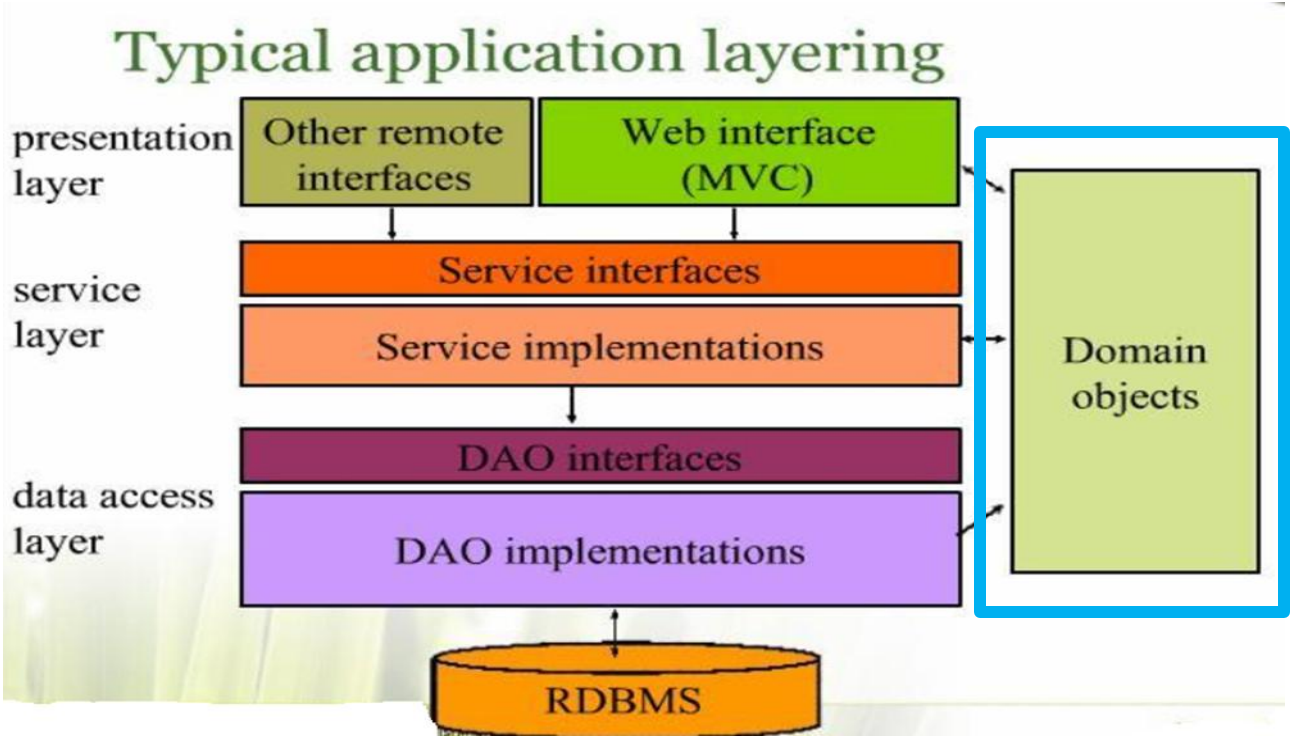
Enterprise Information

Knowledge is Power

The Domain Model is Essential



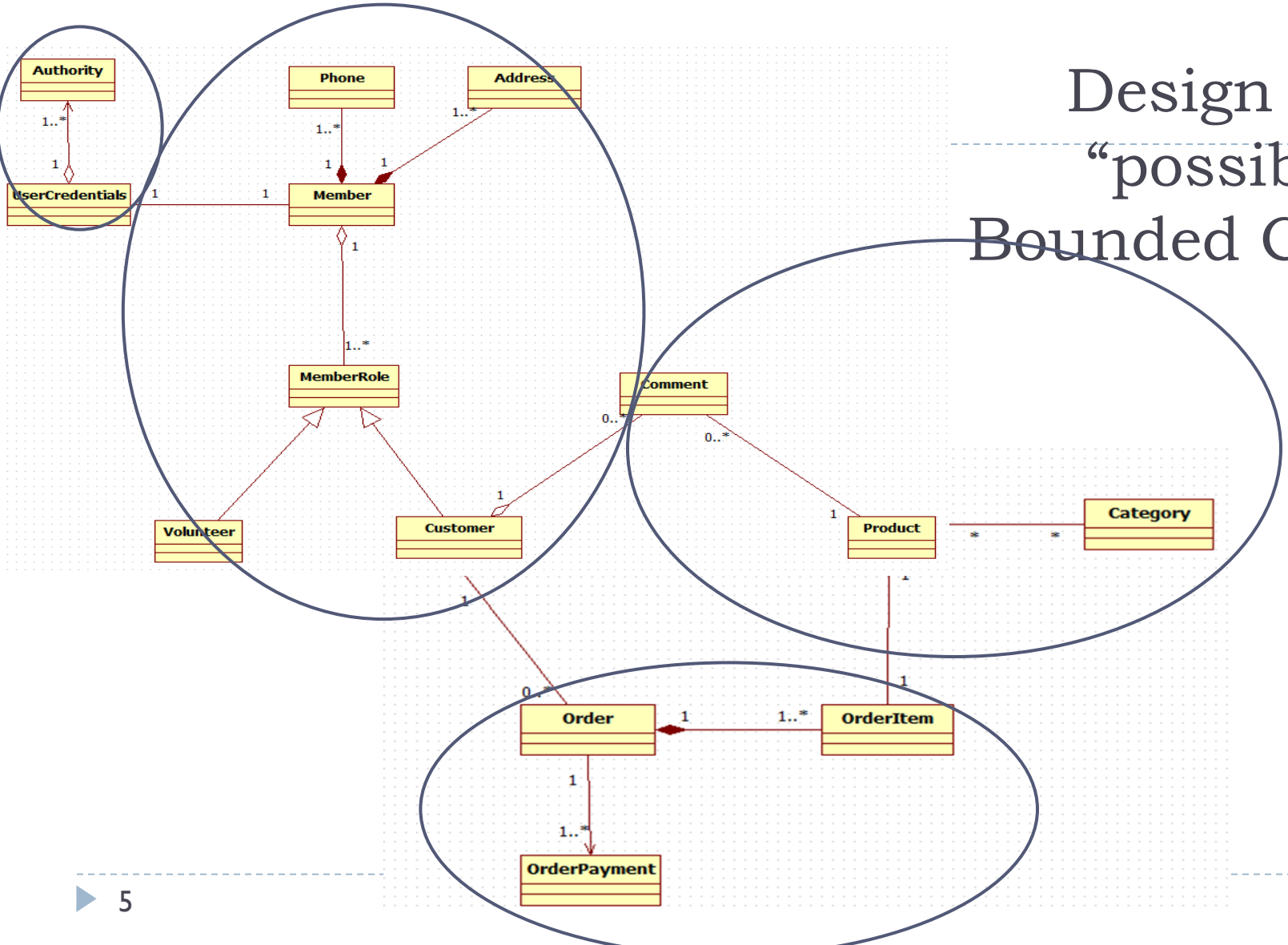
Domain Model

- Application Design is Driven by the Business Domain [Model].
- The domain model is the central, organizing component of the Enterprise.
- ALL application functionality is derived from it.
- The rest of the Enterprise is involved in modifying, validating, moving, translating and presenting …the Domain Model - **DATA**

Domain Driven Design

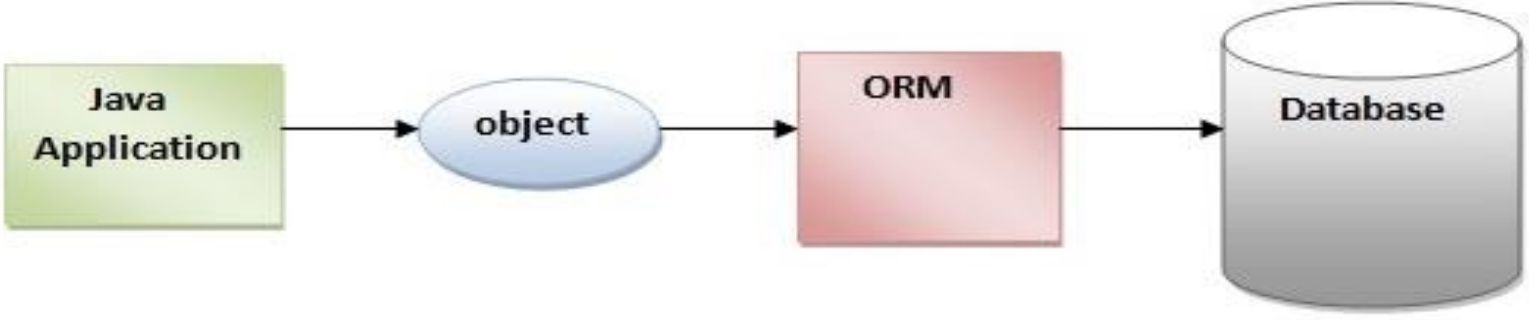
- Technique for clarifying Domain Model complexity With an “eye” towards simplification
- Recognizes that a “single” domain model for a large system is not feasible or cost-effective
- DDD divides up a large system into Bounded Contexts Bounded Context is a central DDD pattern
- DDD influences Microservices

Design Model "possible" Bounded Contexts



Basic Function of ORM

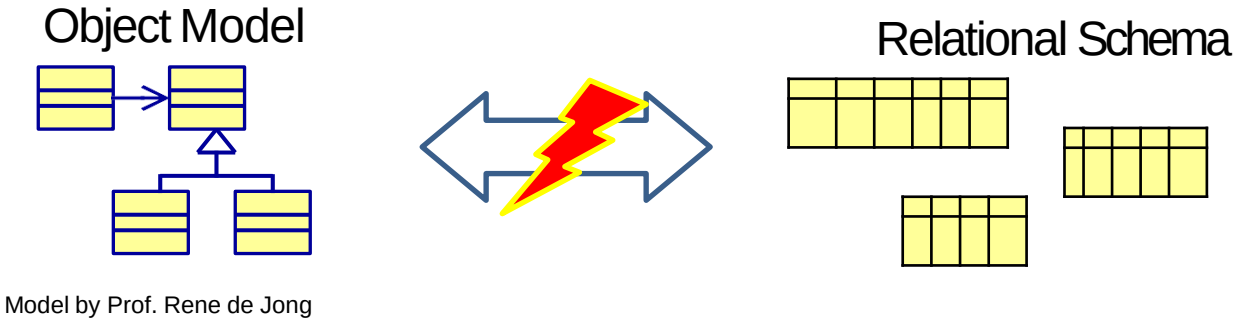
- Acts as a "Gateway" between OO Domain & Relational Database



- Maps Object to Relational Model & vice versa

OO / RDB Mismatch

- OO and Relational can both represent data, but
  - They don't completely match in several areas



Mismatch Categories

Object Oriented	Relational Database
Objects are instantiations of classes and automatically have (mem address) identity (object1 == object2)	In the relational model the table name and a primary key is used to identity a row in a table
Objects have associations (one-to-one, many- to-one, …)	Relational model has foreign keys and link tables
OO has inheritance	Relational model has no such thing
Data can be accessed by following object associations	Data can be accessed using queries and joins

Main Point

An Object Relational Mapping framework provides an Object-Oriented approach to data storage; simplifying the access to the database and effortlessly handling the persistence management for us.

Science of Consciousness: *Transcendental Meditation is an effortless technique to bring us to the simple state of awareness*

ORM Use Case

- For applications based on a rich domain model
  - complex business rules
  - complex interactions between entities
  - Value is dealing with the full complexity of object/relational persistence

- On the other hand
  - An application with only a few entities and simple relationships could be adequately server by direct database table-oriented solutions

WARNING WARNING ORM IS HARD

- **"Let the ORM deal with the database."** – anonymous

BIG MISTAKE!!!

- Works for small applications and loads, but it soon falls apart once "things" get interesting.
- **ORM == 80% of the mapping problems**
- **The other 20% requires "manual labor" at times by a Relational Database expert [RE: DBA type]**

- Basic ORM benefit is automating the grunt work – relieves the developer from writing all that tedious boiler plate CRUD column to property mapping code.

Basic ORM features

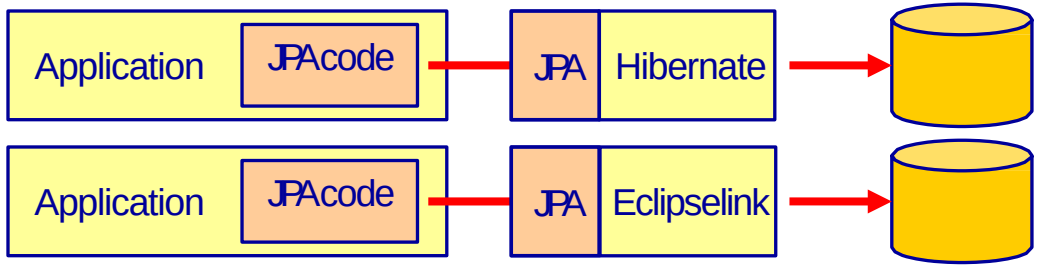
- Mapping Classes To Tables
- Out Of The Box CRUD Functionality
- Hydrating Entities **
- Executing Custom "OO" Queries
- Cache management
- Concurrency support
- Transaction management

** "Automatically" Populate Table data to Object including Relationships

Java Persistence API

- JPA is a specification – not an implementation.
- JPA 1.0 (2006). JPA 2.0 (2009) JPA 2.1(2013)
- Standardizes interface across industry platforms
- Object/Relational Mapping
  - **Specifically Persistence for RDBMS**

- Major Implementations [since 2006]:
  - Toplink - Oracle implementation [donated to Eclipse foundation for merge with Eclipselink 2008]
  - Hibernate - Most deployed framework. Major contributor to JPA specification.
  - OpenJPA - (openjpa.apache.org) which is an extension of Kodo implementation.



Model by Prof. Rene de Jong

Range of ORM implementations

- JDBC "ORM"
- Hibernate XML
- Hibernate Annotations
- Hibernate Spring Transactions
- Hibernate Spring – JPA
- Hibernate Spring Data

JDBC "ORM" - VehicleDAOImpl

```
public void insert(Vehicle vehicle) {
    String sql = "INSERT INTO VEHICLE (VEHICLE_NO, COLOR, WHEEL, SEAT) "
            + "VALUES (?, ?, ?, ?)";
    Connection conn = null;
    try {
        conn = dataSource.getConnection();
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, vehicle.getVehicleNo());
        ps.setString(2, vehicle.getColor());
        ps.setInt(3, vehicle.getWheel());
        ps.setInt(4, vehicle.getSeat());
        ps.executeUpdate();
        ps.close();
    } catch (SQLException e) {
        throw new RuntimeException(e);
    } finally {
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
            }
        }
    }
}
```

Database "managed" transaction
JDBC Connection is in *auto-commit* mode by default

SEE JDBCDao DEMO

Hibernate GenericDAOImpl

BACK

```
public abstract class GenericDaoImpl<T> implements GenericDao<T> {

    @Autowired
    protected SessionFactory sessionFactory;

    protected Session getSession() {
        return sessionFactory.getCurrentSession();
    }

    @Override
    public void save(T entity) {
        Transaction tx = null;
        try {
            tx = this.getSession().beginTransaction();
            this.getSession().save(entity);
            tx.commit();
        } catch (Exception e) {
            if (tx != null) tx.rollback();
            throw e;
        } finally {
            getSession().close();
        }

    }
}
```

Explicitly managed Session & Transaction

Hibernate XML Domain Mapping File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-
mapping-3.0.dtd">
<hibernate-mapping
package="edu.mum.domain">
    <class name="Vehicle" table="VEHICLE">
        <id name="id" column="ID">
            <generator class="native"/>
        </id>
        <property name="color"
column="COLOR" />
        <property name="wheel"
column="WHEEL" />
        <property name="seat" column="SEAT" />
        <property name="vehicleNo"
column="VEHICLE_NO" />
    </class>
</hibernate-mapping>
```

SEE HibernateSolo DEMO

```
@Entity(name = "VEHICLE")
public class Vehicle {
        @Id

    @GeneratedValue(strategy=GenerationType.IDENT
    ITY)
      @Column(name="ID")
      private long id;

      @Column(name="VEHICLE_NO")
      private String vehicleNo;

      @Column(name="COLOR")
      private String color;
      @Column(name="WHEEL")
       private int wheel;
      @Column(name="SEAT")
       private int seat;

      //default constructor, setters&getters
}
```

SEE HibernateAnnotations DEMO

Spring ORM Support

- Comprehensive transaction support is one of the compelling reasons to use the Spring Framework.

- Integration with Hibernate, Java Persistence API (JPA)…
- Hibernate Support
  - First-class integration support through IoC/DI
  - Easier testing Resource management
  - Integrated transaction management

Spring Framework Data Access

Hibernate Spring Managed Transactions

```
public abstract class GenericDaoImpl<T> implements GenericDao<T> {

    @Autowired
    protected SessionFactory sessionFactory;

    // Opens session   - clsed when TX ends
    protected Session getSession() {
        return sessionFactory.getCurrentSession();
    }

    public void save(T entity) {
        this.getSession().save(entity);
    }
}
```

Reduction in code*:
manage transaction
open/close session
*** Compared to** *Hibernate Solo*

```
@Service
@Transactional
public class VehicleServiceImpl implements VehicleService {

    @Autowired
    private VehicleDao vehicleDao;

    public void update( Vehicle vehicle) {
        vehicleDao.update(vehicle);
    }
}
```

→ **Starts Transaction**

SEE HibernateTransactions DEMO

JPA Example

```
public abstract class GenericDaoImpl<T> implements
    GenericDao<T> {

        @PersistenceContext
        protected EntityManager entityManager;


        @Override
        public void save(T entity) {
            entityManager.persist(entity);
        }
}
```

EntityManager "replaces" Session

SEE
HibernateSpringJPA DEMO
HibernateSpringJPAJConfig
HibernateSpringBoot

JPQL - Data Object Queries

- JPQL is different from SQL in that it operates on objects, attributes and relationships instead of tables and columns.
- Queries are declared in the DAO implementation

```
@Repository
public class MemberDaoImpl extends GenericDaoImpl<Member> implements MemberDao {

   public MemberDaoImpl() {
      super.setDaoType(Member.class );
      }

   public Member findByMemberNumber(Integer number) {

      Query query = entityManager.createQuery("select m from MEMBER m  where m.memberNumber
=:number");
      return (Member) query.setParameter("number", number).getSingleResult();

   }

}
```

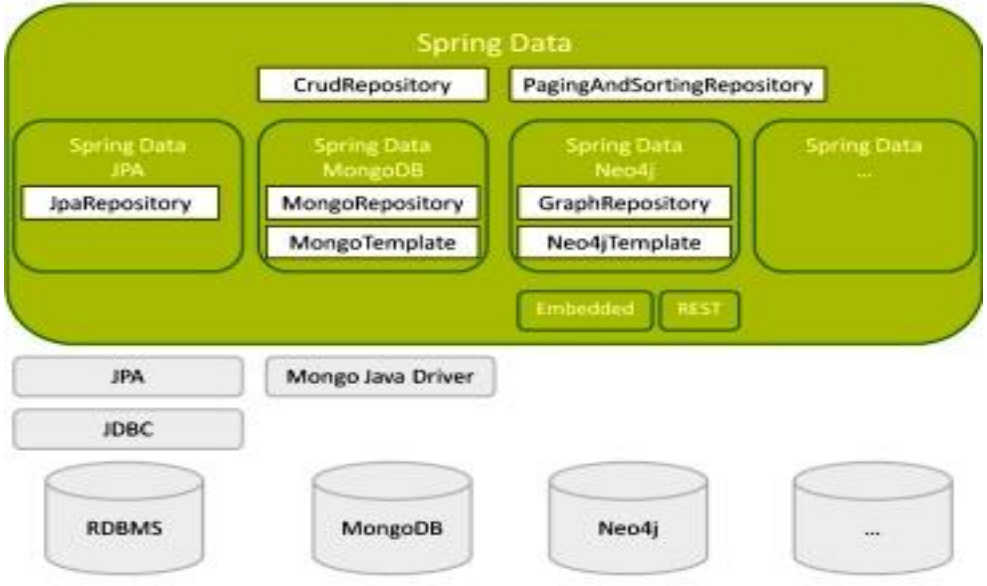
Spring Data

- Spring Data
  - High level Spring project whose purpose is to unify and ease the access to different kinds of persistence stores, both relational database systems and NoSQL data stores.

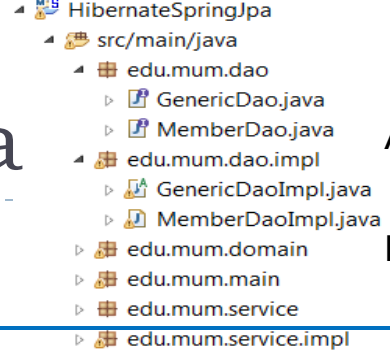
- Hibernate ORM
  - (Hibernate for short) is an object-relational mapping Java library; a framework for mapping an object-oriented domain model to a traditional relational database. Distributed under the GNU Lesser General Public License

Spring Data Project

- High level Spring project whose purpose is to unify and ease the access to different kinds of persistence stores, both relational database systems and NoSQL data stores.

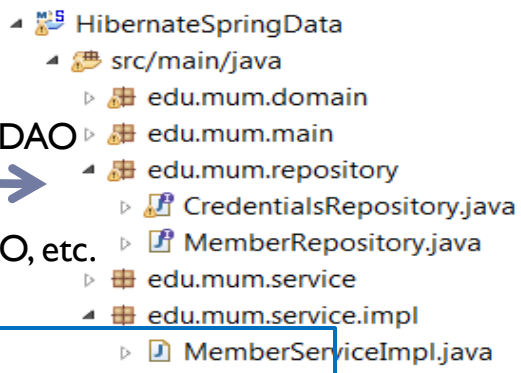


Spring Data



AUTO-GENERATES the DAO

No Need for GenericDAO, etc.




```
@Repository
    public class MemberDaoImpl extends GenericDaoImpl<Member> implements MemberDao {

      public MemberDaoImpl() {
        super.setDaoType(Member.class );
        }

      public Member findByMemberNumber(Integer number) {

        Query query = entityManager.createQuery("select m from MEMBER m  where m.memberNumber =:number");
        return (Member) query.setParameter("number", number).getSingleResult();

      }

     }
```

```
    @Repository
    public interface MemberRepository extends CrudRepository<Member, Long> {
        Member findByMemberNumber(int memberNumber);
    }
```


See HibernateSpringData

Main Point

- JPA is a specification not an implementation. It provides a consistent, reliable mechanism for data storage and retrieval that alleviates the application developer from the details involved in the persistence layer.

- *The mechanism of transcending allows the individual to tap into the Home of all the Laws of Nature alleviating the stress of mundane day to day issues.*