PRÀCTICA Arquitectura del Software

1er Lliurament

Grup 12

QT 15-16

Shows.com

Pere Berge Sànchez
Daniel Gil Sancho
Albert Suàrez Molgó
Víctor Pérez Martos

ÍNDEX

1.- Hibernate

1.1.- Codi Java de les classes

1.1.1.- Classe Local

```
package hibernateAS;

import java.beans.Transient;
import java.util.ArrayList;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name="local")
public class Local{
	@Id
	@Column(name = "nom")
	private String nom;
	@Column(name = "adreça")
	private String adreça;
	@JoinColumn(name = "local")
	@javax.persistence.Transient
	private ArrayList<Seient> seients;


	public Local(){
		seients = new ArrayList<Seient>();
	}
	public Local(String n, String a){
		this.seients = new ArrayList<Seient>();
		this.nom = n;
		this.adreça = a;
	}
	public String getnom() {
		return nom;
	}
	public void setnom(String nom) {
		this.nom = nom;
	}
```

```
	public String getadreça() {
		return adreça;
	}
	public void setadreça(String adreça) {
		this.adreça = adreça;
	}
	public void afegirSeient(Seient seient){
		this.seients.add(seient);
	}
	public void eliminarSeient(Seient seient){
		this.seients.remove(seient);
	}
	public void setSeients(ArrayList<Seient> s){
		this.seients = s;
	}

}
```

1.1.2.- Classe Seient

```
package hibernateAS;

import java.io.Serializable;

import javax.persistence.Basic;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.IdClass;
import javax.persistence.JoinColumn;
import javax.persistence.JoinColumns;
import javax.persistence.ManyToOne;
import javax.persistence.Transient;

import org.hibernate.annotations.ForeignKey;


@Entity
@IdClass(SeientPK.class)
public class Seient implements Serializable {

	@Id
	private int fila;
	@Id
```

```
	private int columna;
	@Id
	@ForeignKey(name = "local")
	@JoinColumns({
		@JoinColumn(name = "local", referencedColumnName = "nom")
	})
	@ManyToOne
	private String local;

	public Seient(){

	}
	public Seient(int fila, int col, String local){
		this.fila = fila;
		this.columna = col;
		this.local = local;
	}
	public int getfila() {
		return fila;
	}
	public void setfila(int fila) {
		this.fila = fila;
	}
	public int getcolumna() {
		return columna;
	}
	public void setcolumna(int columna) {
		this.columna = columna;
	}
	public String getlocal(){
		return local;
	}
	public void setlocal(String local){
		this.local = local;
	}
}
```

```
package hibernateAS;

import java.io.Serializable;


public class SeientPK implements Serializable {

	private int fila;
	private int columna;
	private String local;

	public SeientPK(){

	}

}
```

1.1.3.- Classe Test (programa de prova)

```
package hibernateAS;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;
import org.hibernate.tool.hbm2ddl.SchemaExport;

public class Test {

	public static void main(String[] args){
		AnnotationConfiguration config = new
AnnotationConfiguration();
		config.addAnnotatedClass(Local.class);
		config.addAnnotatedClass(Seient.class);
		config.addAnnotatedClass(SeientPK.class);
		config.configure("hibernate.cfg.xml");

		new SchemaExport(config).create(true,true);

		SessionFactory factory = config.buildSessionFactory();
		Session session = factory.getCurrentSession();

		session.beginTransaction();

		//local A
```

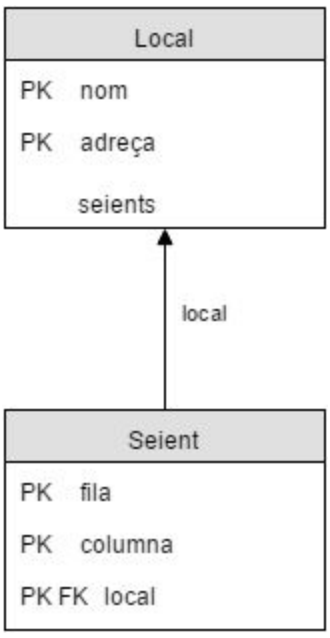
```
            Local A = new Local("El Molino", "C/Vila i Vilà, 99"); //crea
local
            Seient A11 = new Seient(1,1,A.getnom()); //crea seients
            Seient A12 = new Seient(1,2,A.getnom());
            A.afegirSeient (A11);  //afegeix seients al local A
            A.afegirSeient(A12);

            //local B
            Local B = new Local(); //crea local
            B.setnom("El Liceu");        //Modifica atributs
            B.setadreça("C/Les Rambles, 51-59");
            Seient B11 = new Seient(1,1,B.getnom()); //crea seients
            Seient B12 = new Seient(1,2,B.getnom());
            B.afegirSeient(B11);   //afegeix seients al local B
            B.afegirSeient(B12);

            session.save(A);
            session.save(B);
            session.save(A11);
            session.save(A12);
            session.getTransaction().commit();

      }
}
```

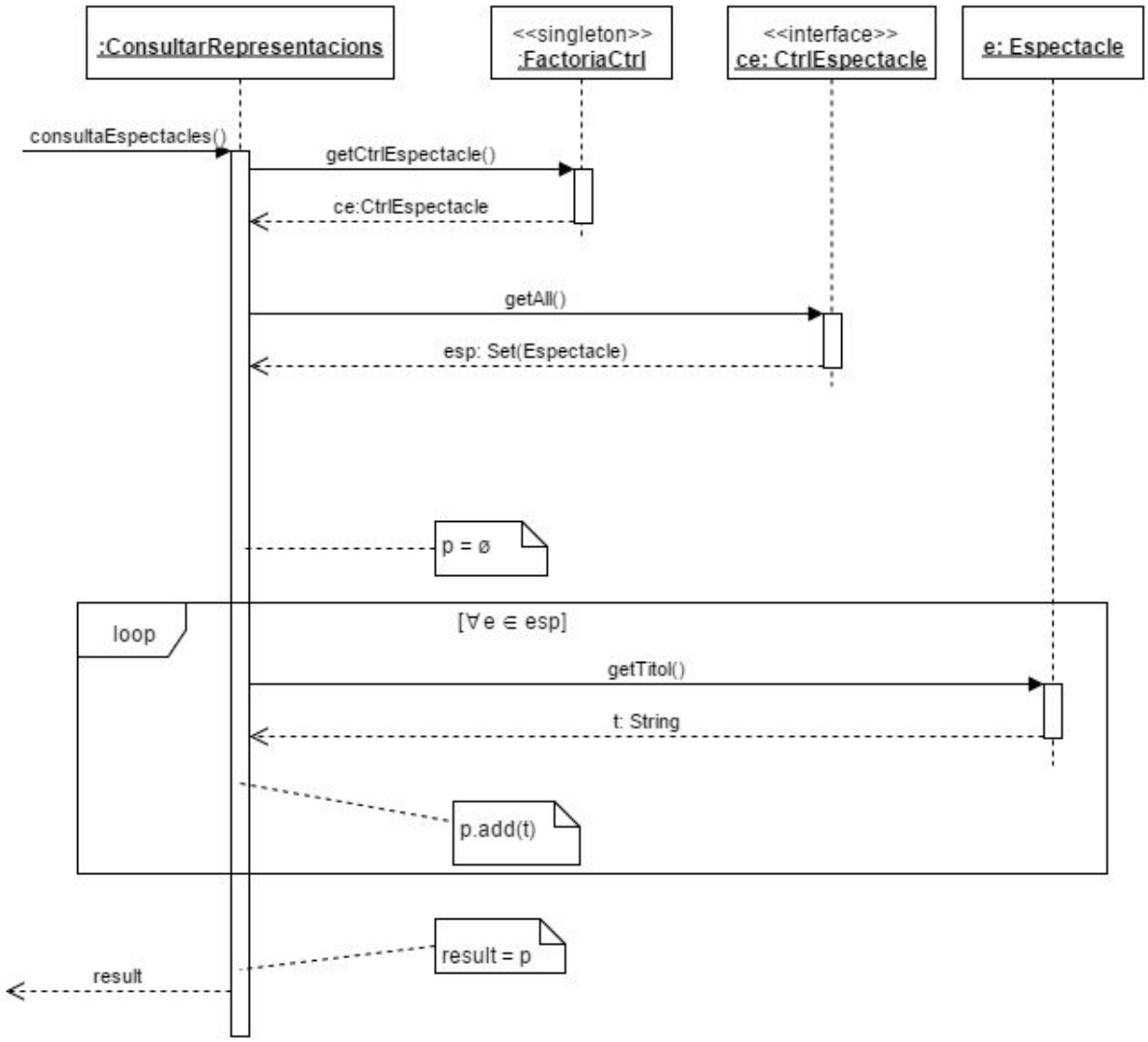
1.2.- Esquema de la BD



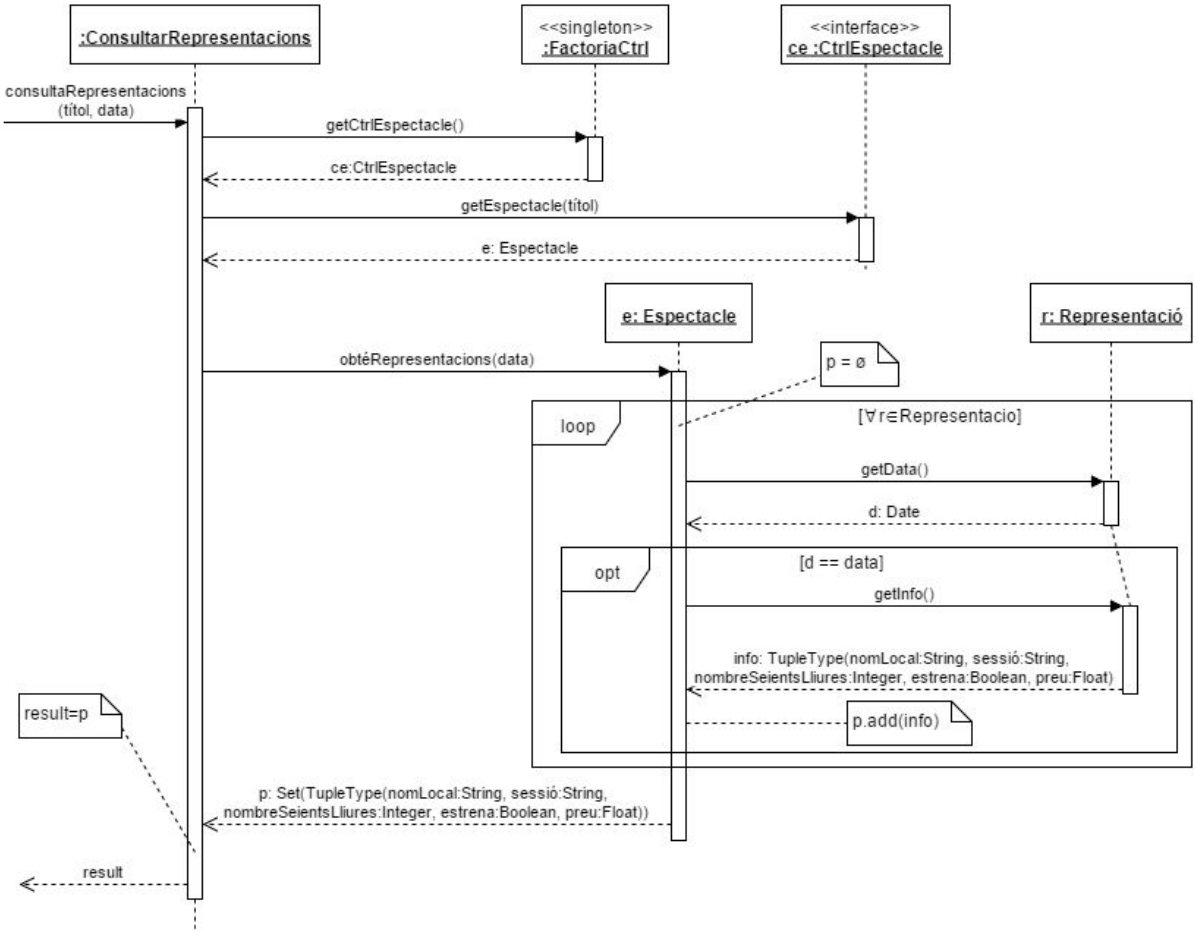
2.- Diagrames de seqüència

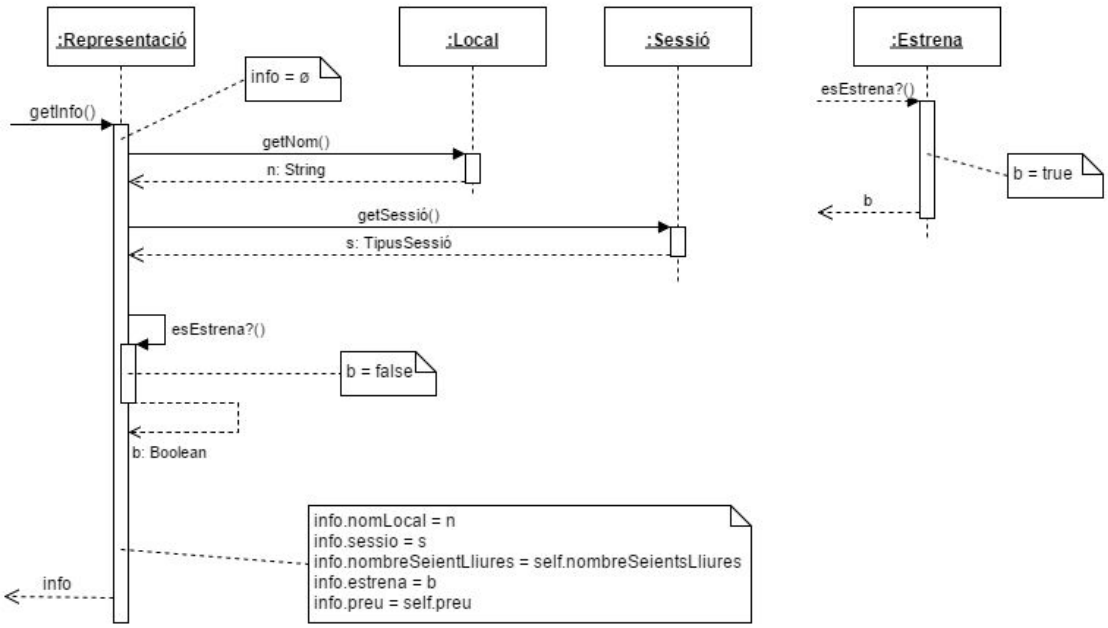
2.1.- Cas d'ús Consultar Representacions

2.1.1.- Consulta Espectacles



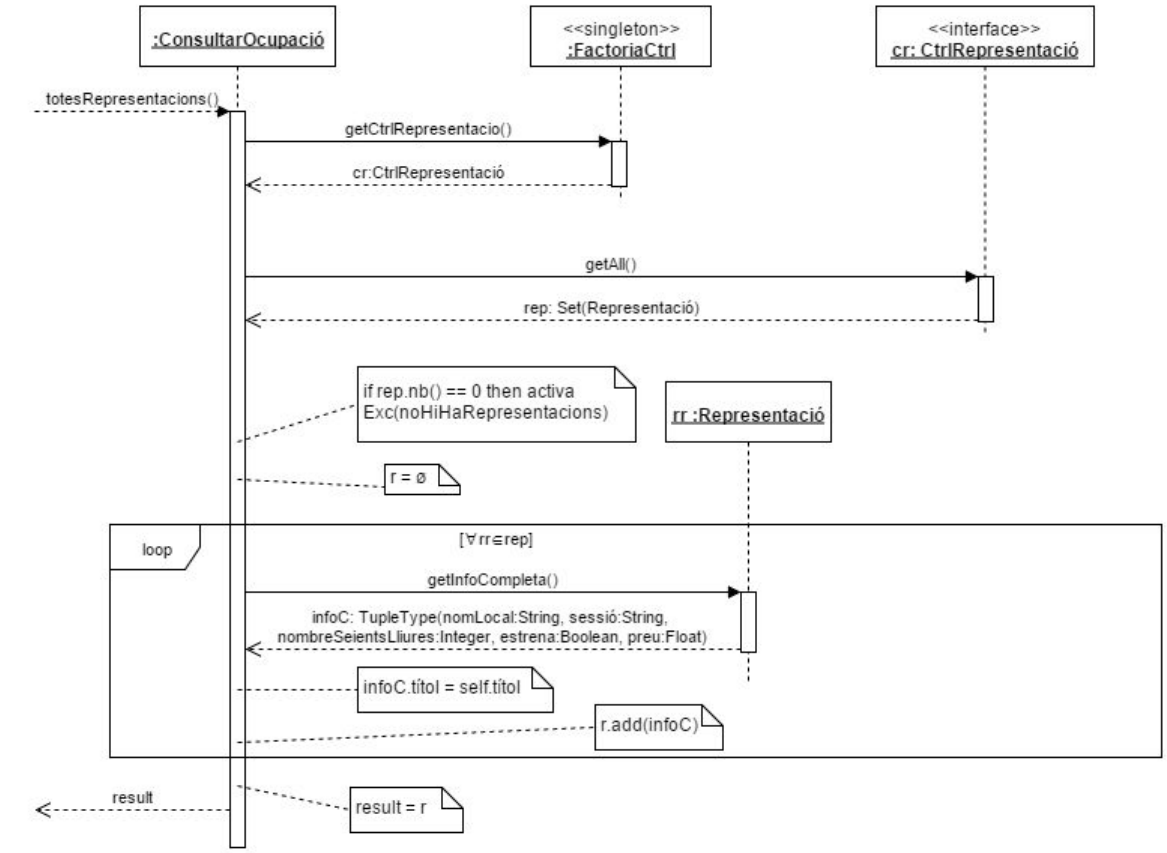
2.1.2.- Consulta Representacions

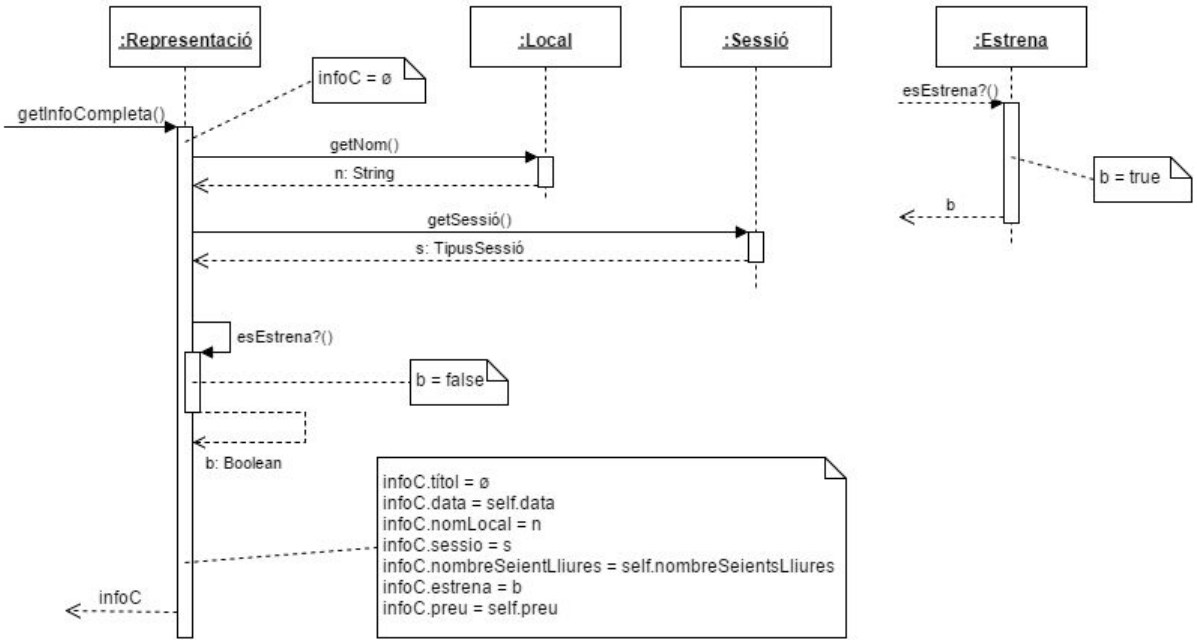




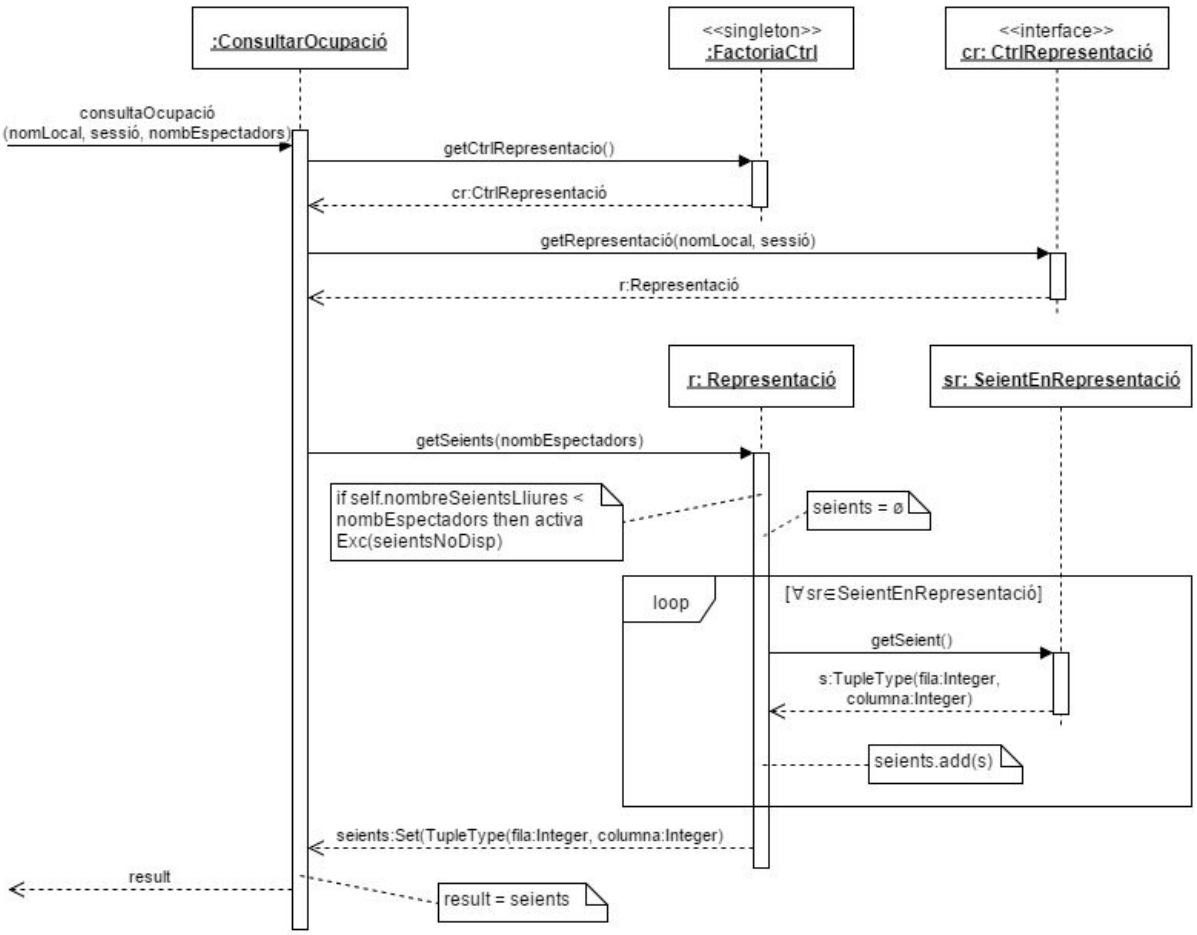
2.2.- Cas d'ús Consultar Ocupació

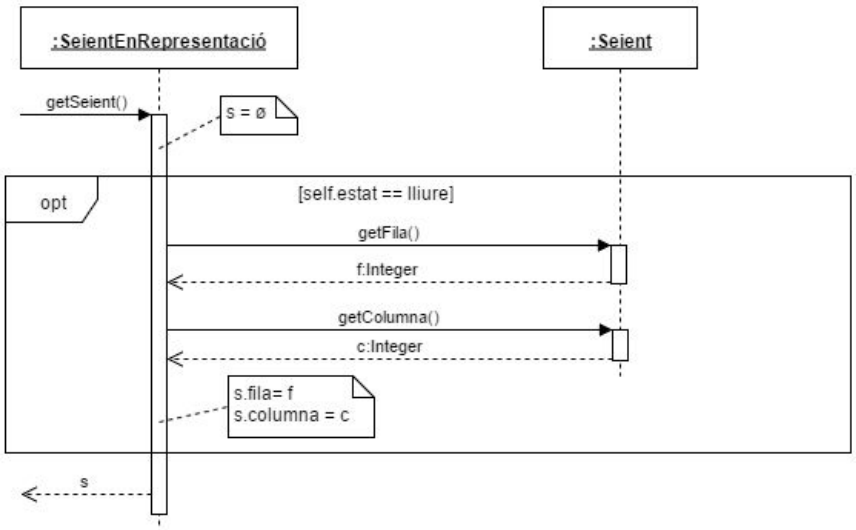
2.2.1.- Totes Representacions





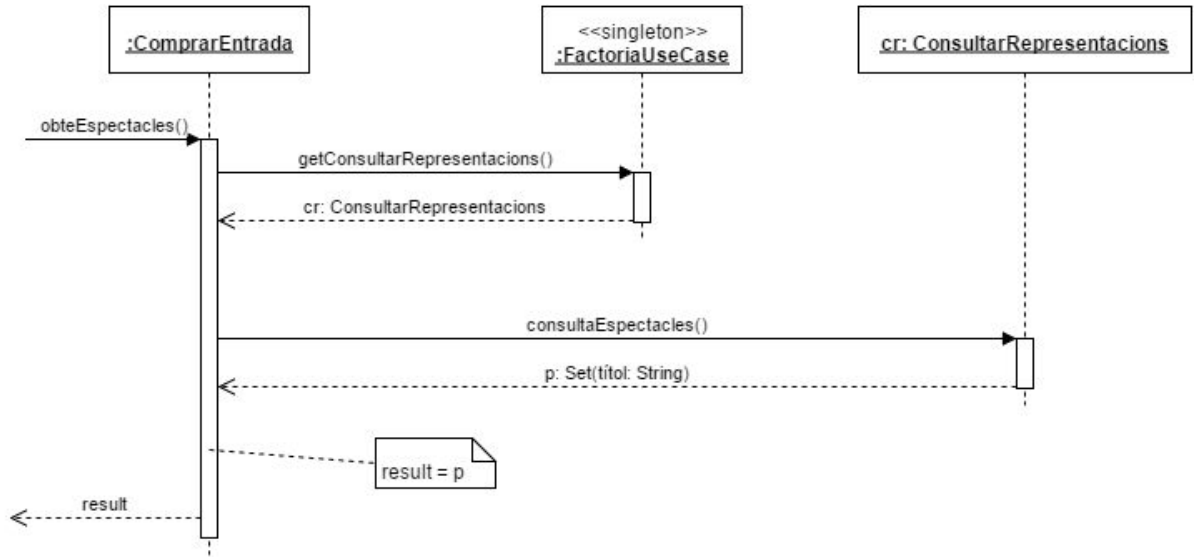
2.2.2.- Consulta Ocupació



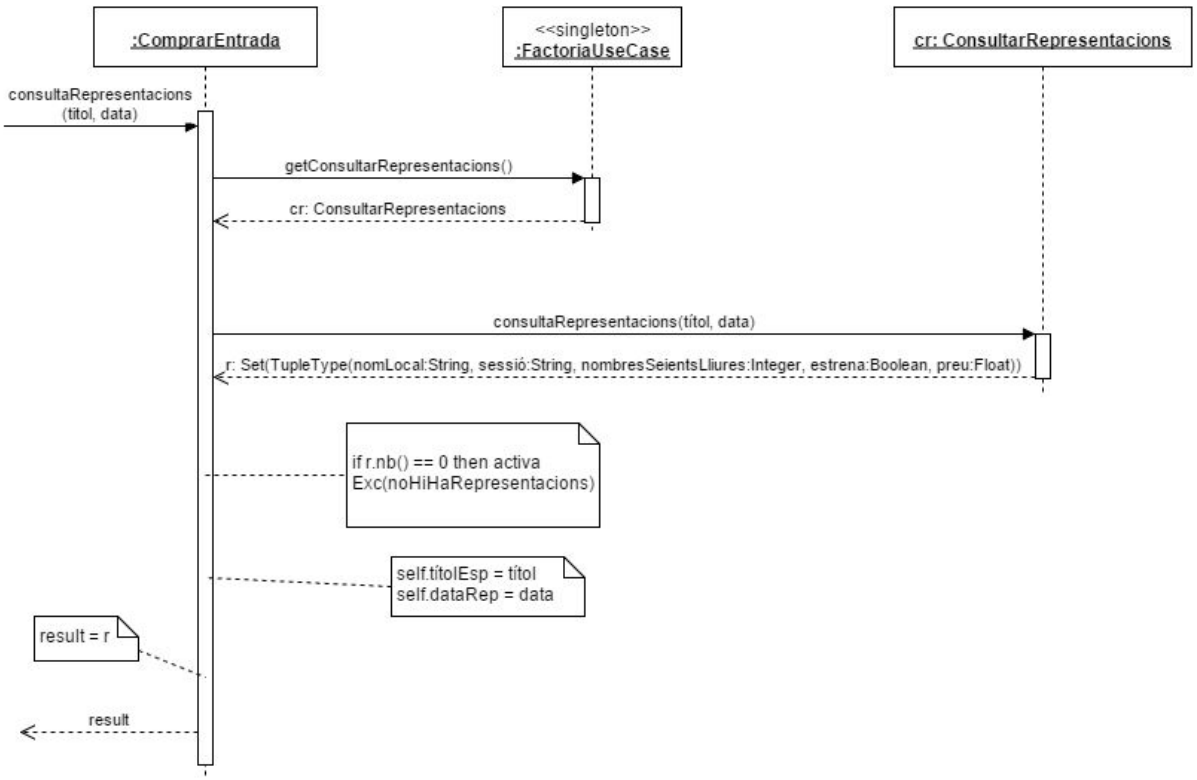


2.3.- Cas d'ús Comprar Entrada

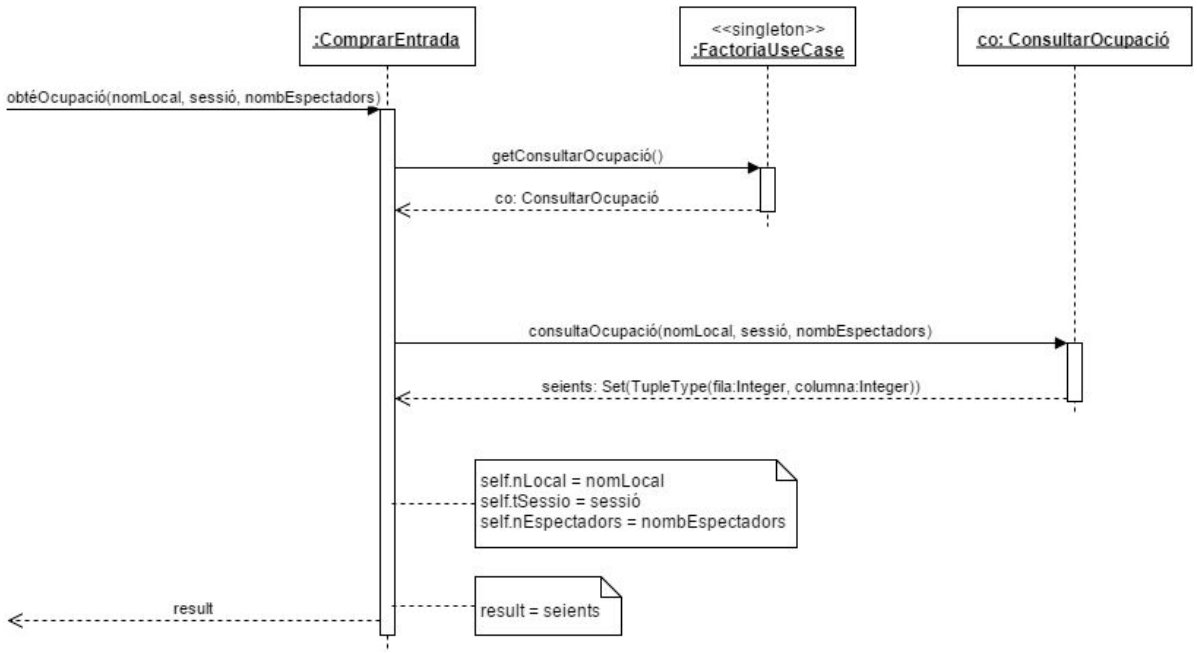
2.3.1.- Obté Espectacles



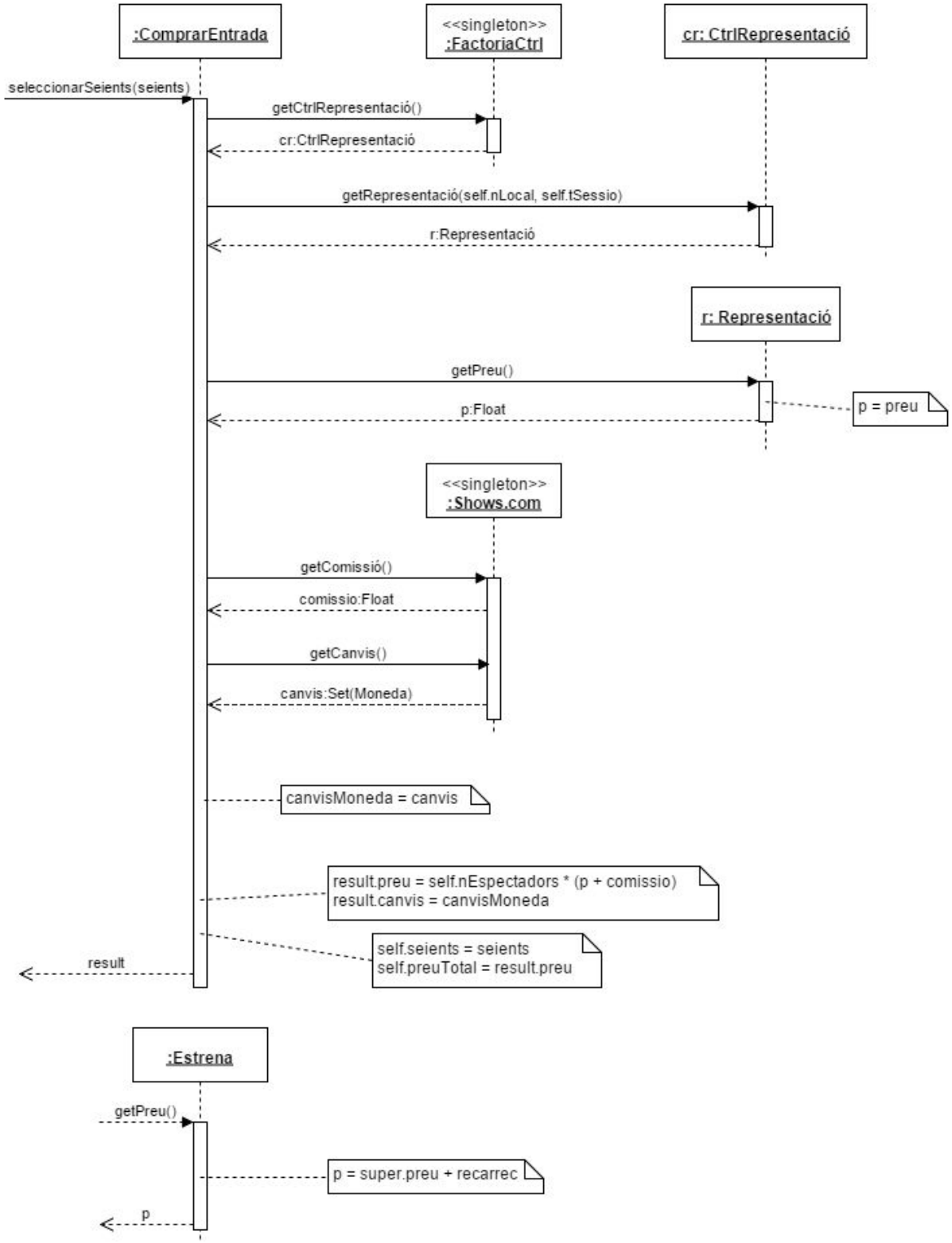
2.3.2.- Obté Representacions



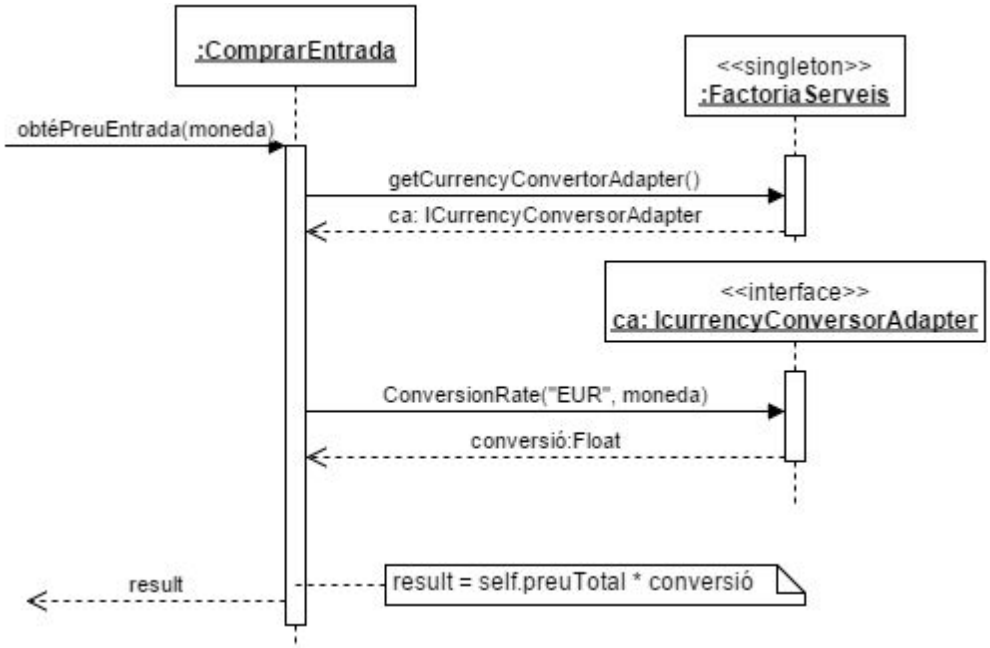
2.3.3.- Obté Ocupació



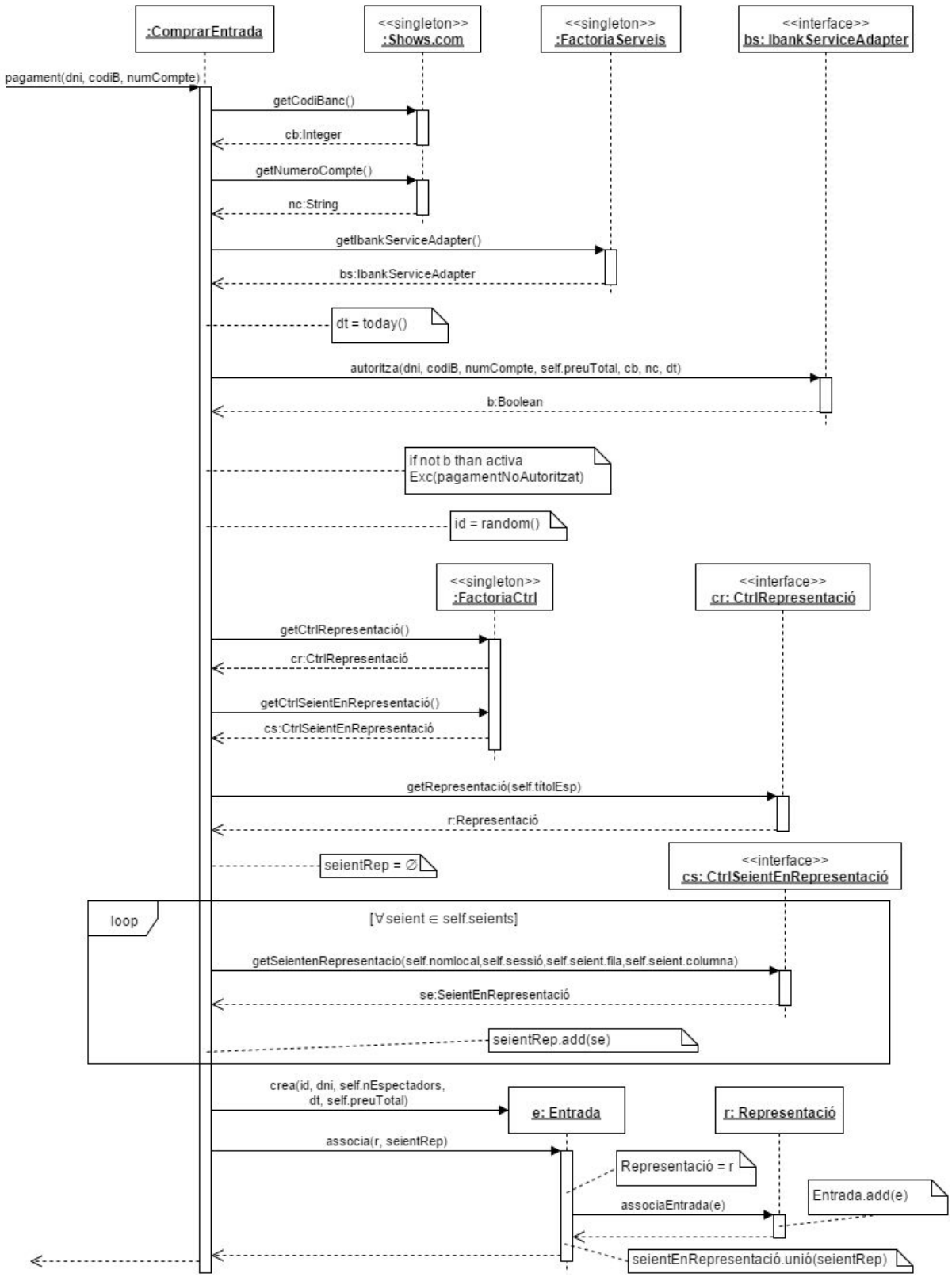
2.3.4.- Seleccionar Seients



2.3.5.- Obté Preu Moneda

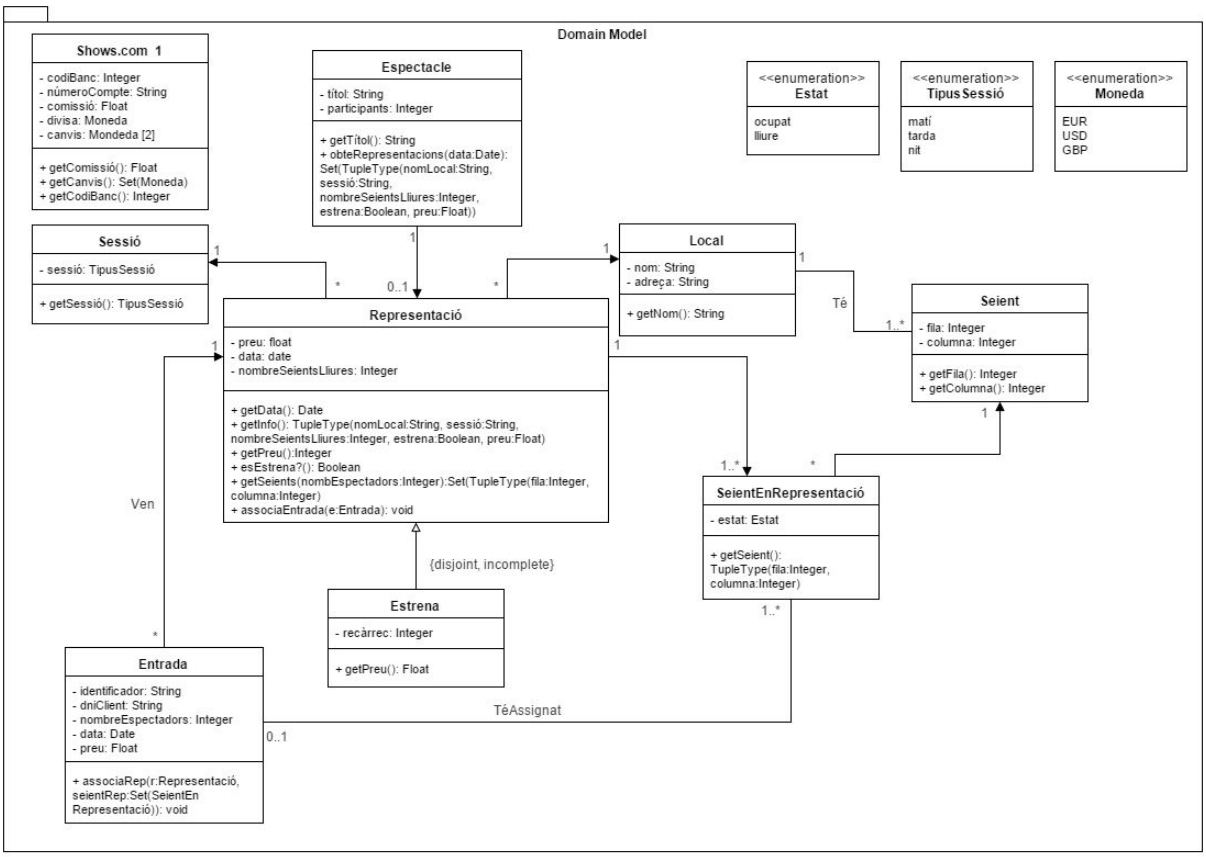


2.3.6.- Pagament



3.- Diagrama de classes de la capa de domini

3.1.- Domain Model



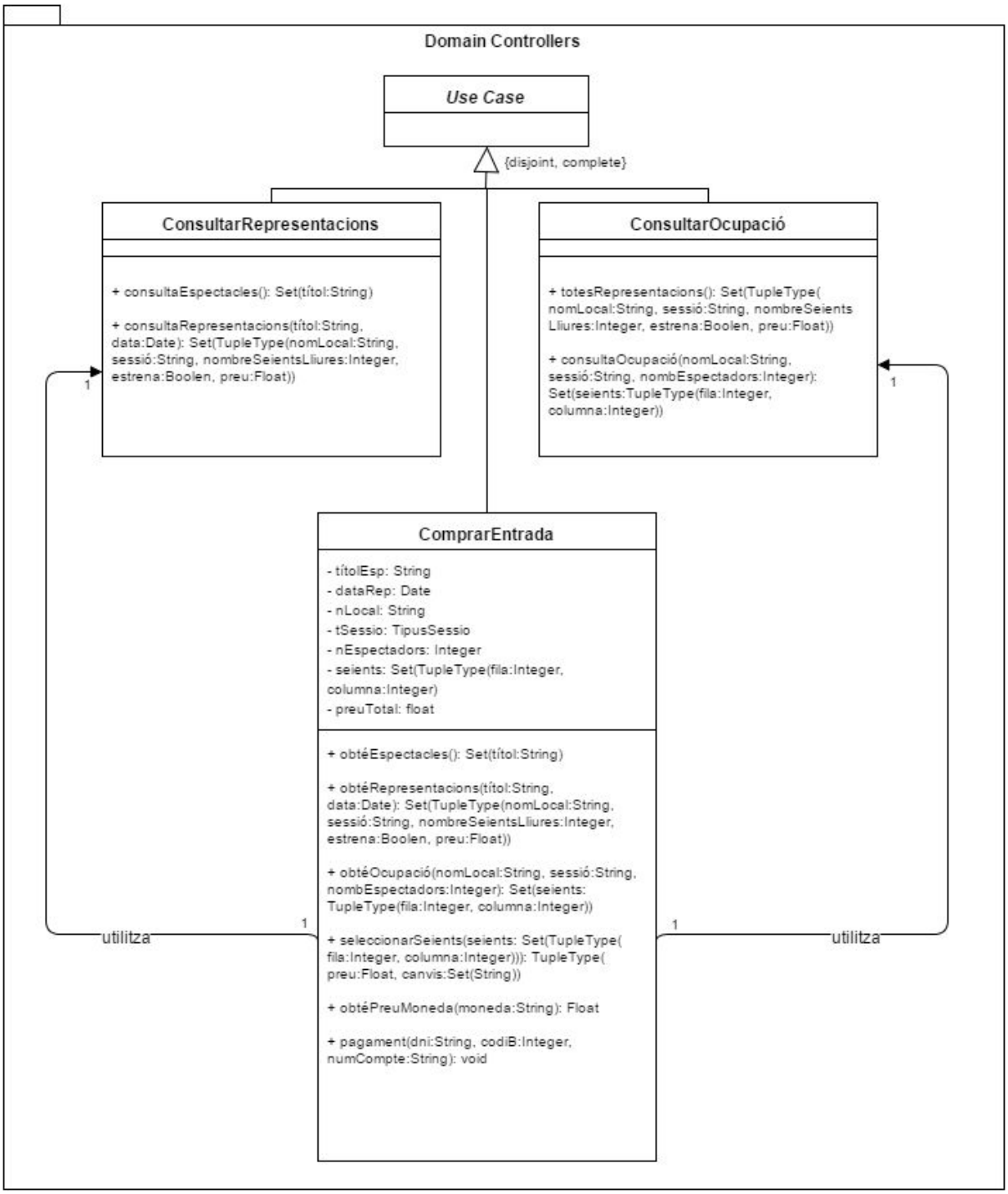
RestriccionsTextuals

RT01.- Espectacle s'identifica per títol.
RT02.- Sessió s'identifica per sessió.
RT03.- Local s'identifica per nom.
RT04.- Seient s'identifica per nom (de Local), fila i columna.
RT05.- Entrada s'identifica per identificador.
RT06.- Els seients d'una representació han de ser els seients del local on es fa la representació.
RT07.- La representació associada a una entrada i la representació associada als seients reservats per l'entrada han de coincidir.
RT08.- Els atributs participants, fila, columna, preu, comissió, codiBanc, recàrrec, nombreSeientsLliures i nombreEspectadors han de ser més grans que 0.
La data de compra de l'entrada ha de ser anterior a la data de la representació.
RT09.- L'atribut estat de SeientEnRepresentació tindrà el valor ocupat si hi ha una entrada associada. En cas contrari, tindrà el valor lliure.
RT10.- El valor de l'atribut nombreSeientsLliures serà igual al nombre de seients lliures que hi ha per aquella representació.
RT11.- El preu d'una entrada serà igual al nombre d'espectadors * (el preu de la representació + comissió Shows.com)
si la representació no és una estrena i serà el nombre d'espectadors * (el preu de la representació + comissió Shows.com +recàrrec) si és una estrena.
RT12.- La divisa de Shows.com és l'euro.
RT13.- Els canvis que proporciona Shows.com són dòlars i lliures. Altres restriccions no rellevants pel nostre disseny...

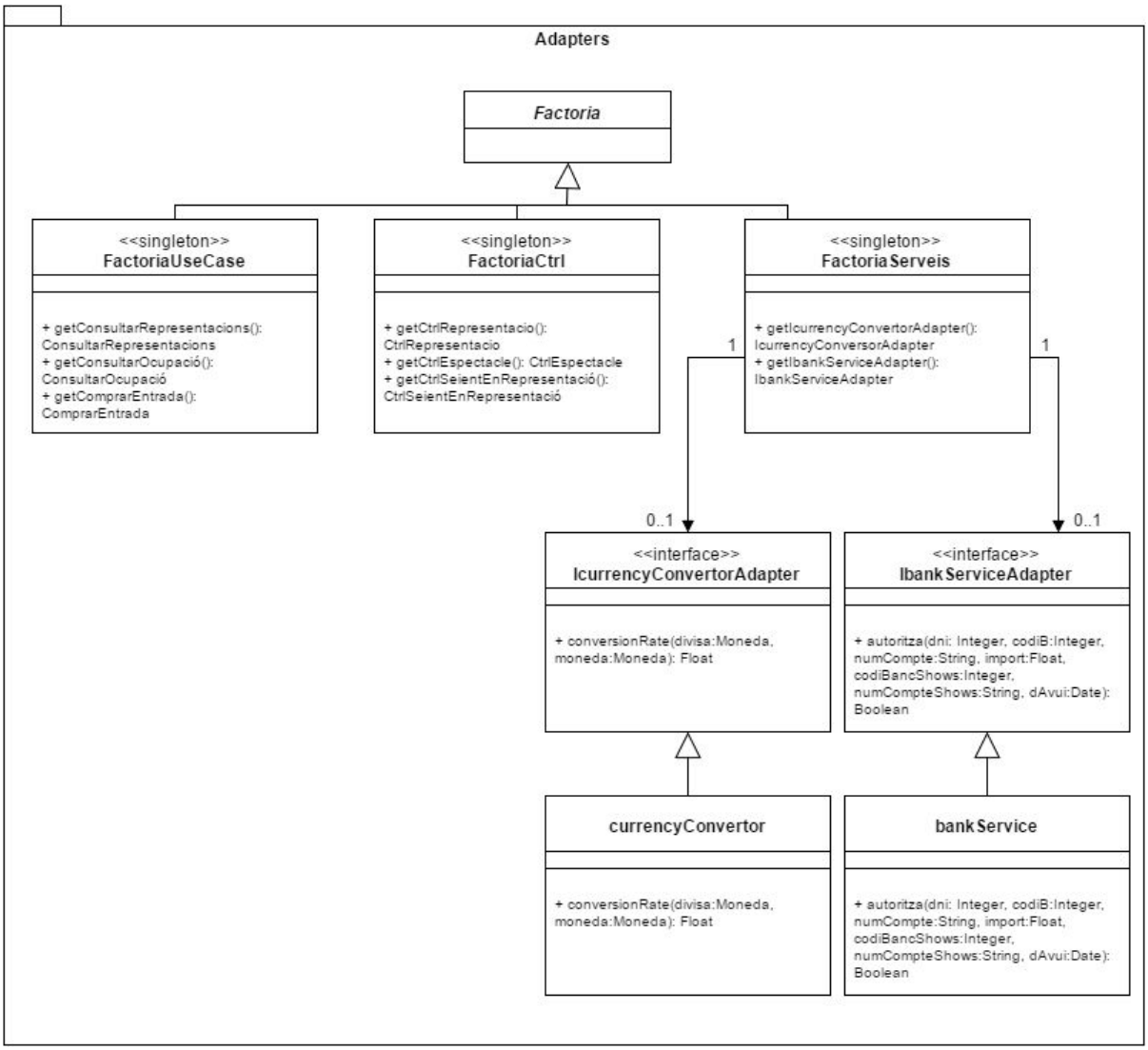
Restriccions Afegides

RT14.- Per a cada Representació, hi ha només una Sessió i un Local.
RT15.- Per a cada SeientEnRepresentació, només hi ha una Representació i un Seient.

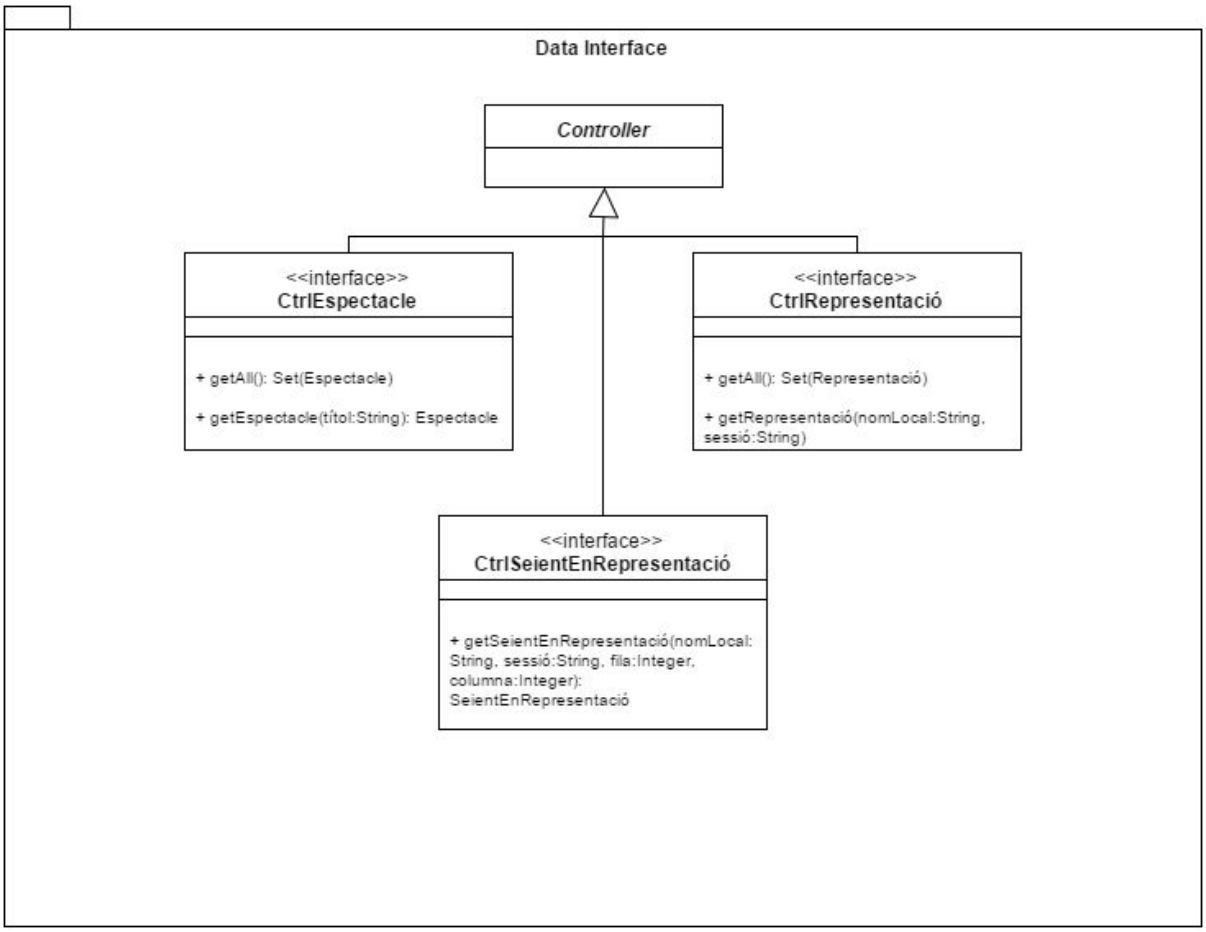
3.2.- Domain Controllers



3.3.- Adapters


Adapters
Factoria
<<singleton>>
FactoriaUseCase
+ getConsultarRepresentacions(): ConsultarRepresentacions
+ getConsultarOcupació(): ConsultarOcupació
+ getComprarEntrada(): ComprarEntrada
<<singleton>>
FactoriaCtrl
+ getCtrlRepresentacio(): CtrlRepresentacio
+ getCtrlEspectacle(): CtrlEspectacle
+ getCtrlSeientEnRepresentació(): CtrlSeientEnRepresentació
<<singleton>>
FactoriaServeis
+ getIcurrencyConvertorAdapter(): IcurrencyConversorAdapter
+ getIbankServiceAdapter(): IbankServiceAdapter
1
1
0..1
0..1
<<interface>>
IcurrencyConvertorAdapter
+ conversionRate(divisa:Moneda, moneda:Moneda): Float
<<interface>>
IbankServiceAdapter
+ autoritza(dni: Integer, codiB:Integer, numCompte:String, import:Float, codiBancShows:Integer, numCompteShows:String, dAvui:Date): Boolean
currencyConvertor
+ conversionRate(divisa:Moneda, moneda:Moneda): Float
bankService
+ autoritza(dni: Integer, codiB:Integer, numCompte:String, import:Float, codiBancShows:Integer, numCompteShows:String, dAvui:Date): Boolean

3.4.- Data Interface



4.- Justificació

Hem aplicat els següents patrons de disseny:

Patró Adaptador: Patró utilitzat per resoldre els problemes d'incompatibilitat amb el servei extern del canvi de moneda i el servei de pagament, hem utilitzat un adaptador per a cada un. A més a més, hem afegit una factoria que ens servirà per si en un futur incorporem més serveis externs al sistema.

Patró Factoria: Patró utilitzat per emmagatzemar i crear les classes independents hem utilitzat 3 classes *Factoria* (*FactoriaCtrl*, *FactoriaServei*, *FactoriaUseCase*), així li donem tota la responsabilitat de crear aquestes classes i les agrupem per tipus sense tenir en compte quines classes estan relacionades amb elles.

Patró Singleton: Patró utilitzat per indicar que algunes classes només poden tenir una instància. Segons l'enunciat del projecte la classe *Shows.com* té aquest requisit, així que se li aplica aquest patró. Addicionalment, apliquem aquest patró sobre les classes *Factoria* ja que només ens interessa disposar d'una instància de cadascuna d'aquestes classes.

Patró DataController: Patró utilitzat per accedir a la base de dades i obtenir els objectes que necessitem de cada classe hem utilitzat unes classes que són controladors de les dades i s'encarreguen de mantenir la persistència de dades. Aquestes concretament són *CtrlEspectacle*, *CtrlRepresentació* i *CtrlSeientEnRepresentació*.