CS544

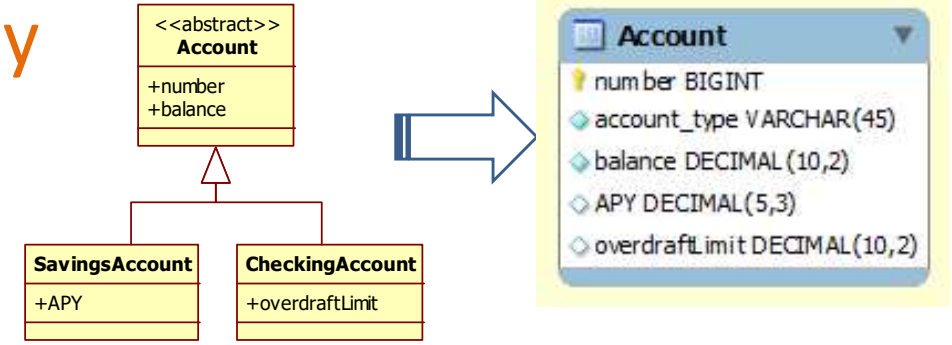
LESSON 5
JPA MAPPING 2

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
March 28 **Lesson 1** Enterprise Architecture introduction and Spring Boot	March 29 **Lesson 2** Dependency injection AOP	March 30 **Lesson 3** JDBC JPA	March 31 **Lesson 4** JPA mapping 1	April 1 **Lesson 5** JPA mapping 2	April 2 **Lesson 6** JPA queries	April 3
April 4 **Lesson 7** Transactions	April 5 **Lesson 8** MongoDB	April 6 **Midterm Review**	April 7 **Midterm exam**	April 8 **Lesson 9** REST webservices	April 9 **Lesson 10** SOAP webservices	April 10
April 11 **Lesson 11** Messaging	April 12 **Lesson 12** Scheduling Events Configuration	April 13 **Lesson 13** Monitoring	April 14 **Lesson 14** Testing your application	April 15 **Final review**	April 16 **Final exam**	April 17
April 18 **Project**	April 19 **Project**	April 20 **Project**	April 21 **Presentations**			

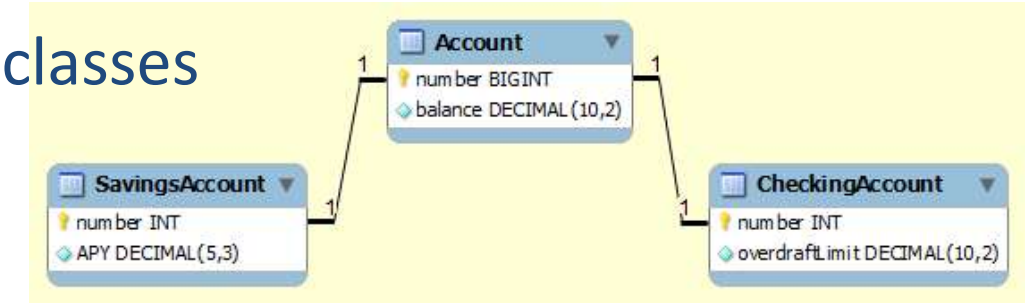
INHERITANCE MAPPING

Three ways to map

- You can map inheritance in one of three ways:
  - Single Table per Hierarchy
    - De-normalized schema
    - Fast polymorphic queries



  - Joined Tables
    - Normalized & similar to classes
    - Slower queries



  - Table per Concrete Class
    - Uses UNION instead of JOIN
    - All needed columns in each table

Single Table

ACCOUNT_TYPE	NUMBER	BALANCE	OVERDRAFTLIMIT	APY
checking	1	500	200	
savings	2	100		2.3
checking	3	23.5	0	

APY is null for checking accounts, overdraft limit is null for savings

+ Simple, Easy to implement
+ Good performance on all queries, polymorphic and non polymorphic

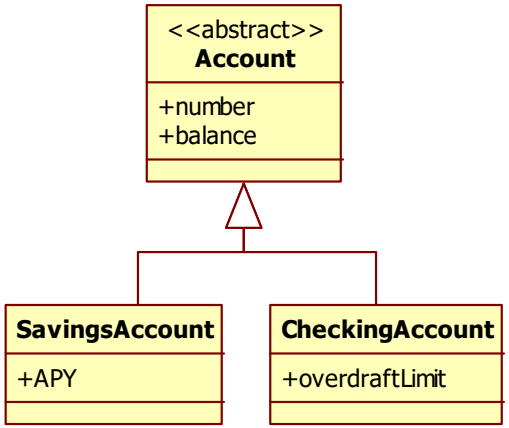
– Nullable columns / de-normalized schema
– Table may have to contain lots of columns
– A change in any class results in a change of this table

Single Table

Specify the SINGLE_TABLE strategy

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(
    name="account_type",
    discriminatorType=DiscriminatorType.STRING
)
public abstract class Account
  @Id
  @GeneratedValue
  private long number;
  private double balance;

  ...
```

Optional annotation @DiscriminatorColumn

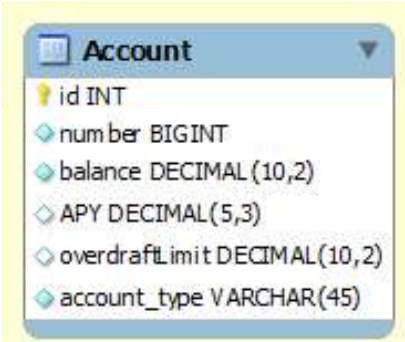



```
@Entity
@DiscriminatorValue("savings")
public class SavingsAccount extends Account {
  private double APY;

  ...
```

Specify discriminator value

```
@Entity
@DiscriminatorValue("checking")
public class CheckingAccount extends Account {
  private double overdraftLimit;

  ...
```

Specify discriminator value



Joined Tables

Account Table

NUMBER	BALANCE
1	500
2	100
3	23.5

SavingsAccount

NUMBER	APY
2	2.3

CheckingAccount

NUMBER	OVERDRAFTLIMIT
1	200
3	0

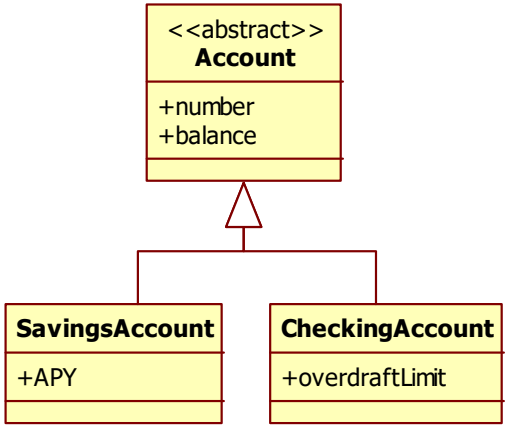
+ Normalized Schema
+ Database view is similar to domain view

– Inserting or updating an entity results in multiple insert or update statements
– Necessary joins can give bad query performance

Joined

Just specify the inheritance strategy, nothing else

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Account {
   @Id
   @GeneratedValue
   private long number;
   private double balance;

   ...
```




```
@Entity
public class SavingsAccount extends Account {
  private double APY;

  ...
```

Subclasses can be mapped as normal entity classes, but without identifiers

```
@Entity
public class CheckingAccount extends Account {
  private double overdraftLimit;

  ...
```

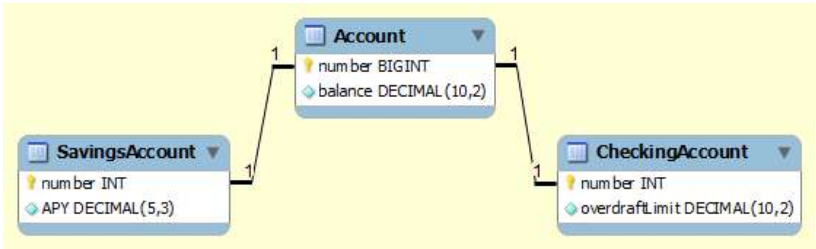


Table per Class

SavingsAccount

NUMBER	BALANCE	APY
2	100	2.3

CheckingAccount

NUMBER	BALANCE	OVERDRAFTLIMIT
1	500	200
3	23.5	0

+ Simple table structure
  + No Null values
+ Very efficient non-polymorphic queries
  + No joins needed

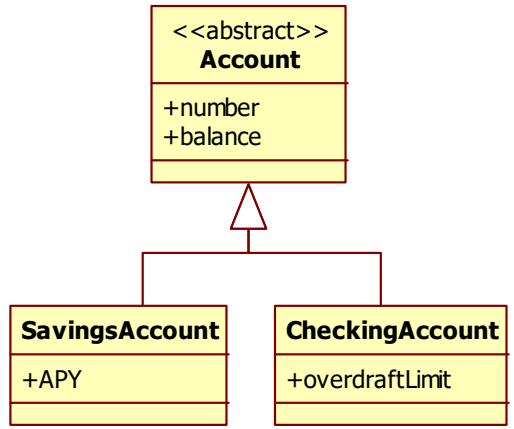
– Can not use Identity column ID generation
– JPA does not require its implementation (optional)
– Requires a UNION for polymorphic queries

Table per Class

Just specify the inheritance strategy, nothing else

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Account {
   @Id
   @GeneratedValue(strategy=GenerationType.TABLE)
   private long number;
   private double balance;

   ...
```

Id generation can not use identity column



Normal @Entity mapping

```
@Entity
public class SavingsAccount extends Account {
   private Double APY;

   ...
```

Java.util.Double instead of primitive double type


```
@Entity
public class CheckingAccount extends Account {
   private Double overdraftLimit;

   ...
```

Java.util.Double instead of primitive double type

Main point

- Class inheritance can be mapped in 3 different ways in the database.

  Science of Consciousness: The transcendental field of pure consciousness is the field of all possibilities.

COMPLEX MAPPING

Complex Mappings

- In this module we will cover:
  - Secondary tables – allow a class to be mapped to multiple tables
  - Embedded classes – allow multiple classes to be mapped to a single table
  - Composite keys – can be made using embedded classes

Secondary Tables

- Last module we used a secondary table to join a table to a single table per hierarchy strategy
- Secondary tables can be used anywhere to move properties into separate table(s)

Secondary table example from last module

```
@Entity
@DiscriminatorValue("savings")
@SecondaryTable(
  name="SavingsAccount",
  pkJoinColumns=@PrimaryKeyJoinColumn(name="number")
)
public class SavingsAccount extends Account {
  @Column(table="SavingsAccount")
  private double APY;

  ...
```

Secondary Table

@SecondaryTables can specify multiple @SecondaryTable

pkJoinColumns can be used to specify a multi column join

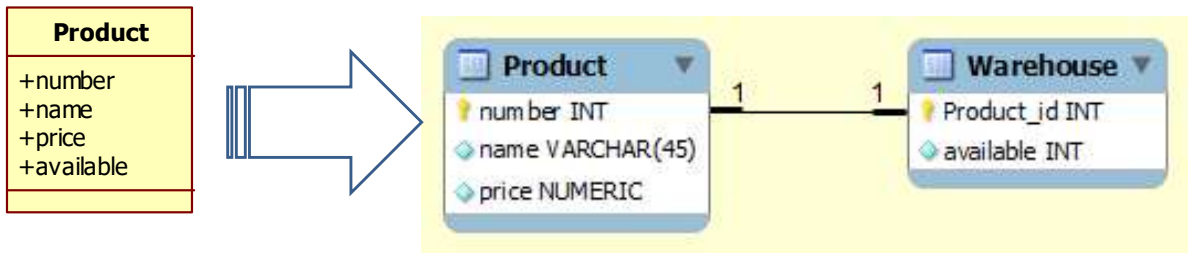
```
@Entity
@SecondaryTables(
  @SecondaryTable(name="warehouse", pkJoinColumns = {
    @PrimaryKeyJoinColumn(name="product_id", referencedColumnName="number")
  }
))
public class Product {
  @Id
  @GeneratedValue
  private int number;
  private String name;
  private BigDecimal price;
  @Column(table="warehouse")
  private boolean available;

  ...
```

JoinColumn name can differ from the referenced column

Properties need to specify the secondary table to be on it

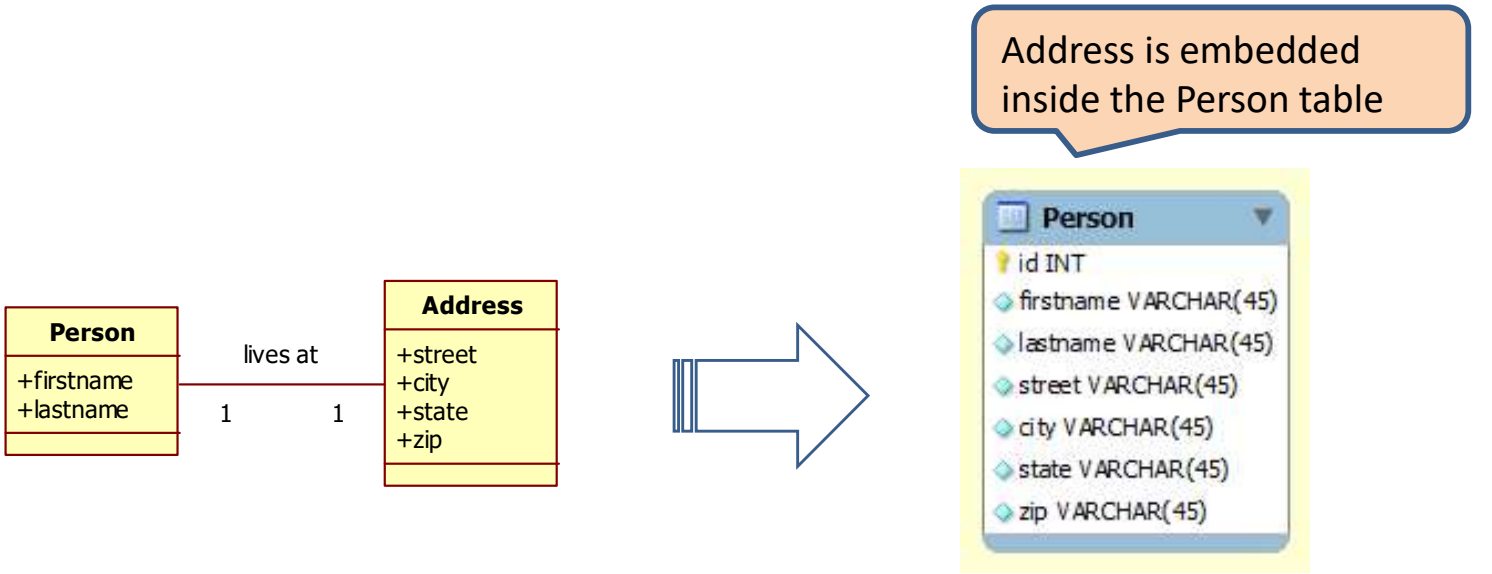
All you really need is @SecondaryTable and a name, the rest is optional

```
@Entity
@SecondaryTable(name="warehouse")
public class Product {
  @Id
  @GeneratedValue
  private int number;
  private String name;
  private BigDecimal price;
  @Column(table = "warehouse")
  private int available;

  ...
```



Embedded Classes

- Combine multiple classes in a single table
- Especially useful for tight associations
- These classes are considered value classes rather than entity classes



Embeddable

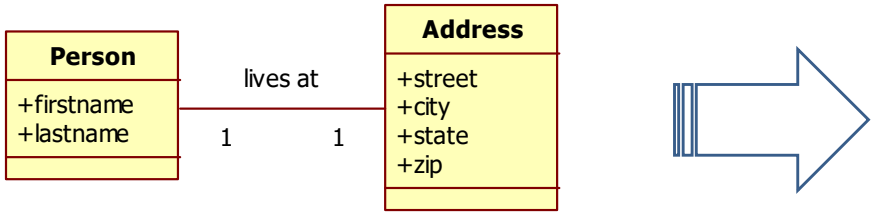
@Embedded annotation is used for embeddable objects

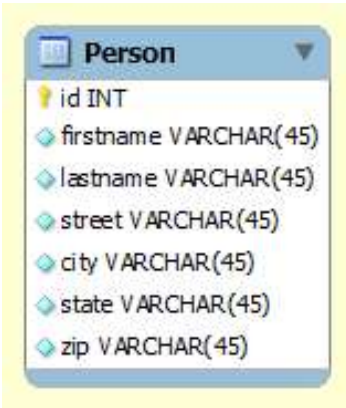
```
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;

  @Embedded
  private Address address;

  ...
```

@Embeddable instead of @Entity

```
@Embeddable
public class Address {
  private String street;
  private String city;
  private String state;
  private String zip;

  ...
```

No @Id in embeddable





ID	FIRSTNAME	LASTNAME	STREET	CITY	STATE	ZIP
1	Frank	Brown	45 N Main St	Chicago	Illinois	51885

Multiple Embedded Addresses

```
@Entity
public class Customer {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;

  @Embedded
  @AttributeOverrides( {
    @AttributeOverride(name="street", column=@Column(name="ship_street")),
    @AttributeOverride(name="city", column=@Column(name="ship_city")),
    @AttributeOverride(name="state", column=@Column(name="ship_state")),
    @AttributeOverride(name="zip", column=@Column(name="ship_zip"))
  })
  private Address shipping;

  @Embedded
  @AttributeOverrides( {
    @AttributeOverride(name="street", column=@Column(name="bill_street")),
    @AttributeOverride(name="city", column=@Column(name="bill_city")),
    @AttributeOverride(name="state", column=@Column(name="bill_state")),
    @AttributeOverride(name="zip", column=@Column(name="bill_zip"))
  })
  private Address billing;

  ...
```

Rename the column names for the embedded object using @AttributeOverrides

ID	FIRSTNAME	LASTNAME	SHIP_STREET	SHIP_CITY	SHIP_STATE	SHIP_ZIP	BILL_STREET	BILL_CITY	BILL_STATE	BILL_ZIP
1	Frank	Brown	45 N Main St	Chicago	Illinois	51885	100 W Adams St	Chicago	Illinois	60603

Composite Keys

- Composite Keys are multi-column Primary Keys
  - By definition these are natural keys
  - Have to be set by the application (not generated)
  - Generally found in legacy systems
  - Also create multi-column Foreign Keys

Composite Ids

@Embeddable

```
@Embeddable
public class Name implements Serializable {
  private String firstname;
  private String lastname;

  ...
```

Also requires hashCode and equals methods (see next slide)

```
@Entity
public class Employee {
  @Id
  private Name name;
  @Temporal(TemporalType.DATE)
  private Date startDate;

  ...
```

Embeddable object as identifier creates composite key

PK is made of Both firstname and lastname

equals() & hashCode()

```
@Embeddable
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 a unique int based on the class contents

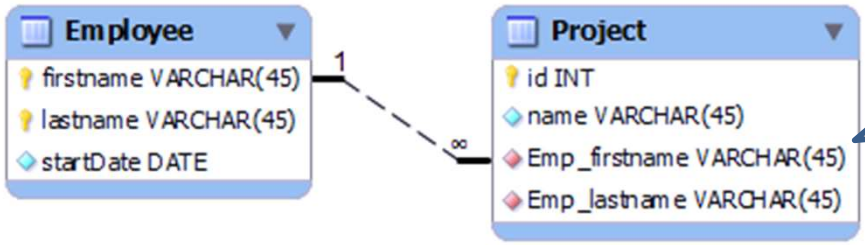
Foreign Keys to Composite Ids

```
@Entity
public class Employee {
  @Id
  private Name name;
  @Temporal(TemporalType.DATE)
  private Date startDate;
  @OneToMany(mappedBy = "owner")
  private List<Project> projects = new ArrayList<Project>();

  ...
```

Same Name embeddable @Id as before

Normal mappedBy on this side

```
@Entity
public class Project {
  @Id
  @GeneratedValue
  private int id;
  private String name;
  @ManyToOne
  @JoinColumns( {
    @JoinColumn(name = "Emp_firstname", referencedColumnName = "firstname"),
    @JoinColumn(name = "Emp_lastname", referencedColumnName = "lastname")
  })
  private Employee owner;

  ...
```

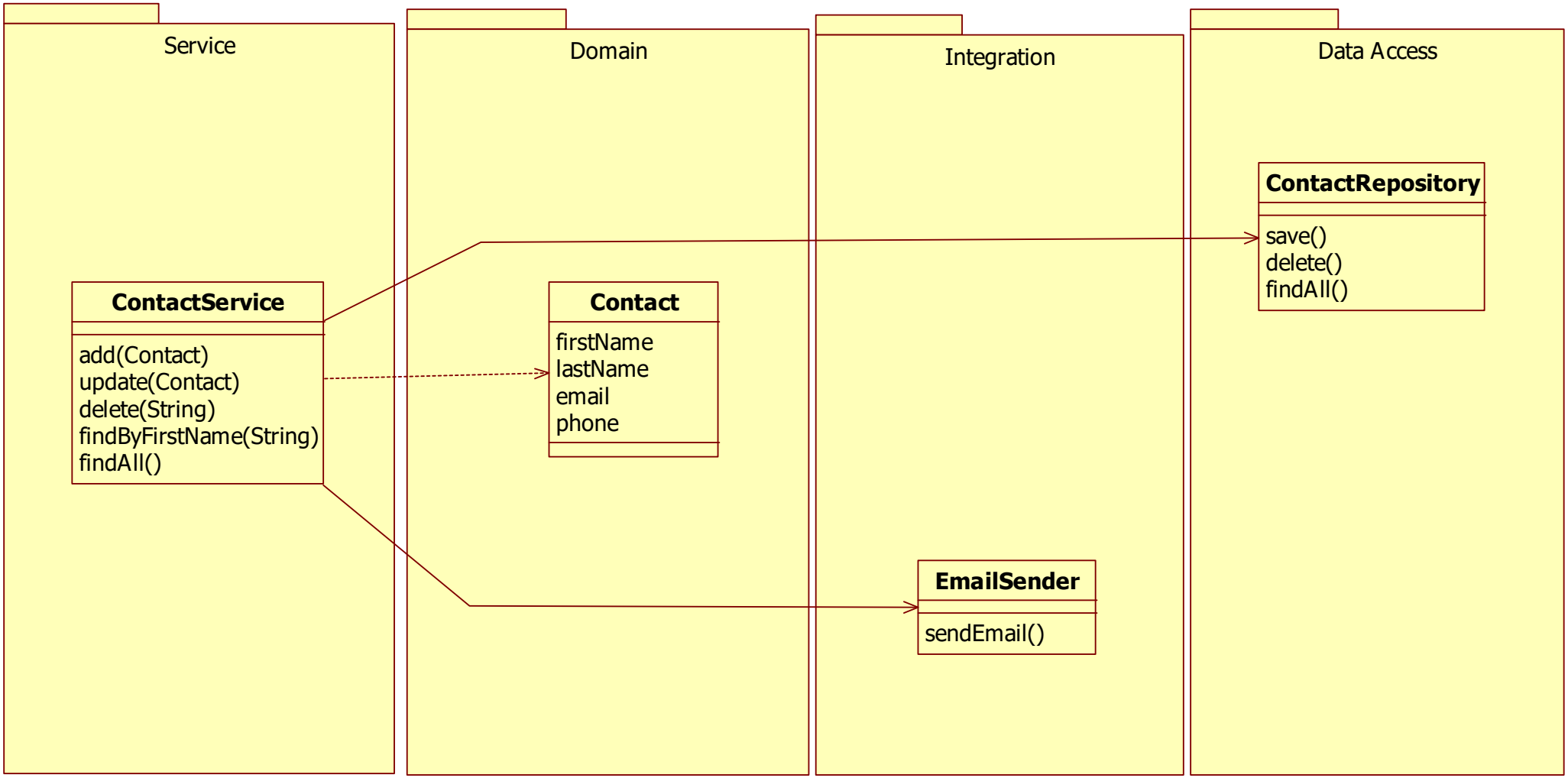


Two column Foreign Key

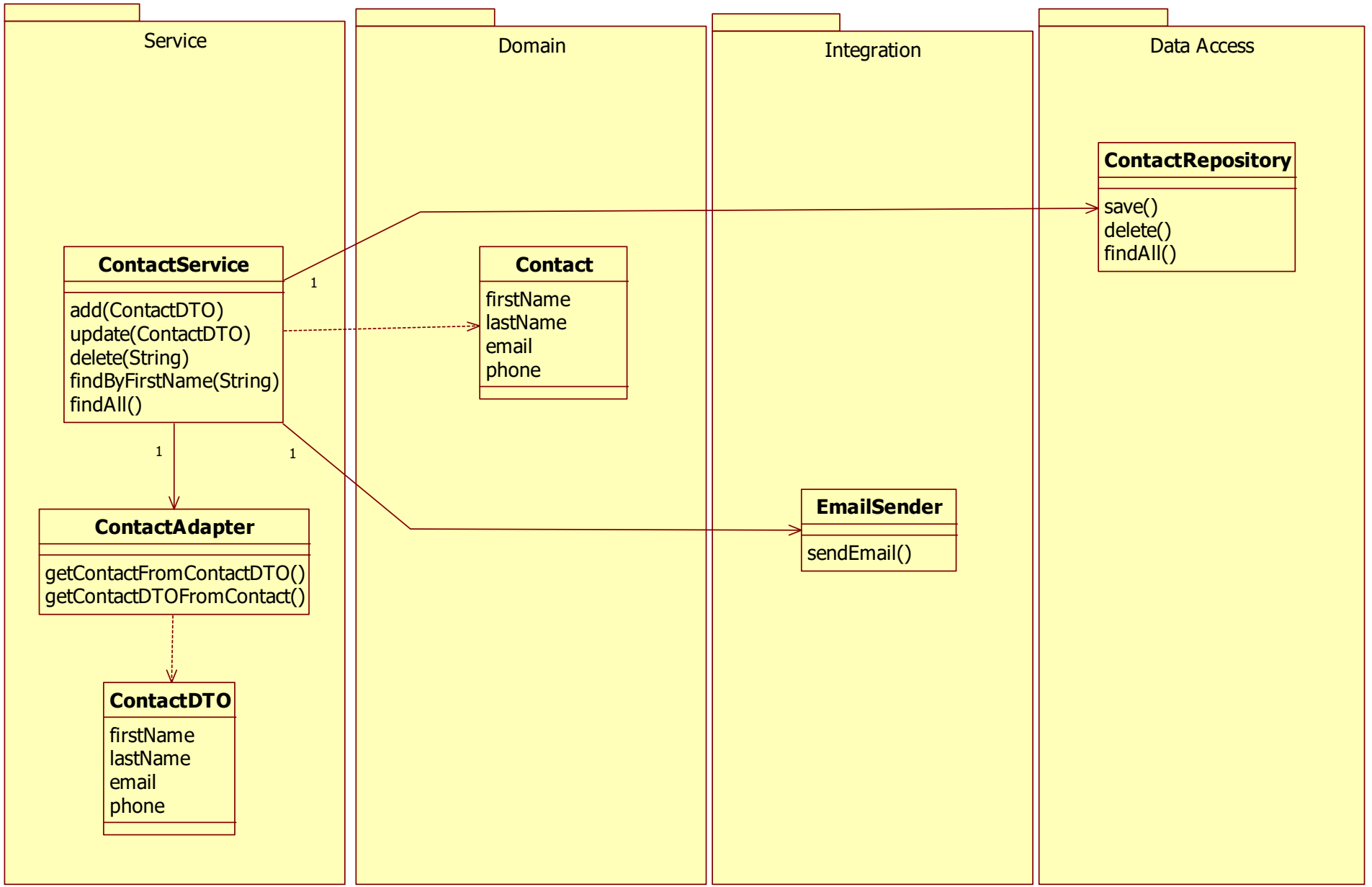
Two column FK specification

DATA TRANSFER OBJECTS (DTO)

What does findByFirstName return?



Data Transfer Objects (DTO)



Main point

- Using DTO's gives loose coupling through information hiding.

  Science of Consciousness: Through the daily practice of TM one gets more and more access to the intelligence of creation.