Hands-On: Using CriteriaBuilder and Criteria Query with JPA in Java

Prerequisites:

- Basic knowledge of Java and object-oriented programming
- Familiarity with JPA and databases
- An IDE such as Eclipse or IntelliJ IDEA

We will be using the following entities for this lab:

```
    @Entity
    public class Book {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;
        private String title;
        private String author;
        private int year;
        // constructors, getters, setters, etc.
    }

    @Entity
    public class User {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;
        private String name;
        private int age;
        // constructors, getters, setters, etc.
}
```

1. Setting up the EntityManager

First, we need to create an EntityManager to interact with the database. We can do this by injecting the EntityManagerFactory into our class and using it to create an EntityManager:

```
@Stateless
public class MyService {
    @PersistenceUnit(unitName = "MyPersistenceUnit")
    private EntityManagerFactory entityManagerFactory;

    public EntityManager getEntityManager() {
        return entityManagerFactory.createEntityManager();
    }
}
```

2. Creating records

To create a new record in the database, we first need to create an instance of the entity and set its properties. Then, we can use the EntityManager to persist the entity:

```
public void createBook(String title, String author, int year) {
    EntityManager em = getEntityManager();
```

```
    Book book = new Book();
    book.setTitle(title);
    book.setAuthor(author);
    book.setYear(year);

    em.getTransaction().begin();
    em.persist(book);
    em.getTransaction().commit();

    em.close();
}
```

3. Reading records

To read records from the database, we can use CriteriaQuery to create a query that will return the desired records. We start by creating a CriteriaBuilder, which is used to construct the query:

```
public List<Book> getBooksByAuthor(String author) {
    EntityManager em = getEntityManager();

    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<Book> query = cb.createQuery(Book.class);

    Root<Book> root = query.from(Book.class);
    query.select(root).where(cb.equal(root.get("author"), author));

    TypedQuery<Book> typedQuery = em.createQuery(query);

    List<Book> result = typedQuery.getResultList();

    em.close();

    return result;
}
```

This method returns a list of Book objects whose author property matches the specified value.

4. Updating records

To update a record in the database, we first need to retrieve the entity using its ID. Then, we can modify its properties and use the EntityManager to persist the changes:

```
public void updateBookTitle(Long id, String title) {
    EntityManager em = getEntityManager();

    Book book = em.find(Book.class, id);
    book.setTitle(title);

    em.getTransaction().begin();
    em.persist(book);
    em.getTransaction().commit();
```

```
    em.close();
}
```

This method updates the title of the Book with the specified ID.

5. Deleting records

To delete a record from the database, we first need to retrieve the entity using its ID. Then, we can use the EntityManager to remove the entity:

```
public void deleteBook(Long id) {
    EntityManager em = getEntityManager();

    Book book = em.find(Book.class, id);

    em.getTransaction().begin();
    em.remove(book);
    em.getTransaction().commit();

    em.close();
}
```

This method deletes the Book with the specified ID.

That's it for this hands-on lab! We covered how to use CriteriaBuilder, Criteria Query, Root, and TypedQuery to perform basic CRUD operations on a database using JPA.

Remember to always properly manage your EntityManagers by closing them after each operation. Also, keep in mind that this is just a basic example and there are many more features and options available with CriteriaBuilder and JPA in general.