Ciências ULisboa
Faculdade de Ciências da Universidade de Lisboa

2019/2020

Construção de Sistemas de Software

JPA — The Java Persistence API

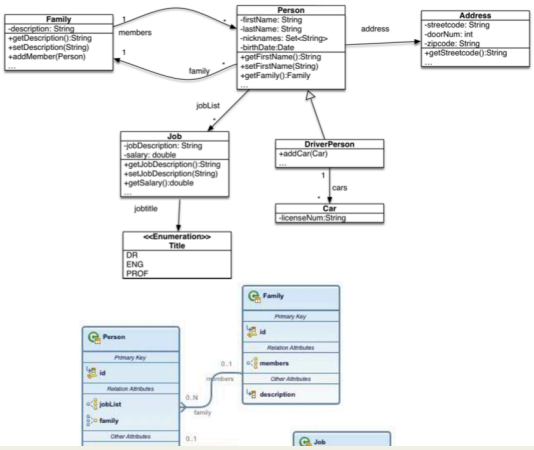
Exercícios

Ciências ULisboa | Informática

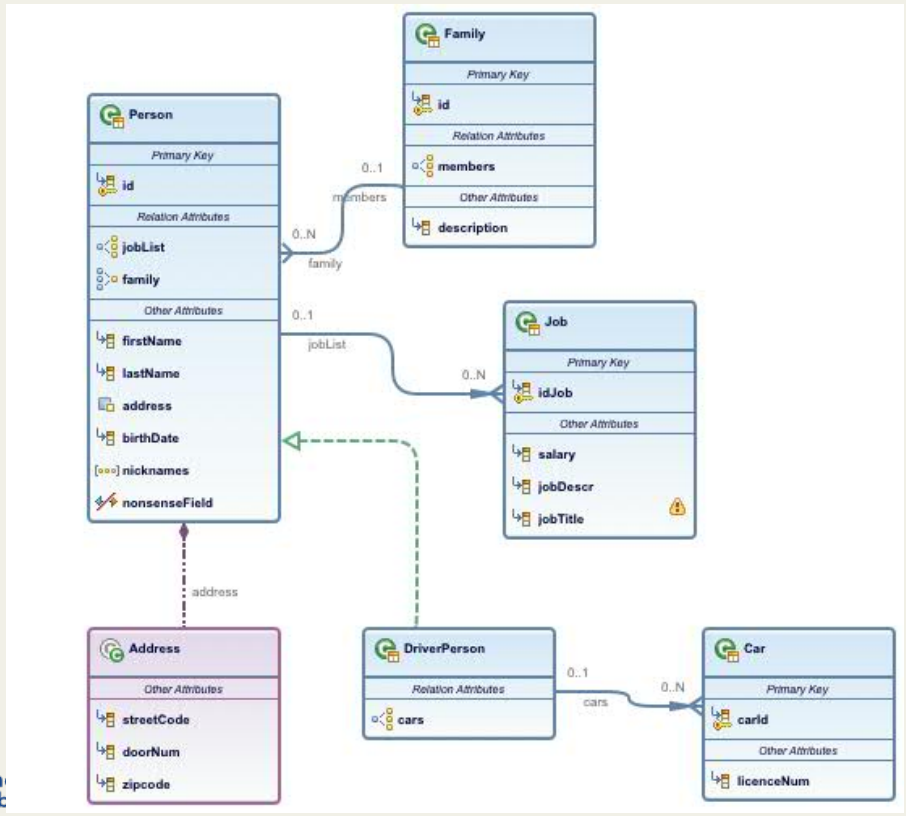
CSS (2019/2020)
Caderno de Exercícios
Folha 6

JPA Mapper (ORM Behaviour Patterns)

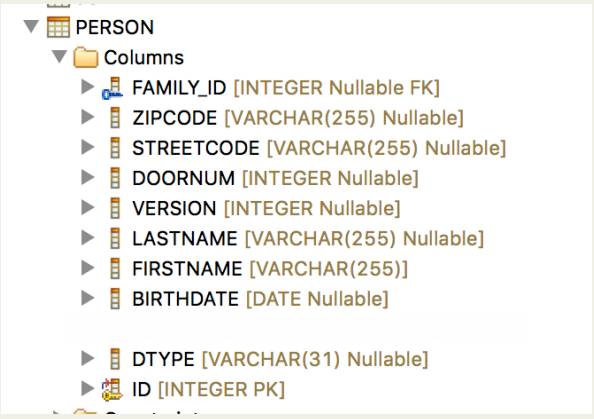
1. No exercício 1 da folha anterior, recorrendo a anotações JPA, estabelecemos o mapeamento entre o Modelo de Objetos abaixo e um Modelo Relacional. Este mapeamento é descrito no diagrama seguinte.

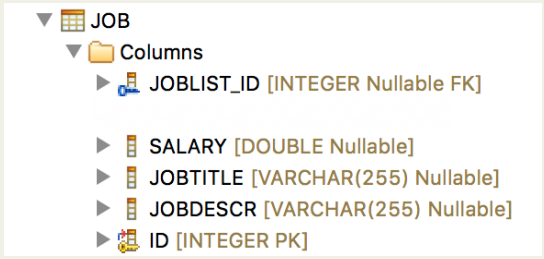


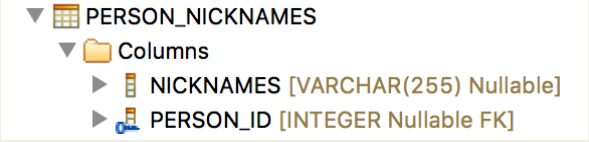
Exercícios



Exercícios

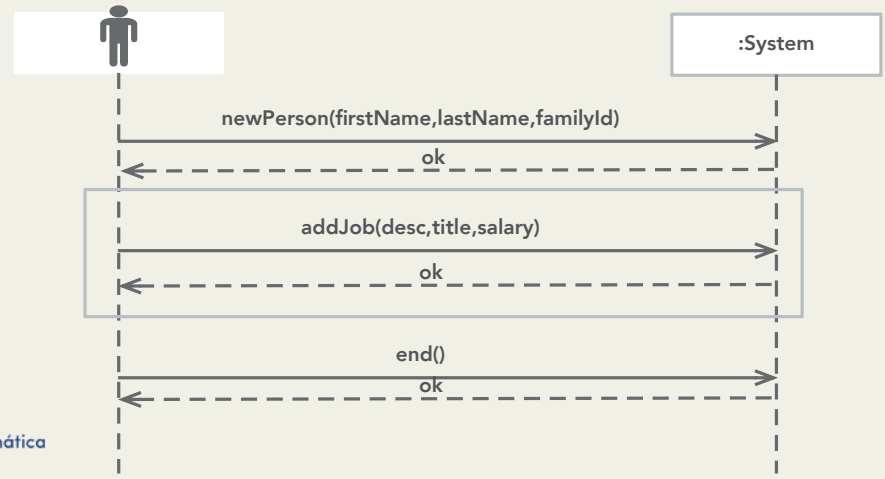






Exercícios

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.



Exercícios

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.

Alternativas

- **Alternativa 1**
  - handler guarda os catálogos que recebe no construtor e chama operações sobre esses catálogos
  - catálogos recebem um EntityManagerFactory no construtor e responsáveis pela criação dos EntityManagers e transações

- **Alternativa 2**
  - exemplificada no SaleSys que receberam na meta 4
  - 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)

Exercícios

- PersonHandler(PersonCatalog personCatalog, FamilyCatalog familyCatalog)

e métodos

- **int** newPerson (String firstName, String surname, **int** idFamily) — cria uma pessoa com os dados fornecidos, que passa a ser a pessoa corrente, e devolve o seu identificador
- **void** addJobToPerson (**double** salary, String descJob) — adiciona o emprego com os dados fornecidos à pessoa corrente, passando esse a ser o seu último emprego
- **void** findPerson (**int** id) — a pessoa corrente passa a ser a que tem o identificador dado
- **void** changeCurrentSalaryPerson (**double** newSalary) — altera o salário do último emprego da pessoa corrente
- **double** higherSalary () — indica o salário mais alto alguma vez auferido pela pessoa corrente

(a) Suponha que estas classes fazem parte de uma aplicação multi-utilizador. Que problemas vão existir quando houver dois objetos do tipo PersonHandler sobre os quais é executado: findPerson com o mesmo id e seguidamente é executado changeCurrentSalaryPerson cada um com o seu valor?

```java
public class PersonHandler {

    /** The person's catalog. */
    private PersonCatalog personCatalog;

    /** The family's catalog. */
    private FamilyCatalog familyCatalog;

    /** The current person. */
    private Person currentPerson;

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

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

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

    public void findPerson (int id) throws ApplicationException {
        currentPerson = personCatalog.findById(id);
    }

    public void changeCurrentSalaryPerson (double newSalary) throws ApplicationException {
        currentPerson = personCatalog.changeCurrentSalaryPerson(currentPerson, newSalary);
    }

    public double higherSalary () throws ApplicationException {
        return personCatalog.higherSalary(currentPerson);
    }

    public void close () {
        currentPerson = null;
    }

}
```

```java
public class PersonCatalog {

    public Person changeCurrentSalaryPerson(Person p, double newSalary) throws ApplicationException {
        EntityManager em = emf.createEntityManager();
        try {
            em.getTransaction().begin();
            p = em.merge(p);
            p.changeSalaryCurrentJob(newSalary);
            em.getTransaction().commit();
        } catch (Exception e) {
            if (em.getTransaction().isActive())
                em.getTransaction().rollback();
            throw new ApplicationException("Error changing salary of current job of person", e);
        } finally {
            em.close();
        }
        return p;
    }
```

```java
public class Person {

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

```java
public void changeSalaryCurrentJob(double newSalary) {
    if(!jobList.isEmpty()) {
        jobList.get(jobList.size()-1).setSalary(newSalary);
    }
}
```

Exercícios

```
Lopes (251) with members [ 253 259 ]
Martins (252) with members [ 256 ]
Antonia Lopes (253) family: 251 jobs: [Prof (254) Salary 1000.0, Mum (255) Salary 0.0] address:
Francisco Martins (256) family: 252 jobs: [Prof (257) Salary 1000.0, Engineer (258) Salary 1500.0] address:
Deolinda Lopes (259) family: 251 jobs: [Nurse (260) Salary 500.0] address:
Prof (254) Salary 1000.0
Mum (255) Salary 0.0
Prof (257) Salary 1000.0
Engineer (258) Salary 1500.0
Nurse (260) Salary 500.0
```

```java
Thread t1 = new Thread(){
    @Override public void run() {
        FamilyCatalog fc = new FamilyCatalog(emf);
        PersonCatalog pc = new PersonCatalog(emf);
        PersonHandler h = new PersonHandler(pc, fc);
        try {
            h.findPerson(idFrancisco);
            h.changeCurrentSalaryPerson(3000);
        } catch (ApplicationException e) {
```

```java
Thread t2 = new Thread(){
    @Override public void run() {
        FamilyCatalog fc = new FamilyCatalog(emf);
        PersonCatalog pc = new PersonCatalog(emf);
        PersonHandler h = new PersonHandler(pc, fc);
        try {
            h.findPerson(idFrancisco);
            h.changeCurrentSalaryPerson(5000);
        } catch (ApplicationException e) {
```


Exercícios

(b) Explique como pode tirar partido do mecanismos de gestão de concorrência baseado em versões oferecido pelo JPA para evitar os problemas identificados. O que vai passar a acontecer durante a execução concorrente descrita anteriormente?

Exercícios

```java
@Entity
public class Job {
    public static final String TITLE = "title";
    public static final String SALARY = "salary";

    @Id
    @GeneratedValue
    private int idJob;

    @Version
    private int version;

    private double salary;

    private String jobDescr;

    @Enumerated(EnumType.STRING)
    private Title jobTitle;

    public int getId() {
        return idJob;
    }
```

```java
public class Person {
    @Id @GeneratedValue
    private int id;

    @Version
    private int version;

    @Column(nullable = false)
    private String firstName;
    private String lastName;

    @Embedded
    private Address address;

    @ManyToOne
    private Family family;

    @Temporal(TemporalType.DATE)
    private Date birthDate;

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

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

Ciências ULisboa | Informática

Entidade Job com @Version

```
[EL Warning]: 2018-04-05 16:13:54.158--UnitOfWork(1651265)--Exception [EclipseLink-5006] (Eclipse Persistence Services - 2.6.4.v20160829-44060b6): org.
Exception Description: The object [Engineer (308) Salary 5000.0] cannot be updated because it has changed or been deleted since it was last read.
Class> model.Job Primary Key> 308
Error: Error changing salary of current job of person
Cause:
model.ApplicationException: Error changing salary of current job of person
DEPOIS
        at model.PersonCatalog.changeSalaryPerson(PersonCatalog.java:78)
        at model.PersonHandler.changeCurrentSalaryPerson(PersonHandler.java:46)
        at main.MainConcurrentHandlers$2.run(MainConcurrentHandlers.java:55)
Caused by: javax.persistence.RollbackException: javax.persistence.OptimisticLockException: Exception [EclipseLink-5006] (Eclipse Persistence Services -
Exception Description: The object [Engineer (308) Salary 5000.0] cannot be updated because it has changed or been deleted since it was last read.
Class> model.Job Primary Key> 308
```

```
Lopes (201) with members [ 203 209 ]
Martins (202) with members [ 206 ]
Antonia Lopes (203) family: 201 jobs: [Prof (204) Salary 1000.0, Mum (205) Salary 0.0] address:
Francisco Martins (206) family: 202 jobs: [Prof (207) Salary 1000.0, Engineer (208) Salary 3000.0] address:
Deolinda Lopes (209) family: 201 jobs: [Nurse (210) Salary 500.0] address:
Prof (204) Salary 1000.0
Mum (205) Salary 0.0
Prof (207) Salary 1000.0
Engineer (208) Salary 3000.0
Nurse (210) Salary 500.0
------------------------
```

Ciências ULisboa | Informática

Exercícios

(c) Suponha agora tem dois objetos do tipo PersonHandler sobre os quais é executado: findPerson com o mesmo id e seguidamente é executado sobre um changeCurrentSalaryPerson e sobre o outro higherSalary . O que acontece neste caso?

Entidade Job com atributo anotado com @Version

```
Thread t1 = new Thread(){
    @Override public void run() {
        FamilyCatalog fc = new FamilyCatalog(emf);
        PersonCatalog pc = new PersonCatalog(emf);
        PersonHandler h = new PersonHandler(pc, fc);
        try {
            h.findPerson(idFrancisco);
            System.out.println("Salario Maior "+ h.higherSalary());
        } catch (ApplicationException e) {
```

```
Thread t2 = new Thread(){
    @Override public void run() {
        FamilyCatalog fc = new FamilyCatalog(emf);
        PersonCatalog pc = new PersonCatalog(emf);
        PersonHandler h = new PersonHandler(pc, fc);
        try {
            h.findPerson(idFrancisco);
            h.changeCurrentSalaryPerson(5000);
        } catch (ApplicationException e) {
```

```
Salario Maior 1500.0
DEPOIS
------------------------

Lopes (501) with members [ 503 509 ]
Martins (502) with members [ 506 ]
Antonia Lopes (503) family: 501 jobs: [Prof (504) Salary 1000.0, Mum (505) Salary 0.0] address:
Francisco Martins (506) family: 502 jobs: [Prof (507) Salary 1000.0, Engineer (508) Salary 5000.0] address:
Deolinda Lopes (509) family: 501 jobs: [Nurse (510) Salary 500.0] address:
Prof (504) Salary 1000.0
Mum (505) Salary 0.0
Prof (507) Salary 1000.0
Engineer (508) Salary 5000.0
Nurse (510) Salary 500.0
------------------------
```

Exercícios

(d) Explique como pode através da utilização de *locks* evitar que o valor fornecido pelo higherSalary não esteja desatualizado?

JPA: *Locking*

- O JPA também permite aplicar *locks* de várias formas
  - *EntityManager*: *lock(object,mode), find(...,mode), refresh(...,mode)*
  - *Queries: setLockMode(mode)*
- Estão definidos diferentes níveis de estratégias de *locking*
  - **OPTIMISTIC**: *the version number is compared and has to match before the transaction is committed.*
  - **OPTIMISTIC_FORCE_INCREMENT**: *the version number is compared and has to match before the transaction is committed; force a version number increase as well*
  - **PESSIMISTIC_READ**: *apply a database-level read lock when the lock operation is requested: roughly concurrent readers are allowed but no writer is allowed.*
  - **PESSIMISTIC_WRITE:** *apply a database-level write lock when the lock operation is requested: roughly no reader nor writer is allowed.*

Uma não solução: porque não serve?

```java
public class Person {
    @Id @GeneratedValue
    private int id;

    @Version
    private int version;
```

```java
public double higherSalary(Person p) throws ApplicationException {
    EntityManager em = emf.createEntityManager();
    double result = 0;
    try {
        em.getTransaction().begin();
        p = em.merge(p);
        em.lock(p, LockModeType.OPTIMISTIC);
        result = p.higherSalary();
        em.getTransaction().commit();
    } catch (Exception e) {
        if (em.getTransaction().isActive())
            em.getTransaction().rollback();
        throw new ApplicationException("Error calculating higher salary", e);
    } finally {
        em.close();
    }
    return result;
}
```


Uma solução ?

```java
public double higherSalary(Person p) throws ApplicationException {
    EntityManager em = emf.createEntityManager();
    double result = 0;
    try {
        em.getTransaction().begin();
        p = em.merge(p); //which is defined to cascade to jobs
        if(!p.getJobList().isEmpty())
            em.lock(p.getJobList().get(p.getJobList().size()-1), LockModeType.OPTIMISTIC);
        result = p.higherSalary();
        em.getTransaction().commit();
    } catch (Exception e) {
        if (em.getTransaction().isActive())
            em.getTransaction().rollback();
        throw new ApplicationException("Error calculating higher salary", e);
    } finally {
        em.close();
    }
    return result;
}
```

Uma solução?

```
[EL Warning]: 2018-04-05 17:09:20.669--UnitOfWork(1540875968)--Exception [EclipseLink-5006] (Eclipse Persistence Services - 2.6.4.v20160829-44060b6): c
Exception Description: The object [Engineer (608) Salary 1500.0] cannot be updated because it has changed or been deleted since it was last read.
Class> model.Job Primary Key> 608                                    org.eclipse.persistence.exceptions.OptimisticLockException
Error: Error calculating higher salary
Cause:
model.ApplicationException: Error calculating higher salary
DEPOIS
------------------------

        at model.PersonCatalog.higherSalary(PersonCatalog.java:99)
        at model.PersonHandler.higherSalary(PersonHandler.java:50)
        at main.MainConcurrentHandlers$1.run(MainConcurrentHandlers.java:36)
Caused by: javax.persistence.RollbackException: javax.persistence.OptimisticLockException: Exception [EclipseLink-5006] (Eclipse Persistence Services -
Exception Description: The object [Engineer (608) Salary 1500.0] cannot be updated because it has changed or been deleted since it was last read.
Class> model.Job Primary Key> 608
        at org.eclipse.persistence.internal.jpa.transaction.EntityTransactionImpl.commit(EntityTransactionImpl.java:157)
        at model.PersonCatalog.higherSalary(PersonCatalog.java:95)
        ... 2 more
```

- O que acontece se a *thread* que calcula o salário mais alto fizer *commit* antes da *thread* que altera o salário?
- Como podemos também garantir que a alteração do salário não é bem sucedida se entretanto os dados desse job tiverem sido utilizados para calcular o salário mais alto?

Ciências ULisboa | Informática

Uma solução?

```
public double higherSalary(Person p) throws ApplicationException {
    EntityManager em = emf.createEntityManager();
    double result = 0;
    try {
        em.getTransaction().begin();
        p = em.merge(p); //which is defined to cascade to jobs
        if(!p.getJobList().isEmpty())
            em.lock(p.getJobList().get(p.getJobList().size()-1), LockModeType.OPTIMISTIC);
        result = p.higherSalary();
        em.getTransaction().commit();
    } catch (Exception e) {
        if (em.getTransaction().isActive())
            em.getTransaction().rollback();
        throw new ApplicationException("Error calculating higher salary", e);
    } finally {
        em.close();
    }
    return result;
}
```

- E se for possível alterar o salário de qualquer emprego?
- Esta solução evita que valor calculado esteja desatualizado se concorrentemente for adicionado um *job* com valor mais alto?

Ciências ULisboa | Informática

Efeito das versões no *merge*

- Atenção que no *merge* de uma entidade o número da versão é comparado e tem que ser igual.

```
em = factory.createEntityManager();
em.getTransaction().begin();
Job j = new Job();
j.setJobDescr("CS Professor");
j.setTitle(Title.PROF);
ArrayList<Job> jl = new ArrayList<>();
jl.add(j);
p = em.merge(p);
p.setJobList(jl);
em.persist(j);
em.getTransaction().commit();
em.close();
```

```
em = factory.createEntityManager();
em.getTransaction().begin();
Job j2 = em.find(Job.class,j.getId());
j2.setSalary(2000);
em.getTransaction().commit();
em.close();

em = factory.createEntityManager();
em.getTransaction().begin();
j = em.merge(j);
j.setSalary(3000);
em.getTransaction().commit();
em.close();
```

```
Persistence Services - 2.6.4.v20160829-44060b6): org.eclipse.persistence.exceptions.OptimisticLockException
Exception Description: The object [CS Professor (1203) Salary 0.0 version: 1] cannot be merged because it has changed
or been deleted since it was last read.
Class> model.Job
        at org.eclipse.persistence.internal.jpa.EntityManagerImpl.mergeInternal(EntityManagerImpl.java:555)
        at org.eclipse.persistence.internal.jpa.EntityManagerImpl.merge(EntityManagerImpl.java:530)
        at main.SimpleMain2.main(SimpleMain2.java:85)
Caused by: Exception [EclipseLink-5010] (Eclipse Persistence Services - 2.6.4.v20160829-44060b6):
org.eclipse.persistence.exceptions.OptimisticLockException
Exception Description: The object [CS Professor (1203) Salary 0.0 version: 1] cannot be merged because it has changed
or been deleted since it was last read.
Class> model.Job
        at
org.eclipse.persistence.exceptions.OptimisticLockException.objectChangedSinceLastMerge(OptimisticLockException.java:
152)
```