**Difference between JPA, Hibernate and Spring Data JPA**   
  
Java Persistence API (JPA)

* JSR 338 Specification for persisting, reading and managing data from Java objects
* Does not contain concrete implementation of the specification
* Hibernate is one of the implementation of JPA

Hibernate

* ORM Tool that implements JPA

Spring Data JPA

* Does not have JPA implementation, but reduces boiler plate code
* This is another level of abstraction over JPA implementation provider like Hibernate
* Manages transactions

**Refer code snippets below on how the code compares between Hibernate and Spring Data JPA**  
**Hibernate**

   /\* Method to CREATE an employee in the database \*/

   public Integer addEmployee(Employee employee){

      Session session = factory.openSession();

      Transaction tx = null;

      Integer employeeID = null;

      try {

         tx = session.beginTransaction();

         employeeID = (Integer) session.save(employee);

         tx.commit();

      } catch (HibernateException e) {

         if (tx != null) tx.rollback();

         e.printStackTrace();

      } finally {

         session.close();

      }

      return employeeID;

   }

**Spring Data JPA**  
EmployeeRespository.java

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

EmployeeService.java

@Autowire

  private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

  employeeRepository.save(employee);

  }

**Reference Links:**   
<https://dzone.com/articles/what-is-the-difference-between-hibernate-and-sprin-1>   
<https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html>

## **Output:**

## **Java Persistence API (JPA)**

* **Definition**:  
   JPA is a **specification** (defined by JSR 338) for managing, persisting, and accessing data between Java objects and relational databases. It is a set of interfaces and annotations, but it does not provide any implementation itself.
* **Key Points**:
  + Standardizes object-relational mapping for Java
  + Makes it easier to switch between JPA implementations (e.g., Hibernate, EclipseLink)
  + Defines concepts like Entity, EntityManager, Query, etc.
  + Does **not** contain any code to persist data by itself
* **Example**:  
   We annotate classes with JPA annotations like @Entity, @Table, and use EntityManager interfaces, but the underlying work is performed by the implementation.

## **Hibernate**

* **Definition**:  
   Hibernate is a **concrete implementation of JPA** and a popular ORM (Object Relational Mapping) tool for Java.
* **Key Points**:
  + Provides the real code to persist Java objects to a relational database
  + Supports JPA annotations, but also offers extra Hibernate-specific features if needed
  + Manages:
    - sessions
    - transactions
    - caching
    - lazy loading
  + One of the most widely used ORM tools in the Java ecosystem

**Example (Traditional Hibernate code)**:

public Integer addEmployee(Employee employee) {

Session session = factory.openSession();

Transaction tx = null;

Integer employeeID = null;

try {

tx = session.beginTransaction();

employeeID = (Integer) session.save(employee);

tx.commit();

} catch (HibernateException e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

return employeeID;

}

Here we **manually** manage the session, transaction, exception handling, and closing resources.

## **Spring Data JPA**

* **Definition**:  
   Spring Data JPA is part of the Spring Data family of projects. It provides an **abstraction layer** on top of a JPA implementation (like Hibernate) to significantly reduce boilerplate code.
* **Key Points**:
  + Provides *repositories* such as JpaRepository that you can extend to perform CRUD operations automatically
  + Uses JPA/Hibernate internally to execute the queries
  + Manages transactions, sessions, and exceptions for you
  + Allows writing *query methods* using just method names (derived queries)
  + Makes JPA usage extremely simple and clean
* **Example (Spring Data JPA style)**:

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Integer> { }

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

Here we **do not** manually open sessions or handle transactions — Spring takes care of all of that.