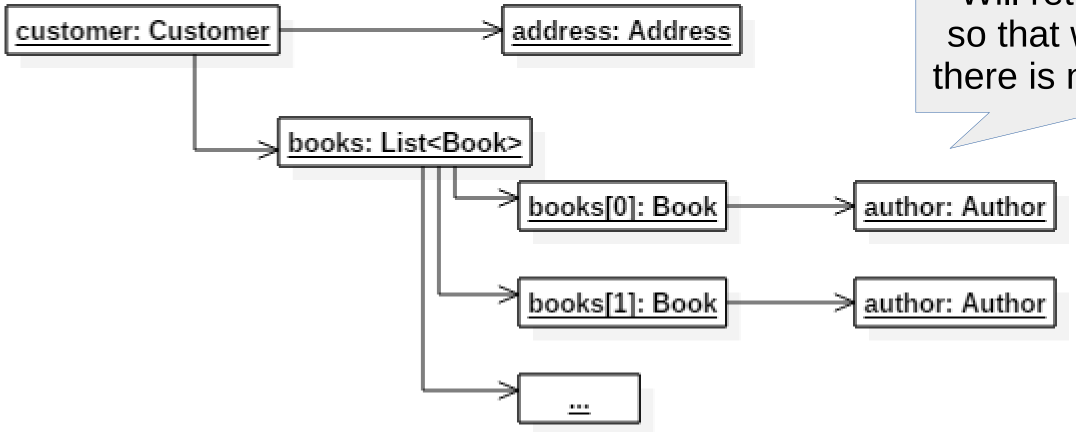
CS544 EA

Hibernate

Optimization: Entity Graph

Entity Graph

- Added in JPA 2.1 (most recent)
  - Specify a **Graph of connected Entities** to retrieve
  - Example: When retrieving a customer we also want to get his address, and all the books he bought, and the author of each of those books



Will retrieve all of this so that when traversed there is no Lazy Loading

Domain

```java
Customer cust1 = new Customer("Frank", "Brown");
Customer cust2 = new Customer("Jane", "Terrien");
Customer cust3 = new Customer("John", "Doe");
cust1.addBook(
      new Book("Harry Potter and the Deathly Hallows",
            new Author("J.K. Rowlings")));
cust1.addBook(
      new Book("Unseen Academicals (Discworld)",
            new Author("Terry Pratchett")));
cust1.addBook(
      new Book("The Color of Magic (Discworld)",
            new Author("Terry Pratchett")));
cust2.addBook(
      new Book("Twilight (The Twilight Saga, Book1)",
            new Author("Stephenie Meyer")));
cust1.setAddress(new Address("Fairfield", "Iowa"));
cust2.setAddress(new Address("Chicago", "Illinois"));
em.persist(cust1);
em.persist(cust2);
em.persist(cust3);
```

```java
@Entity
public class Customer {
      @Id
      @GeneratedValue
      private Long id;
      private String firstName;
      private String lastName;
      @OneToOne(cascade=CascadeType.ALL)
      private Address address;
      @OneToMany(cascade=CascadeType.ALL)
      @JoinColumn
      private List<Book> books = new ArrayList<>();
```

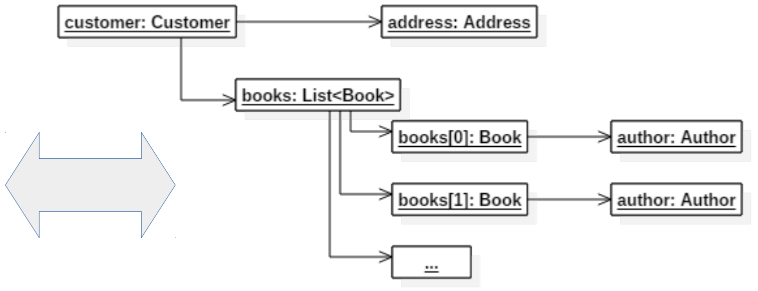
```java
@Entity
public class Book {
      @Id
      @GeneratedValue
      private Long id;
      private String name;
      @OneToOne(cascade=CascadeType.ALL)
      private Author author;
```

```java
@Entity
public class Address {
      @Id
      @GeneratedValue
      private Long id;
      private String city;
      private String state;
```

EntityGraph

- The purpose of the entity graph is:
  - To indicate which references should change to load eagerly (in a query or .find())
  - AttributeNodes specify attributes / references
  - SubGraph can be used to go into other Entities

```
EntityGraph<Customer> graph =
     em.createEntityGraph(Customer.class);
graph.addAttributeNodes("address");
graph.addSubgraph("books").addAttributeNodes("author");
```



.createQuery()

```java
EntityGraph<Customer> graph =
em.createEntityGraph(Customer.class);
graph.addAttributeNodes("address");
graph.addSubgraph("books").addAttributeNodes("author");

TypedQuery<Customer> query = em.createQuery(
     "from Customer where firstName like :name",
     Customer.class);
query.setParameter("name", "J%");
query.setHint("javax.persistence.fetchgraph", graph);

List<Customer> customers = query.getResultList();
System.out.println(customers.size());
```

```
Hibernate:
    select
        customer0_.id as id1_3_0_,
        customer0_.address_id as address_4_3_0_,
        customer0_.firstName as firstNam2_3_0_,
        customer0_.lastName as lastName3_3_0_,
        address1_.id as id1_0_1_,
        address1_.city as city2_0_1_,
        address1_.state as state3_0_1_,
        books2_.books_id as books_id4_2_2_,
        books2_.id as id1_2_2_,
        books2_.id as id1_2_3_,
        books2_.author_id as author_i3_2_3_,
        books2_.name as name2_2_3_,
        author3_.id as id1_1_4_,
        author3_.name as name2_1_4_
    from
        Customer customer0_
    left outer join
        Address address1_
            on customer0_.address_id=address1_.id
    left outer join
        Book books2_
            on customer0_.id=books2_.books_id
    left outer join
        Author author3_
            on books2_.author_id=author3_.id
    where
        customer0_.id=?
2
```

The EntityGraph is passed as a query Hint

Hibernate loads the entire graph into cache While returning the Customer as query result

.find()

```
EntityGraph<Customer> graph
     = em.createEntityGraph(Customer.class);
graph.addAttributeNodes("address");
graph.addSubgraph("books").addAttributeNodes("author");

Map<String, Object> properties = new HashMap<>();
properties.put("javax.persistence.fetchgraph", graph);

em.find(Customer.class, 1L, properties);
```

Hints are passed as properties Map to .find()

```
Hibernate: 
    select
        customer0_.id as id1_3_0_,
        customer0_.address_id as address_4_3_0_,
        customer0_.firstName as firstNam2_3_0_,
        customer0_.lastName as lastName3_3_0_,
        address1_.id as id1_0_1_,
        address1_.city as city2_0_1_,
        address1_.state as state3_0_1_,
        books2_.books_id as books_id4_2_2_,
        books2_.id as id1_2_2_,
        books2_.id as id1_2_3_,
        books2_.author_id as author_i3_2_3_,
        books2_.name as name2_2_3_,
        author3_.id as id1_1_4_,
        author3_.name as name2_1_4_ 
    from
        Customer customer0_ 
    left outer join
        Address address1_ 
            on customer0_.address_id=address1_.id 
    left outer join
        Book books2_ 
            on customer0_.id=books2_.books_id 
    left outer join
        Author author3_ 
            on books2_.author_id=author3_.id 
    where
        customer0_.id=?
```

Hibernate loads the entire graph into cache giving us the Root (Customer) entity

Entity Graph and N+1

- An entity graph can be a solution to N+1
  - Load all the needed entities in one query

- Potential problems:
  - You can not make more than one collection eager
  - Eager associations from your graph towards other entities (outside your graph) **still cause N+1** (see eager references N+1)

If you have references going out
make sure they're lazy!