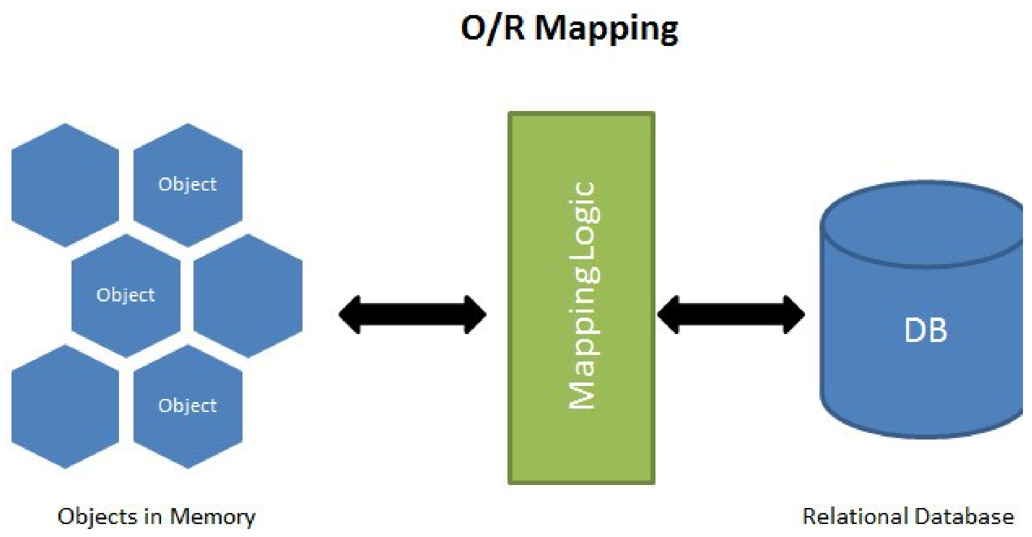
Spring Data JPA

Piya Lumyong

Object Relational Mapping

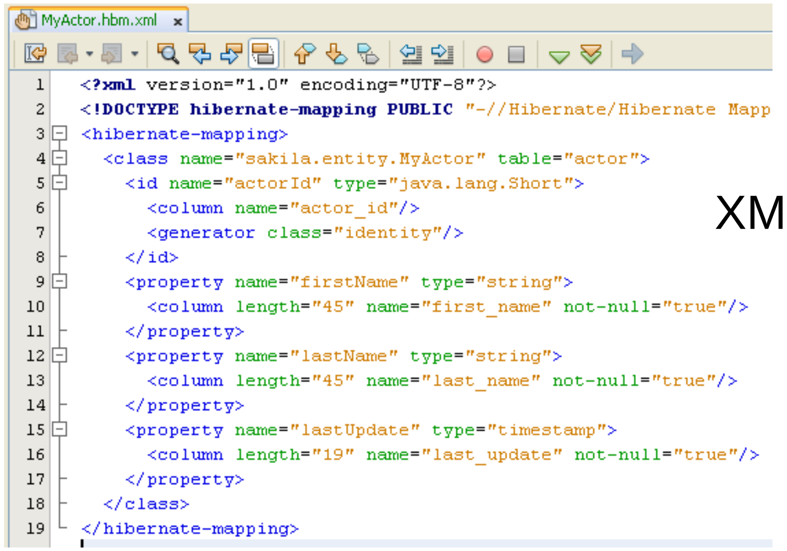


Mapping

Annotation

```
@Entity
@Table(name="employee")
public class Employee {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name="emp_id")
    private Integer emp_id;

    @Column(name="emp_first_name")
    private String emp_first_name;

    @Column(name="emp_last_name")
    private String emp_last_name;

    @Column(name="emp_start_date")
    private Timestamp emp_start_date;

    @Column(name="emp_end_date")
    private Timestamp emp_end_date;

    @Column(name="designation_id")
    private Integer designation_id;

    @ManyToOne
    @JoinColumn(name = "designation_id", referencedColumnName = "designation_id", insertable = false, updatable = false, nullable = true)
    private Designation designation;

    @OneToMany
    @JoinColumn(name= "emp_id", referencedColumnName = "emp_id", insertable = false, updatable = false)
    private Set<RoleEmployee> employeeRoleAssn;

    @OneToMany
    @JoinColumn(name= "emp_id", referencedColumnName = "emp_id", insertable = false, updatable = false)
    private Set<ProjectEmployee> employeeProjectAssn;
```

XML


MyActor.hbm.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapp
<hibernate-mapping>
  <class name="sakila.entity.MyActor" table="actor">
    <id name="actorId" type="java.lang.Short">
      <column name="actor_id"/>
      <generator class="identity"/>
    </id>
    <property name="firstName" type="string">
      <column length="45" name="first_name" not-null="true"/>
    </property>
    <property name="lastName" type="string">
      <column length="45" name="last_name" not-null="true"/>
    </property>
    <property name="lastUpdate" type="timestamp">
      <column length="19" name="last_update" not-null="true"/>
    </property>
  </class>
</hibernate-mapping>
```

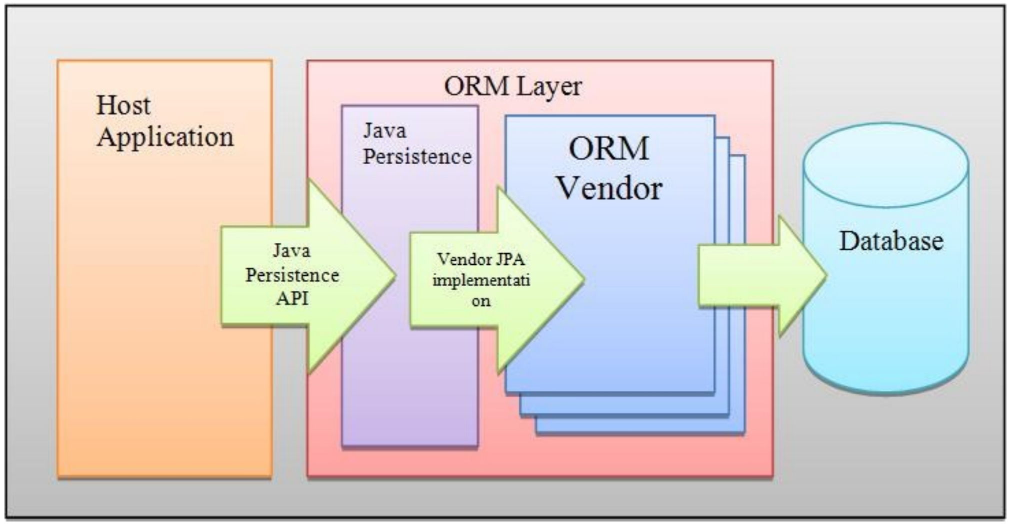
Manay Vendor

HIBERNATE

OpenJPA

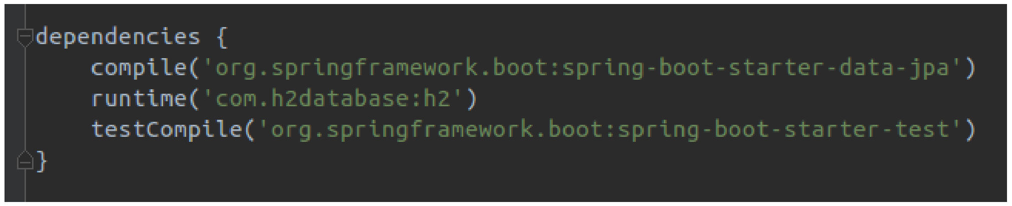
eclipselink

Java Persistence API JSR 338



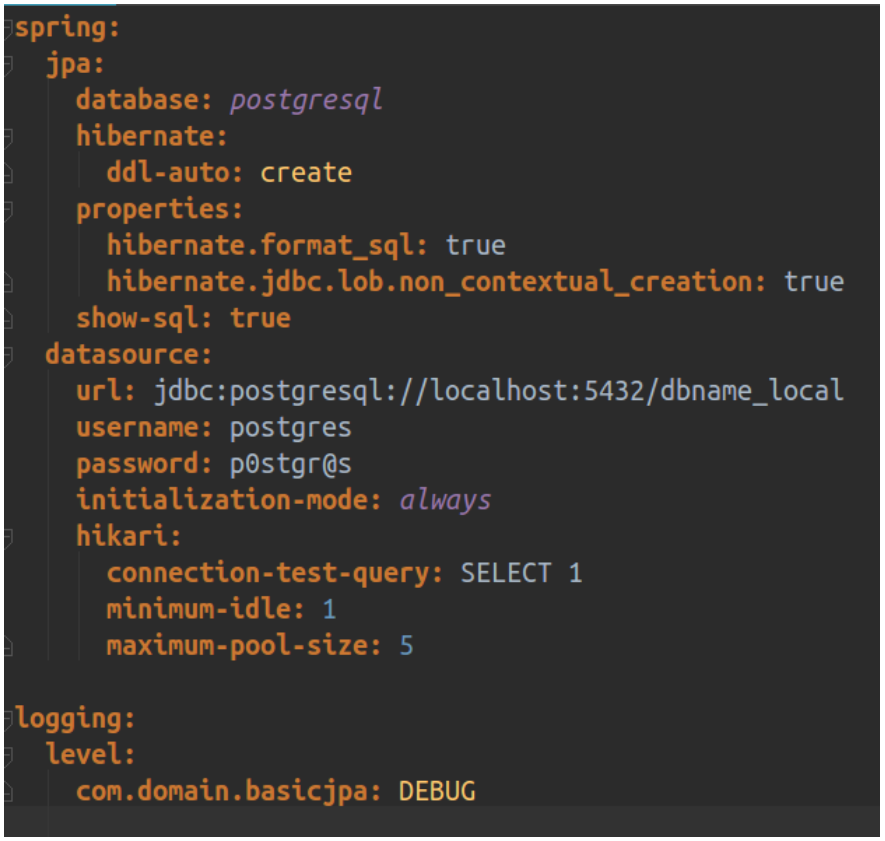
Spring Data JPA

Just add dependency, Automatically configure:
- Embedded DB
- Persistence Context
- Rollback after test



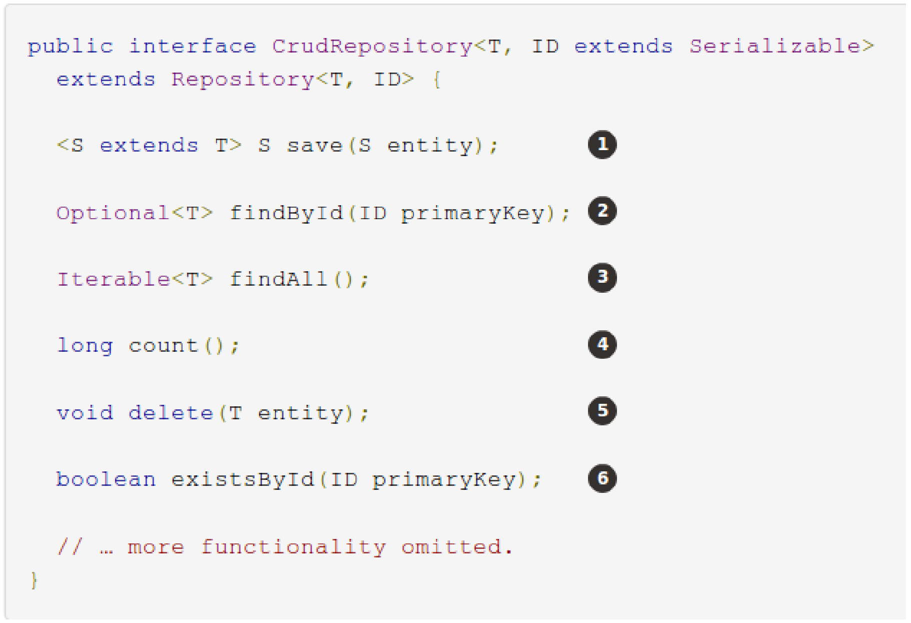
```
dependencies {
    compile('org.springframework.boot:spring-boot-starter-data-jpa')
    runtime('com.h2database:h2')
    testCompile('org.springframework.boot:spring-boot-starter-test')
}
```

Custom Configuration



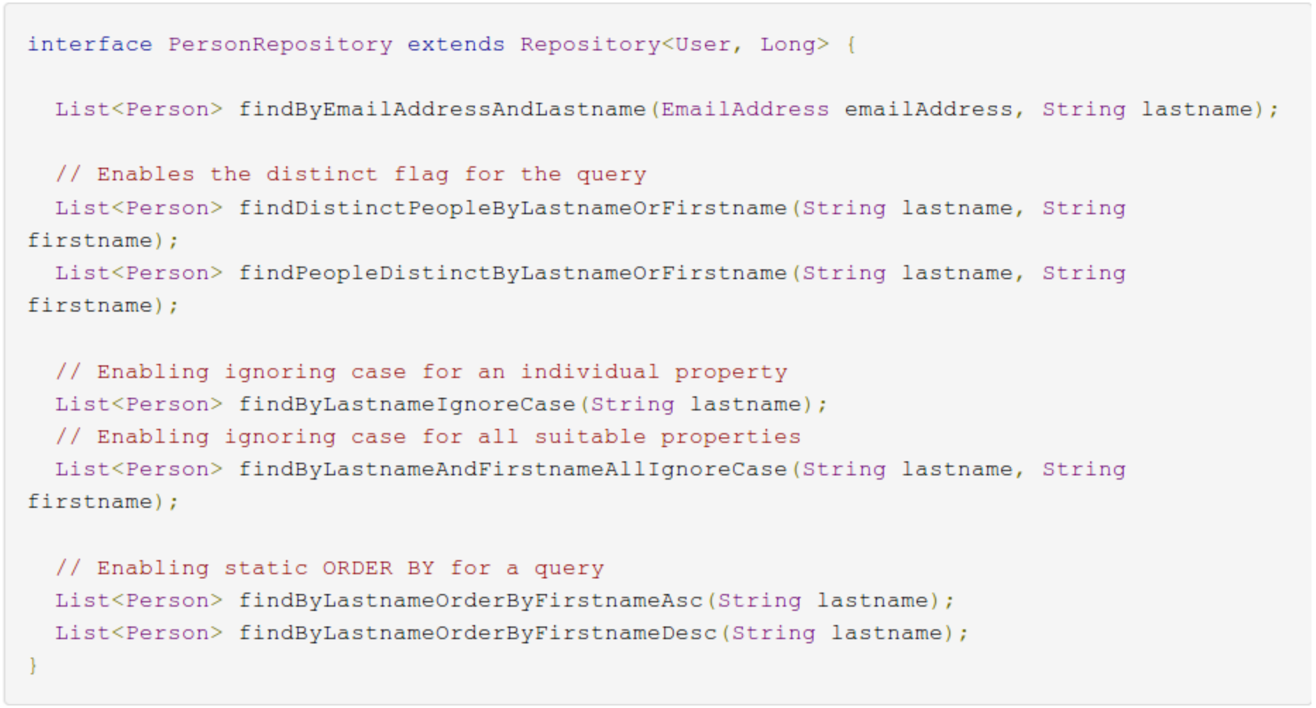
```yaml
spring:
  jpa:
    database: postgresql
    hibernate:
      ddl-auto: create
    properties:
      hibernate.format_sql: true
      hibernate.jdbc.lob.non_contextual_creation: true
    show-sql: true
  datasource:
    url: jdbc:postgresql://localhost:5432/dbname_local
    username: postgres
    password: p0stgr@s
    initialization-mode: always
    hikari:
      connection-test-query: SELECT 1
      minimum-idle: 1
      maximum-pool-size: 5

logging:
  level:
    com.domain.basicjpa: DEBUG
```

CrudRepository

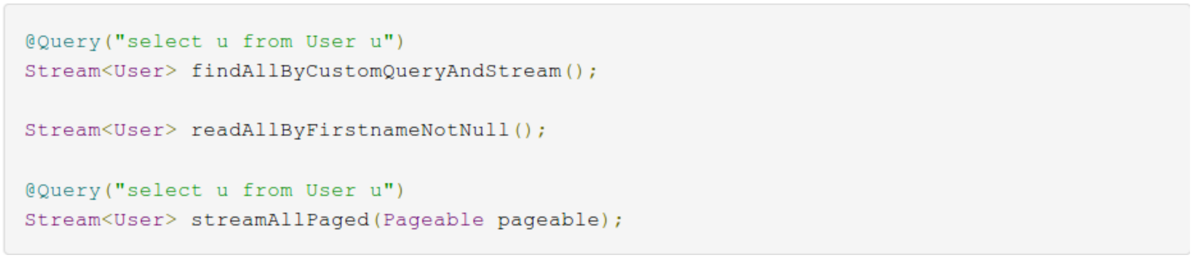


```
public interface CrudRepository<T, ID extends Serializable>
  extends Repository<T, ID> {

  <S extends T> S save(S entity);      ❶

  Optional<T> findById(ID primaryKey); ❷

  Iterable<T> findAll();               ❸

  long count();                        ❹

  void delete(T entity);               ❺

  boolean existsById(ID primaryKey);   ❻

  // … more functionality omitted.
}
```

Custom Repository

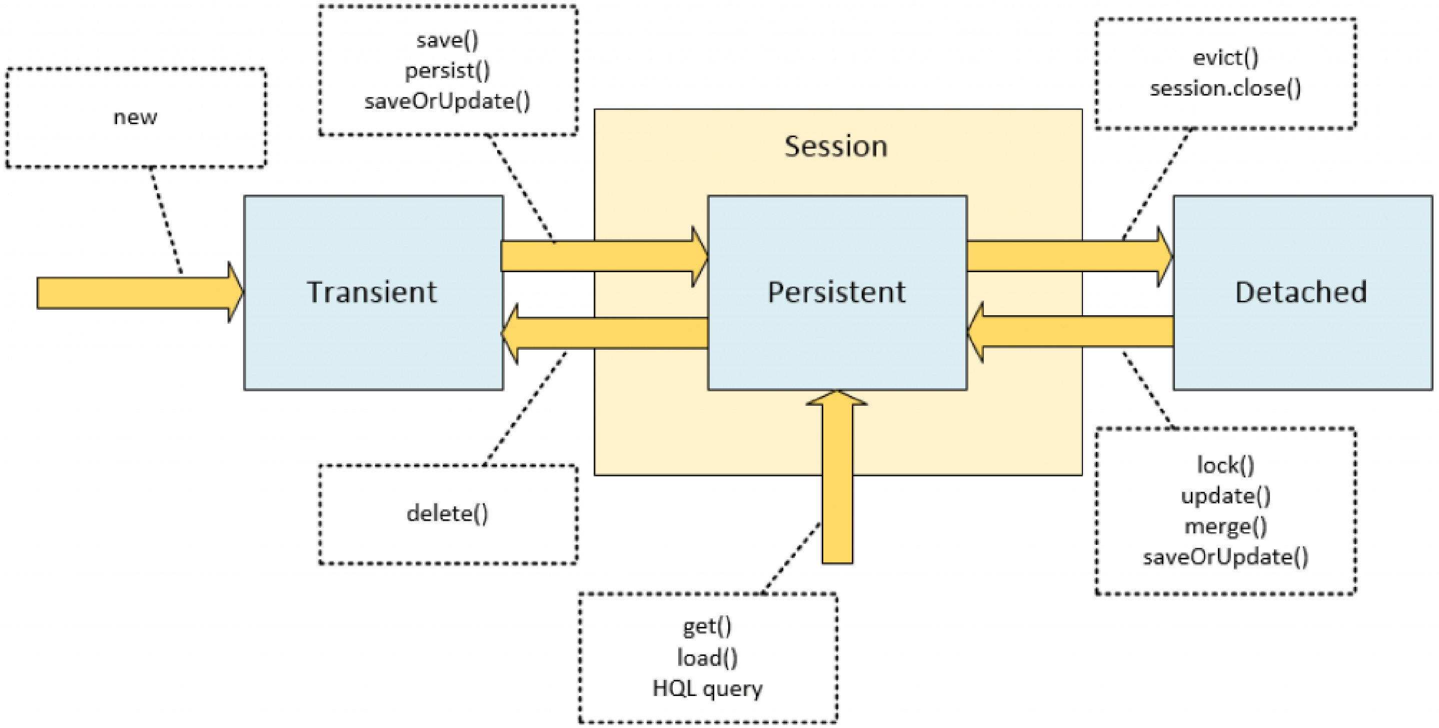


```
interface PersonRepository extends Repository<User, Long> {

  List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);

  // Enables the distinct flag for the query
  List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String
firstname);
  List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String
firstname);

  // Enabling ignoring case for an individual property
  List<Person> findByLastnameIgnoreCase(String lastname);
  // Enabling ignoring case for all suitable properties
  List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String
firstname);

  // Enabling static ORDER BY for a query
  List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
  List<Person> findByLastnameOrderByFirstnameDesc(String lastname);
}
```



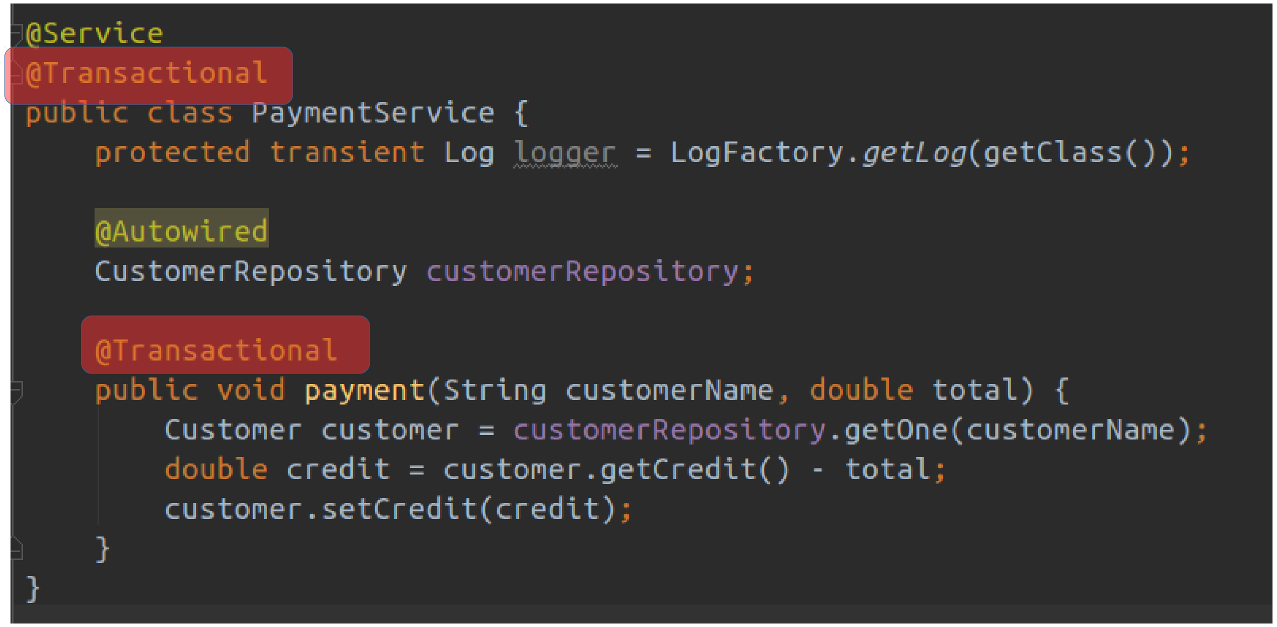
```
@Query("select u from User u")
Stream<User> findAllByCustomQueryAndStream();

Stream<User> readAllByFirstnameNotNull();

@Query("select u from User u")
Stream<User> streamAllPaged(Pageable pageable);
```

States of Entity Instances



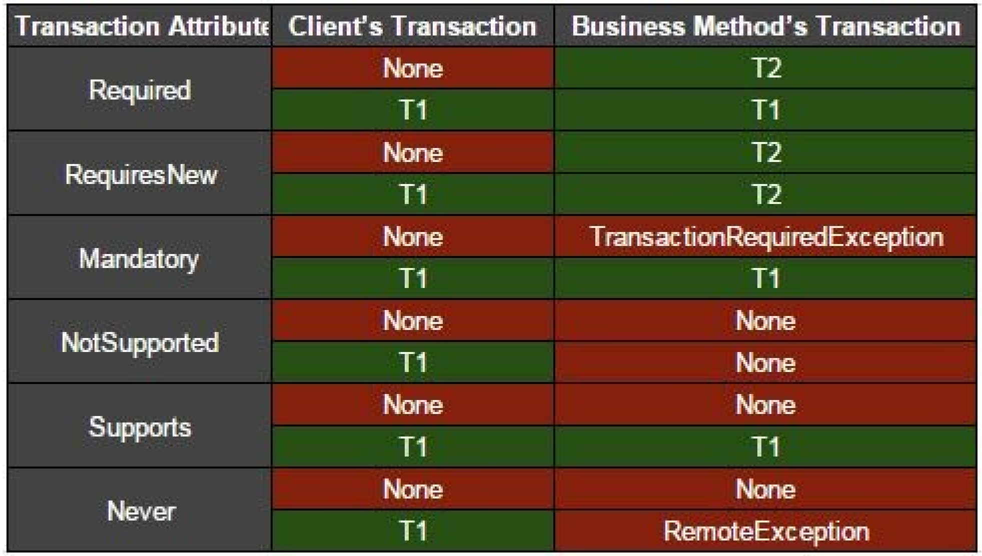
Ref. States of Entity Instances

Declaration Transaction



```
@Service
@Transactional
public class PaymentService {
    protected transient Log logger = LogFactory.getLog(getClass());

    @Autowired
    CustomerRepository customerRepository;

    @Transactional
    public void payment(String customerName, double total) {
        Customer customer = customerRepository.getOne(customerName);
        double credit = customer.getCredit() - total;
        customer.setCredit(credit);
    }
}
```

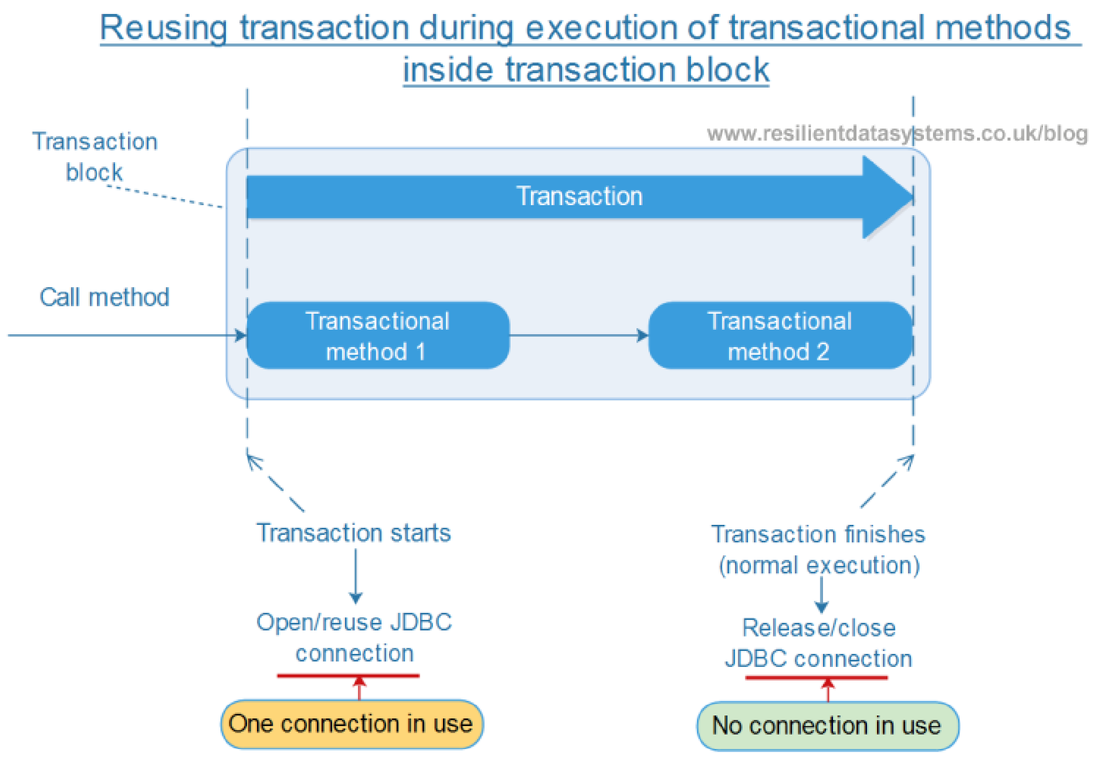
Transaction Propagate



Transaction Attribute	Client's Transaction	Business Method's Transaction
Required	None	T2
	T1	T1
RequiresNew	None	T2
	T1	T2
Mandatory	None	TransactionRequiredException
	T1	T1
NotSupported	None	None
	T1	None
Supports	None	None
	T1	T1
Never	None	None
	T1	RemoteException

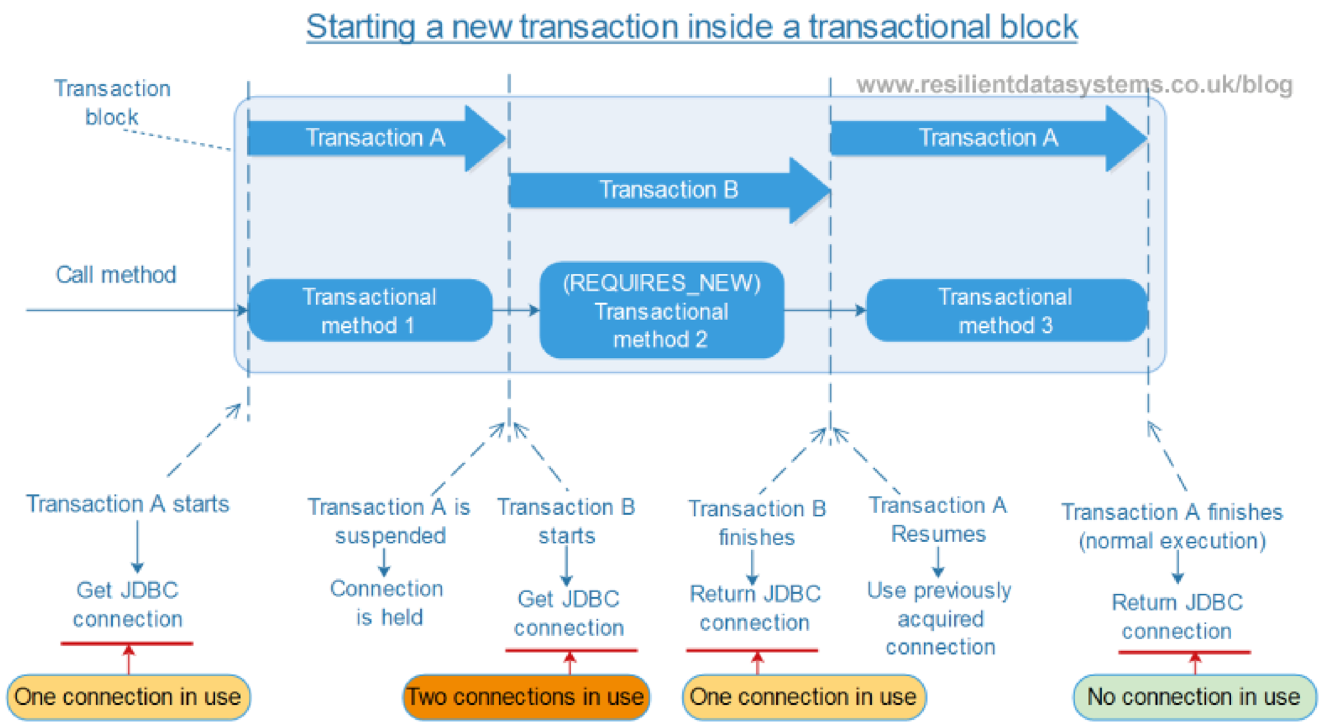
Container Management Transaction

Require Propagate



Reusing transaction during execution of transactional methods inside transaction block
Transaction block
www.resilientdatasystems.co.uk/blog
Transaction
Call method
Transactional method 1
Transactional method 2
Transaction starts
Open/reuse JDBC connection
One connection in use
Transaction finishes (normal execution)
Release/close JDBC connection
No connection in use

RequireNew Propagate



Starting a new transaction inside a transactional block
www.resilientdatasystems.co.uk/blog
Transaction block
Transaction A
Transaction B
Transaction A
Call method
Transactional method 1
(REQUIRES_NEW) Transactional method 2
Transactional method 3
Transaction A starts
Get JDBC connection
One connection in use
Transaction A is suspended
Connection is held
Transaction B starts
Get JDBC connection
Two connections in use
Transaction B finishes
Return JDBC connection
One connection in use
Transaction A Resumes
Use previously acquired connection
Transaction A finishes (normal execution)
Return JDBC connection
No connection in use