11ª Emisión

DIPLOMADO

Desarrollo de Sistemas con Tecnología Java

Módulo 11

Persistencia con Jakarta

Dr. Omar Mendoza González

omarmendoza564@aragon.unam.mx

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Dirección General de Cómputo y de Tecnologías de información y Comunicación
Dirección de Docencia en TIC

DGTIC

50 AÑOS Educación Continua 1971 - 2021

Objetivo

- El participante integrará a sus aplicaciones de tipo empresarial los mecanismos apropiados de persistencia para almacenar la información en una base de datos.

Clases embebidas en las entidades

- Al mapear colecciones, hay tres tipos de objetos que se pueden almacenar en colecciones mapeadas.
  - Colecciones de entidades
  - Embeddables
  - Tipos básicos

- Las colecciones de tipos básicos y Embeddables son ***no relaciones***; son simplemente ***colecciones de elementos***

- Las relaciones definen asociaciones entre entidades independientes, mientras que las colecciones de elementos ***contienen objetos que dependen de la entidad de referencia*** y solo se pueden recuperar a través de la entidad que los contiene.

Embedded Objects

- Un ***Embedded Object*** es aquel que depende de una entidad para su identidad.
- No tiene identidad propia, sino que es parte del estado de la entidad que ha sido definido y almacenado en un objeto Java separado que cuelga de la entidad.
- En Java, los ***Embedded Object*** parecen similares a las relaciones en el sentido de que son referenciados por una entidad y aparecen como una asociación.
- En la BD, sin embargo, el estado del ***Embedded Object*** se almacena con el resto del estado de la entidad en la fila, sin distinción entre el estado de la entidad Java y el de su ***Embedded Object*** .

Embedded Objects

- Un tipo *Embedded* se marca como tal agregando la anotación ***@Embeddable*** a la definición de clase.

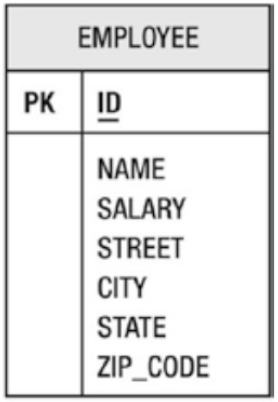
- Esta anotación sirve para distinguir la clase de otros tipos regulares de Java.

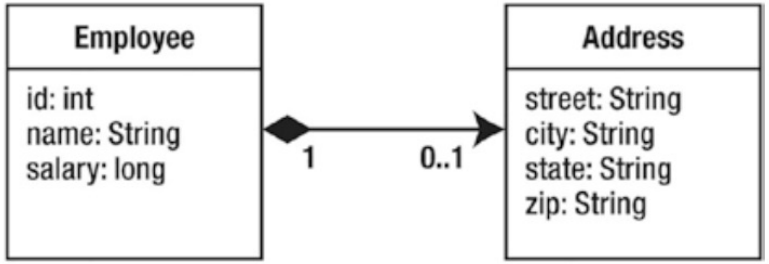
- Una vez que una clase ha sido designada como *Embeddable*, sus campos y propiedades serán persistentes como parte de una entidad.

- Se podría definir el tipo de acceso del *embeddable object* para que se acceda a él de la misma manera, independientemente de la entidad en la que esté embebido.

Embedded Objects

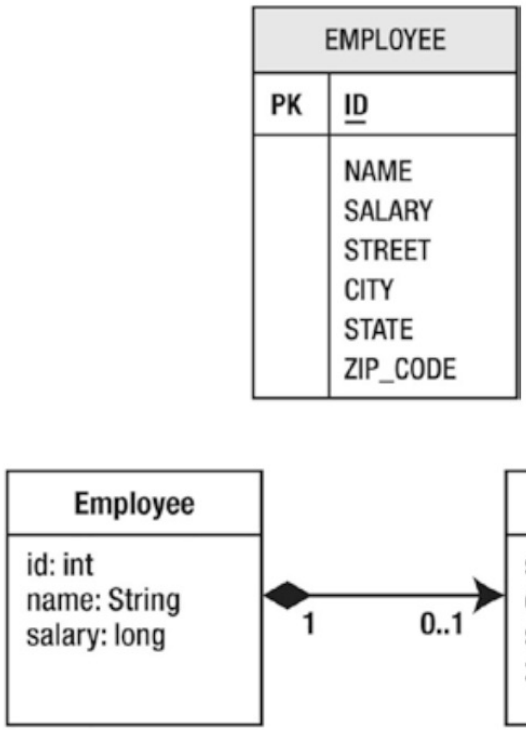
```
@Entity
public class Employee {
   @Id private long id;
   private String name;
   private long salary;
   @Embedded private Address address;
   // ...
}
```





Embedded Objects

```
@Embeddable @Access(AccessType.FIELD)
public class Address {
   private String street;
   private String city;
   private String state;
   @Column(name="ZIP_CODE")
   private String zip;
   // ...
}
```



Embedded Objects

- La anotación ***@AttributeOverride*** permite sobreescribir el nombre del atributo que sera mapeado.

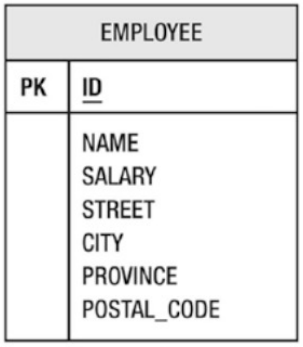
- Anotamos el campo o la propiedad incrustados en la entidad y especificamos en el elemento de nombre el campo o la propiedad en el objeto incrustado que estamos anulando.

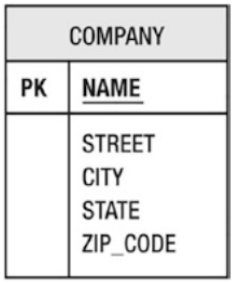
- Puede usarse la anotación plural ***@AttributeOverrides*** y anidar varias anotaciones ***@AttributeOverride*** dentro de ella.

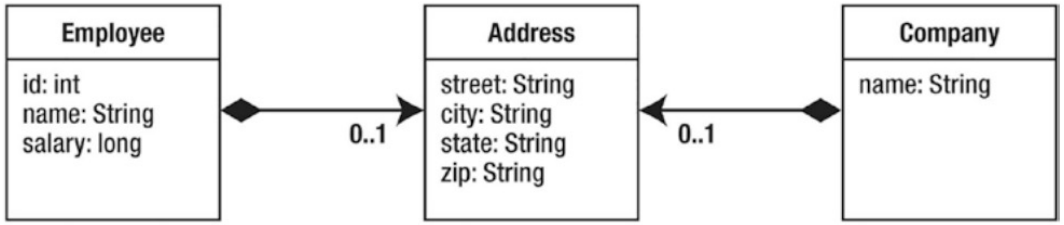
Embedded Objects

```
@Entity
public class Employee {
   @Id private long id;
   private String name;
   private long salary;
   @Embedded
   @AttributeOverride(name="state", column=@Column(name="PROVINCE")),
   @AttributeOverride(name="zip", column=@Column(name="POSTAL_CODE"))
   private Address address;
// ... }

@Entity
public class Company {
   @Id private String name;
   @Embedded
   private Address address;
   // ...
}
```







Collection Mapping

- Una colección de elementos se indica mediante la anotación *@ElementCollection*

```
@Embeddable
public class VacationEntry {
    @Temporal(TemporalType.DATE)
    private Calendar startDate;
    @Column(name="DAYS")
    private int daysTaken;
    // ...
}
```

Collection Mapping

- Pueden usarse diferentes tipos de colecciones para conservar los objetos.
  - *List*
  - *Set*
  - *Map*
- Las colecciones de elementos requieren una tabla separada llamada ***tabla de colección***.

- Cada tabla de colección debe tener una ***join column*** que haga referencia a la tabla de entidad contenedora.

- Se utilizan columnas adicionales en la tabla de colección para mapear los atributos del elemento ***embeddable***, o el estado del elemento básico si el elemento es de un tipo básico.

Collection Mapping

```
@Entity
public class Employee {
   @Id private int id;
   private String name;
   private long salary;
   // ...
   @ElementCollection(targetClass=VacationEntry.class)
   private Collection vacationBookings;
   @ElementCollection
   private Set<String> nickNames;
// ... }
```

Collection Mapping

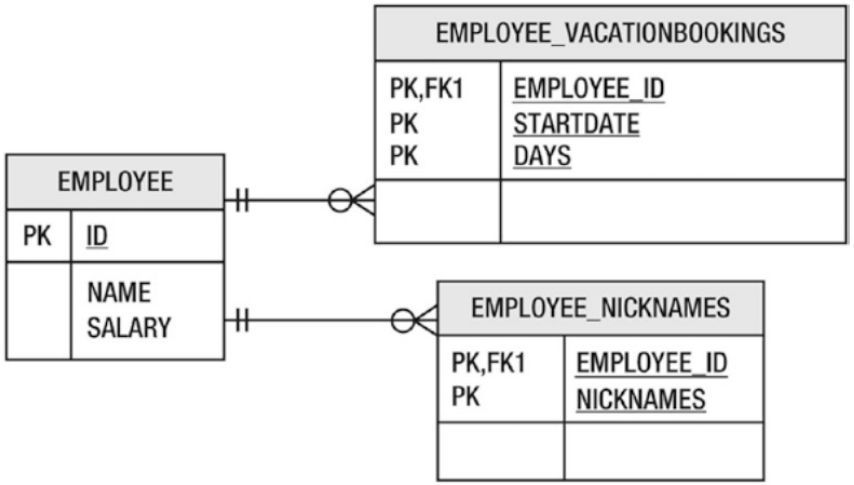
- Puede especificarse una tabla de colección con la anotación ***@CollectionTable***, que permite designar el nombre de la tabla, así como la join column.

- El nombre de la tabla será el nombre predeterminado de la entidad de referencia, junto con un guión bajo y el nombre del atributo de la entidad que contiene la colección de elementos.

- El valor predeterminado de la join column es, de manera similar, el nombre de la entidad de referencia, con un guión bajo y el nombre de la columna de pk de la tabla de entidades.

Collection Mapping

```
@Entity
public class Employee {
   @Id private int id;
   private String name;
   private long salary;
   // ...
   @ElementCollection(targetClass=VacationEntry.class)
   @CollectionTable(
      name="VACATION", joinColumns=@JoinColumn(name="EMP_ID"))
    @AttributeOverride(
       name="daysTaken", column=@Column(name="DAYS_ABS"))
    private Collection vacationBookings;
    @ElementCollection
    @Column(name="NICKNAME")
    private Set<String> nickNames;
// ... }
```



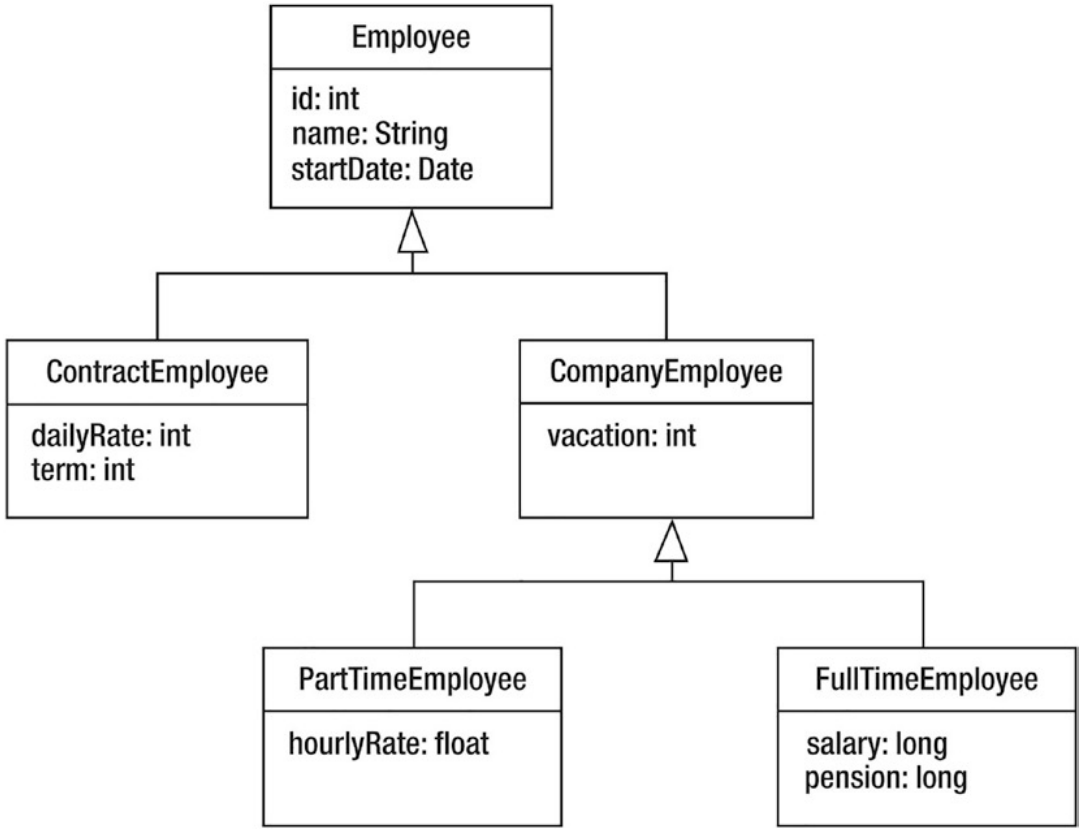
Herencia

- La herencia es una de las características fundamentales de la POO, aunque en la actualidad se le tenga cierta aversión debido a su empleo abusivo.

- Es una técnica fundamental en la definición de modelos de datos y en la aplicación de algunos patrones de diseño.

- Cuando se instancia una entidad de subclase, tiene su propia versión o copia tanto de su estado definido localmente como de su estado heredado, todo lo cual es persistente

Herencia


Employee
id: int
name: String
startDate: Date
ContractEmployee
dailyRate: int
term: int
CompanyEmployee
vacation: int
PartTimeEmployee
hourlyRate: float
FullTimeEmployee
salary: long
pension: long

Herencia

- Dado que el modelo relacional no contempla el mecanismo de herencia, JPA define cuatro estrategias fáciles de utilizar para vincular jerarquías de entidades con tablas.

  – Ignorar la jerarquía (**Mapped Superclass**)
  – Una tabla para todo (**Single Table**)
  – Una tabla por clase (**Table per Class**)
  – Tablas unidas (**Joined Table**)

Discriminator Value

- EL valor del *Discriminator Value* debe ser del mismo tipo que el especificado o predeterminado como elemento ***discriminatorType*** en la anotación ***@DiscriminatorColumn***.

- Si no se especifica la anotación ***@DiscriminatorValue***, el proveedor utilizará una forma específica para obtener el valor.
  - *DiscriminatorType*.***INTEGER***
  - *DiscriminatorType*.***STRING***

Discriminator Value

- Cada fila de la tabla tendrá un valor denominado ***Discriminator Value***, o ***indicador de clase***, para indicar el tipo de entidad que se almacena en esa fila.

- Cada entidad concreta en la jerarquía de herencia, ***Discriminator Value*** específico.

- Se una una anotación ***@DiscriminatorValue*** en cada clase de entidad concreta. El valor de cadena en la anotación especifica el valor discriminador que se asignará a las instancias de la clase cuando se inserten en la base de datos.

Mapped Superclass

```
@Entity
public class Employee {
   @Id private int id;
   private String name;
   @Temporal(TemporalType.DATE)
   @Column(name="S_DATE")
private Date startDate;
// ... }

@Entity
public class ContractEmployee extends Employee {
   @Column(name="D_RATE")
   private int dailyRate;
   private int term;
   // ...
}
```

```
@MappedSuperclass
public abstract class CompanyEmployee extends Employee {
   private int vacation;
// ... }
@Entity
public class FullTimeEmployee extends CompanyEmployee {
   private long salary;
   private long pension;
   // ...
}
@Entity
public class PartTimeEmployee extends CompanyEmployee {
   @Column(name="H_RATE")
   private float hourlyRate;
   // ...
}
```

Single Table

```
@Entity
@Table(name="EMP")
@Inheritance
@DiscriminatorColumn(name="EMP_TYPE")
public abstract class Employee { ... }
@Entity
public class ContractEmployee extends Employee { ... }
@MappedSuperclass
public abstract class CompanyEmployee extends Employee { ... }
@Entity
@DiscriminatorValue("FTEmp")
public class FullTimeEmployee extends CompanyEmployee { ... }
@Entity(name="PTEmp")
public class PartTimeEmployee extends CompanyEmployee { ... }
```

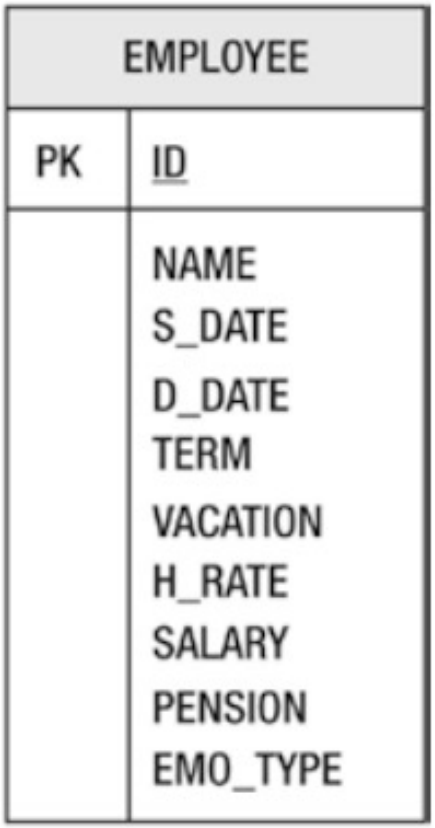
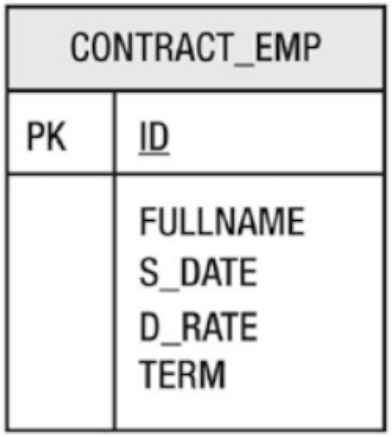
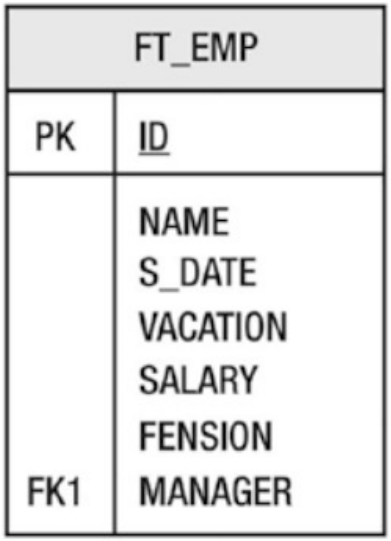


Table per Class





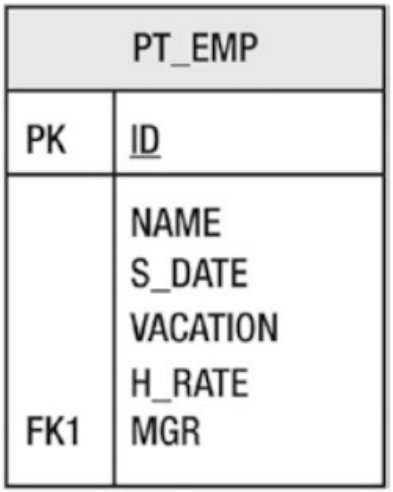
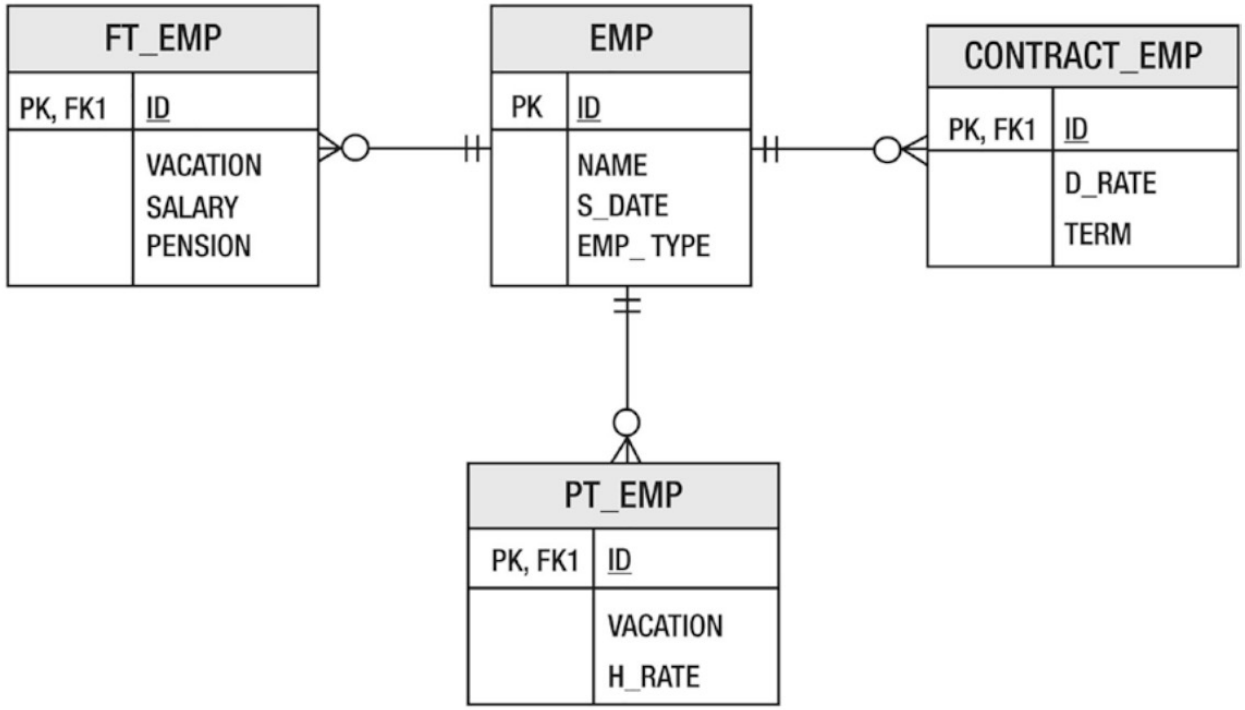


Table per Class

```
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public abstract class Employee {
   @Id private int id;
   private String name;
   @Temporal(TemporalType.DATE)
   @Column(name="S_DATE")
   private Date startDate;
   // ...
}
@Entity
@Table(name="CONTRACT_EMP")
@AttributeOverride(name="name", column=@Column(name="FULLNAME"))
@AttributeOverride(name="startDate", column=@Column(name="SDATE"))
public class ContractEmployee extends Employee {
   @Column(name="D_RATE")
   private int dailyRate;
   private int term;
   // ...
}
```

```
@MappedSuperclass
public abstract class CompanyEmployee extends Employee {
   private int vacation;
   @ManyToOne
   private Employee manager;
   // ...
}
@Entity @Table(name="FT_EMP")
public class FullTimeEmployee extends CompanyEmployee {
   private long salary;
   @Column(name="PENSION")
 private long pensionContribution;
// ... }
@Entity
@Table(name="PT_EMP")
@AssociationOverride(name="manager",
                joinColumns=@JoinColumn(name="MGR"))
public class PartTimeEmployee extends CompanyEmployee {
   @Column(name="H_RATE")
   private float hourlyRate;
   // ...
}
```

Joined Table



Joined Table

```
@Entity
@Table(name="EMP")
@Inheritance(strategy=InheritanceType.JOINED)
@DiscriminatorColumn(name="EMP_TYPE",

discriminatorType=DiscriminatorType.INTEGER)
public abstract class Employee { ... }
@Entity
@Table(name="CONTRACT_EMP")
@DiscriminatorValue("1")
public class ContractEmployee extends Employee { ...
}
```

```
@MappedSuperclass
public abstract class CompanyEmployee extends
Employee { ... }
@Entity
@Table(name="FT_EMP")
@DiscriminatorValue("2")
public class FullTimeEmployee extends
CompanyEmployee { ... }
@Entity
@Table(name="PT_EMP")
@DiscriminatorValue("3")
public class PartTimeEmployee extends
CompanyEmployee { ... }
```

Contacto

Dr. Omar Mendoza González

omarmendoza564@aragon.unam.mx