Day 22- Facilitation Guide

Index

For (1.5 hrs) ILT

During the previous session, we explored JPA,ORM Tool, HIbernate and some Important annotations.

We covered a variety of important concepts:

JPA (Java Persistence API): JPA stands for Java Persistence API, and it's a Java specification that provides a standardized way to manage relational data in Java applications. JPA allows developers to work with database records using Java objects, simplifying the interaction between the application and the database.

ORM (Object-Relational Mapping) Tool: ORM, or Object-Relational Mapping, is a programming technique and a set of tools that map database records to objects in an object-oriented programming language. ORM tools like Hibernate provide a way to work with databases using object-oriented code rather than writing SQL queries directly.

Hibernate: Hibernate is a popular open-source ORM framework and a JPA (Java Persistence API) implementation. It simplifies database access by allowing developers to work with Java objects and automatically managing the mapping between these objects and database tables.

In this session, we will delve into Hibernate configuration, the concept of Session Factory and Session, Hibernate State, and Transaction Management.

I. Hibernate configuration

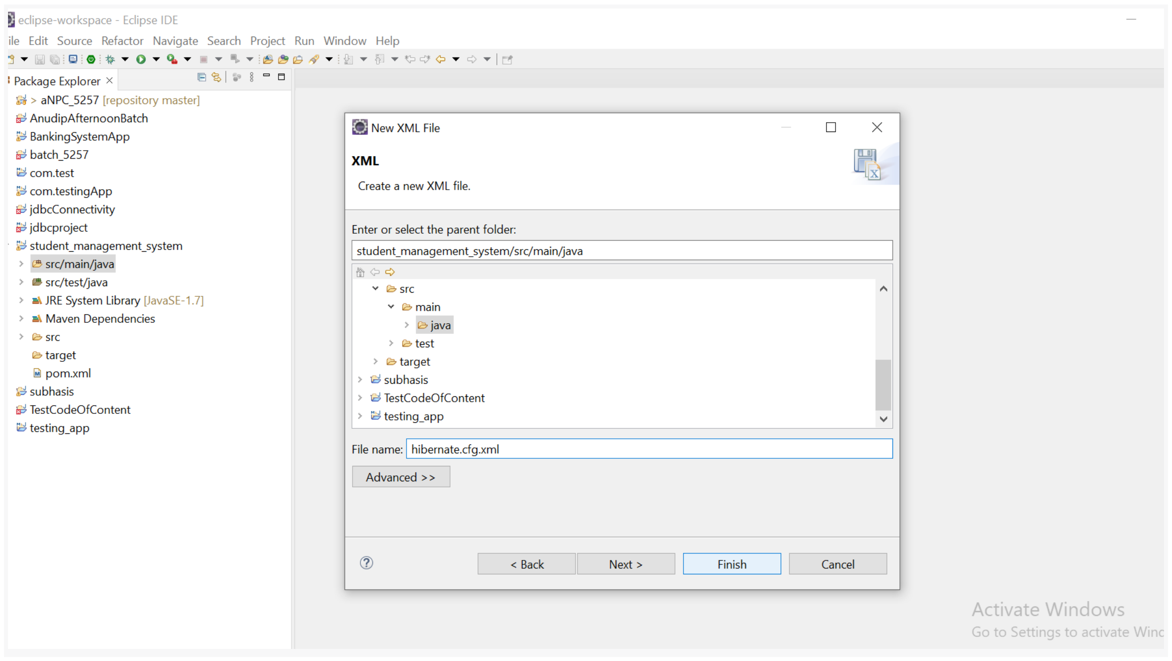
As Hibernate can operate in different environments, it requires a wide range of configuration parameters. These configurations contain the mapping information that provides different functionalities to Java classes. Generally, we provide database related mappings in the configuration file. Hibernate facilitates to provide the configurations either in an XML file (like hibernate.cfg.xml) or properties file (like hibernate.properties).

An instance of Configuration class allows specifying properties and mappings to applications. This class also builds an immutable **SessionFactory**.

Let's examine the practical implementation of the configuration:

Configuring Hibernate is an essential step to use it in your Java application. Hibernate is typically configured using a configuration file (often named hibernate.cfg.xml), where you specify various settings, such as database connection details and other Hibernate-specific options. Here's an example of how you can configure Hibernate:

To begin, initiate the configuration process by following these steps: Start by right-clicking on src/main/java, proceed to select "New," then choose "XML file," and proceed to rename the newly created file as "hibernate.cfg.xml."



New XML File

XML

Create a new XML file.

Enter or select the parent folder: student_management_system/src/main/java

File name: hibernate.cfg.xml

Hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

        <hibernate-configuration>

        <session-factory>

        <!-- Database connection settings -->
<property name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
<property name="connection.url">jdbc:mysql://localhost:3306/studentmanagementsystem</property>
<property name="connection.username">root</property>
<property name="connection.password">mysql</property>

<!-- Automatically create or update database schema -->
```

```
<property name="hbm2ddl.auto">update</property>

<!-- Specify the dialect for your database -->
<property name="dialect">org.hibernate.dialect.MySQL8Dialect</property>

<!-- Enable or disable showing SQL statements in the console -->
<property name="show_sql">true</property>
<property name="format_sql">true</property>

<!-- Specify the package(s) where your entity classes are located →
 <mapping class="com.sms.Student"/>

 <!-- Additional mappings can be included here -->
</session-factory>

</hibernate-configuration>
```

Here's a breakdown of the key configuration elements in this file:

<hibernate-configuration>: This is the root element of the configuration file.

<session-factory>: This section contains various properties and mappings specific to the Hibernate session factory.

connection.driver_class, connection.url, connection.username, and connection.password: These properties specify the database connection details.

dialect: This property defines the database dialect, which is necessary for Hibernate to generate appropriate SQL statements.

show_sql,format_sql: Setting this property to true will make Hibernate print SQL statements to the console in a formatted way, which is helpful for debugging.

hbm2ddl.auto: This property controls database schema generation. Options include create, update, validate, and more. Be cautious when using this in production.

<mapping>: Inside the session-factory section, you can specify the mapping for your entity classes. In the example, com.sms.Student is an entity class that's included in the configuration.

This is a basic Hibernate configuration. Depending on your application's needs and the database you're using, you may need to customize it further. Once you have this configuration file, you can use it to create a Hibernate SessionFactory, which is the entry point for working with Hibernate and performing database operations in your application.

Having grasped the essentials of Hibernate configuration, it's time to delve into understanding SessionFactory and Session before proceeding to the practical demonstration.

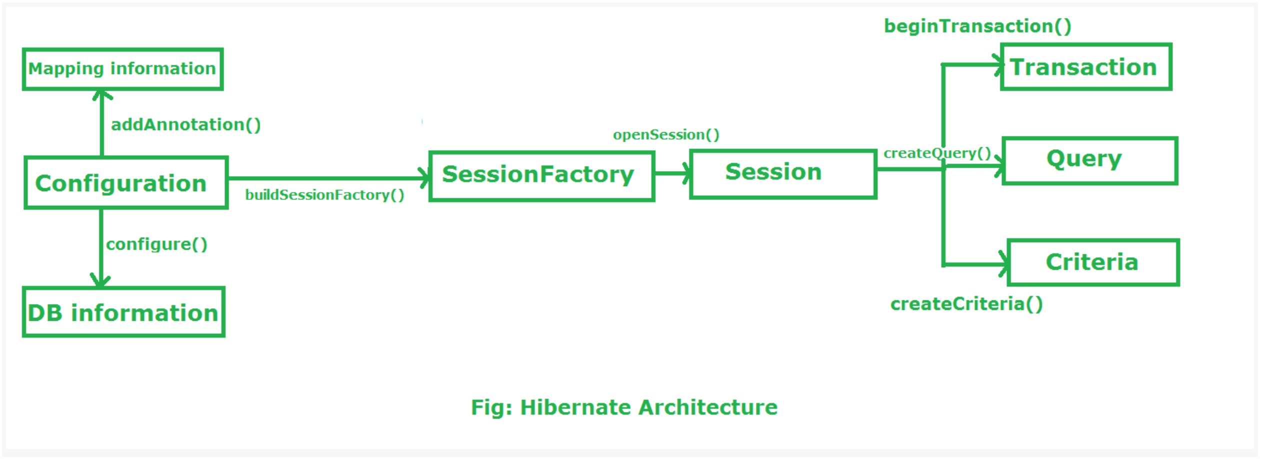
II. Introduction to Sessionfactory and session

Session Factory:

The Session Factory is a crucial component in Hibernate, serving as a factory for creating Session objects. It is a heavyweight object created once at the application's startup and is designed to be thread-safe, meaning it can be shared across different parts of the application. The Session Factory is responsible for managing database connections, transaction management, and handling the mapping of Java objects to database tables. It is essential for optimizing Hibernate's performance, as creating a SessionFactory is a relatively expensive operation.

Session:

A Session, on the other hand, is a lightweight, short-lived object in Hibernate. It represents a single unit of work with the database. Sessions are typically created when needed and are used to perform various database operations like saving, updating, deleting, and querying entities. A Session is bound to a specific database connection and a single transaction, making it suitable for encapsulating a specific piece of work. Once the work is completed, the Session is closed, releasing the database connection and any associated resources.



Now, let's explore the entire Hibernate example.

III. Hibernate Application Using Annotation

Note: we have already set up a Hibernate configuration file (hibernate.cfg.xml) and have the necessary dependencies in our project.

Entity Class:

Let's create a simple entity class, annotated with Hibernate annotations:

```
package com.sms;
import java.time.LocalDate;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
@Entity
public class Student {
        @Id
        @Column(name = "StudentId", length = 10)
        private String studentId;

        @Column(name = "FirstName", length = 50)
        private String firstName;

        @Column(name = "LastName", length = 25)
        private String lastName;
```

```
@Column(name = "DateOfBirth")
private LocalDate dateOfBirth;

@Column(name = "Gender", length = 25)
private String gender;

@Column(name = "Email", length = 30)
private String email;

@Column(name = "Phone", length = 25)
private String phone;

//Setter And Getter
public String getStudentId() {
        return studentId;
}
public void setStudentId(String studentId) {
        this.studentId = studentId;
}
public String getFirstName() {
        return firstName;
}
public void setFirstName(String firstName) {
        this.firstName = firstName;
}
public String getLastName() {
        return lastName;
}
public void setLastName(String lastName) {
        this.lastName = lastName;
}
public LocalDate getDateOfBirth() {
        return dateOfBirth;
}
public void setDateOfBirth(LocalDate dateOfBirth) {
        this.dateOfBirth = dateOfBirth;
}
public String getGender() {
        return gender;
```

```
	}
	public void setGender(String gender) {
		this.gender = gender;
	}
	public String getEmail() {
		return email;
	}
	public void setEmail(String email) {
		this.email = email;
	}
	public String getPhone() {
		return phone;
	}
	public void setPhone(String phone) {
		this.phone = phone;
	}

	//All argument Constructor
	public Student(String studentId, String firstName, String lastName, LocalDate
dateOfBirth, String gender,
			String email, String phone) {
		super();
		this.studentId = studentId;
		this.firstName = firstName;
		this.lastName = lastName;
		this.dateOfBirth = dateOfBirth;
		this.gender = gender;
		this.email = email;
		this.phone = phone;
	}

	//Default Constructor
	public Student() {
		super();
	}

	//ToString method
	@Override
	public String toString() {
		return "Student [studentId=" + studentId + ", firstName=" + firstName + ", 
lastName=" + lastName
				+ ", dateOfBirth=" + dateOfBirth + ", gender=" + gender + ", 
email=" + email + ", phone=" + phone + "]";
	}
}
```

Hibernate Configuration:

We have already set up a Hibernate configuration file hibernate.cfg.xml.

```
package com.sms;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtil {
	private static final SessionFactory sessionFactory = buildSessionFactory();
	private static SessionFactory buildSessionFactory() {
		try {
			return new 
Configuration().configure("hibernate.cfg.xml").addAnnotatedClass(Student.class)
					.buildSessionFactory();
		} catch (Throwable ex) {
			throw new ExceptionInInitializerError(ex);
		}
	}
	public static SessionFactory getSessionFactory() {
		return sessionFactory;
	}
}
```

Note: Configuration is a class which is present in org.hibernate.cfg package. It activates the Hibernate framework. It reads both configuration file and mapping files.It checks whether the config file is syntactically correct or not.If the config file is not valid then it will throw an exception. If it is valid then it creates a meta-data in memory and returns the meta-data to the object to represent the config file.

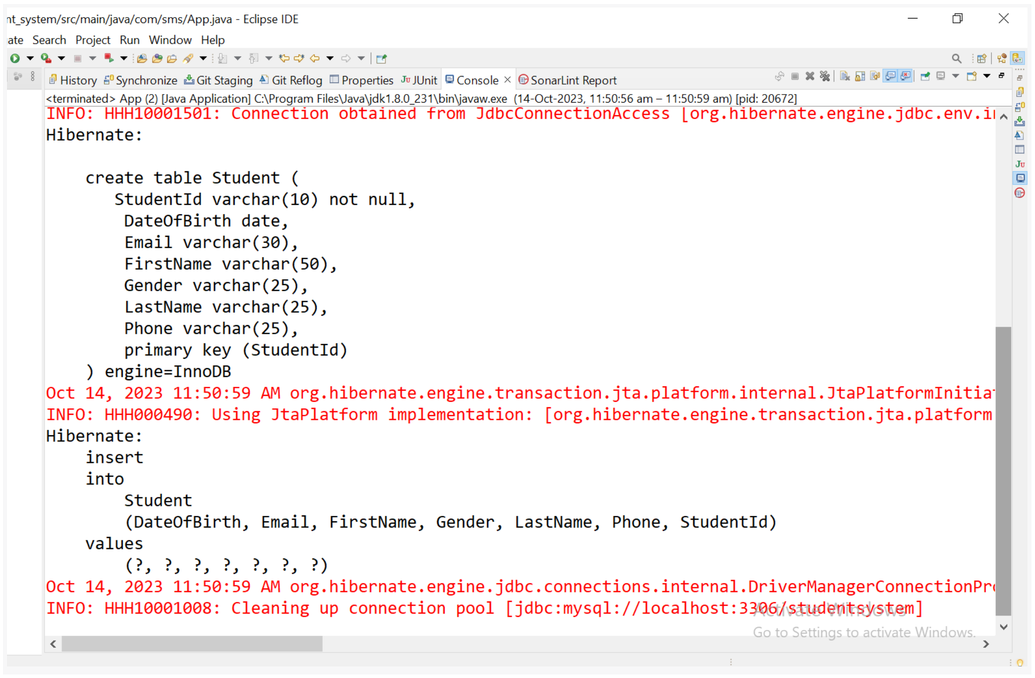
buildSessionFactory() method gathers the meta-data which is in the cfg Object.
From cfg object it takes the JDBC information and creates a JDBC Connection.

Main Application:

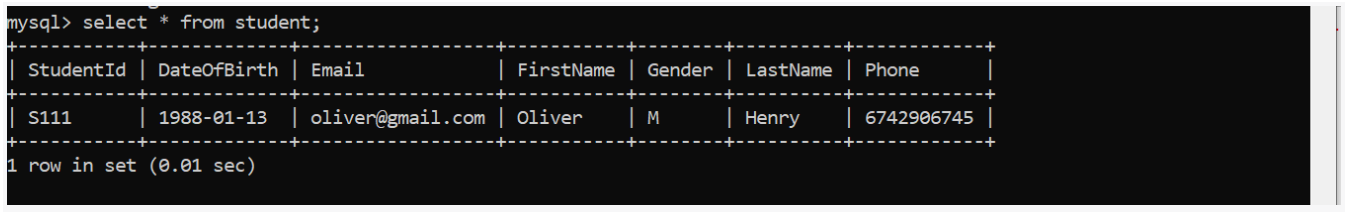
In your application, you can use the SessionFactory to create a Session and perform various database operations:

```
package com.sms;
import java.time.LocalDate;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

public class App {
        public static void main(String[] args) {

                // Obtain a Hibernate SessionFactory
                SessionFactory factory = HibernateUtil.getSessionFactory();

                // Create a new Student
                LocalDate date1 = LocalDate.of(1988, 1, 13);
                Student Student1 = new Student("S111", "Oliver", "Henry", date1, "M",
"oliver@gmail.com", "6742906745");

                // Open a new session
                Session session = factory.openSession();

                // Begin a transaction
                Transaction transaction = session.beginTransaction();

                // Save the student to the database
                session.save(Student1);

                // Commit the transaction
                transaction.commit();

                // Close the Session
                session.close();
                // Close the Session Factory
                factory.close();
        }
}
```

Output:


nt_system/src/main/java/com/sms/App.java - Eclipse IDE
ate Search Project Run Window Help
History Synchronize Git Staging Git Reflog Properties JUnit Console SonarLint Report

```
<terminated> App (2) [Java Application] C:\Program Files\Java\jdk1.8.0_231\bin\javaw.exe (14-Oct-2023, 11:50:56 am – 11:50:59 am) [pid: 20672]
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.i
Hibernate:

    create table Student (
       StudentId varchar(10) not null,
        DateOfBirth date,
        Email varchar(30),
        FirstName varchar(50),
        Gender varchar(25),
        LastName varchar(25),
        Phone varchar(25),
        primary key (StudentId)
    ) engine=InnoDB
Oct 14, 2023 11:50:59 AM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitia
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform
Hibernate:
    insert
    into
        Student
        (DateOfBirth, Email, FirstName, Gender, LastName, Phone, StudentId)
    values
        (?, ?, ?, ?, ?, ?, ?)
Oct 14, 2023 11:50:59 AM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionPr
INFO: HHH10001008: Cleaning up connection pool [jdbc:mysql://localhost:3306/studentsystem]
```



```
mysql> select * from student;
+-----------+-------------+-----------------+-----------+--------+----------+------------+
| StudentId | DateOfBirth | Email           | FirstName | Gender | LastName | Phone      |
+-----------+-------------+-----------------+-----------+--------+----------+------------+
| S111      | 1988-01-13  | oliver@gmail.com | Oliver   | M      | Henry    | 6742906745 |
+-----------+-------------+-----------------+-----------+--------+----------+------------+
1 row in set (0.01 sec)
```

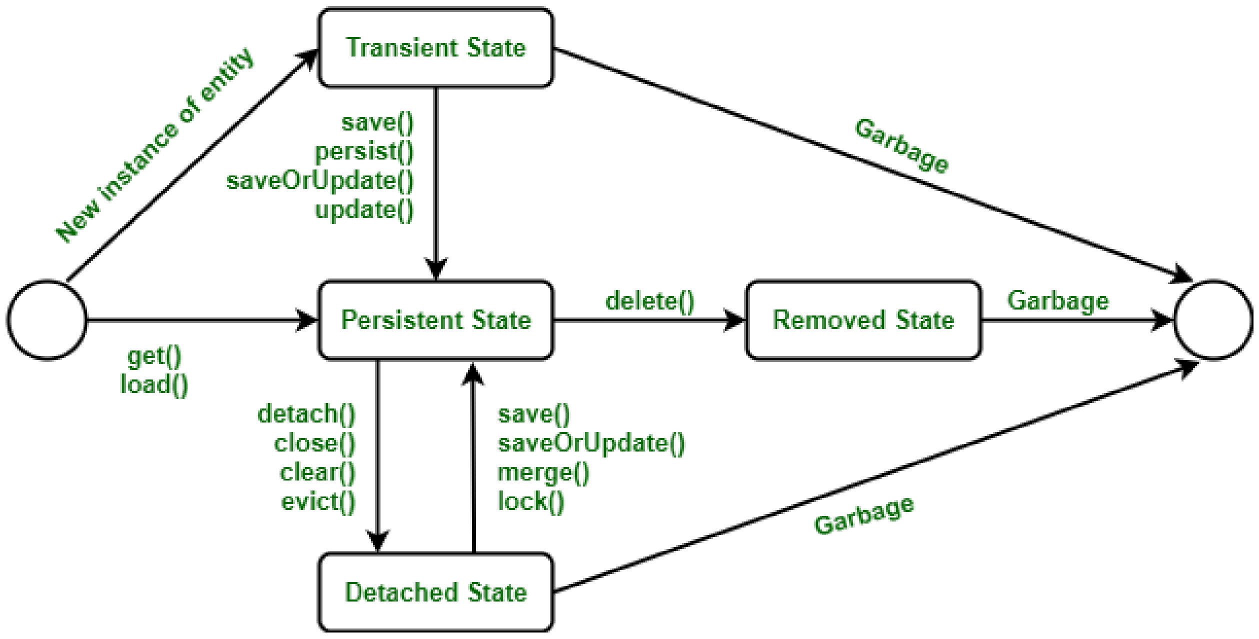
This example demonstrates creating an Student entity, persisting it to the database, and properly closing the resources.

Note: In this session, we will address transaction management.

IV. Hibernate State

In Hibernate, we can either create a new object of an entity and store it into the database, or we can fetch the existing data of an entity from the database. These entity is connected with the lifecycle and each object of the entity passes through the various stages of the life cycle.
There are mainly four states of the Hibernate Lifecycle :

1. Transient State
2. Persistent State
3. Detached State
4. Removed State



State 1: Transient State

The transient state is the first state of an entity object. When we instantiate an object of a POJO class using the new operator then the object is in the transient state. This object is not connected with any hibernate session. As it is not connected to any Hibernate Session, So this state is not connected to any database table. So, if we make any changes in the data of the POJO Class then the database table is not altered. Transient objects are independent of Hibernate, and they exist in the heap memory.

New instance of entity
Transient State

There are two layouts in which transient state will occur as follows:

1. When objects are generated by an application but are not connected to any session.
2. The objects are generated by a closed session.

Here, we are creating a new object for the Student class. Below is the code which shows the initialization of the Employee object :

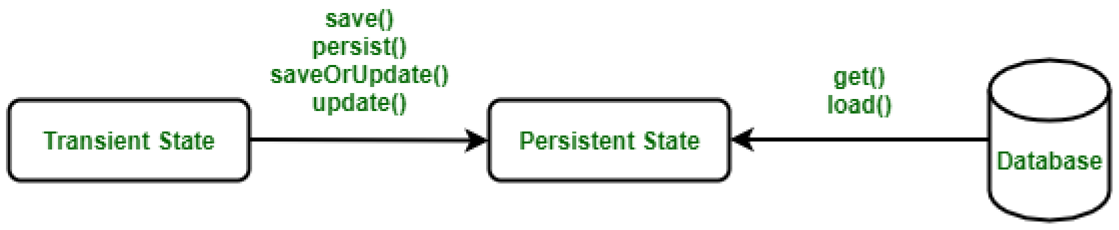
Student Student1 = new Student("S111", "Oliver", "Henry", date1, "M", "oliver@gmail.com", "6742906745");

State 2: Persistent State

Once the object is connected with the Hibernate Session then the object moves into the Persistent State. So, there are two ways to convert the Transient State to the Persistent State :

1. Using the hibernate session, save the entity object into the database table.
2. Using the hibernate session, load the entity object into the database table.

In this state. Each object represents one row in the database table. Therefore, if we make any changes in the data then hibernate will detect these changes and make changes in the database table.



Converting Transient State to Persistent State

Following are the methods given for the persistent state:

- session.persist(Student1);
- session.save(Student1);
- session.saveOrUpdate(Student1);
- session.update(Student1);
- session.merge(Student1);

- session.lock(Student1);

// Transient State

Student Student1 = new Student("S111", "Oliver", "Henry", date1, "M", "oliver@gmail.com", "6742906745");

//Persistent State

session.save(Student1);

State 3: Detached State

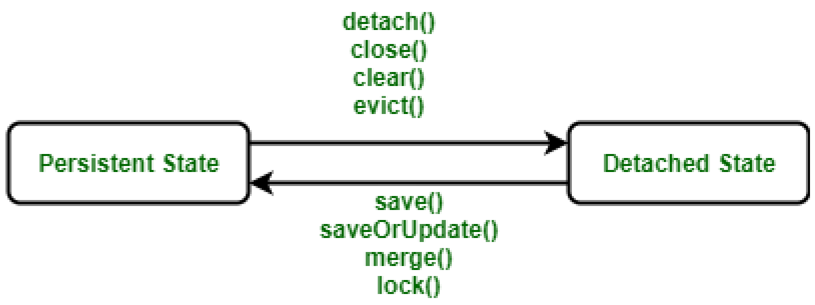
For converting an object from Persistent State to Detached State, we either have to close the session or we have to clear its cache. As the session is closed here or the cache is cleared, then any changes made to the data will not affect the database table. Whenever needed, the detached object can be reconnected to a new hibernate session.

To reconnect the detached object to a new hibernate session, we will use the following methods as follows:

- merge()
- update()
- load()
- refresh()
- save()
- update()

Following are the methods used for the detached state :

- session.detach(Student1);
- session.evict(Student1);
- session.clear();
- session.close();



Converting Persistent State to Detached State

// Transient State

Student Student1 = new Student("S111", "Oliver", "Henry", date1, "M", "oliver@gmail.com", "6742906745");

//Persistent State

session.save(Student1);

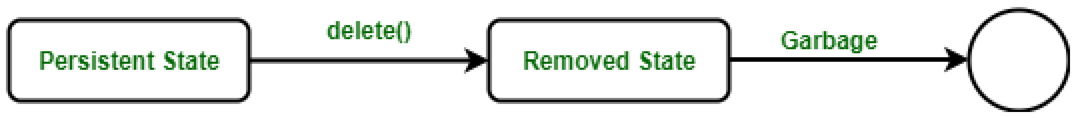
// Detached State

session.close();

State 4: Removed State

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. As the entity object is in the removed state, if any change will be done in the data will not affect the database table.

Note: To make a removed entity object we will call session.delete().



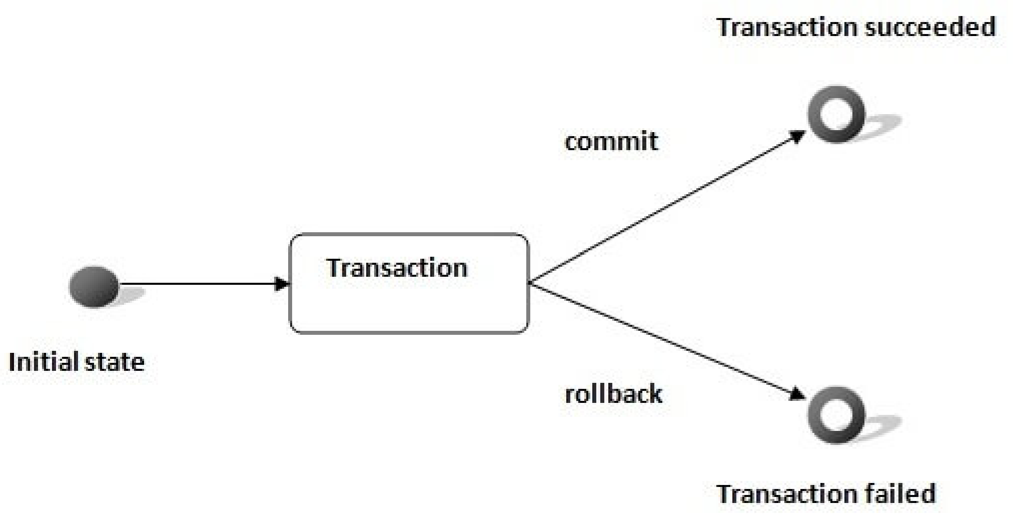
Converting Persistent State to Removed State

// Transient State

```
Student Student1 = new Student("S111", "Oliver", "Henry", date1, "M",
"oliver@gmail.com", "6742906745");

//Persistent State

session.save(Student1);

// Removed State
session.delete(e);
```

V. Transaction Management

A **transaction** simply represents a unit of work. In such case, if one step fails, the whole transaction fails (which is termed as atomicity). A transaction can be described by ACID properties (Atomicity, Consistency, Isolation and Durability).



Transaction Interface in Hibernate

In the hibernate framework, we have a Transaction interface that defines the unit of work. It maintains abstraction from the transaction implementation (JTA,JDBC).

A transaction is associated with Session and instantiated by calling **session.beginTransaction()**.

The methods of Transaction interface are as follows:

1. **void begin()** starts a new transaction.
2. **void commit()** ends the unit of work unless we are in FlushMode.NEVER.
3. **void rollback()** forces this transaction to rollback.
4. **void setTimeout(int seconds)** it sets a transaction timeout for any transaction started by a subsequent call to begin on this instance.
5. **boolean isAlive()** checks if the transaction is still alive.
6. **void registerSynchronization(Synchronization s)** registers a user synchronization callback for this transaction.
7. **boolean wasCommited()** checks if the transaction is commited successfully.
8. **boolean wasRolledBack()** checks if the transaction is rolledback successfully.

Example:

```
// Open a new session
Session session = factory.openSession();

// Begin a transaction
Transaction transaction = session.beginTransaction();

// Commit the transaction
transaction.commit();
```

Note: We have already handled transactions within our Hibernate application, and it is essential to commit each transaction in order to persist the objects.

Food for thought..

The trainer can ask the students the following question to engage them in a discussion:

What happens when a Hibernate transaction encounters an exception or an error, and how can you handle and recover from such situations?

Knowledge check…

Now that we've covered JPA introduction and understanding of ORM tool, it's time to test your understanding. The Trainer will conduct a short poll quiz to assess your knowledge on these topics.

1.What is the initial state of an object in Hibernate when it's just created and not associated with any Hibernate session?

a. Detached
b. Persistent
c. Transient
d. Removed

2.In Hibernate, which method is commonly used to begin a transaction?
a. startTransaction()
b. beginTransaction()
c. initiateTransaction()
d. createTransaction()

At the end of the quiz,the Trainer will provide feedback and explanations to the participants, enhancing the learning experience and understanding of the content.

VI. Retrieving record using get() & load() method

Let’s deal with Fetching objects in Spring Hibernate:

Hibernate provides different methods to fetch data from the database.

1. get()
2. load()

1. get() method

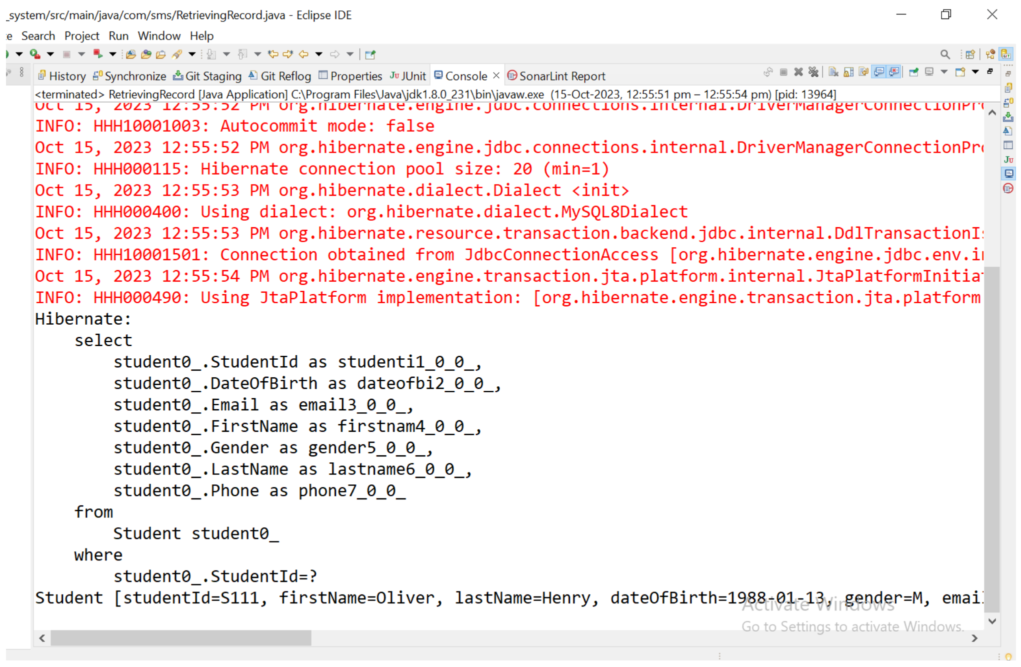
- get() method is used to retrieve a persistent object from the database. It is a member of the Session interface, and it takes the class of the object to be retrieved and the primary key of the object as arguments.
- get() method only hits the database if the object is not present in the session cache. If the same object is already present in the cache then it returns previously stored data from the cache.
- get() method returns null if there is no object present in the database.

Let us see the complete example:

We already have our "Students" Entity Class populated with some records. Now, let's retrieve the records using the "get()" method.

```
package com.sms;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
public class RetrievingRecord {
        public static void main(String[] args) {
                // Obtain a Hibernate SessionFactory
                SessionFactory factory = HibernateUtil.getSessionFactory();
                // Open a new session
                Session session = factory.openSession();

                // Begin a transaction
                Transaction transaction = session.beginTransaction();

                // Retrieve the object using the primary key
                Student student=session.get(Student.class, "S111");

                //display data using toString() method
                System.out.println(student);

                // Commit the transaction
                transaction.commit();
                // Close the session
                session.close();
        }
}
```

Output:



```
Oct 15, 2023 12:55:52 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionPr
INFO: HHH10001003: Autocommit mode: false
Oct 15, 2023 12:55:52 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionPr
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
Oct 15, 2023 12:55:53 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQL8Dialect
Oct 15, 2023 12:55:53 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionI
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.i
Oct 15, 2023 12:55:54 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitia
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform
Hibernate: 
    select
        student0_.StudentId as studenti1_0_0_,
        student0_.DateOfBirth as dateofbi2_0_0_,
        student0_.Email as email3_0_0_,
        student0_.FirstName as firstnam4_0_0_,
        student0_.Gender as gender5_0_0_,
        student0_.LastName as lastname6_0_0_,
        student0_.Phone as phone7_0_0_ 
    from
        Student student0_ 
    where
        student0_.StudentId=?
Student [studentId=S111, firstName=Oliver, lastName=Henry, dateOfBirth=1988-01-13, gender=M, emai
```

2. load() method

- load() method is used to retrieve an object from the database by its identifier (primary key). It is used to initialize a proxy object instead of a fully-initialized object, so it can be used to lazily load the object when it is needed.
- load() method does not retrieve the object from the database when it is called. Instead, it returns a proxy object that represents the object. The actual object is only retrieved from the database when it is needed, such as when a method of the object is called or a property is accessed. This technique is known as “lazy loading” and it is used to improve the performance of Hibernate by avoiding unnecessary database queries.

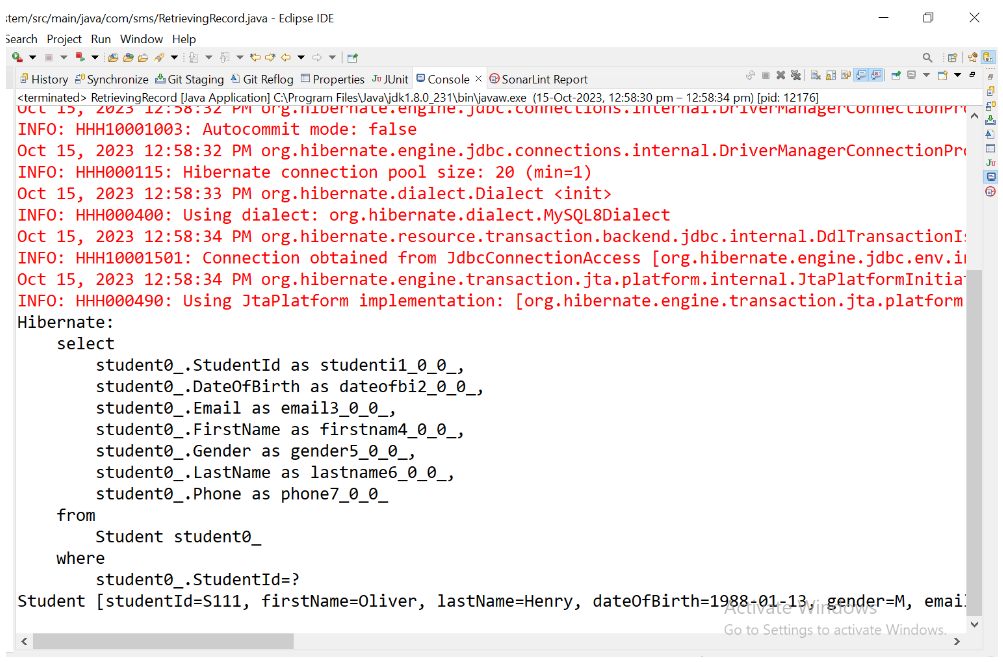
- load() method throws ObjectNotFoundException if there is no object found in the database.

Let us see the complete example:

We already have our "Students" Entity Class populated with some records. Now, let's retrieve the records using the "load()" method.

```
package com.sms;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
public class RetrievingRecord {
        public static void main(String[] args) {
                // Obtain a Hibernate SessionFactory
                SessionFactory factory = HibernateUtil.getSessionFactory();
                // Open a new session
                Session session = factory.openSession();

                // Begin a transaction
                Transaction transaction = session.beginTransaction();

                // Retrieve the object using the primary key
                Student student=session.load(Student.class, "S111");
                //display data using toString() method
                System.out.println(student);

                // Commit the transaction
                transaction.commit();

                // Close the session
                session.close();
        }
}
```

Output:


tem/src/main/java/com/sms/RetrievingRecord.java - Eclipse IDE
Search Project Run Window Help
History Synchronize Git Staging Git Reflog Properties JUnit Console SonarLint Report
Activate Windows
Go to Settings to activate Windows.

```
<terminated> RetrievingRecord [Java Application] C:\Program Files\Java\jdk1.8.0_231\bin\javaw.exe (15-Oct-2023, 12:58:30 pm – 12:58:34 pm) [pid: 12176]
Oct 15, 2023 12:58:32 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionPr
INFO: HHH10001003: Autocommit mode: false
Oct 15, 2023 12:58:32 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionPr
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
Oct 15, 2023 12:58:33 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQL8Dialect
Oct 15, 2023 12:58:34 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionI
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.i
Oct 15, 2023 12:58:34 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitia
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform
Hibernate:
    select
        student0_.StudentId as studenti1_0_0_,
        student0_.DateOfBirth as dateofbi2_0_0_,
        student0_.Email as email3_0_0_,
        student0_.FirstName as firstnam4_0_0_,
        student0_.Gender as gender5_0_0_,
        student0_.LastName as lastname6_0_0_,
        student0_.Phone as phone7_0_0_
    from
        Student student0_
    where
        student0_.StudentId=?
Student [studentId=S111, firstName=Oliver, lastName=Henry, dateOfBirth=1988-01-13, gender=M, emai
```

Question: In what situations would you prefer to use the load() method over the get() method, and vice versa?

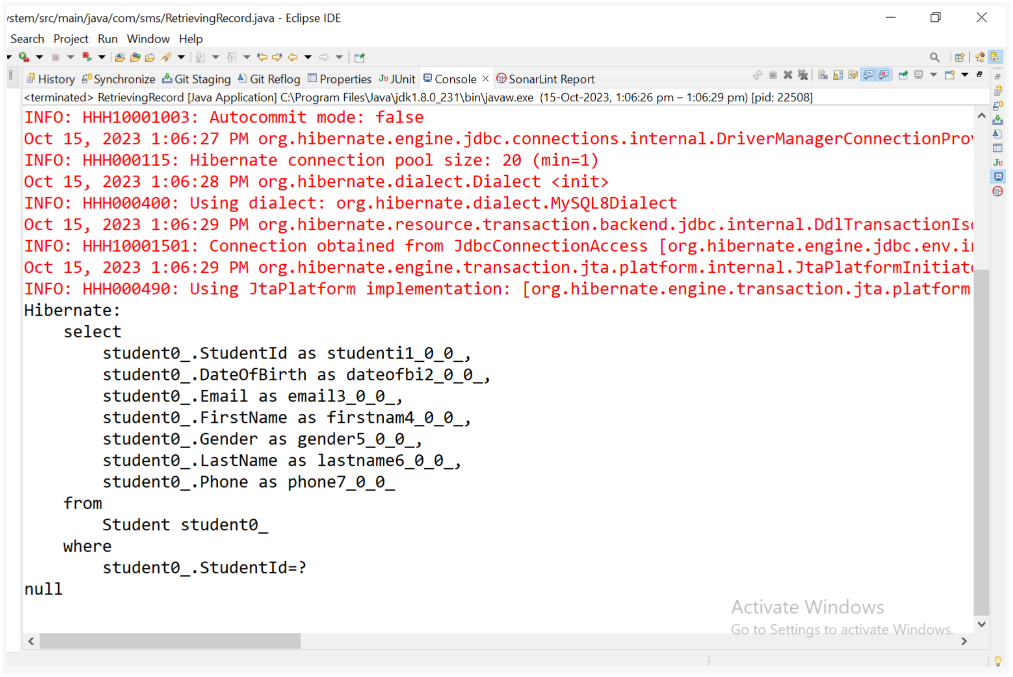
Now, let's explore the distinction between the "get()" and "load()" methods.

Both the get() and load() methods are used for fetching objects from the database based on their unique identifiers (usually the primary key). However, there is a crucial difference between the two:

get() Method:

- When you use get(), Hibernate immediately retrieves the object from the database and returns it.
- If no matching record is found, get() returns null.
- It hits the database with a SQL SELECT statement right away, and the object is fully initialized.
- This method is suitable when you want to ensure the existence of the object, and it's safe to return null if the object is not found.

// Retrieve the object using the primary key
Student student=session.get(Student.class, "S112");


/stem/src/main/java/com/sms/RetrievingRecord.java - Eclipse IDE
Search Project Run Window Help
History Synchronize Git Staging Git Reflog Properties JUnit Console SonarLint Report
Activate Windows
Go to Settings to activate Windows.

```
<terminated> RetrievingRecord [Java Application] C:\Program Files\Java\jdk1.8.0_231\bin\javaw.exe (15-Oct-2023, 1:06:26 pm – 1:06:29 pm) [pid: 22508]
INFO: HHH10001003: Autocommit mode: false
Oct 15, 2023 1:06:27 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionPro
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
Oct 15, 2023 1:06:28 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQL8Dialect
Oct 15, 2023 1:06:29 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIs
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.i
Oct 15, 2023 1:06:29 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiat
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform
Hibernate: 
    select
        student0_.StudentId as studenti1_0_0_,
        student0_.DateOfBirth as dateofbi2_0_0_,
        student0_.Email as email3_0_0_,
        student0_.FirstName as firstnam4_0_0_,
        student0_.Gender as gender5_0_0_,
        student0_.LastName as lastname6_0_0_,
        student0_.Phone as phone7_0_0_ 
    from
        Student student0_ 
    where
        student0_.StudentId=?
null
```

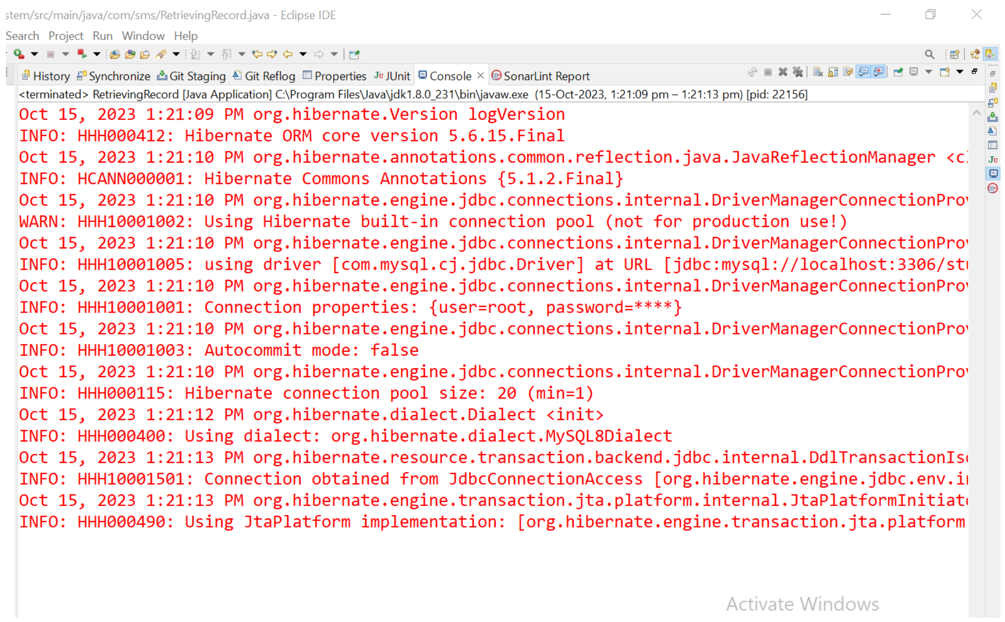
In this example, you can observe that the "Student" with ID S112 is not found in our database, which is why the "get" method is returning a null result.

load() Method:

- The load() method returns a proxy object (a placeholder) without hitting the database immediately. The actual database query is executed only when you access the properties or methods of the proxy object.
- If no matching record is found, load() will throw ObjectNotFoundException.
- The primary use of load() is for lazy loading. It's efficient when you might not need to access the object's properties immediately and want to minimize database queries.

Now, let's examine the lazy loading with an example:

```
// Retrieve the object using the primary key
Student student=session.load(Student.class, "S111");
```

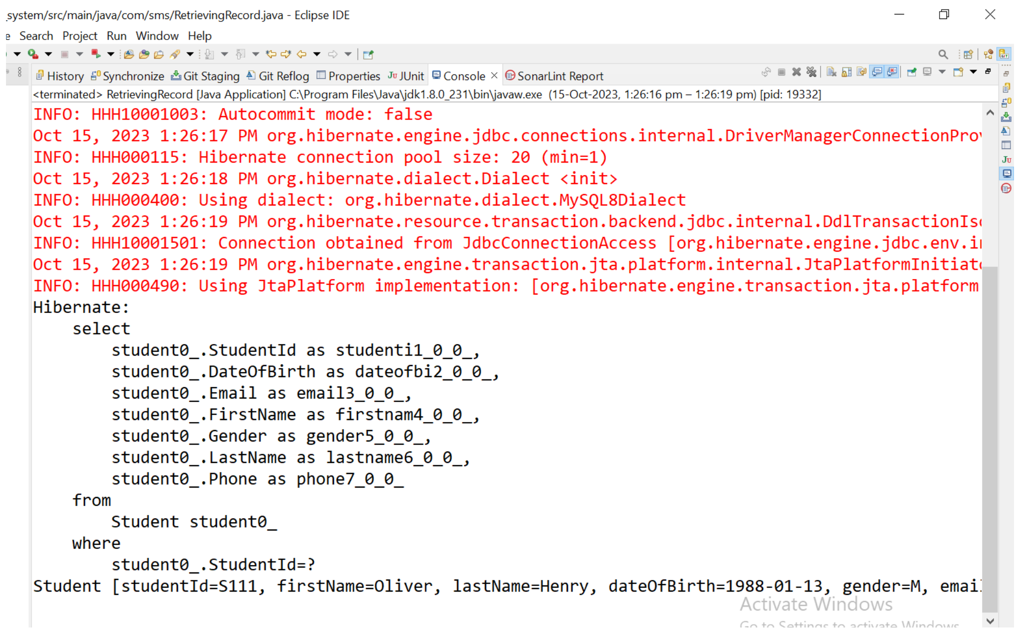

stem/src/main/java/com/sms/RetrievingRecord.java - Eclipse IDE
Search Project Run Window Help
History Synchronize Git Staging Git Reflog Properties JUnit Console SonarLint Report
Activate Windows

```
<terminated> RetrievingRecord [Java Application] C:\Program Files\Java\jdk1.8.0_231\bin\javaw.exe (15-Oct-2023, 1:21:09 pm – 1:21:13 pm) [pid: 22156]
Oct 15, 2023 1:21:09 PM org.hibernate.Version logVersion
INFO: HHH000412: Hibernate ORM core version 5.6.15.Final
Oct 15, 2023 1:21:10 PM org.hibernate.annotations.common.reflection.java.JavaReflectionManager <c
INFO: HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
Oct 15, 2023 1:21:10 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionPro
WARN: HHH10001002: Using Hibernate built-in connection pool (not for production use!)
Oct 15, 2023 1:21:10 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionPro
INFO: HHH10001005: using driver [com.mysql.cj.jdbc.Driver] at URL [jdbc:mysql://localhost:3306/st
Oct 15, 2023 1:21:10 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionPro
INFO: HHH10001001: Connection properties: {user=root, password=****}
Oct 15, 2023 1:21:10 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionPro
INFO: HHH10001003: Autocommit mode: false
Oct 15, 2023 1:21:10 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionPro
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
Oct 15, 2023 1:21:12 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQL8Dialect
Oct 15, 2023 1:21:13 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIs
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.i
Oct 15, 2023 1:21:13 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiat
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform
```

Here you can see,database queries will be executed when you access entity properties or methods.

Now, let's invoke the toString() method and observe the outcome.

```
// Retrieve the object using the primary key
Student student=session.load(Student.class, "S111");

//display data using toString() method
System.out.println(student);
```



system/src/main/java/com/sms/RetrievingRecord.java - Eclipse IDE

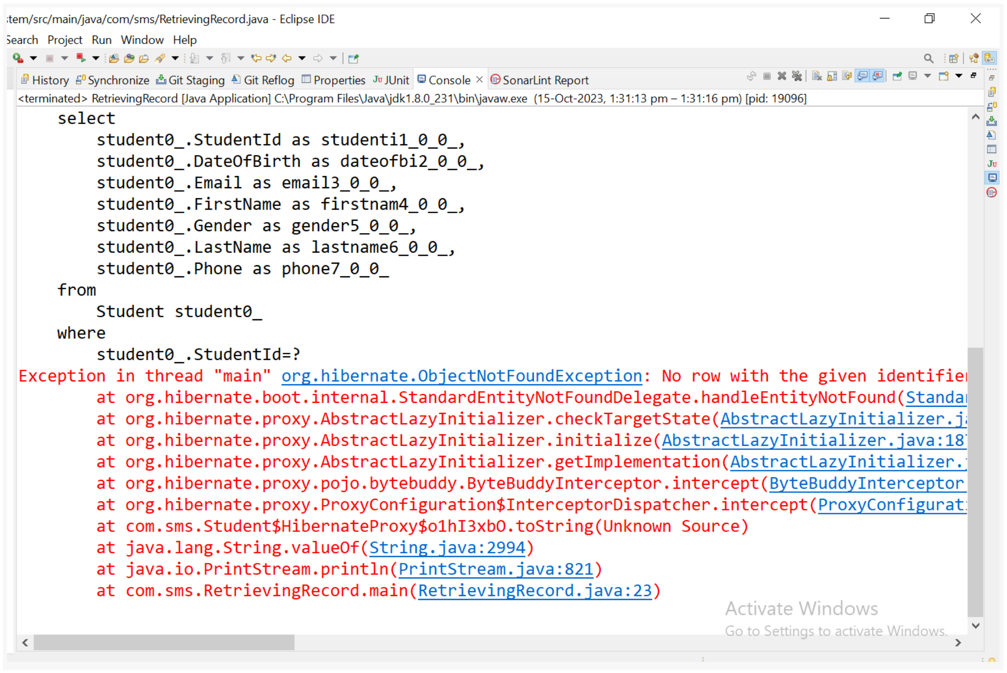
Search Project Run Window Help
History Synchronize Git Staging Git Reflog Properties JUnit Console SonarLint Report

```
<terminated> RetrievingRecord [Java Application] C:\Program Files\Java\jdk1.8.0_231\bin\javaw.exe  (15-Oct-2023, 1:26:16 pm – 1:26:19 pm) [pid: 19332]
INFO: HHH10001003: Autocommit mode: false
Oct 15, 2023 1:26:17 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionPro
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
Oct 15, 2023 1:26:18 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQL8Dialect
Oct 15, 2023 1:26:19 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIs
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.i
Oct 15, 2023 1:26:19 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiat
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform
Hibernate:
    select
        student0_.StudentId as studenti1_0_0_,
        student0_.DateOfBirth as dateofbi2_0_0_,
        student0_.Email as email3_0_0_,
        student0_.FirstName as firstnam4_0_0_,
        student0_.Gender as gender5_0_0_,
        student0_.LastName as lastname6_0_0_,
        student0_.Phone as phone7_0_0_
    from
        Student student0_
    where
        student0_.StudentId=?
Student [studentId=S111, firstName=Oliver, lastName=Henry, dateOfBirth=1988-01-13, gender=M, emai
```

Here Database query is executed as we have access entity properties or methods.

Now, let's examine what happens when a Student ID is not present.

// Retrieve the object using the primary key
Student student=session.load(Student.class, "S112");



tem/src/main/java/com/sms/RetrievingRecord.java - Eclipse IDE

Search Project Run Window Help
History Synchronize Git Staging Git Reflog Properties JUnit Console SonarLint Report

```
<terminated> RetrievingRecord [Java Application] C:\Program Files\Java\jdk1.8.0_231\bin\javaw.exe  (15-Oct-2023, 1:31:13 pm – 1:31:16 pm) [pid: 19096]
    select
        student0_.StudentId as studenti1_0_0_,
        student0_.DateOfBirth as dateofbi2_0_0_,
        student0_.Email as email3_0_0_,
        student0_.FirstName as firstnam4_0_0_,
        student0_.Gender as gender5_0_0_,
        student0_.LastName as lastname6_0_0_,
        student0_.Phone as phone7_0_0_
    from
        Student student0_
    where
        student0_.StudentId=?
Exception in thread "main" org.hibernate.ObjectNotFoundException: No row with the given identifie
        at org.hibernate.boot.internal.StandardEntityNotFoundDelegate.handleEntityNotFound(Standa
        at org.hibernate.proxy.AbstractLazyInitializer.checkTargetState(AbstractLazyInitializer.j
        at org.hibernate.proxy.AbstractLazyInitializer.initialize(AbstractLazyInitializer.java:18
        at org.hibernate.proxy.AbstractLazyInitializer.getImplementation(AbstractLazyInitializer.
        at org.hibernate.proxy.pojo.bytebuddy.ByteBuddyInterceptor.intercept(ByteBuddyInterceptor
        at org.hibernate.proxy.ProxyConfiguration$InterceptorDispatcher.intercept(ProxyConfigurat
        at com.sms.Student$HibernateProxy$o1hI3xbO.toString(Unknown Source)
        at java.lang.String.valueOf(String.java:2994)
        at java.io.PrintStream.println(PrintStream.java:821)
        at com.sms.RetrievingRecord.main(RetrievingRecord.java:23)
```

In this example, you can observe that the "Student" with ID S112 is not found in our database, which is why the "load" method is throwing an ObjectNotFoundException result.

Exercise:

Use ChatGPT to explore Hibernate:

Put the below problem statement in the message box and see what ChatGPT says.

I'm currently studying Hibernate transactions and am keen to delve deeper into the topic of ACID properties. Could you please provide a tutorial on the ACID properties?