```java
package com.sda.hib.chapter3.one.to.one;

import com.sda.hib.HibernateUtils;
import com.sda.hib.chapter3.one.to.one.model.Husband;
import com.sda.hib.chapter3.one.to.one.model.Wife;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.query.Query;

import java.util.List;

public class HibernateChapter3OneToOne {

    public static void main(String... args) {


        HibernateChapter3OneToOne hibernateChapter3OneToOne = new
        HibernateChapter3OneToOne();
        hibernateChapter3OneToOne.deleteObjects(); //sprzątamy po poprzedniej
        instancji
        hibernateChapter3OneToOne.createAndSaveObjects(); //przygotowywujemy obiekty
        typu Husbant  i Wife wraz z relacją pomiędzy nimi.
        hibernateChapter3OneToOne.husbantToWifeRelation(); // bawimy się relacją,
        idziemy od Husbant do Wife
        hibernateChapter3OneToOne.wifeToHusbandRelation(); // bawimy się relacją,
        idziemy od Wife do Husbant

    }

    private void createAndSaveObjects() {
        try (SessionFactory factory = HibernateUtils.buildSessionFactory()) {
            Session session = factory.getCurrentSession();
            session.getTransaction().begin();

            Husband husband = new Husband("George");
            Wife wife = new Wife("Mary");
            husband.setWife(wife);

            session.save(husband);
            session.save(wife);

            session.getTransaction().commit();
            session.close();
        }
    }

    private void husbantToWifeRelation() {
        try (SessionFactory factory = HibernateUtils.buildSessionFactory()) {
            Session session = factory.getCurrentSession();
            session.getTransaction().begin();

            String hql = "Select h from Husband h ";


            Query<Husband> query = session.createQuery(hql);
            List<Husband> husbands = query.getResultList();

            Husband husband = husbands.get(0); //ogólnie to jest słabe, bo w
            normalnej aplikacji nie mamy gwarancji na to że coś będzie w liście pod
            tym indexem, tutaj w ten sposób wyciągamy bo aplikacja jest mała i wiemy
            co wrzuciliśmy do bazy

            // UWAGA UWAGA :
            System.out.println(husband.getWife().getName()); // oczywiście wyświetli
            się ale wife została już wcześniej pobrana (tj w 50 linii)

            // dlaczego ?
            // bo mamy tutaj relację @OneToOne z husbant do wife, defaultowo jest
            ona w typie EAGER (pobiera wife nie ważne czy ją potrzebujemy czy nie )
            // @OneToOne jest równoważne z  @OneToOne(fetch = FetchType.EAGER), //
            mozesz dla ćwiczenia ustawić w klasie Husband @ManyToOne(fetch =
            FetchType.LAZY) i zobaczyć co sięstanie

```

```
62              session.close();
63              // akurat w tym przykladzie jest tylko jedna para husbant-wife, ale
                gdybybyło ich więcej to poza sesją też można było by wyciągać obiekty
                wife z husband
64              // wszystko dzięki defaultowej wartośći FetchType.EAGER w adnotacji
                @OneToOne
65          }
66      }
67
68      private void wifeToHusbandRelation() {
69          try (SessionFactory factory = HibernateUtils.buildSessionFactory()) {
70              Session session = factory.getCurrentSession();
71              session.getTransaction().begin();
72
73              String hql = "Select w from Wife w";
74
75
76              Query<Wife> query = session.createQuery(hql);
77              List<Wife> wifes = query.getResultList();
78
79              Wife wife = wifes.get(0);//ogólnie to jest słabe, bo w normalnej
                aplikacji nie mamy gwarancji na to że coś będzie w liście pod tym
                indexem, tutaj w ten sposób wyciągamy bo aplikacja jest mała i wiemy co
                wrzuciliśmy do bazy
80              // wszystko dzieje się tak samo co metodzie wyżej, tylko idziemy od
                drugiej strony
81              System.out.println(wife.getHusband().getName());
82
83              session.close();
84
85          }
86      }
87
88      private void deleteObjects() {
89          try (SessionFactory factory = HibernateUtils.buildSessionFactory()) {
90              Session session = factory.getCurrentSession();
91              session.getTransaction().begin();
92
93              String hql = "Select h from Husband h";
94
95              Query<Husband> query = session.createQuery(hql);
96              List<Husband> husbands = query.getResultList();
97
98              husbands.forEach(husband -> session.delete(husband));
99
100             String hql2 = "Select w from Wife w";
101
102             Query<Wife> query2 = session.createQuery(hql2);
103             List<Wife> wifes = query2.getResultList();
104
105             wifes.forEach(wife -> session.delete(wife));
106
107             session.getTransaction().commit();
108             session.close();
109
110             //alternatywnie możecie w klasie Husband w adnotacji @OneToOne dopisać
                kaskadowość czyli @OneToOne(cascade = CascadeType.REMOVE)
111             // wtedy nie będziecie musili jawnie kasować obiekty typu Wife bo będą
                one leciały wraz z kasowaniem przypisanych do nich obiektów Husband
112             // to samo można zrobić w adnotacji @OneToMany (klasa Mom, chapter 2),
                wtedy kasując obiekt typu Mom, lecą wszystkie dzieci.
113             //oraz w @ManyToMany (klasa Student, Course, chapter 4)
114         }
115     }
116 }
117
118
```