# Difference Between JPA, Hibernate, and Spring Data JPA

## Java Persistence API (JPA)

* **JPA** is a **specification (JSR 338)** for accessing, persisting, and managing data between Java objects and relational databases.
* It defines a set of interfaces and annotations to interact with the database.
* **Does not contain any implementation.**
* Requires an implementation (e.g., Hibernate, EclipseLink) to function.

## Hibernate

* **Hibernate** is a popular **ORM (Object Relational Mapping)** framework.
* **Implements the JPA specification** and adds additional features.
* Can work with or without JPA.
* Manages database operations using Session, Transaction, etc.
* Requires more boilerplate code compared to Spring Data JPA.

## Spring Data JPA

* **Spring Data JPA** is a part of the **Spring Data** project.
* Provides **abstraction over JPA** implementations (e.g., Hibernate).
* **Does not implement JPA**, but helps in reducing boilerplate code.
* Provides JpaRepository and other interfaces to interact with the database easily.
* Automatically handles transactions and CRUD operations.

# Code Comparison

## Hibernate Example

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 Example

### EmployeeRepository.java

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {  
 // No implementation needed  
}

### EmployeeService.java

@Service  
public class EmployeeService {  
  
 @Autowired  
 private EmployeeRepository employeeRepository;  
  
 @Transactional  
 public void addEmployee(Employee employee) {  
 employeeRepository.save(employee);  
 }  
}