Let’s talk about Bean Mapping

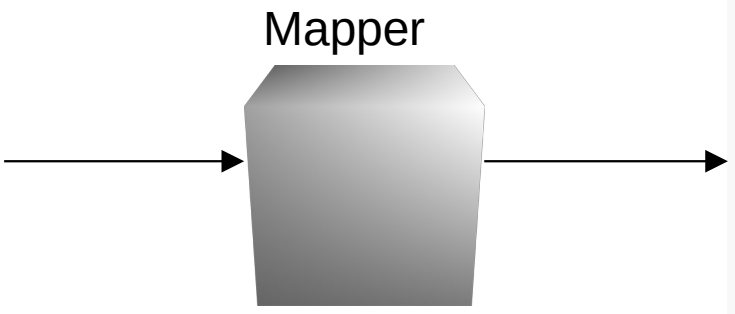
JHUG March 2021
Periklis Ntanasis – pntanasis@gmail.com

What is a bean/object mapper?

A mapper transfers data between objects

Source

```
class User {
  String username;
  String email;
  String password;
  String salt;
  LocalDate created;
  Boolean activated;
  Boolean locked;
  //... getters and setters
}
```



Target

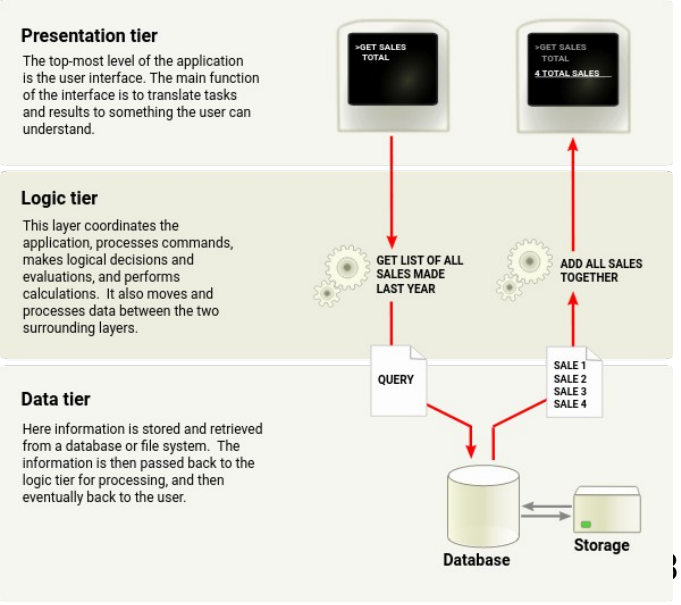
```
class UserDTO {
  enum Status {
    ACTIVATED,
    LOCKED,
    NEW
  }
  String username;
  String email;
  LocalDate creationTs;
  Status status;
  //... getters and setters
}
```

Why do we need mappers?

- Data Transfer Objects (DTO)

”An object that carries data between processes in order to reduce the number of method calls” – Martin Fowler

- Loose coupling of modules/layers (i.e. 3-tier architecture)

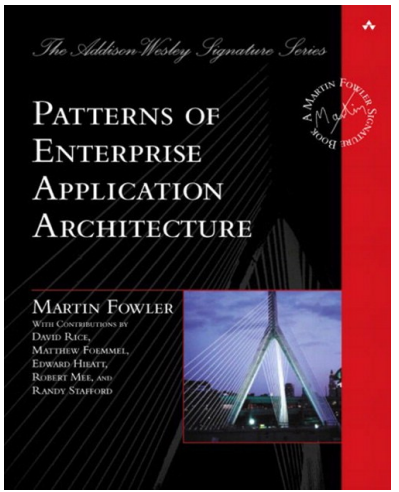


Sources: https://martinfowler.com/aboutMe.html
https://martinfowler.com/eaaCatalog/dataTransferObject.html
https://en.wikipedia.org/wiki/Multitier_architecture#Three-tier_architecture

Can we live without mappers?

- Sure! But there are limitations.

”Don't underestimate the cost of [using DTOs].... It's significant, and it's painful - perhaps second only to the cost and pain of object-relational mapping.” - Randy Stafford

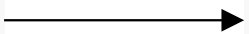


Sources:https://martinfowler.com/bliki/LocalDTO.html
https://martinfowler.com/books/eaa.html

How do mappers work?

- Manual mapping

```java
class User {
  String username;
  String email;
  String password;
  String salt;
  LocalDate created;
  Boolean activated;
  Boolean locked;
  //... getters and setters
}
```



```java
class Mapper {
  static UserDTO userToDto(User user) {
    UserDTO dto = new UserDTO();
    dto.setUsername(user.getUsername());
    dto.setEmail(user.getEmail());
    dto.setCreationTs(user.getCreated());
    if (user.getLocked) {
      dto.setStatus(Status.LOCKED);
    } else if (user.getActivated) {
      dto.setStatus(Status.ACTIVATED);
    } else {
      dto.setStatus(Status.NEW);
    }
    return dto;
  }
}
```

```java
class UserDTO {
  enum Status {
    ACTIVATED,
    LOCKED,
    NEW
  }
  String username;
  String email;
  LocalDate creationTs;
  Status status;
  //... getters and setters
}
```

How do mappers work?

- Manual mapping

  Pros:
  1. Simple
  2. Efficient

  Cons:
  1. Counter-productive
  2. High maintenance cost

How do mappers work?

- Reflection

```
public static <T> T map(Object src, Class<T> c) throws NoSuchMethodException,
    InstantiationException, IllegalAccessException, IllegalArgumentException,
    InvocationTargetException {
  if (src == null) {
    return null;
  }
  Constructor<T> defaultPublicConstructor = c.getConstructor();
  T targetObject = defaultPublicConstructor.newInstance();
  Field[] srcFields = src.getClass().getDeclaredFields();
  for (Field srcField : srcFields) {
    try {
      Field trgField = targetObject.getClass().getDeclaredField(srcField.getName());
      srcField.setAccessible(true);
      trgField.setAccessible(true);
      trgField.set(targetObject, srcField.get(src));
    } catch (NoSuchFieldException ex) {
      System.out.println(String
          .format("Field %s not found on target class", srcField.getName()));
    }
  }
  return targetObject;
}
```

How do mappers work?

- Reflection

Pros:
1. Can map private members
2. Can map objects loaded at runtime
3. Easy to use
4. Easy to implement

Cons:
1. Toooo slow
2. Reflection support may be unavailable
3. Risk of unexpected runtime behavior

How do mappers work?

- Code generation

1. Mapper annotation
2. Annotation processor
3. Dynamic retrieval of the implemented mapper
4. Client side Interface

How do mappers work?

- Code generation – Mapper annotation

```java
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.SOURCE)
public @interface Mapper {

}
```

How do mappers work?

- Code generation – Annotation processor

```java
@SupportedAnnotationTypes(
  "masterex.github.com.customstaticcodegenerationmapper.Mapper")
@SupportedSourceVersion(SourceVersion.RELEASE_11)
@AutoService(Processor.class)
public class MapperGenerator extends AbstractProcessor {

  @Override
  public boolean process(Set<? extends TypeElement> annotations, RoundEnvironment re) {
    List<Element> annotatedElements = annotations.stream()
          .flatMap(annotation -> re.getElementsAnnotatedWith(annotation).stream())
          .filter(x -> x.getKind() == ElementKind.INTERFACE)
          .collect(Collectors.toList());

    annotatedElements.stream()
          .forEach(this::createImplementation);

    return false;
  }
```

How do mappers work?

- Code generation – Annotation processor

```
private void createImplementation(Element element) {
  String implementationClassName = String.format("%s%s",
        element.getSimpleName().toString(), SUFFIX);
  String interfaceQualifiedName = ((TypeElement) element).getQualifiedName().toString();
  String packageName = interfaceQualifiedName.substring(0, interfaceQualifiedName
                                                          .lastIndexOf('.'));
  try {
    JavaFileObject mapperFile = processingEnv.getFiler().
        createSourceFile(packageName + "." + implementationClassName);
    try (PrintWriter out = new PrintWriter(mapperFile.openWriter())) {
      if (packageName != null) {
        out.print("package ");
        out.print(packageName);
        out.println(";");
        out.println();
      }

      out.print("public class ");
      out.print(implementationClassName);
      out.print(" implements " + interfaceQualifiedName);
      out.println(" {");
      out.println();

      element.getEnclosedElements().stream()
        .filter(e -> e.getKind() == ElementKind.METHOD)
```

How do mappers work?

- Code generation - Dynamic retrieval of the implemented mapper

```
public static <T> T getMappers(Class<T> c) {
  try {
    Constructor<T> constructor = (Constructor<T>) c.getClassLoader()
                                 .loadClass(c.getName() + SUFFIX).getDeclaredConstructor();
    constructor.setAccessible(true);
    return constructor.newInstance();
  } catch (ClassNotFoundException | NoSuchMethodException | SecurityException |
     InstantiationException | InvocationTargetException | IllegalAccessException ex) {
    throw new RuntimeException(ex);
  }
}
```

How do mappers work?

- Code generation

  Client side Interface

```java
@Mapper
public interface StaticCodeGenerationMapper {

  StaticCodeGenerationMapper INSTANCE = Mappers.getMappers(StaticCodeGenerationMapper.class);

  TargetBean map(SourceBean s);

}
```

  Client code

```java
StaticCodeGenerationMapper INSTANCE = Mappers.getMappers(StaticCodeGenerationMapper.class);
```

How do mappers work?

- Code generation

Pros:
1. Performance comparable to a hand-written mapper
2. No extra dependencies except the mapper loader
3. Easy to understand how the mapping is performed and debug it if required
4. Generated mappers may be copied into the project and eliminate any need of external dependencies

Cons:
1. Limitation in mapping objects added at runtime
2. Cannot map private members
3. A tiny bit more effort from the client side integration
4. The mapper implementation is more complex
5. Artifact size is affected by the number of generated mappers

How do mappers work?

- Byte-Code instrumentation

```java
public interface AbstractMapper<S, D> {

    public D map(S o);

}
```

```java
public class InstrumentationMapper<S, D> {

    private AbstractMapper<S, D> mapper;

    public InstrumentationMapper(final Class<S> source, final Class<D> destination) {
        try {
            this.mapper = getMapper(source, destination);
        } catch (InstantiationException | IllegalAccessException | NoSuchMethodException
                | IllegalArgumentException | InvocationTargetException ex) {
            throw new RuntimeException("Failed to initialize mapper", ex);
        }
    }

    public D map(S source) {
        return mapper.map(source);
    }
```

How do mappers work?

- Byte-Code instrumentation

```
private synchronized AbstractMapper<S, D> getMapper(final Class<S> source,
                                                    final Class<D> destination)
        throws InstantiationException, IllegalAccessException, NoSuchMethodException,
        IllegalArgumentException, InvocationTargetException {
    String mapperClassName = (source.getName() + destination.getName())
                                                        .replaceAll("\\.", "");
    try {
        return (AbstractMapper<S, D>) destination.getClassLoader()
                .loadClass(mapperClassName).getDeclaredConstructor().newInstance();
    } catch (ClassNotFoundException e) {
        return (AbstractMapper<S, D>) generateMapper(destination.getClassLoader(),
                mapperClassName, source, destination)
                                        .getDeclaredConstructor().newInstance();
    }
}
```

How do mappers work?

- Byte-Code instrumentation

```
private Class<?> generateMapper(ClassLoader classLoader, String className,
                                final Class<S> source, inal Class<D> destination) {
    try {
        ClassPool cp = ClassPool.getDefault();
        CtClass cc = cp.makeClass(className);
        cc.setInterfaces(new CtClass[]{cp.get(AbstractMapper.class.getName())});
        CtNewConstructor.defaultConstructor(cc);
        CtClass returnType = cp.get(Object.class.getName());
        CtClass[] arguments = new CtClass[]{cp.get(Object.class.getName())};
        CtMethod ctMethod = new CtMethod(returnType, "map", arguments, cc);
        ctMethod.setBody(mappingMethodBody(source, destination));
        cc.addMethod(ctMethod);
        cc.setModifiers(cc.getModifiers() & ~Modifier.ABSTRACT);
        Class<?> genereted Class = cc.toClass();
        return genereted Class;
    } catch (NotFoundException | CannotCompileException ex) {
        Logger.getLogger(InstrumentationMapper.class.getName()).log(Level.SEVERE,
                                                                    null, ex);
    }
    throw new RuntimeException("Failed to generate mapper.");
}
```

How do mappers work?

- Byte-Code instrumentation

```
private String mappingMethodBody(final Class<S> src, final Class<D> destination) {
    StringBuilder sb = new StringBuilder();
    sb.append("{").append(String.format("%s dst = new %s();", destination.getName(),
                                                              destination.getName()));
    sb.append(String.format("%s src = (%s) $1;", src.getName(), src.getName()));
    Method[] srcMethods = src.getDeclaredMethods();
    for (Method srcMethod : srcMethods) {
        if (!srcMethod.getName().startsWith("get")) {
            continue;
        }
        try {
            Method trgMethod = destination.getMethod(srcMethod.getName()
                                .replaceFirst("get", "set"), srcMethod.getReturnType());
            sb.append(String
               .format("dst.%s(src.%s());", trgMethod.getName(), srcMethod.getName()));
        } catch (NoSuchMethodException | SecurityException ex) {
            System.out.println(String
              .format("Method %s not found or accessible on source or target class",
                                                                              ex));
        }
    }
    sb.append("return dst;").append("}");
    return sb.toString();
```

How do mappers work?

- Byte-Code instrumentation

Pros:
1. Performance comparable to hand-written mappers (minus the initialization overhead)
2. Can handle objects loaded at runtime
3. Easy to use
4. Probably brings the best of the 2 previous approaches

Cons:
1. Cannot map private members (without modifying the mapped classes or using reflection)
2. Debugging is difficult
3. Implementation is complex
4. Initialization overhead
5. Reflection is a dependency

How mappers differ?

- Performance
- Ability to handle objects loaded at runtime
- Sane default configuration/conventions
- Expressiveness and format of the configuration API (Java, XML, Annotations)
- Ability to change/load mapping configuration at runtime
- Ability to handle advanced custom mapping configuration
- Documentation/Community/Maintenance

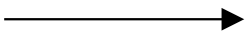
Mapping frameworks in Java

Name	Type	License	Configuration	GH Stars	First Release	Last Release
bean-cp	Bytecode Inst.	GPLv3	Java API	11	1.0 (2014)	1.0.2 (2017)
BeanUtils	Reflection	Apache 2.0	Java API	187	1.0 (2005)	1.9.4 (2019)
BULL	Reflection	Apache 2.0	Java API, Annotations	151	1.1.19 (2019)	1.7.6 (2021)
Datus	Bytecode Inst.	MIT	Java API	33	0.9.0 (2019)	1.5.0 (2020)
Dozer	Refection	Apache 2.0	XML, Annotations	1.8k	2.0.1 (2006)	6.5.0 (2019)
JMapper	Bytecode Inst.	Apache 2.0	Java API, XML, Annotations	155	1,1,0 (2012)	1..6.0.1 (2016)
MapStruct	C. Generation	Apache 2.0	Java API, Annotations	3.6k	1.0.0 (2013)	1.4.1 (2020)
ModelMapper	Bytecode Inst.	Apache 2.0	Java API	1.7k	0.3.1 (2011)	2.3.9 (2020)
MooMapper	Reflection	BSD	Java API, Annotations	25	2.0 (2014)	2.1.0 (2018)
Nomin	Bytecode Inst.	Apache 2.0	Groovy	40	1.0.0 (2010)	1.2 (2020)
Orika	Bytecode Inst.	Apache 2.0	Java API	1k	1.0 (2012)	1.5.4 (2019)
ReMap	Bytecode Inst.	Apache 2.0	Java API	74	0.0.3 (2017)	4.2.5 (2020)
Selma	C. Generation	Apache 2.0	Java API, Annotations	200	0.1 (2014)	1.0 (2017)

Which is the fastest mapper?

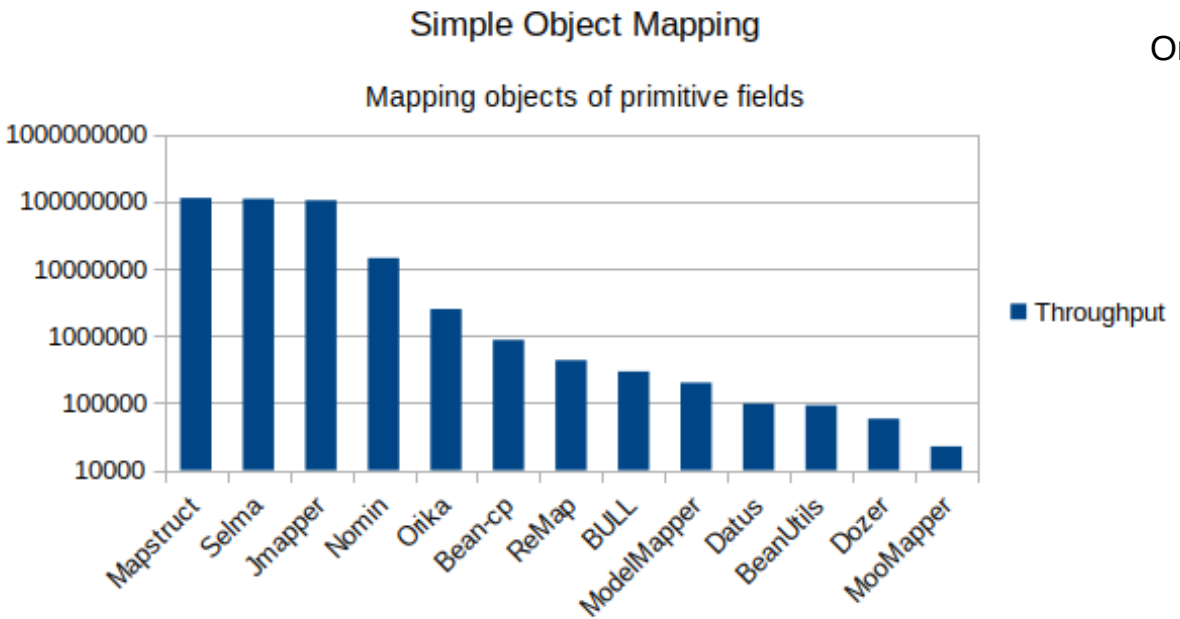
Benchmark #1

```java
public class SourceSimplePrimitiveObject {

    private int field1;
    private boolean field2;
    private String field3;
    private char field4;
    private double field5;
    // ...getters and setters
}
```



```java
public class TargetSimplePrimitiveObject {

    private int field1;
    private boolean field2;
    private String field3;
    private char field4;
    private double field5;
    // ...getters and setters
}
```

Benchmark #1



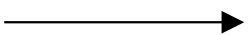
Order of magnitude

- 10^8 operations: MapStruct, Selma, JMapper
- 10^7 operations: Nomin
- 10^6 operations: Orika
- 10^5 operations: bean-cp, ReMap, BULL, ModelMapper
- 10^4 operations: Datus, BeanUtils, Dozer, MooMapper

Reflection
Bytecode instrumentation
Code generation

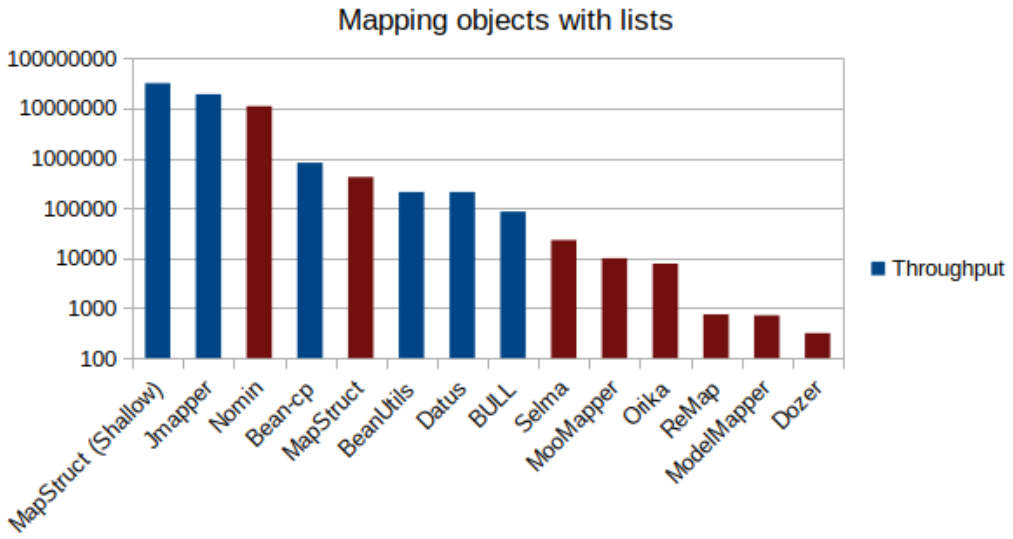
Benchmark #2

```java
public class SourceSimpleListObject {

    private List<Integer> list1 = new ArrayList<>();
    private List<Boolean> list2 = new ArrayList<>();
    private List<String> list3 = new ArrayList<>();
    private List<Character> list4 = new ArrayList<>();
    private List<Double> list5 = new ArrayList<>();
    // ...getters and setters
}
```



```java
public class TargetSimpleListObject {

    private List<Integer> list1 = new ArrayList<>();
    private List<Boolean> list2 = new ArrayList<>();
    private List<String> list3 = new ArrayList<>();
    private List<Character> list4 = new ArrayList<>();
    private List<Double> list5 = new ArrayList<>();
    // ...getters and setters
}
```

Benchmark #2



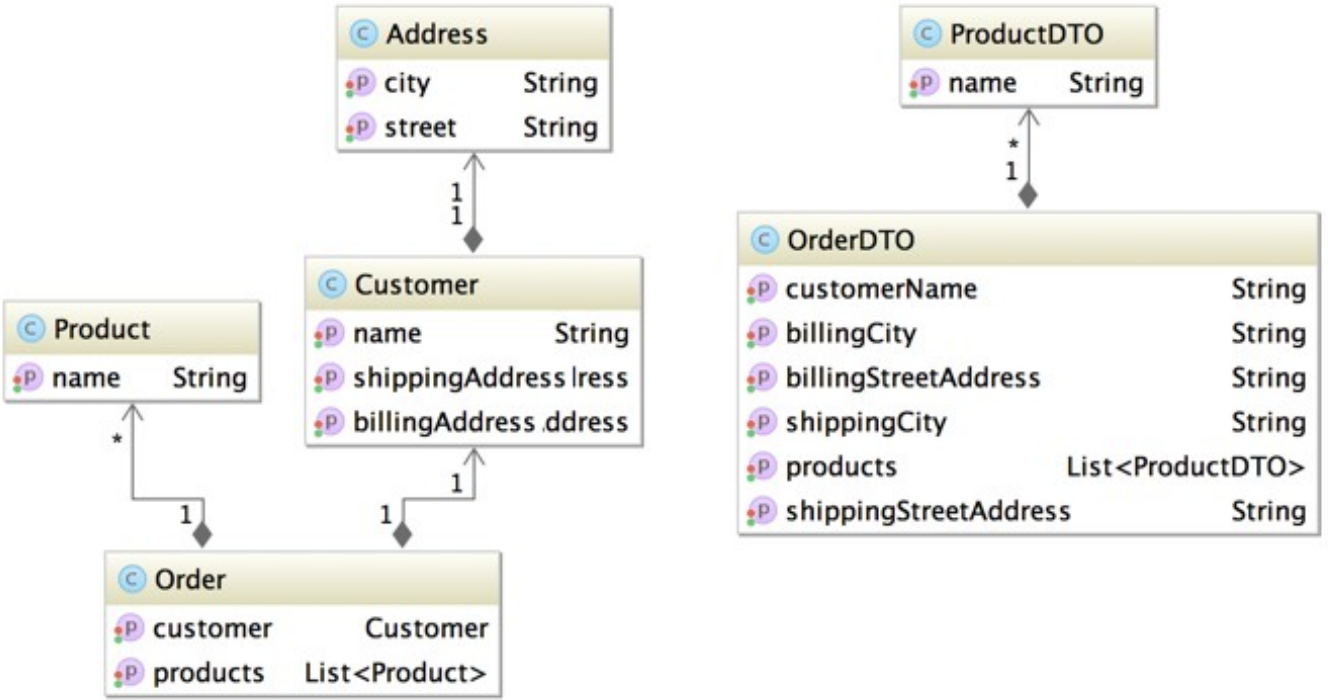
Order of magnitude

- 10^7 operations: MapStruct (Shallow), JMapper, Nomin
- 10^5 operations: bean-cp, MapStruct, BeanUtils, Datus
- 10^4 operations: BULL, Selma
- 10^3 operations: MooMapper, Orika
- 10^2 operations: ReMap, ModelMapper, Dozer

Creates new List on destination object.

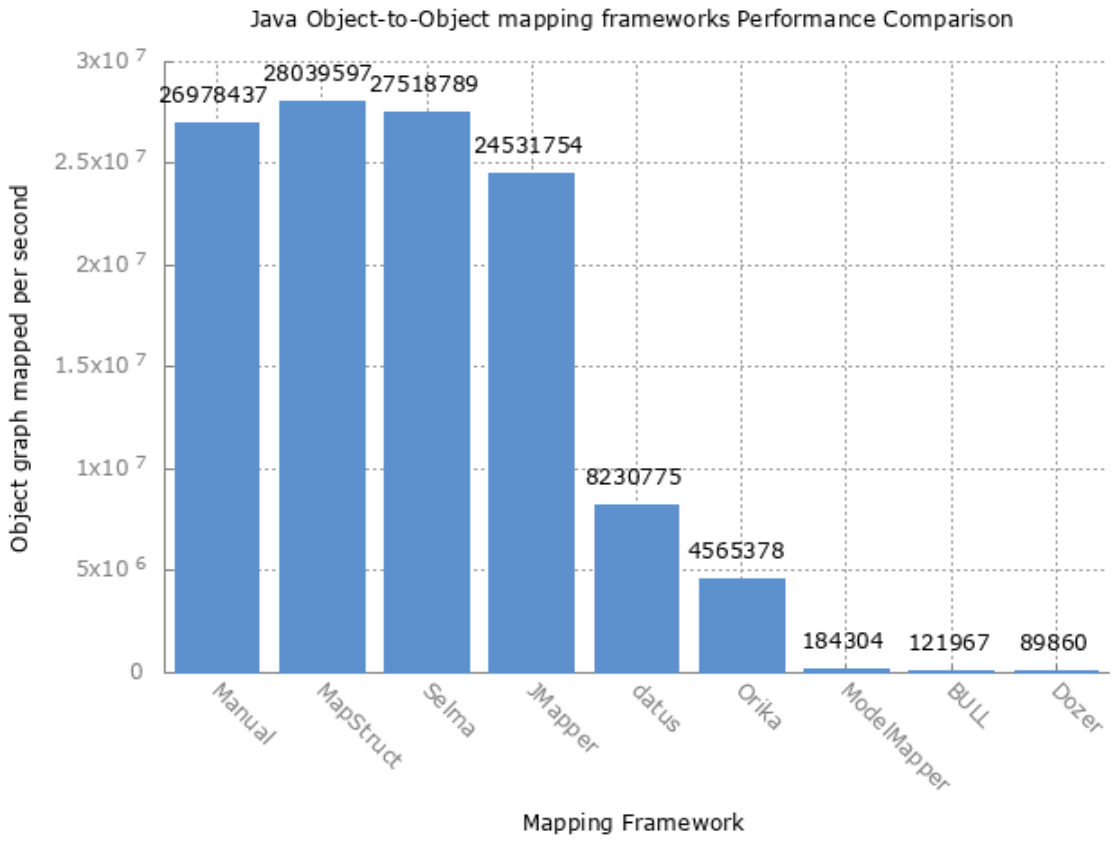
Reflection
Bytecode instrumentation
Code generation

Benchmark #3



Source: https://github.com/arey/java-object-mapper-benchmark

Benchmark #3



Order of magnitude

10^7 operations: MapStruct, Selma, Manual, JMapper

10^6 operations: Datus, Orika

10^5 operations: ModelMapper, BULL

10^4 operations: Dozer

Reflection
Bytecode instrumentation
Code generation

Source: https://github.com/arey/java-object-mapper-benchmark

Lessons learned

- Mapping is not free. Make sure that you need it
- Manual mapping is OK
- Do not re-invent the wheel
- Test that the mapping works as intended
- Avoid using slow mappers
- Consider using a combination of slow/fast mapper if a slow one is required due to “functionality” limitations of the faster ones
- Write your own benchmark if you have special performance requirements

Thank you :-)

- This presentation was based on this article:
  https://masterex.github.io/archive/2021/02/08/java-bean-mapping-in-depth.html
- Custom mappers source code:
  https://github.com/MasterEx/custom-bean-mapping
- Benchmark source code:
  https://github.com/MasterEx/bean-mapping-benchmark