(Video 12 Spring Boot @Transactional Annotation - Part 1)

Pre-Requisite to watch this video is

* Concurrency Control
* AOP (Aspect oriented Programming)

Recap about Critical section :-

Code Segment where shared resources are being accessed and modified. Lets say 4 customers are trying to book the same cab.

All will show CAB booking done if no LOCK is there.

```
{ Read Car Row with id: 1001
    if Status is Available:
        Update it to Booked.
}
```

Solution to this is usage of Transaction

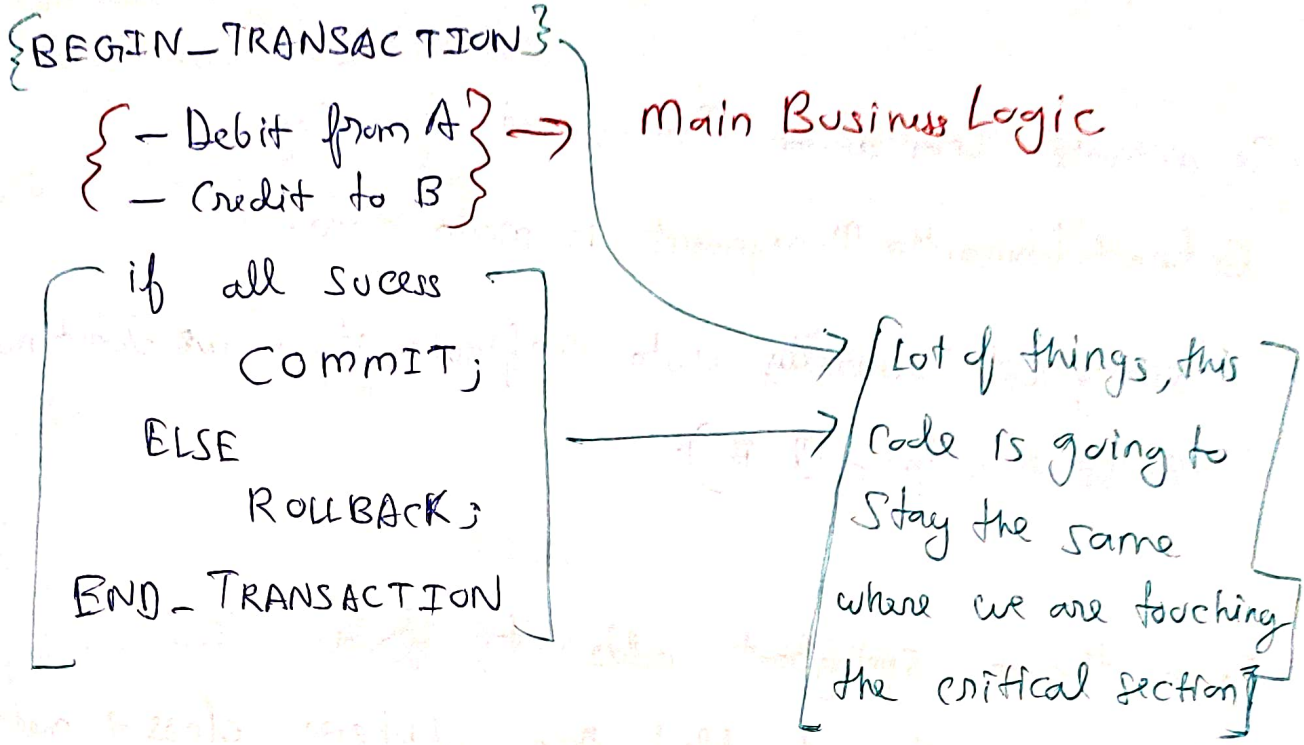
A (Atomicity) → Any operation fails, entire transaction is rolled back

C (Consistency) → Ensures DB fetch before and after transaction. should be consistent. All calls should fetch the same result

I (Isolation) → Even though multiple transactions happen together but transactions should run agnostic to each other.

Durability → (Persistence of transactions is very Imp)

for maintaining Atomicity



So if 100 such methods are there which are critical section, then it would be redundant to write the code to maintain ACID property

@Transactional annotation → does the Job for You.

Few Dependencies we have to add.

1) Based on type of DB we are using, we add dependency

```
<dependency>
    <groupId> org.springframework.boot </groupID>
    <artifactID> Spring-boot-starter-data-jpa </artifactID>
</dependency>
```

2) Database driver dependency is also required
(Next video we shall see)

Activate Transaction Management

To activate Transaction Management by using @EnableTransactionManagement in main class.

(spring boot generally auto configures it, so we don't need to specifically add it).

Even though spring boot adds it, ~~if it~~

Lets suppose its not added, then whichever class & method has [@Transactional] will not be read.

```
@SpringBootApplication
@EnableTransactionManagement
public class SpringBootApplication{

    psvm(){ };

}
```

→ Enabled, Now Spring Boot will scan @Transactional classes

@Transactional.

At Class Level

→ Transaction will only be applied to all the public methods of the class

At method level

→ Transaction applied to particular method only

```
@Component
@Transactional
public class CarService{

    public void updateCar(){

    }

    public void updateCarsBulk(){

    }

    private void helperMethod(){

    }

}
```

(Class Level)

```
@Component
public class CarService{

    @Transactional
    public void updateCar(){

    }

    public void updateCarsBulk(){

    }

}
```

(method Level).

Transactional Management in Spring Boot uses AOP.

- Uses PointCut expression to search @@within(org.springframework.transaction.annotation.Transactional)
- Once PointCut expression matches, run @Around type Advice (TransactionalInterceptor.invokeWithinTransaction)

AoP → As we saw earlier, helps us focus on the main business logic and ~~lets us do~~ it takes care of duplicate code..

Example)

```
@RestController
@RequestMapping("/api")
public class UserController{

    @Autowired
    User user;

    @GetMapping("/update user")
    public String updateUser(){

        user.updateUser();

        return "          ";

    }
}

@Component
public class User{

    @Transactional
    public void updateUser(){

        sout("UPDATE QUERY To update user values");

    }

}
```

What happens inside the interceptor?

TransactionalInterceptor.

invokeWithinTransaction()

(~~XXX~~) V-Imp [~~All~~ @transactional goes through this]

BEGIN_TRANSACTION

↳ createTransactionIfNecessary();

(YOUR TASK) try{

retVal = invocation.proceed WithInvocation();

}

catch(Throwable ex){

completeTransactionAfterThrowing();

↓ ERROR.

Rollback}

finally{

cleanupTransactionInfo();

}

commitTransactionInfo();

↓

(All sucess } then Commit)