Collection Mapping

CS544: Enterprise Architecture

Collections

- Java uses collections of references to implement the many side(s) of associations
  - So far we've always mapped collections as bags
- There are 4 Types of collections:
  - Bags, Sets, Lists, and Maps
- These 4 can be split into 2 categories
  - Indexed, and non-indexed collections

Program to Interface

- Hibernate requires you to program to interface (P2I) for your collections
  - Replaces collections with its own implementations
  - P2I is also a general Java best practice

```
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany
  private List<Car> cars = new ArrayList();

  ...
```

collection initialized right away

Interface

Hibernate replaces ArrayList with its own List implementation once the collection has been retrieved or persisted

Collection Mapping

COLLECTION: BAG

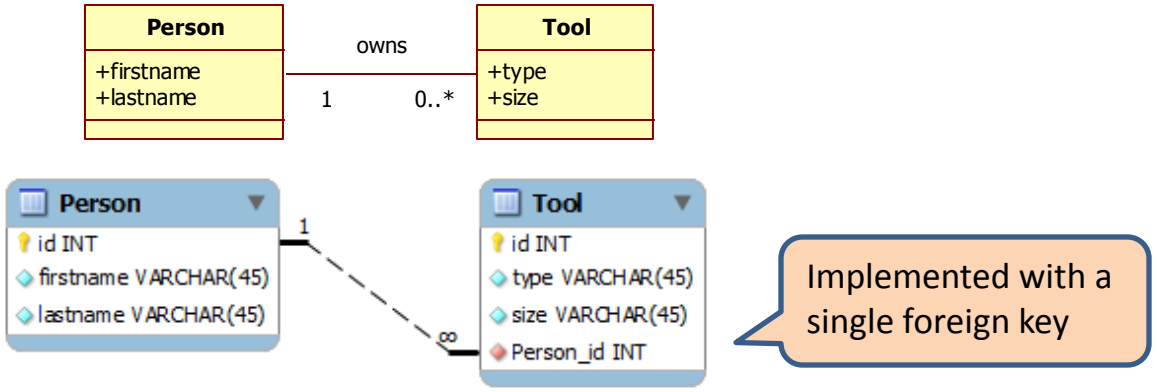
Bags

- The most basic collection is a bag
  - A bag has no inherent order
  - A bag can contain duplicates

- People often own a ‘bag’ of hardware tools
  - No inherent order
  - May contain duplicate tools

Bag Implementation

- java.util.Collection is a bag interface
  - Java has no official bag implementation
- Bags are non-indexed collections
  - A relational database can implement a bag using only a Foreign Key, no additional index



Mapping a Bag

- java.util.Collection maps as a Bag

```
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany(mappedBy="owner", cascade=CascadeType.PERSIST)
  private Collection<Tool> tools = new ArrayList();
```

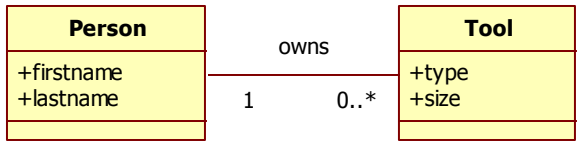
Hibernate will map a Collection as a Bag

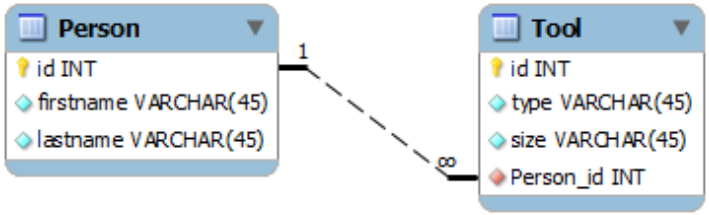
We use an ArrayList since there is no official java Bag implementation

```
@Entity
public class Tool {
  @Id
  @GeneratedValue
  private int id;
  private String type;
  private String size;
  @ManyToOne
  private Person owner;

  ...
```

We've mapped this collection as a bi-directional one to many





Mapping a Bag

- By default java.util.List maps as a Bag

```
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany(mappedBy="owner", cascade=CascadeType.PERSIST)
  private List<Tool> tools = new ArrayList();
```

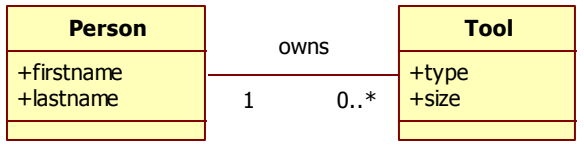
By default List also maps to a Bag

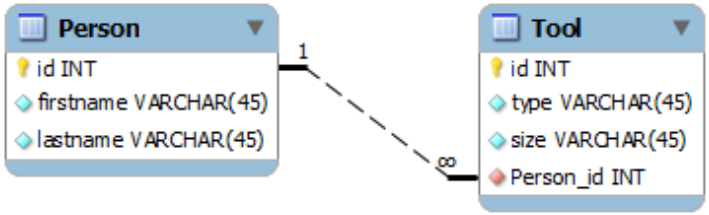
ArrayList is the most common List implementation

```
@Entity
public class Tool {
  @Id
  @GeneratedValue
  private int id;
  private String type;
  private String size;
  @ManyToOne
  private Person owner;

  ...
```

Same bi-directional one to many as last slide





Bag XML

```
<hibernate-mapping package="bag_collection">
  <class name="Person" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="firstname" />
    <property name="lastname" />
    <bag name="tools" inverse="true" access="field" cascade="persist">
      <key column="owner_id" />
      <one-to-many class="Tool"/>
    </bag>
  </class>
</hibernate-mapping>
```

The XML bag mappings for java.util.Collection and java.util.List are the same

Only the name attribute is required

Use the <bag> tag to map bag

<key> and an association tag are required

```
<hibernate-mapping package="bag_collection">
  <class name="Tool" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="type" />
    <property name="size" />
    <many-to-one name="owner" column="owner_id" class="Person" />
  </class>
</hibernate-mapping>
```

For the sake of completeness: the other side of the bi-directional one to many association

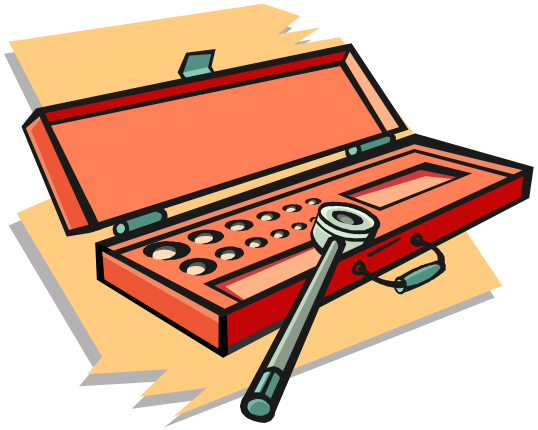
Collection Mapping

COLLECTION: SET

Sets

- Sets are bags that can not contain duplicates:
  - A set still has no inherent order
  - A set can not contain duplicates

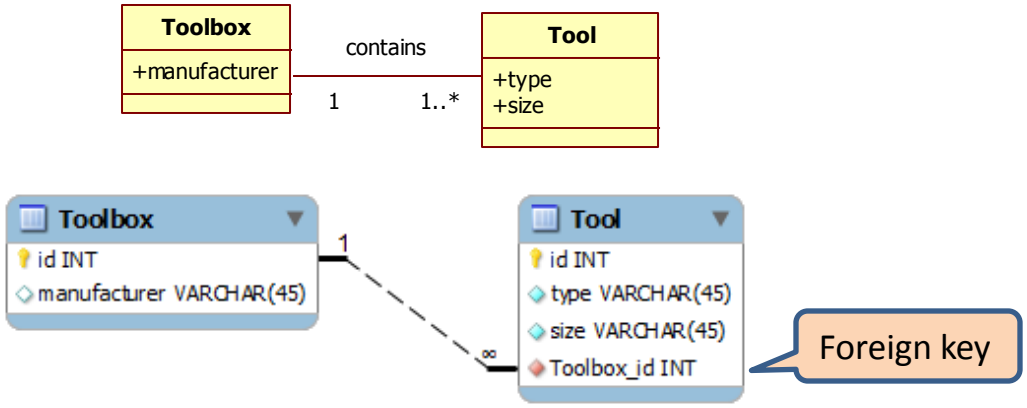
- Store bought toolboxes are generally a set
  - No duplicates
  - No inherent order*



Set Implementation

- Java has the java.util.Set interface
  - java.util.HashSet is the general implementation

- Like bags, sets are non-indexed collections
  - A set can be implemented using a Foreign Key, no additional index



equals() & hashCode()

```
@Entity
public class Name {
  private String firstname;
  private String lastname;

  ...

  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if ((obj == null) || obj.getClass() != this.getClass())
      return false;
    Name n = (Name) obj;
    if (firstname == n.firstname || (firstname != null && firstname.equals(n.firstname))
      && lastname == n.lastname || (lastname != null && lastname.equals(n.lastname))) {
      return true;
    } else {
      return false;
    }
  }

  public int hashCode() {
    int hash = 1234;
    if (firstname != null)
      hash = hash + firstname.hashCode();
    if (lastname != null)
      hash = hash + lastname.hashCode();
    return hash;
  }
```

Compares object contents for equality

Generates an int based on the class contents

Mapping a Set

- java.util.Set maps as a Set

```
@Entity
public class Toolbox {
  @Id
  @GeneratedValue
  private int id;
  private String manufacturer;
  private String model;
  @OneToMany(mappedBy="toolbox", cascade=CascadeType.PERSIST)
  private Set<Tool> tools = new HashSet();
```

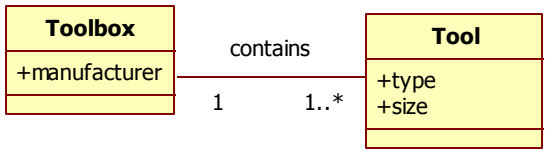
Set maps as a set

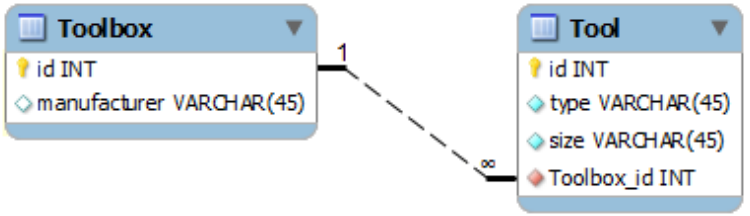
HashSet is the most common Set implementation

```
@Entity
public class Tool {
  @Id
  @GeneratedValue
  private int id;
  private String type;
  private String size;
  @ManyToOne
  private Toolbox toolbox;

  ...
```

Tool class completes the bi-directional many to one





Set XML

```
<hibernate-mapping package="set">
  <class name="Toolbox" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="manufacturer" />
    <set name="tools" inverse="true" access="field" cascade="persist">
      <key column="toolbox_id" />
      <one-to-many class="Tool"/>
    </set>
  </class>
</hibernate-mapping>
```

Only difference between mapping a set and a bag is in the tag name <set>

As with the bag mapping <key> and an association tag are required

```
<hibernate-mapping package="set">
  <class name="Tool" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="type" />
    <property name="size" />
    <many-to-one name="toolbox" column="toolbox_id" class="Toolbox" />
  </class>
</hibernate-mapping>
```

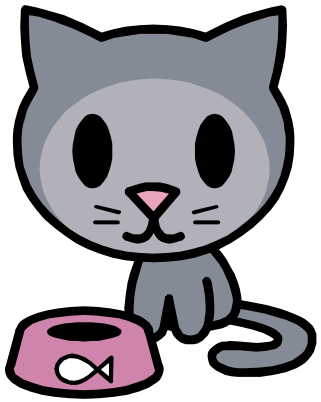
Collection Mapping

COLLECTION: MAP

Maps

- A Map ‘maps’ a set of keys to a bag of values:
  - Each value in the bag has a unique key
  - Given a key, the map can quickly retrieve the value
  - No inherent order in either keys or values

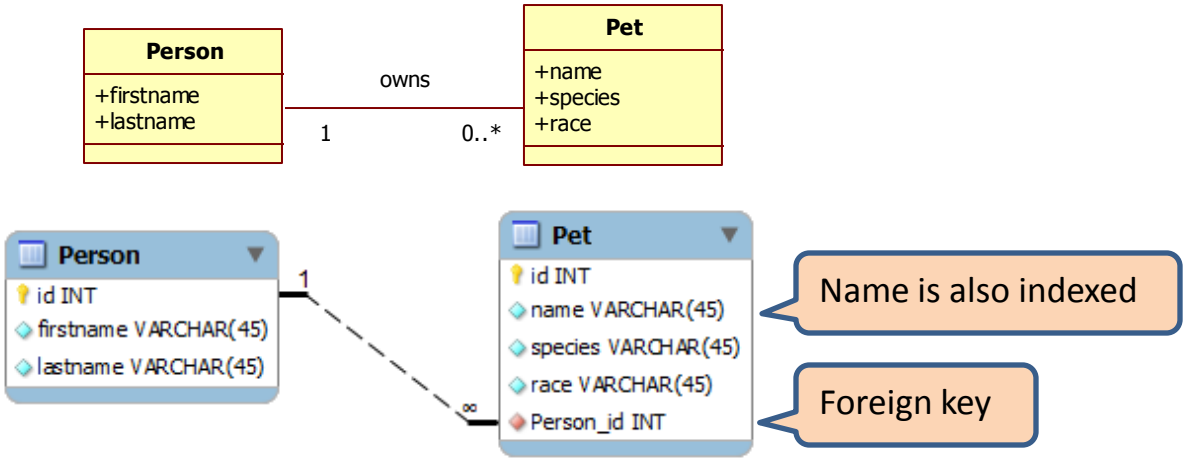
- Pet owner ship can be modeled as a map.
  - Each pet has a unique name*
  - To find a pet, you use its name
  - No inherent order in names or pets



Map Implementation

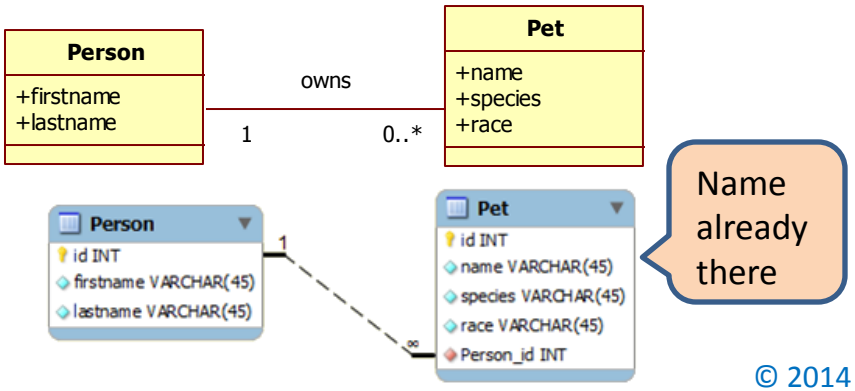
- Java has the java.util.Map interface
  - Java.util.HashMap is the most common map

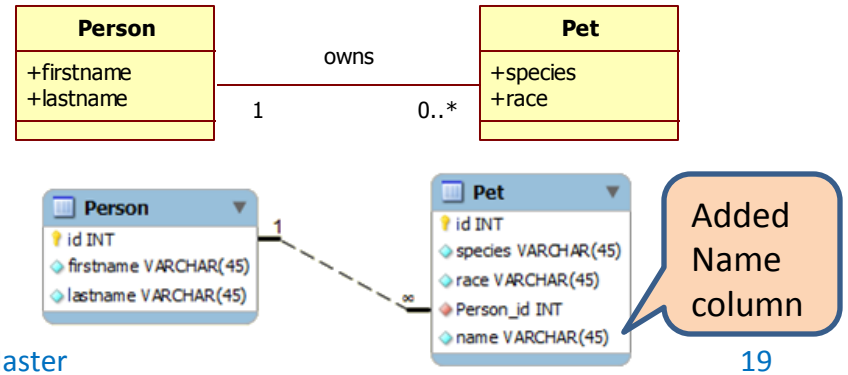
- Maps are indexed collections
  - They need a foreign key and an additional column for the keys which also needs to be indexed



Map Mapping Issues

- There are two types of map mappings:
  1. The key is already part of the value entity class
     - E.g. name is already part of the Pet class
  2. The key is not part of the value entity class
     - The key index column becomes an additional column, just like the index column for a list
     - Mapping issues similar to List (coming up)





Key is part of the Entity Class

```
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany(mappedBy="owner", cascade=CascadeType.PERSIST)
  @MapKey(name="name")
  private Map<String, Pet> pets = new HashMap();

  ...
```

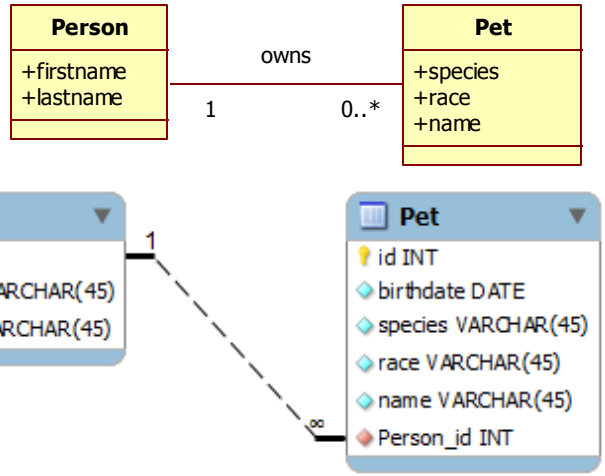
@MapKey specifies the key column on the remote class

Normal @OneToMany

```
@Entity
public class Pet {
  @Id
  @GeneratedValue
  private int id;
  private String name;
  private String species;
  private String race;
  @ManyToOne
  private Person owner;
  ...
```

Normal @ManyToOne

Specified by @MapKey, will be indexed



XML

```xml
<hibernate-mapping package="map">
  <class name="Person" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="firstname" />
    <property name="lastname" />
    <map name="pets" access="field" cascade="persist" inverse="true">
      <key column="owner_id" />
      <map-key type="string" column="name"/>
      <one-to-many class="Pet"/>
    </map>
  </class>
</hibernate-mapping>
```

<map> similar to <bag> and <set>

<map-key> specifies the key column on the remote table

```xml
<hibernate-mapping package="map">
  <class name="Pet">
    <id name="id">
      <generator class="native" />
    </id>
    <property name="name" />
    <property name="species" />
    <property name="race" />
    <many-to-one name="owner" column="owner_id" class="Person" />
  </class>
</hibernate-mapping>
```

Regular <many-to-one>

Map – Separate Key

```java
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany(cascade = CascadeType.PERSIST)
  @JoinColumn(name="owner_id")
  @MapKeyColumn(name="name", nullable=true)
  private Map<String, Pet> pets = new HashMap();

  ...
```

@JoinColumn and its name attribute are required

@OneToMany is the owning side

@MapKeyColumn defaults to not nullable ->insertion problems

```java
@Entity
public class Pet {
  @Id
  @GeneratedValue
  private int id;
  private String species;
  private String race;
  @ManyToOne
  @JoinColumn(name="owner_id", updatable=false, insertable=false)
  private Person owner;

  ...
```

@ManyToOne does not have a mappedBy attribute

False updateble and insertable on @JoinColumn instead of mappedBy

Separate Key XML

```
<hibernate-mapping package="map_seperateKey">
  <class name="Person" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="firstname" />
    <property name="lastname" />
    <map name="pets" access="field" cascade="persist" >
      <key column="owner_id" />
      <map-key type="string" column="name" />
      <one-to-many class="Pet"/>
    </map>
  </class>
</hibernate-mapping>
```

Map side is the owning side (no inverse)

When <map-key> specifies a non-existing column Hibernate will add it as the key column

```
<hibernate-mapping package="map_seperateKey">
  <class name="Pet">
    <id name="id">
      <generator class="native" />
    </id>
    <property name="species" />
    <property name="race" />
    <many-to-one name="owner" column="owner_id" class="Person"
        insert="false" update="false" />
  </class>
</hibernate-mapping>
```

<many-to-one> does not have an inverse attribute to specify that it is not the owning side

Uses insert and update false instead to emulate inverse behavior

Collection Mapping

COLLECTION: LIST

Lists

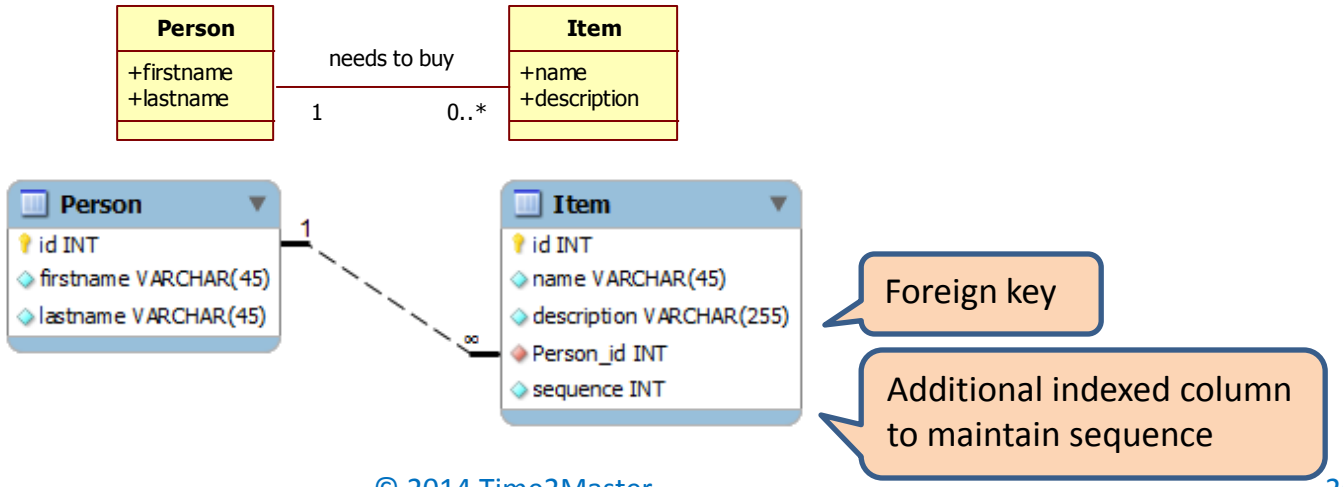
- Lists are bags with an inherent order:
  - A List has an inherent, arbitrary order
  - A List can still contain duplicates

- A shopping list is a typical list example
  - An inherent, although often arbitrary order
  - May contain duplicates

List Implementation

- Java has the java.util.List interface
  - Java.util.ArrayList is the most common list

- Lists are indexed collections
  - A List needs an additional indexed sequence column to maintain its sequence



Incorrectly Mapped One to Many bi-directional List

```
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany(mappedBy="buyer",cascade=CascadeType.PERSIST)
  @OrderColumn(name="sequence")
  private List<Item> shopList = new ArrayList();
```

@OrderColumn specifies the additional column on the Item table

How you would expect to map it, but it doesn't work

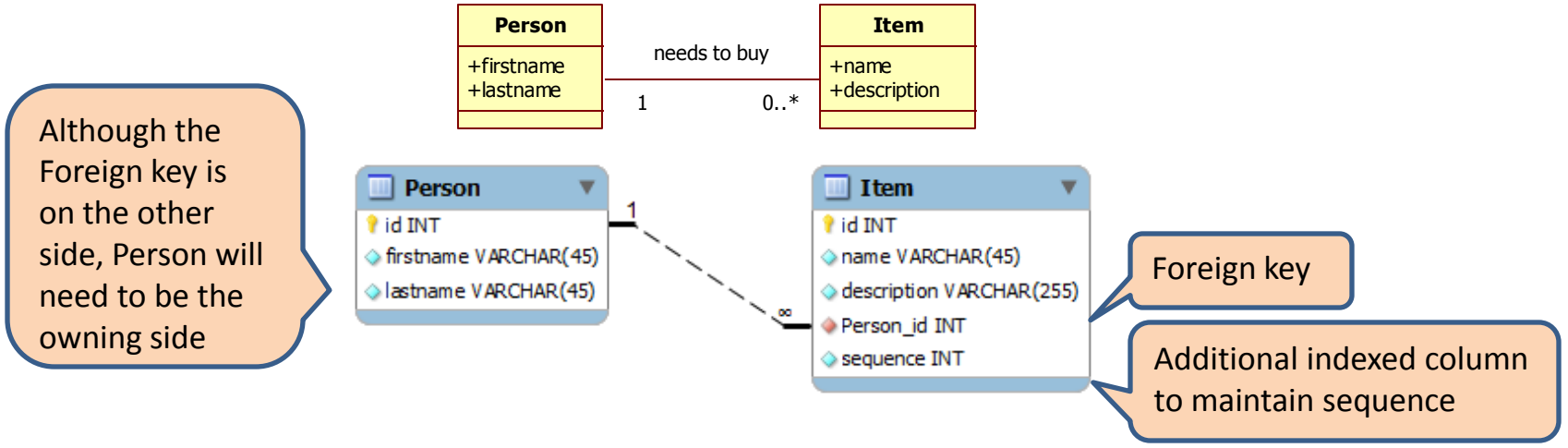
```
@Entity
public class Item {
  @Id
  @GeneratedValue
  private int id;
  private String name;
  private String description;
  @ManyToOne
  private Person buyer;

  ...
```

Normal @ManyToOne

Hibernate List Mapping Issues

- List side needs to be the owning side
- Bi-directional problems:
  - In a bi-directional one to many, the side ***without*** the Foreign Key needs to be the owning side
  - In a bi-directional many to many with two lists, both will have to be 'owner' (2 uni != bi-direct)



One to Many bi-directional List

```java
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany(cascade=CascadeType.PERSIST)
  @JoinColumn(name="buyer_id")
  @OrderColumn(name="sequence")
  private List<Item> shopList = new ArrayList();

  ...
```

@OneToMany is the owning side

@JoinColumn and its name attribute are required (Why?)

```java
@Entity
public class Item {
  @Id
  @GeneratedValue
  private int id;
  private String name;
  private String description;
  @ManyToOne
  @JoinColumn(name="buyer_id", updatable=false, insertable=false)
  private Person buyer;

  ...
```

@ManyToOne does not have a mappedBy attribute

Updateble and insertable false on @JoinColumn to emulate mappedBy

List XML

```
<hibernate-mapping package="list">
  <class name="Person" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="firstname" />
    <property name="lastname" />
    <list name="shopList" access="field" cascade="persist">
      <key column="buyer_id"/>
      <list-index column="sequence"/>
      <one-to-many class="Item"/>
    </list>
  </class>
</hibernate-mapping>
```

<list> is similar to <bag> and <set>

List side is the owning side (no inverse)

Also requires a <list-index> tag not just a <key> and an association tag

```
<hibernate-mapping package="list">
  <class name="Item">
    <id name="id">
      <generator class="native" />
    </id>
    <property name="name" />
    <property name="description" />
    <many-to-one name="buyer" column="buyer_id" class="Person"
        insert="false" update="false" />
  </class>
</hibernate-mapping>
```

<many-to-one> does not have an inverse attribute to specify that it is not the owning side

Uses insert and update false instead of inverse

Collection Mapping

ORDER BY

Order By

- Hibernate can add an 'ORDER BY' SQL clause when retrieving a collection
  - Sets, bags, and maps can be ordered in this way, the order is done by the database
  - Collections mapped as list can not be re-ordered, they already have a specific order

```
@Entity
public class Toolbox {
  @Id
  @GeneratedValue
  private int id;
  private String manufacturer;
  private String model;
  @OneToMany(mappedBy="toolbox", cascade=CascadeType.PERSIST)
  @OrderBy("size ASC")
  private Set<Tool> tools = new HashSet<Tool>();
```

Tools set will be ordered by size ascending

Ordered Bag Example

```
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany(mappedBy="owner", cascade=CascadeType.PERSIST)
  @OrderBy("type DESC")
  private List<Tool> tools = new ArrayList<Tool>();

  ...
```

java.util.List persisted as a bag

Order tools by 'type DESC'

```
@Entity
public class Tool
  @Id
  @GeneratedValue
  private int id;
  private String type;
  private String size;
  @ManyToOne
  private Person owner;

  ...
```

Will be ordered by type desc

```
<hibernate-mapping package="bag_collection">
  <class name="Person" >
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="firstname" />
    <property name="lastname" />
    <bag name="tools" inverse="true" access="field" cascade="persist" order-by="type ASC">
      <key column="owner_id" />
      <one-to-many class="Tool"/>
    </bag>
  </class>
</hibernate-mapping>
```

In XML <bag>, <set>, and <map> can specify the order-by attribute

Collection Mapping

WRAPPING UP

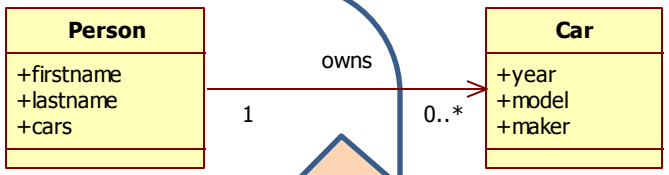
Mapping Tip

- Recommend always creating accessor convenience methods
  - Not just for bi-directional associations

```
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany
  private List<Car> cars = new ArrayList<Car>();

  ...

  public boolean addCar(Car car) { return cars.add(car); }
  public boolean removeCar(Car car) { return cars.remove(car); }
}
```



Uni-directional one-to-many

Uni-directional convenience methods

Active Learning

- What does it mean to have inherent order?

- What are the characteristics of a Map?

Module Summary

- We've covered the following collections:
  - Bags – Allow duplicates, no guaranteed order, but can be ordered by Hibernate
  - Sets – Do not allow duplicates, no guaranteed order, can be ordered by Hibernate
  - Lists – Allow duplicates, a guaranteed order by using an additional index column in the db
  - Maps – Link a set of keys to a bag of values, by having an (additional) key that is used as key
- Default to using bags, use other types as needed

Main Point

- Hibernate can map collections of references as Bags, Sets, Lists and Maps, which can be divided into non-indexed and indexed collections. While most collections can be mapped as a bag, it is important to know about and understand the other types of collections so that they can be used appropriately when needed.
- *Science of Consciousness*: Take the right angle and let go, but the key to finding that angle is having a settled mind (a benefit of TM).