Spring Data JPA

The Master Class

By Ramesh Fadatare (Java Guides)

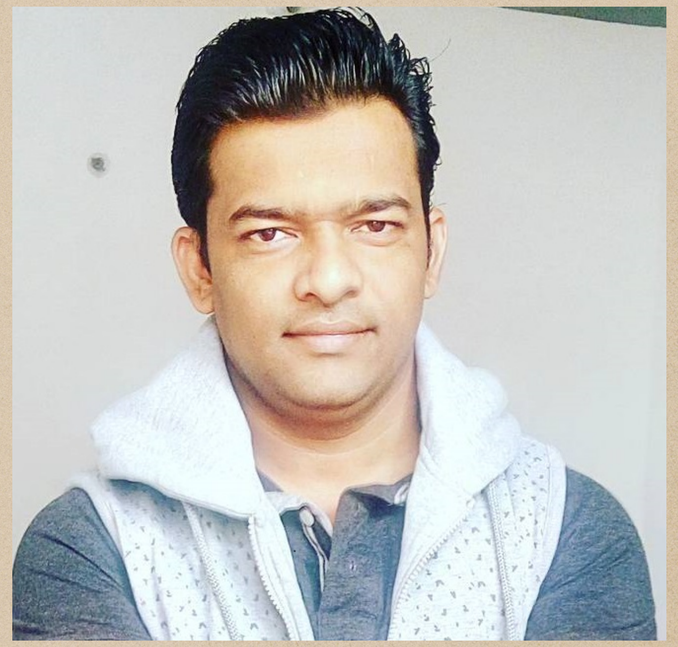
Your Instructor

My name is Ramesh and working as a Tech Lead in IT company. I have 10 years of experience in IT.

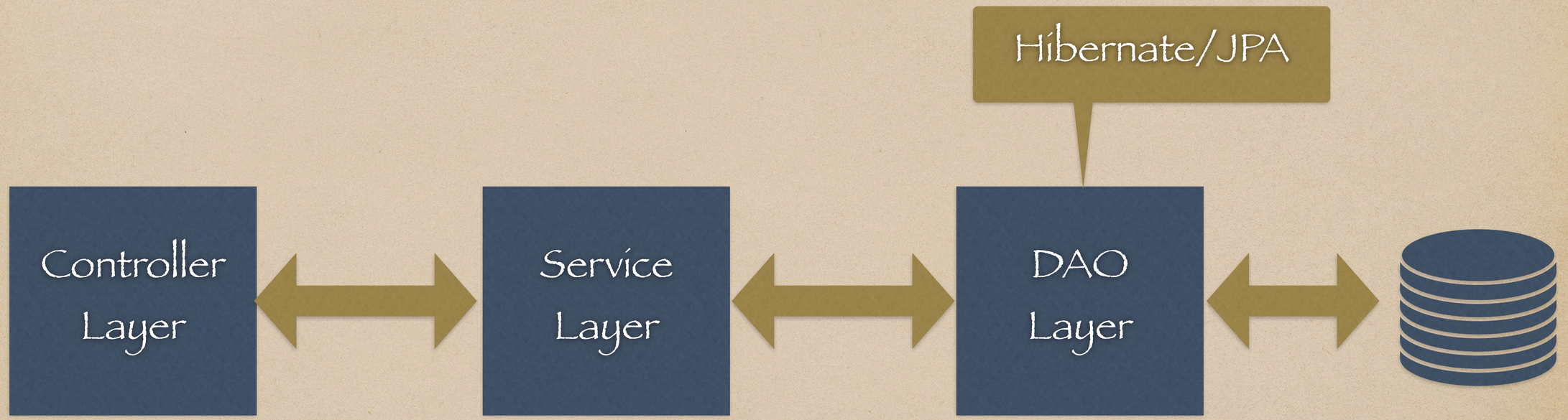
I am a founder and author of top Java blog website JavaGuides (1M views per month)

I am a YouTuber at JavaGuides (52K Subscribers)

I have published around 200+ free projects on GitHub for learning purposes (2K followers)



Application Architecture



Creating DAO for Single Entity

```java
public interface EmployeeDAO {

	public List<Employee> findAll();

	public Employee findById(int theId);

	public void save(Employee theEmployee);

	public void deleteById(int theId);

}
```

```java
@Repository
public class EmployeeDAOJpaImpl implements EmployeeDAO {

	private EntityManager entityManager;

	@Autowired
	public EmployeeDAOJpaImpl(EntityManager theEntityManager) {
		entityManager = theEntityManager;
	}

	@Override
	public List<Employee> findAll() {
		// create a query
		Query theQuery = entityManager.createQuery("from Employee");
		// execute query and get result list
		List<Employee> employees = theQuery.getResultList();
		// return the results
		return employees;
	}

	@Override
	public Employee findById(int theId) {
		// get employee
		Employee theEmployee = entityManager.find(Employee.class, theId);
		// return employee
		return theEmployee;
	}

	@Override
	public void save(Employee theEmployee) {
		// save or update the employee
		Employee dbEmployee = entityManager.merge(theEmployee);
		// update with id from db ... so we can get generated id for save/insert
		theEmployee.setId(dbEmployee.getId());
	}

	@Override
	public void deleteById(int theId) {
		// delete object with primary key
		Query theQuery = entityManager.createQuery("delete from Employee where id=:employeeId");
		theQuery.setParameter("employeeId", theId);
		theQuery.executeUpdate();
	}
}
```

The Problem

What if we create DAO for other entities

Student, Product, Order, Customer etc

We are repeating the same code for other entities

```
public interface EmployeeDAO {

        public List<Employee> findAll();

        public Employee findById(int theId);

        public void save(Employee theEmployee);

        public void deleteById(int theId);

}
```

```
@Repository
public class EmployeeDAOJpaImpl implements EmployeeDAO {

        private EntityManager entityManager;

        @Autowired
        public EmployeeDAOJpaImpl(EntityManager theEntityManager) {
                entityManager = theEntityManager;
        }

        @Override
        public List<Employee> findAll() {
                // create a query
                Query theQuery = entityManager.createQuery("from Employee");
                // execute query and get result list
                List<Employee> employees = theQuery.getResultList();
                // return the results
                return employees;
        }

        @Override
        public Employee findById(int theId) {
                // get employee
                Employee theEmployee = entityManager.find(Employee.class, theId);
                // return employee
                return theEmployee;
        }

        @Override
        public void save(Employee theEmployee) {
                // save or update the employee
                Employee dbEmployee = entityManager.merge(theEmployee);
                // update with id from db ... so we can get generated id for save/insert
                theEmployee.setId(dbEmployee.getId());
        }

        @Override
        public void deleteById(int theId) {
                // delete object with primary key
                Query theQuery = entityManager.createQuery("delete from Employee where id=:employeeId");
                theQuery.setParameter("employeeId", theId);
                theQuery.executeUpdate();
        }
}
```

DAO Pattern

```java
@Override
public List<Employee> findAll() {

        // create a query
        Query theQuery =
                        entityManager.createQuery("from Employee");

        // execute query and get result list
        List<Employee> employees = theQuery.getResultList();

        // return the results
        return employees;
}

@Override
public Employee findById(int theId) {

        // get employee
        Employee theEmployee =
                        entityManager.find(Employee.class, theId);

        // return employee
        return theEmployee;
}
```

JPQL query for Entity Type

Entity Type

Primary Key

Generic Code

We can eliminate some boilerplate code by writing generic code:

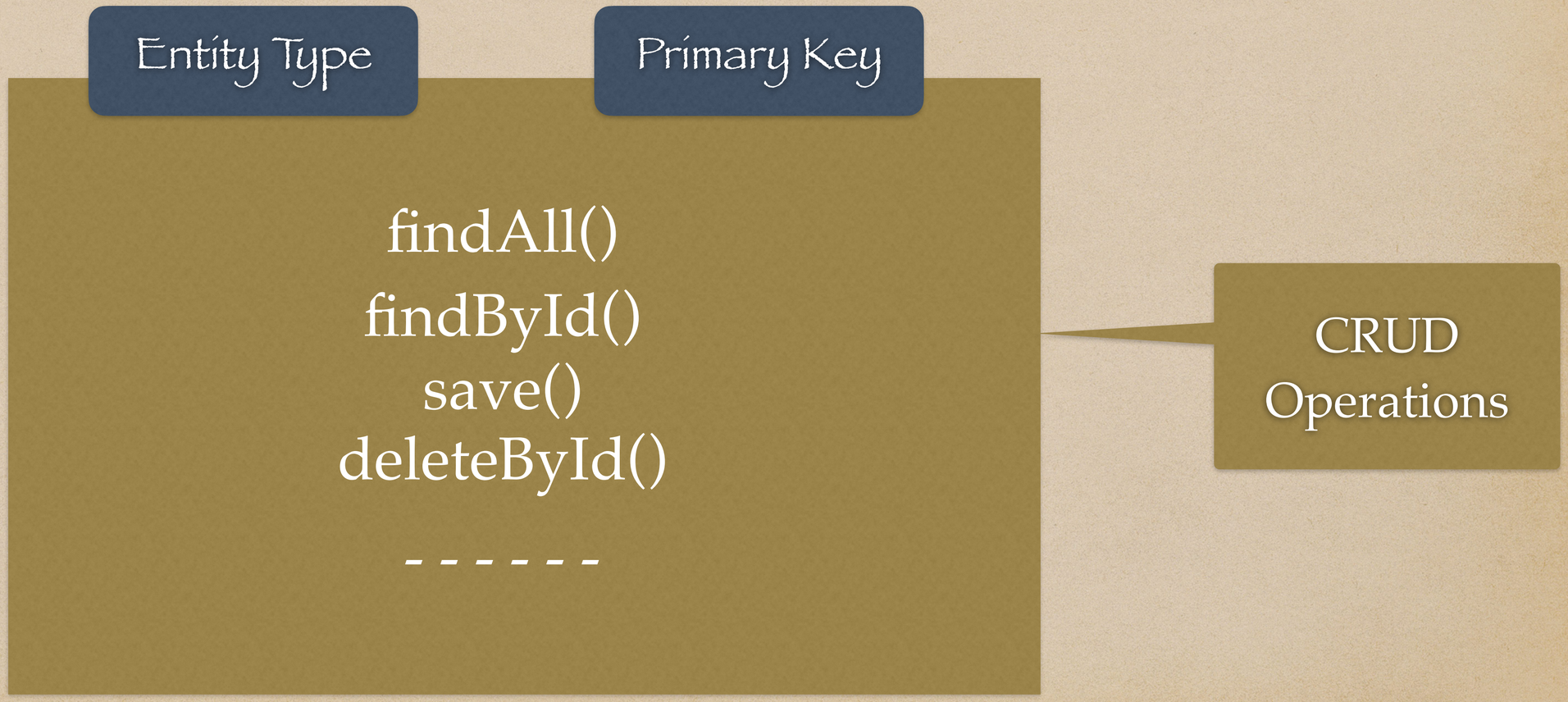
1. **Create an abstract base repository class that provides CRUD operations for entities.**

2. **Create the concrete repository class that extends the abstract base repository class.**

The problem of this approach is that we still have to write the code that creates our database queries and invokes them. To make matters worse, we have to do this every time when we want to create a new database query. This is a waste of time.

What if I would tell you that we can create JPA repositories without writing any boilerplate code (without creating queries unless we required) ?

There Should be Standard Solution Generate Query

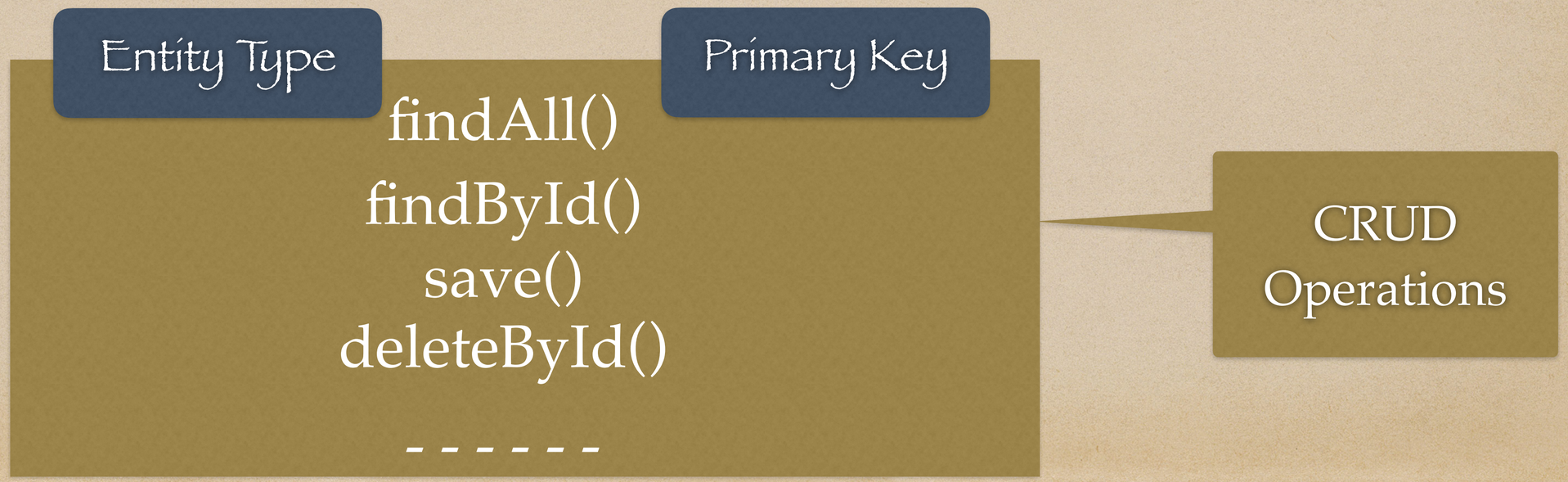


Spring Data JPA - Solution

Reduce the amount of boilerplate code required to implement data access object (DAO) layer.

Spring Data JPA is not a JPA provider. It simply "hides" the Java Persistence API (and the JPA provider) behind its repository abstraction.

Spring Data JPA uses Hibernate as a default JPA provider.



Minimised boilerplate code

Before Spring Data JPA

```java
public interface EmployeeDAO {

	public List<Employee> findAll();

	public Employee findById(int theId);

	public void sav

	public void del

}
```

```java
@Repository
public class EmployeeDAOJpaImpl implements EmployeeDAO {

	private EntityManager entityManager;

	@Autowired
	public EmployeeDAOJpaImpl(EntityManager theEntityManager) {
		entityManager = theEntityManager;
	}

	@Override
	public List<Employee> findAll() {
		// create a query
		Query theQuery = entityManager.createQuery("from Employee");
		// execute query and get result list
		List<Employee> employees = theQuery.getResultList();
		// return the results
		return employees;
	}

	@Override
	public Employee findById(int theId) {
		// get employee
		Employee theEmployee = entityManager.find(Employee.class, theId);
		// return employee
		return theEmployee;
	}

	@Override
	public void save(Employee theEmployee) {
		// save or update the employee
		Employee dbEmployee = entityManager.merge(theEmployee);
		// update with id from db ... so we can get generated id for save/insert
		theEmployee.setId(dbEmployee.getId());
	}

	@Override
	public void deleteById(int theId) {
		// delete object with primary key
		Query theQuery = entityManager.createQuery("delete from Employee where id=:employeeId");
		theQuery.setParameter("employeeId", theId);
		theQuery.executeUpdate();
	}
}
```

2 Files
30+ lines of code

After Spring Data JPA

```java
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

	// that's it ... no need to write any code LOL!

}
```

1 File
3 lines of code

No need for implementation Class

Application Architecture