Using the @Transactional Annotation

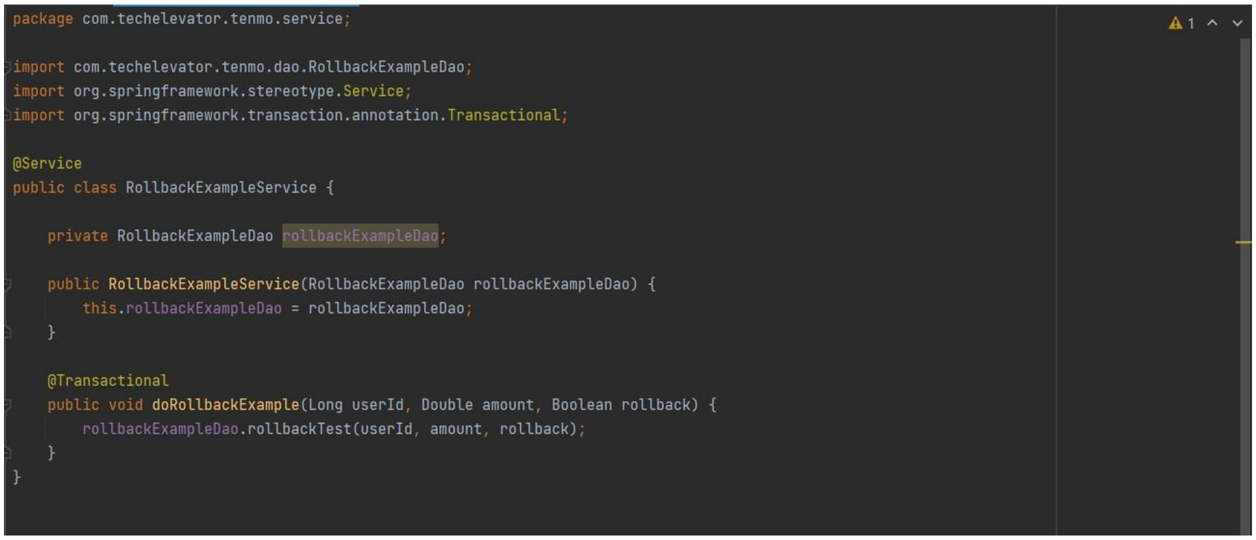
The Spring Framework provides an annotation that will mark a method as transactional.

The webapp has to be configured to use transactions, but if it is, it is every easy to make your code use transactions.

If transactions are configured in your Spring webapp, all you have to do is mark a method with the @Transactional annotation and it will rollback everything in that method if a RuntimeException of some sort is thrown. If your database code throws an exception, the @Transactional annotation will cause all database calls made in the annotated method to be rolled back.

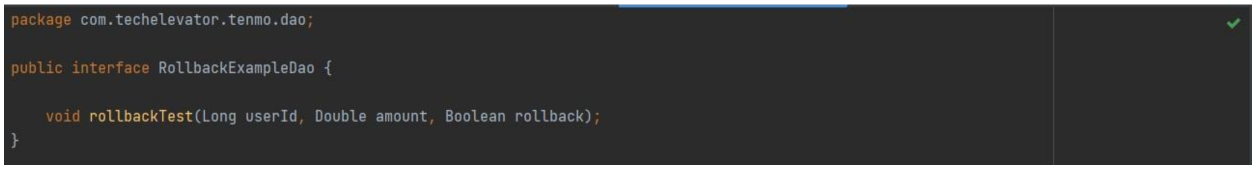
Let's look at an example…

Here is a sample service that has a method called doRollbackExample which is marked with the @Transactional annotation.



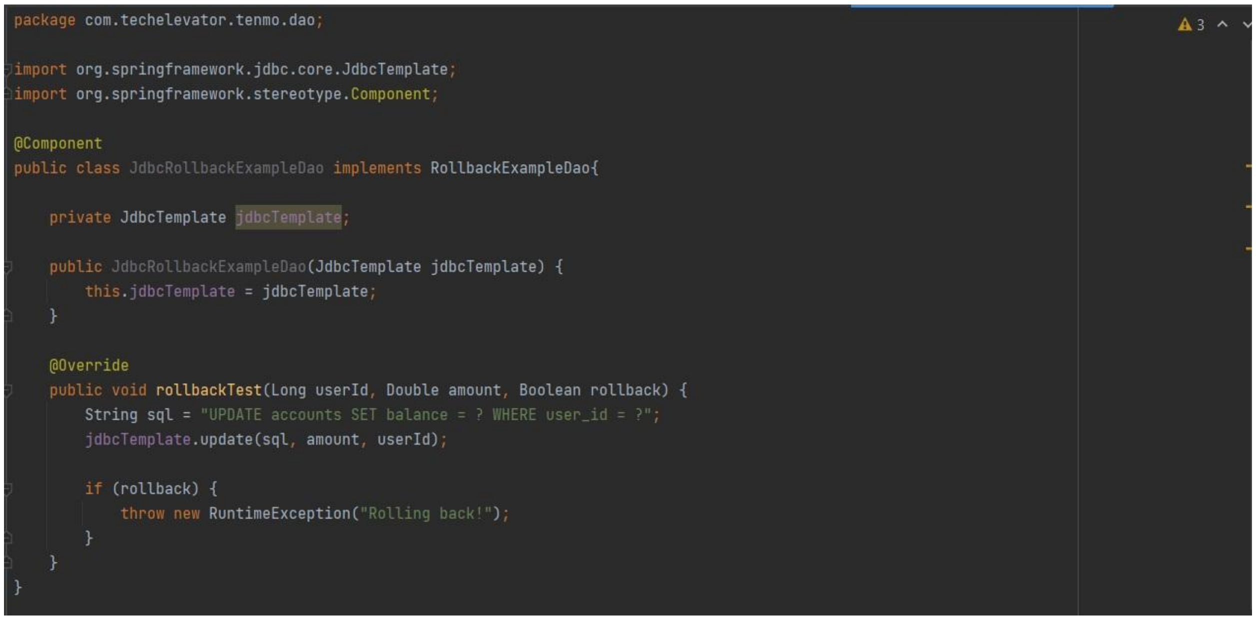
```java
package com.techelevator.tenmo.service;

import com.techelevator.tenmo.dao.RollbackExampleDao;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
public class RollbackExampleService {

    private RollbackExampleDao rollbackExampleDao;

    public RollbackExampleService(RollbackExampleDao rollbackExampleDao) {
        this.rollbackExampleDao = rollbackExampleDao;
    }

    @Transactional
    public void doRollbackExample(Long userId, Double amount, Boolean rollback) {
        rollbackExampleDao.rollbackTest(userId, amount, rollback);
    }
}
```

The service uses a RollbackExampeDao interface that looks like this:



```java
package com.techelevator.tenmo.dao;

public interface RollbackExampleDao {

    void rollbackTest(Long userId, Double amount, Boolean rollback);
}
```

Here's the JdbcRollbackExampleDao implementation class:



```java
package com.techelevator.tenmo.dao;

import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Component;

@Component
public class JdbcRollbackExampleDao implements RollbackExampleDao{

    private JdbcTemplate jdbcTemplate;

    public JdbcRollbackExampleDao(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    @Override
    public void rollbackTest(Long userId, Double amount, Boolean rollback) {
        String sql = "UPDATE accounts SET balance = ? WHERE user_id = ?";
        jdbcTemplate.update(sql, amount, userId);

        if (rollback) {
            throw new RuntimeException("Rolling back!");
        }
    }
}
```

This example takes a boolean variable to indicate whether a rollback should be simulated.

If the rollback boolean variable is set to true, the code will simulate a database exception by throwing a RuntimeException.

When this happens, if the method that calls this code (which in this case is the doRollbackExample method in the service) is annotated with the @Transactional tag then the database calls in that method will be rolled back.

If no RuntimeException occurs, the database calls in the method will be committed.

Note that **ONLY RuntimeExceptions cause a transaction to be rolled back** when using @Transacational - **checked Exceptions DO NOT cause a rollback** if they are thrown!

That's all there is to it!