Ciências ULisboa
Faculdade de Ciências da Universidade de Lisboa

2019/2020

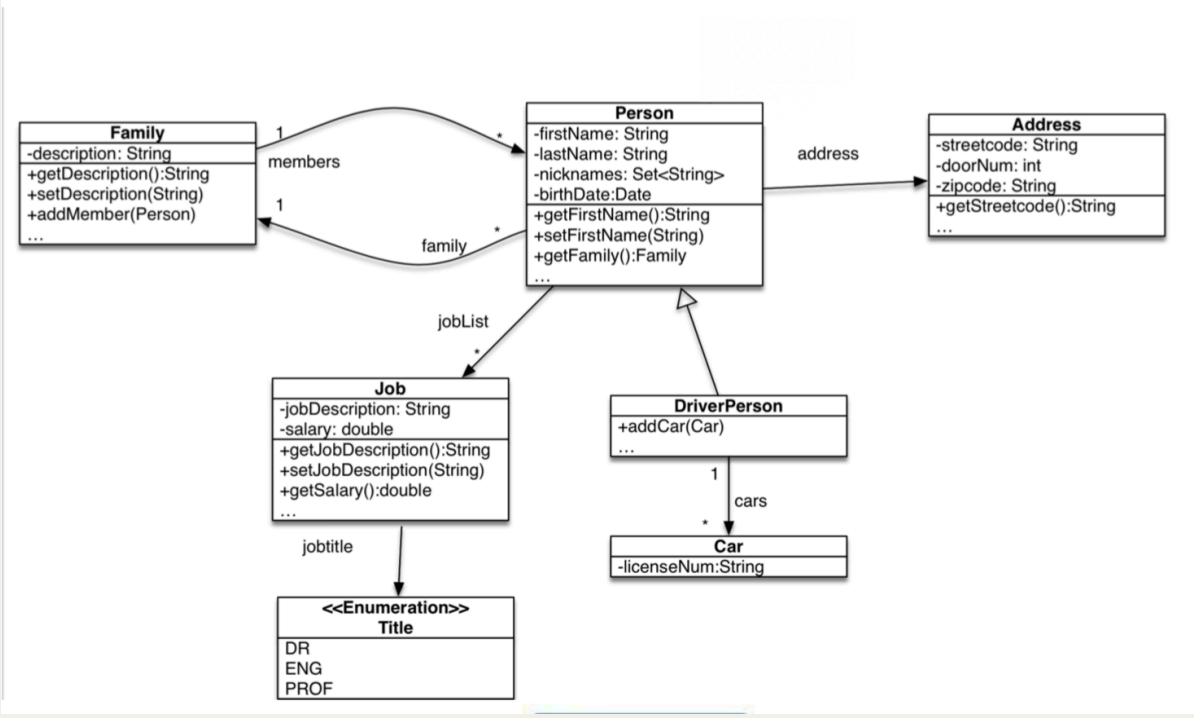
Construção de Sistemas de Software

JPA — The Java Persistence API
ORM Behavioural Patterns

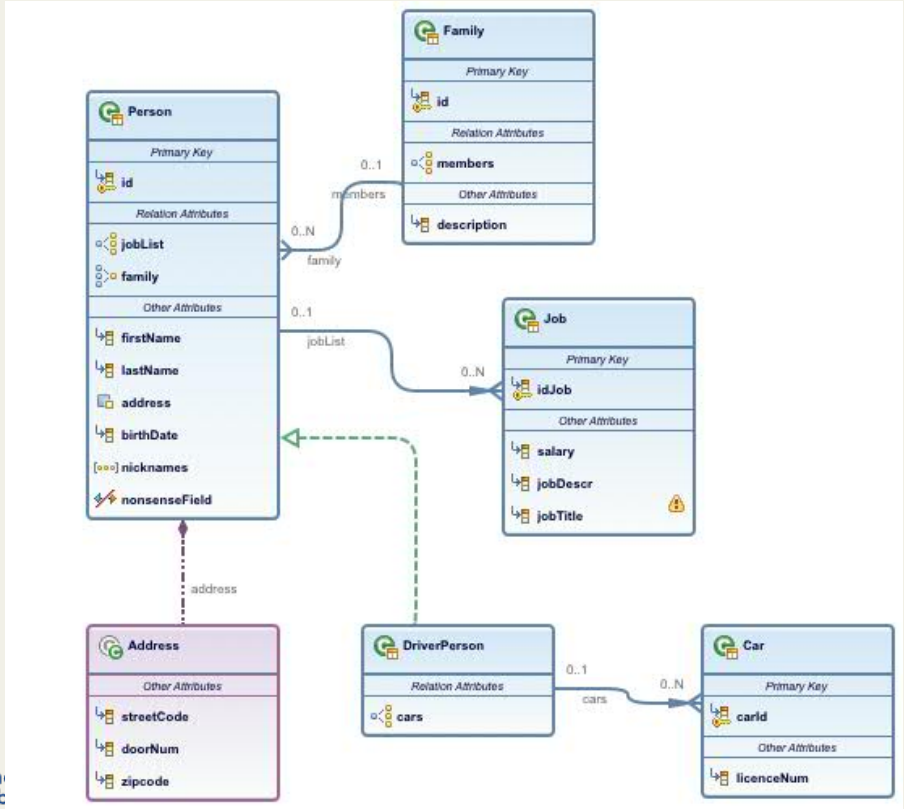
Aspetos Comportamentais do Mapeamento: Padrões

- ***Identity Map*** *controls load of data so taht there are no two objetcs loaded that represent the same table row*
- ***Unit of Work*** *maintains objects affected by a business transaction and coordinates the changes and the resolution of concurrency problems*
- ***Lazy Load*** *An object that does not contain all the data that you need but knows how to get it*
- ***Optimistic Offline Lock*** *validates that the changes about to be commited by one session does not conflict with the changes of another session*
- ***Query Object*** *it is a structure of objects that can form itself into a SQL query*

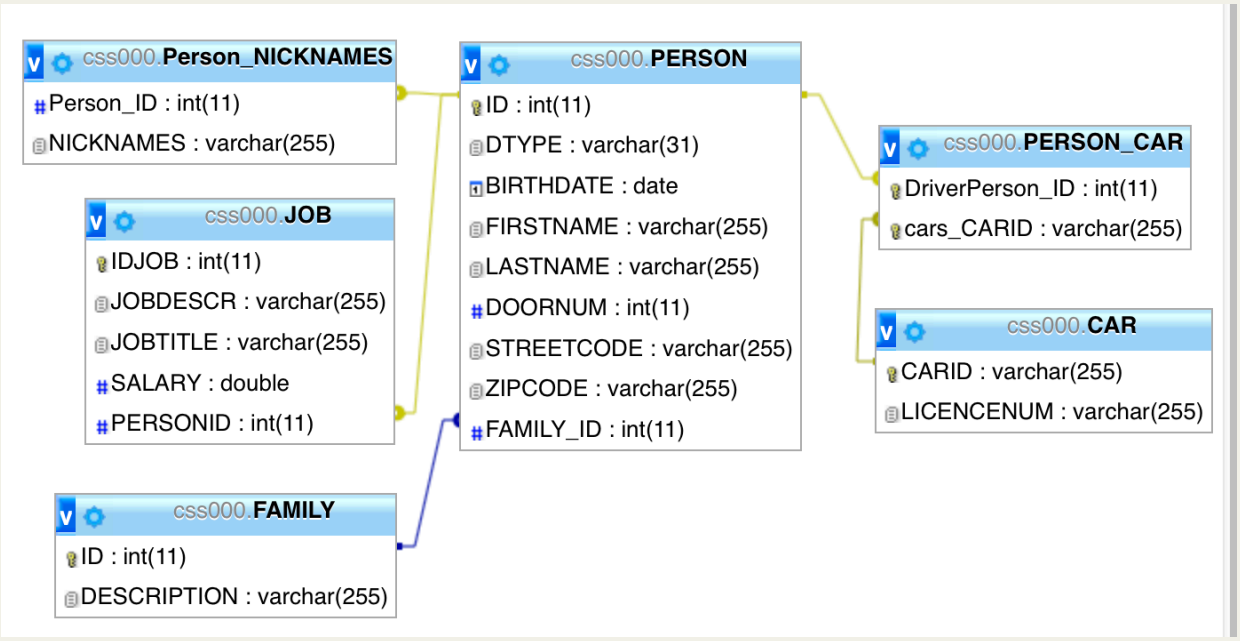
Exercícios



Exercícios



Exercícios

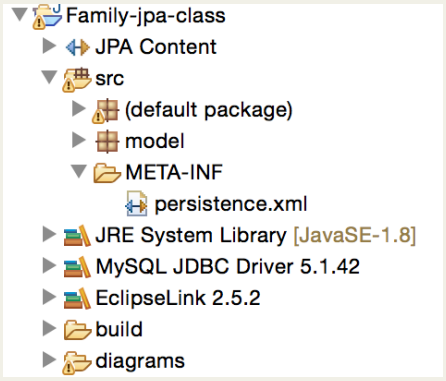


Exercícios

Pretende-se neste exercício exercitar a gestão de entidades no contexto de *application managed persistence*, com *resource local transactions*.

(a) Que tipos (classes e interfaces) definidos pelo JPA nos permitem fazer este tipo de gestão de entidades? Qual o papel de cada um deles? E qual o papel do `persistence.xml`?

Persistência gerida pela aplicação (*non-managed EM*)




```
EntityManagerFactory emf;
try {
    // Connects to the database
    emf = Persistence.createEntityManagerFactory(PERSISTENT_UNIT);
} catch (Exception e) {
    throw new ApplicationException("Error connecting database", e);
}
```

```
emf.close();
```

Persistence.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
     http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
    version="2.0" xmlns="http://java.sun.com/xml/ns/persistence">

    <persistence-unit name="firstjpa" transaction-type="RESOURCE_LOCAL">


        <class>css.firstjpa.model.Person</class>
        <class>css.firstjpa.model.DriverPerson</class>
        <class>css.firstjpa.model.Family</class>
        <class>css.firstjpa.model.Job</class>
        <class>css.firstjpa.model.Address</class>
        <class>css.firstjpa.model.Car</class>

        <properties>
            <property name="javax.persistence.jdbc.driver"
                  value="org.apache.derby.jdbc.EmbeddedDriver" />
            <property name="javax.persistence.jdbc.url"
                  value="jdbc:derby:data/firstjpaDb;create=true" />
            <property name="javax.persistence.jdbc.user" value="css000" />
            <property name="javax.persistence.jdbc.password" value="css000" />

            <!-- EclipseLink should create the database schema automatically -->
            <property name="eclipselink.ddl-generation" value="create-tables" />
            <property name="eclipselink.ddl-generation.output-mode" value="database" />
            <property name="eclipselink.exclude-eclipselink-orm" value="false"/>
        </properties>

    </persistence-unit>
</persistence>
```

nome da unidade de persistência

tipo de transações

Persistência gerida pela aplicação (*non-managed EM*)

```
EntityManager em = emf.createEntityManager();
try {
    em.getTransaction().begin();

                ...

    em.getTransaction().commit();
} catch (Exception e) {
    if (em.getTransaction().isActive())
        em.getTransaction().rollback();
    //possibly throw another exception
} finally {
    em.close();
}
```

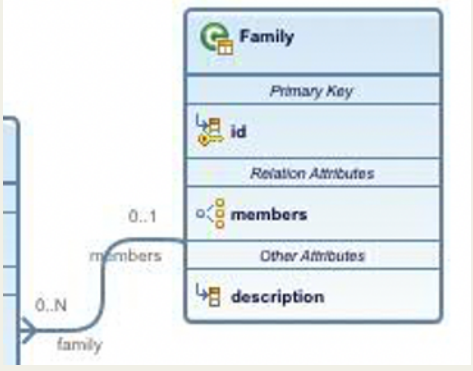
```
em.persist(Object entity)
em.find(Class entityClass,Object key)
em.remove(Object entity)
em.merge(Object entity)
...
```


Exercícios

(b) Numa classe CRUD, programe um método de classe para criar uma família com a assinatura

Family createFamily(EntityManagerFactory emf, String description)

assumindo que a classe Family tem os métodos de que precisa.



Exercícios

```java
public static Family createFamily(EntityManagerFactory emf, String name) {
    EntityManager em = emf.createEntityManager();
    em.getTransaction().begin();

    Family f = new Family();
    f.setDescription(name);
    em.persist(f);

    em.getTransaction().commit();
    em.close();
    return f;
}
```


Exercícios

(c) Indique, justificando, que dados ficam na tabela Family após a execução do seguinte código:

```
1
2  Family fLopes = CRUD.createFamily(emf, "Lopess");
3
4  em = emf.createEntityManager();
5  em.getTransaction().begin();
6  fLopes.setDesignation("Lopes");
7  em.getTransaction().commit();
8  em.close();
```

Exercícios

(c) Indique, justificando, que dados ficam na tabela Family após a execução do seguinte código:

```
1
2  Family fLopes = CRUD.createFamily(emf, "Lopess");
3
4  em = emf.createEntityManager();
5  em.getTransaction().begin();
6  fLopes.setDesignation("Lopes");
7  em.getTransaction().commit();
8  em.close();
```

ID [INTEGER]	DESCRIPTION [VARCHAR(255)]
1051	Lopess

Exercícios

```
em = emf.createEntityManager();
em.getTransaction().begin();
fLopes = em.merge(fLopes);
fLopes.setDescription("Lopes");
em.getTransaction().commit();
em.close();
```

```
em = emf.createEntityManager();
em.getTransaction().begin();
fLopes.setDescription("Lopes");
em.merge(fLopes);
em.getTransaction().commit();
em.close();
```

Exercícios

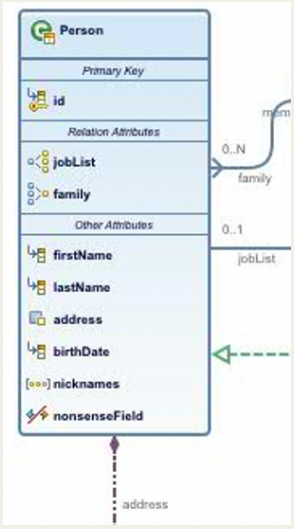
```
em = emf.createEntityManager();
em.getTransaction().begin();
fLopes = em.find(Family.class, fLopes.getId());
fLopes.setDescription("Lopes");
em.getTransaction().commit();
em.close();
```


Exercícios

(e) Ainda na classe CRUD, programe um método de classe para criar uma pessoa, com a assinatura

Person createPerson(EntityManagerFactory emf, Family f, String fn, String ln)

assumindo que as classes Family e Person têm os métodos de que precisa.



Exercícios

```
public static Person createPersonBuggy(EntityManagerFactory emf, String firstName,
        String lastName, Family family) {

    EntityManager em = emf.createEntityManager();
    Person p = null;
    try{
        em.getTransaction().begin();
        p = new Person();
        em.persist(p);
        p.setFirstName(firstName);
        p.setLastName(lastName);
        p.setFamily(family);
        family.addMember(p);
        em.getTransaction().commit();
    }
    catch (Exception e) {
        if (em.getTransaction().isActive())
            em.getTransaction().rollback();
    }
    finally {
        em.close();
    }
    return p;
}
```

Exercícios

```
public static Person createPerson(EntityManagerFactory emf, String firstName,
        String lastName, Family family) {
    EntityManager em = emf.createEntityManager();
    Person p = null;
    try{
        em.getTransaction().begin();
        p = new Person();
        em.persist(p);
        p.setFirstName(firstName);
        p.setLastName(lastName);
        p.setFamily(family);
        family.addMember(p);
        em.merge(family);
        em.getTransaction().commit();
    }
    catch (Exception e) {
        if (em.getTransaction().isActive())
            em.getTransaction().rollback();
    }
    finally {
        em.close();
    }
    return p;
}
```

Exercícios

```
//exemplo 1
Family fL = CRUD.createFamily(emf, "Lopes");
CRUD.createPersonBuggy(emf,"Antonia", "Lopes",fL);

//exemplo 2
Family fM = CRUD.createFamily(emf, "Martins");
CRUD.createPerson(emf,"Francisco", "Martins",fM);
```

FAMILY | CRUD.java | MainVa

ID [INTEGER]	DESCRIPTION [VARCHAR(255)]
51	Lopes
52	Martins

PERSON | CRUD.java | MainVariosEM.java

ID [INTEGER]	DTYPE [V.	AL	BIRTHDATE [DATE]	FIRSTNAME [VARCHAR(255)]	LASTNAME [\	VER	DOOI	STREETC	ZIPCODE [VARCHAR(255)]	FAMILY_ID [INTEGER]
51	Person			Antonia	Lopes	1				51
52	Person			Francisco	Martins	1				52

Exercícios

```
//exemplo 1
Family fL = CRUD.createFamily(emf, "Lopes");
CRUD.createPersonBuggy(emf,"Antonia", "Lopes",fL);

//exemplo 2
Family fM = CRUD.createFamily(emf, "Martins");
CRUD.createPerson(emf,"Francisco", "Martins",fM);
```

```
// print data
em = emf.createEntityManager();
Util.printData(em);
em.close();
```

```
------------------------

Lopes (51) with members [ ]
Martins (52) with members [ 52 ]
Antonia Lopes (51) family: 51 jobs: {[]} address:
Francisco Martins (52) family: 52 jobs: {[]}
address:
------------------------
```

Exercícios

(e) Ainda na classe CRUD, programe um método de classe para criar uma pessoa, com a assinatura

`Person createPerson(EntityManagerFactory emf, Family f, String fn, String ln)`

assumindo que as classes Family e Person têm os métodos de que precisa.

(f) Programe um método de classe alternativo para criar uma pessoa, com a assinatura

`Person createPerson(EntityManagerFactory emf, int idFamily, String fn, String ln)`

Exercícios

```
public static Person createPerson(EntityManagerFactory emf, String firstName,
        String lastName, int idFamily) {
    EntityManager em = emf.createEntityManager();
    Person p = null;

    try {
        em.getTransaction().begin();
        p = new Person();
        em.persist(p);
        p.setFirstName(firstName);
        p.setLastName(lastName);
        Family family = em.find(Family.class, idFamily);
        p.setFamily(family);
        family.addMember(p);
        em.getTransaction().commit();
    }
    catch (Exception e) {
        if (em.getTransaction().isActive())
            em.getTransaction().rollback();
    }
    finally {
        em.close();
    }
    return p;
}
```

Exercícios

```
//exemplo 1
Family fL = CRUD.createFamily(emf, "Lopes");
CRUD.createPersonBuggy(emf,"Antonia", "Lopes",fL);

//exemplo 2
Family fM = CRUD.createFamily(emf, "Martins");
CRUD.createPerson(emf,"Francisco", "Martins",fM);

//exemplo 3
Family fF = CRUD.createFamily(emf, "Fonseca");
CRUD.createPerson(emf,"Alcides", "Fonseca",fF.getId());
```

```
// print data
em = emf.createEntityManager();
Util.printData(em);
em.close();
```

```
------------------------

Lopes (151) with members [ ]
Martins (152) with members [ 152 ]
Fonseca (153) with members [ 153 ]
Antonia Lopes (151) family: 151 jobs: {[]} address:
Francisco Martins (152) family: 152 jobs: {[]} address:
Alcides Fonseca (153) family: 153 jobs: {[]} address:
------------------------
```

Ciências ULisboa | Informática

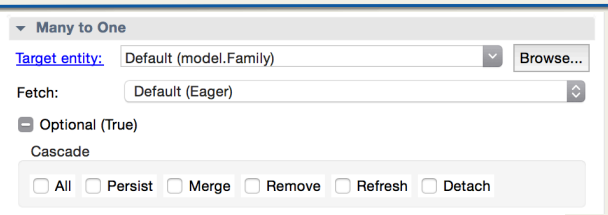
Exercícios

(g) Indique como recorrendo a anotações JPA se pode condicionar a informação que é carregada para memória, na forma de objetos e ligações, quando é feito o find de um Person.

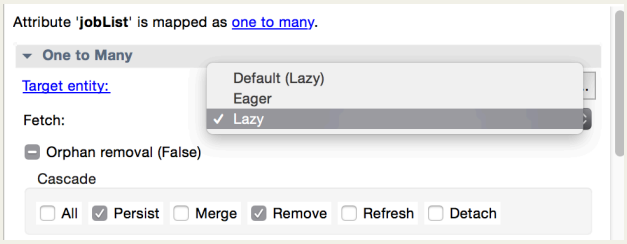
(h) Indique como recorrendo a anotações JPA pode se condicionar o efeito que tem nas diferentes ligações de um objeto Person (nicknames, jobs e family) a execução de operações como o persist, remove e merge.

Ciências ULisboa | Informática

Exercícios

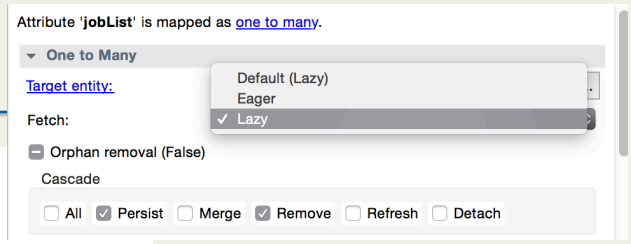



```
@ManyToOne
private Family family;

@Temporal(TemporalType.DATE)
private Date birthDate;

@ElementCollection(fetch=FetchType.LAZY)
private Set<String> nicknames;

@OneToMany(fetch=FetchType.LAZY, cascade = { PERSIST, REMOVE })
@JoinColumn(name ="ID")
private List<Job> jobList = new ArrayList<Job>();
```



Ciências ULisboa | Informática

Lazy Loading




```
static void printData(EntityManager em) {
    printFamilies(em);
    printPersons(em);
    printJobs(em);
    System.out.println("------------------------\n");
}
```

```
static void printPersons(EntityManager em) {
    // read the existing entries and write to console
    TypedQuery<Person> q = em.createQuery("select t from Person t",Person.class);
    List<Person> listp = q.getResultList();
    for (Person p : listp) {
        System.out.println(p);
    }
}
```

```
Lopes (251) with members [ 253 259 ]
Martins (252) with members [ 256 ]
Antonia Lopes (253) family: 251 jobs: {IndirectList: not instantiated} address:
Francisco Martins (256) family: 252 jobs: {IndirectList: not instantiated} address:
Deolinda Lopes (259) family: 251 jobs: {IndirectList: not instantiated} address:
Prof (254)
Mum (255)
Prof (257)
Engineer (258)
Nurse (260)
```

Lazy Loading

```
static void printPersons(EntityManager em) {
    // read the existing entries and write to console
    TypedQuery<Person> q = em.createQuery("select t from Person t",Person.class);
    List<Person> listp = q.getResultList();
    for (Person p : listp) {
        p.getJobList().size();
        System.out.println(p);
    }
}
```

```
Lopes (351) with members [ 353 359 ]
Martins (352) with members [ 356 ]
Antonia Lopes (353) family: 351 jobs: {[Prof (354), Mum (355)]} address:
Francisco Martins (356) family: 352 jobs: {[Prof (357), Engineer (358)]} address:
Deolinda Lopes (359) family: 351 jobs: {[Nurse (360)]} address:
Prof (354)
Mum (355)
Prof (357)
Engineer (358)
Nurse (360)
```


Eager Loading

```
@OneToMany(fetch=FetchType.EAGER, cascade = { PERSIST, REMOVE, MERGE})
@JoinColumn
private List<Job> jobList = new ArrayList<>();
```

```
Lopes (351) with members [ 353 359 ]
Martins (352) with members [ 356 ]
Antonia Lopes (353) family: 351 jobs: {[Prof (354), Mum (355)]} address:
Francisco Martins (356) family: 352 jobs: {[Prof (357), Engineer (358)]} address:
Deolinda Lopes (359) family: 351 jobs: {[Nurse (360)]} address:
Prof (354)
Mum (355)
Prof (357)
Engineer (358)
Nurse (360)
```

Cascade

```
@OneToMany(fetch=FetchType.EAGER, cascade = { PERSIST, REMOVE, MERGE})
@JoinColumn
private List<Job> jobList = new ArrayList<>();
```

```
public void addJob(double salary, String desc) {
    Job job = new Job();
    job.setJobDescr(desc);
    job.setSalary(salary);
    this.jobList.add(job);
}
```

```
//exemplo 2
Family fM = CRUD.createFamily(emf, "Martins");
Person francisco = CRUD.createPerson(emf,"Francisco", "Martins",fM);
```

```
//exemplo 5
em = emf.createEntityManager();
em.getTransaction().begin();
francisco = em.merge(francisco);
francisco.addJob(1000, "Prof");
em.getTransaction().commit();

//exemplo 6
em = emf.createEntityManager();
em.getTransaction().begin();
francisco = em.merge(francisco);
francisco.addJob(500, "Sailor");
em.getTransaction().commit();
```



```
Francisco Martins (52) family: 52 jobs: [Prof (54) Salary 1000.0, Sailor (55) Salary 500.0]
```

Exercícios

Ainda no contexto do mesmo exemplo, equipe a classe Person com interrogações com nome que lhe permitam juntar à classe CRUD

(a) um método de classe para calcular a lista de pessoas com um determinado nome próprio com assinatura

List<Person> findAllPersonsByFirstName(EntityManager em, String fname)

(b) um método de classe para calcular a lista de pessoas de uma família com um determinado identificador

List<Person> findAllPersonsByFamilyId(EntityManager em, int id)

(c) um método de classe para calcular a lista de pessoas com um salário superior a um dado valor

List<Person> findAllPersonsWithLargerSalary(EntityManager em, int salary)

Interrogações JPQL

- `TypedQuery<?>` permite fazer interrogações sobre os objetos

```
TypedQuery query =
    em.createQuery("select m from Livro m where m.id=:id",
                   Livro.class);
query.setParameter("id", 1234);
Livro livro = query.getSingleResult();
```

- Melhor usar interrogações com nome —`NamedQuery`— porque fica a sua definição separada da utilização
- Faz sentido definir as `NamedQuery` nas classes que definem as entidades que são alvo da interrogação.
- Para ter mais do que uma `NamedQuery` na mesma classe entidade é preciso embrulhá-las com `@NamedQueries`
- Uma outra boa prática é recorrer à definição de constantes para os nomes das interrogações e dos parâmetros que estas definem

Interrogações JPQL

```
@NamedQuery(name="Person.findByLastName",
            query="SELECT p FROM Person p WHERE p.lastName LIKE :name")
```

```
public static List<Person> findAllPersonsByLastName(EntityManager em, String lastName) {
    TypedQuery<Person> q = em.createNamedQuery("Person.findByLastName",Person.class);
    q.setParameter("name", lastName);
    return q.getResultList();
}
```

Interrogações JPQL

```
@Entity
@NamedQueries({
    @NamedQuery(name=Person.FIND_BY_FIRSTNAME,
            query="SELECT p FROM Person p WHERE p.firstName LIKE :"+ Person.NAME),
    @NamedQuery(name=Person.FIND_BY_LASTNAME,
    query="SELECT p FROM Person p WHERE p.lastName LIKE :"+ Person.NAME),
    @NamedQuery(name=Person.FIND_BY_FAMILYID,
    query="SELECT p FROM Person p WHERE p.family.id = :" + Family.ID),
    @NamedQuery(name=Person.FIND_BY_FAMILYID_ORDERED,
    query="SELECT p FROM Person p WHERE p.family.id = :" + Family.ID + " ORDER BY p.birthDate"),
    @NamedQuery(name=Person.FIND_BY_SALARY,
    query="SELECT p FROM Person p, IN(p.jobList) AS j WHERE j.salary > :" + Job.SALARY),









})
@Inheritance(strategy=SINGLE_TABLE)
public class Person {
```


Exercícios

```
@Inheritance(strategy=SINGLE_TABLE)
public class Person {

    // Named query name constants
    public static final String FIND_BY_FIRSTNAME = "Person.findByFirstName";
    public static final String FIND_BY_LASTNAME = "Person.findByLastName";
    public static final String NAME = "name";
    public static final String FIND_BY_FAMILYID = "Person.findByFamilyId";
    public static final String FIND_BY_SALARY = "Person.findBySalary";
```

```
    public static List<Person> findAllPersonsByFistName(EntityManager em, String firstName) {
        TypedQuery<Person> q = em.createNamedQuery(Person.FIND_BY_FIRSTNAME,Person.class);
        q.setParameter(Person.NAME, firstName);
        return q.getResultList();
    }
```

```
  public static List<Person> findAllPersonsByJobSalary(EntityManager em, int salary) {
      TypedQuery<Person> q = em.createNamedQuery(Person.FIND_BY_SALARY,Person.class);
      q.setParameter(Job.SALARY, salary);
      return q.getResultList();
  }
```

Exercícios

Ainda no contexto do mesmo exemplo, considere o caso de uso para criar uma pessoa. Neste caso de uso há uma primeira operação para indicar o nome, apelido e o identificador da sua família, uma segunda operação, que pode ser chamada várias vezes, para adicionar à pessoa corrente um emprego dada a descrição, título e salário e uma terceira operação para sinalizar o fim do caso de uso.

Desenhe as várias operações recorrendo a uma classe CreatePersonHandler com um construtor que recebe apenas um FamilyCatalog e um PersonCatalog, identificando as responsabilidades que têm de ser cumpridas por estas duas classes. Considere duas alternativas:

(a) o construtor do handler recebe os catálogos e na criação de cada catálogo é passado um objeto do tipo EntityManagerFactory

(b) o construtor do handler recebe um objeto do tipo EntityManagerFactory e é o handler que cria um EntityManagerpara tratar de cada transação de negócio, demarcando a transação e criando os catálogos necessários, a quem passa o EntityManager já criado.

Alternativa 1

```
public class PersonHandler {

     * The sale's catalog
    private PersonCatalog personCatalog;

     * The customer's catalog
    private FamilyCatalog familyCatalog;

     * The current person
    private Person currentPerson;

    public PersonHandler(PersonCatalog personCatalog, FamilyCatalog familyCatalog) {
        this.personCatalog = personCatalog;
        this.familyCatalog = familyCatalog;
    }

    public void newPerson (String firstName, String surname, int idf) throws ApplicationException {
        Family f = familyCatalog.getFamily(idf);
        currentPerson = personCatalog.newPerson(firstName, surname, f);
    }

    public void addJobToPerson (double salary, String desc) throws ApplicationException {
        currentPerson = personCatalog.addJobToPerson(currentPerson, salary, desc);
    }

    public void close () {
        currentPerson = null;
    }

}
```

Alternativa 1

```
public class PersonCatalog {

    /**
     * Entity manager factory for accessing the persistence service
     */
    private EntityManagerFactory emf;

    /**
     * Constructs a person's catalog giving a entity manager factory
     */
    public PersonCatalog(EntityManagerFactory emf) {
        this.emf = emf;
    }

    public Person newPerson (String firstName,
            String lastName, Family family) throws ApplicationException {
        EntityManager em = emf.createEntityManager();
        try {
            em.getTransaction().begin();
            Person p = new Person();
            em.persist(p);
            p.setFirstName(firstName);
            p.setLastName(lastName);
            family.addMember(p);
            p.setFamily(family);
            em.merge(family);
            em.getTransaction().commit();
            return p;
        } catch (Exception e) {
            if (em.getTransaction().isActive())
                em.getTransaction().rollback();
            throw new ApplicationException("Error adding person", e);
        } finally {
            em.close();
        }
    }
```

Alternativa 1

```
@OneToMany(fetch=FetchType.EAGER, cascade = { PERSIST, REMOVE, MERGE})
@JoinColumn
private List<Job> jobList = new ArrayList<>();
```

```
public class PersonCatalog {

    /**
     * Entity manager factory for accessing the persistence service
     */
    private EntityManagerFactory emf;

    /**
     * Constructs a person's catalog giving a entity manager factory
     */
    public PersonCatalog(EntityManagerFactory emf) {
        this.emf = emf;
    }

    public Person newPerson (String firstName, ..

    public Person addJobToPerson (Person p, double salary, String desc)
            throws ApplicationException {
        EntityManager em = emf.createEntityManager();
        try {
            em.getTransaction().begin();
            p.addJob(salary, desc);
            em.merge(p);
            em.getTransaction().commit();
        } catch (Exception e) {
            if (em.getTransaction().isActive())
                em.getTransaction().rollback();
            throw new ApplicationException("Error adding job to person", e);
        } finally {
            em.close();
        }
        return p;
    }
}
```

```
public void addJob(double salary, String desc) {
    Job job = new Job();
    job.setJobDescr(desc);
    job.setSalary(salary);
    this.jobList.add(job);
}
```

Alternativa 2

- Nesta alternativa, que é a exemplificada no SaleSys que receberam na meta 3
  - os entity managers são criados nos métodos dos handlers e passados aos catálogos — os quais não têm estado e são criados sempre que são precisos
  - tudo o que é preciso fazer para realizar uma operação do sistema faz parte da mesma transação (que é uma vantagem pois a realização de uma operação pode implicar fazer várias coisas que ou são realizadas todas com sucesso ou não é realizada nenhuma)

```
public void newPerson (String firstName, String surname, int idf) throws ...{
     EntityManager em = emf.createEntityManager();
     try {
          em.getTransaction().begin();

          FamilyCatalog familyCatalog = new FamilyCatalog(em);
          PersonCatalog personCatalog = new PersonCatalog(em);

          Family f = familyCatalog.getFamily(idf);
          currentPerson = personCatalog.newPerson(firstName, surname, f);

          em.getTransaction().commit();
          }
     {
     ...
```


Alternativa 2

```
public void newPerson (String firstName, String surname, int idf) throws ...{
     EntityManager em = emf.createEntityManager();
     try {
          em.getTransaction().begin();

          FamilyCatalog familyCatalog = new FamilyCatalog(em);
          PersonCatalog personCatalog = new PersonCatalog(em);

          Family f = familyCatalog.getFamily(idf);
          currentPerson = personCatalog.newPerson(firstName, surname, f);

          em.getTransaction().commit();
          }
     } catch (Exception e) {
          if (em.getTransaction().isActive())
               em.getTransaction().rollback();
               throw new ApplicationException("Error adding person", e);
     } finally {
          em.close();
}
```