STARMOUNT

Orika

●●●

simpler, lighter and faster Java bean mapping

Why?

- It is common to find that we need to convert objects into different formats to accommodate different APIs
- We may even need to convert formats between different architectural layers of our own for design reasons
- We are left to the rather boring task of writing mapping code to copy the values from one type to another.

Why?

STARMOUNT

```java
class BasicPerson {
  private String name;
  private int age;
  private Date birthDate;
  // getters/setters omitted
}
```

```java
class BasicPersonDto {
  private String fullName;
  private int currentAge;
  private Date birthDate;
  // getters/setters omitted
}
```

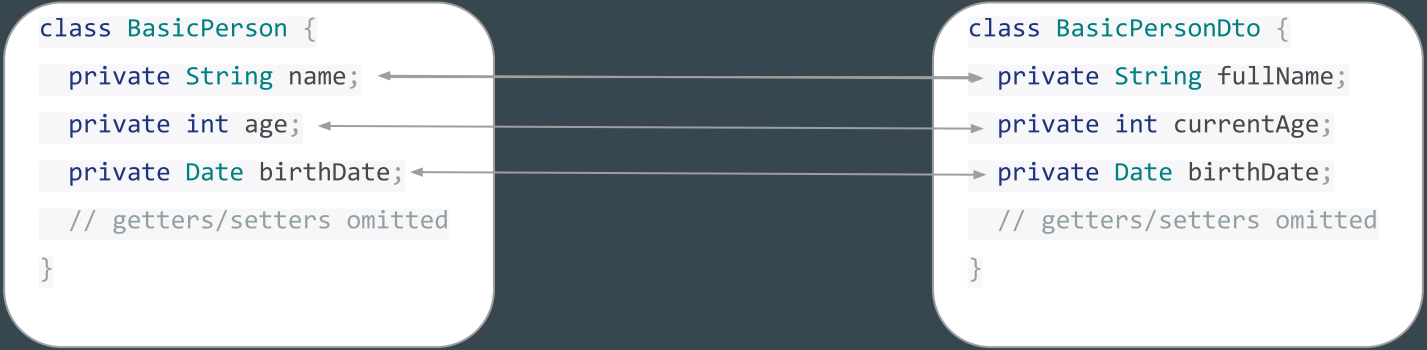
```java
public BasicPersonDto map (BasicPerson person) {
      BasicPersonDto dto = new BasicPersonDto();
      dto.setFullName(person.getName());
      dto.currentAge(person.getAge());
      dto.birthDate(person.getBirthDate());
}
```

How?

Orika attempts to perform this tedious work for you, with little measurable tradeoff on performance

It will automatically collect meta-data of your classes to generate mapping objects which can be used together to copy data from one object graph to another, recursively. Orika attempts to provide many convenient features while remaining relatively simple and open -- giving you the possibility to extend and adapt it to fit your needs.

How? - Declarative Mapping



```java
class BasicPerson {
  private String name;
  private int age;
  private Date birthDate;
  // getters/setters omitted
}
```

```java
class BasicPersonDto {
  private String fullName;
  private int currentAge;
  private Date birthDate;
  // getters/setters omitted
}
```

```java
MapperFactory mapperFactory = new DefaultMapperFactory.Builder().build();
mapperFactory.classMap(BasicPerson.class, BasicPersonDto.class)
      .fieldAToB("name", "fullName")
      .exclude("age")currentAge")
      .byDefault()
      .register()
```

How? - Advanced Mapping Configurations

STARMOUNT

Mapping null and nested values

```
mapperFactory.classMap(BasicPerson.class,
BasicPersonDto.class)
    .mapNulls(true).mapNullsInReverse(true)
    .field("field1", "fieldOne")
    .mapNulls(false).mapNullsInReverse(false)
    .field("field2.value", "fieldTwo")
    .byDefault()
    .register();
```

Ad-hoc/in-line property definitions

```
mapperFactory.classMap(BasicPerson.class, BasicPersonDto.class)
     .field("children.size:{size()|type=int}",
"numberOfChildren");

// «name» :{ «getter» | «setter» [ | type= «type» ] }
// «name» :{ «getter» [ | type= «type» ] }
// «name» :{| «setter» [ | type= «type» ] }
```

Customizing individual ClassMaps

```
mapperFactory.classMap(BasicPerson.class, BasicPersonDto.class)
      .byDefault()
      .customize(
       new CustomMapper<BasicPerson, BasicPersonDto> {
            public void mapAtoB(A a, B b, MappingContext context) {
                    // add your custom mapping code here    }
       }
    .register();
```

How? - Converters

Custom Converters

```
public class MyConverter extends CustomConverter<Date,MyDate> {
   public MyDate convert(Date source, Type<? extends MyDate> destinationType) {
      // return a new instance of destinationType with all properties filled
   }
}
```

Bi-Directional Converters

```
public class MyConverter extends BidirectionalConverter<Date,MyDate> {

   public MyDate convertTo(Date source, Type<MyDate> destinationType) {
      // convert in one direction
   }
   public Date convertFrom(MyDate source, Type<Date> destinationType) {
      // convert in the other direction
   }
}
```

How? - Converters II

Registering Globally or at field level

```
ConverterFactory converterFactory = mapperFactory.getConverterFactory();
converterFactory.registerConverter(new MyConverter());
//OR
converterFactory.registerConverter("myConverterIdValue", new MyConverter());
```

Applying field level converter

```
mapperFactory.classMap( Source.class, Destination.class )
    .fieldMap("sourceField1", "sourceField2").converter("myConverterIdValue").add()
    ...
    .register();
```

How? - Object Factories

STARMOUNT

Custom Object Factory

```java
public class PersonFactory implements ObjectFactory<Person> {

    public Person create(Object source, Type<Person> destinationType) {
        Person person = new Person();
        // set the default address
        person.setAddress(new Address("Morocco", "Casablanca"));
        return person;
    }
}
```

Registering an object factory

```java
mapperFactory.registerObjectFactory(new PersonFactory(), Person.class);
```

How? - Use it

```
MapperFactory mapperFactory = new DefaultMapperFactory.Builder().build();
MapperFacade mapper = mapperFactory.getMapperFacade();
BasicPerson source = new BasicPerson();
//set some fields
BasicPersonDto destination = mapper.map(source, BasicPersonDto.class);
// mapper.map(source, destination);
```

When?

- Converter
- Mapper
- ObjectFactory
- ByDefault

Where?

- TPP: Preserve -> Poslog (SaleLineItemMapping)
- Enact: Preserve/ConnectModel -> View Model (ItemMappingDefinition, ContactMappingDefinition)
- Inventory-Hub: Cassandra -> Business Model -> DTO (CassandraRowConverterDefinition, InventoryAPIMappingDefinition)
- Connect: Data Import -> Preserve (AttributeMappingDefinition)
- Connect: OMS -> Sterling/Manhattan Requests (PaymentMappingDefinition)
- Zumiez: Poslog -> Apropos (SaleMapping)
- Engage: Using their own converters

Questions?

References:

- http://orika-mapper.github.io/orika-docs/