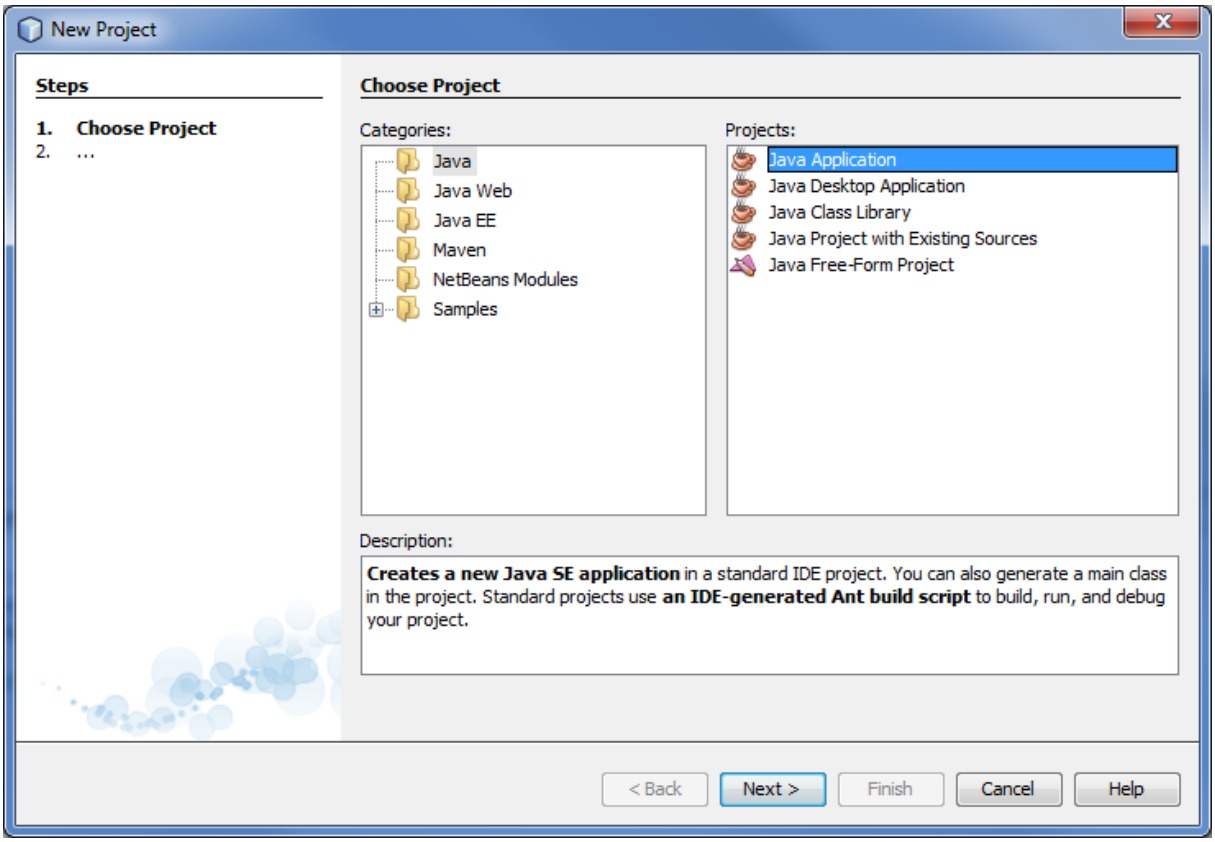
Java Persistence API (JPA)

Do wykonania ćwiczeń potrzebne jest zintegrowane środowisko programistyczne NetBeans IDE 7.0 wraz z serwerem GlassFish v3 (do pobrania z http://www.netbeans.org/) oraz środowisko JDK 6 (wymagane do instalacji NetBeans, do pobrania z http://java.sun.com/).

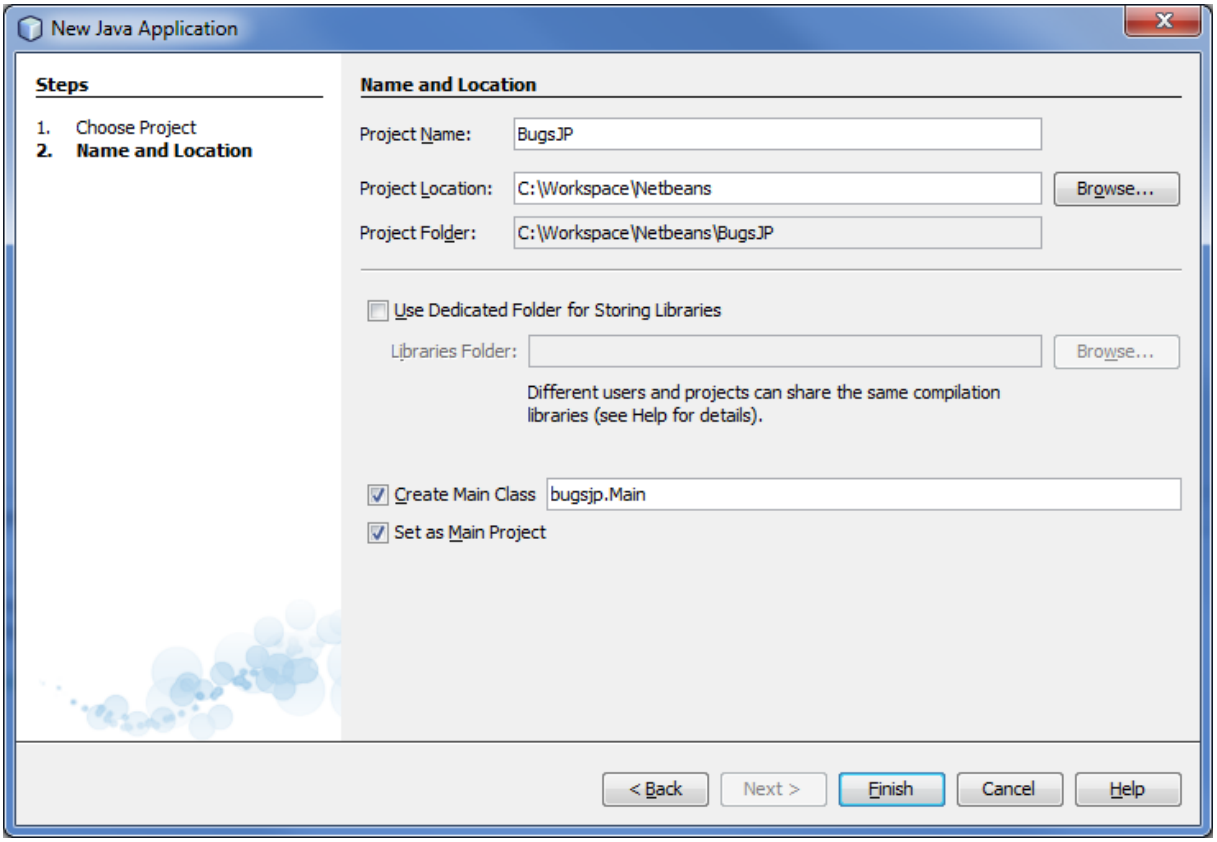
1. Celem pierwszego ćwiczenia jest przygotowanie prostej aplikacji konsolowej Java SE realizującej odczyt i zapis danych z/do bazy danych poprzez Java Persistence API. W pierwszym kroku ćwiczenia utworzysz nowy projekt.

   a) Uruchom środowisko NetBeans IDE.

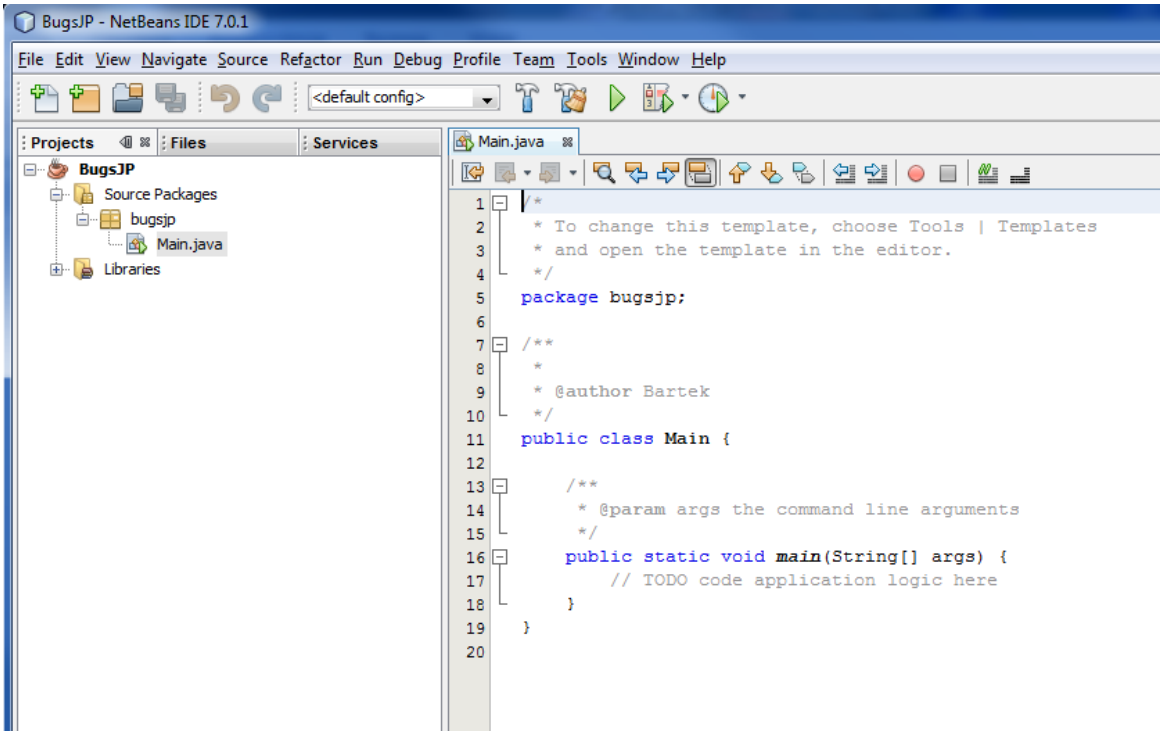
   b) Z menu głównego wybierz File → New Project. Wybierz kategorię Java i typ projektu Java Application. Kliknij przycisk Next >.



c) Podaj nazwę projektu **BugsJP**. W polu Project Location powinien być wskazany katalog, w którym masz prawo zapisu. Pola wyboru Set as Main Project i Create Main Class powinny być zaznaczone. W polu przy opcji Create Main Class popraw wpis na `bugsjp.Main`. Kliknij przycisk Finish.

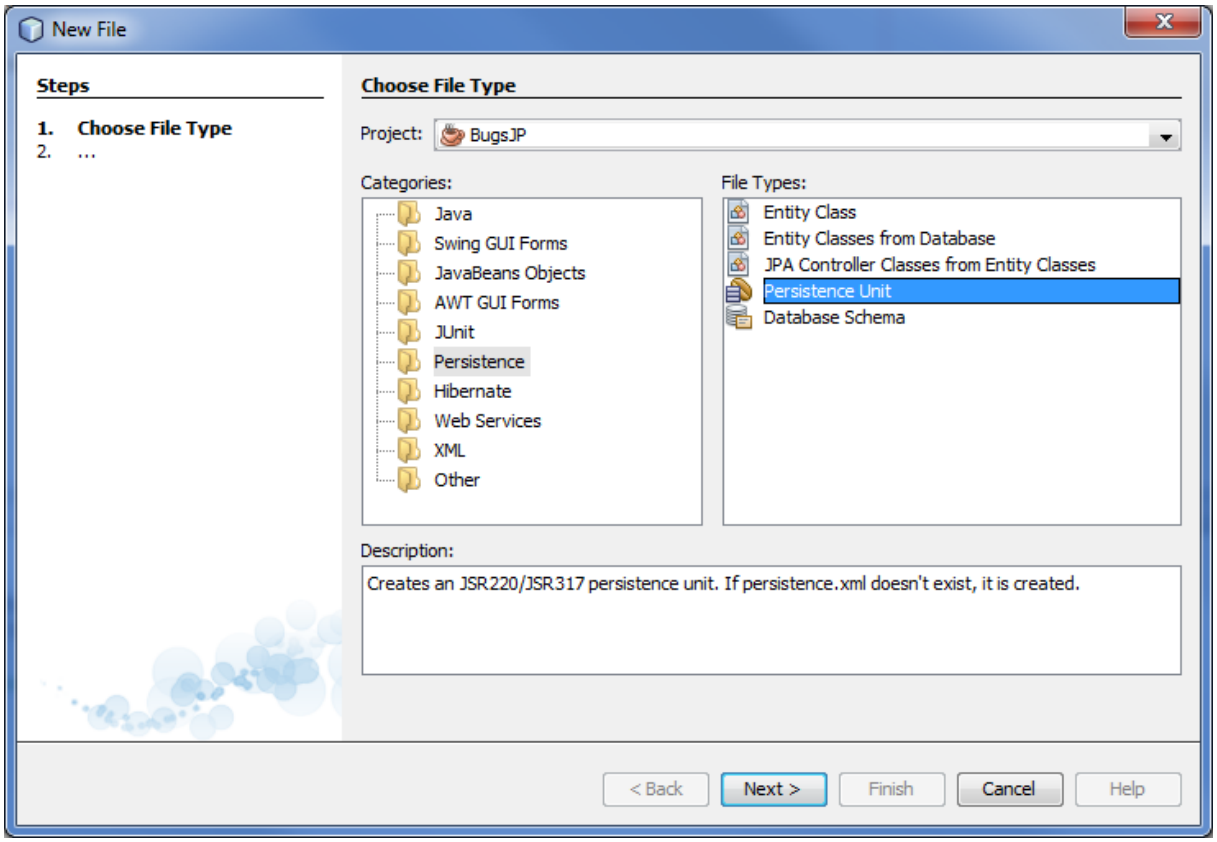


Efektem działania kreatora powinien być projekt zawierający klasę `Main` z metodą `main()`.



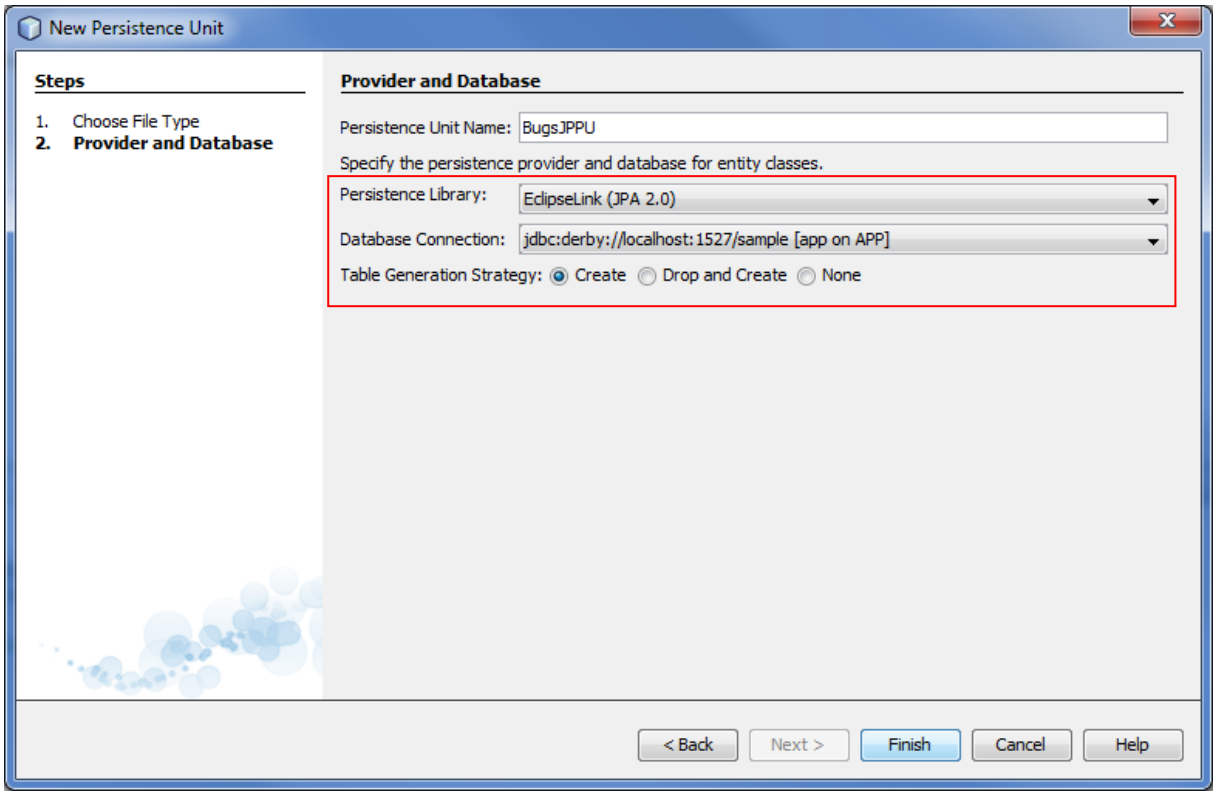
Teraz utworzysz jednostkę trwałości, w ramach której obiekty aplikacji będą zachowywane w bazie danych

d) Kliknij lewym przyciskiem myszy na ikonie projektu w drzewie projektów zaznaczając go, a potem z menu głównego wybierz File → New File. Następnie z kategorii Persistence wybierz typ pliku Persistence Unit. Kliknij przycisk Next >.

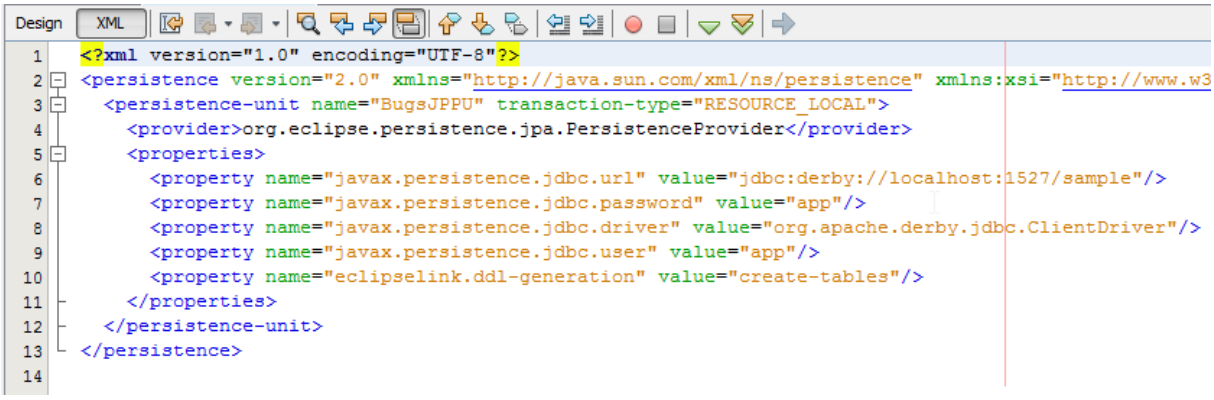


Alternatywnie, możesz kliknąć prawym klawiszem myszy na nazwę projektu i z menu kontekstowego wybrać pozycję New → Persistence Unit....

e) W kolejnym oknie pozostaw niezmienioną zaproponowaną przez kreator nazwę jednostki trwałości i domyślną bibliotekę do obsługi trwałości (Persistence Library): EclipseLink. Jako połączenia JDBC (Database Connection) wybierz z listy połączenie z bazą danych sample na lokalnym serwerze Derby. Jako strategię tworzenia tabel w bazie danych (Table Generation Strategy) pozostaw Create, czyli tworzenie w momencie instalacji aplikacji jeśli nie istnieją. Kliknij przycisk Finish.

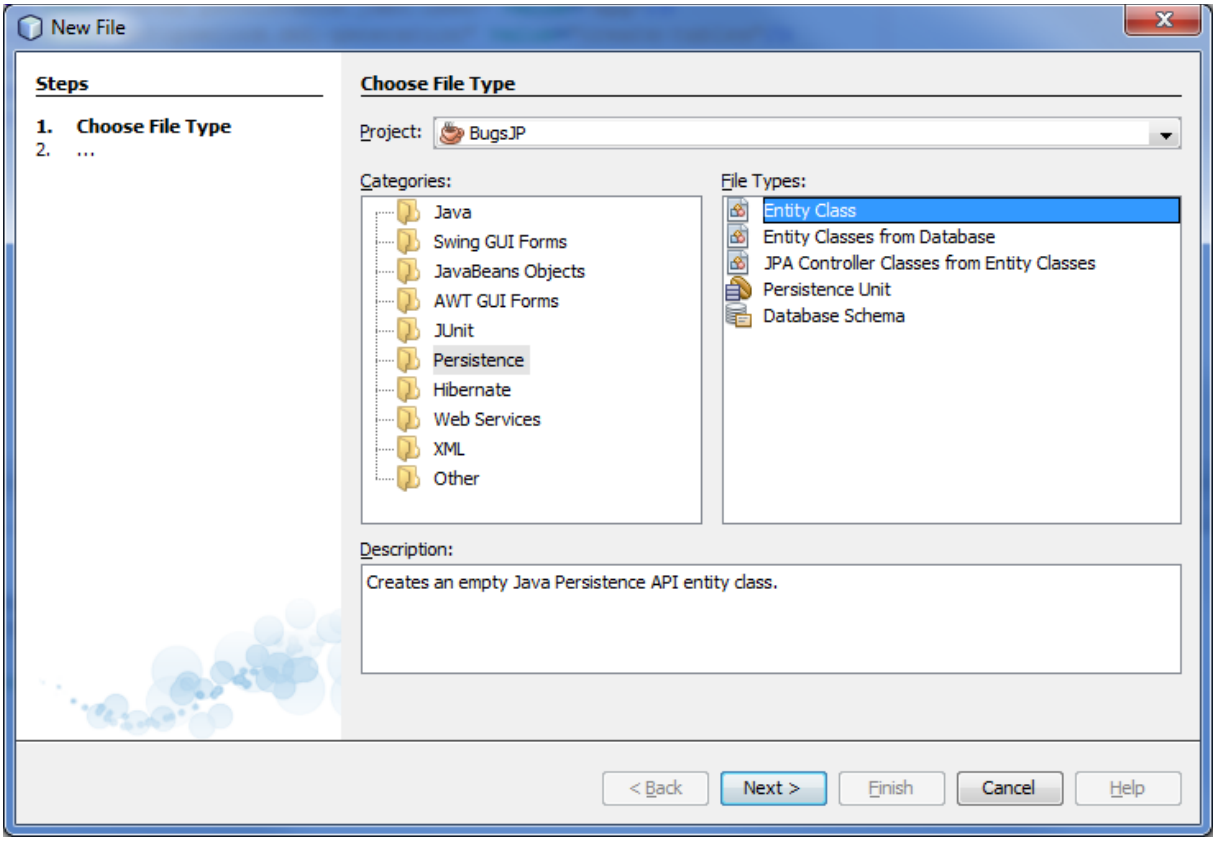


Obejrzyj w trybie XML zawartość pliku `persistence.xml`, wygenerowanego przez kreator. Zwróć uwagę na elementy `<provider>` i `<property>` zawierające wartości odpowiadające opcjom wybranym w oknie kreatora. Sprawdź, czy zarówno właściwość `javax.persistence.jdbc.user` i `javax.persistence.jdbc.password` mają wartość `app` i jeśli nie, to popraw zawartość pliku.

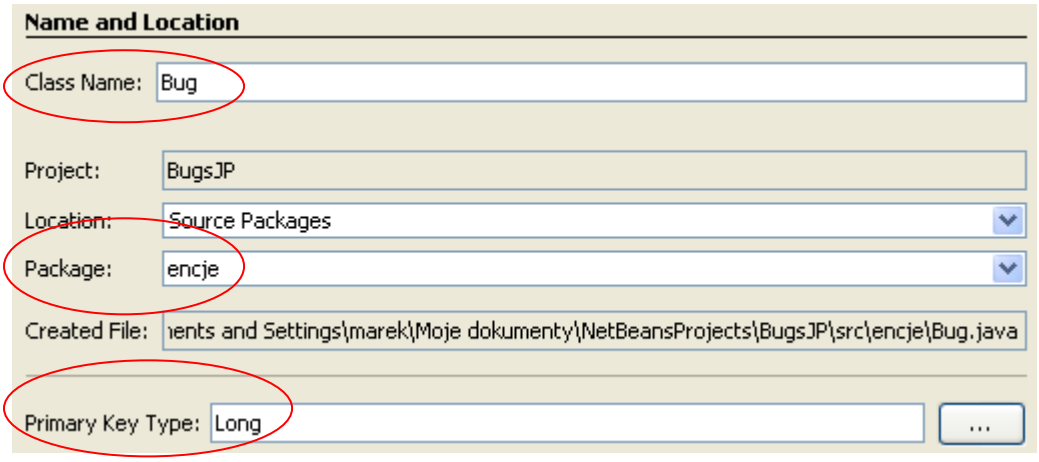


Teraz utworzysz klasę encji `Bug`.

f) Kliknij lewym przyciskiem myszy na ikonie projektu w drzewie projektów zaznaczając go, a potem z menu głównego wybierz File → New File. Następnie z kategorii Persistence wybierz typ pliku Entity Class. Kliknij przycisk Next >.



g) Jako nazwę klasy podaj `Bug`, a jako nazwę pakietu `encje`. Pozostaw `Long` jako typ klucza głównego (Primary Key Type). Kliknij przycisk Finish.



h) Ponownie obejrzyj zawartość pliku `persistence.xml` w trybie XML. Zwróć uwagę na element `<class>` dodany przez kreator tworzenia klasy encji. Atrybut ten oznacza, że utworzona klasa encji została wskazana jako zarządzana klasa trwała dla jednostki trwałości BugsJPPU.

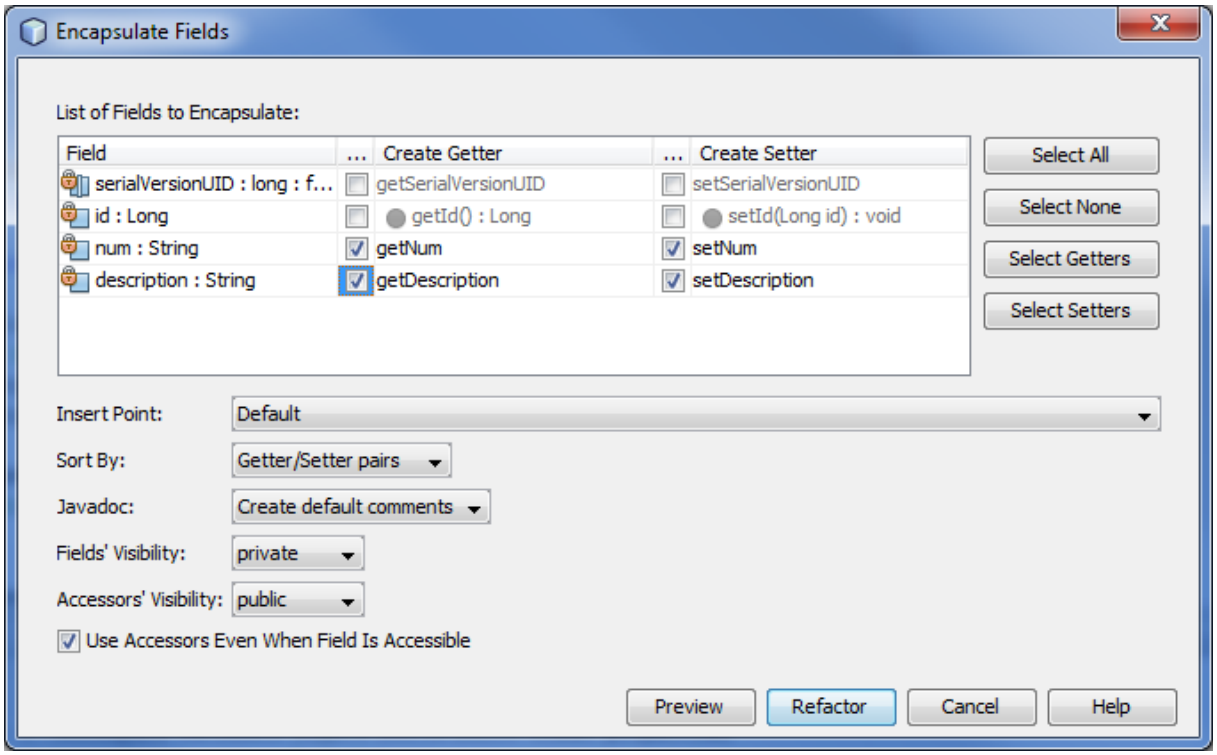
i) Przejdź do edycji utworzonego pliku `Bug.java`. Pod adnotacją `@Entity` dodaj wiersz z adnotacją `@Table(name="BUGS")`, aby obiekty klasy były składowane w

tabeli o nazwie `BUGS` (domyślnie nazwa tabeli byłaby taka jak nazwa klasy – w liczbie pojedynczej). Jeśli wprowadzona adnotacja zostanie podkreślona jako błąd, będąc kursorem w wierszu z adnotacją naciśnij kombinację klawiszy Alt+Enter i wybierz zaproponowaną opcję Add import for javax.persistence.Table.

j) Dodaj w klasie `Bug` (poniżej pola `id`) dwa prywatne pola: `num` typu `String` i `description` typu `String`.

```
@Entity
@Table(name="BUGS")
public class Bug implements Serializable {
...
    private String num;
    private String description;
...
```

k) W oknie edycji klasy `Bug` prawym klawiszem myszy wywołaj menu kontekstowe i wybierz opcję Refactor → Encapsulate fields w celu utworzenia w klasie metod setter/getter dla pól (tzw. akcesorów). Upewnij się, że dla atrybutu `id` metody dostępu zostały już wcześniej wygenerowane pola. Zaznacz do generacji metody setter/getter dla atrybutów `num` i `description`. Kliknij przycisk Refactor.



Obejrzyj w kodzie klasy wygenerowane metody.

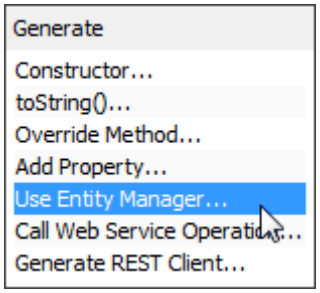
l) Zdefiniuj w klasie encji `Bug` nazwane zapytanie do wyszukiwania błędów (w formie kolekcji instancji encji) zawierających w opisie (atrybut `description`) podane słowo kluczowe (reprezentowane w treści zapytania jako zmienna wiązana `:keyword`) poprzez umieszczenie bezpośrednio po wierszu z adnotacją `@Table` wiersza z poniższą adnotacją `@NamedQuery` uzupełniając treść zapytania JPQL:

```
@NamedQuery(name = "findByKeyword", query = "... like :keyword")
```

Zaimportuj klasę `javax.persistence.NamedQuery`.

Dodasz teraz w klasie `Main` kod zapisujący obiekty `Bug` do bazy danych i odczytujący obiekty `Bug` z bazy danych.

m) Przejdź do edycji pliku `Main.java`. W oknie edycji klasy `Main` prawym klawiszem myszy wywołaj menu kontekstowe i wybierz opcję Insert Code. Następnie z okienka, które zostanie wyświetlone, wybierz pozycję Use Entity Manager.



Wykonana operacja spowoduje dodanie w klasie `Main` kodu tworzącego obiekt `EntityManagerFactory` i metodę `persist()`, ilustrującą sposób tworzenia obiektu `EntityManager` i realizację transakcji. Zwróć uwagę na instrukcję rozpoczęcia, zatwierdzenia i wycofania transakcji JDBC za pośrednictwem zarządcy encji (kreator dodał te instrukcje, gdyż zarządca encji będzie użyty w komponencie innego rodzaju niż EJB).

n) Dodaj do kodu klasy `Main` dyrektywy import importujące klasy z pakietu `encje` i klasę biblioteczną `java.util.List`:

```
import encje.*;
import java.util.List;
```

o) Przenieś deklarację i inicjalizację obiektu `EntityManagerFactory` z metody `persist()` na poziom klasy, jednocześnie czyniąc go składową statyczną:

```
private static EntityManagerFactory emf =
               Persistence.createEntityManagerFactory("BugsJPPU");
```

p) Dodaj w klasie `Main` publiczną, statyczną metodę `addBug()`, tworzącą obiekt `Bug` na podstawie podanego numeru i opisu a następnie zapisującą go do bazy danych poprzez metodę `persist()`. Uczyń metodę `persist()` metodą statyczną aby mogła być wywołana z poziomu metody `addBug()`.

```
public static void addBug(String pNum, String pDesc) {
    Bug b = ...
    ...
    persist(b);
}
```

q) Dodaj w klasie `Main` metodę `findBugs()`, wyszukującą w bazie danych błędy, których opisy zawierają podane słowo kluczowe, poprzez nazwane zapytanie zdefiniowane wcześniej w klasie encji.

```
public static List<Bug> findBugs(String pKeyword) {
      EntityManager em = emf.createEntityManager();
      List<Bug> wyn = null;
      try {
        wyn = ...
      } catch (Exception e) {
          e.printStackTrace();
      } finally {
          em.close();
      }
      return wyn;
}
```

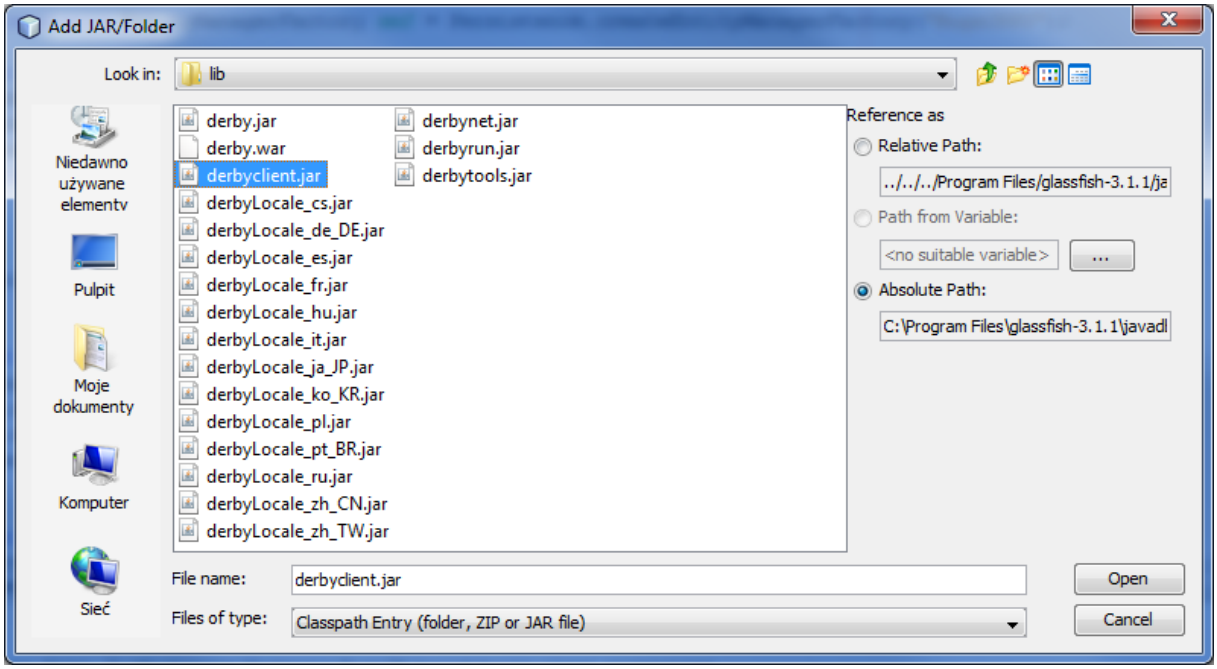
r) Dodaj w klasie `Main` statyczną metodę `bulkDeleteBugs()`, usuwającą wszystkie obiekty `Bug` z bazy danych za pomocą masowej operacji DELETE.

```
public static void bulkDeleteBugs() {
      EntityManager em = emf.createEntityManager();
      em.getTransaction().begin();
      try {
          em.createQuery("DELETE FROM Bug").executeUpdate();
          em.getTransaction().commit();
      } catch (Exception e) {
          e.printStackTrace();
          em.getTransaction().rollback();
      } finally {
          em.close();
      }
}
```

s) Dodaj w metodzie `main()` klasy `Main` kod zapisujący do bazy danych kilka obiektów `Bug` metodą `addBug()`, wyszukujący obiekty `Bug` zawierające w opisie określone słowo („wifi”) metodą `findBugs()` i wyświetlający wyniki wyszukiwania na konsoli. Na końcu kod ma usunąć wszystkie obiekty klasy `Bug`:

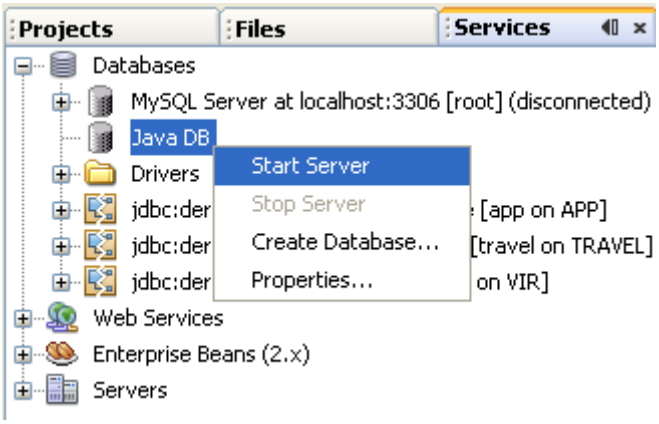
```
public static void main(String[] args) {
      addBug("b001", "Lost wifi connection");
      addBug("b002", "Turning off wifi does not work");
      addBug("b003", "Execution slow when on battery");
      for (Bug b : findBugs("%wifi%")) {
        System.out.println("Number: " + b.getNum() +
                           " Description: " + b.getDescription());
      }
      bulkDeleteBugs();
}
```

t) Dodaj do projektu sterownik JDBC do współpracy z wbudowaną bazą danych Derby (Java DB). W tym celu wywołaj dla węzła Libraries w drzewie projektu opcję Add JAR/Folder, a następnie dodaj do projektu archiwum JAR `derbyclient.jar` (znajdziesz je w jednym z podkatalogów katalogu, w którym zainstalowany został serwer aplikacji GlassFish).



u) Zapisz wszystkie zmiany (File→Save All lub ikona w pasku narzędzi).

v) Uruchom serwer bazy danych Java DB korzystając z panelu Services.



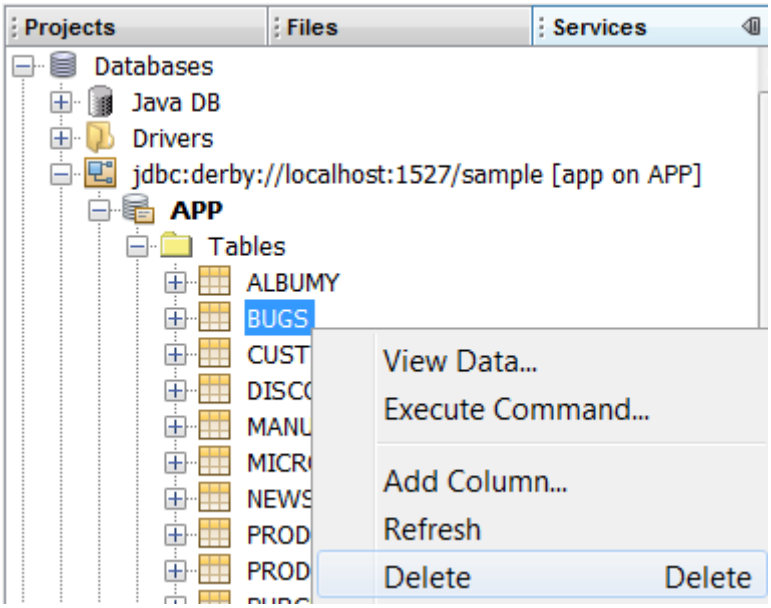
w) Uruchom aplikację wybierając opcję Run z menu kontekstowego dla projektu. Wyświetlone na konsoli dane odczytane z bazy danych mogą być poprzedzone komunikatami informacyjnymi i ostrzeżeniami zgłaszanymi przez EclipseLink. Spróbuj je zinterpretować.

x) Aby zmniejszyć ograniczyć liczbę komunikatów generowanych przez EclipseLink wstaw w pliku `persistence.xml` poniższy wiersz:

```
<property name="eclipselink.logging.level" value="SEVERE"/>
```

Następnie skompiluj projekt (Clean and Build) i ponownie uruchom aplikację.

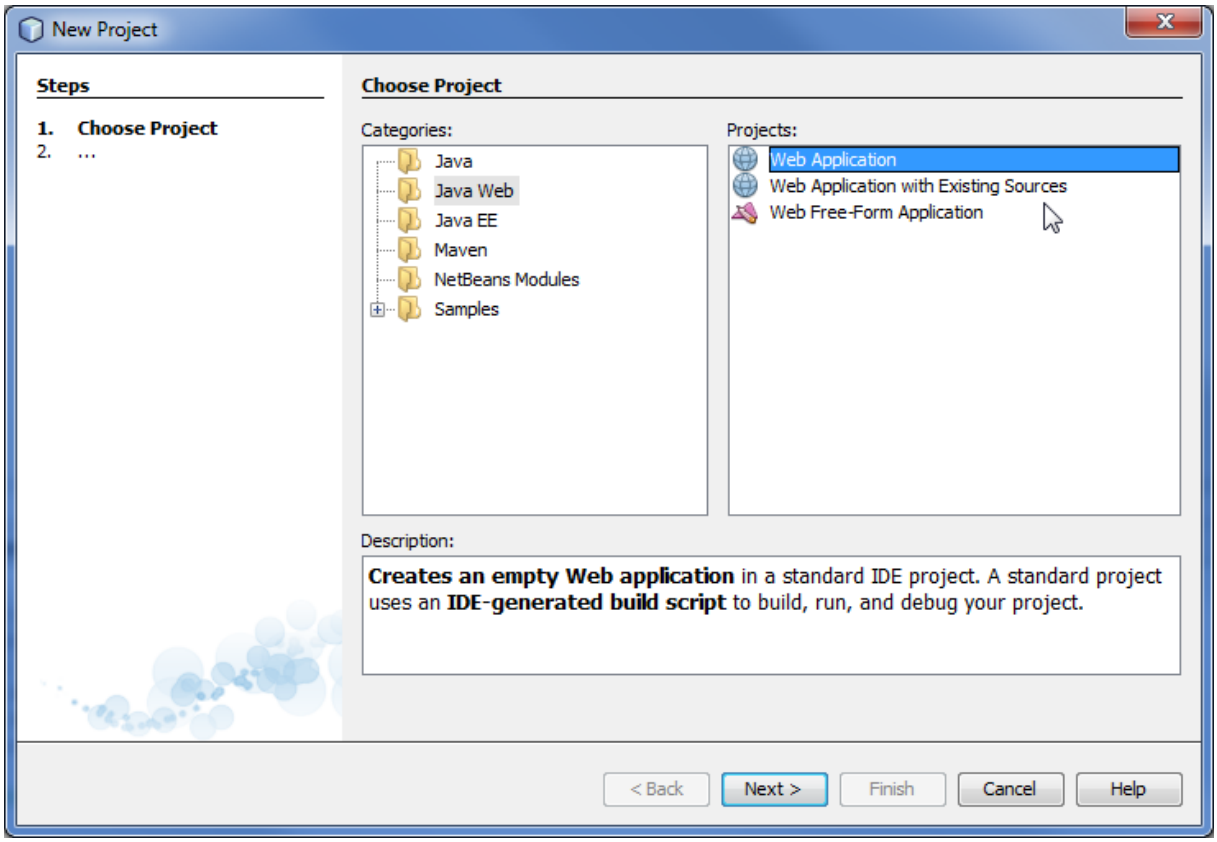
y) Na zakończenie ćwiczenia poprzez panel Services usuń z poziomu NetBeans tabelę `BUGS` z bazy danych.



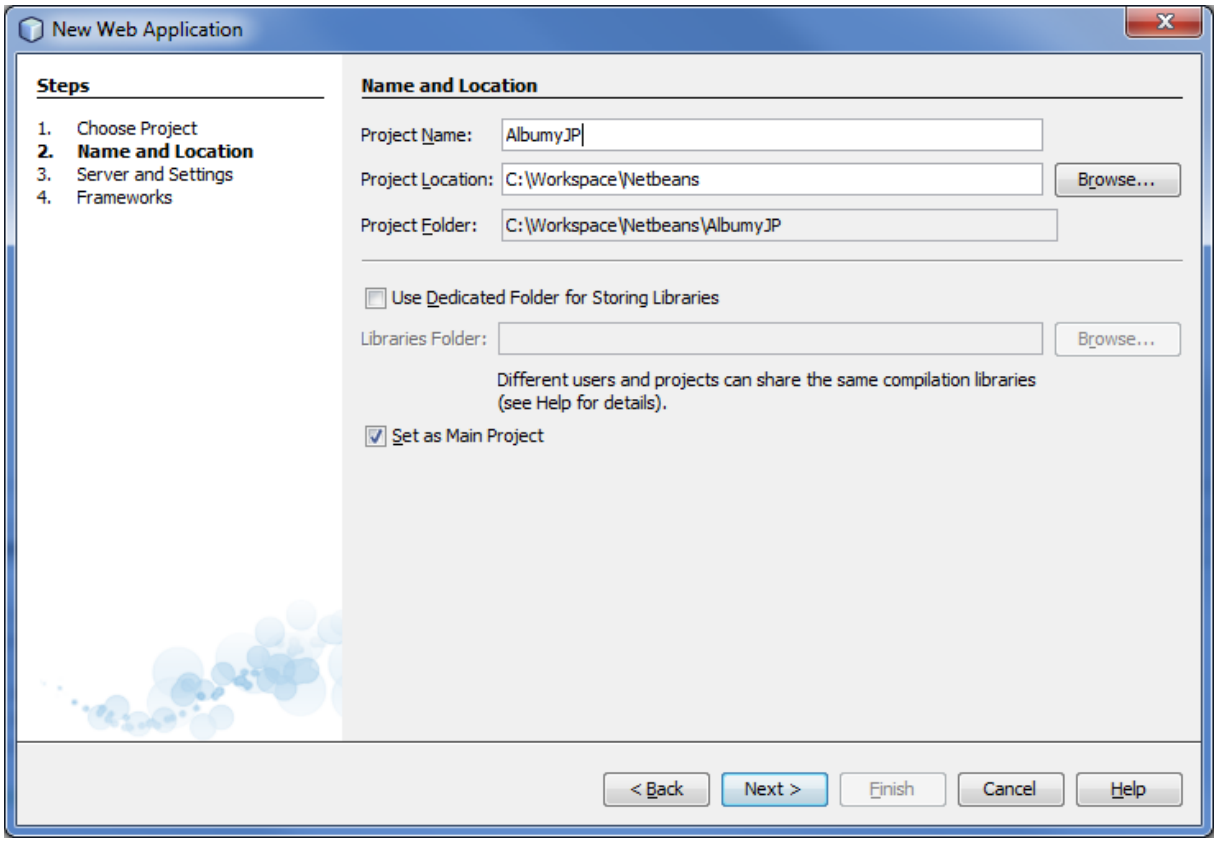
2. Celem drugiego ćwiczenia jest:
   - przygotowanie modelu obiektowego dla aplikacji, obejmującego dwie powiązane ze sobą encje i odwzorowanie go w schemat relacyjnej bazy danych;
   - wygenerowanie za pomocą kreatora aplikacji Java EE opartej o JSF, pracującej na utworzonym modelu i obsługującej trwałość poprzez Java Persistence API.

   W pierwszym kroku ćwiczenia utworzysz nowy projekt.

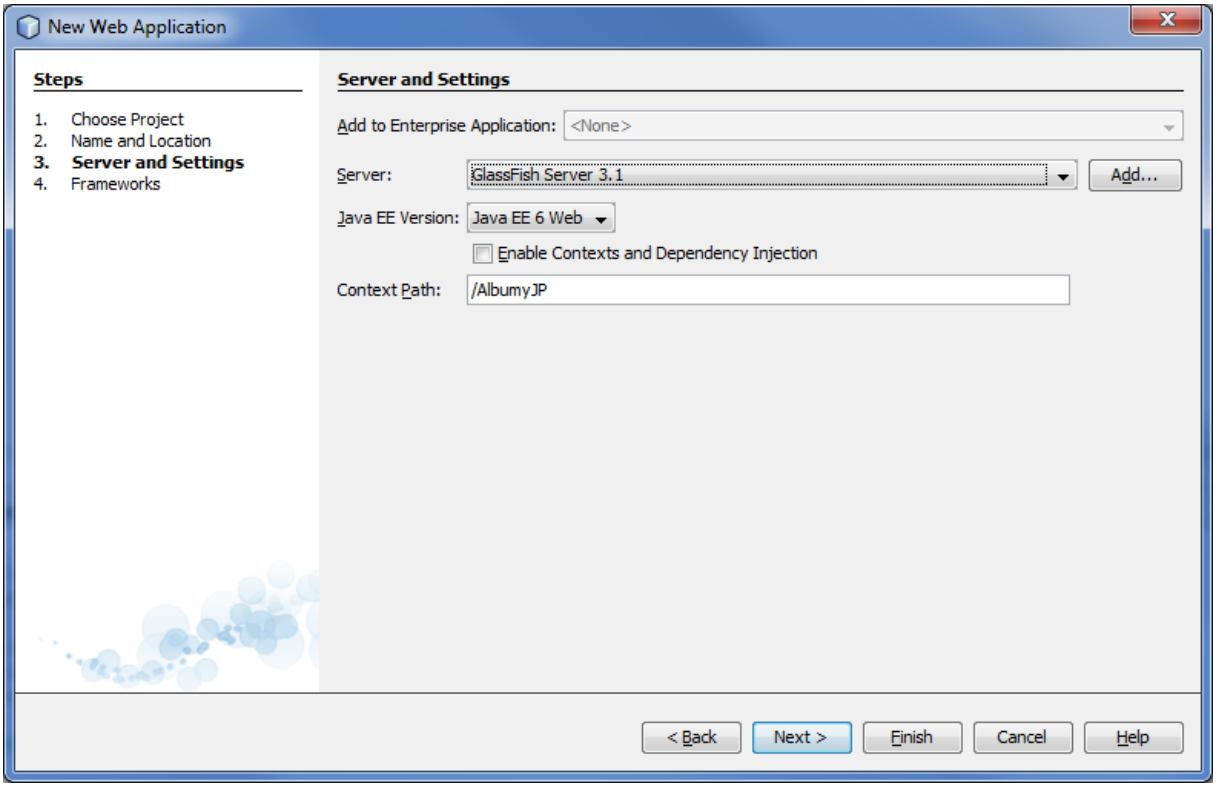
   a) Z menu głównego środowiska NetBeans IDE wybierz File → New Project. Zaznacz kategorię Java Web i typ projektu Web Application. Kliknij przycisk Next >.



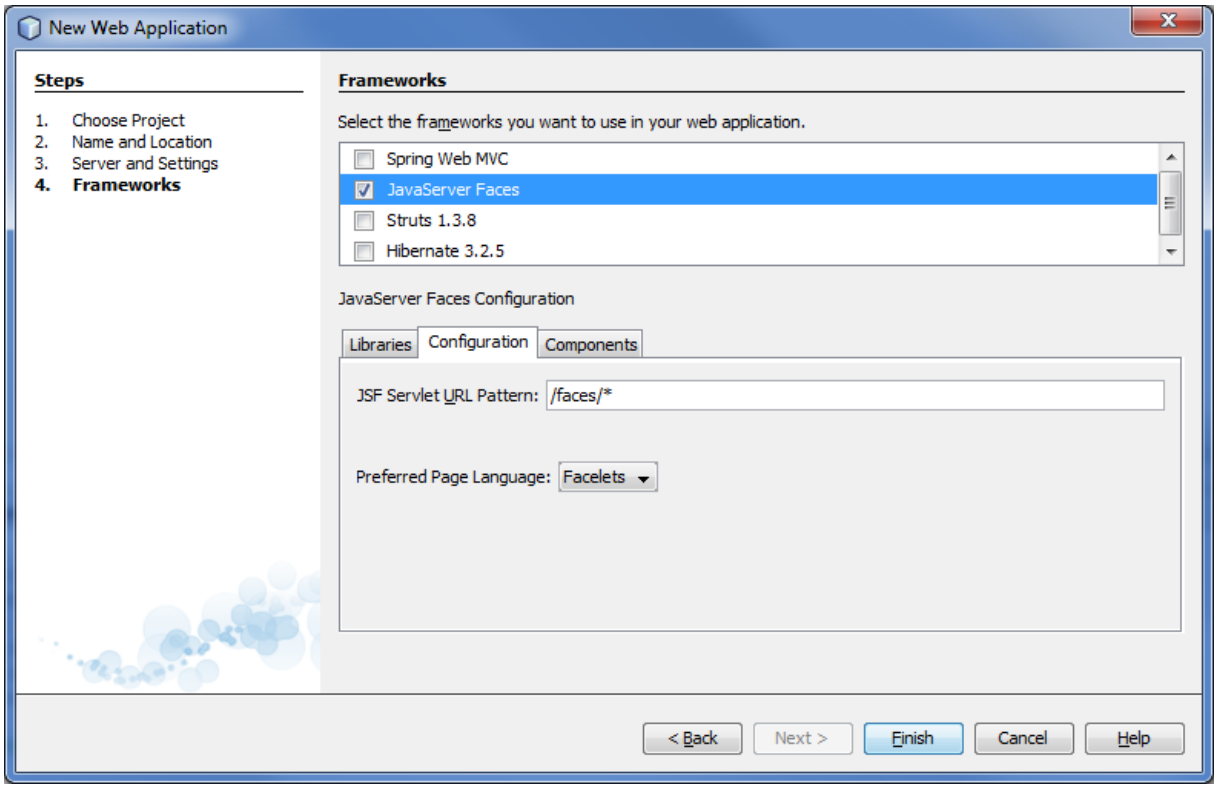
   b) Podaj nazwę projektu **AlbumyJP**. W polu Project Location powinien być wskazany katalog, w którym masz prawo zapisu. Pozostaw zaznaczoną opcję Set as Main Project.



c) W kolejnym kroku kreatora jako Server wybierz najnowszą wersję serwera GlassFish, a jako Java EE Version – Java EE 6 Web. Pozostaw bez zmian zaproponowany Context Path, czyli katalog wirtualny na serwerze WWW, który będzie prowadził do aplikacji. Kliknij przycisk Next >.

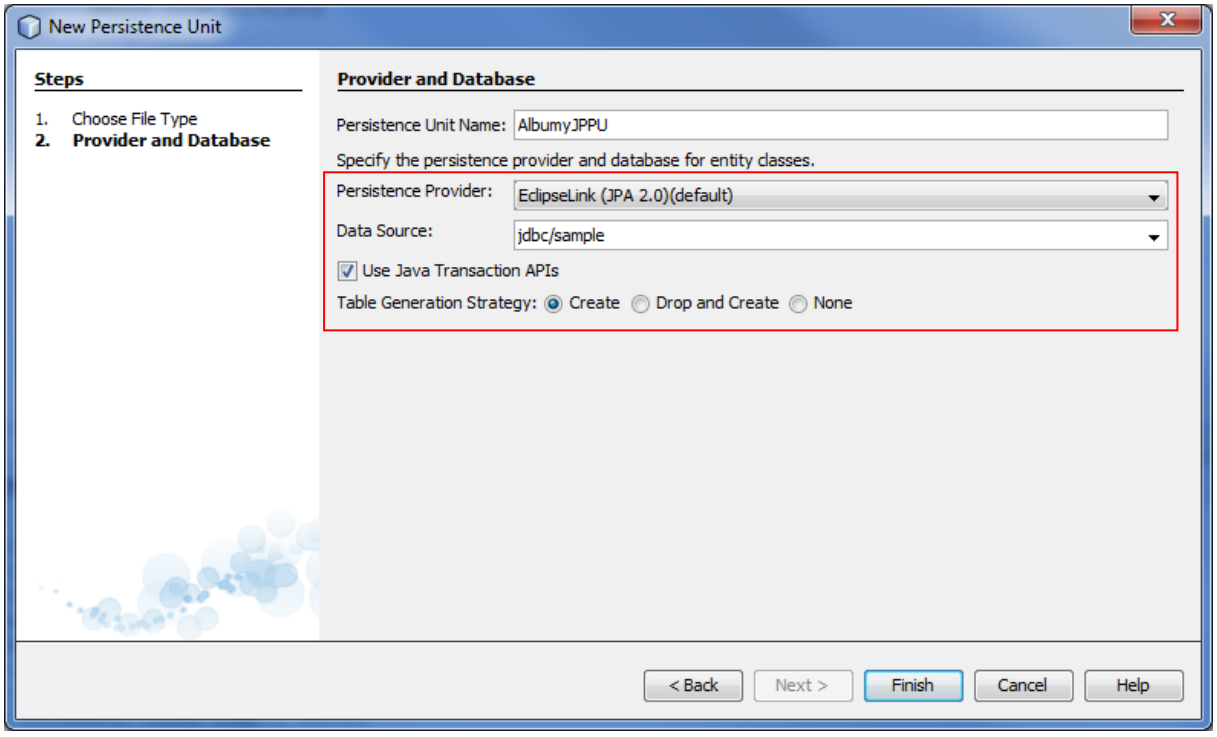


d) W ostatnim kroku zaznacz Java Server Faces w panelu Frameworks. Krok ten jest niezbędny, gdyż w ostatnim etapie ćwiczenia do prezentacji danych z bazy danych będzie wykorzystana aplikacja WWW oparta o JSF. Pozostaw zaproponowaną przez kreator konfigurację JSF i kliknij przycisk Finish.



Utworzysz teraz jednostkę trwałości, w ramach której obiekty aplikacji będą zachowywane w bazie danych

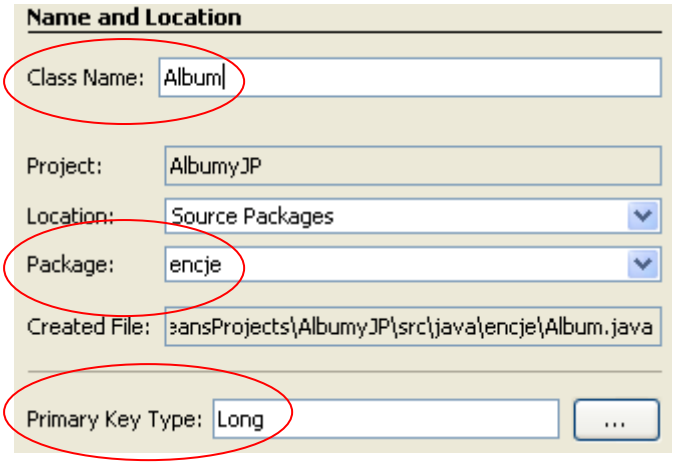
e) Kliknij prawym przyciskiem myszy na ikonie projektu w drzewie projektów i z menu kontekstowego wybierz New → Persistence Unit. W kolejnym oknie pozostaw niezmienioną zaproponowaną przez kreator nazwę jednostki trwałości i domyślną bibliotekę do obsługi trwałości (Persistence Library): EclipseLink. Jako źródło danych (Data Source) wybierz z listy połączenie z bazą danych jdbc/sample na lokalnym serwerze Derby. (W aplikacji Java EE połączenia z bazą danych będą realizowane poprzez źródło danych. Stąd inny zestaw pól w oknie kreatora niż w przypadku aplikacji Java SE tworzonej w ćwiczeniu 1.) Upewnij się, że pole wyboru Use Java Transaction API jest zaznaczone. Jako strategię tworzenia tabel w bazie danych (Table Generation Strategy) pozostaw Create, czyli tworzenie w momencie instalacji aplikacji jeśli nie istnieją. Kliknij przycisk Finish.



Obejrzyj zawartość wygenerowanego przez kreator pliku `persistence.xml` w trybie XML. Zwróć uwagę na elementy `<provider>` i `<jta-data-source>` zawierające wartości odpowiadające opcjom wybranym z list rozwijanych w oknie kreatora.

Utworzysz teraz klasy encji `Album` i `Wykonawca`. Z każdym albumem będzie związany jeden wykonawca. Każdy wykonawca będzie posiadał kolekcję albumów.

f) Kliknij prawym przyciskiem myszy na ikonie projektu w drzewie projektów i z menu kontekstowego wybierz New → Entity Class. Jako nazwę klasy podaj `Album`, jako nazwę pakietu `encje`. Pozostaw `Long` jako typ klucza głównego (Primary Key Type). Kliknij przycisk Finish.



g) Przejdź do edycji utworzonego pliku `Album.java`. Pod adnotacją `@Entity` dodaj wiersz z adnotacją `@Table(name="ALBUMY")`, aby obiekty klasy były składowane w tabeli o nazwie `ALBUMY` (domyślnie, bez adnotacji, nazwa tabeli byłaby taka jak nazwa klasy – w liczbie pojedynczej). Jeśli wprowadzona adnotacja zostanie podkreślona jako błąd, będąc kursorem w wierszu z adnotacją naciśnij kombinację klawiszy Alt+Enter i wybierz zaproponowaną opcję Add import for javax.persistence.Table.

h) Dodaj w klasie `Album` (poniżej pola `id`) trzy prywatne pola: `tytul` typu `String`, `wykonawca` typu `Wykonawca` i `rok` typu `int`. Zignoruj fakt, że pole `wykonawca` zostało oznaczone jako błąd. Błąd ten wynika z faktu użycia jako typu pola klasy `Wykonawca`, która jeszcze nie została utworzona.

```
@Entity
@Table(name="ALBUMY")
public class Album implements Serializable {
...
    private String tytul;
    private Wykonawca wykonawca;
    private int rok;
...
```

i) W celu utworzenia drugiej klasy, ponownie kliknij prawym przyciskiem myszy na ikonie projektu w drzewie projektów i z menu kontekstowego wybierz New → Entity Class. Jako nazwę klasy podaj `Wykonawca`, a jako nazwę pakietu `encje`. Pozostaw `Long` jako typ klucza głównego (Primary Key Type). Kliknij przycisk Finish.

j) Przejdź do edycji utworzonego pliku `Wykonawca.java`. Pod adnotacją `@Entity` dodaj wiersz z adnotacją `@Table(name="WYKONAWCY")`. Zaimportuj klasę `javax.persistence.Table` podobnie jak wcześniej dla klasy `Album`.

k) Dodaj w klasie `Wykonawca` (poniżej pola `id`) dwa prywatne pola: `nazwa` typu `String` i `albumy` typu `Collection<Album>` (nie zapomnij o zaimportowaniu `java.util.Collection`).

```
@Entity
@Table(name="WYKONAWCY")
public class Wykonawca implements Serializable {
...
    private String nazwa;
    private Collection<Album> albumy;
...
```

l) W oknie edycji klasy `Wykonawca` prawym klawiszem myszy wywołaj menu kontekstowe i wybierz opcję Refactor→Encapsulate fields w celu utworzenia w klasie metod set/get dla pól. Upewnij się, że pola wyboru metod setter/getter dla atrybutów `nazwa` i `albumy` są zaznaczone. Kliknij przycisk Refactor.

m) Wróć do edycji klasy `Album` i prawym klawiszem myszy wywołaj menu kontekstowe. Wybierz z menu kontekstowego opcję Refactor→Encapsulate fields w celu utworzenia w klasie metod setter/getter dla pól. Upewnij się, że pola wyboru metod setter/getter dla atrybutów `tytul`, `wykonawca` i `rok` są zaznaczone. Kliknij przycisk Refactor.

Obejrzyj w kodzie wygenerowane metody.

Zdefiniujesz teraz związek 1:N między klasami encji `Wykonawca` i `Album`

n) Oznacz adnotacją `@OneToMany(mappedBy="wykonawca")` pole `albumy` w klasie `Wykonawca`. Zaimportuj klasę adnotacji jak wcześniej dla adnotacji `@Table`. Uwaga: Adnotacjami JPA można opisywać pola lub właściwości (tj. metody `getter`), ale należy konsekwentnie trzymać się jednego rozwiązania.

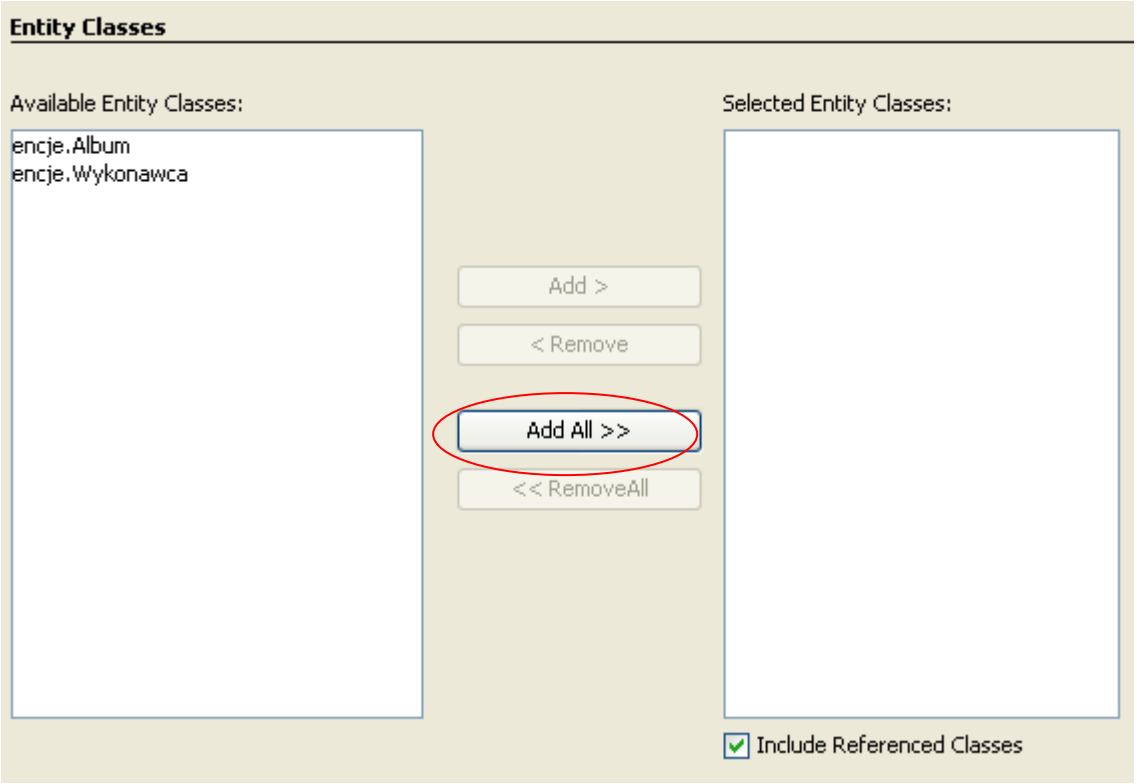
```
    @OneToMany(mappedBy="wykonawca")
    private Collection<Album> albumy;
```

o) Oznacz w klasie `Album` pole `wykonawca` adnotacją `@ManyToOne`. Zaimportuj klasę adnotacji.

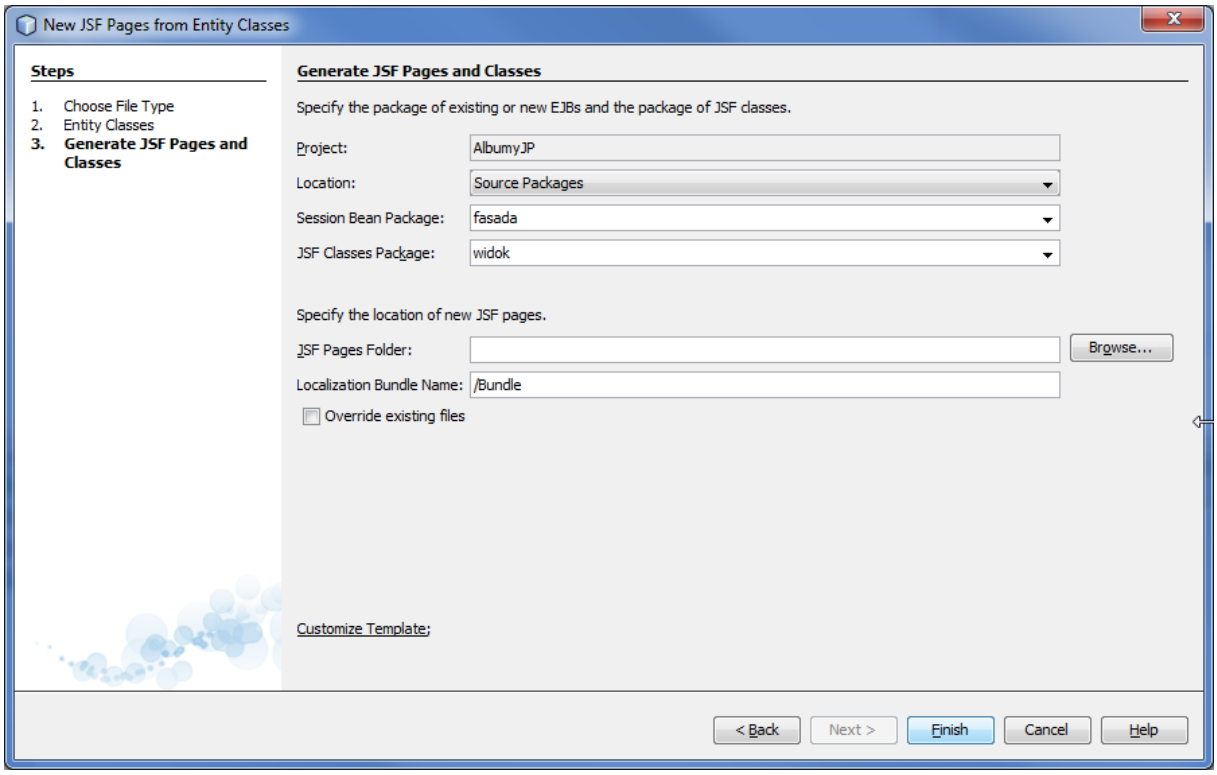
```
    @ManyToOne
    private Wykonawca wykonawca;
```

W ostatniej części ćwiczenia wygenerujesz za pomocą kreatora aplikację JSF do przeglądania i edycji zawartości bazy danych.

p) Kliknij prawym przyciskiem myszy na ikonie projektu w drzewie projektów i z menu kontekstowego wybierz New → Other... → JavaServer Faces → JSF Pages from Entity Classes. Kliknij przycisk Next >. Następnie wybierz wszystkie encje jako źródło dla aplikacji JSF klikając przycisk Add All.



q) Kliknij przycisk Next >. W kolejnym oknie zmień nazwy pakietów dla generowanych klas: `fasada` dla JPA Session Bean Package i `widok` dla JSF Classes Package. Pozostałe ustawienia pozostaw domyślne i kliknij przycisk Finish.



Wynikiem działania kreatora oprócz zestawów stron JSF do realizacji podstawowych operacji na encjach (po 4 strony dla każdej encji) są komponenty EJB pełniące funkcję fasady dla encji, komponenty zarządzane JSF pośredniczące w dostępie do fasady z poziomu stron JSF oraz towarzyszące im klasy pomocnicze. Rozwiń wszystkie gałęzie drzewa projektu w oknie Projects, aby obejrzeć wygenerowane pliki. Szczególną uwagę zwróć na:

- Operacje na encjach udostępniane przez komponenty fasady
- Sposób wykorzystania komponentów fasadowych z poziomu komponentów zarządzanych JSF
- Zasięg komponentów zarządzanych – Dlaczego nie mniejszy?
- Odwołania do komponentów zarządzanych z poziomu stron JSF
- Brak jawnego zatwierdzania transakcji po operacjach dokonywanych na encjach – W jaki sposób realizowane są transakcje i dlaczego inaczej niż w aplikacji z ćwiczenia 1?

r) Zapisz wszystkie zmiany (File→Save All lub ikona w pasku narzędzi).

s) Uruchom aplikację i przetestuj jej możliwości. W tym celu zacznij od dodania dwóch wykonawców i dla każdego z nich dwóch albumów. Następnie dokonaj edycji któregoś z albumów, a inny usuń.

3. Celem ostatniego ćwiczenia jest ilustracja problemu zarządzania współbieżnością i optymistycznego podejścia do jego rozwiązania w technologii JPA.

   a) Sprawdź domyślne zarządzanie współbieżnością na przykładzie aplikacji z ćwiczenia 2. W tym celu:

      - Uruchom aplikację.
      - Otwórz drugą (inną) przeglądarkę internetową i wprowadź w niej adres strony startowej aplikacji.
      - W obu przeglądarkach przejdź do edycji tego samego albumu.
      - W każdej przeglądarek zmień rok wydania albumu na inny (różne wartości w obu przeglądarkach).
      - Zachowaj zmiany w obu przeglądarkach. Co się stało?

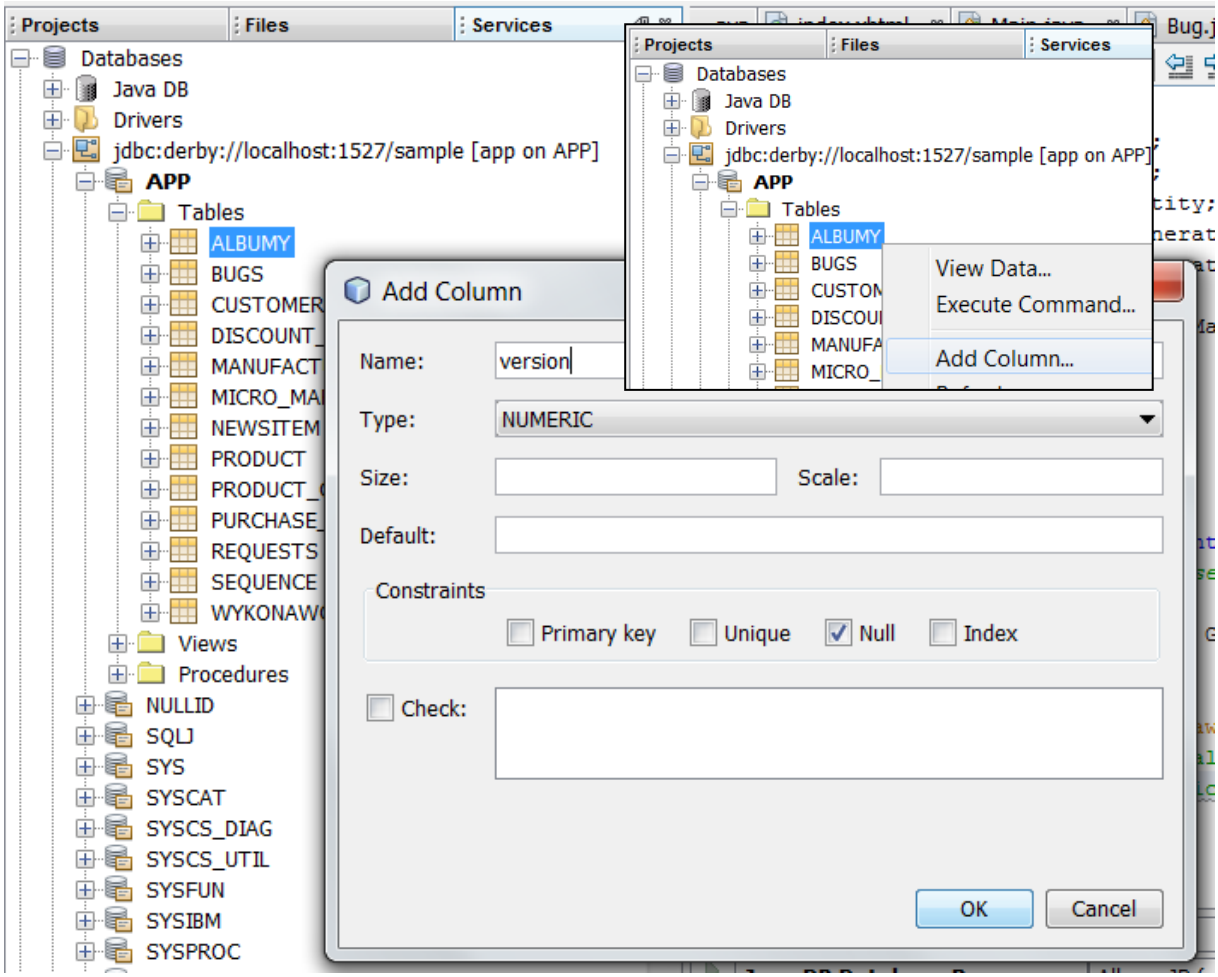
      Komentarz: Domyślnie JPA nie stosuje żadnych mechanizmów zarządzania współbieżnością. Zaobserwowana anomalia to „lost update" – jedna transakcja nadpisała efekt działania drugiej nie będąc tego świadomą. Jest to w ogólności działanie niepożądane, gdyż np. w przypadku współbieżnej modyfikacji pensji pracownika jedna z podwyżek/obniżek nie zostałaby odzwierciedlona w bazie danych. Należy zwrócić uwagę, że w kontekście aplikacji internetowej w kwestii ochrony przed tą anomalią nie można polegać na system zarządzania bazą danych. Dlaczego? Dlatego, że w aplikacji internetowej odczyt danych w celu ich prezentacji użytkownikowi i ich modyfikacja w odpowiedzi na edycję przez użytkownika będą wykonane w dwóch odrębnych transakcjach, przez co np. założenie blokady na odczytanym wierszu (`SELECT … FOR UPDATE` na poziomie SQL) nie zda egzaminu, gdyż blokada będzie zdjęta wraz z końcem pierwszej transakcji. Wprawdzie technologia Java EE pozwala na realizację transakcji obejmującej interakcję z użytkownikiem w aplikacji internetowej, jest to generalnie niezalecane. W aplikacjach internetowych preferowane jest zamykanie transakcji w obrębie pojedynczego żądania HTTP i optymistyczne zarządzanie współbieżnością.

Dodasz teraz atrybut wersji do encji w celu włączenia mechanizmu optymistycznego zarządzania współbieżnością na poziomie JPA.

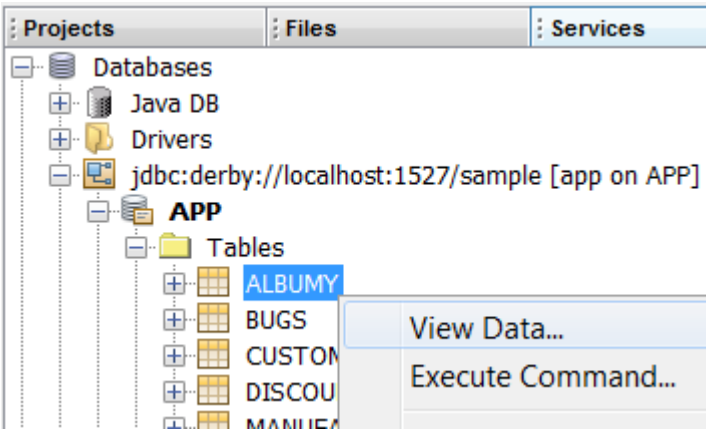
b) Dodaj w encjach `Album` i `Wykonawca` atrybut wersji (zaimportuj adnotację):

```
@Version private Long version;
```

c) Dodaj w tabelach `ALBUMY` i `WYKONAWCY` kolumnę `VERSION` typu `NUMERIC`



d) Zmodyfikuj dane w tabelach `ALBUMY` i `WYKONAWCY` ustawiając 1 jako numer wersji dla wszystkich wierszy.



e) Przebuduj projekt (Clean and Build) i uruchom aplikację

f) Powtórz eksperyment ze współbieżną modyfikacją roku wydania tego samego albumu w dwóch przeglądarkach. Czy tym razem wystąpiła anomalia utraconej modyfikacji?

Komentarz: JPA automatycznie zapewnia optymistyczne zarządzanie współbieżnością pod warunkiem dodania atrybutu wersji do encji. W przypadku gdy modyfikacja prowadziłaby do anomalii lost update, JPA do niej nie dopuści rzucając wyjątek. Sposób informowania o problemie w aplikacji z kreatora jest daleki od ideału. W rzeczywistości należałoby wyświetlić komunikat informujący o przyczynie odrzucenia modyfikacji wraz z prośbą o odświeżenie ekranu i ewentualne ponowienie modyfikacji.

g) Na zakończenie ćwiczenia poprzez panel Services usuń z poziomu NetBeans tabele `ALBUMY` i `WYKONAWCY` z bazy danych.