

Sarrera

Laborategi honetan objectdb datu-basearen erabilera sakonduko dugu.

Helburuak

- Bi norabideko asoziazioen implementazioa ezagutu.
- Objektu anidatuen sorketa, eguneraketa eta ezabaketa ezagutu.
- SQL eta JPQL antzekoak direla ezagutu baina bigarrena OZ-ko ezaugarriekin,
- Datu-basea urruneko zerbitzari batean ipintzeko eta atzitzeko aukerak ikusi.

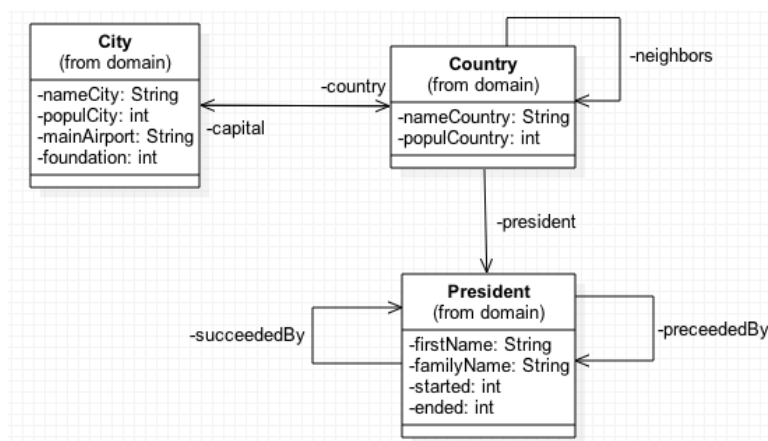
Pausoak

1.Lan egingo dugun proiektua deskargatu.

Lehendabizi <https://github.com/jononekin/lab4Objectdb2> helbidean dagoen **git** proiektua eclipsean kargatu. Proiektua **Maven** egitura dauka, eta erroan dagoen **pom.xml** fitxategian proiektuak behar dituen liburutegi guztiak definitzen dira. Gure kasuan objectdb-ko liburutegiak adierazten dira. Proiektua kargatzen denean, java inguruneak pom.xml fitxategian behar diren liburutegiak identifikatzen ditu, pom-ean definitutako errepositoriotik (<https://m2.objectdb.com>) **automatikoki** deskargatzen ditu (Maven Dependencies karpeta) eta proiektuaren classpath-era gehitzen ditu.

Proiektua konfiguratu ondoren, hurrengo egitura duela frogatu:

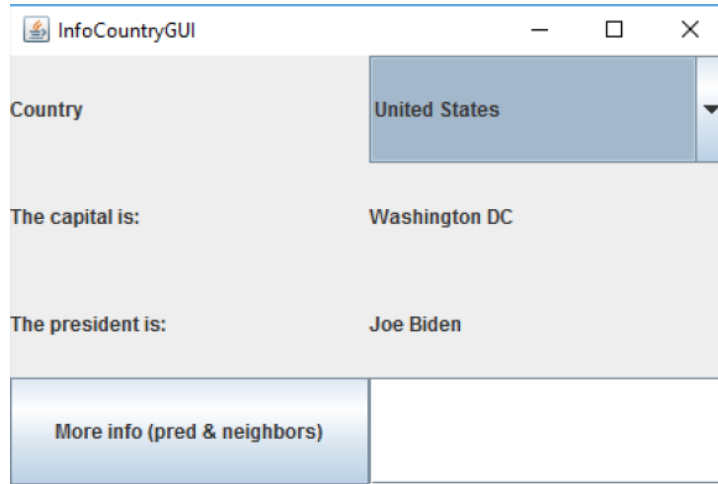
- **domain** Irudi 1-eko **America.odb** datu-basean agertzen diren domeinuzko klase eta erlazioen pakete bat da
- **gui** bukatu gabeko interfaze grafiko klase batzuen pakete bat da. Klase hauek lagungarriak izango dira datuak datu-basean txertatzeko.
- **bussinesLogic** negozio logikaren **FacadeInterface** eta **FacadeImpl** klaseen pakete bat da. Klase hauen metodoak DataAccess mailari deituko diote.
- **dataAccess** Klaseak objektuak datu basean gordetzeko eragiketekin.



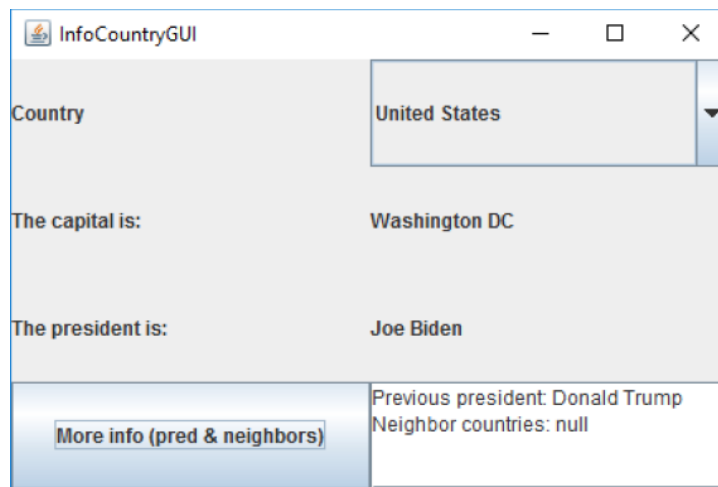
Irudia 1: America.db4o datubasearen Domeinuzko Klase Diagrama.

2. Objektu anidatuen kontsulta

InfoCountryGUI interfazea nazio desberdinen informazioa begiratzeko erabili daiteke. “United States” aukeratzen badugu bere hiriburua eta presidentea ikusiko dugu. Beste nazioentzako informazioa osatugabe dago,



“More info (pred & neighbors)” sakatzen badugu, aurreko presidentea eta bere ondorengo nazioen informazioa agertuko zaigu.



Nahiz eta ondorengo nazioen informazioa datu-basean gordeta egon (objectdb-ren EXPLORER tresnarekin frogatu daiteke), hauek ez dira agertzen interfazean. Baina, “Donald Trump” aurreko presidentea izan dela bai agertzen da. Portaera honen arrazoia, objektu baten objektu erlazionatuak noiz berreskuratzen diren politikarekin erlazionatuta dago.

ObjectDB-k Java Persistence API (JPA) API-a inplementatzen du. Espezifikazio honetan klaseen arteko erlazioen propietateak anotazioen bidez espezifikatu daitezke.

Gure adibidean, klaseen erlazioak `@OneToOne` eta `@OneToMany` anotazioen bitartez anotatzen dira. `@OneToOne` anotazioarekin Country klasearen `capital` atributuan, Country objektu bat City objektu batekin erlazionatuta dagoela definitzen da. Gauza bera gertatzen da `president` atributuarekin. `@OneToMany` anotazioarekin `neighbors` atributuan, Country objektu bat Country objektu multzo batekin erlazionatuta dagoela deskribatzen da.

```
@Entity
public class Country {
    @Id
    private String nameCountry;
    @OneToOne
    private City capital;
    private int populCountry;
    @OneToMany
    private Set<Country> neighbors=new HashSet<Country>();
    @OneToOne
    private President president;
```

```
@Entity
public class City {
    @Id
    private String nameCity;
    @OneToOne
    private Country country;
    private int populCity;
    private String mainAirport;
    private int foundation;
```

```
@Entity @IdClass (President.class)
public class President {
    @Id
    private String firstName;
    @Id
    private String familyName;
    private int started;
    private AccessMode accessPower;
    private int ended;
    @OneToOne
    private President preceededBy;
    @OneToOne
    private President succeededBy;
```

Ohar: `@IdClass(President.class)` anotazioarekin `firstName` eta `familyName` atributuek *President*-aren oinarritzko gako konposatua (primary key) osatzen dutela adierazten da.

Anotazioa ez da guztiz beharrezkoa gako konposatu bat definitzeko, baina beharrezkoa da, baldin eta klaseko objektu bat bere oinarritzko gakoaren bidez bilatu nahi bada:

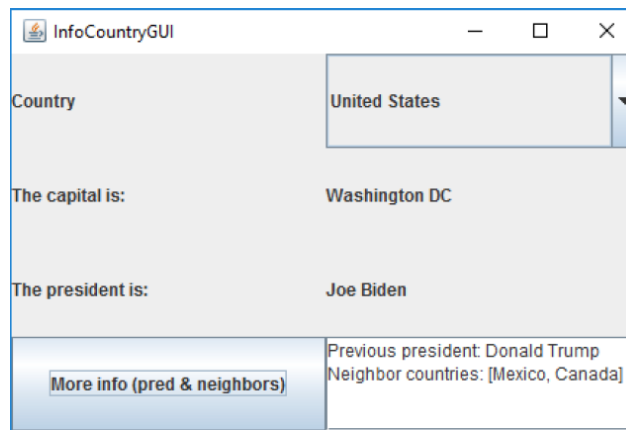
```
db.find(President.class, new President("Joe","Biden"))
```

Puntu honetan hurrengoak egin behar da:

- Aurreko anotazioak ipini, eta **InfoCountryGUI** klasea berriz exekutatu.
- Frogatu portaera ez dela aldatu: “United States” objektua ekartzen denean ez dira bere ondokoak ekartzen.
- Country klasea aldatu, Country objektu bat berreskuratzen denean bere ondokoak ere berreskuratzeko politika definituz (EAGER edo “irrika” estrategia erabiliz)

```
@OneToMany(fetch=FetchType.EAGER)
private Set<Country> neighbors=new HashSet<Country>();
```

- Exekutatu berriz aplikazioa eta frogatu, orain bai, “United States”-en ondokoak agertzen direla.



- Country klasea aldatu, Country klaseko objektu bat berreskuratzen denean, bere ondokoak objektuak ez berreskuratzeko behar diren momentu arte (LAZY edo “alfer” estrategia erabiliz).

```
@OneToMany(fetch=FetchType.LAZY)
private Set<Country> neighbors=new HashSet<Country>();
```

- Frogatu aplikazioa hasieran bezala exekutatzen dela. Hau gertatzen da @OneToMany defektuzko portaera LAZY delako. Bestalde, @OneToOne anotazioan, EAGER da defektuzko portaera, eta horregatik “Donald Trump” agertu da exekuzio guztietan.

Ohar: Bitxikeria gisa, nahiz eta @OneToOne anotazioan **LAZY** portaera ipintzen badugu, objectDB-k ematen du ez duela inplementatzen, derrigorrezkoa ez delako, hurrengo dokumentazioan agertzen den bezala:

<http://www.objectdb.com/api/java/jpa/OneToOne>

```
@OneToOne(fetch=FetchType.LAZY)  
private President president;
```

g) DataAccess dagoen getAllCountries() metodoan hurrengo kodea de-komentatu.

```
//      for (Country c: results) {  
//          System.out.println(c+" with neighbors: "+c.getNeighbors());  
//      }
```

eta **LAZY** estrategia erabili **neighbors**-ak berreskuratzeko (Country klasean)

```
@OneToMany(fetch=FetchType.LAZY)  
private Set<Country> neighbors=new HashSet<Country>();
```

InfoCountryGUI klasea berriz exekutatu, "United States" aukeratu eta "More info" botoia sakatu ondoren, "United States"-en ondokoak agertzen direla frogatu.

Galdera: Zergatik agertzen dira?

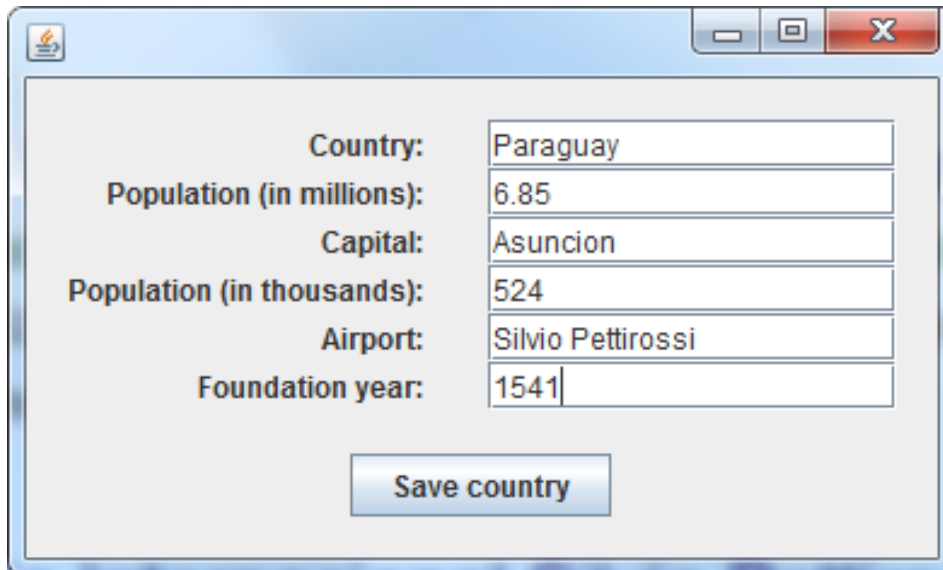
Erantzuna: **LAZY** estrategiarekin, ondoko nazioak berreskuratzen dira eskatzen diren momentuan, hau da, DataAccess klasearen getAllCountries() metodoan **c.getNeighbors()** deialdia egiten denean. Ondokoak berreskuratu direnez, getAllCountries() metodoak nazio lista itzultzen duenean, ondokoak ere eskuragarri daude **InfoCountryGUI** interfazeaz.

Bestalde, getAllCountries() metodoan **c.getNeighbors()** exekutatzen ez bada, **LAZY** estrategia aplikatuz, ondorengo nazioak ez dira berreskuratzen, eta datu-basea itxi ondoren, objektu horiek ezin izango dira berreskuratu.

3.Objektu anidatuen sorketa.

NewCountriesGUI interfazea nazio berriak datu-basean txertatzeko erabili daiteke. **Save country** botoiak listener bat dauka ziurtatzen diguna sartutako datuak (country, capital) bikote batean gordetzen direla. Baina egindako deialdia **dataAccess** mailan inplementatu gabe dago. Eskatzen da:

- a) **DataAccess.saveCountryAndCity(countryName, countryPopul, capitalName, capitalPopul, airportName, foundationYear)** inplementatu hurrengo moduan: datu-base konexio bat ireki, transakzio berri bat sortu, Country objektu berri bat sortu (**countryName** eta **countryPopul** balioekin) eta **country**-aldagaiari gorde, City objektu bat sortu (**capitalName, capitalPopul, airportName, foundationYear** eta **country**) eta **city**-an gorde, **city** ipini **country** hiriburu bezala, nazio berria gorde **db.persist(country)** eginez, transakzioa baieztatu (commit), datu-basearen konexioa itxi, eta objektua sortzerakoan dena ondo joan den (edo ez) boolear bat itzuli.
- b) Nazio berri bateko datuekin exekuzioa frogatu, adibidez Paraguay. Uharte txikiak diren nazioak salbuetsik, Paraguay, Guyana and Uruguay nazioak falta dira datu-basean. **NewCountriesGUI** klasea eta **saveCountryAndCity** metodoaren bitartez web-ean aurkitutako datuak txertatu (buruz ez badakizkizu).



The screenshot shows a Java Swing window titled "NewCountriesGUI". It contains a form with the following fields and values:

Field	Value
Country:	Paraguay
Population (in millions):	6.85
Capital:	Asuncion
Population (in thousands):	524
Airport:	Silvio Pettirossi
Foundation year:	1541

At the bottom of the form is a button labeled "Save country".

- c) **America.odt** datu-basea ireki objetdb-ren EXPLORER-ekin eta frogatu arazo bat dagoela: Nazio berriari pertsistentzia eman zaio (Paraguay), hiriburu bat esleituta daukadana (Asunción), baina City objektu hori (Asunción) ez da modu egokian gorde. Hiriburuaren izena agertzen da soilik, baina ez beste datuak.

[19]	Country#Paraguay	0
populCountry	int	6849999
nameCountry	String	"Paraguay"
capital	City	0
foundation	int	null
populCity	int	null
mainAirport	String	null
nameCity	String	"Asuncion"
country	Country	null
president	President	null
neighbors	HashSet<Country>	: []

- d) Zuzendu baino lehen, nazioa datu-basetik ezabatu egin behar da: EXPLORER-ean eskuineko botoia sakatu objektuan => Delete => EXPLORER itxi => Aldaketak gorde datu-basean.

Hiriburuaren pertsistentzia modu egokian egiteko bi aukera daude:

- 1- Modu esplizituan, `db.persist (city)` instrukzioa gehituz `saveCountryAndCity` metodoan. Jarraian `NewCountriesGUI` exekutatu, falta den nazio baten datuak sartu (adibidez Paraguay) eta EXPLORER-ean frogatu, orain bai, Asunción hiriburua modu egokian txertatu dela.
- 2- Modu inplizitu edo deklaratioan, Country eta City-ren arteko erlazioaren `@OneToOne` anotazioaren `cascade` propietatean `PERSIST` balioa ipiniz, Country objektua gordetzen denean, bere hiriburua (City objektua) ere gorde behar dela adierazten da.

```
@OneToOne(cascade=CascadeType.PERSIST)
private City capital;
```

Bukatzeko `NewCountriesGUI` exekutatu, eta falta diren nazioak txertatu (adibidez Uruguay eta Guayana) eta EXPLORER-ekin frogatu nazioak eta bere hiriburuak modu egokian txeratu direla.

4.Objektu anidatuen eguneraketa

Orain **ChooseNeighborsGUI** antzeko interfaze bat erabiliko dugu nazioen arteko ondokoak txertatzeko. Bere gertaera fluxua hurrengoa da:

- 1.Sistemak nazio guztien izenak aurkezten ditu.
- 2.Erabilizaileak nazio A aukeratzen du.
- 3.Sistemak A nazioa ezabatzen du nazio zerrendatik eta zerrenda berria aurkezten du.
- 4.Erabilizaileak nahi dituen B1, B2, ..., Bn nazio aukeratzen ditu A-ren ondokoak bezala (Ctrl+Command botoiak sakatzuz)
- 5.Sistemak B1, B2, ..., Bn nazioak A-ren ondokoak bezala gordetzen du eta alderantzizko erlazioak ere, A (B1, B2, Bn)-ren ondokoa da.

ChooseNeighborsGUI interfazeko **Add selected as neighbor countries** botoia sakatzen denean, DataAccess mailan hurrengo metodoa exekutatzen da:

```
public void assignNeighbors (Country home, List<Country> neighbors) {  
    try {  
        openDb();  
        db.getTransaction().begin();  
        Country homeDB = db.find(Country.class, home.getNameCountry());  
        for (Country neighbor : neighbors) {  
            Country neighDB = db.find(Country.class, neighbor);  
            homeDB.addNeighbor(neighDB);  
            neighDB.addNeighbor(homeDB);  
        }  
        // db.persist(homeDB);  
        // db.persist(neighDB);  
        db.getTransaction().commit();  
    } catch (Exception e) {  
        e.printStackTrace();  
    } finally { closeDb(); }  
}
```

Aurreko metodoan hurrengoa egiten da: datu-baseakin konexioa ireki, transakzio bat ireki, datu-basetik berreskuratu aukeratutako nazioa (**homeDB**) eta esleitu nahi diren ondokoen nazioak (**neighborDB**), **homeDB** nazioari esleitu bere ondoko **neighborDB** bakoitza, eta ondoko **neighborDB** bakoitzari **homeDB** nazioa, eta bukatzeko aldaketak baieztatzen dira datu-basean (commit eginez) eta datubasea ixten da.

Duda bakarra, komentatuta dauden `db.persist` `homeDB` eta `neighDB` beharrezko instrukzioak diren ala ez jakitea da. Aplikazioa frogatu `Country` klasearen `neighbors` erlazioaren cascade atributua **PERSIST** balioa daukadanean, jarraian deskribatzen den bezala:

```
@OneToMany(fetch=FetchType.LAZY, cascade=CascadeType.PERSIST)
private Set<Country> neighbors=new HashSet<Country>();
```

Galdera: Zeintzuk dira beharrezko instrukzioak? Jakiteko, aukera desberdinak frogatu daitezke. Txertatu adibidez, Argentinaren bost ondoko nazioak eta jarraian EXPLORER-ekin baieztatu ondo gorde direla.

Ebazpena: Ez da beharrezkoa inongo `persist` eragiketarik, objectdb-ren lehendabiziko laborategian aurkeztu zen bezala, eguneraketak modu gardenean egiten baitira. Datu-basean konektatutako objektuak transakzio baten barruan aldatzen badira automatikoki datu-basean eguneratuko dira. Objektuak datubasean konektatuta daudela ziurtatzeko, aurretik `db.find(Country.class, ...)` egin behar da.

5. Objektu anidatuen ezabaketa

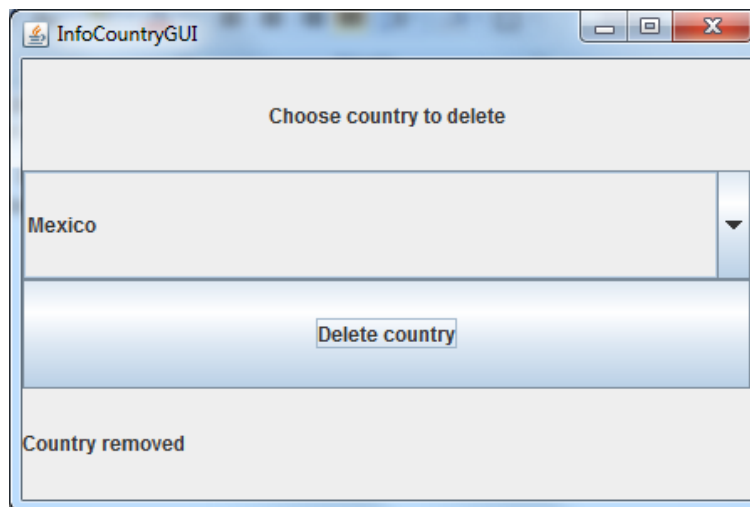
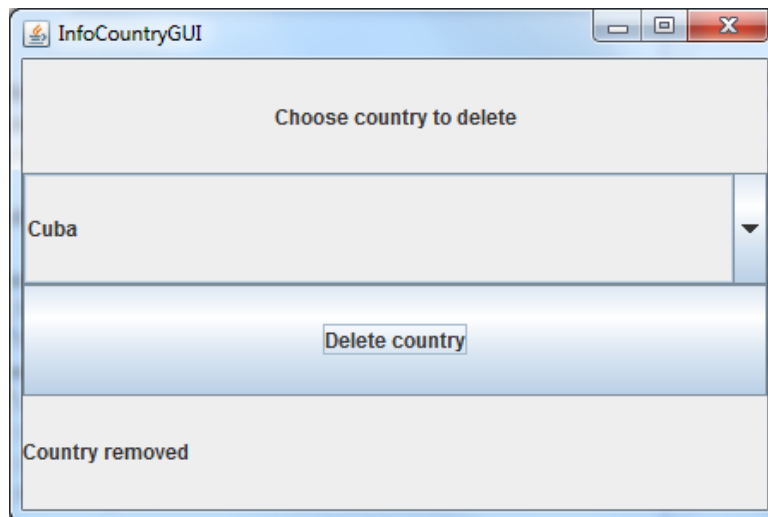
Atal honetan nazioak ezabatuko dira, beraz **komenigarria da uneko datu-basearen kopia bat gordetzea** (originala sortutako nazio berriekin). Proiektua dagoen **America.odb** fitxategiaren kopia bat egin **America-kopia.odb** fitxategira.

RemoveCountryGUI interfazea grafikoak nazioak ezabatzen ditu. **Delete country** boiaren "listener"-ak `removeCountry` metodoari deitzen dio. Metodoa `DataAccess` klasean inplementatuta dago hurrengo moduan:

```
public boolean removeCountry (Country country){
    boolean res=true;
    try {
        openDb();
        db.getTransaction().begin();
        Country c=db.find(Country.class, country.getNameCountry());
        db.remove(c);
        db.getTransaction().commit();
        System.out.println("object removed "+country.getNameCountry());
    } catch (Exception e) {
        e.printStackTrace();
        res=false;
    } finally { closeDb(); }
    return res;
}
```

Metodoak hurrengoa egiten du: datu-basearen konexio bat ireki, transakzio bat ireki, datu-basetik berreskuratu ezabatu nahi den **country** nazioa, ezabatu lortutako nazioa datu-basetik, transakzioa baieztatu (commit) eta datu-basea itxi.

Cuba eta Mexiko nazioak ezabatu. Horretarako **RemoveCountryGUI** exekutatu eta banan-banan Cuba eta Mexico JComboBox-etik aukeratuz “Delete country” botoia sakatu.



ObjectDb-ren EXPLORER-a erabiliz frogatu daiteke hiriburuak ez direla ezabatu datu-basetik. Adibidez, Cuba-ren kasuan, La Habana ez da ezabatu.

Hurrengo irudian agertzen den bezala, nahiz eta Cuba objektua Country objektu bezala ez agertu, La Habana objektuarekin erlazionatuta dago (baina ez balio guztiekin, nazioaren izenarekin soilik)

[8]	City#La Habana	0
foundation	int	1515
populCity	int	2100000
mainAirport	String	"José Martí"
nameCity	String	"La Habana"
country	Country	0
populCountry	int	null
nameCountry	String	"Cuba"
capital	City	null
president	President	null
neighbors	Set<Country>	null

Nazioa eta bere hiriburua ezabatzeko hurrengo anotazioa ipini daiteke **capital** propietatean:

```
@OneToOne(cascade=CascadeType.REMOVE)
private City capital;
```

Baina, **PERSIST** bezala anotatuta zegoenez, biak mantentzeko hurrengo moduan definitu daiteke:

```
@OneToOne(cascade=CascadeType.ALL)
private City capital;
```

RemoveCountryGUI berriz exekutatzen bada (**America-kopia.odt America.odt**-ra lehendik kopiatuz) Cuba eta Mexico ezabatzeko, orain bai frogatu daiteke, La Habana eta Ciudad de Mexico ere ezabatu direla.

Cuba eta Mexico-ren ondoko nazioei buruz, frogatu daiteke, nahiz eta nazio bezala ezabatu datu-basetik, Mexico United States-en ondoko bezala jarraitzen duela, ez objektu bezala bere balio guztiekin, baizik eta bere izenarekin soilik.

[19]	Country#United States	0
populCountry	int	320000000
nameCountry	String	"United States"
capital	City#Washington DC	0
president	President#[Ljava.lang.Object;@122d4cf	0
neighbors	HashSet<Country>	2 objects: [0, 0]
[0]	Country#Canada	0
[1]	Country	0
populCountry	int	null
nameCountry	String	"Mexico"
capital	City	null
president	President	null
neighbors	Set<Country>	null

Aurrekoa ez da Cuba-rekin gertatzen, ez uharte bat delako, ondokorik ez daukadalako datu-basean baizik.

Hurrengo arazoaren ebazpena ez da hain erraza. Norbaitek pentsatu dezake Country klasean bere ondoko guztien ezabaketa "cascade" moduan egitea:

```
@OneToMany(fetch=FetchType.LAZY, cascade=CascadeType.REMOVE)
private Set<Country> neighbors=new HashSet<Country>();
```

Baina hori ez da nahi duguna: Mexico ezabatzen denean bere ondokoak ezabatzea (United States). Berez, Mexico ezabatzen denean, United States-en ondokoan zerrendatik Mexico ezabatu nahi da, gauza zeharo deberdina.

Arazo hau konpontzeko, ez dago inongo modu automatikorik. Hau egiteko, DataAccess klasearen `public boolean removeCountry (Country country)` metodoan hurrengo kodea gehitu beharko genuke nazioa esplizituki ezabatzeko ondokoan zerrendatik:

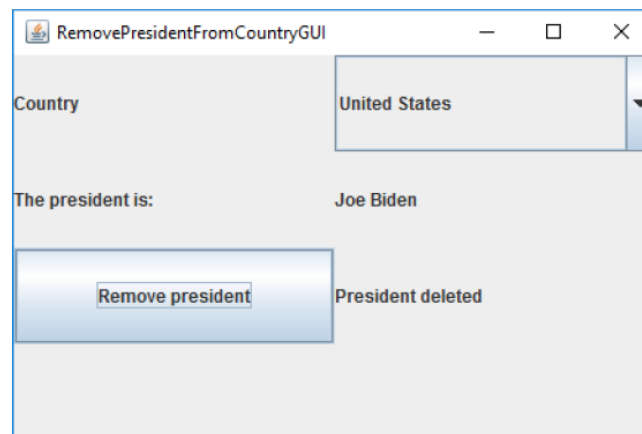
```
for (Country nc: c.getNeighbors())
    nc.getNeighbors().remove(c);
```

Software garatzailea honetaz arduratu behar da, programa baten bitartez.

6.Objektu baten objektu erlazionatu/agregatu baten ezabaketa

Orokorrean, ezabatu nahi den objektua beste objektu batekin erlazionatuta dago. Objektu hauek arduratzen dira objektu hauen sorketaz, eta ondorioz, bere ezabaketaz ere arduratu behar dira¹. Gure kasuan, Country klasea arduratuko da bere presidentearen `president` sorketaz, eguneraketaz eta ezabaketaz.

Horretarako `RemovePresidentFromCountryGUI` interfazea erabiltzen da.



¹ Hau horrela da, GRASP CREATOR patroia erabiltzen denean.

Ezabaketa implementatzen duen metodoa DataAccess klasean hurrengoa da:

```
public boolean removePresidentFromCountry (Country country){
    boolean res=true;
    try {
        openDb();
        db.getTransaction().begin();
        Country c=db.find(Country.class, country.getNameCountry());
        c.setPresident(null);
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
        res=false;
    } finally { closeDb(); }
    return res;
}
```

Metodoak hurrengoa egiten du: datu-basearekin konexio bat ireki, transakzio bat ireki, ezabatu nahi den presidentearen c nazioa berreskuratu datu-basetik, nazio horren presidentearen informazioa ezabatu (null-era ipini), transakzioaren ezabaketa baieztatu (commit) eta datu-basea itxi. Kontuan hartu, instrukzio garrantzitsua `c.setPresident(null)` dela. Instrukzio honek ez du presidente objektua ezabatzen, c nazioaren presidentea null ipini baizik.

EXPLORER-ekin frogatu daiteke presidentea `null` dela.

[21]	Country#United States	{}
populCountry	int	320000000
nameCountry	String	"United States"
capital	City#Washington DC	{}
president	President	null
neighbors	HashSet<Country>	2 objects: [{}]

Baita ere frogatu daiteke presidentearen objektua datu-basean jarraitzen duela, nahiz eta inongo naziora erlazionatua egon. Emaita hau zuzena da, berez inongo ezabaketarik egin ez baitugu.

[2]	President#domain.President@43d3f07d	{}
ended	int	0
started	int	0
familyName	String	"Biden"
firstName	String	"Joe"
accessPower	AccessMode	null
precededBy	President#domain.President@5c86e02	{}
succeededBy	President	null

Baina presidentea datu-basetik ezabatzea interesgarria izango litzateke, beste inongo objektutik erreferentziatua ez balego. Erreferentziatuta ez dagoen objektu bat datu-basetik ezabatzeko, Country klasearen `president` atributuan `orphanRemoval=true` anotazioa ipini behar da.

```
@OneToOne(orphanRemoval=true)
private President president;
```

RemovePresidentFromCountryGUI klasea berriz exekutatu, `orphanRemoval=true` `president` atributuan ipini eta **America-kopia.odt** **America.odt**-ra kopiatu ondoren. Frogatu “Joe Biden” presidentea ezabatu dela.

1. “Joe Biden” beste nazioko presidentea izango balitz, ez litzateke borratuko.
2. `orphanRemoval` atributua `OneToMany` erlazioetan erabili daiteke. Adibidez, presidente baten ordeztu, asko egongo balitz:

```
@OneToMany(orphanRemoval=true)
private Set<President> presidents=new HashSet<President>();

p presidentea ezabatzeko, “HashSet”-etik ezabatu beharko litzateke
presidents.remove(p) instrukzioarekin (ez null esleituz). p presidentea
datubasetik ezabatuko litzateke soilik, p presidentea inongo presidente
zerrendetan egongo ez balitz.
```

7.JPQL: Datubase kontsultatzeko lengoaia.

Lehendabiziko ObjectDb laborategian JPQL kontsulta sinple batzuk aurkeztu ziren, orain kontsulta gehiago aurkeztuko dira.

1. `popul` baino handiago dituzten hiriburuak. JPQL kontsulta SQL-ren sintaxi berdina dauka (SELECT, FROM eta WHERE atalekin).

```
SELECT ci FROM City ci WHERE ci.populCity>="+popul
```

.DataAccess klasearen hurrengo metodoan inplementatuta dago:

```
public Collection<City> hugeCities(int popul)
```

2. `popul` baino handiago dituzten hiriburuaren nazioak. JPQL kontsulta JOIN bat egin behar du COUNTRY eta CAPITAL-en artean nazio horiek lortzeko:

```
SELECT c FROM Country c join c.capital ci where ci.populCity>="+popul
```

.DataAccess klasearen hurrengo metodoan inplementatuta dago:

```
public Collection<Country> countriesWithHugeCapitals1(int popul)
```

Beste modu errazago bat JOIN erabili gabe hurrengo da. Kasu honetan City objektutik bere country atzitu daiteke.

```
SELECT ci.country FROM City ci where ci.populCity>="+popul
```

.DataAccess klasearen hurrengo metodoan inplementatuta dago:

```
public Collection<Country> countriesWithHugeCapitals2(int popul)
```

3. Beste modu bat JOIN erabili gabe, Country objektuaren `capital`-era nabigatuz (City-rena) inplementatu daiteke:

```
SELECT c FROM Country c where c.capital.populCity>="+popul
```

.DataAccess klasearen hurrengo metodoan inplementatuta dago:

```
public Collection<Country> countriesWithHugeCapitals3(int popul)
```

Azkeneko kontsultan `c.capital.populCity` erabiltzen da. Kasu honetan kontsulta hau ez da posible SQL-n egitea, Objektu Zuzendutako lengoaien ezaugarri bat delako, JPQL bezala.

4. Bukatzeko, galdera batzuk ezin izango dira JPQL-n definitu (adibidez, bere ondoko guztien hiriburuaren populazioa baino handiago dituzten hiriburuak itzuli) . Kasu honetan programazio lengoia baten bitartez egin beharko dira.

Aurreko galdera (`popul` baino handiago dituzten hiriburuak) `.DataAccess`-eko `countriesWithHugeCapitals4` metodoan agertzen da:

```
public Collection<Country> countriesWithHugeCapitals4(int popul){  
  
    ArrayList<Country> res=new ArrayList();  
    try {  
        openDb();  
        TypedQuery<Country> query =  
            db.createQuery("SELECT c FROM Country c", Country.class);  
        List<Country> results = query.getResultList();  
  
        for (Country pais : results) {  
            if (pais.getCapital().getPopulCity()>=popul)  
                res.add(pais);  
        }  
    } catch(Exception e) {e.printStackTrace();}  
    finally { closeDb(); }    return res;  
}
```

Kontsulta guztiak Main exekutatzuz frogatu daitezke.



8. Urruneko datu-base baten atzipena.

Datu atzipen geruza inplementatzen duen klasea aldatu behar dugu, urruneko datu-base batera atzitzeko. Horretarako hurrengo egin beharko da:

1. objectDB datu-base zerbitzaria martxan ipini nahi den makinan (urruneko makinaren rola jokatu du). Hasieran, urruneko makina zuen makina izan daiteke.

<http://www.objectdb.com/java/jpa/tool/server> objectDB-ko dokumentazioan agertzen den bezala, nahikoa da zerbitzaria exekutatu (objectdb.jar fitxategian dagoen com.objectdb.Server klasea exekutatzen da) komando lerrotik (cmd), datu-base zerbitzaria eskaerak onartzeko 6136 portuan. Gelditzeko, "stop" ipini behar da "start"-en ordeztu.

```
> java -cp objectdb.jar com.objectdb.Server -port 6136 start
```

Zerbitzaria ere exekutatu daiteke komando lerro bat irekiz (cmd) eta objectdb-2.6.9_09\bin dagoen explorer.exe fitxategia exekutatuz.

Beste aukera bat laborategi honetan eskaintzen den ObjectdbManagerServer klasea exekutatzea da.

2. DataAccess klasea aldatu, datu-base konexio bat irekitzen den bakoitzean, urruneko datu-basearekin konektatzeko. Beste datu-basearen eragiketak, kontsultak, txertaketak, eguneraketak eta ezabaketak berdinak dira.

Datu-base bat lokalean irekitzeko hurrengo instrukzioak ipini behar dira:

```
emf = Persistence.createEntityManagerFactory(dbName);  
db = emf.createEntityManager();
```

Urruneko datu-base batera konektatzeko *createEntityManagerFactory* metodora bidaltzen zaion informazioa aldatu egin behar da: makinaren helbidea, portua eta erabiltzailearen derrigorrezko informazioa (login eta password) bezero/zerbitzari moduan lan egiteko.

Hau da DataAccess openDB metodoaren kode berria:


```
private void openDb() {  
    if (!local) {  
        Map<String, String> properties = new HashMap<String, String>();  
        properties.put("javax.persistence.jdbc.user", user);  
        properties.put("javax.persistence.jdbc.password", pass);  
        emf = Persistence.createEntityManagerFactory(  
            "objectdb://" + ip + ":" + port + "/" + dbName, properties);  
        db = emf.createEntityManager();  
    }  
    else {  
        emf = Persistence.createEntityManagerFactory(dbName);  
        db = emf.createEntityManager();  
    }  
    System.out.println("Database opened");  
}
```

Hurrengo balioak erabiltzen duena (DataAccess atributu bezala definituta daudenak).

```
private String dbName = "America.odb";  
private boolean local=false;  
String ip="localhost";  
int port=6136;  
String user="admin";  
String pass="admin";
```

Puntu honetan, hurrengo egin daiteke:

Aurreko funtzionalitateak exekutatu: **InfoCountryGUI**, **NewCountriesGUI**, **ChooseNeighborsGUI**, Main, ... Arazoak badaude, zerbitzaria **America.odb** datu-basea aurkitzen ez duelako izan daiteke.

Horretarako objectDB datu-basea bilatzen duen leku egokian utzi behar da, hau da, proiektua aurkitzen den "db" direktorio batean (ObjectdbManagerServer klasearen bidez exekutatu bada), edo objectdb-2.6.9_09 aurkitzen den direktorioan (zerbitzaria lerro komandotik exekutatu bada).

Ohar: Trikimailu bat jakiteko **America.odb** non utzi, **NewCountriesGUI** exekutatu eta jarraian begiratu aplikazioak non sortu duen fitxategia.

Azkenik, benetako urruneko zerbitzaria bat atzitu (eta ez localhost), beste ikaskide baten makinaren ip-a ipiniz DataAccess-ean.

```
String ip="158.227...";
```