Assignment 5

- **Bidirectional @OneToOne**
- **Bidirectional @OneToMany**

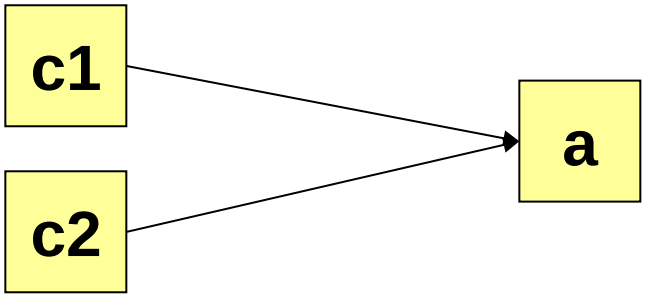
1) Bidirectional OneToOne

```
Customer c1 = new Customer("Lee", 44);
Customer c2 = new Customer("Gosling", 55);
Address a = new Address("Infinite Loop 1", "Cupertino");
c1.setAddress(a);
c2.setAddress(a);

em.persist(a);
em.persist(c1);
em.persist(c2);
```

- em.persist(a) is not necessary if *cascade=CascadeType.PERSIST* is defined on address association
- Not a 1:1 association
- Inconsistency in DB as well

ID	AGE	NAME	ADDRESS_ID
7	44	Lee	4
8	55	Gosling	4



1) Bidirectional OneToOne

```
public void setAddress(Address address) {
   if(this.address != null){
      this.address.setCustomer(null);
   }
   this.address = address;
   if(this.address != null){
      this.address.setCustomer(this);
   }
}
```

```
public void setCustomer(Customer c) {
   if(this.customer != null){
      this.customer.setAddress(null);
   }
   this.customer = c;
   if(this.customer != null){
      this.customer.setAddress(this);
   }
}
```

- **c1.setAddress(a);**

  => Leads to an infinite loop

1) Bidirectional OneToOne

```
public void setAddress(Address address) {
   if(this.address != address){
      if(this.address != null){
         this.address.setCustomer(null);
      }
      this.address = address;
      if(this.address != null){
         this.address.setCustomer(this);
      }
   }
}
```

```
public void setCustomer(Customer customer) {
   if(this.customer != customer){
      if(this.customer != null){
         this.customer.setAddress(null);
      }
      this.customer = customer;
      if(this.customer != null){
         this.customer.setAddress(this);
      }
   }
}
```

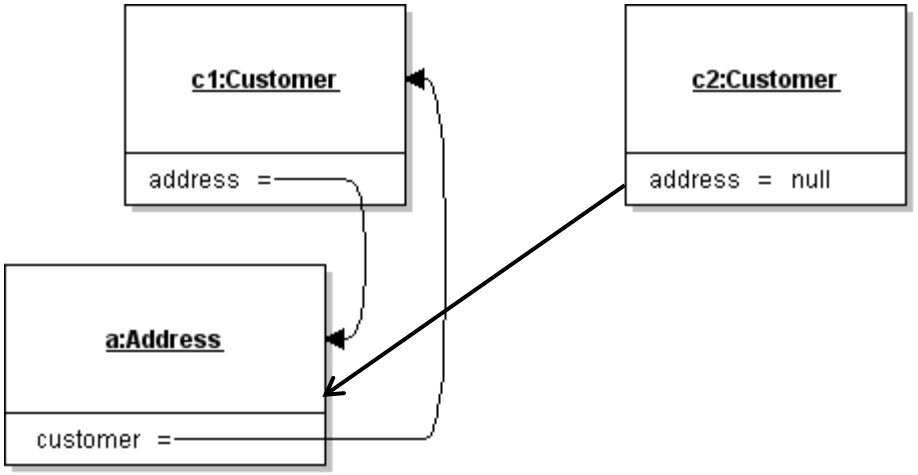
1) Bidirectional OneToOne

- **c1.setAddress(a);**
- **c2.setAddress(a);**
  - c2.address = a
  - a.setCustomer(c2)
    - c1.setAddress(null)
      - a.setCustomer(null)
        - c1.setAddress(null)
          - …

=> Leads to an infinite loop with the calls

c1.setAddress(null)

a.setCustomer(null)



1) Bidirectional OneToOne

```
public void setAddress(Address address) {
   if(this.address != address){
      Address oldAddress = this.address;
      this.address = address;
      if(oldAddress != null){
         oldAddress.setCustomer(null);
      }
      if(this.address != null){
         this.address.setCustomer(this);
      }
   }
}
```

- **c1.setAddress(a);**
  c2.setAddress(a);

  => All references are set to null

1) Bidirectional OneToOne

```
public void setAddress(Address address) {
   if(this.address != address){
      Address oldAddress = this.address;
      this.address = address;
      if(oldAddress != null){
         oldAddress.setCustomer(null);
      }
      if(address != null){
         address.setCustomer(this);
      }
   }
}
```

- **Seems to work...**
  - until someone finds a counter example

1) Bidirectional OneToOne

```
private transient boolean setting = false;
public void setAddress(Address address) {
   if( !setting ){
      setting = true;
      if(this.address != address ){ // optimization only
         if(this.address != null)
            this.address.setCustomer(null);
         this.address = address;
         if(this.address != null)
            this.address.setCustomer(this);
      }
      setting = false;
   }
}
```

- **Avoids reentrant calls**
  - Problem: is not thead-safe!
    => Variant: use ThreadLocal to store the *setting* flag

Assignment 5

- **Bidirectional @OneToOne**
- **Bidirectional @OneToMany**

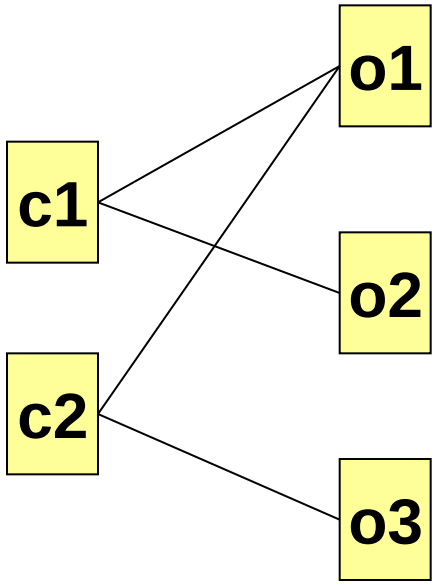
2) Bidirectional OneToMany

```
Customer c1 = new Customer("Haller", 52);
Customer c2 = new Customer("Schneider", 66);
Order o1 = new Order(),
Order o2 = new Order();
Order o3 = new Order();

c1.addOrder(o1);
c1.addOrder(o2);

c2.addOrder(o1);
c2.addOrder(o3);

em.persist(c1);
em.persist(c2);
```



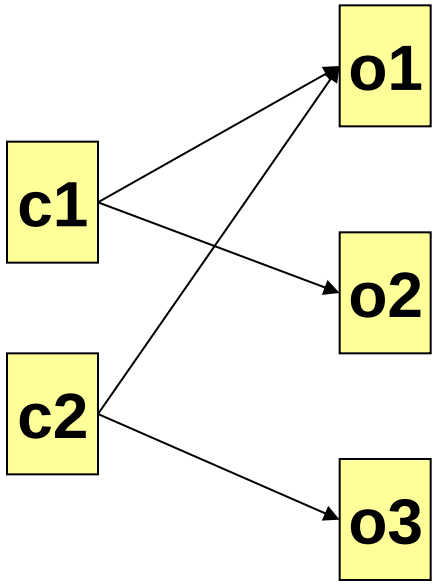
- Assumption: cascade=CascadeType.*PERSIST*

2) Bidirectional OneToMany

```
Customer:
   @OneToMany(mappedBy = "customer",
                cascade=CascadeType.PERSIST)
   private List<Order> orders
                = new ArrayList<Order>();

   public void addOrder(Order o){
      orders.add(o);
   }
```

```
Order:
   @ManyToOne
   private Customer customer
```



ID	CUSTOMER_ID
4	
5	
6	

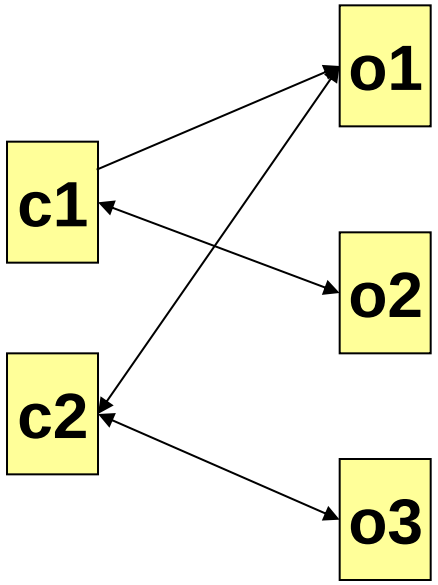
2) Bidirectional OneToMany

```
Customer:
   @OneToMany(mappedBy = "customer",
                cascade=CascadeType.PERSIST)
   private List<Order> orders
                = new ArrayList<Order>();

   public void addOrder(Order o){
      orders.add(o);
      o.setCustomer(this);
   }
```

```
Order:
   @ManyToOne
   private Customer customer;

   public void setCustomer(Customer c){
      this.customer = c;
   }
```



ID	CUSTOMER_ID
10	20
11	19
12	20

2) Bidirectional OneToMany

- **Customer**

```
public void addOrder(Order order) {
   if(orders.add(order)) order.setCustomer(this);
}
public void removeOrder(Order o){
   if(orders.remove(o)) o.setCustomer(null);
}
```

Breaks recursion if it is a set and not a list, otherwise not!

- **Order**

```
public void setCustomer(Customer customer){
   if(this.customer != null)
         this.customer.removeOrder(this);
   this.customer = customer;
   if(this.customer != null)
         this.customer.addOrder(this);
}
```

2) Bidirectional OneToMany

- **Customer**

```
public void addOrder(Order o){
   o.setCustomer(this);
}
public void removeOrder(Order o){
   o.setCustomer(null);
}
```

- **Order**

```
public void setCustomer(Customer customer){
   if(this.customer != null)
         this.customer.getOrders().remove(this);
   this.customer = customer;
   if(this.customer != null)
         this.customer.getOrders().add(this);
}
```

Provided that getOrders does not return null

Provided that getOrders does not return a copy or an immutable collection.

2) Bidirectional OneToMany

- **Customer**

```
public Collection<Order> getOrders() {
  return Collections.unmodifiableList(this.orders);
}
void addOrder(Order o){ this.orders.add(o); }
void removeOrder(Order o){ this.orders.remove(o); }
```

Prevents that orders are added over the getter

Not public!

- **Order**

```
public void setCustomer(Customer customer){
   if(this.customer != null)
         this.customer.removeOrder(this);
   this.customer = customer;
   if(this.customer != null)
         this.customer.addOrder(this);
}
```

Automatic Consistency

- **Advantage**
  - Consistent objects, and as a consequence consistency in the DB
  - Easy to use

- **Disadvantage**
  - Runtime-overhead, even if consistency is not needed (e.g. as transaction commits)
  - Setters do more than what is expected from a setter
  - JPA: Field access is needed