## **BigDecimal -> Decimal128**

### **1. Maven Dependency**

Ensure Spring Data MongoDB is included in your pom.xml:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-mongodb</artifactId>

</dependency>

### **2. Custom Converters for BigDecimal ↔ Decimal128**

import org.bson.types.Decimal128;

import org.springframework.core.convert.converter.Converter;

import java.math.BigDecimal;

public class BigDecimalToDecimal128Converter implements Converter<BigDecimal, Decimal128> {

@Override

public Decimal128 convert(BigDecimal source) {

return new Decimal128(source);

}

}

public class Decimal128ToBigDecimalConverter implements Converter<Decimal128, BigDecimal> {

@Override

public BigDecimal convert(Decimal128 source) {

return source.bigDecimalValue();

}

}

### **3. Custom MongoTemplate Configuration**

Use the converters in a custom configuration class that defines a named MongoTemplate bean.

import com.mongodb.client.MongoClients;

import org.bson.types.Decimal128;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.core.convert.converter.Converter;

import org.springframework.data.mongodb.MongoDatabaseFactory;

import org.springframework.data.mongodb.core.MongoTemplate;

import org.springframework.data.mongodb.core.SimpleMongoClientDatabaseFactory;

import org.springframework.data.mongodb.core.convert.\*;

import org.springframework.data.mongodb.core.mapping.MongoMappingContext;

import java.math.BigDecimal;

import java.util.List;

@Configuration

public class CustomMongoConfig {

@Bean(name = "metadataMongoDBFactory")

public MongoDatabaseFactory metadataMongoDBFactory() {

return new SimpleMongoClientDatabaseFactory(

MongoClients.create("mongodb://localhost:27017"), // update with your URI

"your\_db\_name"

);

}

@Bean(name = "metaDataMongoTemplate")

public MongoTemplate metaDataMongoTemplate(MongoDatabaseFactory mongoDatabaseFactory) {

DbRefResolver dbRefResolver = new DefaultDbRefResolver(mongoDatabaseFactory);

MongoCustomConversions conversions = new MongoCustomConversions(List.of(

(Converter<BigDecimal, Decimal128>) Decimal128::new,

(Converter<Decimal128, BigDecimal>) Decimal128::bigDecimalValue

));

MongoMappingContext mappingContext = new MongoMappingContext();

mappingContext.setSimpleTypeHolder(conversions.getSimpleTypeHolder());

mappingContext.afterPropertiesSet();

MappingMongoConverter converter = new MappingMongoConverter(dbRefResolver, mappingContext);

converter.setCustomConversions(conversions);

converter.setCodecRegistryProvider(mongoDatabaseFactory);

converter.afterPropertiesSet();

converter.setTypeMapper(new DefaultMongoTypeMapper(null)); // disable \_class field

return new MongoTemplate(mongoDatabaseFactory, converter);

}

}

### **4. Model Class with BigDecimal**

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;

import java.math.BigDecimal;

@Document("products")

public class Product {

@Id

private String id;

private String name;

private BigDecimal price; // Will be stored as Decimal128 in MongoDB

// Getters and setters omitted for brevity

}

### **5. Saving and Retrieving Data**

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.math.BigDecimal;

@Service

public class ProductService {

@Autowired

private MongoTemplate metaDataMongoTemplate;

public void saveProduct() {

Product p = new Product();

p.setName("Banana");

p.setPrice(new BigDecimal("199.99")); // Stored as Decimal128

metaDataMongoTemplate.save(p);

}

public void readProduct(String id) {

Product p = metaDataMongoTemplate.findById(id, Product.class);

System.out.println("Product: " + p.getName() + ", Price: " + p.getPrice());

}

}

### **6. MongoDB Document Format**

When saved, the document will appear like this in MongoDB:

{

"\_id": "123...",

"name": "Banana",

"price": NumberDecimal("199.99")

}