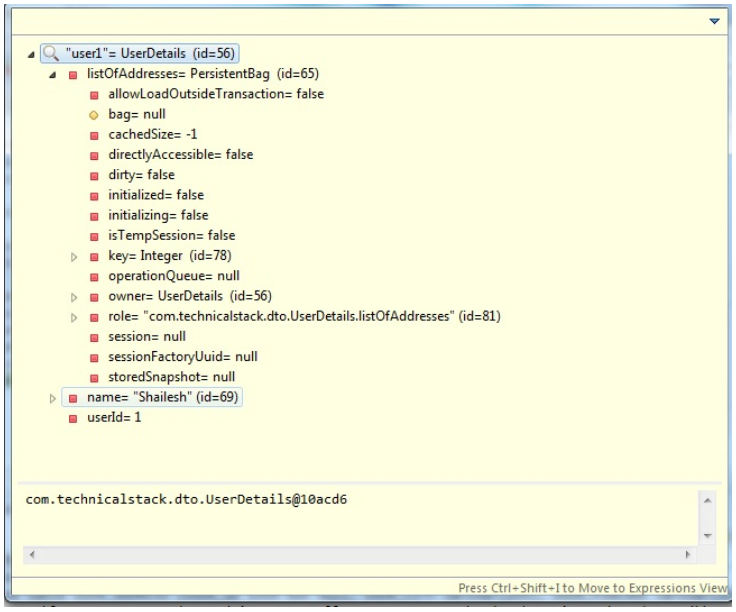
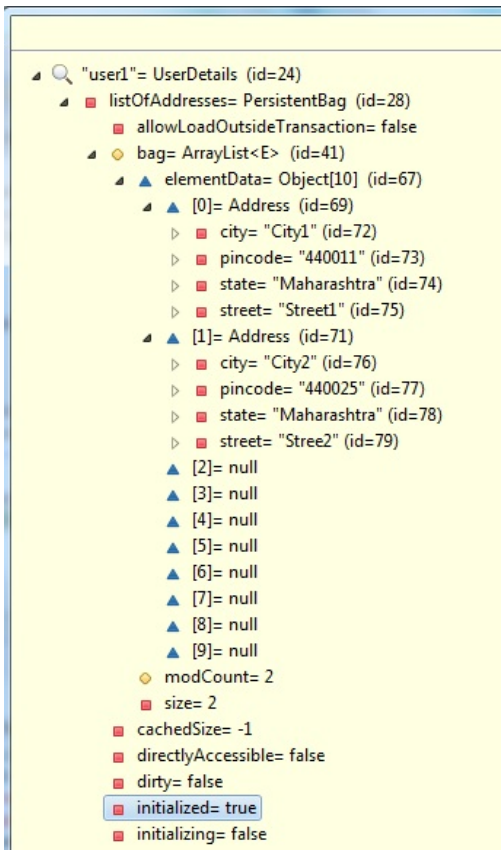
Proxy Objects,Eager and Lazy Fetch Types

Lazy Initialisation:
Lazy Fetch Types:
Lazy Initialisation do not initialise the entire object,it first initialise only the first level variables and it initialise the second level such as a list of address only when we actually want to access it.
Lazy intialization is the default behaviour of Hibernate
Now in our example suppose we have to fetch the values for UserDetails with primary-Id as 1,we will fetch it as below,

user1 = (UserDetails)session.get(UserDetails.class, 1);
At this point if we do an inspect on user1 in eclipse you will see below:



Observer the attributes of listOfAddress,
The bag = null & initalized = false
So user1 will have the values of the first level member variables
Now when we call user1.getListOfAddresses() when required then at this point the list gets populated and if we do and inspect at this point you will observe that the bag is now populated with the values and initalized=true.




```
package com.technicalstack.dto;

import javax.persistence.GeneratedValue;
import javax.persistence.Id;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateTest {

 public static void main(String[] args) {
  UserDetails user1 = new UserDetails();
 // user1.setUserId(1);  as we are using @Id @GeneratedValue so we don't need to set it.
  user1.setName("Shailesh");

  Address add1 = new Address();
  add1.setCity("City1");
  add1.setState("Maharashtra");
  add1.setPincode("440011");
  add1.setStreet("Street1");
  user1.getListOfAddresses().add(add1);

  Address add2 = new Address();
```

```
  add2.setCity("City2");
  add2.setState("Maharashtra");
  add2.setPincode("440025");
  add2.setStreet("Stree2");
  user1.getListOfAddresses().add(add2);

  SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();
  Session session = sessionFactory.openSession();
  session.beginTransaction();

  session.save(user1);

  session.getTransaction().commit();
  session.close();

  user1 = null;
  session = sessionFactory.openSession();
  user1 = (UserDetails)session.get(UserDetails.class, 1);
  session.close();
  System.out.println(user1.getListOfAddresses().size());


  }

}
```

Output:

```
Hibernate: select nextval ('hibernate_sequence')
Hibernate: insert into USER_DETAILS (USER_NAME, userId) values (?, ?)
Hibernate: insert into USER_ADDRESS (USER_ID, USER_CITY, USER_PINCODE, USER_STATE, USER_STREET) values (?, ?, ?, ?, ?)
Hibernate: insert into USER_ADDRESS (USER_ID, USER_CITY, USER_PINCODE, USER_STATE, USER_STREET) values (?, ?, ?, ?, ?)
Exception in thread "main" org.hibernate.LazyInitializationException: failed to lazily initialize a collection of role: com.technicalstack.dto.UserDetails.listOfAddresses, could not initialize proxy - no Session
 at org.hibernate.collection.internal.AbstractPersistentCollection.throwLazyInitializationException(AbstractPersistentCollection.java:587)
 at org.hibernate.collection.internal.AbstractPersistentCollection.withTemporarySessionIfNeeded(AbstractPersistentCollection.java:204)
 at org.hibernate.collection.internal.AbstractPersistentCollection.readSize(AbstractPersistentCollection.java:148)
 at org.hibernate.collection.internal.PersistentBag.size(PersistentBag.java:261)
 at com.technicalstack.dto.HibernateTest.main(HibernateTest.java:44)
Hibernate: select userdetail0_.userId as userId1_1_0_, userdetail0_.USER_NAME as USER_NAM2_1_0_ from USER_DETAILS userdetail0_ where userdetail0_.userId=?
```

Now suppose we have a requirement that we need to access the values of the list even after we close the session then how to do that.

Eager Fetch Types:
That can be achieved using Eager loading with the annotation @ElementCollection(fetch=FetchType.EAGER)
So if we want to retrive the values even when we use the get() method we can do it with FetchType.EAGER.

```
ElementCollection(fetch=FetchType.EAGER)
@JoinTable(name="USER_ADDRESS",
 joinColumns=@JoinColumn(name="USER_ID")
 )
/*@GenericGenerator(name="sequence-gen",strategy="sequence")
@CollectionId(columns = { @Column(name="ADDRESS_ID") }, generator = "sequence-gen", type = @Type(type="long"))
private Collection<Address> listOfAddresses = new ArrayList<Address>();
public Collection<Address> getListOfAddresses() {
 return listOfAddresses;
}
```