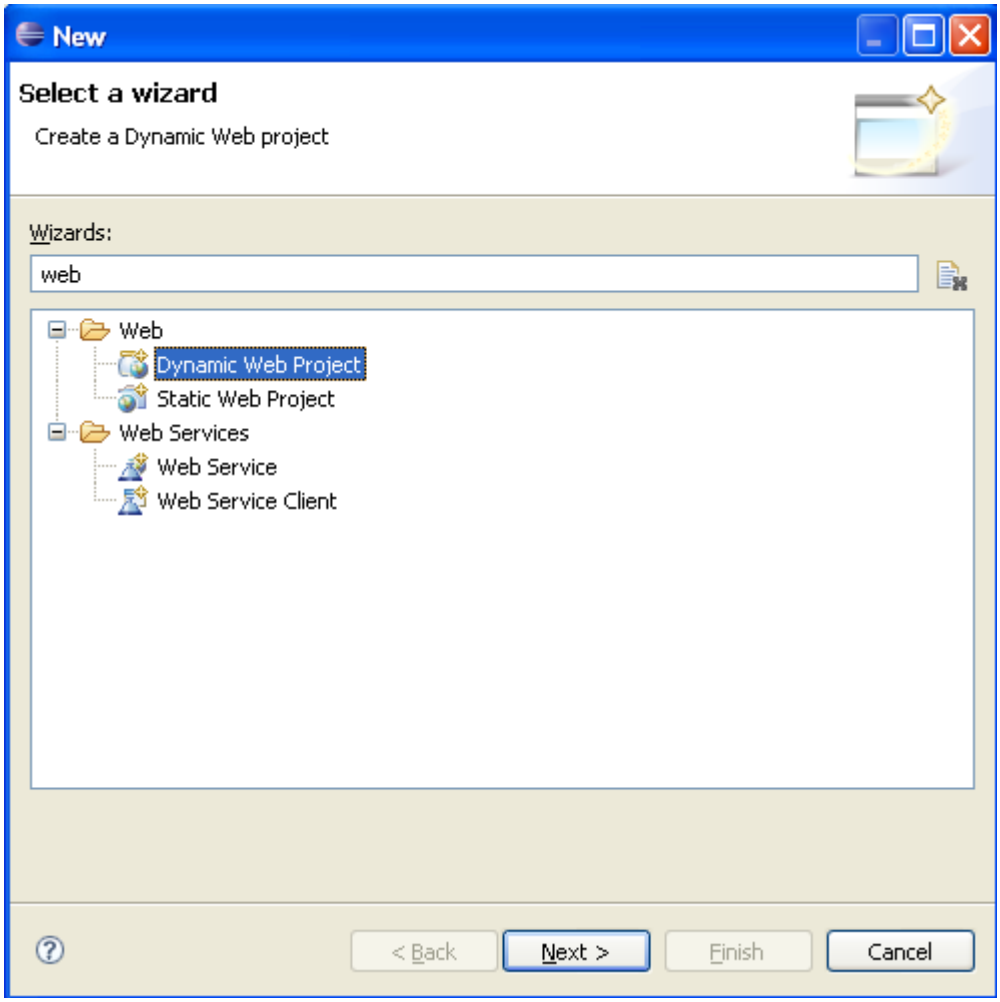
EJB - Tworzenie aplikacji opartych na komponentach

Cel zadania: Utworzenie aplikacji internetowej – Egzaminy wykorzystującej komponenty EJB 3.0 jako reprezentacja warstwy biznesowej

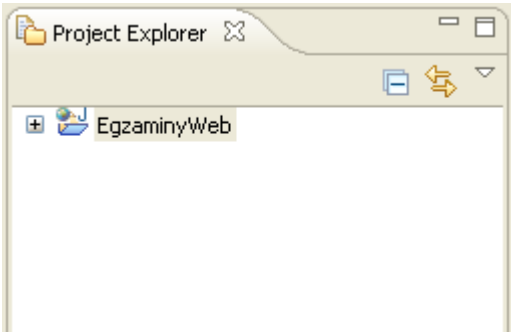
Tutorial przygotowany na podstawie materiałów (wraz z opisem przez Jacka Laskowskiego). W materiałach pojawia się wzmainaka o serwerze GlassFish – my oczywiście zmieniamy go na JBoss.

1. Tworzenie projektu aplikacji Web

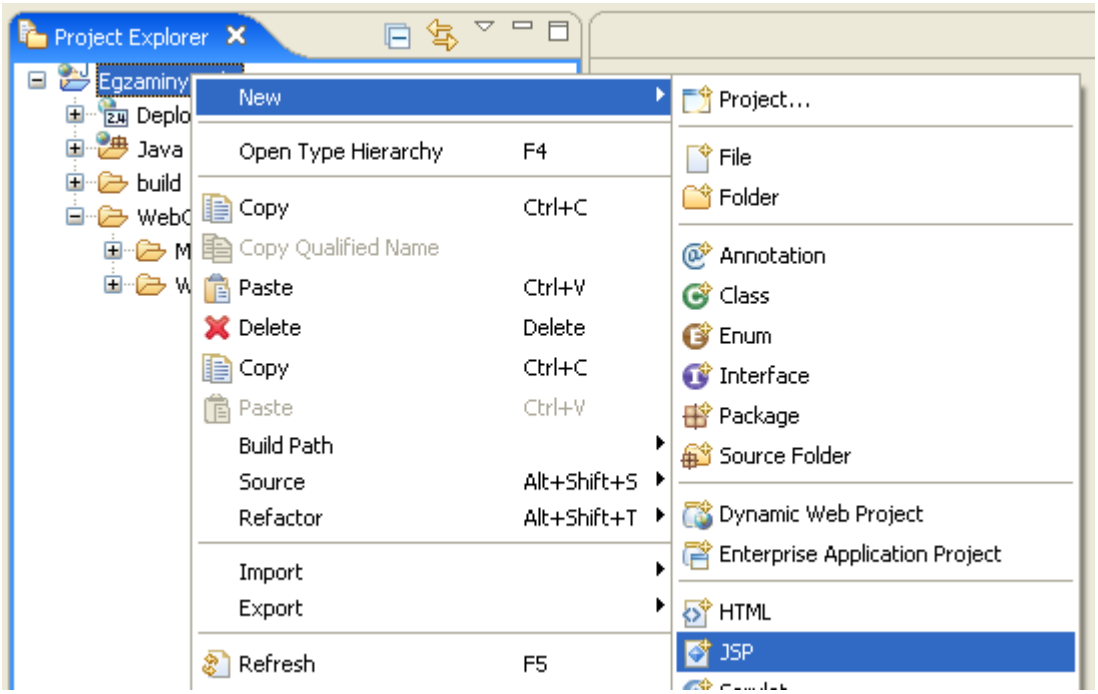
Wybieramy **File->New->Other...**, w pole **Wizards** wpisujemy **web** i wybieramy pozycję **Dynamic Web Project**.



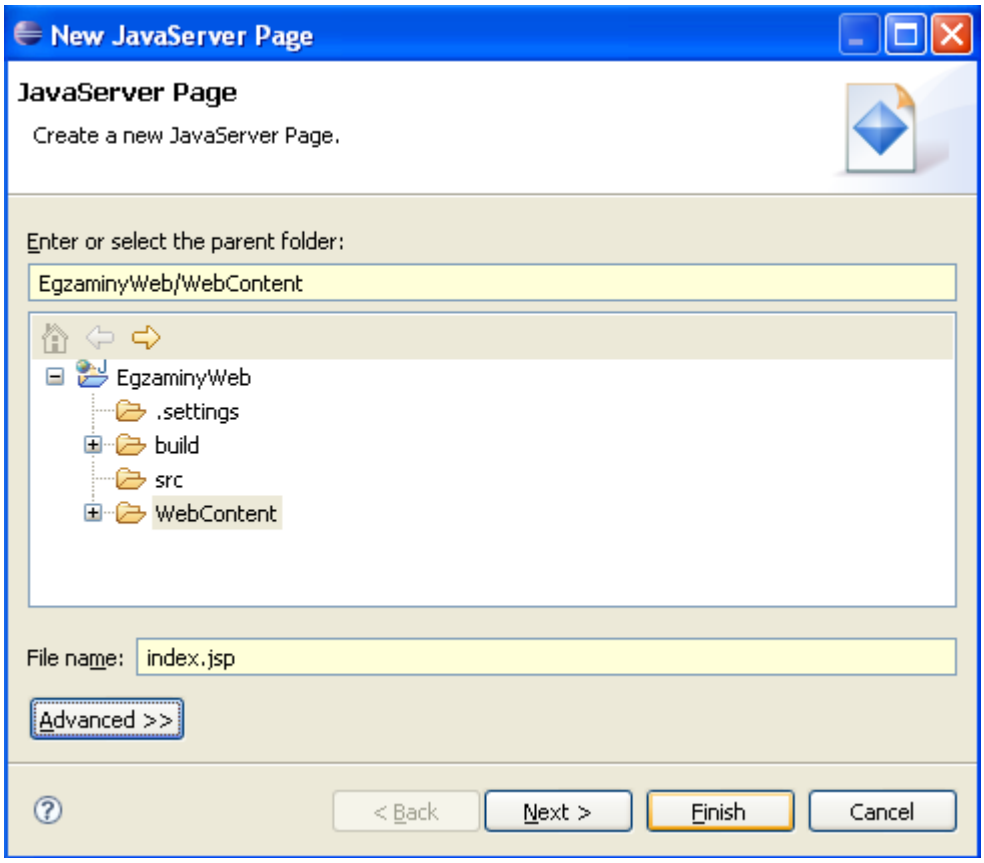
Jako **Project name** podajemy **EgzaminyWeb**, resztę pozostawiając bez zmian.



W projekcie EgzaminyWeb tworzymy stronę startową - **index.jsp**. W menu kontekstowym projektu wybieramy **New->JSP**.



W pole **File name** wpisujemy **index.jsp** i wciskamy przycisk **Finish**.

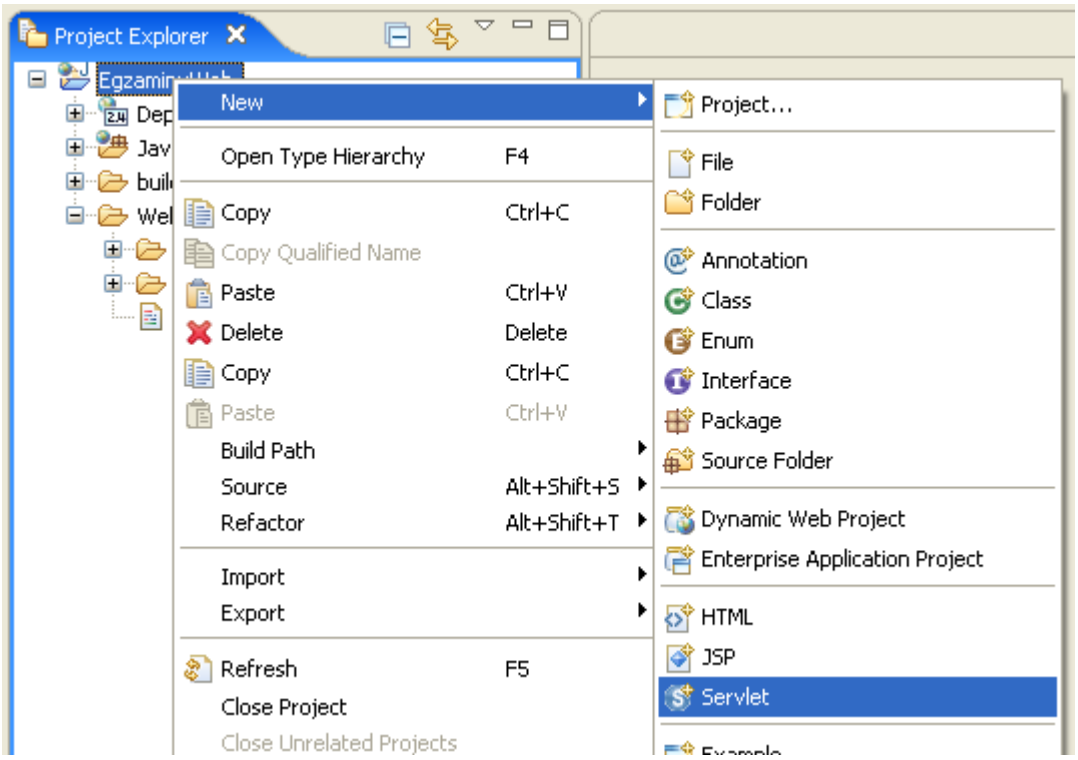


Podmieniamy treść strony index.jsp na poniższą.

```
<%@page contentType="text/html"%>

<%@page pageEncoding="UTF-8"%>


<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>


<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

  "http://www.w3.org/TR/html4/loose.dtd">


<html>

  <head>

    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

    <title>Egzaminy</title>

  </head>
```

```
    <body>

    <form action="ExecuteEjbServlet">
        <input type="submit" value="Dostępne egzaminy" />
    </form>
    <hr>
    <c:forEach items="${exams}" var="exam">
        <p><a href="ExecuteEjbServlet?exam=${exam.name}">${exam.name}</a></p>
    </c:forEach>

    </body>
</html>
```

Podobnie jak miało to miejsce z plikiem index.jsp, tworzymy nowy servlet, którego zadaniem będzie przechwytywanie zleceń ze strony i wywoływanie akcji na komponencie EJB.



W polu **Java package** wpisujemy np. **pl.*TwojeImie*.servlet**, a w **Class name** wpisujemy **ExecuteEjbServlet**.

Podmieniamy zawartość servletu na poniższą.

```
package pl.TwojeImie.servlet;


import java.io.IOException;

import java.util.List;


import javax.ejb.EJB;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;


public class ExecuteEjbServlet extends HttpServlet {

   @EJB

   private ExamScheduler examScheduler;


   protected void service(HttpServletRequest request, HttpServletResponse response) throws
ServletException,

         IOException {

      String page = "/index.jsp";

      List<Exam> exams = examScheduler.getExams();

      request.setAttribute("exams", exams);

      RequestDispatcher rd = getServletContext().getRequestDispatcher(page);

      rd.forward(request, response);
```

```
    }
}
```

Ponownie zapisujemy.

I podnosimy wersję specyfikacji zgodnej z naszą aplikacją internetową do wersji 2.5. Innymi słowy modyfikujemy plik **web.xml** tak, aby ostatecznie wyglądał jak poniżej (plik znajduje się w katalogu **WebContent/WEB-INF** projektu).

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
        <display-name>EgzaminyWeb</display-name>
        <servlet>
                <description></description>
                <display-name>ExecuteEjbServlet</display-name>
                <servlet-name>ExecuteEjbServlet</servlet-name>
                <servlet-class>
                        pl.TwojeImie.servlet.ExecuteEjbServlet
                </servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>ExecuteEjbServlet</servlet-name>
                <url-pattern>/ExecuteEjbServlet</url-pattern>
        </servlet-mapping>
        <welcome-file-list>
                <welcome-file>index.html</welcome-file>
                <welcome-file>index.htm</welcome-file>
                <welcome-file>index.jsp</welcome-file>
```

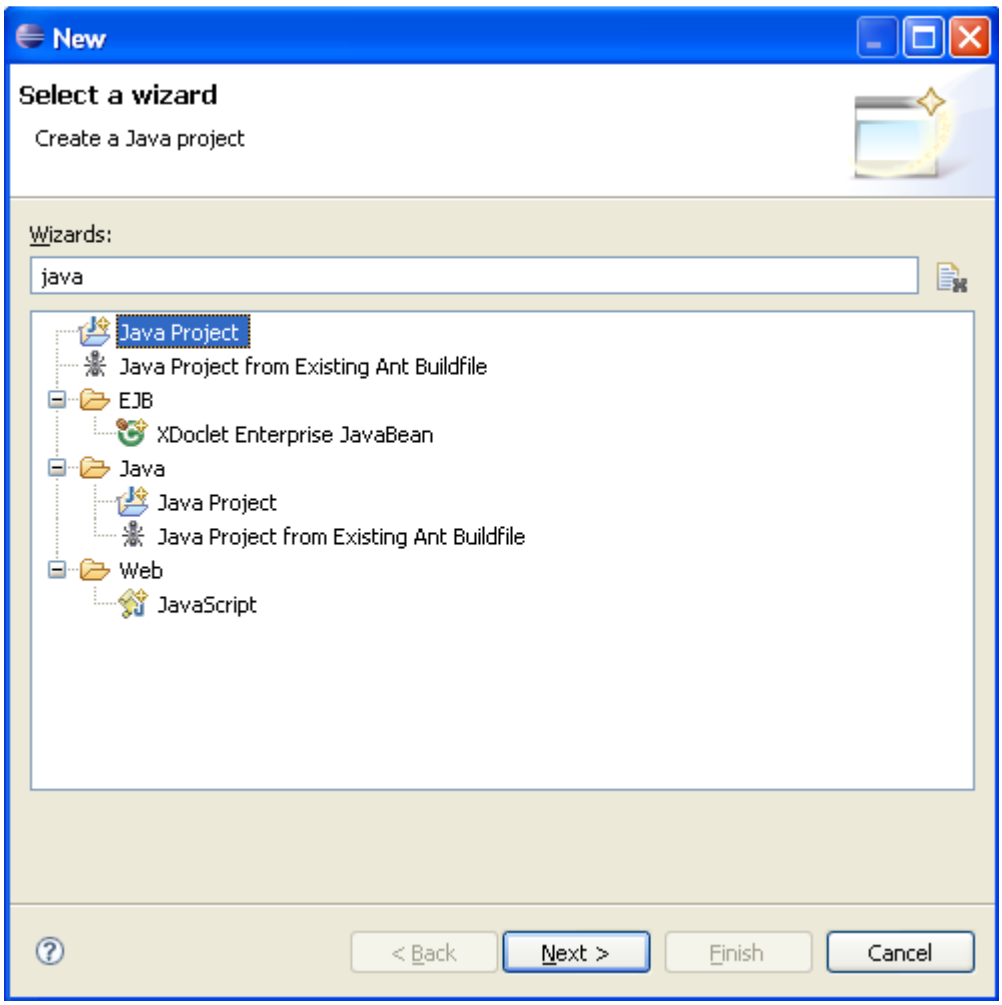
```
            <welcome-file>default.html</welcome-file>
            <welcome-file>default.htm</welcome-file>
            <welcome-file>default.jsp</welcome-file>
        </welcome-file-list>
</web-app>
```

Tymczasowo akceptujemy błędy, które wynikają z nieistniejącej klasy komponentu EJB.

2. Utworzenie modułu EJB - EgzaminyEjb

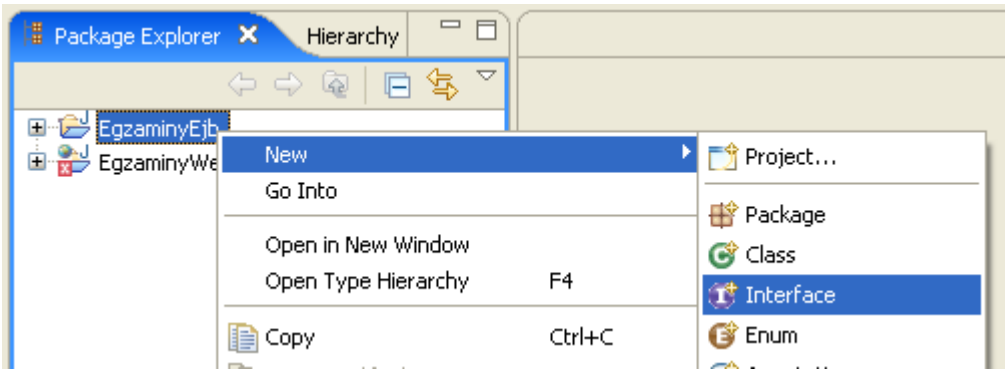
Komponenty EJB w nowej specyfikacji EJB3 są po prostu zwykłymi projektami Java (plikami jar), więc tworzenie ich nie różni się wiele od tworzenia zwykłego projektu Java, którego wynikiem będzie plik jar.

Z menu głównego wybieramy **File->New->Other...**, wpisujemy **java** w pole **Wizards** i wybieramy **Java Project**.



W polu **Project name** wpisujemy **EgzaminyEjb**, resztę pozostawiając bez zmian.

Tworzymy interfejs biznesowy - **ExamScheduler**, czyli innymi słowy najzwyklejszy interfejs Java. Wybieramy z menu kontekstowego projektu menu **New->Interface**.



W pole **Package** wpisujemy **pl.*TwojeImie*.exam.scheduler**, natomiast **ExamScheduler** w pole **Name**.

Podmieniamy zawartość interfejsu ExamScheduler na poniższy.

```
package pl.TwojeImie.exam.scheduler;


import java.util.List;


public interface ExamScheduler {
   List<Exam> getExams();
}
```

Pojawią się błędy związane z niedostępnością klasy Exam, którą właśnie teraz stworzymy.

Z menu kontekstowego projektu EgzaminyEjb wybieramy menu **New->Class**.

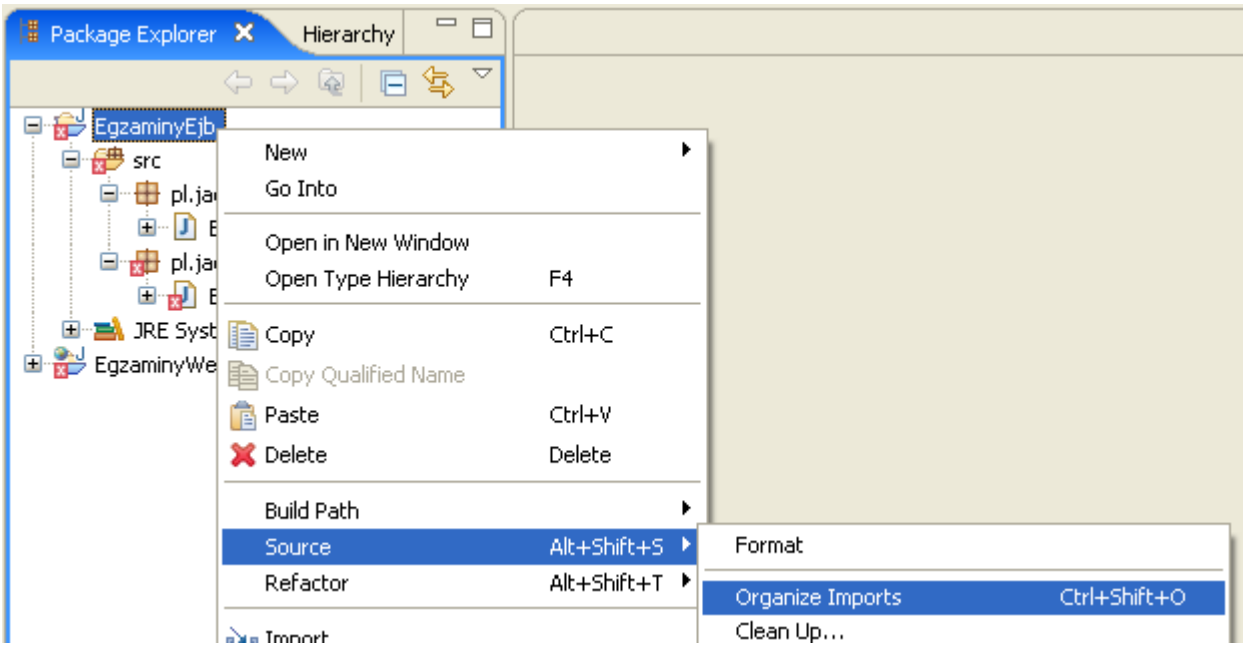
W pole **Package** wpisujemy **pl.*TwojeImie*.exam.beans**, natomiast **Exam** w pole **Name**.

Podmieniamy zawartość klasy Exam na poniższą.

```
package pl.TwojeImie.exam.beans;


public class Exam {
   private String name;
```

```
    public Exam(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

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

Z menu kontekstowego projektu EgzaminyEjb wybieramy menu **Source->Organize Imports**.



Błędy w projekcie EgzaminyEjb powinny zostać naprawione.

Tworzymy klasę komponentu EJB - **ExamSchedulerBean**.

Z menu kontekstowego projektu wybieramy **New->Class** i w
pole **Package** wpisujemy **pl.*TwojeImie*.exam.scheduler** oraz **ExamSchedulerBean** w pole **Name**.

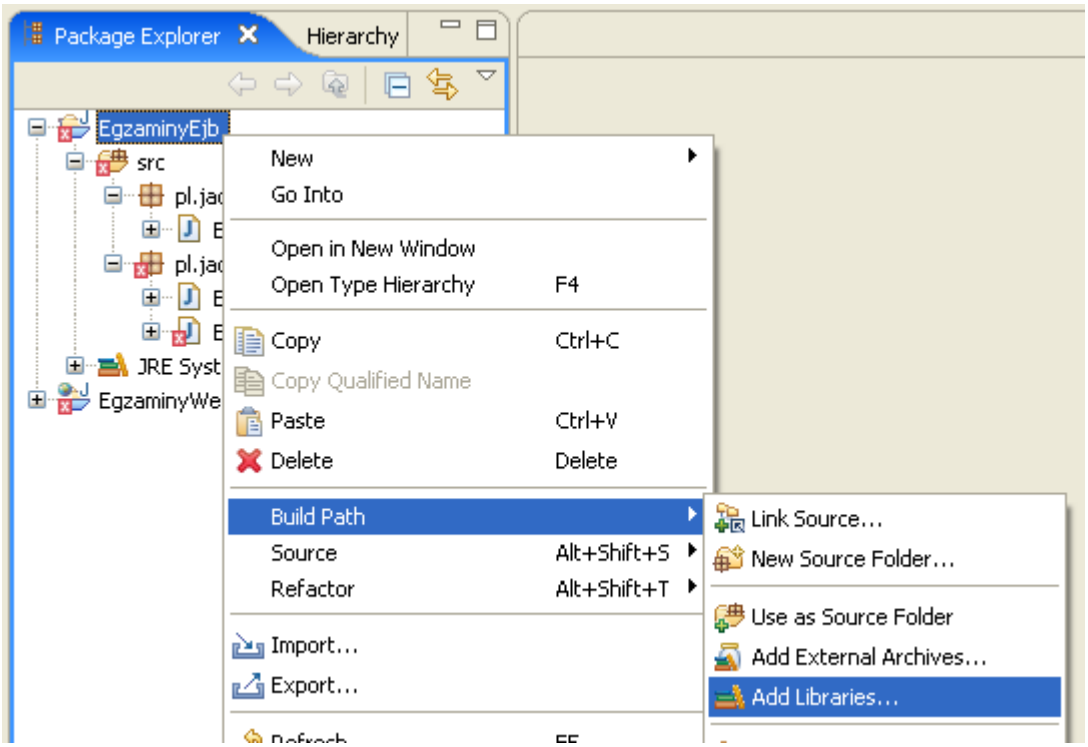
Podmieniamy zawartość klasy ExamSchedulerBean na poniższą.

```
package pl.TwojeImie.exam.scheduler;


import java.util.ArrayList;

import java.util.List;


import pl.TwojeImie.exam.beans.Exam;

@Stateless

public class ExamSchedulerBean implements ExamScheduler {


   private final List<Exam> exams = new ArrayList<Exam>();

   {

      Exam exam = new Exam("Angielski");

      exams.add(exam);

      exam = new Exam("SOA");

      exams.add(exam);

      exam = new Exam("Algorytmy");

      exams.add(exam);

      exam = new Exam("Sieci Komputerowe");

      exams.add(exam);

      exam = new Exam("Systemy Operacyjne");

      exams.add(exam);

     }   }


   public List<Exam> getExams() {
```

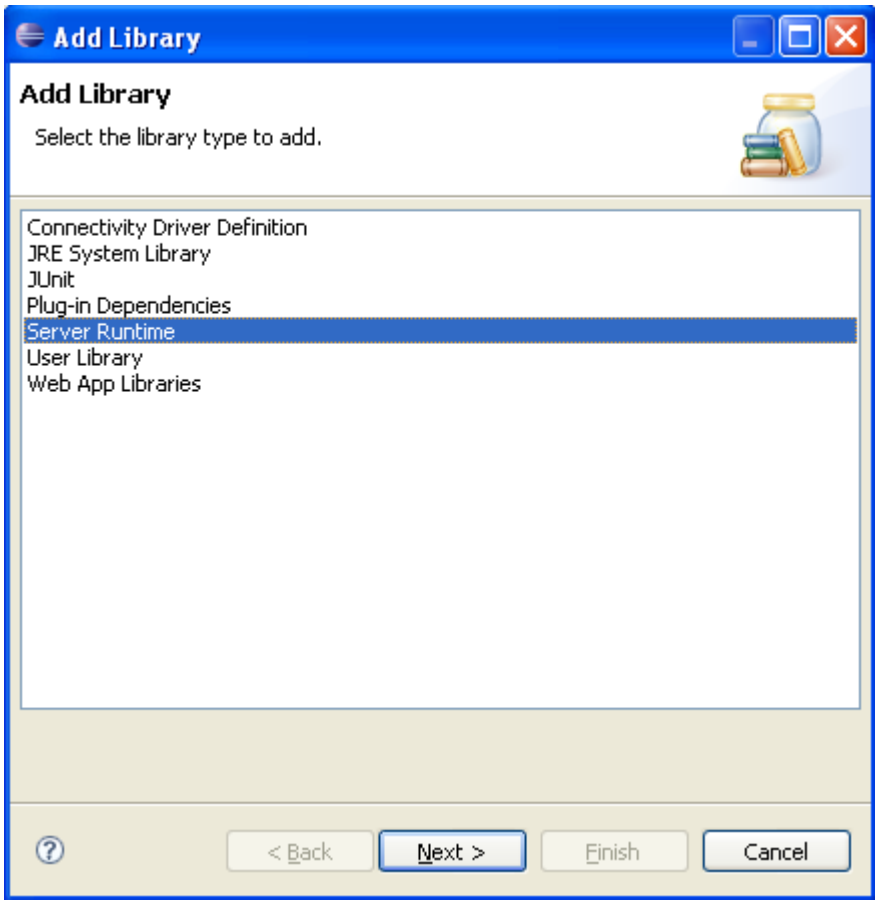
```
        return exams;
    }
}
```

Pojawi się błąd związany z brakiem definicji annotacji **@Stateless**, która znajduje się w pakiecie **javax.ejb** dostarczanym przez definicję serwera aplikacyjnegow Eclipse.

W menu kontekstowym projektu wybieramy menu **Build Path->Add Libraries...**.



Wybieramy pozycję **Server Runtime**.



Wciskamy przycisk **Next >**. Wybieramy **serwer aplikacyjny** z listy (wtedy dopiero uaktywni się przycisk **Finish**).

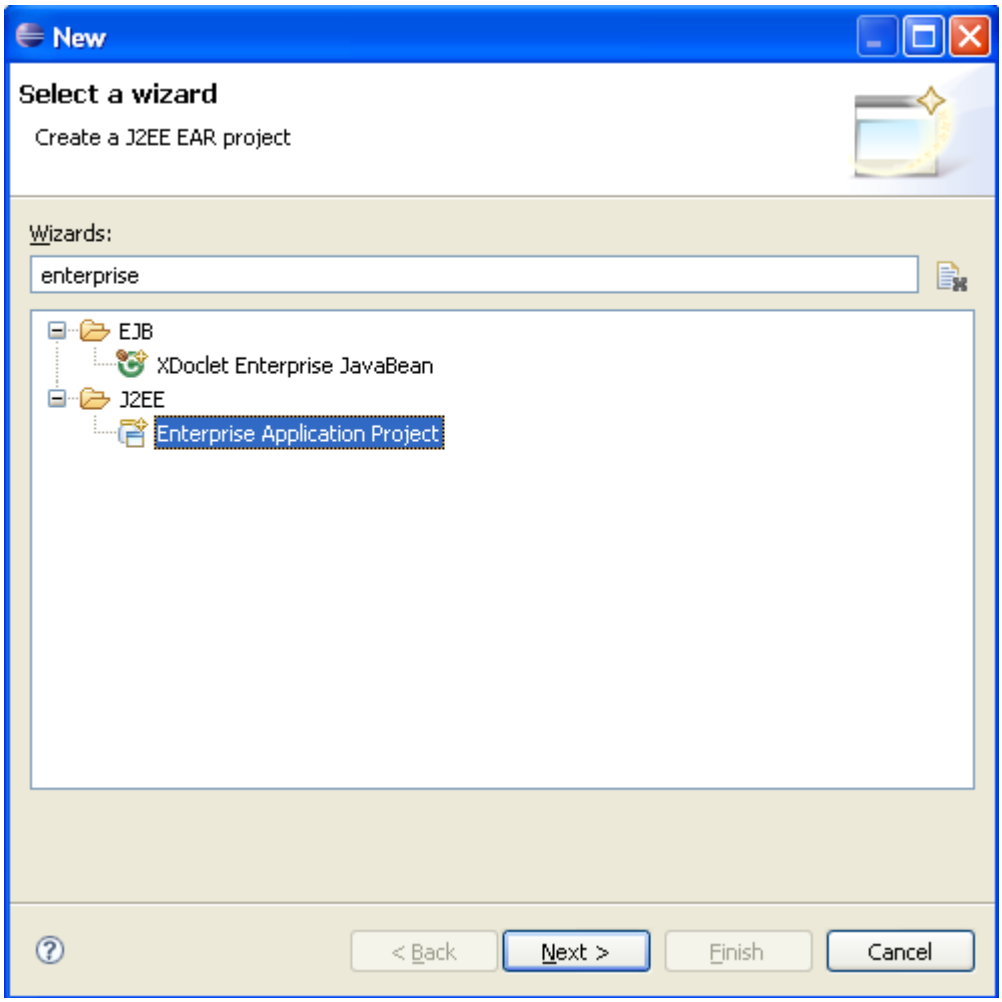
Ponownie z menu kontekstowego projektu wybieramy menu **Source->Organize Imports**, co rozwiąże nasze problemy z brakującą definicją annotacji @Stateless.

Nadal będą zgłoszone błędy związane z projektem EgzaminyWeb, ale do tego wrócimy za moment.

Tym samym skończyliśmy tworzenie komponentu EJB o nazwie ExamSchedulerBean z interfejsem biznesowym (lokalnym) - ExamScheduler.

3. Utworzenie projektu aplikacji przemysłowej - EgzaminyEAR

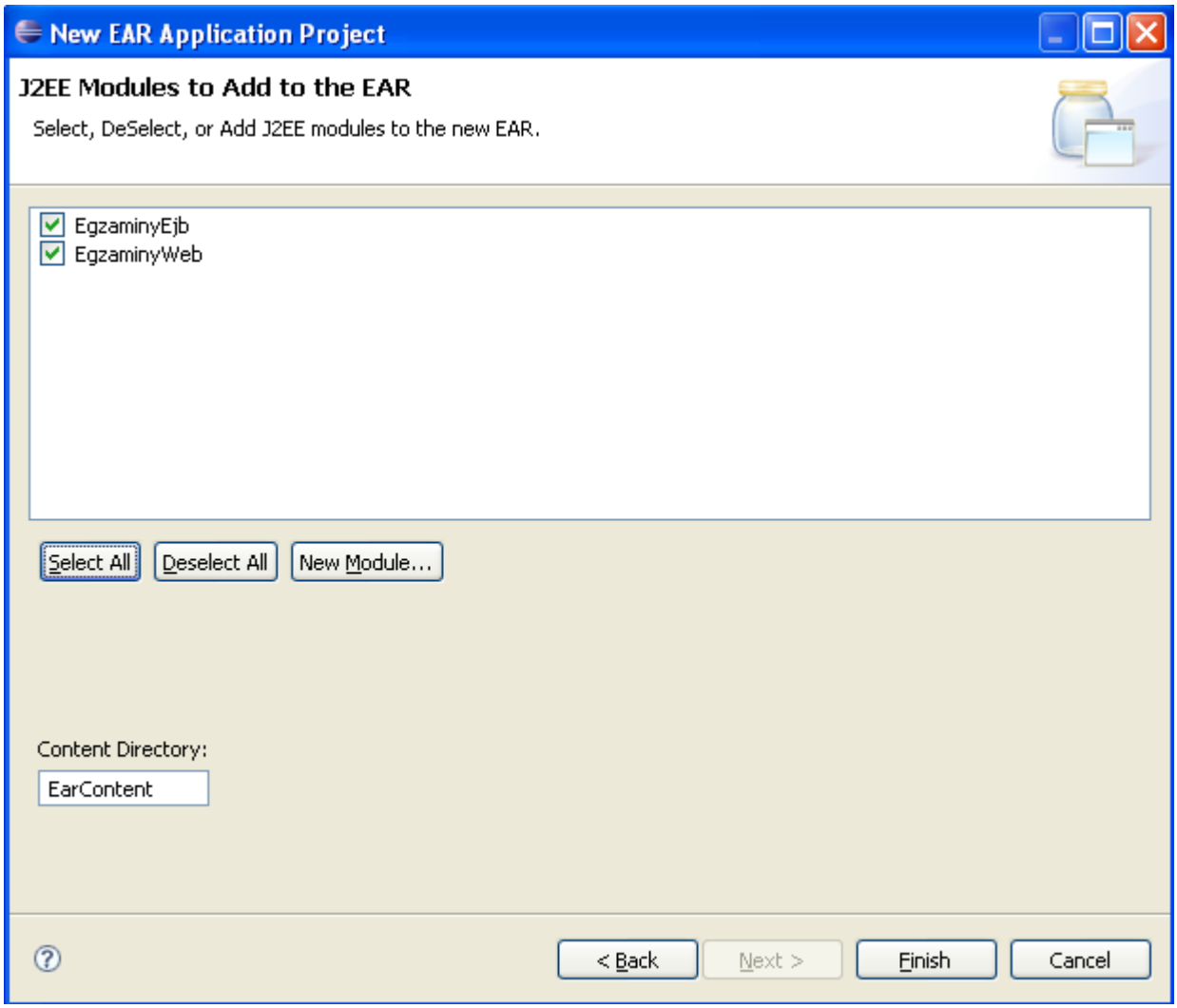
Z menu głównego wybieramy **File->New->Other...**, wpisujemy **enterprise** w pole **Wizards** i wybieramy **Enterprise Application Project**.



Wciskamy przycisk **Next >**.

W polu **Project name** wpisujemy **EgzaminyEar**, resztę pozostawiając bez zmian.

Wciskamy przycisk **Next >** dwukrotnie, aż dojdziemy do formatki **J2EE Modules to Add to the EAR**, gdzie zaznaczamy wszystkie z dostępnych projektów (EgzaminyEjb i EgzaminyWeb) wciskając przycisk **Select All**.



Modyfikujemy zawartość deskryptora instalacji **application.xml** na poniższą (plik znajduje się w katalogu **EarContent/META-INF**).

```
<?xml version="1.0" encoding="UTF-8"?>

<application version="5" xmlns="http://java.sun.com/xml/ns/javaee"

        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/application_5.xsd">

        <display-name>EgzaminyEar</display-name>

        <module>

                <web>

                        <web-uri>EgzaminyWeb.war</web-uri>

                        <context-root>/EgzaminyWeb</context-root>
```

```
        </web>
    </module>
    <module>
        <ejb>EgzaminyEjb.jar</ejb>
    </module>
</application>
```

Zapisujemy zmiany.

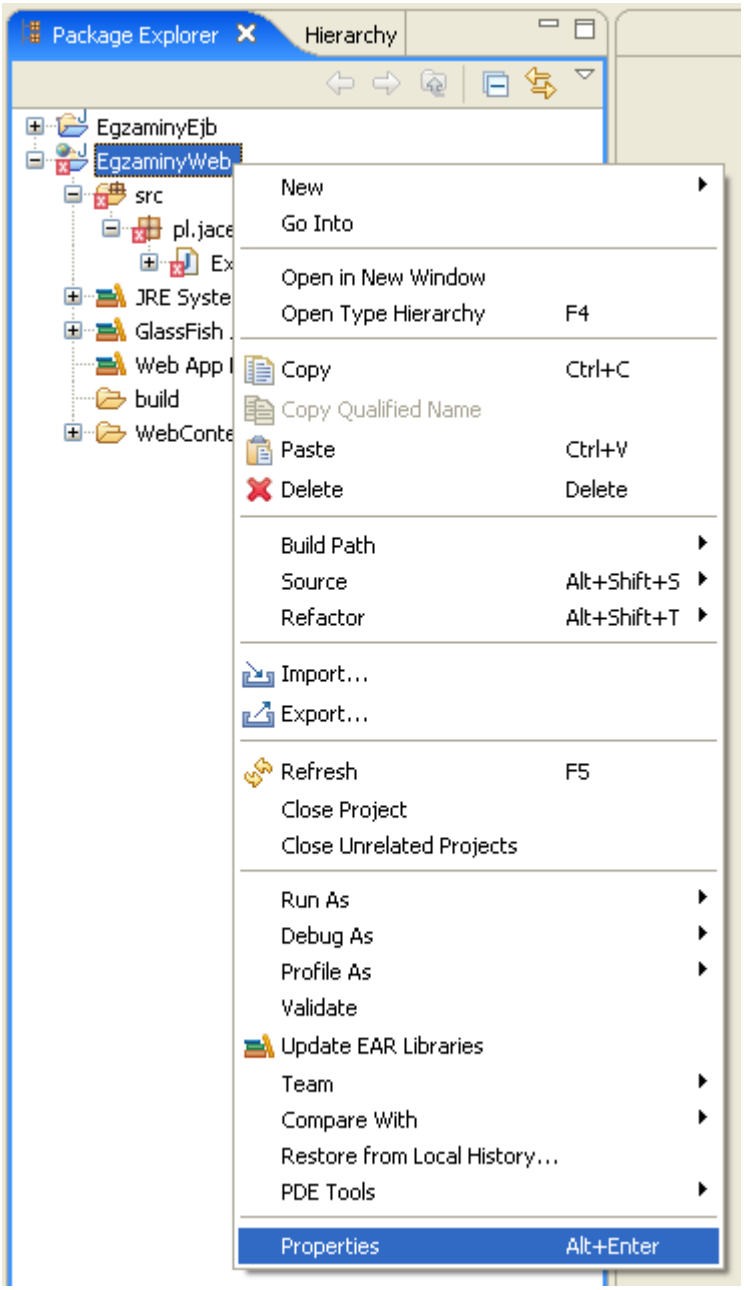
Pojawi się kolejny błąd związany z brakiem deskryptora instalacji dla naszego modułu EJB, ale szczęśliwie nie będzie on nam kolidował z możliwością uruchomienia projektu, kiedy będziemy już do tego gotowi. Innymi słowy: ignorujemy go.

4. Związanie projektów EgzaminyWeb i EgzaminyEjb

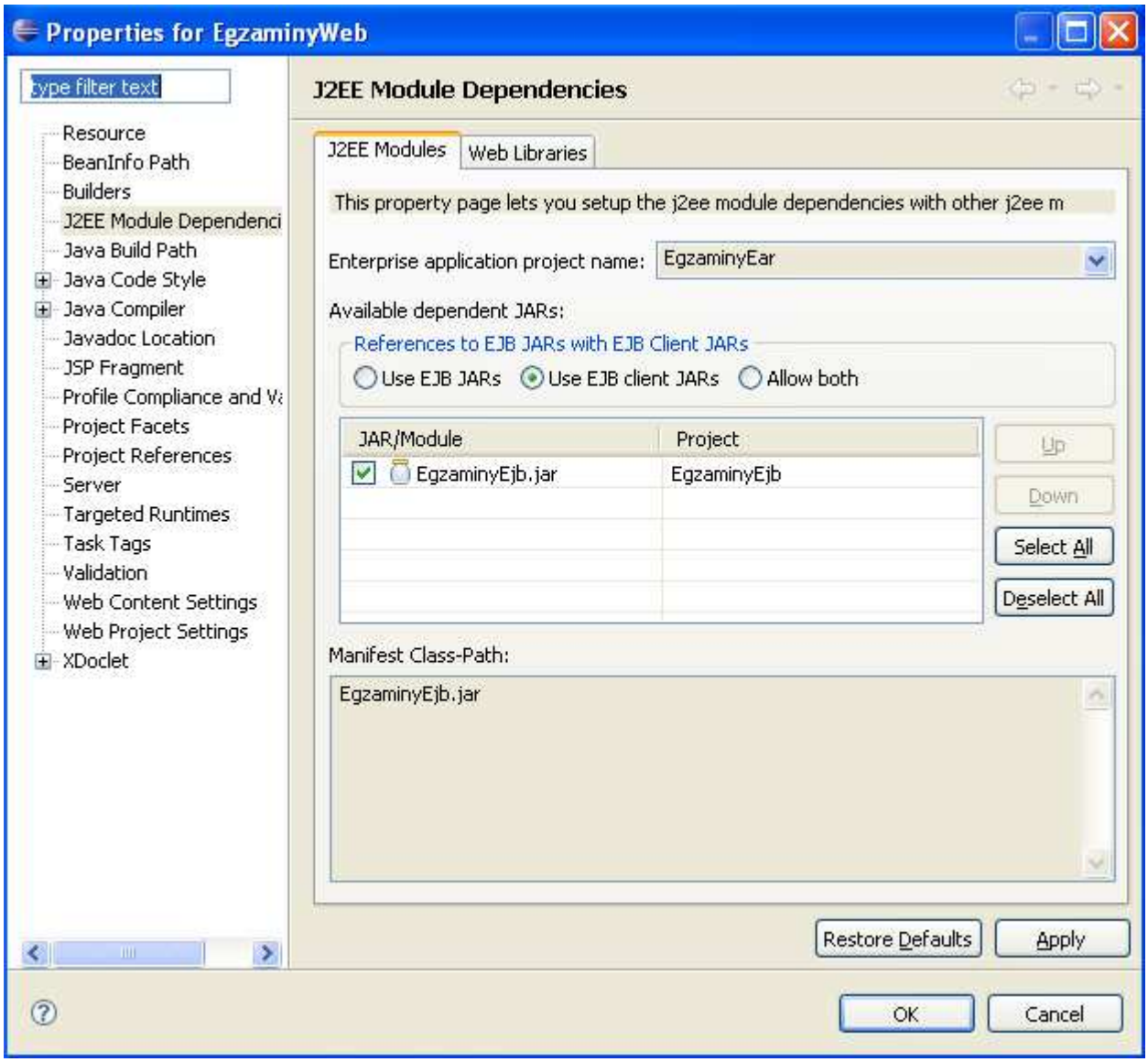
Projekt EgzaminyWeb zależy (choć jeszcze nieformalnie) od EgzaminyEjb. Przypomnę, że korzystamy z annotacji @EJB w servlecie ExecuteEjbServlet do wstrzelenia zależności przez serwer aplikacyjny.

Wróćmy zatem do naszego projektu EgzaminyWeb i zdefiniujmy zależność projektową z EgzaminyEjb.

Z menu kontekstowego projektu EgzaminyWeb wybieramy menu **Properties**.



Wybieramy pozycję **J2EE Module Dependencies**, gdzie zaznaczamy projekt **EgzaminyEjb**.



W ten sposób rozwiązaliśmy błędy związane z niedostępnością klas w projekcie EgzaminyWeb, zgłaszane przez Eclipse. Możemy przystąpić do uruchomienia aplikacji.

5. Uruchomienie aplikacji

Z menu kontekstowego projektu EgzaminyWeb wybieramy pozycję **Run As->Run On Server**.

Po chwili na konsoli powinny pojawić się wpisy o uruchamianiu GlassFish'a oraz postęp wykonania skryptu instalującego aplikację.

Buildfile: C:\.eclipse\javaee5-glassfish\.metadata\.plugins\org.eclipse.jst.server.generic.core\serverdef\sunappsrv-ant.xml

deploy.j2ee.web:

[echo] C:/.eclipse/javaee5-glassfish/.metadata/.plugins/org.eclipse.wst.server.core/tmp0/EgzaminyWeb

[jar] Building jar: C:\.eclipse\javaee5-glassfish\.metadata\.plugins\org.eclipse.jst.server.generic.core\serverdef\EgzaminyWeb.war

tools:

deploy:

[exec] Command deploy executed successfully.

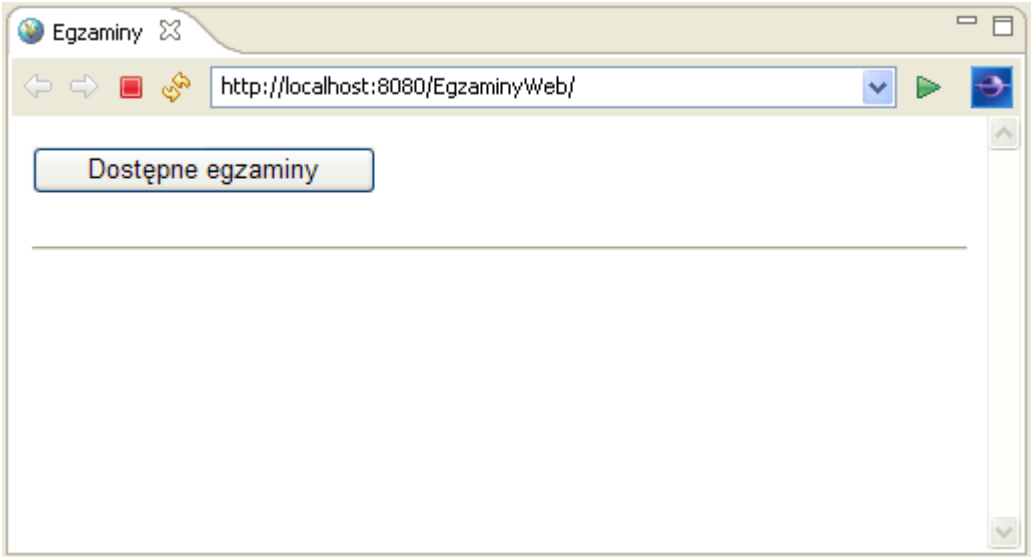
deploy-url-message:

[echo] Application Deployed at: http://127.0.0.1:8080/EgzaminyWeb

BUILD SUCCESSFUL

Total time: 5 seconds

Jeśli wszystko przejdzie pomyślnie, Eclipse otworzy okno przeglądarki z pierwszą stroną naszej aplikacji.



Wybierając przycisk **Dostępne egzaminy** upewniamy się, że aplikacja działa (wywoła ona komponent EJB, który zwróci listę dostępnych egzaminów).