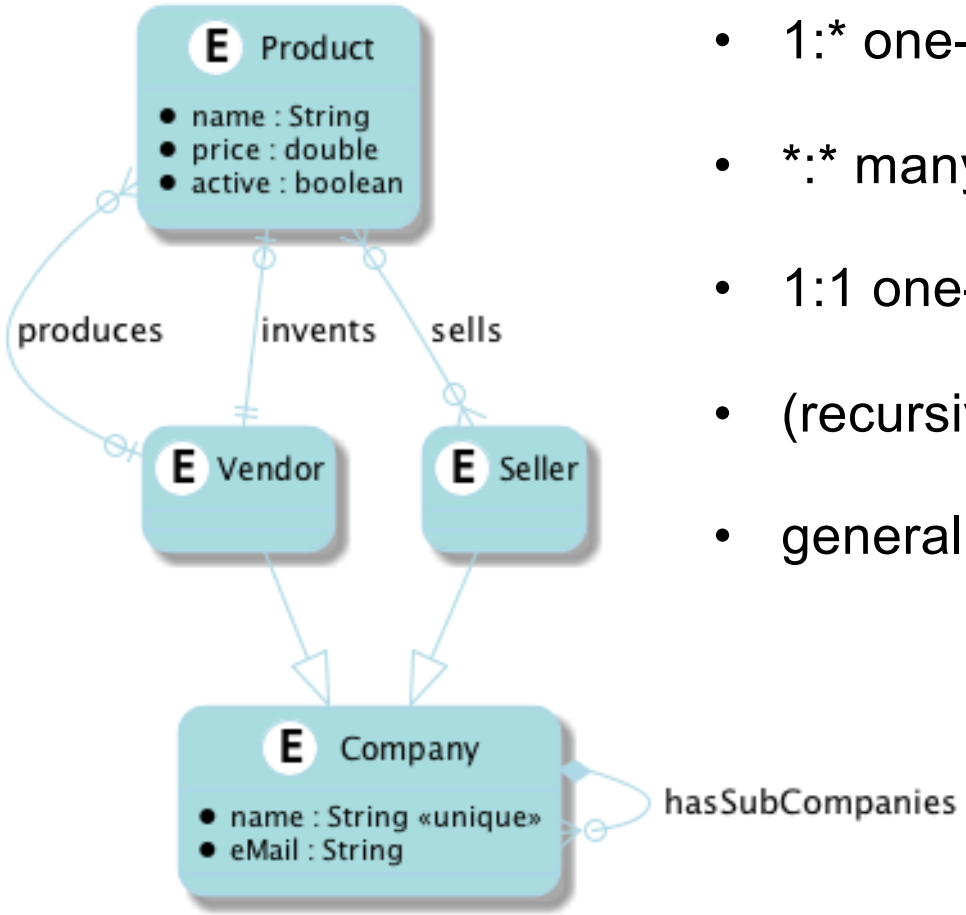
O/R-Mapping by Spring Data JPA

Given, a Conceptual Model

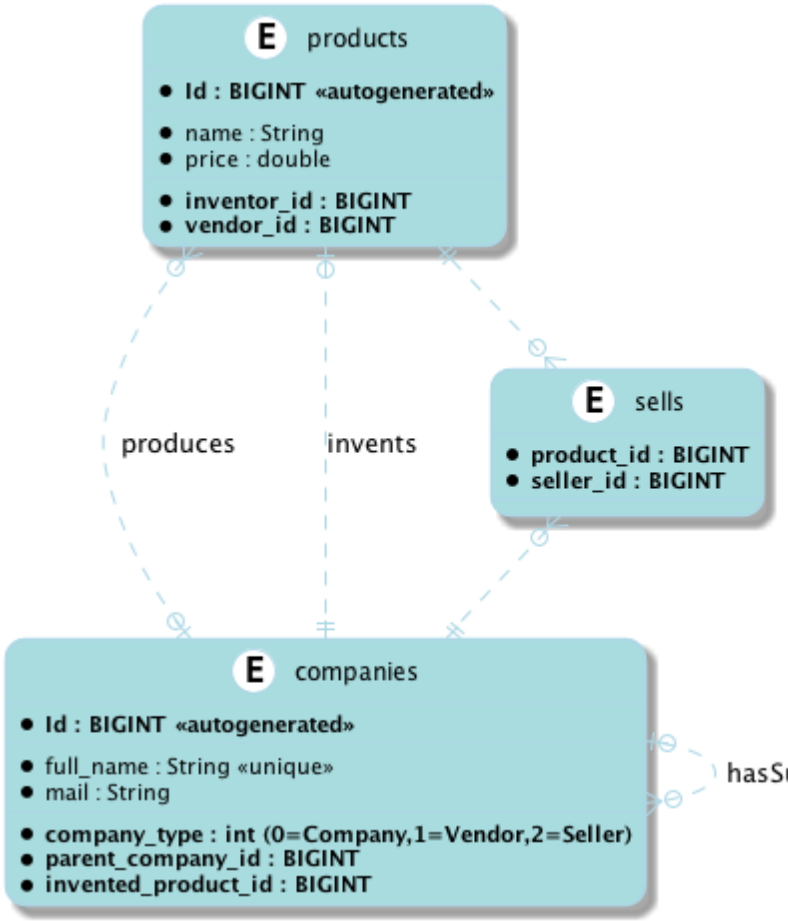
in form of an EER diagram where a customer just knows identifying attributes for real-world instances, i.e. entities



- 1:* one-to-many “produces”
- *:* many-to-many “sells”
- 1:1 one-to-one “invents” – very particular here ;-)
- (recursive) aggregation “hasSubCompanies”
- generalisation/specialisation

to generate its Logical Relational Model by JPA

with primary and foreign keys, but neither generalisation/specialisation, nor aggregation nor many-to-many relationship




```
SELECT * FROM db.products;
```

id	name	price	inventor_id	vendor_id
1	iPhone	10.9	3	3
2	MacBook	12.3	3	3
3	Surface	9.9	6	6
4	computer	399	3	3
5	notebook	499.9	3	6

```
SELECT * FROM db.sells;
```

seller_id	product_id
1	1
2	1
6	1
1	2

```
SELECT * FROM db.companies;
```

company_type	id	mail	full_name	parent_company_id
0	1	office@google.com	Google	NULL
2	2	office@amazon.com	Amazon	NULL
1	3	office@apple.com	Apple	NULL
0	4	info@ibm.com	IBM	1
2	5	info@e4y.com	Electronic4You	3
1	6	info@microsoft.com	Microsoft	5

Solve Generalisation/Specialisation

HTL Villach
Future Inside

choosen approach “Single Table” out of several ones since here we have totally, exclusive/disjoint sub classes

```
@Entity                    // Hibernate JPA entity class
@Table(name = "companies")     // make a table for it
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "company_type",
        discriminatorType = DiscriminatorType.INTEGER)
@DiscriminatorValue("0")
@Data @NoArgsConstructor @AllArgsConstructor @Builder
public class Company implements Serializable {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

```
    public Company(String name, String eMail) {      // needed
        this.name = name;                  // for @Builder only in
        this.eMail = eMail;              // sub classes of Company
    }
}
```

Generalisation/Specialisation – cont'd.

HTL Villach
Future Inside

```java
@Entity
@DiscriminatorValue("1")
@Data @NoArgsConstructor @AllArgsConstructor
public class Vendor extends Company {

    @Builder(builderMethodName = "buildVendor")
    public Vendor(String name, String eMail) { super(name, eMail); }
}
```

```java
@Entity
@DiscriminatorValue("2")
@Data @NoArgsConstructor @AllArgsConstructor
public class Seller extends Company {

    @Builder(builderMethodName = "buildSeller")
    public Seller(String name, String eMail) { super(name, eMail); }
}
```

software inside

Create Objects of Super-/Sub-Classes

HTL Villach
Future Inside

```
                                // create companies and products using lomboks builder pattern
Company c  = Company.builder().name("Google").eMail("office@google.com").build();
Seller  s  = Seller.buildSeller().name("Amazon").eMail("office@amazon.com").build();
Vendor  v  = Vendor.buildVendor().name("Apple").eMail("office@apple.com").build();
Vendor  v2 = Vendor.buildVendor().name("Microsoft").eMail("info@microsoft.com").build();
Seller  s2 = Seller.buildSeller().name("Electronic4You").eMail("info@e4y.com").build();
Company c2 = Company.builder().name("IBM").eMail("info@ibm.com").parentCompany(c).build();
s2.setParentCompany(v);                 // consider part-of and generalisation/specialisation
v2.setParentCompany(s2);
s.setParentCompany(s);                  // s is sub company of itself - endless recursion ???
cRepo.save(c);
```

1 Vendor produces * Products, 1:*

HTL Villach
Future Inside

1:* is propably the most relevant relationship for RDBMs; here we have choosen a bi-directional approach for O/R-mapping

```java
public class Vendor extends Company {

    // at owner side of 1:* we configure details, eg mapped onto property "vendor"
    @OneToMany(mappedBy = "vendor", orphanRemoval = true, cascade = CascadeType.ALL)
    private Set<Product> products;          // produced by this vendor
```

```java
public class Product implements Serializable {

    // at target side of 1:* we only provide a field to be mapped
    @ManyToOne
    private Vendor vendor; // the only one vendor of this product
```

software inside

Aggregation is 1:*, even when recursive

HTL Villach
Future Inside

```
public class Company implements Serializable {

    // recursive 1:* (f)or part-of
    @OneToMany(mappedBy="parentCompany", orphanRemoval = true, cascade = CascadeType.ALL)
    private Set<Company> hasSubCompanies;
    @ManyToOne                  // (optional = true) per default
    private Company parentCompany;
```

software inside

* Sellers sells * Products, *:*

HTL Villach
Future Inside

```java
public class Seller extends Company {

    // at owner side of a *:* we configure details
    @ManyToMany         // products sold by this vendor
    @JoinTable(name = "sells",
            joinColumns = @JoinColumn(name = "seller_id"),
            inverseJoinColumns = @JoinColumn(name = "product_id"))
    private Set<Product> sellsProducts;
```

```java
public class Product implements Serializable {

    // at target side of *:* we only provide a field name to be mapped
    @ManyToMany(mappedBy = "sellsProducts")
    private Set<Seller> soldByCompanies; // vendors selling this product
```

1 Vendor invents 1 Product, 1:1 very perticular here

HTL Villach
Future Inside

```java
public class Vendor extends Company {

    // at owner side of 1:1 we configure details
    @OneToOne(mappedBy = "inventor", orphanRemoval = true, cascade = CascadeType.ALL)
    private Product inventedProduct;
```

```java
public class Product implements Serializable {

    // at target side of 1:1 we provide a field name to be mapped
    @OneToOne
    private Vendor inventor;
```