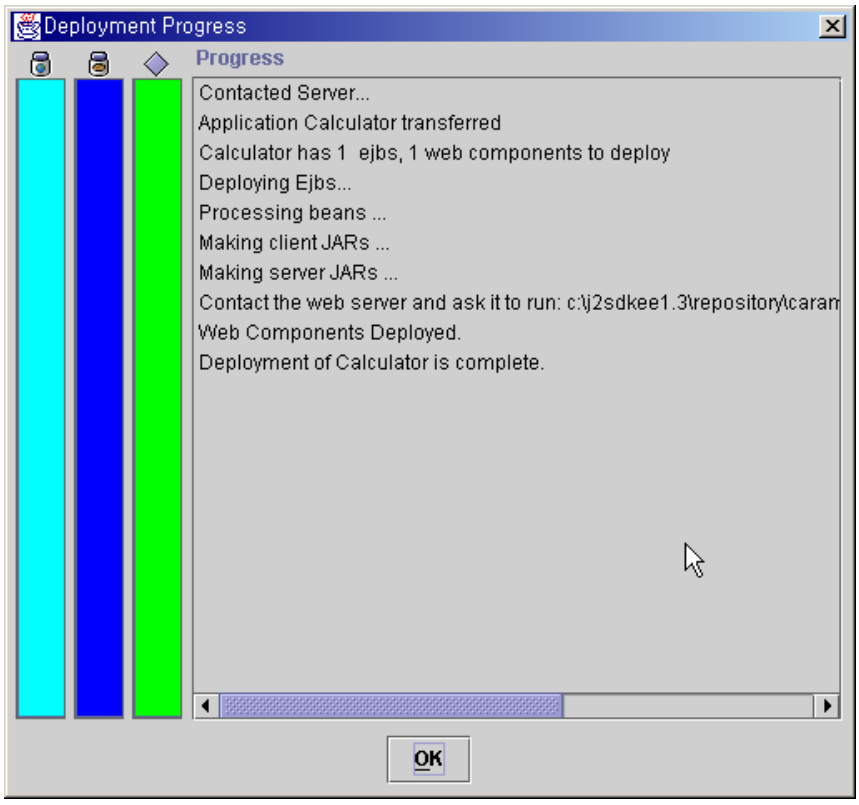

그림 117 디플로이 결과 화면입니다.

f. 서블릿의 실행

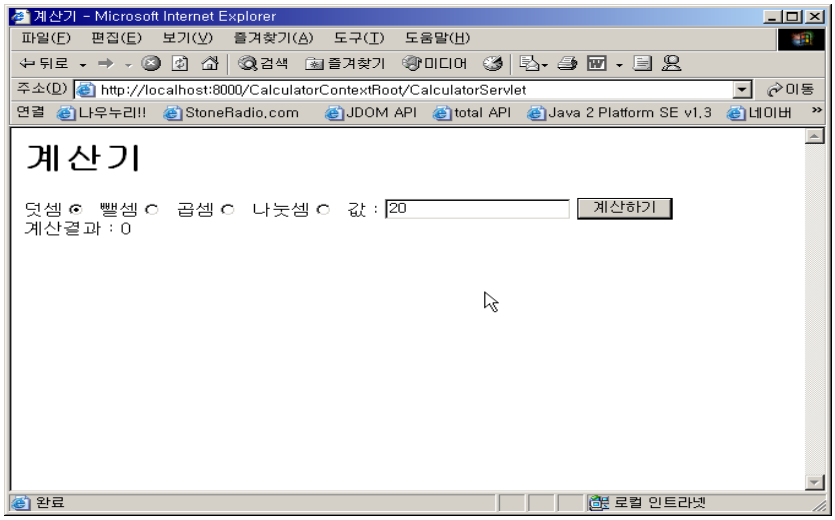

그림 118 서블릿을 처음 실행한 모습입니다.

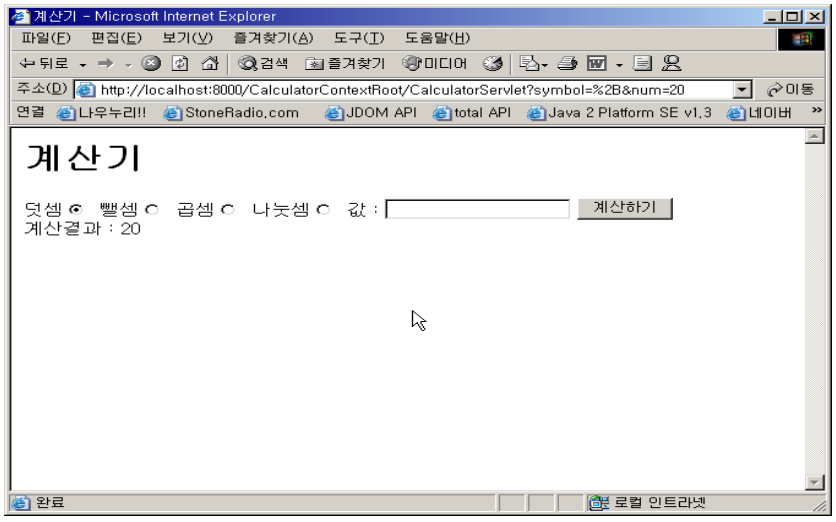

그림 119 메모리상에 있는 결과값과 계속하여 계산되는 것을 알 수 있습니다.

상태유지 세션빈을 init에서 이용하지 않고 doGet에서 이용한다면 결과는 어떻게 나올까요?

10. CMP(Container Managed Persistence) 엔티티 빈

CMP 엔티티빈은 EJB 컨테이너가 자동적으로 데이터의 영속성을 유지시켜 줍니다. 데이터 베이스에서 추가,삭제,변경 등의 작업등을 자동으로 처리해줍니다. CMP 엔티티빈을 이용하게 되면 대부분의 기능들을 자동으로 처리해주기 때문에 코드의 길이가 짧아지고 개발이 쉬워질 수 있습니다. 개발자는 데이터 베이스등에 대한 내용은 신경 쓰지 않고 비즈니스 로직 부분만 신경을 쓰면 됩니다. 하지만 단점은 앞으로 배울 BMP(Bean Managed Persistence)에 비해 정교함이 떨어지며, BMP보다 성능상 떨어질 수 있다는 데 있습니다. (여기에서 알아두어야 할 것은 BMP를 작성하는 사람이 자동으로 관리해주는 CMP보다 프로그래밍 적인 능력이 뛰어난 사람일 경우입니다.) 이제 CMP 엔티티 빈을 그림을 보며 따라해 보도록 합시다. 지금 만드는 엔티티 빈은 id를 가지고 이름을 관리하는 것입니다.

a. Home Interface의 작성

엔티티 빈의 Home Interface는 javax.ejb.EJBHome을 상속받으며, 0또는 1개이상의 create 메소드와 하나이상의 find 메소드를 정의해주어야 합니다. 모든 create 메소드는 java.rmi.RemoteException과 javax.ejb.CreateException을 throws 해줘야 합니다. Home Interface의 create 메소드와 1:1로 대응되도록 빈 클래스에서는 ejbCreate메소드를 구현해줘야 합니다.

find 메소드는 빈 클래스에 있는 find 메소드와 1:1로 대응됩니다. find메소드는

java.rmi.RemoteException과 java.ejb.FinderException을 throws 해줘야 합니다.

ENameHome.java 시작 ----------

```
import java.util.*;
import java.rmi.*;
import javax.ejb.*;

public interface ENameHome extends EJBHome{
    public EName create(String id, String txt) throws RemoteException,CreateException;
    public EName findByPrimaryKey(String id) throws FinderException, RemoteException;
}
```

ENameHome.java 끝 ----------

b. Remote Interface의 작성

Remote Interface는 javax.ejb.EJBObject를 반드시 상속받아야 하며, 모든 비즈니스 메소드를 정의하여 줍니다. 또한 반드시 java.rmi.RemoteException을 throws 해주어야 합니다. 또한 전달되는 값과 리턴하는 값은 몽땅 마샬링(직렬화)될 수 있는 객체이어야 합니다. 즉 RMI 호환형 변수이어야 한다는 의미입니다.

EName.java 시작 ----------

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface EName extends EJBObject{
    public void setName(String txt) throws RemoteException;
    public String getName() throws RemoteException;
}
```

EName.java 끝 ----------

c. Bean Class의 작성

ENameEJB.java 에서 보면 아래와 같은 내용이 있습니다.

```
public String id;
public String txt;
```

해당 변수들은 단순한 멤버 변수들이 아닙니다. 위에 적혀져 있는 id와 txt라는 변수는 자동으로 데이터 베이스에 저장될 필드로 구성이 됩니다. 실제로 Deploy Tool을 실행하여 디

플로이먼트를 실행하는 과정에서 데이터 베이스에 저장될 변수를 정할 수가 있게 됩니다. 이러한 변수들을 컨테이너 관리 필드(Container-managed field)라고 부릅니다. 컨테이너 관리 필드로 올 수 있는 변수는 자바 기본 자료형, 마샬링(직렬화)될 수 있는 객체, 리모트 인터페이스나 홈 인터페이스에 대한 레퍼런스 만이 올수 있습니다. 디플로이먼트 하는 과정에서는 반드시 프라이머리 키 필드의 역할을 해주는 변수가 있어야 합니다.

CMP 엔티티 빈은 반드시 EntityBean을 상속받아서 구현해줘야 하며, 홈 인터페이스에서 선언한 create와 find 메소드에 1:1로 대응되도록 메소드를 구현해 줘야 합니다. 또한 리모트 인터페이스에서 선언한 메도드를 구현해 줘야 합니다. 그리고 파라미터가 없는 생성자를 가지고 있어야 합니다.

ENameEJB.java 시작 ----------

```
import java.util.*;
import javax.ejb.*;

public class ENameEJB implements EntityBean{
	public String id;
	public String txt;
	private EntityContext context;

	public void setName(String txt){
		this.txt = txt;
	}

	public String getName(){
		return txt;
	}

	public String ejbCreate(String id,String txt) throws CreateException{
		if(id == null){
			throw new CreateException("id is required.");
		}
		this.id = id;
		this.txt = txt;
		return null;
	}
```

```
        public void setEntityContext(EntityContext context){
                this.context = context;
        }

        public void ejbActivate(){
                id = (String)context.getPrimaryKey();
        }

        public void ejbPassivate(){
                id = null;
                txt = null;
        }

        public void ejbRemove(){}
        public void ejbLoad(){}
        public void ejbStore(){}
        public void unsetEntityContext(){}
        public void ejbPostCreate(String id, String txt){}
}
```

ENameEJB.java 끝 ----------

d. 디플로이먼트 하기

CMP 엔티티 빈의 디플로이먼트 과정을 그림을 잘 보면서 따라하기 바랍니다.

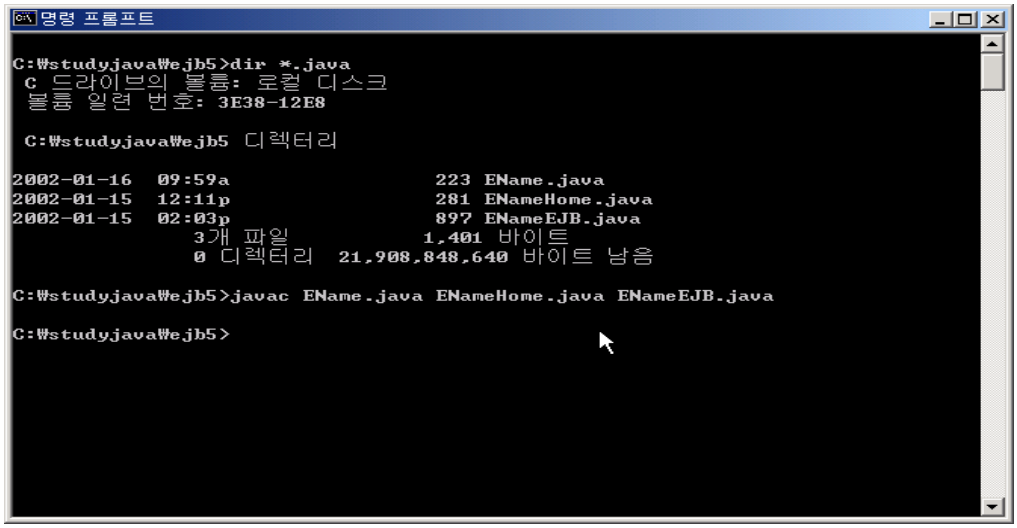

그림 120 홈 인터페이스, 리모트 인터페이스, 빈 클래스를 컴파일 합니다.

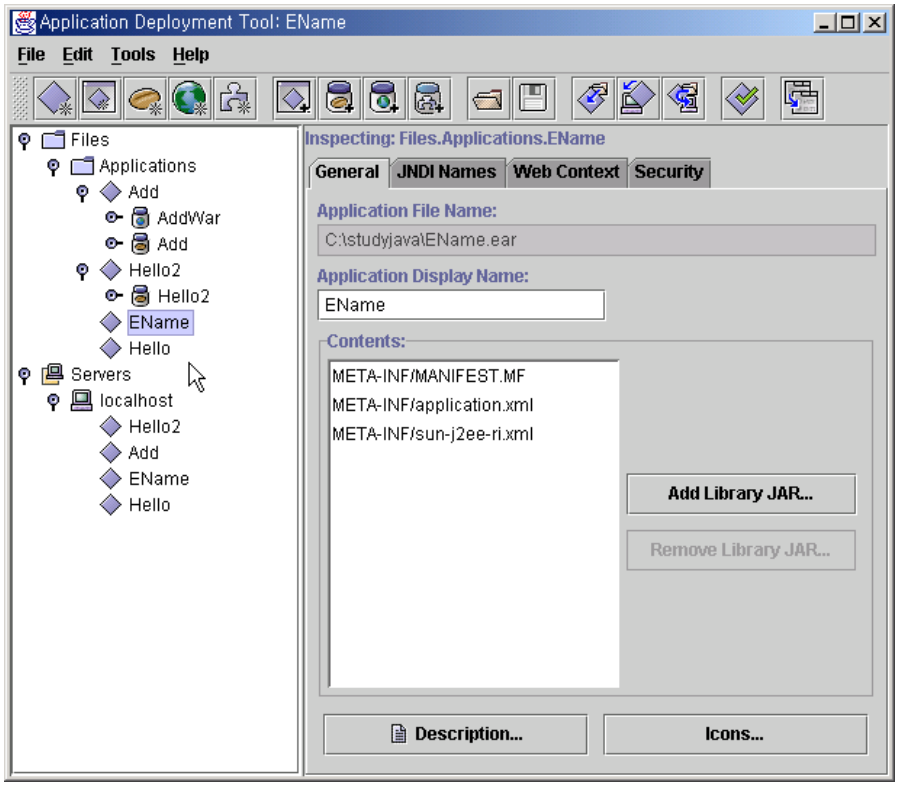

그림 121 그림과 같이 EName 이라는 Application을 생성합니다.

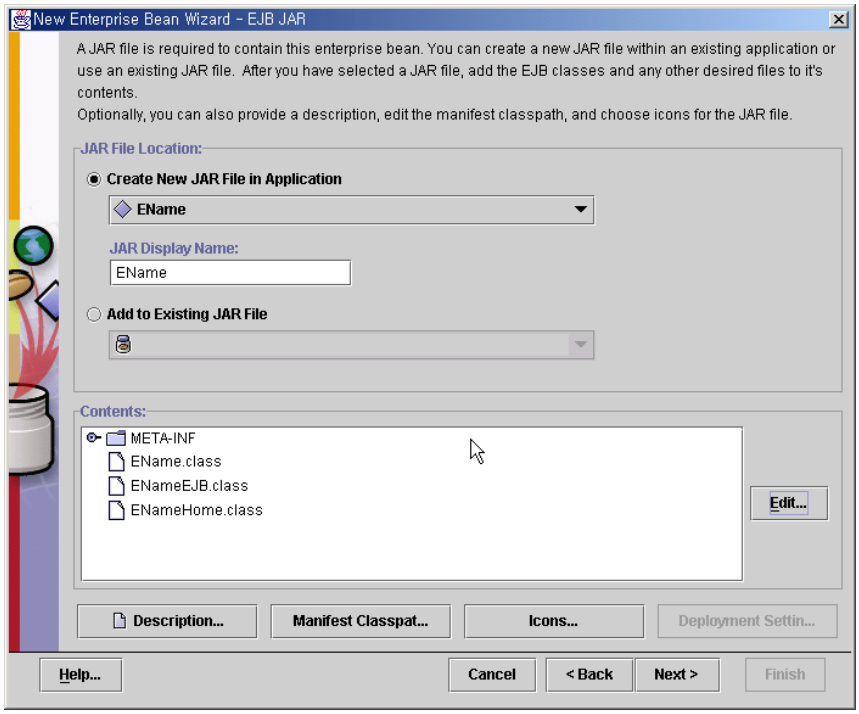

그림 122 그림과 같이 설정한 후 Next 버튼을 클릭합니다.

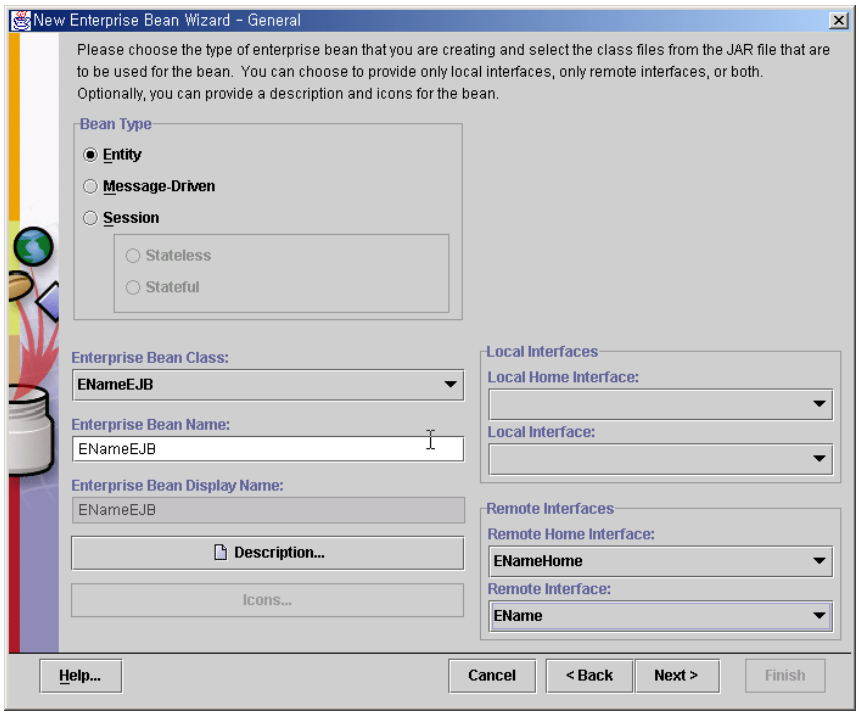

그림 123 그림과 같이 Bean Type을 Entity로 지정한 후 Next버튼을 클릭합니다.

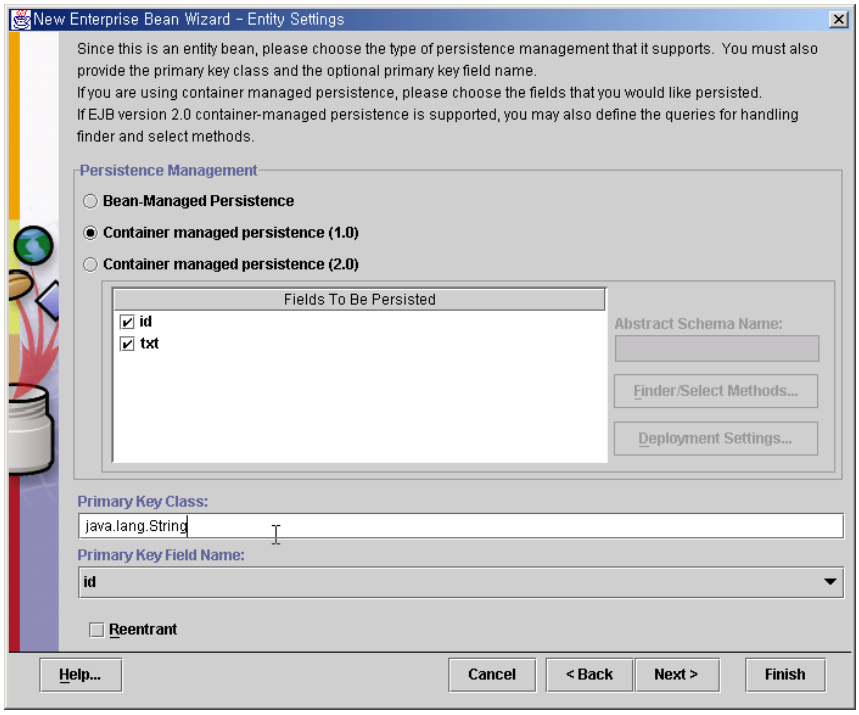

그림 124 CMP 1.0 방식을 선택합니다. 그리고 나서 id와 txt필드를 선택합니다. Primary Key의 역할을 해주는 id 가 String 형이기 때문에 Primary Key Class를 java.lang.String 으로 변경해 준 후 Primary Key Field Name으로 id를 지정하여 줍니다.

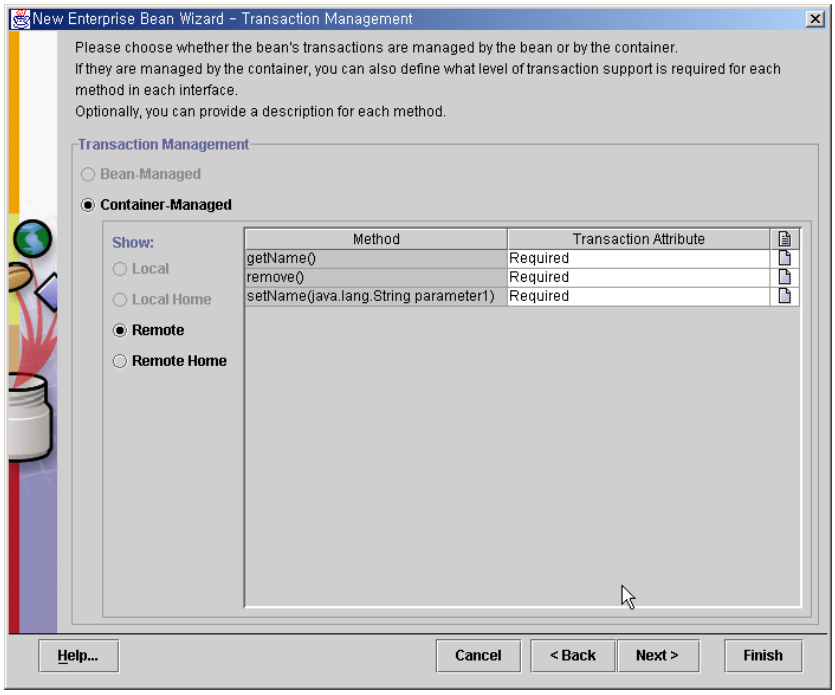

그림 125 Finish 버튼을 클릭합니다.

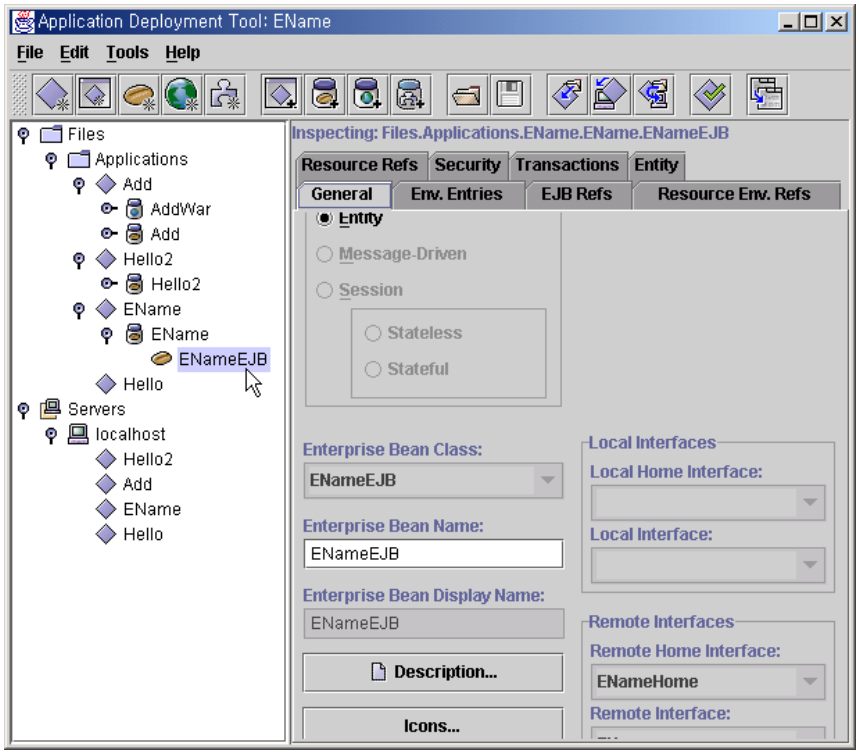

그림 126 그림과 같이 Enterprise Bean이 추가된 것을 알 수 있습니다.

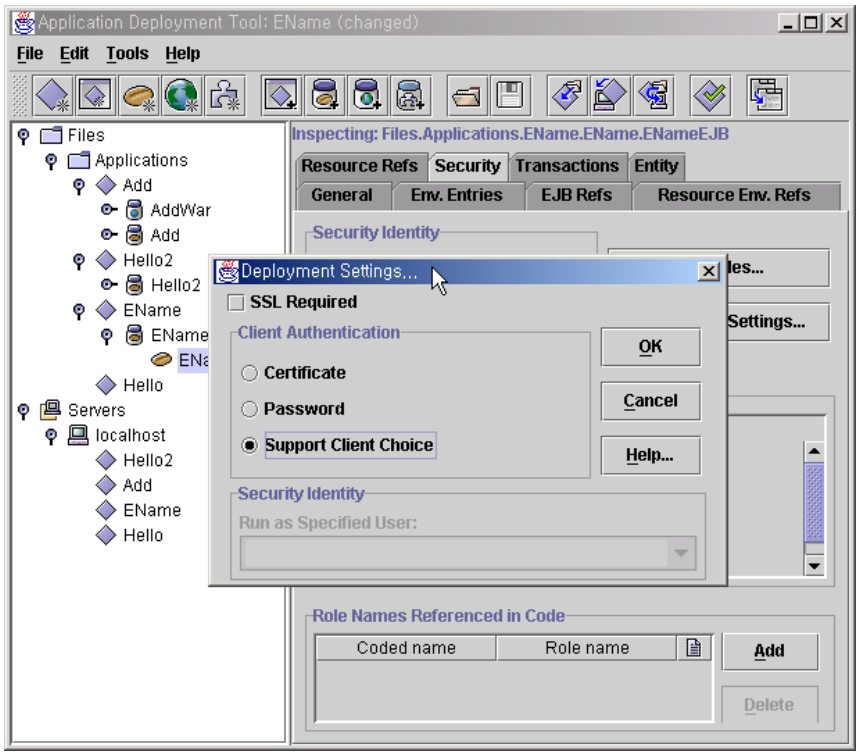

그림 127 Enterprise Bean의 보안 설정을 그림과 같이 바꿉니다.

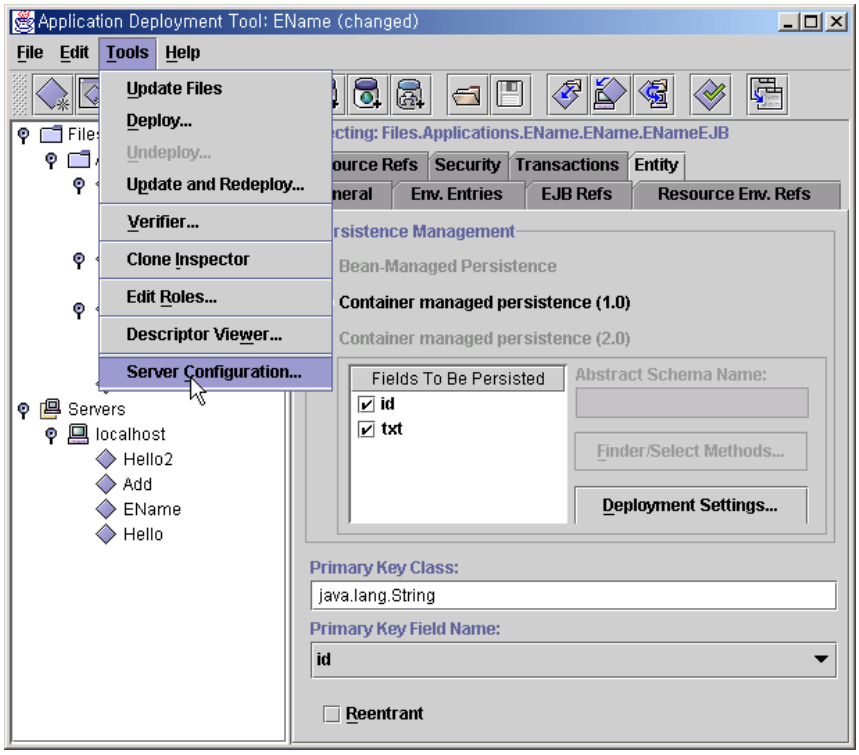

그림 128 Tools – Server Configuration을 선택합니다.

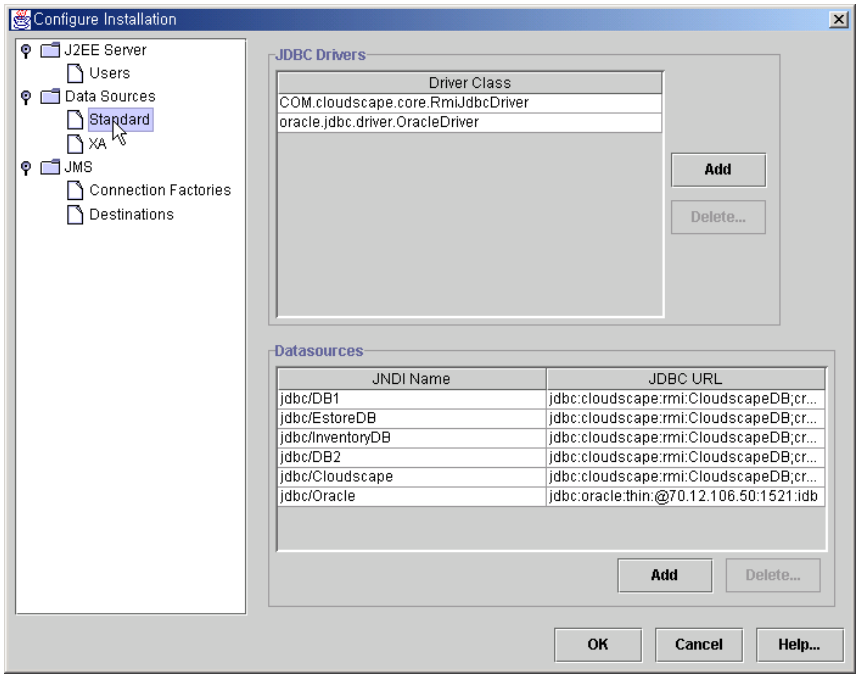

그림 129 오라클 데이터베이스에 연결하기 위하여 JDBC Driver와 JNDI Name, JNDI URL 을 설정합니다. 각각의 값은 oracle.jdbc.driver.OracleDriver , jdbc/Oracle, jdbc:oracle:thin:@host:1521:sid 로 선언합니다.

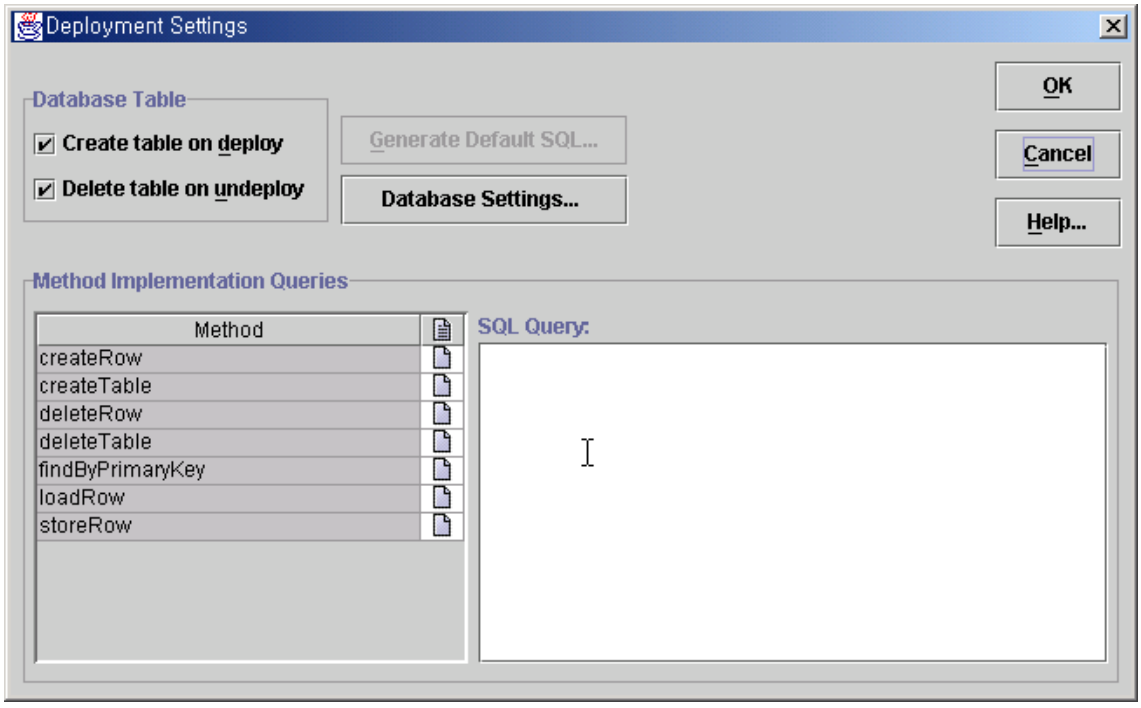

그림 130 Entity tab을 누른 후 Deployment Settings버튼을 누르면 그림과 같은 윈도우가 뜹니다. Create table on deploy, Delete table on undeploy를 check 한 후 Database Settings 버튼을 클릭합니다.

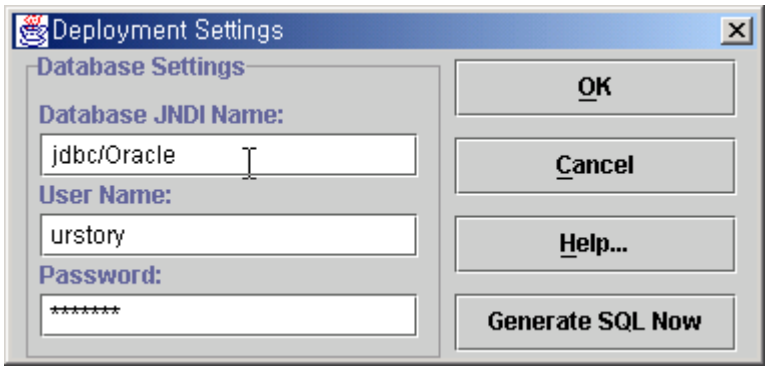

그림 131 그림과 같이 화면이 나오면 그림과 같이 지정합니다. 이때 User Name과 Password는 사용하는 Oracle계정의 아이디와 암호입니다. 그후에 Generate SQL Now 버튼을 클릭합니다.

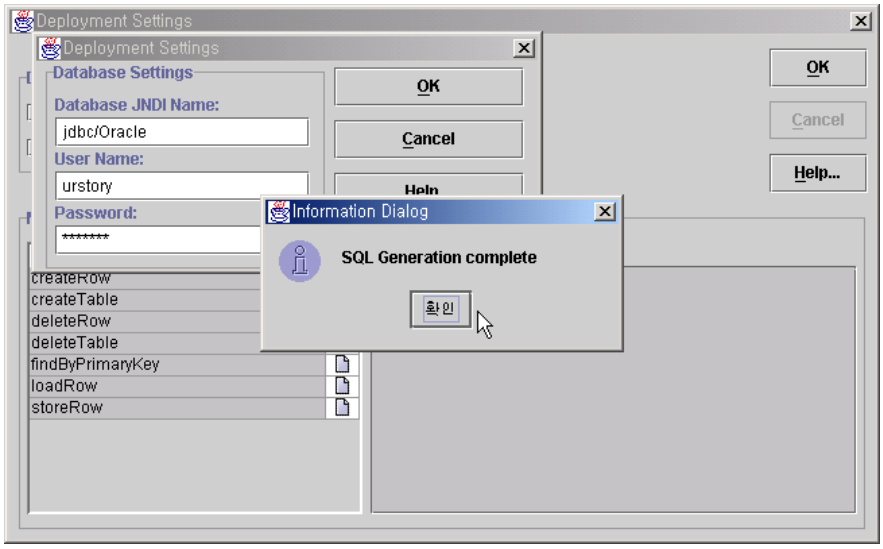

그림 132 그림과 같이 SQL Generation complete라는 창이 뜰 것입니다.

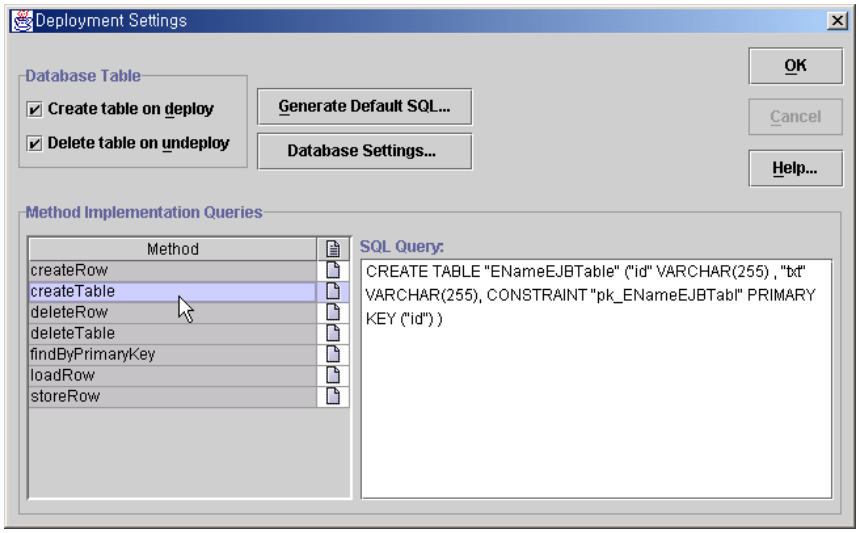

그림 133 Method 부분을 클릭해 보면 자동으로 SQL문장이 만들어 진 것을 알 수 있습니다. 이때 여러분들은 table이름 등을 변경 할 수 있습니다. 이때 테이블 명을 유심히 보면 큰따옴표로 묶여 있는 것을 알 수 있습니다. "ENameEJBTable" 과 ENameEJBTable 은 서로가 다른 이름의 테이블입니다.

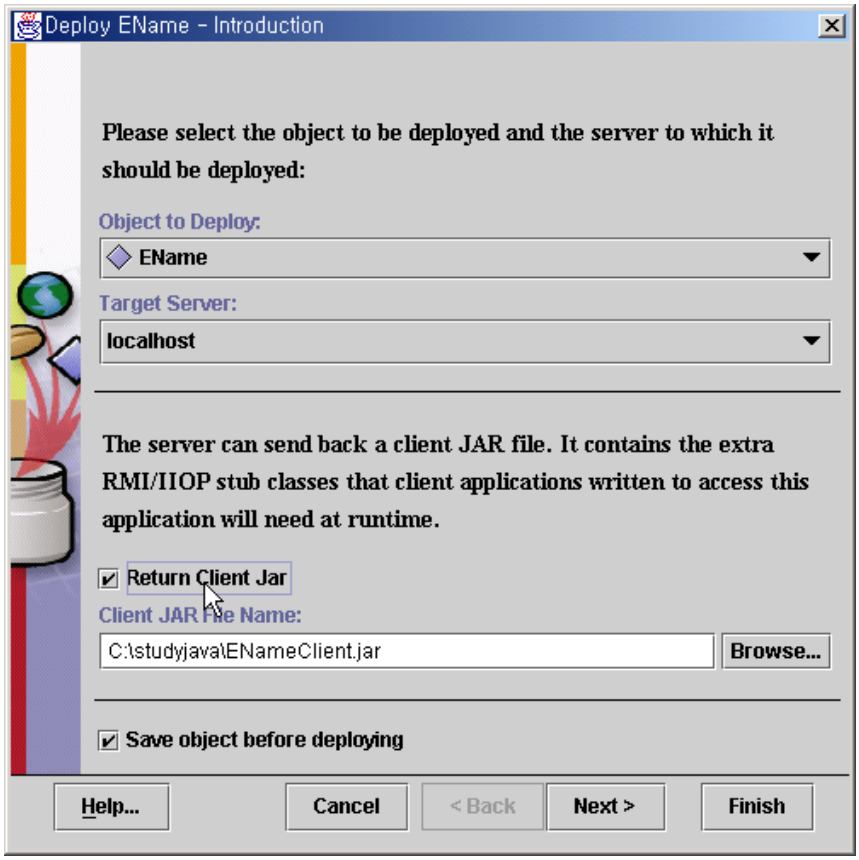

그림 134 Tools – Deploy를 선택하여 Deploy를 합니다. 그림과 같이 셋팅한 후 Next버튼을 클릭합니다.

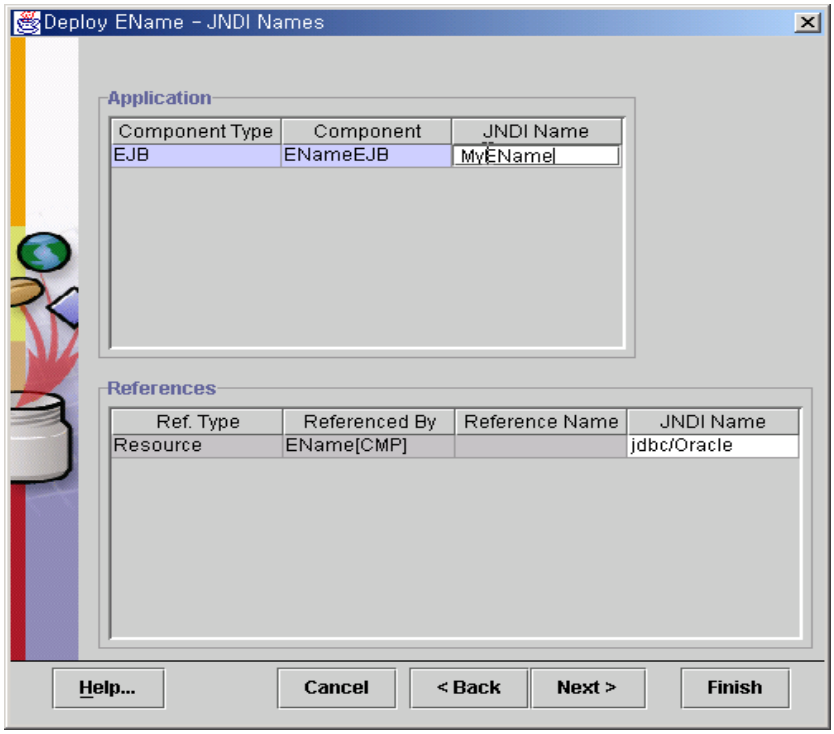

그림 135 Application에 대한 JNDI Name을 그림과 같이 설정한 후 Next버튼을 클릭합니다.

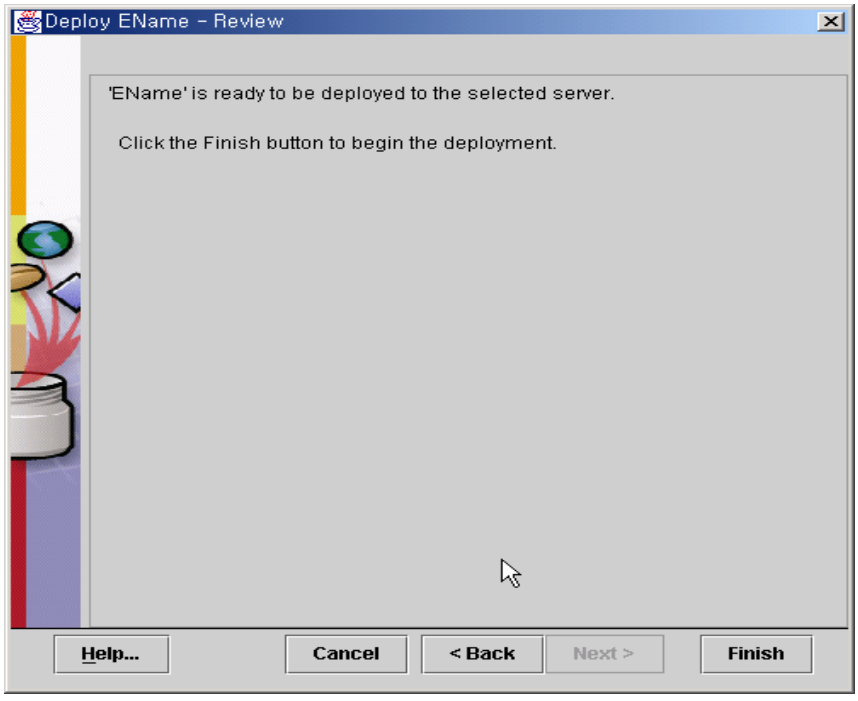

그림 136 Finish 버튼을 클릭합니다.

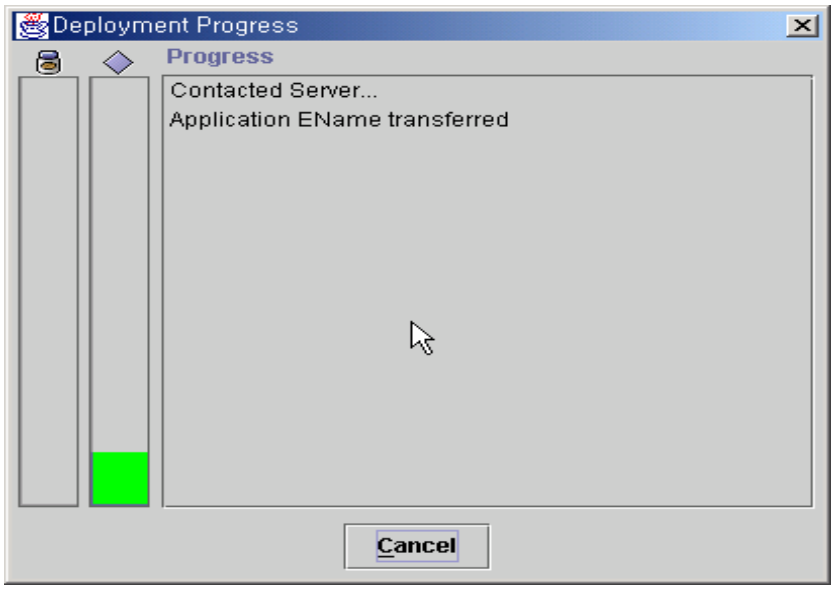

그림 137 deploy되는 과정이 그래프와 함께 보여집니다.

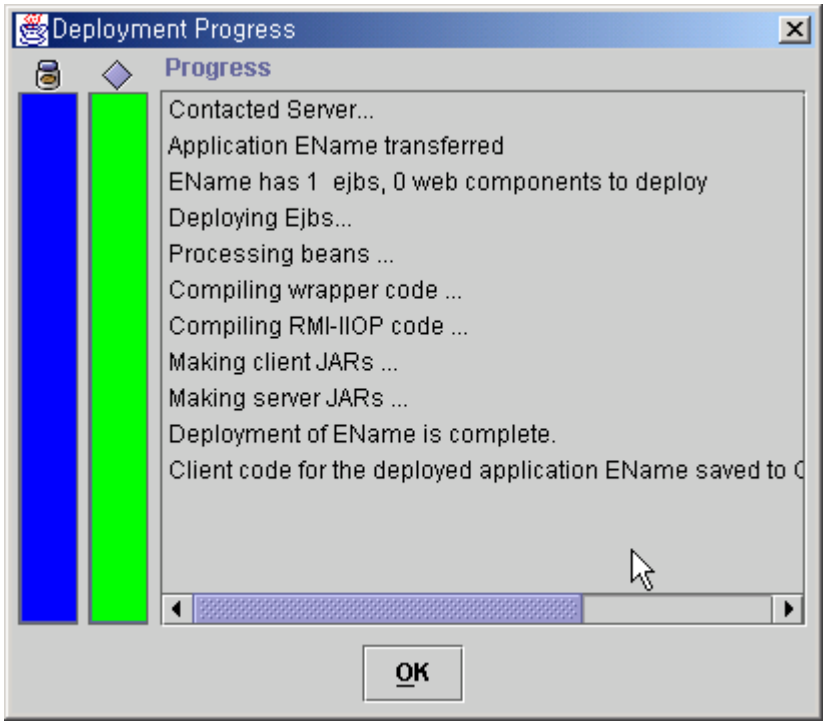

그림 138 deploy가 끝났습니다.

e. 클라이언트 애플리케이션의 작성

CMP 엔티티 빈을 테스트 하기 위한 클라이언트 프로그램을 작성합니다.

ENameClient.java 시작 ----------

```
import java.util.*;
import javax.naming.*;
import javax.rmi.*;
import ENameHome;
import EName;

public class ENameClient{
        public static void main(String args[]){
                try{
                        Context initial = new InitialContext();
                        Object obj = initial.lookup("MyEName");
                        ENameHome home =
(ENameHome)PortableRemoteObject.narrow(obj, ENameHome.class);
                        EName e1 = home.create("2","안녕하세요 오늘도 즐거운 날이
되길 바래요.");
                        System.out.println(e1.getName());
                        EName e2 = home.findByPrimaryKey("2");
                        System.out.println(e2.getName());
                }catch(Exception e){
                        e.printStackTrace();
                }
        } // end main
}
```

ENameClient.java 끝 ----------

f. 클라이언트 프로그램의 실행

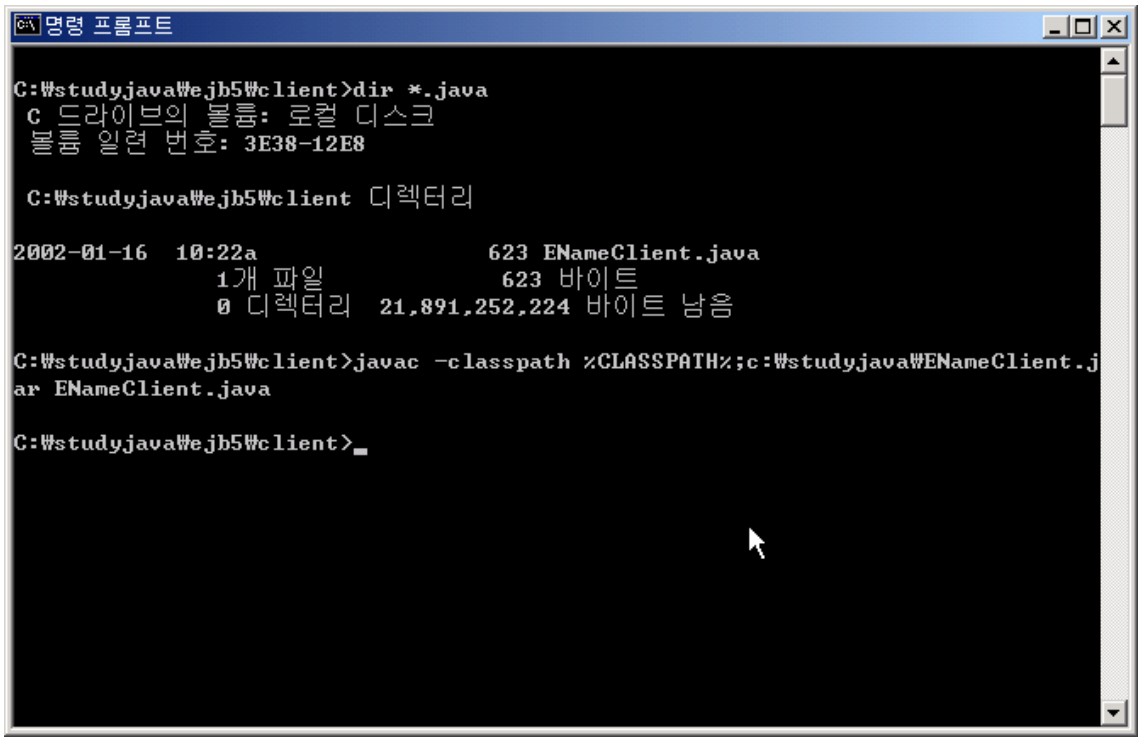

그림 139 CMP 엔티티 빈을 테스트 하는 ENameClient.java 를 컴파일 합니다.

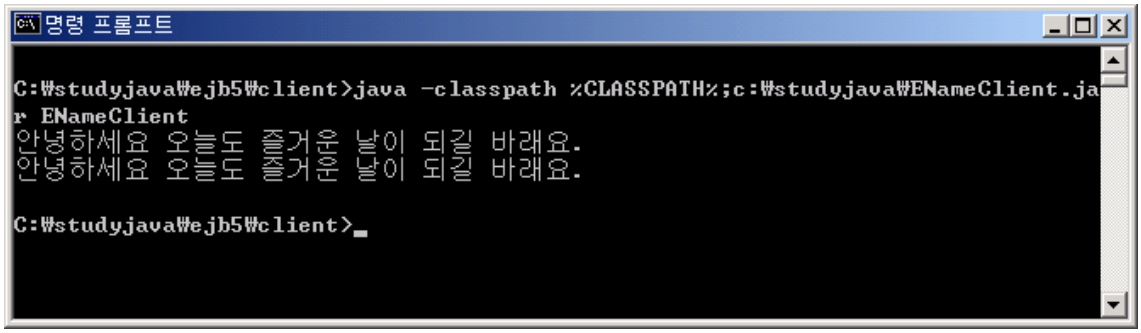

그림 140 CMP 엔티티 빈을 실행한 결과 입니다.

데이터 베이스에는 어떤 값이 저장 되었을까요?

11. BMP(Bean Managed Persistence) 엔티티 빈

BMP 엔티티 빈은 EJB 컨테이너가 자동으로 처리하도록 하는 것이 아니라, 프로그래머가 데이터베이스 관련 메소드, SQL구문 등을 모두 처리해 주어야 합니다. 그렇기 때문에 CMP 엔티티 빈 보다 BMP 엔티티 빈이 프로그래밍 하기에 더욱 까다롭습니다. 하지만, 수공예품이 더욱 세밀하게 만들어져서 팔리는 것처럼 BMP 엔티티 빈은 좀 더 세밀한 작업처리를 할 수 있도록 해 줍니다.

또한 BMP 엔티티 빈은 CMP에서는 자동으로 작성되던 find 메소드를 모두 구현해 주어야 하며, 특정한 데이터베이스에 종속적으로 작성되어질 수 있습니다. 그 말은, CMP 엔티티 빈과 비교하여 재사용성이 떨어질 수 있다는 이야기 입니다.

은행의 잔고를 확인하는 BMP엔티티 빈을 작성하면서 BMP 엔티티 빈에 대하여 알아보도록 하겠습니다.

a. Home Interface의 작성

Home Interface는 엔티티 빈의 생명주기에 관련된 메소드들을 구성합니다. Home Interface 에는 0개 혹은 1개 이상의 create메소드와 1개 이상의 find메소드로 이루어 집니다. create 메소드는 java.rmi.RemoteException과 javax.ejb.CreateException을 throws하여야 합니다. create 메소드는 빈 클래스에서 ejbCreate 메소드와 1:1로 대응됩니다. 매칭되는 ejbCreate메소드는 인자의 형식과 수가 같아야 합니다.

find메소드는 find로 시작합니다. find메소드도 create메소드와 같이 빈 클래스에서 ejbFind로 시작하는 메소드와 1:1로 대응되어야 합니다. find는 매소드는 또한 javax.ejb.FinderException과 javax.rmi.RemoteException을 throws하여야 합니다.

AccountHome.java 시작 ----------

```
import java.util.*;
import java.rmi.*;
import javax.ejb.*;

public interface AccountHome extends EJBHome{
        public Account create(String id, String name, double balance) throws
RemoteException,CreateException;
        public Account findByPrimaryKey(String id) throws FinderException,
RemoteException;
        public Collection findByName(String name) throws FinderException,
RemoteException;
```

```
        public Collection findInRange(double low, double high) throws FinderException,
RemoteException;
}
```

AccountHome.java 끝 ----------

b. Remote Interface의 작성

리모트 인터페이스는 비즈니스 로직을 처리하는 메소드를 정의합니다.

Account.java 시작 ----------

```
import javax.ejb.*;
import java.rmi.*;

public interface Account extends EJBObject{
        // 인출하다
        public void drawout(double amount) throws AccountException,
RemoteException;
        // 예금하다
        public void credit(double amount) throws RemoteException;
        // 이름을 가져오다.
        public void getName() throws RemoteException;
        // 잔액을 가져오다.
        public double getBalance() throws RemoteException;

}
```

Account.java 끝 ----------

c. 사용자 정의 Exception

사용자 정의 Exception은 Exception을 처리하고자 사용됩니다.

AccountException.java 시작 ----------

```
public class AccountException extends Exception{
        public AccountException(){
                super();
        }
        public AccountException(String msg){
                super(msg);
        }
}
```

AccountException.java 끝 ----------

d. Bean Class 작성하기

실제로 BMP 엔티티 빈에서 가장 중요한 것은 Bean class의 작성 부분입니다. CMP 엔티티 빈과는 달리 개발자가 신경 써야 할 부분이 많기 때문일 것입니다.

일단 BMP 엔티티 빈에서 가장 먼저 만들어야 할 메소드는 홈 인터페이스에서 정의된 create메소드와 find메소드에 1:1로 대응되는 ejbCreate메소드와 ejbFind메소드입니다.

ejbCreate메소드는 데이터베이스에 엔티티 상태를 삽입하고, 빈 인스턴스 속성을 초기화하고, 프라이머리 키를 반환하는 과정을 처리합니다. CMP 엔티티빈은 반환되는 값이 무시되므로 null값을 리턴 하도록 프로그래밍 하면 되지만, BMP 엔티티 빈일 경우에는 반드시 프라이머리 키 값이 리턴되도록 하여야 합니다. 또한 ejbCreate 메소드는 반드시 public 메소드이어야 하며, 전달되는 인자가 마샬링 될 수 있어야 하고 메소드 지정자가 final이나 static으로 올 수 없습니다.

find메소드는 CMP 엔티티 빈일 경우 컨테이너가 자동으로 생성하여 주지만, BMP 엔티티 빈일 경우에는 개발자가 모두 구현해 줘야 합니다. find관련 메소드중 findByPrimaryKey메소드는 반드시 구현되어야 할 메소드입니다.

이렇게 ejbCreate메소드와 find메소드를 구현하였다면, 아래와 같은 메소드를 추가로 구현해줘야 합니다.

- ejbPostCreate메소드
- ejbRemove메소드
- ejbLoad메소드
- ejbStore메소드
- setEntityContext메소드
- unsetEntityContext메소드
- ejbActivate메소드
- ejbPassivate메소드

각각의 메소드의 설명은 아래와 같습니다.

ejbPostCreate메소드는 ejbCreate메소드가 호출된 뒤 바로 다음으로 실행되는 메소드이며 보통은 선언만 하고 구현은 하지 않습니다. ejbPostCreate메소드는 EntityContext인터페이스의 getPrimaryKey와 getEJBObject메소드를 호출할 수 있습니다.ejbPostCreate메소드는 ejbCreate메소드와 대응되며 ejbCreate메소드가 가지는 파라미터의 개수와 형태와 일치해야 합니다. ejbPostCreate메소드의 접근 제어자는 반드시 public이어야 하며 메소드 지정자는 final이거나 static일 수 없습니다. 메소드의 반환형은 반드시 void 형이어야 합니다.

ejbRemove메소드는 클라이언트가 remove메소드를 호출할 경우 호출되는 메소드입니다.

ejbRemove메소드가 실행될 경우 컨테이너는 해당 레코드를 삭제하게 됩니다. 만약 엔티티 빈이 삭제될 경우 처리할 내용을 작성하고 싶다면 ejbRemove메소드에서 작성해주면 됩니다.

ejbLoad메소드와 ejbStore메소드는 엔티티빈의 인스턴스 속성값과 데이터베이스에 저장된 값의 동기화를 맞출 때 사용되는 메소드입니다. EJB 컨테이너는 동기화를 맞춰야 할 경우 ejbLoad와 ejbStore메소드를 호출하게 됩니다. 이 두개의 메소드는 클라이언트에서 호출할 수는 없고, EJB가 호출하는 메소드입니다. ejbLoad메소드는 데이터베이스로부터 값을 읽어와 엔티티 빈의 상태 값을 재설정 하기 위한 메소드이고, ejbStore는 반대로 데이터베이스에 엔티티 빈의 상태 값을 저장하는 메소드입니다. 이 두개의 메소드는 사용하기에 따라서 아주 유용하게 사용될 수 있습니다.(어떤 경우일까요?)

setEntityContext메소드는 빈의 JNDI컨텍스트와 데이터베이스에 대한 연결을 설정하는 메소드입니다.

unsetEntityContext메소드는 데이터베이스에 대한 연결 설정을 해제합니다.

ejbActivate메소드는 bean인스턴스를 활성화 시킬 때 이용되는 메소드로 CMP 엔티티빈과 BMP 엔티티빈 모두 동일한 기능을 수행합니다.

ejbPassivate메소드는 bean인스턴스를 비활성화 시킬 때 이용되는 메소드로 CMP 엔티티빈과 BMP 엔티티빈 모두 동일한 기능을 수행합니다.

위에서 설명한 메소드를 모두 구현하였다면, 마지막으로 리모트 인터페이스에서 정의한 비즈니스 로직을 담당하고 있는 메소드를 구현합니다. AccountEJB.java를 보며 각각의 메소드에서 어떻게 정의하고 있는지 살펴보시기 바랍니다.

AccountEJB.java 시작 ----------

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.ejb.*;
import javax.naming.*;

public class AccountEJB implements EntityBean{
        private String id;
        private String name;
```

```
	private double balance;
	private EntityContext context;
	private Connection con;
	private String dbName = "jdbc/Oracle";

	public void drawout(double amount) throws AccountException{
		if(balance - amount < 0){
			throw new AccountException("잔고가 부족합니다.");
		}
		balance -= amount;
	}

	public void credit(double amount){
		balance + = amount;
	}

	public String getName(){
		return name;
	}

	public double getBalance(){
		return balance;
	}

	public String ejbCreate(String id, String name, double balance) throws
CreateException{
		if( balance < 0.00 ){
			throw new CreateException("잔고가 0보다 작을 수는 없습니
다.");
		}
		try{
			insertRow(id, name, balance);
		}catch(Exception e){
			throw new EJBException("create exception: " + e);
		}
		this.id = id;
```

```
                this.name = name;
                this.balance = balance;

                return id;
        }

        public String ejbFindByPrimaryKey(String primaryKey) throws
FinderException{
                boolean result;

                try{
                        result = selectByPrimaryKey(primaryKey);
                }catch(Exception e){
                        throw new EJBException("ejbFindByPrimaryKey exception: "
+ e);
                }
                if(result){
                        return primaryKey;
                }else{
                        throw new ObjectNotFoundException(primaryKey + " 값을 발
견할 수 없습니다.");
                }
        }

        public Collection ejbFindByName(String name) throws FinderException{
                Collection result;

                try{
                        result = selectByName(name);
                }catch(Exception e){
                        throw new EJBException("이름을 찾을 수가 없습니다.");
                }
                // 결과가 없을 경우
                if(result.isEmpty()){
                        throw new ObjectNotFoundException("No rows found");
                }else{
```

```
                return result;
            }
        }

        public Collection ejbFindInRange(double low, double high) throws
FinderException{
            Collection result;
            try{
                result = selectInRange(low, high);
            }catch(Exception e){
                throw new EJBException("Range Exception: "+ e);
            }
            if(result.isEmpty()){
                throw new ObjectNotFoundException("No rows found");
            }else{
                return result;
            }
        }

        public void ejbRemove(){
            try{
                deleteRow(id);
            }catch(Exception e){
                throw new EJBException("ejbRemote: " + e);
            }
        }

        public void setEntityContext(EntityContext context){
            this.context = context;
            try{
                makeConnection();
            }catch(Exception e){
                throw new EJBException("setEntityContext : " + e);
            }
        }
```

```
	public void unsetEntityContext(){
		try{
			con.close();
		}catch(SQLException e){
			throw new EJBException("unsetEntityContext: " + e);
		}
	}

	public void ejbActivate(){
		id = (String)context.getPrimaryKey();
	}

	public void ejbPassivate(){
		id = null;
	}

	public void ejbLoad(){
		try{
			loadRow();
		}catch(Exception e){
			throw new EJBException("ejbLoad: " + e);
		}
	}

	public void ejbStore(){
		try{
			storeRow();
		}catch(Exception e){
			throw new EJBException("ejbStore: " + e);
		}
	}

	public void ejbPostCreate(String id, String name, double balance){}

	/**********	데이터	베이스	핸들링	부분
**********/
```

```
        private void makeConnection() throws NamingException, SQLException{
                InitialContext initial = new InitialContext();
                DataSource ds = (DataSource)initial.lookup(dbName);
                con = ds.getConnection();
        }

        private void insertRow(String id, String name, double balance) throws
SQLException{
                String insertS = "insert into account values (?, ?, ? )";
                PreparedStatement insertps = con.prepareStatement(insertS);
                insertps.setString(1, id);
                insertps.setString(2, name);
                insertps.setDouble(3, balance);
                insertps.executeUpdate();
                insertps.close();
        }

        private void deleteRow(String id) throws SQLException{
                String deleteS = "delete from account where id = ?";
                PreparedStatement deleteps = con.prepareStatement(deleteS);
                deleteps.setString(1, id);
                deleteps.executeUpdate();
                deleteps.close();
        }

        private boolean selectByPrimaryKey(String primaryKey) throws SQLException{
                String selectS = "select id from account where id = ?";
                PreparedStatement selectps = con.prepareStatement(selectS);
                selectps.setString(1, primaryKey);
                ResultSet rs = selectps.executeQuery();
                boolean result = rs.next();
                rs.close();
                selectps.close();
                return result;
        }
```

```
	private Collection selectByName(String name) throws SQLException{
		String selectS = "select id from account where name = ?";
		PreparedStatement selectps = con.prepareStatement(selectS);
		selectps.setString(1, name);
		ResultSet rs = selectps.executeQuery();
		ArrayList a = new ArrayList();
		while(rs.next()){
			String id = rs.getString(1);
			a.add(id);
		}
		rs.close();
		selectps.close();
		return a;
	}

	private Collection selectInRange(double low, double high) throws SQLException{
		String selectS = "select id from account where balance between ? and ?";
		PreparedStatement selectps = con.prepareStatement(selectS);
		selectps.setDouble(1, low);
		selectps.setDouble(2, high);
		ResultSet rs = selectps.executeQuery();
		ArrayList a = new ArrayList();
		while(rs.next()){
			String id = rs.getString(1);
			a.add(id);
		}
		rs.close();
		selectps.close();
		return a;
	}

	private void loadRow() throws SQLException{
		String selectS = "select name, balance from account where id = ?";
		PreparedStatement selectps = con.prepareStatement(selectS);
```

```
                selectps.setString(1, this.id);
                ResultSet rs = selectps.executeQuery();
                if(rs.next()){
                        this.name = rs.getString(1);
                        this.balance = rs.getDouble(2);
                        selectps.close();
                }else{
                        selectps.close();
                        throw new NoSuchEntityException( this.id + "를 찾을 수가 없
군요.(loadRow)");
                }
        }

        private void storeRow() throws SQLException{
                String updateS = "update account set name = ?, balance = ? where id
= ?";
                PreparedStatement updateps = con.prepareStatement(updateS);
                updateps.setString(1, this.name);
                updateps.setDouble(2, this.balance);
                updateps.setString(3, this.id);

                int updateCount = updateps.executeUpdate();
                updateps.close();

                if(updateCount == 0){
                        throw new EJBException("수정된 데이터가 없습니다.");

                }
        }

} // end class
```

AccountEJB.java 끝 ----------

e. 디플로이먼트 하기

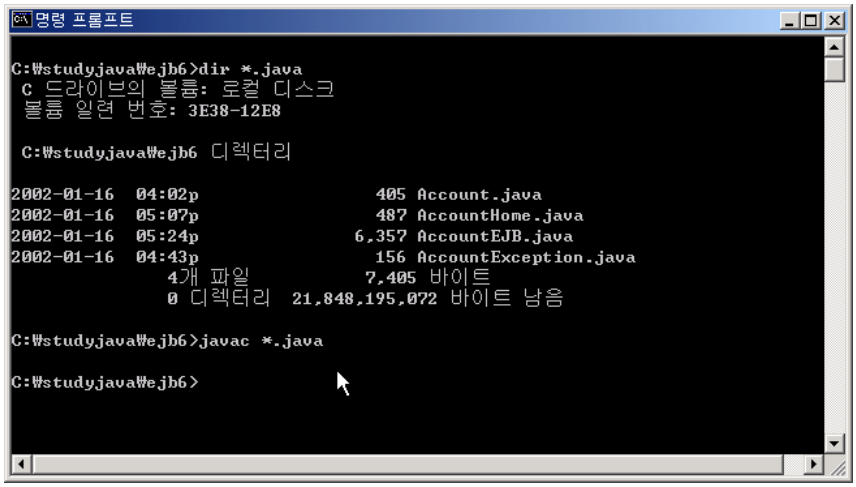

그림 141 홈 인터페이스, 리모트 인터페이스, 빈 클래스, 사용자 정의 Exception을 컴파일 합니다.

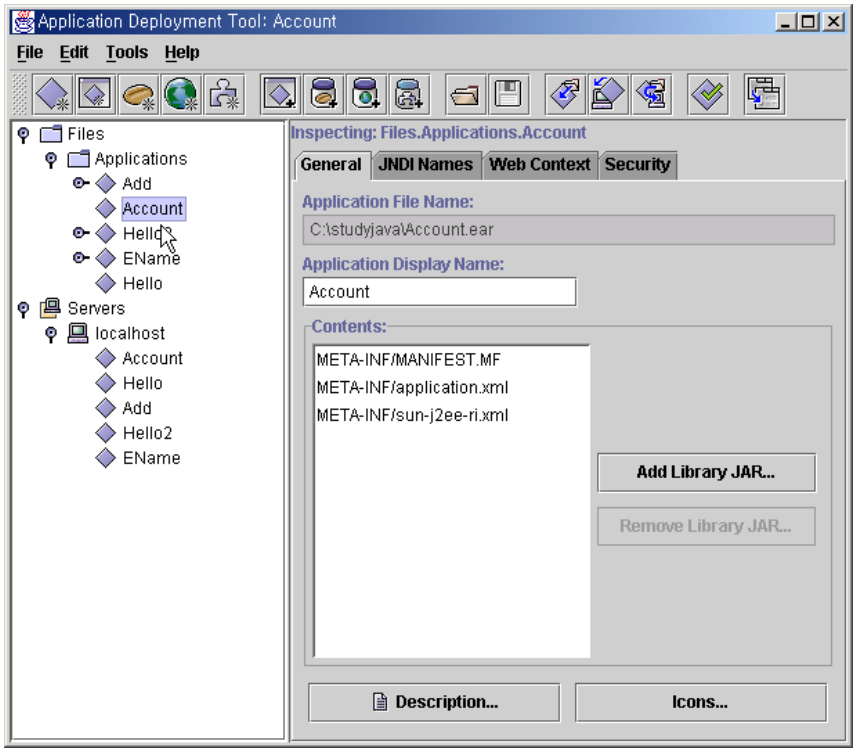

그림 142 deploytool을 실행한 후 File – New – Application 에서 account를 생성한 화면입니다.

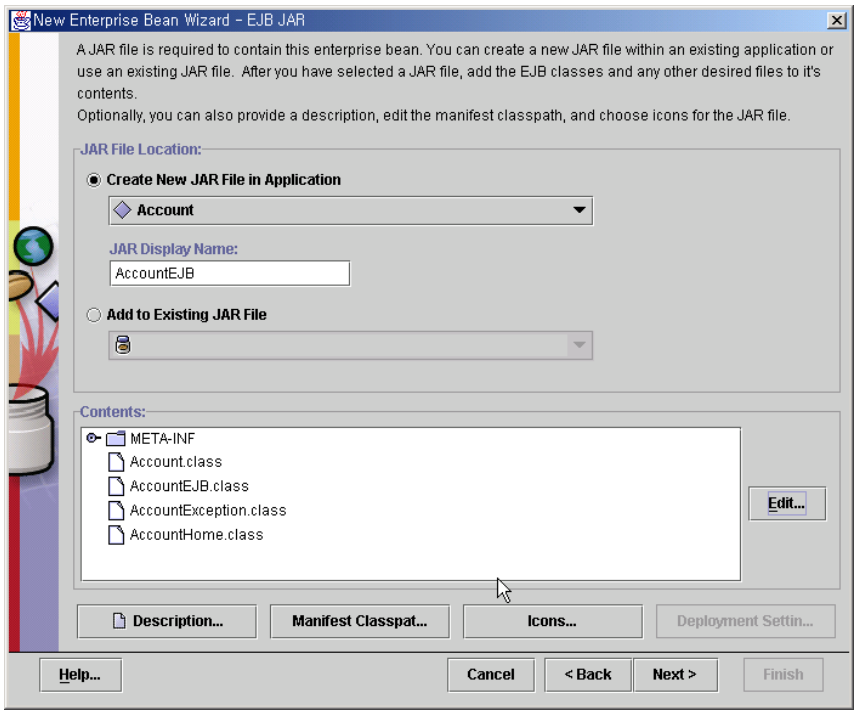

그림 143 File-New-Enterprise Bean을 선택합니다. 그림과 같이 설정한 후 Next버튼을 클릭합니다.

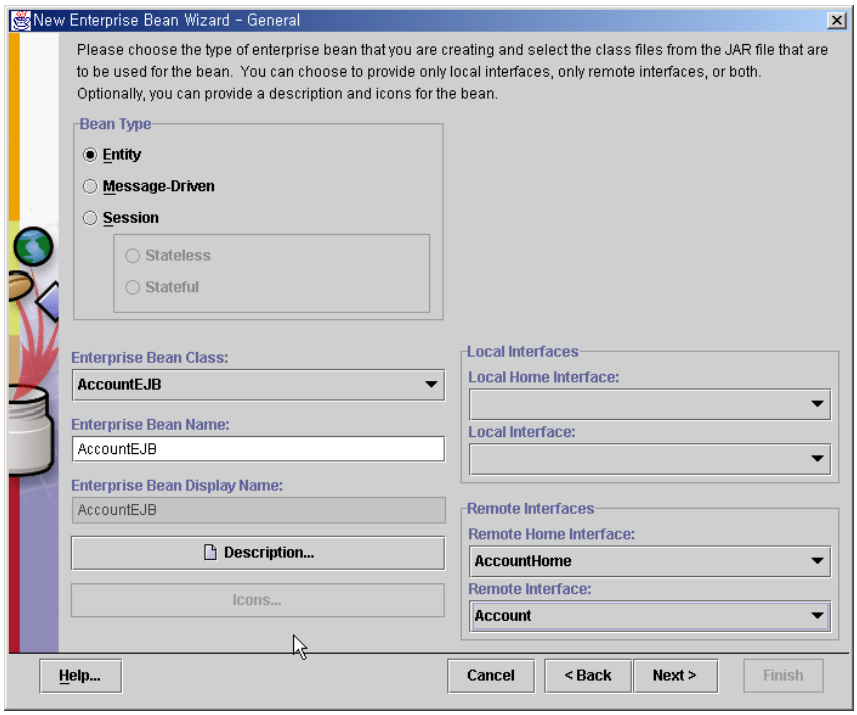

그림 144 그림과 같이 설정한 후 Next버튼을 클릭합니다.

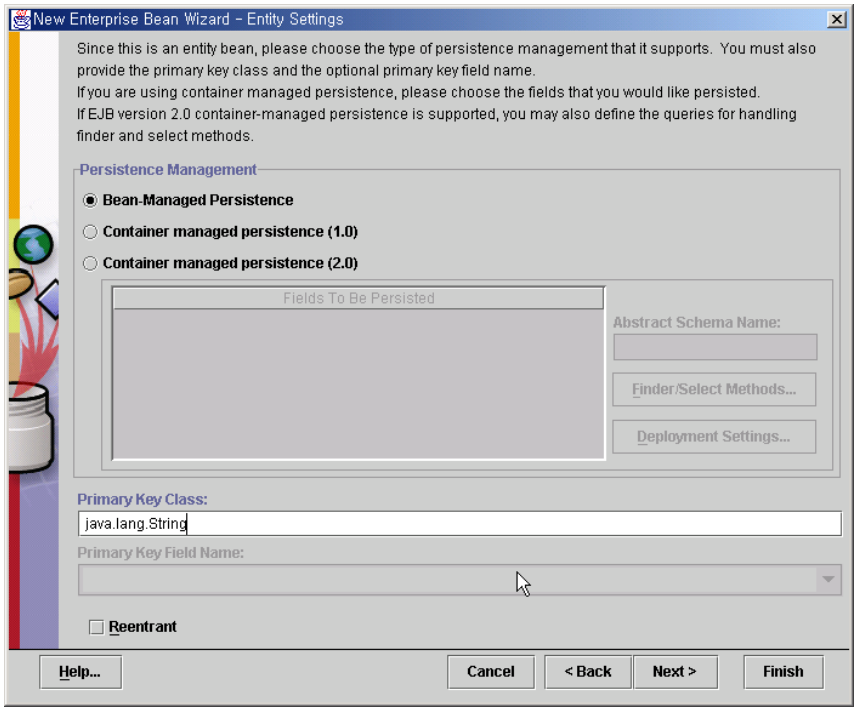

그림 145 Bean Managed Persistence를 선택한 후 Primary Key Class를 java.lang.String으로 변경합니다.

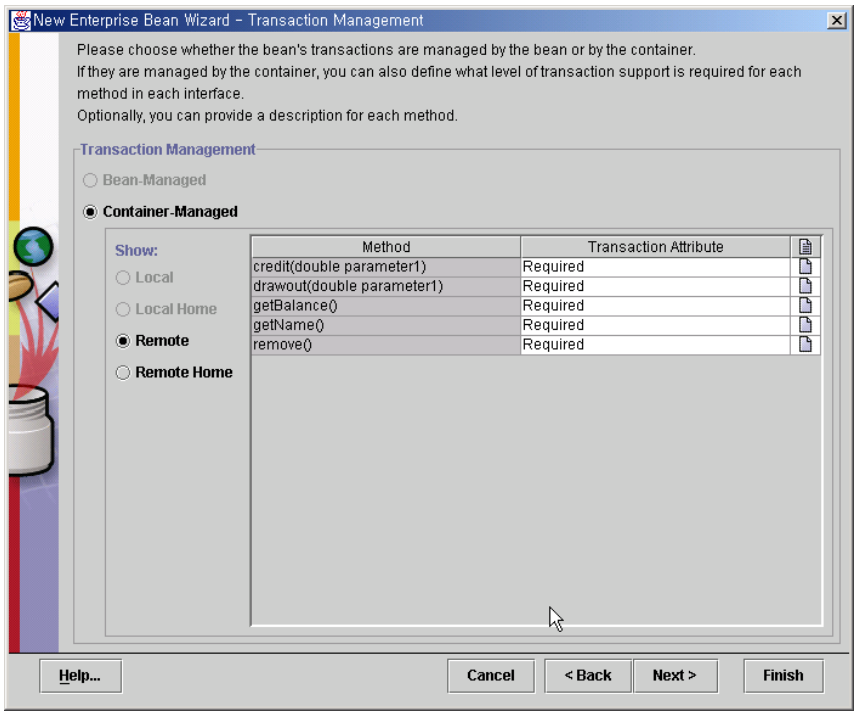

그림 146 그림과 같이 설정한 후 Next버튼을 클릭합니다.

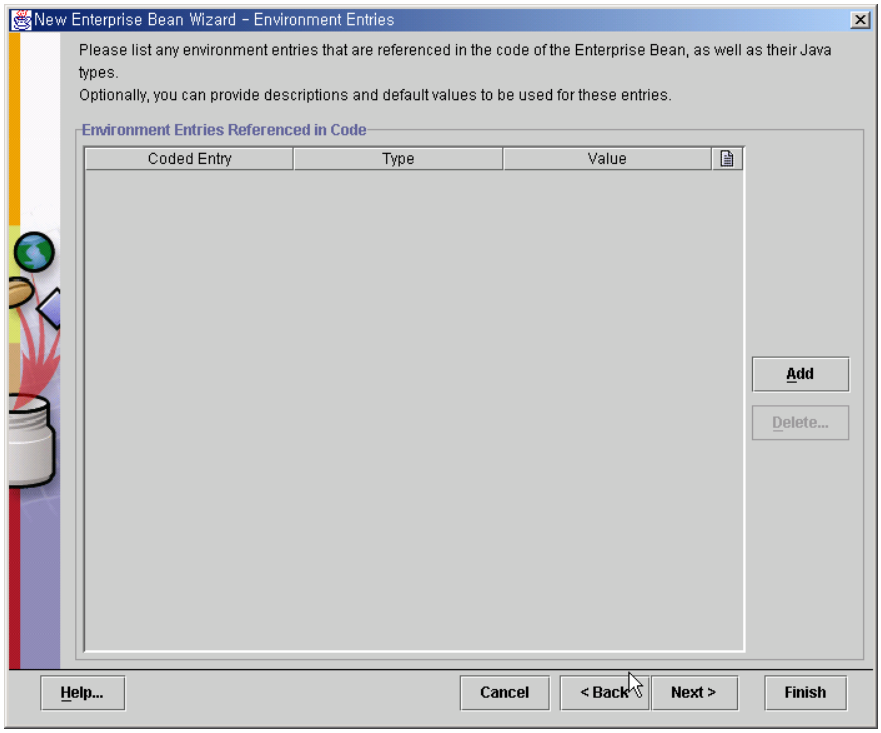

그림 147 Next버튼을 클릭합니다.

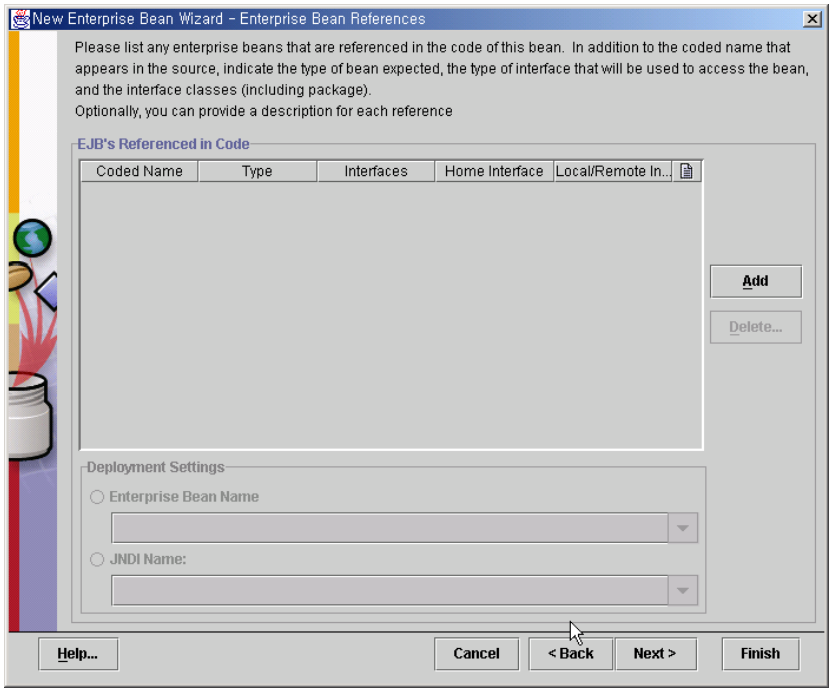

그림 148 Next버튼을 클릭합니다.

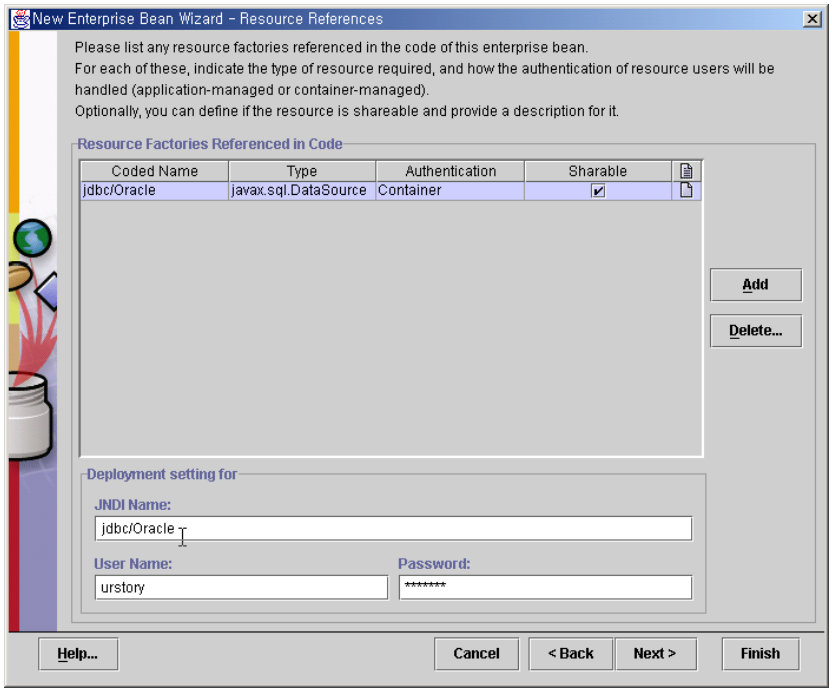

그림 149 그림과 같이 설정합니다. 데이터 베이스에 접속하기 위한 설정이므로 중요합니다. UserName 과 Password는 사용하고 있는 오라클의 userName과 Password를 작성해주시면 됩니다. 설정하였다면 Finish버튼을 클릭합니다.

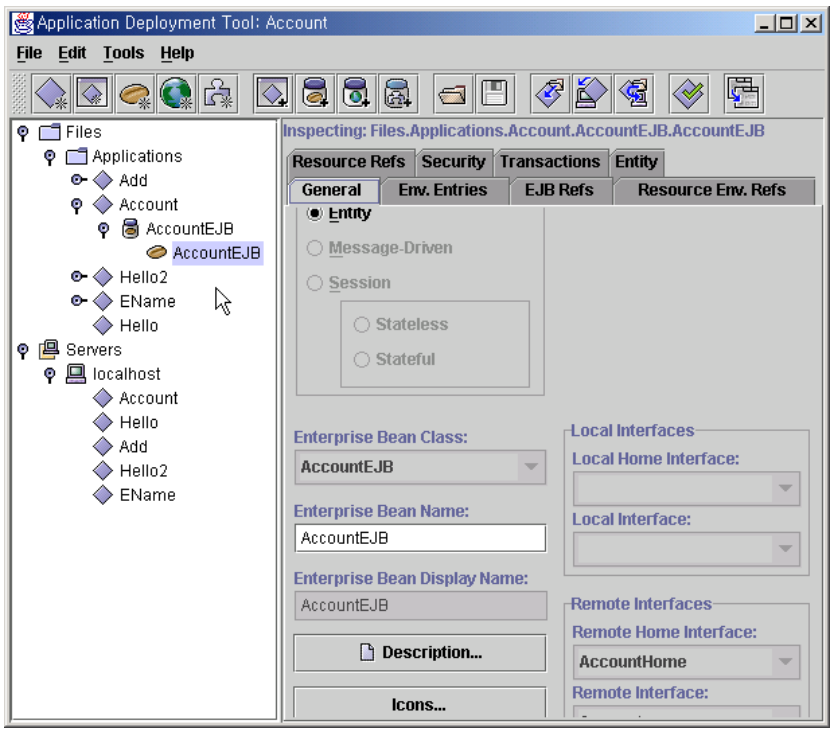

그림 150 그림과 같이 엔터프라이즈 빈이 추가된 것을 알 수 있습니다.

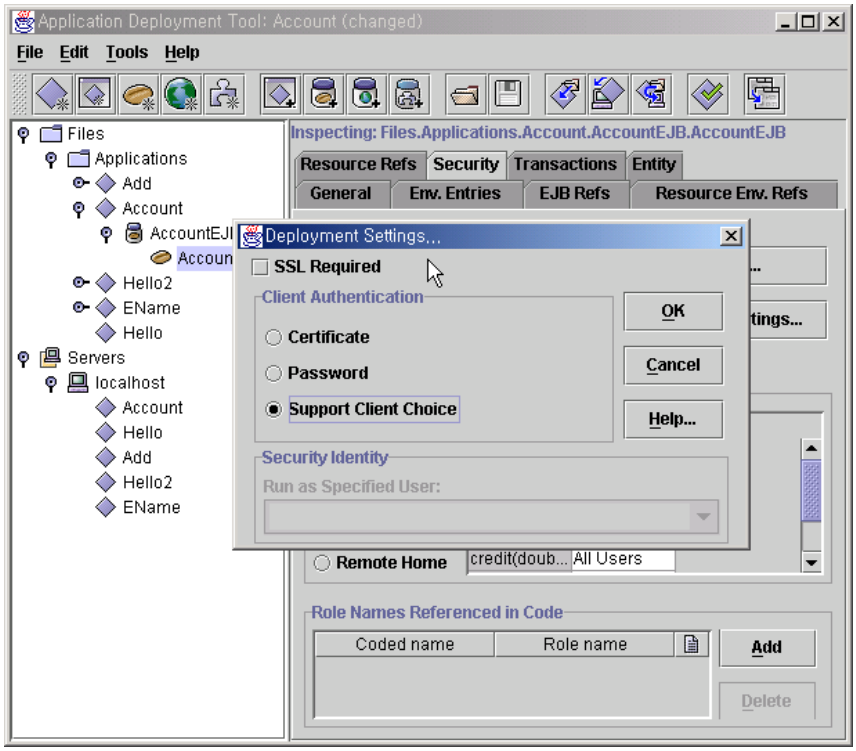

그림 151 Security tab에서 보안설정을 그림과 같이 합니다.

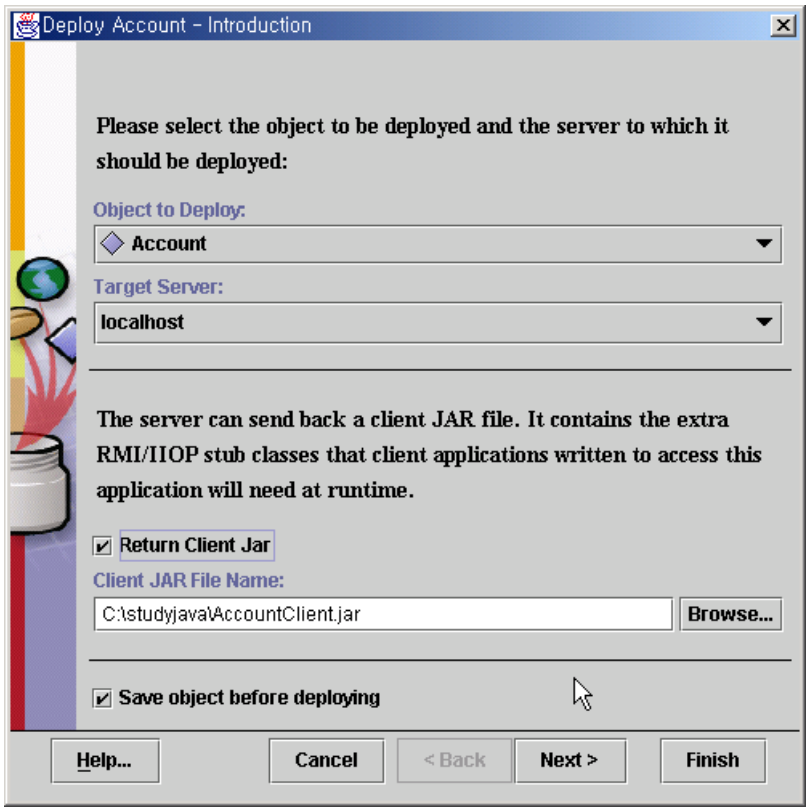

그림 152 Tools – deploy를 선택합니다. 그림과 같이 셋팅한 후 Next버튼을 클릭합니다.

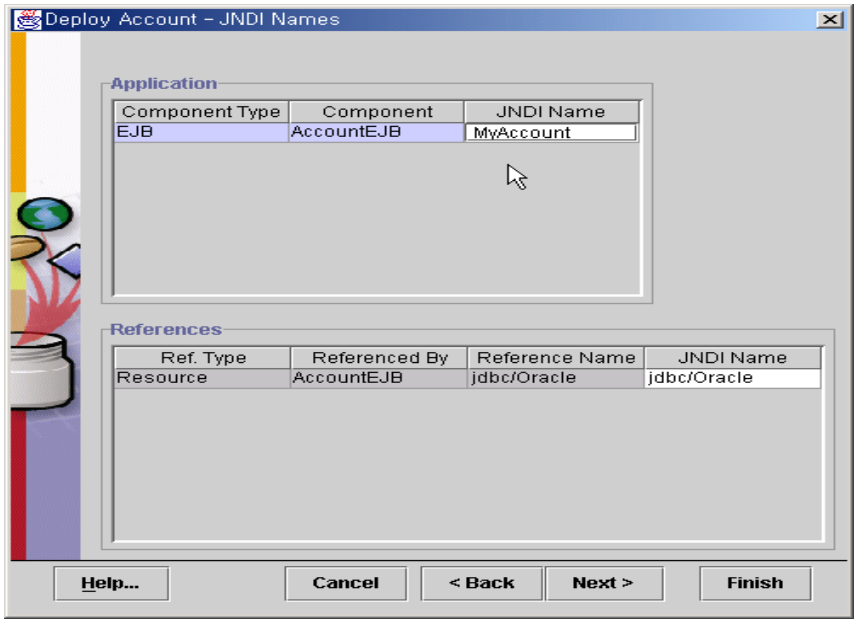

그림 153 JNDI Name을 설정한 후 Next버튼을 클릭합니다.

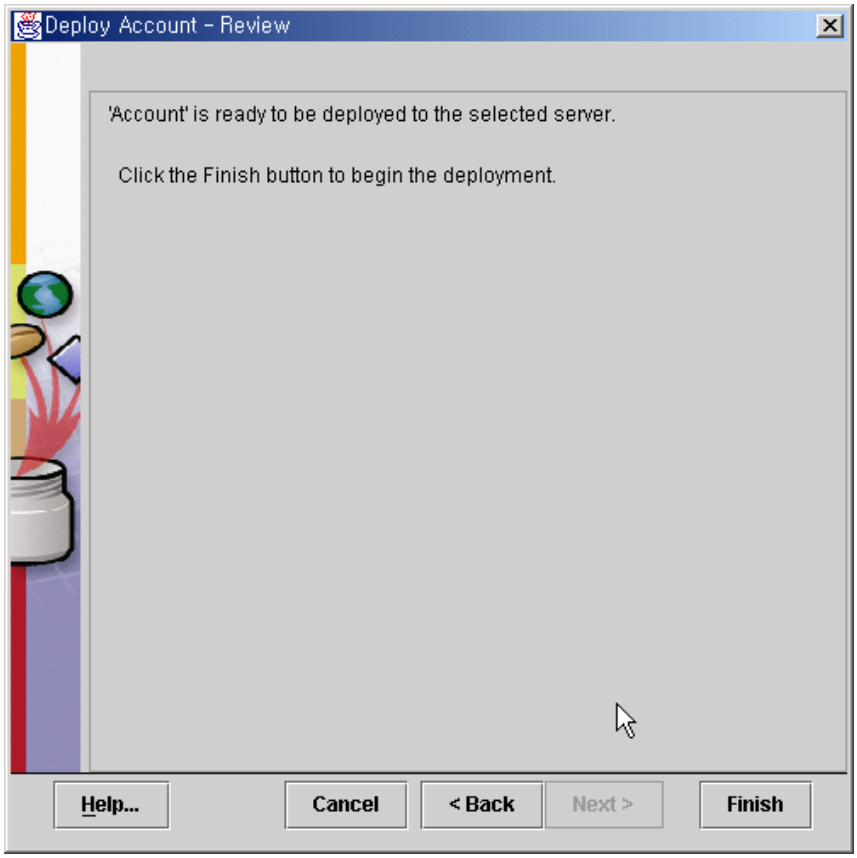

그림 154 Finish버튼을 클릭합니다.

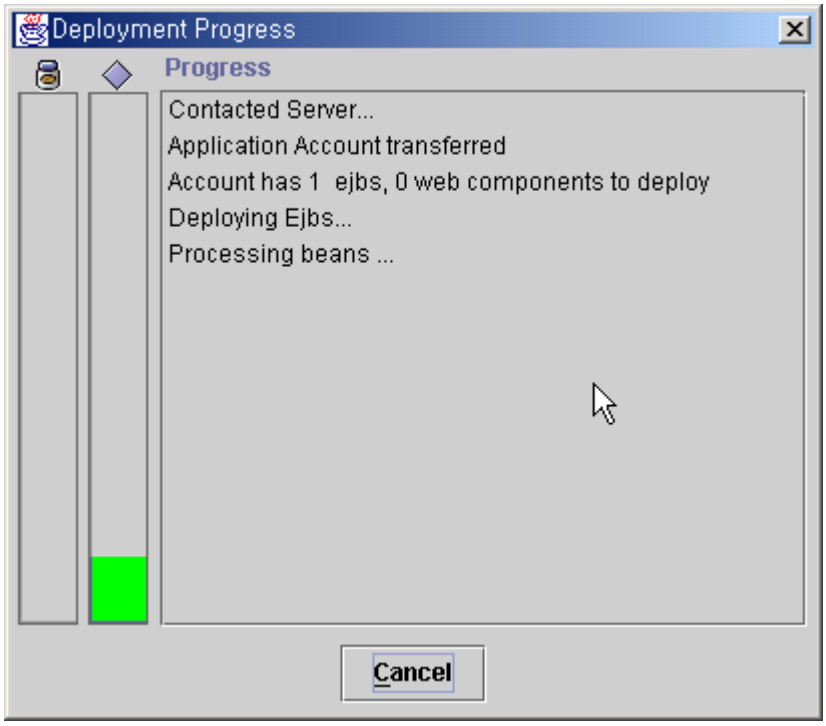

그림 155 디플로이 되는 과정이 그래프와 함께 표시됩니다.

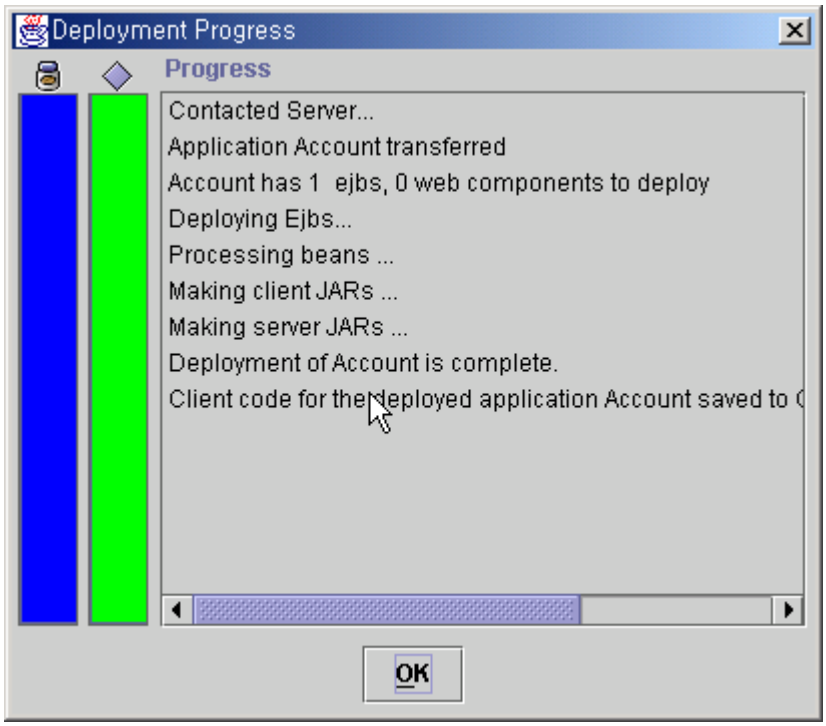

그림 156 디플로이가 끝났습니다.

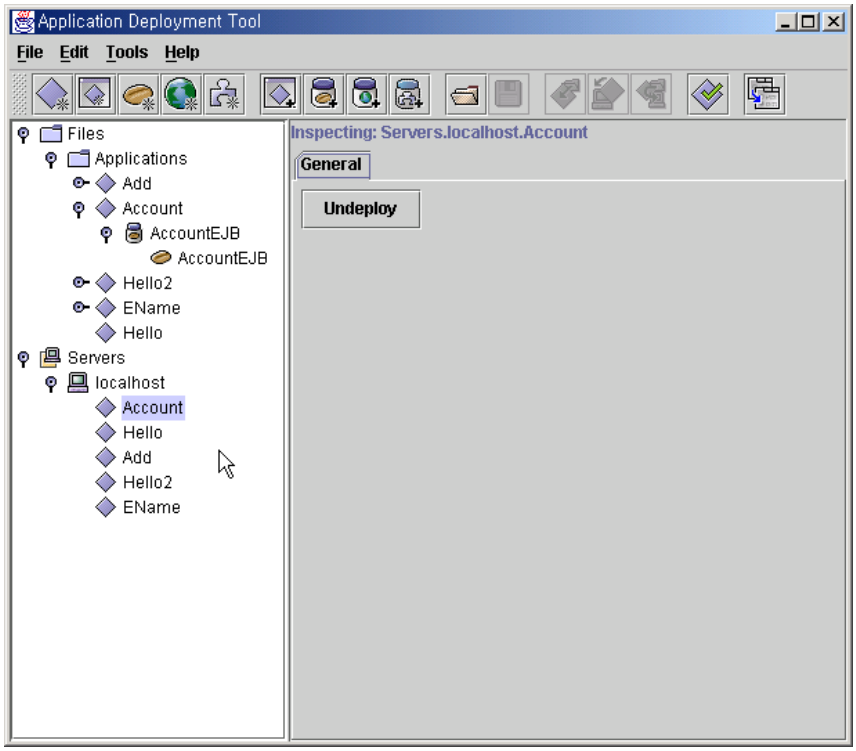

그림 157 Server에 Account가 추가된 것을 알 수 있습니다.

f. 클라이언트 프로그램의 작성과 실행

AccountClient.java는 BMP엔티티 빈을 이용하기 위한 클라이언트 소스입니다.

AccountClient.java 시작 ----------

```
import javax.naming.*;
import javax.rmi.*;

public class AccountClient{
        public static void main(String args[]){
                try{
                        Context initial = new InitialContext();
                        Object obj = initial.lookup("MyAccount");
                        AccountHome home =
(AccountHome)PortableRemoteObject.narrow(obj, AccountHome.class);
                        Account kim = home.create("1", "kim", 0.00);
                        kim.credit(200.00);
                        kim.drawout(100.90);
                        double balance = kim.getBalance();
```

```
                System.out.println("kim의 잔고 :" + balance);
            }catch(Exception e){
                e.printStackTrace();
            }
        } // end main
    }// end class
    AccountClient.java 끝 ------------------------------------------
```

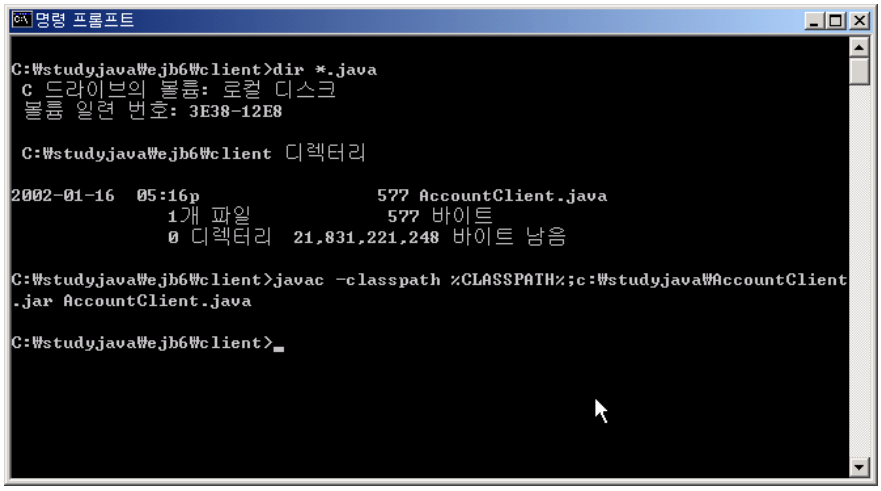

그림 158 AccountClient의 컴파일

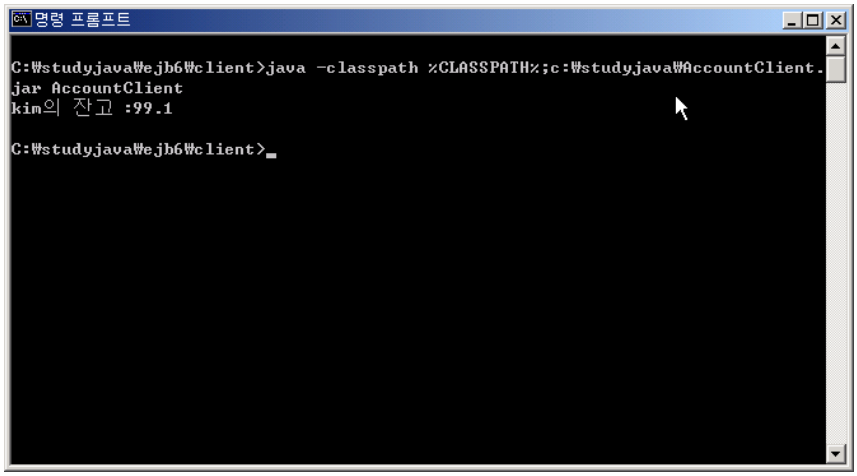

그림 159 AccountClient의 실행

AccountClient에서 BMP 엔티티빈이 가지고 있는 다른 메소드는 어떻게 실행할까?

12. 마치며.....

자바는 역시 쉽지는 않은 것 같다. 라는 생각이 듭니다. 다만 단순하다. 라는 표현이 알맞을 까요? 너무 복잡하게 생각하면 '왜 이렇게 해야만 되지?' 라는 생각만 머리에서 맴돌뿐입니다. 하지만, 단순하게 이렇게 하라니 이렇게 하자. 라고 생각한다면 문제는 의외로 쉽게 풀립니다. **JAVA를 공부하고 EJB공부를 처음 시작하는 사람은, EJB를 소개하고 있는 문서에 먼저 압도되는 것이 사실입니다. 그리고 나서 예제도 제대로 실행하지 못하고 포기하고 맙니다.** 어떤 분은 상용 EJB 컨테이너인 웹 로직 같은 소프트웨어를 설치하다가 포기하는 경우도 보았습니다. 그렇게 실패하고 포기하는 이유는 전체적인 모습을 한번도 본적이 없어서 그렇다. 라고 말해도 과언이 아닙니다.

EJB는 썬사에서 발표한 표준입니다. 그리고 J2EE는 모든 EJB컨테이너가 참고로 하는 소프트웨어입니다. EJB를 이해하기 위하여는 J2EE로 먼저 공부하는 것이 가장 좋다. 라는 것이 저의 생각입니다. 그리고 나서 나중에 상용 EJB컨테이너를 이용하는 것이 알맞을 것 같습니다.

해당 문서만 보아서는 EJB에 대하여 자세히 알 수는 없습니다. 꼭 EJB참고 서적을 구입하신 후 함께 보시기 바랍니다. 물론 책을 사기전에 똑같이 따라 해 볼 수 는 있습니다. EJB를 만드는 과정을 꼼꼼하게 그림으로 모두 표시해 놓았으니 직접 실행하지 않아도 이해하리라 생각합니다.

내공이 깊으면 연공이 자유롭다고 합니다. 사실 EJB를 올바르게 공부할려면 EJB에 대하여 체계적으로 공부해야 합니다. 하지만 해당 문서는 귀납적인 방법을 택하였습니다. 먼저 결과를 보고, 스스로 공부하도록 만든 것입니다.

그래서 먼저 '아! EJB라는 것은 이런 것이구나. 스펙이 이런 것이구나!' 라는 것을 깨닫게 만들고 싶었습니다. 빠른 시간에 작성한 문서라 틀린 부분이 있을 수도 있습니다. 틀린 부분을 발견한다면 저에게 메일(urstory@nownuri.net)을 보내주십시오.

꼬랑지말 ; 해당 문서의 제목은 Enterprise Java Bean #1 입니다. #2 는 언제쯤 작성할까요?

김 성박. 2002년 1월 17일 목요일.
하나님은 당신을 사랑하십니다.