# 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  
• Provides additional features beyond JPA, like caching, lazy loading, etc.  
• Developers manage session and transactions manually (unless integrated with frameworks like Spring)

## Spring Data JPA

• Does not have JPA implementation, but reduces boilerplate code  
• This is another level of abstraction over JPA implementation provider like Hibernate  
• Manages transactions automatically with Spring  
• Uses simple interfaces like JpaRepository to handle database operations

# Code Comparison

## Hibernate (Without Spring)

/\* 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 (With Spring Boot)

// EmployeeRepository.java  
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {  
}  
  
// EmployeeService.java  
@Autowired  
private EmployeeRepository employeeRepository;  
  
@Transactional  
public void addEmployee(Employee employee) {  
 employeeRepository.save(employee);  
}

# Comparison Table

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | JPA | Hibernate | Spring Data JPA |
| Type | Specification (JSR 338) | ORM Framework & JPA Implementation | Abstraction over JPA & ORM Frameworks like Hibernate |
| Purpose | Defines standard APIs for ORM | Provides actual implementation of ORM & JPA | Simplifies JPA-based data access with Spring integration |
| Implementation | No (Only interfaces & annotations) | Yes (Provides full JPA implementation & additional ORM features) | No (Relies on JPA providers like Hibernate) |
| Boilerplate Code | High (Manual EntityManager, Queries, Transactions) | Medium (Session, Transactions handled manually) | Very Low (Uses Repository Interfaces like JpaRepository) |
| Transaction Management | Manual or via Container | Manual (unless integrated with frameworks) | Auto-managed via Spring Framework |
| Learning Curve | Moderate | Higher (More configurations, more APIs to learn) | Easy (Most code handled by Spring) |
| Query Language | JPQL | HQL (Hibernate Query Language) + JPQL | Uses JPQL + Derived Queries + Custom Queries |
| Examples | EntityManager | SessionFactory and Session | JpaRepository, CrudRepository |
| Best Used For | When you want pure standard & flexibility | When you need full control over ORM features | When you want rapid development with minimal code |