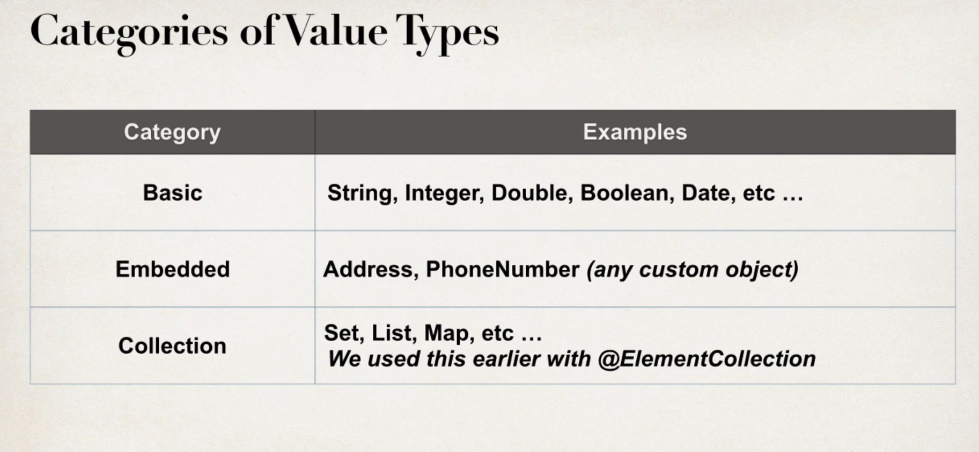
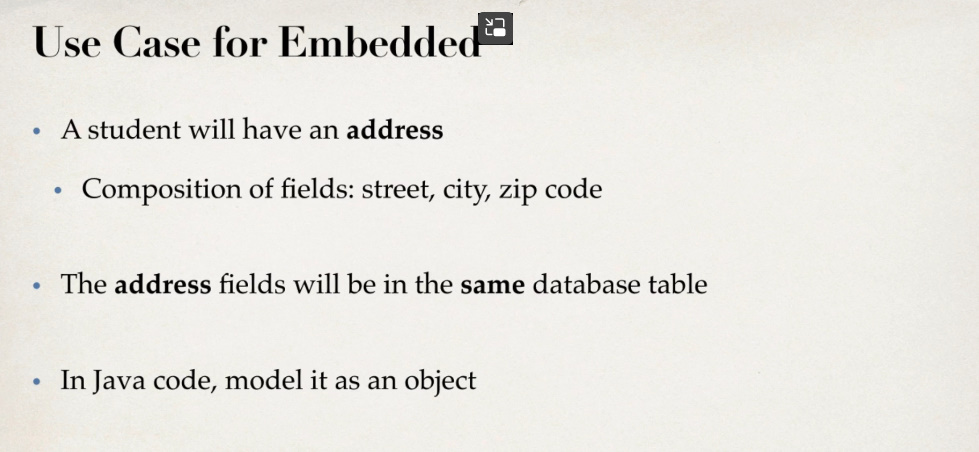
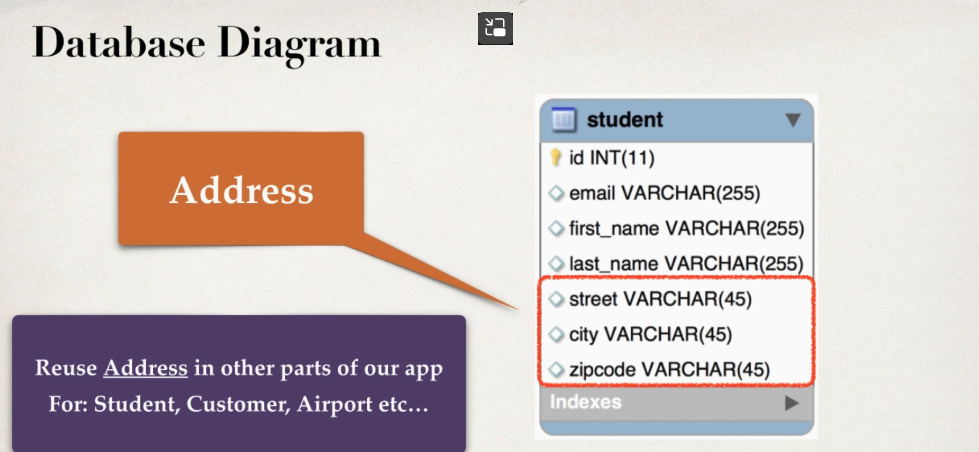


**Categories of value types**

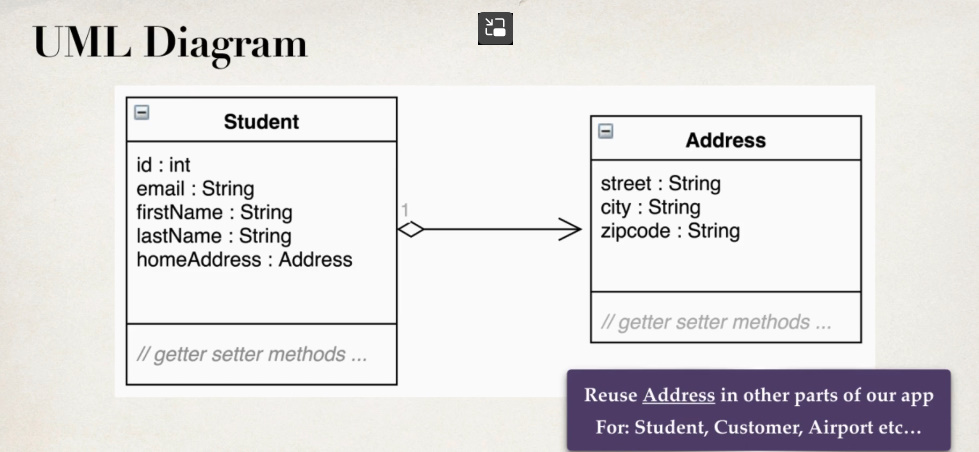


**Embedded Value Type**

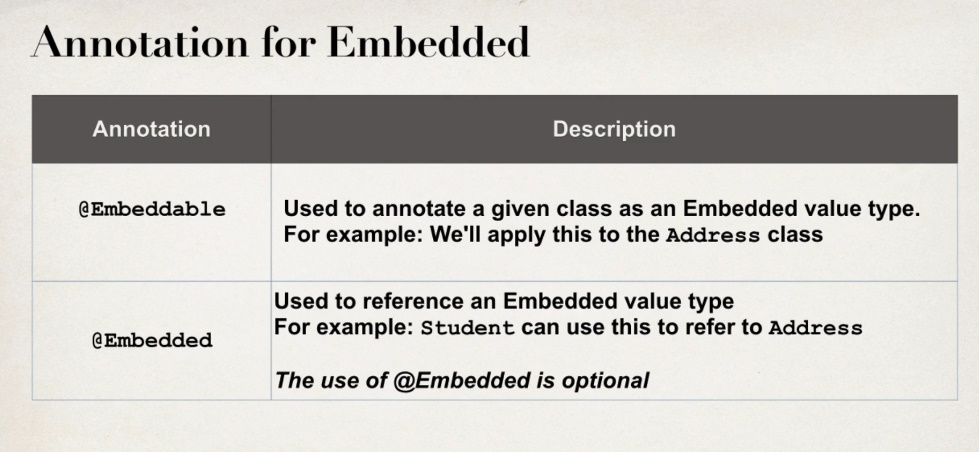
* **Embedded Value Type** este o compozitie de values.
* Embedded Value Type promoveaza refolosirea printre alte entities
* 



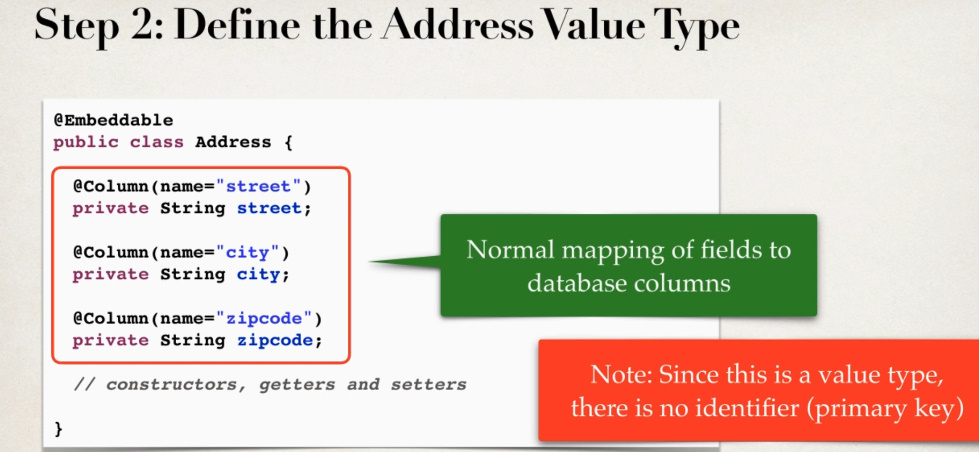
Vedem ca in Tabel avem coloanele street, city, zipcode, si ele sunt Address defapt, dar decat sa le scoatem ca fielduri separate, putem mai bine sa le introducem intr-un obiect Address creat de noi



**Annotation for Embedded**

* 
* **@Embeddable** e pus deasupra la clasa ce va fi folosita pentru a stoca mai multe fielduri
* **@Embedded –** il folosim deja in @Entity pe field, dar e optional. Hibernate si asa isi va da seama

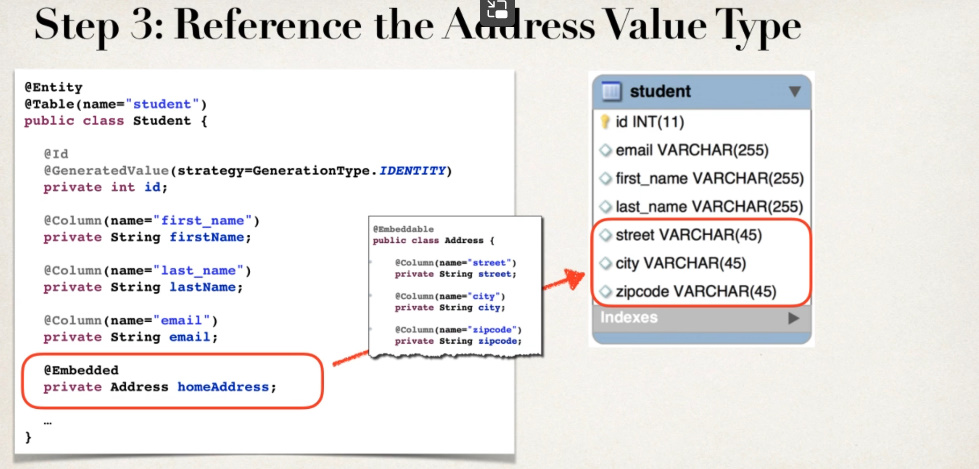
**Crearea la clasa pentru @Embeddable**

****

Anume aici punem **@Embeddable**

* Nu punem @Entity!
* Nu avem @Id!
* Punem doar @Column din tabel

**@Embedded in @Entity**

****

**@Embedded** e optional, caci Hibernate si asa stie ca clasa are @Embeddable7

**Project**

@Embeddable  
@Getter  
@Setter  
@ToString  
public class Address {  
 @Column(name = "city")  
 private String city;  
 @Column(name = "street")  
 private String street;  
 @Column(name = "zipcode")  
 private String zipCode;  
}

@Entity  
@Table(name = "student")  
@Getter  
@Setter  
@ToString  
@NoArgsConstructor  
public class Student {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Column(name = "id")  
 private Integer id;  
 @Column(name = "first\_name")  
 private String firstName;  
 @Column(name = "last\_name")  
 private String lastName;  
 @Column(name = "email")  
 private String email;  
  
 @Embedded  
 private Address address;  
  
 public static class Compare implements Comparator<String>{  
  
 @Override  
 public int compare(String o1, String o2) {  
 return o1.compareTo(o2);  
 }  
 }  
  
 @Builder  
 public Student(String firstName, String lastName, String email, Address address) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.email = email;  
 this.address = address;  
 }  
}

Si vom trimite un Post cu asa payload:

{

    "firstName" : "Eduard",

    "lastName" : "Mititiuc",

    "email" : "test123",

    "address" : {

        "city" : "Dallas",

        "street" : "St. Louis 548",

        "zipCode" : "5A65847"

    }

}

* Address poate fi folosit si in alte @Entity

**@AttributeOverrides**

* Acum, un student pe langa Address, City, Street poate sa mai aiba si atributele work\_addres, work\_city si work\_city. Deci, cam aceleasi atribute, doar ca destinate la work place
* Ar trebui sa cream un nou object identic cu cel creat Address, doar sa modificam valorile din @Column, dar nu ar avea sens
* Pentru a nu face asta, putem pur si simplu sa folosim acelasi Address, doar ca sa-i spunem lui Hibernate ca numele coloanelor vor fi altele:
* @Embedded  
  @AttributeOverrides({  
   @AttributeOverride(name = "city", column = @Column(name = "work\_city")),  
   @AttributeOverride(name = "street", column = @Column(name = "work\_street")),  
   @AttributeOverride(name = "zipCode", column = @Column(name = "work\_zipcode"))  
  }  
  )  
  private Address workAddress;  
  @Embedded  
  private Address address;

name = e numele **fieldurilor** din Address class. **Atentie! A fieldurilor din Address class, nu a coloanelor din tabel!!!**

column – punem care va fi numele la noua coloana ce va trebui sa inlocuiasca pe cea din Address, adica ca si cum suprascriem atributul @Column din Address

* Fie payload pentru Post:

{

    "firstName" : "Eduard",

    "lastName" : "Mititiuc",

    "email" : "test123",

    "address" : {

        "city" : "Dallas",

        "street" : "St. Louis 548",

        "zipCode" : "5A65847"

    },

        "workAddress" : {

        "city" : "San Antonio",

        "street" : "St. Joseph 788",

        "zipCode" : "5B5896C"

    }

}

