# DAY 7

## Eager Loading and Lazy Loading

- **Eager Loading** is a design pattern in which data initialization occurs on the spot.
- **Lazy Loading** is a design pattern that we use to defer initialization of an object as long as it's possible.

### ManyToMany.java

```
import java.util.List;

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

import com.tap.model.Employee;
import com.tap.model.Project;

public class ManyToManyFetchData {

     public static void main(String[] args) {
          SessionFactory sessionFactory = null;
          Session session = null;

          try {
               // Create session Factory

               sessionFactory = new Configuration()
                         .configure()
                         .addAnnotatedClass(Employee.class)
                         .addAnnotatedClass(Project.class)
                         .buildSessionFactory();
               // Create session
               session = sessionFactory.openSession();
```

```
                // Create Transaction
                Transaction transaction =
session.beginTransaction();

                // CRUD Operation
                Employee employee =
session.get(Employee.class, 2);

                transaction.commit();
           } finally {
                // Closing Resources
                session.close();
                sessionFactory.close();
           }
      }
}
```

**Output:**

```
Hibernate: select employee0_.id as id1_0_0_, employee0_.email
as email2_0_0_, employee0_.name as name3_0_0_ from employee
employee0_ where employee0_.id=?
```

If you observe the above output the employee object has fetched employee id, employee name but it didn’t fetch the project object this is because of lazy loading.

Now let’s try to get project object from the employee object and see weather we are getting project object

ManyToMany.java

```java
import java.util.List;

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

import com.tap.model.Employee;
import com.tap.model.Project;

public class ManyToManyFetchData {

     public static void main(String[] args) {
          SessionFactory sessionFactory = null;
          Session session = null;

          try {
               // Create session Factory

               sessionFactory = new Configuration()
                         .configure()
                         .addAnnotatedClass(Employee.class)
                         .addAnnotatedClass(Project.class)
                         .buildSessionFactory();
               // Create session
               session = sessionFactory.openSession();

               // Create Transaction
               Transaction transaction =
session.beginTransaction();

               // CRUD Operation
               Employee employee =
session.get(Employee.class, 2);
```

```
                List<Project> projects =
employee.getProjects();

                transaction.commit();
           } finally {
                // Closing Resources
                session.close();
                sessionFactory.close();

           }
      }

}
```

Output:

```
Hibernate: select employee0_.id as id1_0_0_, employee0_.email
as email2_0_0_, employee0_.name as name3_0_0_ from employee
employee0_ where employee0_.id=?
```

Now if you observe the above output even now the query to get the project object is not executed it clearly indicates that still project object is not fetched.

Now we will try to use the project Object

**ManyToMany.java**

```
import java.util.List;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.tap.model.Employee;
import com.tap.model.Project;
```

```java
public class ManyToManyFetchData {

     public static void main(String[] args) {
          SessionFactory sessionFactory = null;
          Session session = null;
          try {
               // Create session Factory

               sessionFactory = new Configuration()
                         .configure()
                         .addAnnotatedClass(Employee.class)
                         .addAnnotatedClass(Project.class)
                         .buildSessionFactory();
               // Create session
               session = sessionFactory.openSession();

               // Create Transaction
               Transaction transaction =
session.beginTransaction();
               // CRUD Operation
               Employee employee =
session.get(Employee.class, 1);
               List<Project> projects =
employee.getProjects();

               System.out.println(employee );

               for (Project project : projects) {
                    System.out.println(project);
               }

               transaction.commit();
          } finally {
               // Closing Resources
               session.close();
               sessionFactory.close();

          }
     }

}
```

**Output:**

```
Hibernate: select employee0_.id as id1_0_0_, employee0_.email
as email2_0_0_, employee0_.name as name3_0_0_ from employee
employee0_ where employee0_.id=?
Employee [2, Bob, bob@gmail.com]
Hibernate: select projects0_.emp_id as emp_id2_1_0_,
projects0_.proj_id as proj_id1_1_0_, project1_.id as
id1_2_1_, project1_.name as name2_2_1_ from employee_project
projects0_ inner join project project1_ on
projects0_.proj_id=project1_.id where projects0_.emp_id=?
Project [2, Aftereffects]
```

Now if you observe from the above output, the project object is being fetched when you are trying to print the projects in which employees are working. It is because of **lazy loading** unless you are making use of that object it will not fetch the object.

Now we will see how to fetch project object while fetching employee object

- In @ManyToMany annotations we need to mention one more variable which specifies **fetch=FetchType.EAGER** which indicates it is eager loading.

**Logic:**

```java
@ManyToMany(fetch = FetchType.EAGER,cascade =
{CascadeType.DETACH, CascadeType.MERGE, CascadeType.PERSIST,
CascadeType.REFRESH})
@JoinTable(name = "employee_project",joinColumns =
@JoinColumn(name = "emp_id"), inverseJoinColumns =
@JoinColumn(name = "proj_id"))
private List<Project> projects;
```

**Employee.java**

```java
package com.tap.model;

import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;

@Entity
@Table(name = "employee")
public class Employee {

     @Id
```

```
	@Column(name = "id")
	private int id;
	@Column(name = "name")
	private String name;
	@Column(name = "email")
	private String email;

	@ManyToMany(fetch = FetchType.EAGER,cascade =
{CascadeType.DETACH, CascadeType.MERGE,
			CascadeType.PERSIST, CascadeType.REFRESH})
	@JoinTable(name = "employee_project",
			joinColumns = @JoinColumn(name = "emp_id"),
			inverseJoinColumns = @JoinColumn(name =
"proj_id"))
	private List<Project> projects;

	public Employee() {
	}

	public Employee(int id, String name, String email) {
		super();
		this.id = id;
		this.name = name;
		this.email = email;
	}

	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
```

```java
		this.name = name;
	}

	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

	public List<Project> getProjects() {
		return projects;
	}

	public void setProjects(List<Project> projects) {
		this.projects = projects;
	}

	@Override
	public String toString() {
		return "Employee [" + id + ", " + name + ", " +
email + "]";
	}



}
```

**ManyToMany.java**

```
import java.util.List;

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

import com.tap.model.Employee;
import com.tap.model.Project;

public class ManyToManyFetchData {

     public static void main(String[] args) {
          SessionFactory sessionFactory = null;
          Session session = null;

          try {
               // Create session Factory

               sessionFactory = new Configuration()
                         .configure()
                         .addAnnotatedClass(Employee.class)
                         .addAnnotatedClass(Project.class)
                         .buildSessionFactory();
               // Create session
               session = sessionFactory.openSession();

               // Create Transaction
               Transaction transaction =
session.beginTransaction();

               // CRUD Operation
               Employee employee =
session.get(Employee.class, 2);

               transaction.commit();
          } finally {
               // Closing Resources
               session.close();
               sessionFactory.close();

          }
     }

}
```

Output:

```
Hibernate: select employee0_.id as id1_0_0_, employee0_.email
as email2_0_0_, employee0_.name as name3_0_0_,
projects1_.emp_id as emp_id2_1_1_, project2_.id as
proj_id1_1_1_, project2_.id as id1_2_2_, project2_.name as
name2_2_2_ from employee employee0_ left outer join
employee_project projects1_ on
employee0_.id=projects1_.emp_id left outer join project
project2_ on projects1_.proj_id=project2_.id where
employee0_.id=?
```

Now if you observe from the above output it is fetching project objects also this is only eager loading.

| **Eager Loading** | **Lazy Loading** |
|---|---|
| In Eager loading, data loading happens at the time of their parent is fetched | In Lazy loading, associated data loads only when we explicitly call getter or size method. |

## Load() and get()

In hibernate, get() and load() are two methods which are used to fetch data for the given identifier. They both belong to the Hibernate session class. Get() method returns null, If no row is available in the session cache or the database for the given identifier whereas load() method throws an object not found exception.

You have understood how to get the object using get() and load()

**ManyToMany.java**

```
import java.util.List;

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

import com.tap.model.Employee;
import com.tap.model.Project;

public class ManyToManyFetchData {

     public static void main(String[] args) {
          SessionFactory sessionFactory = null;
          Session session = null;

          try {
               // Create session Factory

               sessionFactory = new Configuration()
                         .configure()
                         .addAnnotatedClass(Employee.class)
                         .addAnnotatedClass(Project.class)
                         .buildSessionFactory();
               // Create session
               session = sessionFactory.openSession();
```

```
                // Create Transaction
                Transaction transaction =
session.beginTransaction();

                // CRUD Operation
                Employee employee =session.get(Employee.class,
2);

                transaction.commit();
           } finally {
                // Closing Resources
                session.close();
                sessionFactory.close();

           }
      }

}
```

Output:

```
Hibernate: select employee0_.id as id1_0_0_,
employee0_.email as email2_0_0_, employee0_.name as
name3_0_0_ from employee employee0_ where employee0_.id=?
```

Here if you observe from the output select query is executing it means employee object is fetched.

ManyToMany.java

```java
import java.util.List;

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

import com.tap.model.Employee;
import com.tap.model.Project;

public class ManyToMany {

     public static void main(String[] args) {
          SessionFactory sessionFactory = null;
          Session session = null;

          try {
//             Create session Factory

               sessionFactory = new Configuration()
                         .configure()

.addAnnotatedClass(Employee.class)
                         .addAnnotatedClass(Project.class)
                         .buildSessionFactory();
//             Create session
               session = sessionFactory.openSession();

//             Create Transaction
               Transaction transaction =
session.beginTransaction();

//             CRUD Operation
               Employee employee =
session.load(Employee.class, 2);


              transaction.commit();
         } finally {
//            Closing Resources
              session.close();
              sessionFactory.close();

         }
     }

}
```

Output:

```
Hibernate: alter table employee_project add constraint
FK2edylp1b54fdmioq3lkv7ieud foreign key (emp_id)
references employee (id)
Hibernate: alter table employee_project add constraint
FKc5if00myhjh5qohnmcdvxwlt3 foreign key (proj_id)
references project (id)
```

Now if you observe from the above output, there is no select query it means the employee object is not fetched because load() method acts like lazy loading it will fetch the object when you are using.

Now let's try to use the object that time load() will fetch the data and provide you

ManyToMany.java

```
import java.util.List;

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

import com.tap.model.Employee;
import com.tap.model.Project;

public class ManyToMany {

     public static void main(String[] args) {
          SessionFactory sessionFactory = null;
          Session session = null;

          try {
//             Create session Factory

               sessionFactory = new Configuration()
                         .configure()

.addAnnotatedClass(Employee.class)
                         .addAnnotatedClass(Project.class)
                         .buildSessionFactory();
//             Create session
               session = sessionFactory.openSession();

//             Create Transaction
               Transaction transaction =
session.beginTransaction();
//              CRUD Operation
               Employee employee =
session.load(Employee.class, 2);
               List<Project> projects =
employee.getProjects();

               System.out.println(employee );

               for (Project project : projects) {
                   System.out.println(project);
               }


               transaction.commit();
          } finally {
//              Closing Resources
               session.close();
               sessionFactory.close();

          }
     }

}
```

Output:

```
Hibernate: select employee0_.id as id1_0_0_,
employee0_.email as email2_0_0_, employee0_.name as
name3_0_0_ from employee employee0_ where employee0_.id=?
Employee [2, Bob, bob@gmail.com]
Hibernate: select projects0_.emp_id as emp_id2_1_0_,
projects0_.proj_id as proj_id1_1_0_, project1_.id as
id1_2_1_, project1_.name as name2_2_1_ from
```

```
employee_project projects0_ inner join project project1_
on projects0_.proj_id=project1_.id where
projects0_.emp_id=?
Project [2, Aftereffects]
```

As you can clearly see from the above output load() is fetching the data when you are trying to use it.

Now let's see how get() and load() method will works if the object that are trying to fetch is not present in the database

**ManyToMany.java**

```
import java.util.List;

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

import com.tap.model.Employee;
import com.tap.model.Project;

public class ManyToMany {

     public static void main(String[] args) {
          SessionFactory sessionFactory = null;
          Session session = null;

          try {
//             Create session Factory

               sessionFactory = new Configuration()
                         .configure()

.addAnnotatedClass(Employee.class)
                         .addAnnotatedClass(Project.class)
```

```
                        .buildSessionFactory();
//              Create session
                session = sessionFactory.openSession();

//              Create Transaction
                Transaction transaction =
session.beginTransaction();

//              CRUD Operation
                Employee employee =
session.get(Employee.class, 5);

                System.out.println(employee );

                transaction.commit();
           } finally {
//              Closing Resources
                session.close();
                sessionFactory.close();

           }
      }

}
```

**Output:**

```
Hibernate: select employee0_.id as id1_0_0_,
employee0_.email as email2_0_0_, employee0_.name as
name3_0_0_ from employee employee0_ where employee0_.id=?
null
```

If you can see from the above output when you are trying to fetch the object which is not present that time get() is giving you null.

**ManyToMany.java**

```
import java.util.List;

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

import com.tap.model.Employee;
import com.tap.model.Project;

public class ManyToMany {

    public static void main(String[] args) {
        SessionFactory sessionFactory = null;
        Session session = null;

        try {
//              Create session Factory

            sessionFactory = new Configuration()
                    .configure()

.addAnnotatedClass(Employee.class)
                    .addAnnotatedClass(Project.class)
                    .buildSessionFactory();
//              Create session
            session = sessionFactory.openSession();

//              Create Transaction
            Transaction transaction =
session.beginTransaction();
//              CRUD Operation
            Employee employee =
session.load(Employee.class, 5);

            System.out.println(employee );

            transaction.commit();
        } finally {
//              Closing Resources
            session.close();
            sessionFactory.close();

        }
    }

}
```

Output:

```
Exception in thread "main"
org.hibernate.ObjectNotFoundException: No row with the
given identifier exists: [com.tap.model.Employee#5]
     at
org.hibernate.boot.internal.StandardEntityNotFoundDelegat
e.handleEntityNotFound(StandardEntityNotFoundDelegate.jav
a:28)
     at
org.hibernate.proxy.AbstractLazyInitializer.checkTargetSt
ate(AbstractLazyInitializer.java:298)
     at
org.hibernate.proxy.AbstractLazyInitializer.initialize(Ab
stractLazyInitializer.java:187)
     at
org.hibernate.proxy.AbstractLazyInitializer.getImplementa
```

```
tion(AbstractLazyInitializer.java:322)
     at
org.hibernate.proxy.pojo.bytebuddy.ByteBuddyInterceptor.i
ntercept(ByteBuddyInterceptor.java:45)
     at
org.hibernate.proxy.ProxyConfiguration$InterceptorDispatc
her.intercept(ProxyConfiguration.java:95)
     at
com.tap.model.Employee$HibernateProxy$F8vgJOY1.toString(U
nknown Source)
     at
java.base/java.lang.String.valueOf(String.java:4215)
     at
java.base/java.io.PrintStream.println(PrintStream.java:10
47)
     at ManyToMany.main(ManyToMany.java:34)
```

So if you can see from the above output load() is giving you the exception when you are trying to fetch the object which is not present

## Difference between load() and get():

| load() | get() |
|---|---|
| Use the load() method if you are sure that the object exists. | If you are not sure that the object exists, then use one of the get() methods. |
| load() method will throw an exception if the unique id is not found in the database. | get() method will return null if the unique id is not found in the database. |
| load() just returns a proxy by default and database won't be hit until the proxy is first invoked | get() will hit the database immediately. |

## Life Cycle:

**Every Hibernate entity naturally has a lifecycle within the framework**

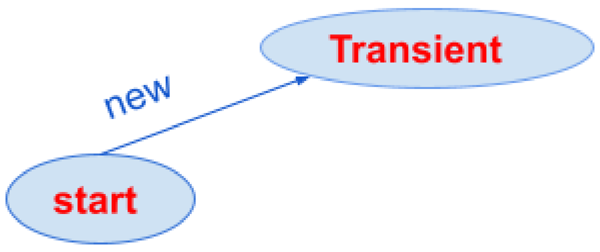
**The Hibernate lifecycle contains the following states:**

- Transient
- Persistent
- Detached
- Removed

**Transient:**

- This is the initial state of an object
- Once the instance of pojo class is created then object entered in transient state

  Project P = new Project(2, “Illustartor”); // Here object enters into transient state



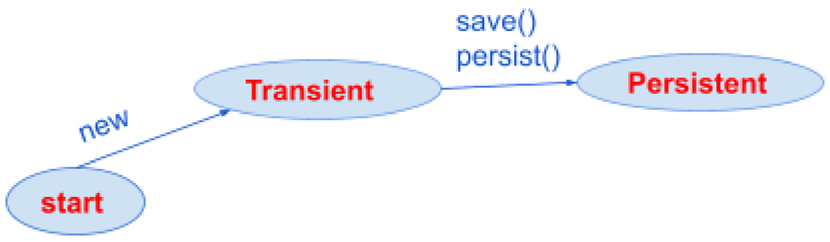
- Now project object is associated with session but it is not available in database

**Persistent:**

- To make a transient entity persistent, we need to call session.save(entity) or session.saveOrUpdate(entity) or session.persist(entity)

  Project p = new Project(2, “Illustartor”); // Here object enters into transient state

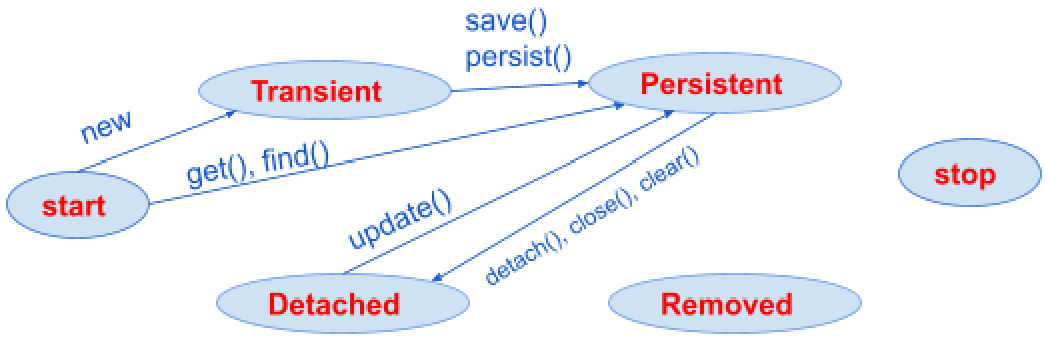
  session.save(p) // Now the object will be in database and with session



- Modification made in data make changes in the database

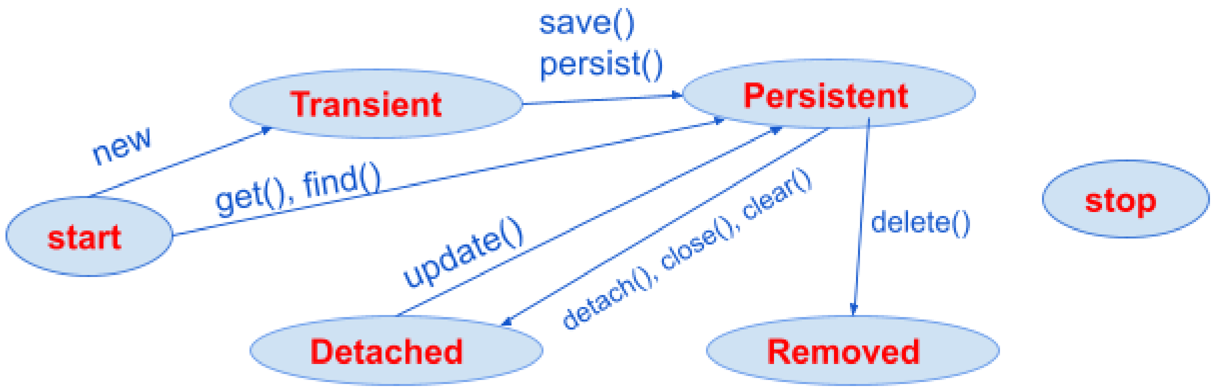
**Detached():**

- Once we close the session, clear its cache and detach object enters into detached state
- Now an object is no more associated with the Session
- In a detached state, changes done by detached objects are not saved to the database.
- To get object from detached to persistent state we need to call the method update()



**Removed():**

In the hibernate lifecycle it is the last state. In the removed state, when the entity object is deleted from the database then the entity object is known to be in the removed state. It is done by calling the **delete() operation**



## @GeneratedValue Identifiers

If we want to automatically generate the primary key value, **we can add the *@GeneratedValue* annotation.**

This can use four generation types: AUTO, IDENTITY, SEQUENCE and TABLE. If we don't explicitly specify a value, the generation type defaults to AUTO.

### GenerationType.IDENTITY:

This type of generation relies on the IdentityGenerator, which expects values generated by an identity column in the database. This means they are **auto-incremented.**

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "roll")
private int roll;
```

### Logic:

```
Student s = new Student(); // create student object
s.setEmail("bob@gmail.com"); //set email for student object
s.setName("Bob"); //set name for student object
s.setMarks(60); //set marks for student object

Student s2 = new Student(); // creating student object
s2.setEmail("alex@gmail.com"); // set email for student
object
s2.setName("Alex"); //set name for student object
s2.setMarks(70);  //set marks for student object

session.save(s);
session.save(s2);
//save the object to the database
```

| **Annotations** | **Action** |
|---|---|
| GenerationType.IDENTITY | This will auto increment the primary key this is similar to AUTO |
| GenerationType.AUTO | The AUTO will automatically set the generated values. It is the default GenerationType. If no strategy is defined, it will consider AUTO by default. |
| GenerationType.SEQUENCE | With this strategy, persistence providers must use a database sequence to get the next unique primary key for the entities. |
| GenerationType.TABLE | With this strategy, persistence provider must use a database table to generate/keep the next unique primary key for the entities. |