

Mappatura delle classi persistenti entities vs. value types

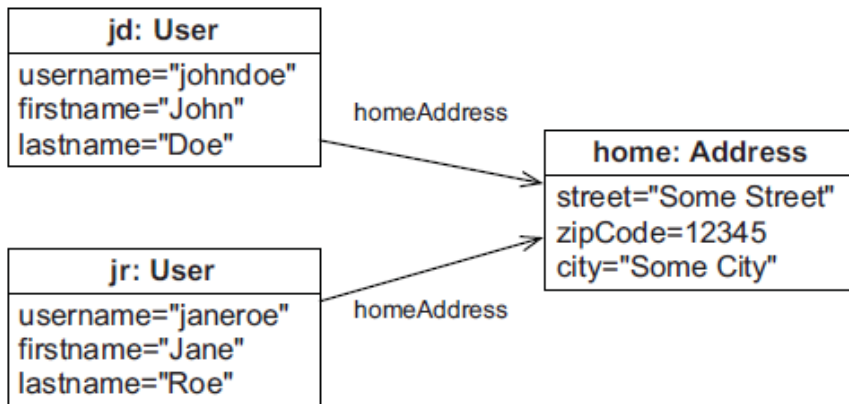
Per gli ORM le classi non sono tutte uguali!

Per Java tutte le classi da rendere persistenti sono uguali, Hibernate distingue tra:

entities e value type

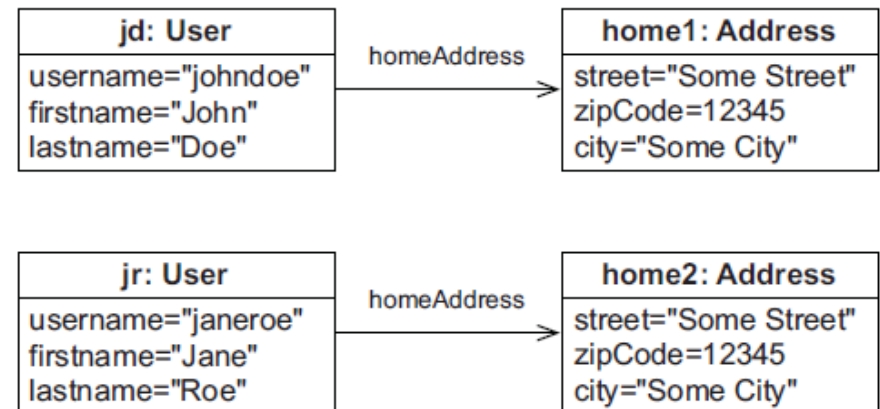
(in fase di analisi dobbiamo decidere in quale dei due casi ricadono le nostre classi)

Mappatura delle classi persistenti entities vs. value types



Entity

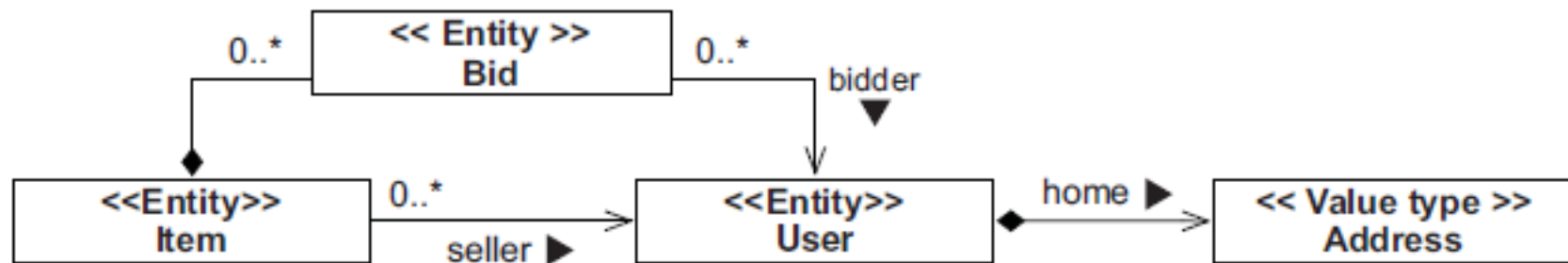
- Ha un id di persistenza
- È referenziato da altre classi
- Ciclo di vita autonomo (di solito..)



Value type (embeddable values)

- Non ha un suo id di persistenza, 'appartiene' ad un'altra classe
- Non può essere referenziato da istanze diverse
- Ciclo di vita non autonomo

Mappatura delle classi persistenti entities vs. value types



Creare i POJO facendo attenzione a:

- Riferimenti condivisi
- Dipendenze nei cicli di vita
- Identità

Mappatura delle classi persistenti (entity)

```
@Entity
public class Item {

    @Id
    @GeneratedValue(generator = "ID_GENERATOR")
    protected Long id;

    public Long getId() {
        return id;
    }
}
```

← Optional but useful

La più semplice classe entity possibile

- @Id è obbligatorio, getId() è opzionale;
- Hibernate userà gli attributi della classe per accedere alle property (@Id scritto sulla property e non sul getter)

per default altre eventuali property saranno rese persistenti

Mappatura delle classi persistenti (entity)

`javax.persistence.GenerationType`

- `GenerationType.AUTO` (default)
- `GenerationType.SEQUENCE`: utilizza (crea) sequence `HIBERNATE_SEQUENCE` chiamata prima di ogni `INSERT`
- `GenerationType.IDENTITY`: utilizza (crea) colonna con chiave primaria auto-incrementante
- `GenerationType.TABLE`: crea tabella (`HIBERNATE_SEQUENCES`) che memorizza l'id successivo di ogni entity class (`SEQUENCE_NAME`, `SEQUENCE_NEXT_HI_VALUE`)

Mappatura delle classi persistenti (entity)

Hibernate id strategy

- native
- sequence
- sequence-identity
- enhanced-sequence: usa sequence (HIBERNATE_SEQUENCE) se supportata dal db, altrimenti la emula con tabella, chiama la sequence prima degli INSERT
- seqhilo
- hilo
- enhanced-table...

Mappatura delle classi persistenti (entity)

Hibernate id strategy

Nella maggior parte dei casi la scelta migliore è enhanced-sequence, utilizzata con id-generator in un package-info.java file:

```
@org.hibernate.annotations.GenericGenerator(  
    name = "ID_GENERATOR",  
    strategy = "enhanced-sequence",           ← ① enhanced-sequence strategy  
    parameters = {  
        @org.hibernate.annotations.Parameter(  
            name = "sequence_name",           ← ② sequence_name  
            value = "JPWH_SEQUENCE"  
        ),  
        @org.hibernate.annotations.Parameter(  
            name = "initial_value",           ← ③ initial_value  
            value = "1000"  
        )  
    }  
)  
))
```

Mappatura delle classi persistenti (entity)

Per default tutti gli attributi di una classe (sia entity che embeddable) sono resi persistenti, ma si può fare l'overriding di questo comportamento:

- `@Basic(optional = false)`
- `@Column(nullable = false)`
- `(@NotNull)`

Mappatura delle classi persistenti (entity)

Anche l'accesso alle property si può sovrascrivere:

```
@Entity
public class Item {

    @Id
    @GeneratedValue(generator = Constants.ID_GENERATOR)
    protected Long id;

    @Access(AccessType.PROPERTY)
    @Column(name = "ITEM_NAME")
    protected String name;

    public String getName() {
        return name;
    }
}
```

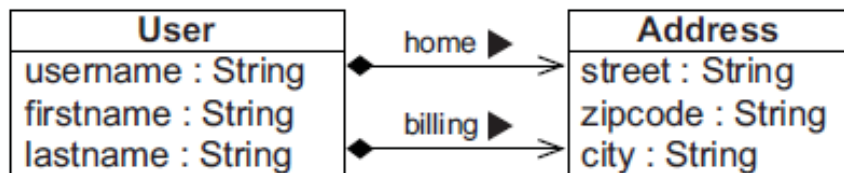
Mappings are still expected here! →

1 @Id is on a field.

2 Switches property to runtime access

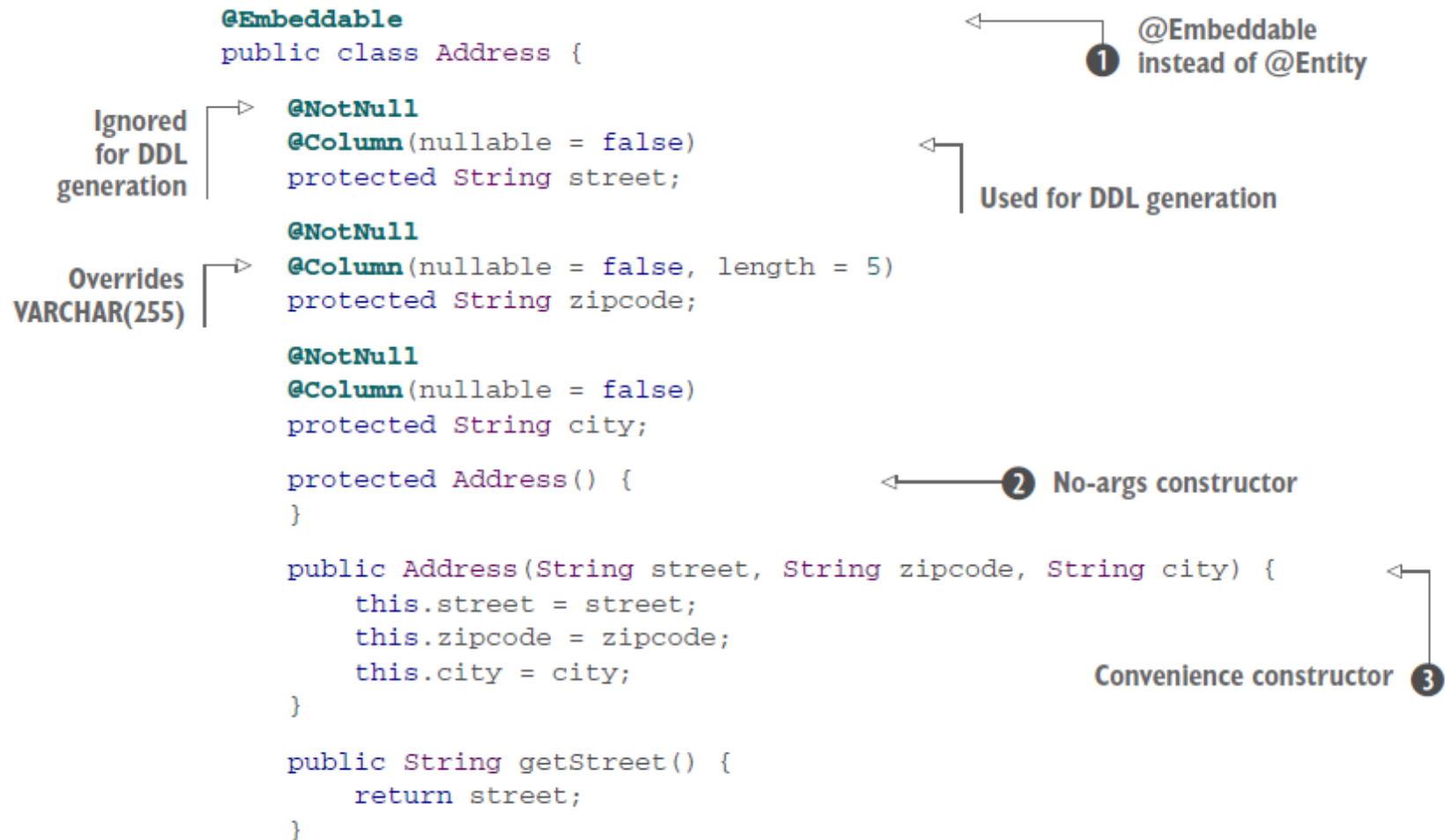
3 Called when loading/storing

Mappatura delle classi persistenti (value type)



<< Table >> USERS	
ID <<PK>>	
USERNAME	
FIRSTNAME	
LASTNAME	
STREET	Component columns
ZIPCODE	
CITY	
BILLING_STREET	
BILLING_ZIPCODE	
BILLING_CITY	

Mappatura delle classi persistenti (value type)



Mappatura delle classi persistenti (value type)

```
@Entity
@Table(name = "USERS")
public class User implements Serializable {

    @Id
    @GeneratedValue(generator = Constants.ID_GENERATOR)
    protected Long id;

    public Long getId() {

    }

    protected Address homeAddress;

    public Address getHomeAddress() {
        return homeAddress;
    }

    public void setHomeAddress(Address homeAddress) {
        this.homeAddress = homeAddress;
    }

    // ...
}
```

← The Address is @Embeddable;
no annotation is needed here.

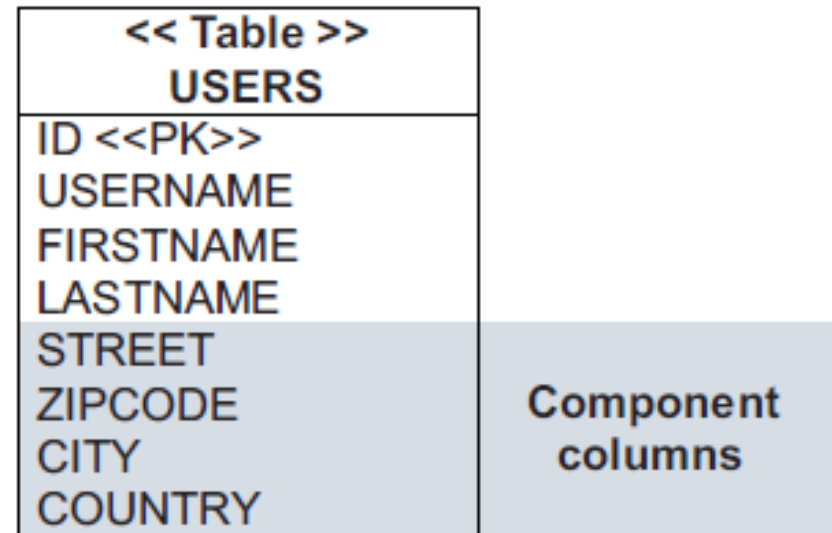
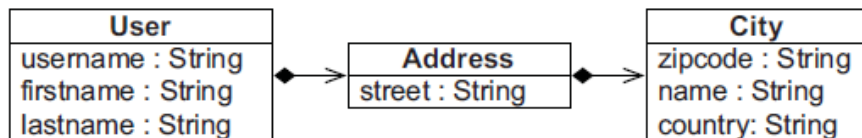
Mappatura delle classi persistenti (value type)

```
@Entity
@Table(name = "USERS")
public class User implements Serializable {

    @Embedded                                     ← Not necessary
    @AttributeOverrides({
        @AttributeOverride(name = "street",
        Nullable! →      column = @Column(name = "BILLING_STREET")),
        @AttributeOverride(name = "zipcode",
        column = @Column(name = "BILLING_ZIPCODE", length = 5)),
        @AttributeOverride(name = "city",
        column = @Column(name = "BILLING_CITY"))
    })
    protected Address billingAddress;

    public Address getBillingAddress() {
        return billingAddress;
    }
}
```

Mappatura delle classi persistenti (value type)



Mappatura delle classi persistenti (value type)

```
@Embeddable
public class Address {

    @NotNull
    @Column(nullable = false)
    protected String street;

    @NotNull
    @AttributeOverrides(
        @AttributeOverride(
            name = "name",
            column = @Column(name = "CITY", nullable = false)
        )
    )
    protected City city;

    // ...
}
```

Mappatura delle classi persistenti (value type)

```
@Embeddable
public class City {

    @NotNull
    @Column(nullable = false, length = 5)           ← Override VARCHAR(255).
    protected String zipcode;

    @NotNull
    @Column(nullable = false)
    protected String name;

    @NotNull
    @Column(nullable = false)
    protected String country;

    // ...
}
```


Mappatura tipi primitivi e numerici

Name	Java type	ANSI SQL type
integer	int, java.lang.Integer	INTEGER
long	long, java.lang.Long	BIGINT
short	short, java.lang.Short	SMALLINT
float	float, java.lang.Float	FLOAT
double	double, java.lang.Double	DOUBLE
byte	byte, java.lang.Byte	TINYINT
boolean	boolean, java.lang.Boolean	BOOLEAN
big_decimal	java.math.BigDecimal	NUMERIC
big_integer	java.math.BigInteger	NUMERIC

Mappatura caratteri e stringhe

Name	Java type	ANSI SQL type
string	<code>java.lang.String</code>	VARCHAR
character	<code>char[]</code> , <code>Character[]</code> , <code>java.lang.String</code>	CHAR
yes_no	<code>boolean</code> , <code>java.lang.Boolean</code>	CHAR(1), 'Y' or 'N'
true_false	<code>boolean</code> , <code>java.lang.Boolean</code>	CHAR(1), 'T' or 'F'
class	<code>java.lang.Class</code>	VARCHAR
locale	<code>java.util.Locale</code>	VARCHAR
timezone	<code>java.util.TimeZone</code>	VARCHAR
currency	<code>java.util.Currency</code>	VARCHAR

Mappatura date e tempi

Name	Java type	ANSI SQL type
date	<code>java.util.Date</code> , <code>java.sql.Date</code>	DATE
time	<code>java.util.Date</code> , <code>java.sql.Time</code>	TIME
timestamp	<code>java.util.Date</code> , <code>java.sql.Timestamp</code>	TIMESTAMP
calendar	<code>java.util.Calendar</code>	TIMESTAMP
calendar_date	<code>java.util.Calendar</code>	DATE
duration	<code>java.time.Duration</code>	BIGINT
instant	<code>java.time.Instant</code>	TIMESTAMP
localdatetime	<code>java.time.LocalDateTime</code>	TIMESTAMP
localdate	<code>java.time.LocalDate</code>	DATE
localtime	<code>java.time.LocalTime</code>	TIME
offsetdatetime	<code>java.time.OffsetDateTime</code>	TIMESTAMP
offsettime	<code>java.time.OffsetTime</code>	TIME
zoneddatetime	<code>java.time.ZonedDateTime</code>	TIMESTAMP

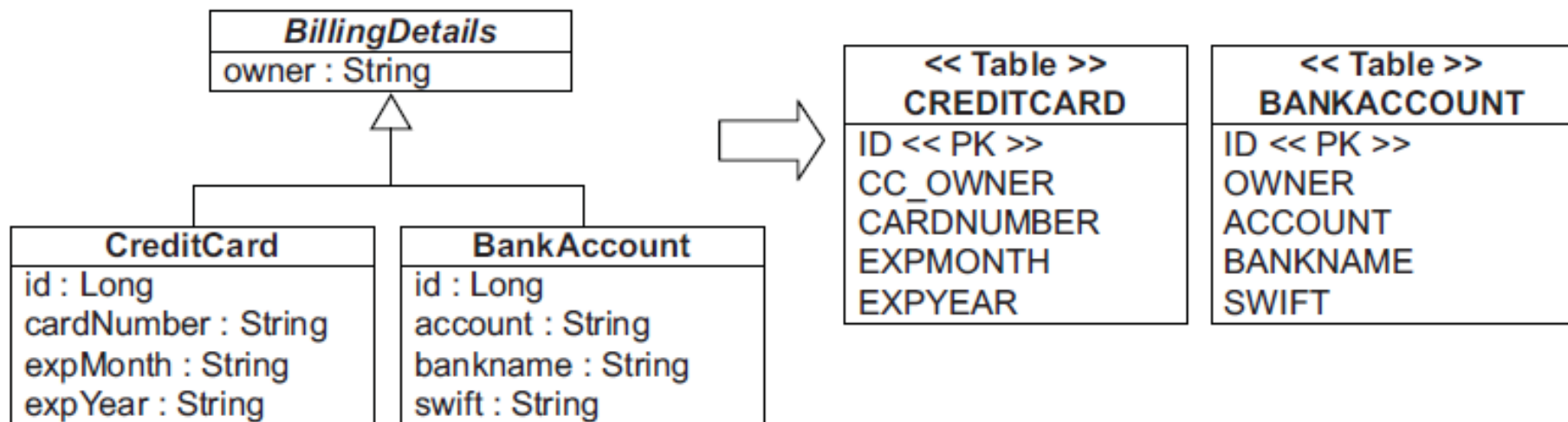
Mappatura file binari

Name	Java type	ANSI SQL type
binary	<code>byte[], java.lang.Byte[]</code>	VARBINARY
text	<code>java.lang.String</code>	CLOB

Name	Java type	ANSI SQL type
clob	<code>java.sql.Clob</code>	CLOB
blob	<code>java.sql.Blob</code>	BLOB
serializable	<code>java.io.Serializable</code>	VARBINARY

Mappatura ereditarietà

1. Una tabella per classe concreta



Mappatura ereditarietà

1. Una tabella per classe concreta

```
@MappedSuperclass
public abstract class BillingDetails {

    @NotNull
    protected String owner;

    // ...
}
```

Mappatura ereditarietà

1. Una tabella per classe concreta

```
@Entity
@AttributeOverride(
    name = "owner",
    column = @Column(name = "CC_OWNER", nullable = false))
public class CreditCard extends BillingDetails {

    @Id
    @GeneratedValue(generator = Constants.ID_GENERATOR)
    protected Long id;

    @NotNull
    protected String cardNumber;

    @NotNull
    protected String expMonth;

    @NotNull
    protected String expYear;

    // ...
}
```

Mappatura ereditarietà

1. Una tabella per classe concreta

Polimorfismo implicito, problemi:

- Se un'altra entity ha un'associazione con **BillingDetails** (ha bisogno di referenziarla con una chiave esterna)?
- Molte colonne, di diverse tabelle, condividono semantica: se cambio nome o tipo di una property della superclasse devo modificare n sottoclassi

Mappatura ereditarietà

1. Una tabella per classe concreta

- Problemi con query polimorfiche: `select bd from BillingDetails bd` genera 2 query:

```
select
    ID, OWNER, ACCOUNT, BANKNAME, SWIFT
from
    BANKACCOUNT
select
    ID, CC_OWNER, CARDNUMBER, EXPMONTH, EXPYEAR
from
    CREDITCARD
```

Mappatura ereditarietà

2. Una tabella per classe concreta con unions

(Stesso schema del caso precedente)

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class BillingDetails {

    @Id
    @GeneratedValue(generator = Constants.ID_GENERATOR)
    protected Long id;

    @NotNull

    protected String owner;

    // ...
}
```

Mappatura ereditarietà

2. Una tabella per classe concreta con unions

```
@Entity
public class CreditCard extends BillingDetails {

    @NotNull
    protected String cardNumber;

    @NotNull
    protected String expMonth;

    @NotNull
    protected String expYear;

    // ...
}
```

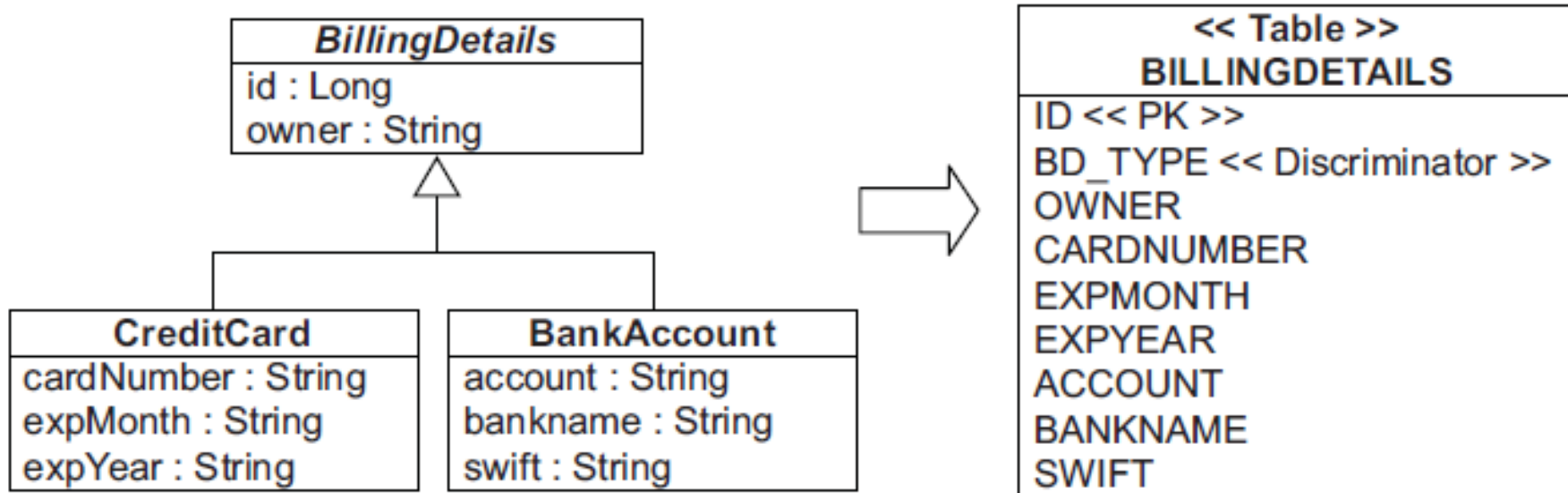
Mappatura ereditarietà

2. Una tabella per classe concreta con unions

```
select
    ID, OWNER, EXPMONTH, EXPYEAR, CARDNUMBER,
    ACCOUNT, BANKNAME, SWIFT, CLAZZ_
from
    ( select
        ID, OWNER, EXPMONTH, EXPYEAR, CARDNUMBER,
        null as ACCOUNT,
        null as BANKNAME,
        null as SWIFT,
        1 as CLAZZ_
    from
        CREDITCARD
    union all
    select
        id, OWNER,
        null as EXPMONTH,
        null as EXPYEAR,
        null as CARDNUMBER,
        ACCOUNT, BANKNAME, SWIFT,
        2 as CLAZZ_
    from
        BANKACCOUNT
    ) as BILLINGDETAILS
```

Mappatura ereditarietà

3. Una tabella per l'intera gerarchia delle classi



Mappatura ereditarietà

3. Una tabella per l'intera gerarchia delle classi

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "BD_TYPE")
public abstract class BillingDetails {

    @Id
    @GeneratedValue(generator = Constants.ID_GENERATOR)
    protected Long id;

    @NotNull
    @Column(nullable = false)
    protected String owner;

    // ...
}

@Entity
@DiscriminatorValue("CC")
public class CreditCard extends BillingDetails {

    @NotNull
    protected String cardNumber;

    @NotNull
    protected String expMonth;

    @NotNull
    protected String expYear;

    // ...
}
```

← Ignored by Hibernate for schema generation!

← Ignored by Hibernate for DDL generation!

Mappatura ereditarietà

3. Una tabella per l'intera gerarchia delle classi

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@org.hibernate.annotations.DiscriminatorFormula(
    "case when CARDNUMBER is not null then 'CC' else 'BA' end"
)
public abstract class BillingDetails {
    // ...
}
```

Feature solo di hibernate, se non è possibile inserire la colonna con il discriminante delle classi

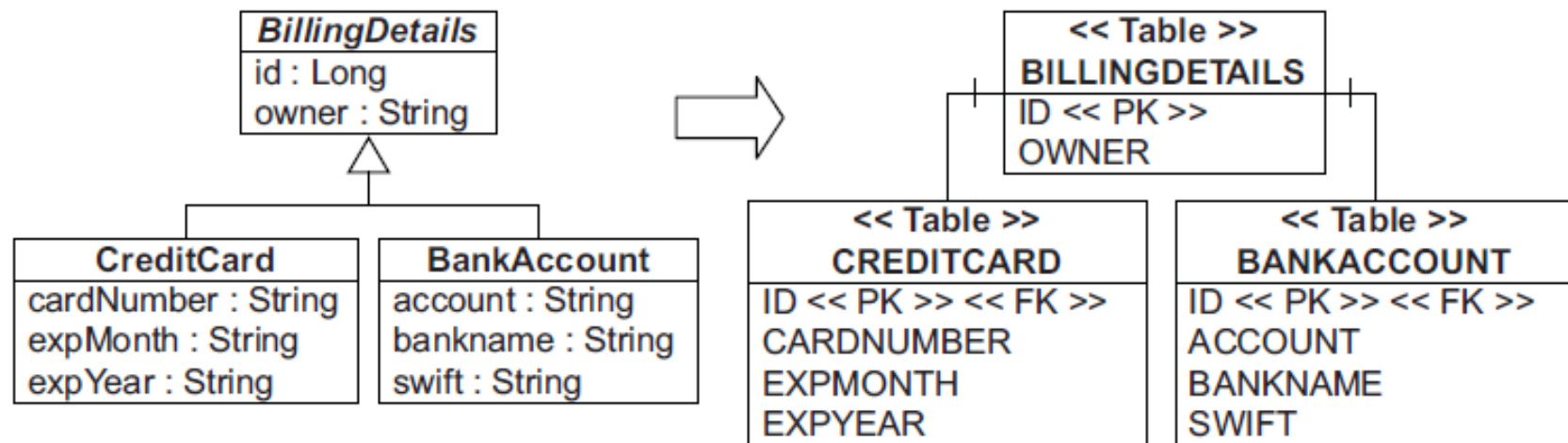
Mappatura ereditarietà

3. Una tabella per l'intera gerarchia delle classi

- Semplice
- performante
- mette a rischio integrità dei dati (si perdono i constraint not null)
- violazione della terza forma normale (chiedere al DBA...)

Mappatura ereditarietà

3. Una tabella per sottoclasse con join



Mappatura ereditarietà

3. Una tabella per sottoclasse con join

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class BillingDetails {

    @Id
    @GeneratedValue(generator = Constants.ID_GENERATOR)
    protected Long id;

    @NotNull
    protected String owner;

    // ...
}

@Entity
public class BankAccount extends BillingDetails {

    @NotNull
    protected String account;

    @NotNull
    protected String bankname;

    @NotNull
    protected String swift;

    // ...
}
```

Mappatura ereditarietà

3. Una tabella per sottoclasse con join

select bd from BillingDetails bd:

```
select
    BD.ID, BD.OWNER,
    CC.EXPMONTH, CC.EXPYEAR, CC.CARDNUMBER,
    BA.ACCOUNT, BA.BANKNAME, BA.SWIFT,
    case
        when CC.CREDITCARD_ID is not null then 1
        when BA.ID is not null then 2
        when BD.ID is not null then 0
    end
from
    BILLINGDETAILS BD
    left outer join CREDITCARD CC on BD.ID=CC.CREDITCARD_ID
    left outer join BANKACCOUNT BA on BD.ID=BA.ID
```

Mappatura ereditarietà

3. Una tabella per sottoclasse con join

select cc from CreditCard cc:

```
select
    CREDITCARD_ID, OWNER, EXPMONTH, EXPYEAR, CARDNUMBER
from
    CREDITCARD
    inner join BILLINGDETAILS on CREDITCARD_ID=ID
```

- Schema normalizzato, facile da mantenere, definizione dei constraint facile
- Problemi di performance con gerarchie di classi complesse

Mappatura ereditarietà

Scelta della strategia

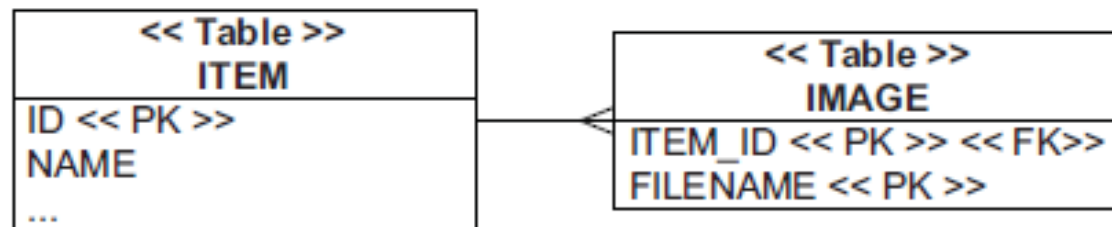
- Verificare quante volte si utilizzano query sulla superclasse e se la superclasse viene referenziata anche da altre classi
- Quanti attributi sono propri delle sottoclassi e quanti sono invece ereditati

Mappatura ereditarietà

Scelta della strategia

- Nel caso ci vogliano associazioni e query polimorfiche e sottoclassi differiscono dalla superclasse soprattutto per il comportamento => `InheritanceType.SINGLE_TABLE`
- Nel caso le sottoclassi dichiarino molte property non ereditate (e non opzionali) => `InheritanceType.JOINED`, alternativamente se gerarchia troppo complessa `InheritanceType.TABLE_PER_CLASS`

Mappatura delle Collections



Mappatura delle Collections

Non è strettamente necessario mappare una collection (Item#images), ma facendolo si hanno i seguenti vantaggi:

- `someltem.getImages()` esegue automaticamente `SELECT * from IMAGE where ITEM_ID = ?` (però le preleva tutte...)
- evita di salvare ogni Image con `entityManager.persist()`, basta fare `someltem.getImages().add()` (e salvare l'Item)
- Il ciclo di vita delle Image dipende da Item, basta cancellare un Item per cancellare in automatico tutte le Image associate

Mappatura delle Collections

Obbligatorio dichiarare collection come interfaccia:

```
<<Interface>> images = new <<Implementation>>
```

Hibernate usa le 'sue' collection, che wrappano comunque quelle più importanti, a meno di estendere Hibernate, queste sono supportate:

- java.util.Set inizializzata come java.util.HashSet
- java.util.SortedSet inizializzata come java.util.TreeSet
- java.util.List inizializzata come java.util.ArrayList
- java.util.Collection inizializzata come java.util.ArrayList (bag)
- java.util.Map inizializzata come java.util.HashMap
- java.util.SortedMap inizializzata come java.util.TreeMap
- Hibernate supporta anche Array, a differenza di JPA

Mappatura delle Collections (Set)

```
@Entity
public class Item {

    @ElementCollection
    @CollectionTable(
        name = "IMAGE",
        joinColumns = @JoinColumn(name = "ITEM_ID"))
    @Column(name = "FILENAME")
    protected Set<String> images = new
        HashSet<String>();

    // ...
}
```

Defaults to ITEM_IMAGES

Default

Defaults to IMAGES

Initialize the field here.

ITEM	
<u>ID</u>	NAME
1	Foo
2	B
3	C

IMAGE	
<u>ITEM_ID</u>	<u>FILENAME</u>
1	foo.jpg
1	bar.jpg
1	baz.jpg
2	b.jpg

Mappatura delle Collections (Bag)



ITEM	
<u>ID</u>	NAME
1	Foo
2	B
3	C

IMAGE		
<u>IMAGE_ID</u>	ITEM_ID	FILENAME
1	1	foo.jpg
2	1	bar.jpg
3	1	baz.jpg
4	1	baz.jpg
5	2	b.jpg

Mappatura delle Collections (List)

```
@Entity
public class Item {

    @ElementCollection
    @CollectionTable(name = "IMAGE")
    @OrderColumn
    @Column(name = "FILENAME")
    protected List<String> images = new ArrayList<String>();

    // ...
}
```

← Enables persistent order; defaults to IMAGES_ORDER

ITEM	
<u>ID</u>	NAME
1	Foo
2	B
3	C


IMAGE		
<u>ITEM_ID</u>	<u>IMAGES_ORDER</u>	FILENAME
1	0	foo.jpg
1	1	bar.jpg
1	2	baz.jpg
1	3	baz.jpg
2	0	b1jpg
2	1	b2.jpg

Mappatura delle Collections (Map)

```
@Entity
public class Item {

    @ElementCollection
    @CollectionTable(name = "IMAGE")
    @MapKeyColumn(name = "FILENAME")
    @Column(name = "IMAGENAME")
    protected Map<String, String> images = new HashMap<String, String>();

    // ...
}
```



ITEM	
<u>ID</u>	NAME
1	Foo
2	B
3	C

IMAGE		
<u>ITEM_ID</u>	<u>FILENAME</u>	IMAGENAME
1	foo.jpg	Foo
1	bar.jpg	Bar
1	baz.jpg	Baz
2	b1.jpg	B1
2	b2.jpg	B2

Mappatura delle Collections (Collection ordinate in memoria)

```
@Entity
public class Item {

    @ElementCollection
    @CollectionTable(name = "IMAGE")
    @MapKeyColumn(name = "FILENAME")
    @Column(name = "IMAGENAME")
    @org.hibernate.annotations.SortComparator(ReverseStringComparator.class)
    protected SortedMap<String, String> images =
        new TreeMap<String, String>();

    // ...
}
```

```
@Entity
public class Item {

    @ElementCollection
    @CollectionTable(name = "IMAGE")
    @Column(name = "FILENAME")
    @org.hibernate.annotations.SortNatural
    protected SortedSet<String> images = new TreeSet<String>();

    // ...
}
```

Mappatura delle Collections (Collection ordinate da query)

```
@Entity
public class Item {

    @ElementCollection
    @CollectionTable(name = "IMAGE")
    @Column(name = "FILENAME")
    // @javax.persistence.OrderBy
    @org.hibernate.annotations.OrderBy(clause = "FILENAME desc")
    protected Set<String> images = new LinkedHashSet<String>();

    // ...
}
```

Only one possible
order: "FILENAME asc"

```
@Entity
public class Item {

    @ElementCollection
    @CollectionTable(name = "IMAGE")
    @Column(name = "FILENAME")
    @org.hibernate.annotations.CollectionId(
        columns = @Column(name = "IMAGE_ID"),
        type = @org.hibernate.annotations.Type(type = "long"),
        generator = Constants.ID_GENERATOR)
    @org.hibernate.annotations.OrderBy(clause = "FILENAME desc")
    protected Collection<String> images = new ArrayList<String>();

    // ...
}
```

Surrogate primary
key allows duplicates

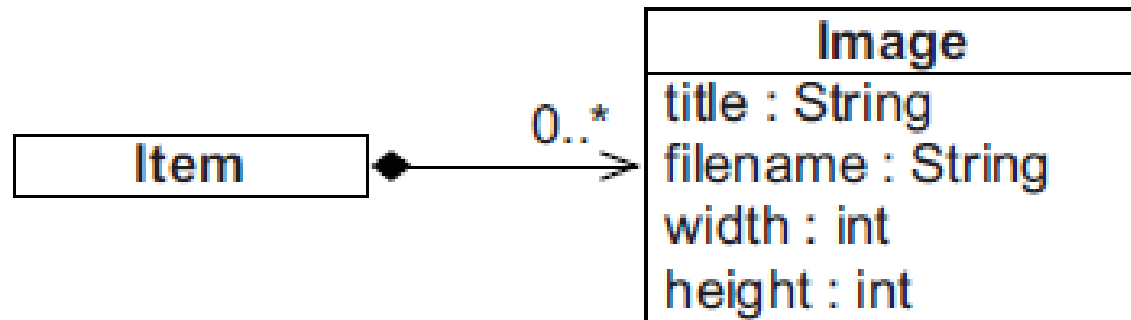
Mappatura delle Collections (Collection ordinate da query)

```
@Entity
public class Item {

    @ElementCollection
    @CollectionTable(name = "IMAGE")
    @MapKeyColumn(name = "FILENAME")
    @Column(name = "IMAGENAME")
    @org.hibernate.annotations.OrderBy(clause = "FILENAME desc")
    protected Map<String, String> images = new LinkedHashMap<String, String>();

    // ...
}
```

Mappatura delle Collections (Components)



```
@Embeddable
public class Image {

    @Column(nullable = false)
    protected String title;

    @Column(nullable = false)
    protected String filename;

    protected int width;

    protected int height;

    // ...
}
```

Mappatura delle Collections (Set of components)

In questo modo avremo 4 immagini nella collection:

```
someItem.getImages().add(new Image(
    "Foo", "foo.jpg", 640, 480
));
someItem.getImages().add(new Image(
    "Bar", "bar.jpg", 800, 600
));
someItem.getImages().add(new Image(
    "Baz", "baz.jpg", 1024, 768
));
someItem.getImages().add(new Image(
    "Baz", "baz.jpg", 1024, 768
));
assertEquals(someItem.getImages().size(), 3);
```

`java.lang.Object#equals()` utilizza `==` per confrontare le istanze

Mappatura delle Collections (Set of components)

`@Embeddable`

```
public class Image {
```

`@Override`

```
public boolean equals(Object o) {  
    if (this == o) return true;  
    if (o == null || getClass() != o.getClass()) return false;  
  
    Image other = (Image) o;  
  
    if (!title.equals(other.title)) return false;  
    if (!filename.equals(other.filename)) return false;  
    if (width != other.width) return false;  
    if (height != other.height) return false;  
  
    return true;  
}
```

1 Equality check



`@Override`

```
public int hashCode() {  
    int result = title.hashCode();  
    result = 31 * result + filename.hashCode();  
    result = 31 * result + width;  
    result = 31 * result + height;  
    return result;  
}
```

2 Must be symmetric



Mappatura delle Collections (Set of components)

`@Entity`

```
public class Item {
```

```
    @ElementCollection
```

```
    @CollectionTable(name = "IMAGE")
```

```
    @AttributeOverride(
```

```
        name = "filename",
```

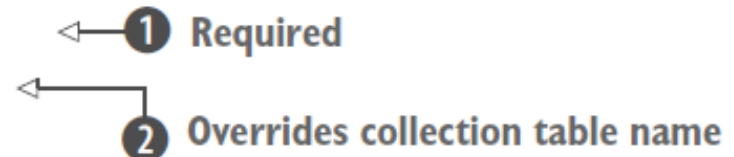
```
        column = @Column(name = "FNAME", nullable = false)
```

```
    )
```

```
    protected Set<Image> images = new HashSet<Image>();
```

```
    // ...
```

```
}
```



Mappatura delle Collections (Set of components)

ITEM

<u>ID</u>	NAME
1	Foo
2	B
3	C

IMAGE

<u>ITEM_ID</u>	<u>TITLE</u>	<u>FNAME</u>	<u>WIDTH</u>	<u>HEIGHT</u>
1	Foo	foo.jpg	640	480
1	Bar	bar.jpg	800	600
1	Baz	baz.jpg	1024	768
2	B	b.jpg	640	480

Mappatura delle Collections (Set of components)

Nella chiave primaria c'è anche ITEM_ID, per avere simmetria tra db e codice java, occorre aggiungerlo a equals(), per esempio `this.getItem().getId().equals(other.getItem().getId())`. Prima va definito così in Image:

```
@Embeddable
public class Image {

    @org.hibernate.annotations.Parent
    protected Item item;

    // ...
}
```

Mappatura delle Collections (Set of components)

Per avere un insieme ordinato al caricamento dal db (e durante l'iterazione):

```
@Entity
public class Item {

    @ElementCollection
    @CollectionTable(name = "IMAGE")
    @OrderBy("filename, width DESC")
    protected Set<Image> images = new LinkedHashSet<Image>();

    // ...
}
```


Mappatura delle Collections (Bag of components)

```
@Embeddable
public class Image {

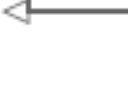
    @Column(nullable = true)
    protected String title;

    @Column(nullable = false)
    protected String filename;

    protected int width;

    protected int height;

    // ...
}
```



Can be null if you have a
surrogate primary key

Mappatura delle Collections (Bag of components)

```
@Entity
public class Item {

    @ElementCollection
    @CollectionTable(name = "IMAGE")
    @org.hibernate.annotations.CollectionId(
        columns = @Column(name = "IMAGE_ID"),
        type = @org.hibernate.annotations.Type(type = "long"),
        generator = Constants.ID_GENERATOR)
    protected Collection<Image> images = new ArrayList<Image>();

    // ...
}
```

Mappatura delle Collections (Bag of components)

ITEM

<u>ID</u>	NAME
1	Foo
2	B
3	C

IMAGE

<u>IMAGE_ID</u>	ITEM_ID	TITLE	FILENAME	WIDTH	HEIGHT
1	1	Foo	foo.jpg	640	480
2	1		bar.jpg	800	600
3	1	Baz	baz.jpg	1024	768
4	1	Baz	baz.jpg	1024	768
5	2	B	b.jpg	640	480

Mappatura delle Collections (Map of components)

```
@Entity
public class Item {

    @ElementCollection
    @CollectionTable(name = "IMAGE")
    @MapKeyColumn(name = "FILENAME")
    protected Map<String, Image> images = new HashMap<String, Image>();

    // ...
}
```

Optional; defaults to IMAGES_KEY



Mappatura delle Collections (Map of components)

ITEM

<u>ID</u>	NAME
1	Foo
2	B
3	C

IMAGE

<u>ITEM_ID</u>	<u>FILENAME</u>	TITLE	WIDTH	HEIGHT
1	foo.jpg	Foo	640	480
1	bar.jpg		800	600
1	baz.jpg	Baz	1024	768
2	b.jpg	B	640	480

Mappatura delle Collections (Map of components)

ITEM

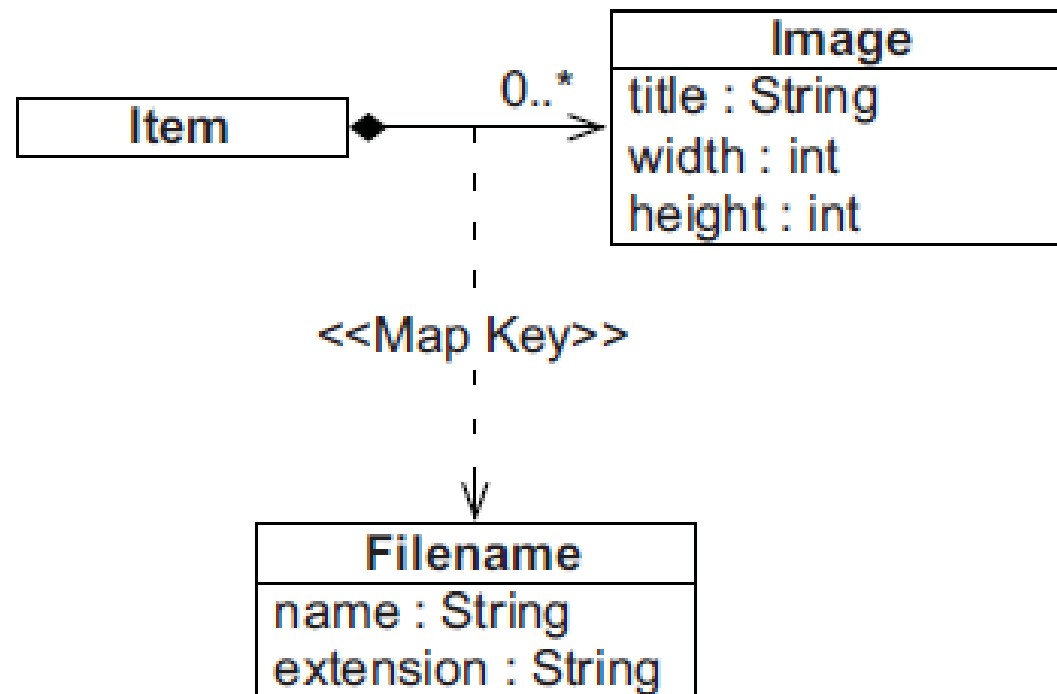
<u>ID</u>	NAME
1	Foo
2	B
3	C

IMAGE

<u>ITEM_ID</u>	<u>FILENAME</u>	TITLE	WIDTH	HEIGHT
1	foo.jpg	Foo	640	480
1	bar.jpg		800	600
1	baz.jpg	Baz	1024	768
2	b.jpg	B	640	480

Mappatura delle Collections (Map of components)

Adesso la key non è la stringa filename, ma un'altra component:



Mappatura delle Collections (Map of components)

```
@Embeddable
public class Filename {

    @Column(nullable = false)
    protected String name;

    @Column(nullable = false)
    protected String extension;

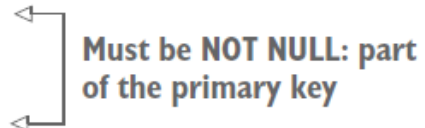
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Filename filename = (Filename) o;

        if (!extension.equals(filename.extension)) return false;
        if (!name.equals(filename.name)) return false;

        return true;
    }

    @Override
    public int hashCode() {
        int result = name.hashCode();
        result = 31 * result + extension.hashCode();
        return result;
    }
}
```



Must be NOT NULL: part of the primary key

Mappatura delle Collections (Map of components)

```
@Entity
public class Item {

    @ElementCollection
    @CollectionTable(name = "IMAGE")
    protected Map<Filename, Image> images = new HashMap<Filename, Image>();

    // ...
}
```

Mappatura delle Collections (Map of components)

ITEM

<u>ID</u>	NAME
1	Foo
2	B
3	C

IMAGE

<u>ITEM_ID</u>	<u>NAME</u>	<u>EXTENSION</u>	TITLE	WIDTH	HEIGHT
1	foo	jpg	Foo	640	480
1	bar	jpg		800	600
1	baz	jpg	Baz	1024	768
2	b	jpg	B	640	480

Mappatura delle Collections (Collection in embeddable component)

```
@Embeddable
public class Address {

    @NotNull
    @Column(nullable = false)
    protected String street;

    @NotNull
    @Column(nullable = false, length = 5)
    protected String zipcode;

    @NotNull
    @Column(nullable = false)
    protected String city;

    @ElementCollection
    @CollectionTable(
        name = "CONTACT",
        joinColumns = @JoinColumn(name = "USER_ID"))
    @Column(name = "NAME", nullable = false)
    protected Set<String> contacts = new HashSet<String>();
    // ...
}
```

The diagram illustrates the mapping of the `contacts` collection in the `Address` class. It shows three annotations with arrows pointing to their default values:

- `@CollectionTable`: An arrow points to `name = "CONTACT"` with the label "Defaults to USER_CONTACTS".
- `@JoinColumn`: An arrow points to `name = "USER_ID"` with the label "Default".
- `@Column`: An arrow points to `name = "NAME"` with the label "Defaults to CONTACTS".

Mappatura delle Collections (Collection in embeddable component)

