

The background features a large, abstract, wavy shape in shades of green. The shape flows from the left side, arching upwards and then downwards towards the bottom right. It has a soft, ethereal quality with some transparency, allowing the white background to show through. A solid dark green horizontal bar runs along the very bottom of the image.

# **Spring Transaction**

# Transaction

- 한 개 이상의 물리적인 동작으로 구성된 논리적인 작업 단위
- 트랜잭션에 포함된 물리적인 작업이 모두 성공하거나 실패하도록 관리
- ACID 특성
  - 원자성 (Atomicity)
  - 일관성 (Consistency)
  - 격리성 (Isolation)
  - 영속성 (Durability)

# Spring 트랜잭션 지원

- 프로그래밍 방식

- TransactionTemplate 클래스 사용

- AOP 방식

- <tx:advice>와 <aop:advisor> 설정을 사용하는 선언적 트랜잭션 방식

- 어노테이션 방식

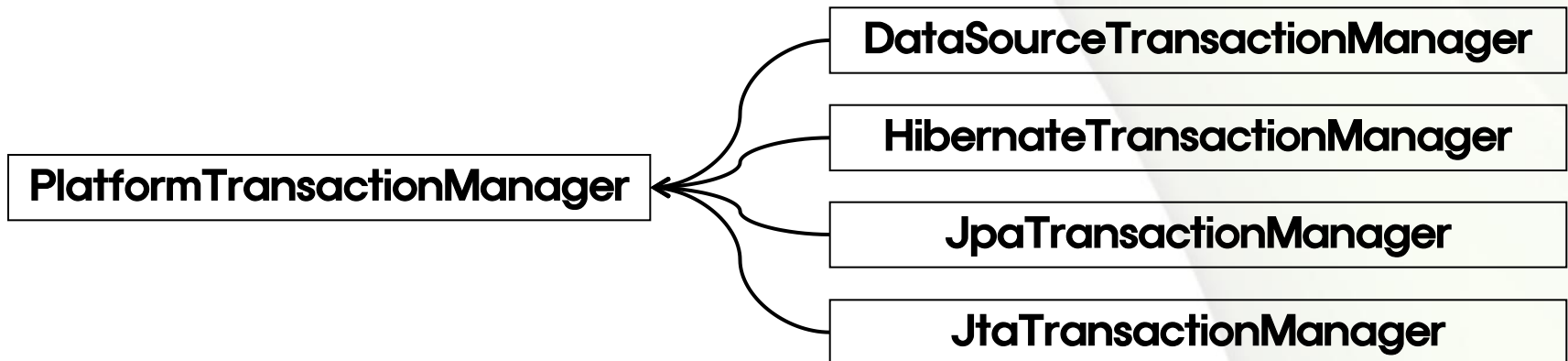
- @Transaction 어노테이션을 사용하는 선언적 트랜잭션 방식

# Spring 트랜잭션 관리자

- 스프링 프레임워크는 데이터 연동 기술에 따라 특화된 트랜잭션 관리자 제공
  - 모든 트랜잭션 관리자는 PlatformTransactionManager 인터페이스 구현

- 종류

트랜잭션 관리자	설명
DataSourceTransactionManager	MyBatis와 같은 JDBC 기반 트랜잭션 관리
HibernateTransactionManager	Hibernate 프레임워크 기반 트랜잭션 관리
JpaTransactionManager	JPA 기반 트랜잭션 관리
JtaTransactionManager	분산 트랜잭션 지원



# 트랜잭션 관리자 구성

## ▪ DataSourceTransactionManager

```
<bean id="txManager"  
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  
    <property name="dataSource" ref="dataSource" />  
</bean>
```

## ▪ HibernateTransactionManager

```
<bean id="txManager"  
    class="org.springframework.orm.hibernate4.HibernateTransactionManager">  
    <property name="sessionFactory" ref="sessionFactory" />  
</bean>
```

## ▪ JpaTransactionManager

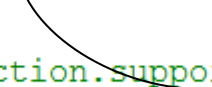
```
<bean id="jpaTransactionManager"  
    class="org.springframework.orm.jpa.JpaTransactionManager">  
    <property name="entityManagerFactory" ref="entityManagerFactory" />  
</bean>
```

# TransactionTemplate 트랜잭션 구현

## ▪ 빈 선언 및 의존성 주입

```
<bean id="txManager"  
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  
    <property name="dataSource" ref="dataSource" />  
</bean>
```

```
<bean id="transactionTemplate"  
    class="org.springframework.transaction.support.TransactionTemplate" >  
    <property name="transactionManager" ref="txManager" />  
</bean>
```



# TransactionTemplate 트랜잭션 구현

## ■ 트랜잭션 적용

```
public void deleteCustomer(final long id) {  
    transactionTemplate.execute(new TransactionCallback<Void>() {  
        @Override  
        public Void doInTransaction(TransactionStatus txStatus) {  
            try {  
                if(id == 0) {  
                    List<CustomerEntity> customers = repository.findAll();  
                    for(CustomerEntity customer : customers)  
                        repository.delete(customer.getId());  
                }  
                else  
                    repository.delete(id);  
            }  
            catch(Exception ex) {  
                logger.info("삭제를 취소하고 롤백합니다!!");  
                txStatus.setRollbackOnly();  
            }  
            return null;  
        }  
    });  
}
```

# 선언적 트랜잭션 속성

## ▪ 속성 종류

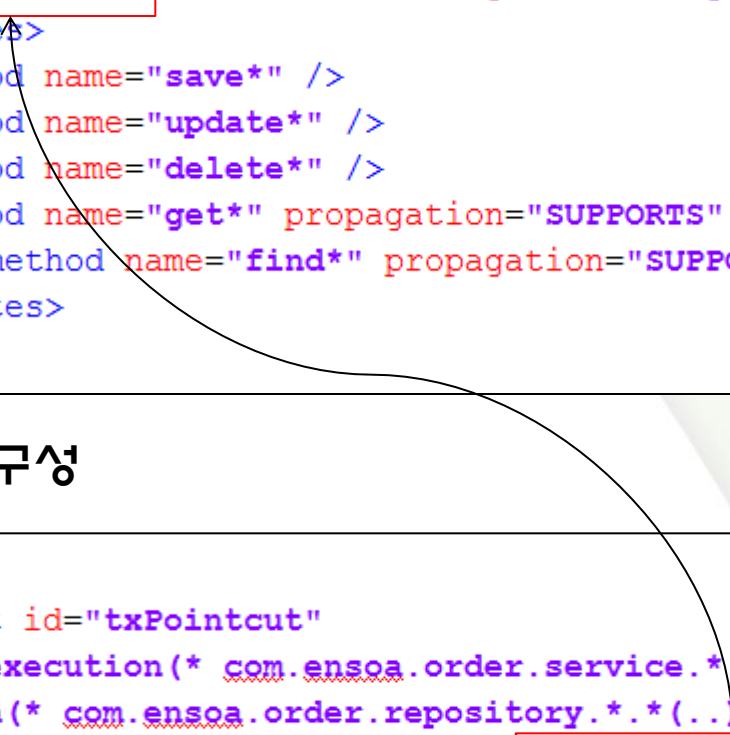
속성 이름	설명
전파 행위	트랜잭션 영역 설정
분리 수준	한 트랜잭션이 다른 트랜잭션에 영향 받는 정도 설정
읽기 전용	데이터 읽기 최적화 적용
타임아웃	트랜잭션 타임아웃 시간 설정
롤백 규칙	롤백이 발생하는 예외 지정



# AOP 트랜잭션

## ■ 트랜잭션 어드바이스 등록

```
<tx:advice id="txAdvice" transaction-manager="txManager">
  <tx:attributes>
    <tx:method name="save*" />
    <tx:method name="update*" />
    <tx:method name="delete*" />
    <tx:method name="get*" propagation="SUPPORTS" read-only="true"/>
    <tx:method name="find*" propagation="SUPPORTS" read-only="true"/>
  </tx:attributes>
</tx:advice>
```



## ■ 트랜잭션 AOP 구성

```
<aop:config>
  <aop:pointcut id="txPointcut"
    expression="execution(* com.ensoa.order.service.*(..)) ||
      execution(* com.ensoa.order.repository.*(..))"/>
  <aop:advisor pointcut-ref="txPointcut" advice-ref="txAdvice" />
</aop:config>
```

# AOP 트랜잭션

## ■ 트랜잭션 적용

```
public void deleteCustomer(long id) {  
    try {  
        if(id == 0) {  
            List<CustomerEntity> customers = repository.findAll();  
            for(CustomerEntity customer : customers)  
                repository.delete(customer.getId());  
        }  
        else  
            repository.delete(id);  
    } catch (RuntimeException e) {  
        logger.info("삭제를 취소하고 롤백합니다!!");  
        TransactionAspectSupport.currentTransactionStatus().setRollbackOnly();  
    }  
}
```

# 트랜잭션 어노테이션

## ■ 스프링 어노테이션 빈 설정

```
<tx:annotation-driven transaction-manager="txManager"/>
```

## ■ 트랜잭션 적용

- 클래스 또는 인터페이스에 지정해서 하위의 모든 메서드에 트랜잭션 적용

```
@Service("customerService")  
@Transactional  
public class CustomerServiceTxAnnotation implements CustomerService {
```

- 메서드에 지정해서 해당 메서드에 트랜잭션 적용

```
@Transactional  
public void deleteCustomer(long id) {
```

# 트랜잭션 설정 가이드

- 트랜잭션은 서비스 레이어에서 시작
- 삽입, 삭제, 변경 기능을 수행하는 메서드에 기본 값인 TRANSACTION\_REQUIRED를 지정해서 기존 트랜잭션에 참여하거나 또는 새 트랜잭션을 시작하도록 설정
  - 읽기 메서드에는 트랜잭션이 필요 없으므로 읽기 최적화 지정
- 웹 애플리케이션의 컨트롤러에 트랜잭션 설정을 피하는 것이 권장됨
- Repository 클래스에는 트랜잭션이 필요한 메서드에 TRANSACTION\_SUPPORTED를 지정해서 기존 트랜잭션에 참여하도록 설정
- RuntimeException 에 대해서만 롤백 하도록 기본 설정 유지
  - 직접 예외 처리할 경우 자동으로 Rollback되지 않으므로 UnexpectedRollbackException 예외에서 명시적인 롤백 처리