What does @transactional annotation do?

It ensures that either entire job would be committed or completely roll back the operation.

// Accounts.java

```
package com.example.transactional;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
@Entity
public class Account {
    @Id
    @GeneratedValue(strategy = GenerationType. AUTO )
    int id;
    String accountHolder;
    double deposits;
    public Account(String accountHolder, double deposits) {
        this.id = id;
        this.accountHolder = accountHolder;
        this.deposits = deposits;
    }
    Account()
    {
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getAccountHolder() {
        return accountHolder;
    }
    public void setAccountHolder(String accountHolder) {
        this.accountHolder = accountHolder;
    }
    public double getDeposits() {
        return deposits;
    }
    public void setDeposits(double deposits) {
        this.deposits = deposits;
    }
}
```

// AccountsRepoitory.java

```
package com.example.transactional;
import org.springframework.data.repository.CrudRepository;
```

```java
import org.springframework.stereotype.Repository;
@Repository
public interface AccountRepository extends
CrudRepository<Account,Integer> {
}
```

// TransactionalApplication.java

package com.example.transactional;

```java
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import java.util.ArrayList;
import java.util.List;
@SpringBootApplication
public class TransactionalApplication {
    public static void main(String[] args) {
        ApplicationContext context =
SpringApplication.run(TransactionalApplication.class, args);
        AccountService accountService=
context.getBean(AccountService.class);
        List<Account> accountList = new ArrayList<>();
        accountList.add(new Account("robin",2000));
        accountList.add(new Account("batman",6000));
        accountService.createRecord(accountList);
        accountService.transferMoney(1000);
    }
}
```

//AccountService

package com.example.transactional;

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
public class AccountService {
    @Autowired
    AccountRepository repository;
    void createRecord(List<Account> accounts)
{repository.saveAll(accounts);
    }
```

```java
@Transactional(rollbackFor = Exception.class)
    public void transferMoney(double money)
    {
        int exception_raiser;
        Account account = repository.findById(1).get();
        account.setDeposits(account.getDeposits()-money);
        repository.save(account);
        exception_raiser=5/0;
        Account account2 = repository.findById(2).get();
        account2.setDeposits(account2.getDeposits()-money);
        repository.save(account2);
    }
}
```

Result Grid Filter Rows: Ed

#	id	accountHolder	deposits
1	56	robin	2000
2	57	batman	6000
*	NULL	NULL	NULL

public void transferMoney(double money) method withoud @Transactional Annotation

If I don't put @Transactional and somehow an exception occurs inbetween, then the transaction before the exception would be reflected and might be undiserable. For example , in case of money transaction, if an amount is deducted from a user account which requires to be added to others, but at the middle of the transaction an exception is raised and amount is not added.

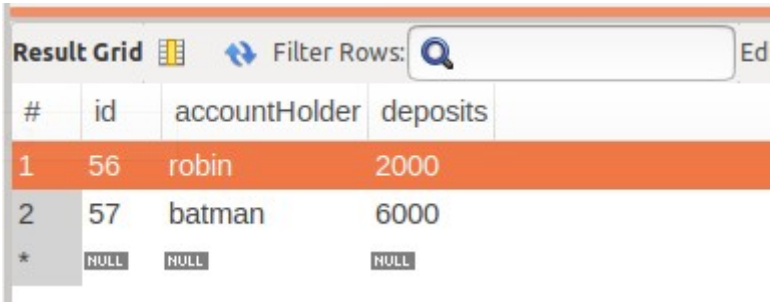
Result Grid Filter Rows:

#	id	accountHolder	deposits
1	56	robin	1000
2	57	batman	6000

```
Caused by: java.lang.ArithmeticException: / by zero
```

`public void transferMoney(double money) method with @Transactional Annotation`

If I put @Transactional and somehow an exception occurs inbetween, then the transaction before the exception would be Rolled Back.



Result Grid Filter Rows: Ed

#	id	accountHolder	deposits
1	56	robin	2000
2	57	batman	6000
*	NULL	NULL	NULL