Writing Annotation Processors to Aid Your Development Process

Ian Robertson
Chief Architect
Overstock.com
@nainostrebor

overstock.com®

Writing Proccesors Is...

- … easy to do, once you know how
- … using well documented APIs
- … not as well documented as a process

overstock.com®

Goals for this talk

- Give an overview of how to write a processor
- Cover a number of non-obvious aspects
- Non goal: be something that you can grok completely in one sitting
- Both slides and code are available!

overstock.com®

Agenda

- Review of annotations
- What is annotation processing?
- Uses of annotation processing
- How to write an annotation processor

overstock.com®

Review of Annotations

- Annotations attach metadata to code.

```
package javax.persistence;

import java.lang.annotation.*;

@Documented
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface Entity {
    String name() default "";
}
```

```
@Entity(name = ”customer”)
public class Customer { … }
```

Annotation element types

- Primitives
- Classes (can have bounds)
- Strings
- Enums
- Annotations
- Arrays of any of these

Annotation target types

- Types (classes and interfaces)
- Annotation types
- Fields
- Local Variables
- Parameters
- Methods
- Constructors
- Packages
- More to come in JDK8 (JSR 308)

overstock.com®

Runtime access

- For annotations with @RetentionType of Runtime

```
class<?> clazz = …
Entity entity = clazz.getAnnotation (Entity.class)
String name = entity.name();
```

overstock.com®

Annotation Processing

- Runs as part of compilation.
- Processors are discovered from the compile classpath.
- Processors may:
  - Create new resources
  - Create new source files
  - Issue notes, warnings and errors
    - Errors cause compilation to fail!
- Processors may not modify existing resources or classes

Uses of Processing

- Generated code (JPA typesafe queries)
- Generated resources
  - For example, could list annotated elements
- Adding compile time validations
  - IDEs will pick these up
  - Can add more “type safety” to your builds

overstock.com®

JpaProcessor

- Verifies that @Entity-annotated classes have a no-argument constructor
- For properties annotated with @OneToMany:
  - The child entity must have a corresponding property annotated with @ManyToOne
  - The @OneToMany annotation must have mappedBy pointing to the property on the child

JpaProcessor demo

Writing a processor

- Put name of processor class in
  META-INF/services/javax.annotation.processing.processor
- Turn off annotation processing when compiling the processor.
- Implement javax.annotation.processing.Processor
- Better: extend AbstractProcessor and annotate your processor with @SupportedAnnotationTypes and @SupportedSourceVersion

overstock.com®

Writing a processor

```
@SupportedAnnotationTypes(
    {"javax.persistence.Entity", "javax.persistence.OneToMany"})
@SupportedSourceVersion(SourceVersion.RELEASE_7)
public class JpaProcessor extends AbstractProcessor {
  @Override
  public void init(ProcessingEnvironment env) {
    super.init(env);
    ….
  }

  @Override
  public boolean process(
      Set<? extends TypeElement> annotations,
      RoundEnvironment roundEnv) {
            ….
            return false; // let others work on these annotations as well
  }
}
```

overstock.com®

ProcessingEnvironment

- Passed into the init method
  - AbstractProcessor hangs on to it for you
- getElementUtils(): Elements
- getTypeUtils(): Types
- getFiler(): Filer – access/create resources
- getMessager(): Messanger – warnings, errors
- getOptions(): Map<String, String> – set via -A

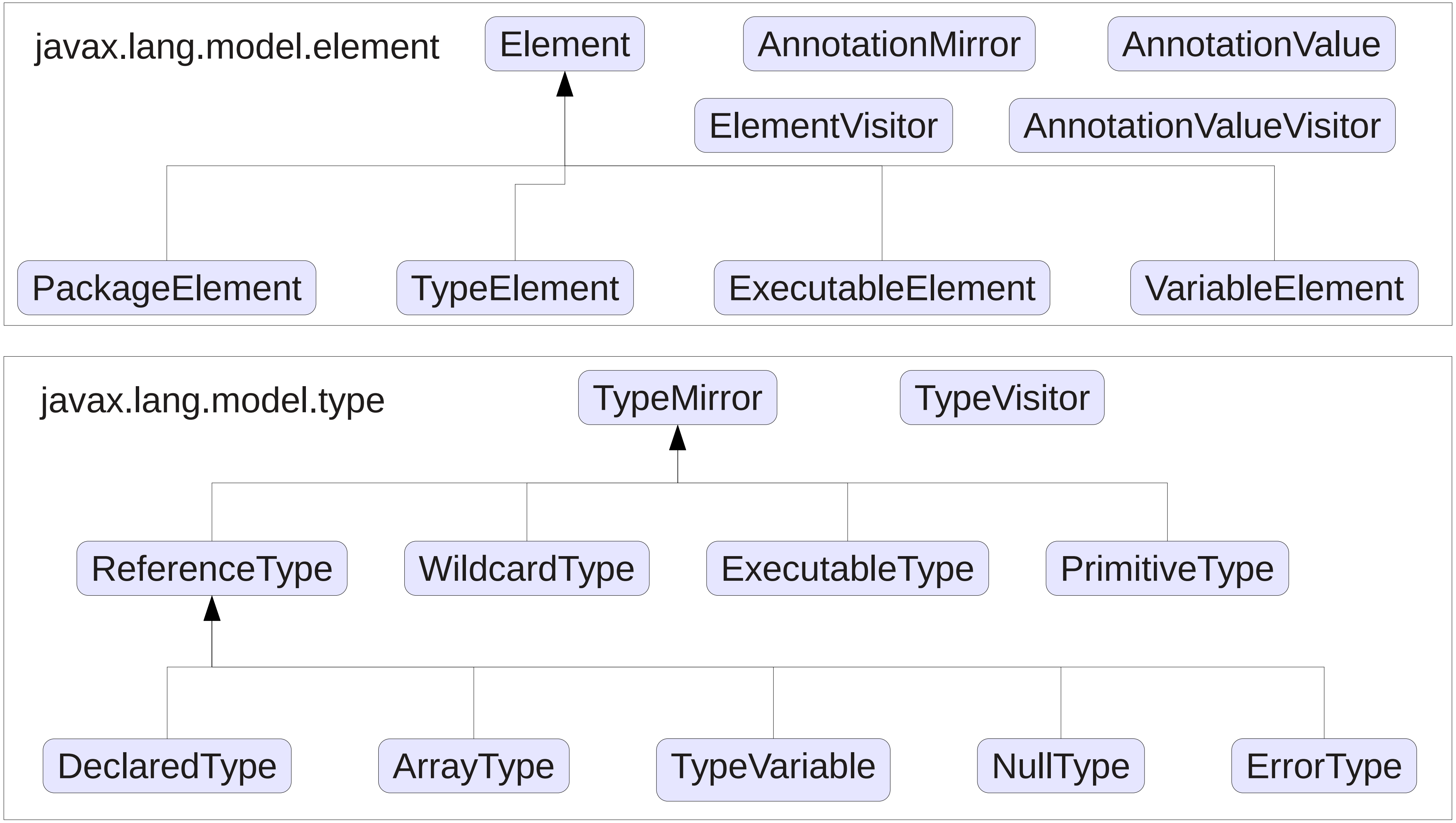
overstock.com®

Elements and Types

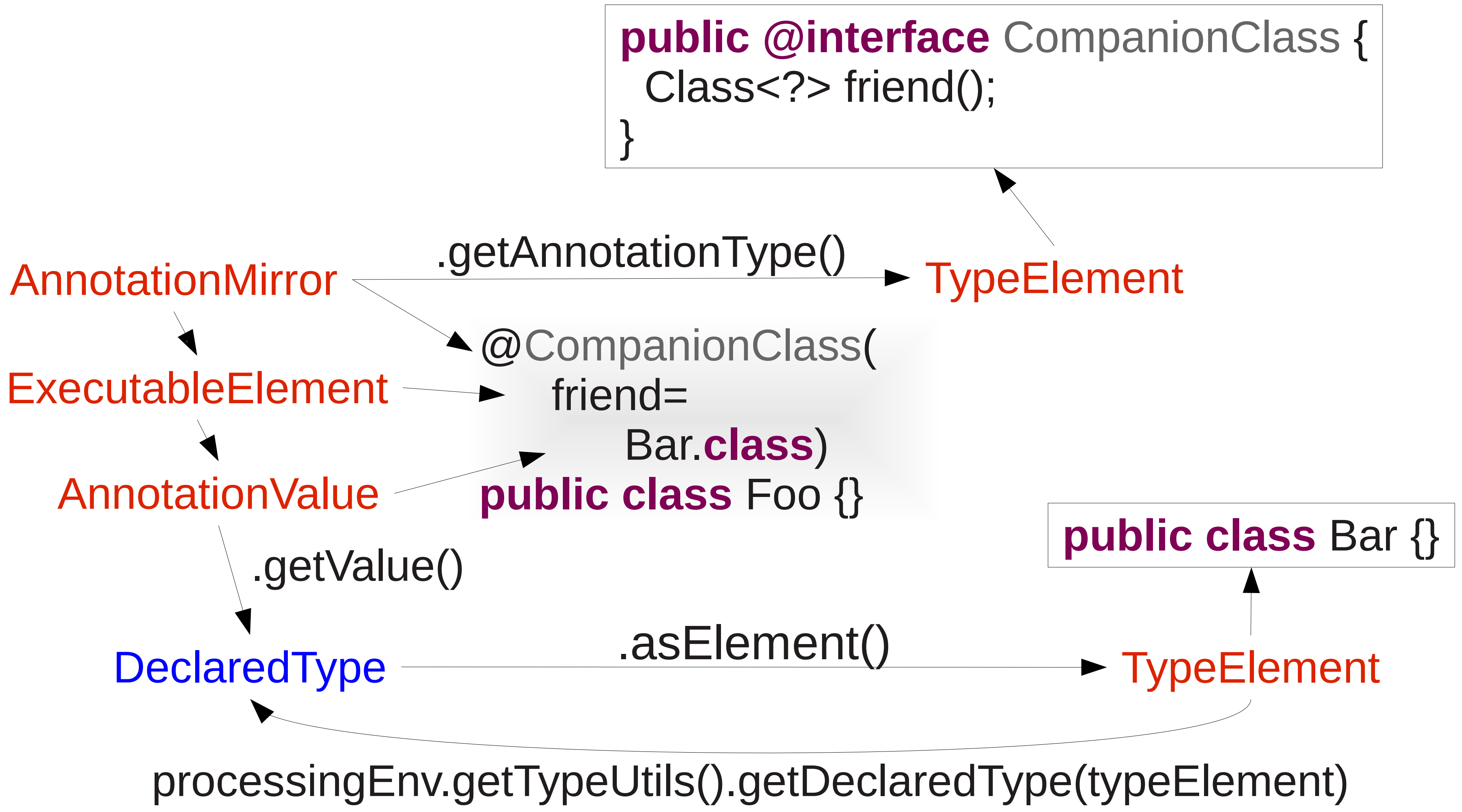
- Classes used for compilation are *not* available to load as classes!
- Instead, processors are given *mirrors*
- javax.lang.model.element:
  - interfaces for modeling elements such as classes, methods, variables, annotations, etc.
- javax.lang.model.type:
  - interfaces for modeling types (classes, interfaces, primitives, arrays, type variables, etc...)

overstock.com®

Elements and Types



Elements and Types



```
public @interface CompanionClass {
  Class<?> friend();
}
```

```
@CompanionClass(
   friend=
      Bar.class)
public class Foo {}
```

```
public class Bar {}
```

overstock.com®

Accessing Types

```
@Override
public boolean process(Set<? extends TypeElement> annotations,
    RoundEnvironment roundEnv) {

    TypeElement entityTypeElement = processingEnv.getElementUtils()
        .getTypeElement(“javax.persistence.Entity”);

    DeclaredType entityType = processingEnv.getTypeUtils()
        .getDeclaredType(entityTypeElement);

    Set<? extends Element> entityAnnotated =
        roundEnv.getElementsAnnotatedWith(entityTypeElement);

    for (TypeElement type: ElementFilter.typesIn(entityAnnotated)) {
        checkForNoArgumentConstructor(type);
    }

    return false;
}
```

overstock.com®

Looking at constructors

```
private void checkForNoArgumentConstructor(TypeElement type) {
  for (ExecutableElement constructor :
      ElementFilter.constructorsIn(type.getEnclosedElements())) {
    List<? extends VariableElement> constructorParameters =
      constructor.getParameters();
    if (constructor.getParameters().isEmpty()) {
      return;
    }
  }

  processingEnv.getMessager().printMessage(
    Kind.ERROR,
    "missing no argument constructor",
    type);
}
```

overstock.com®

Messager

```
void printMessage(Diagnostic.Kind kind,
                  CharSequence msg);

void printMessage(Diagnostic.Kind kind,
                  CharSequence msg,
                  Element e);

void printMessage(Diagnostic.Kind kind,
                  CharSequence msg,
                  Element e,
                  AnnotationMirror a);

void printMessage(Diagnostic.Kind kind,
                  CharSequence msg,
                  Element e,
                  AnnotationMirror a,
                  AnnotationValue v);
```

overstock.com®

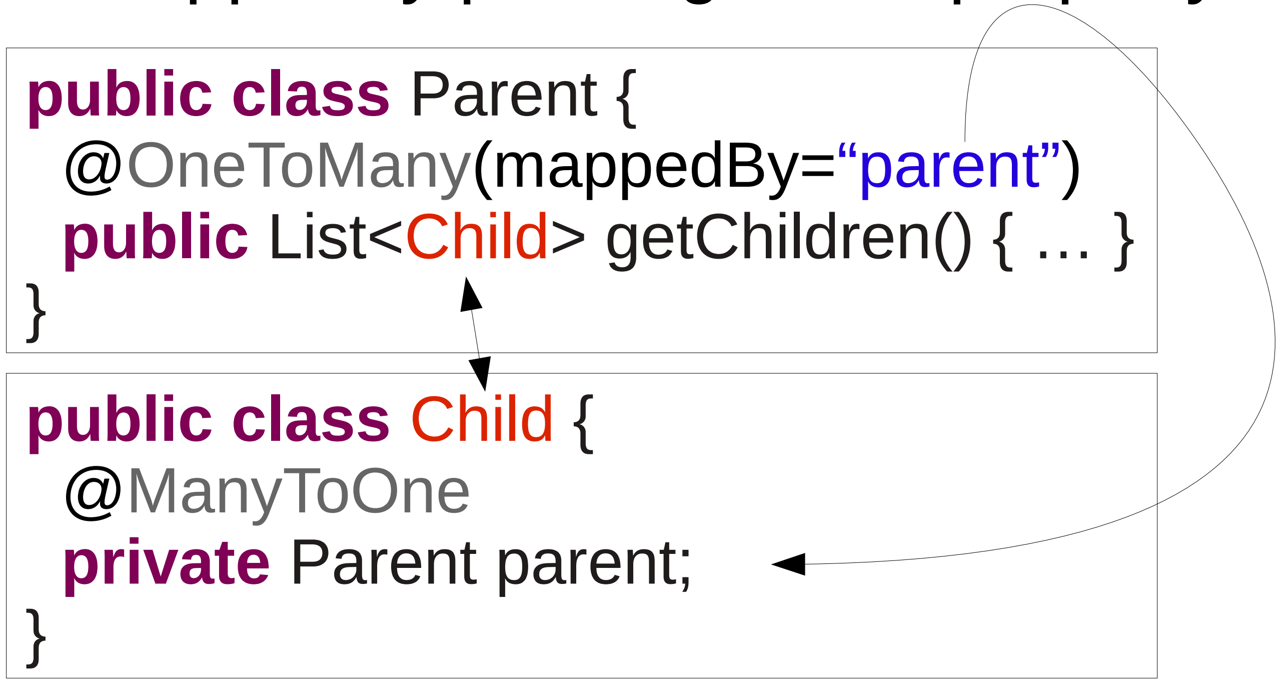
Breaking the build

```
private void checkForNoArgumentConstructor(TypeElement type) {
  for (ExecutableElement constructor :
      ElementFilter.constructorsIn(type.getEnclosedElements())) {
    List<? extends VariableElement> constructorParameters =
      constructor.getParameters();
    if (constructor.getParameters().isEmpty()) {
      return;
    }
  }

  processingEnv.getMessager().printMessage(
    Kind.ERROR,  // raises a compiler error
    "missing no argument constructor",
    typeElement);
}
```

overstock.com®

JPA Collection Mapping

- For properties annotated with @OneToMany:
  - The child entity must have a corresponding property annotated with @ManyToOne
  - The @OneToMany annotation must have mappedBy pointing to the property on the child



```
public class Parent {
  @OneToMany(mappedBy="parent")
  public List<Child> getChildren() { … }
}
```

```
public class Child {
  @ManyToOne
  private Parent parent;
}
```

overstock.com®

Find @OneToMany's

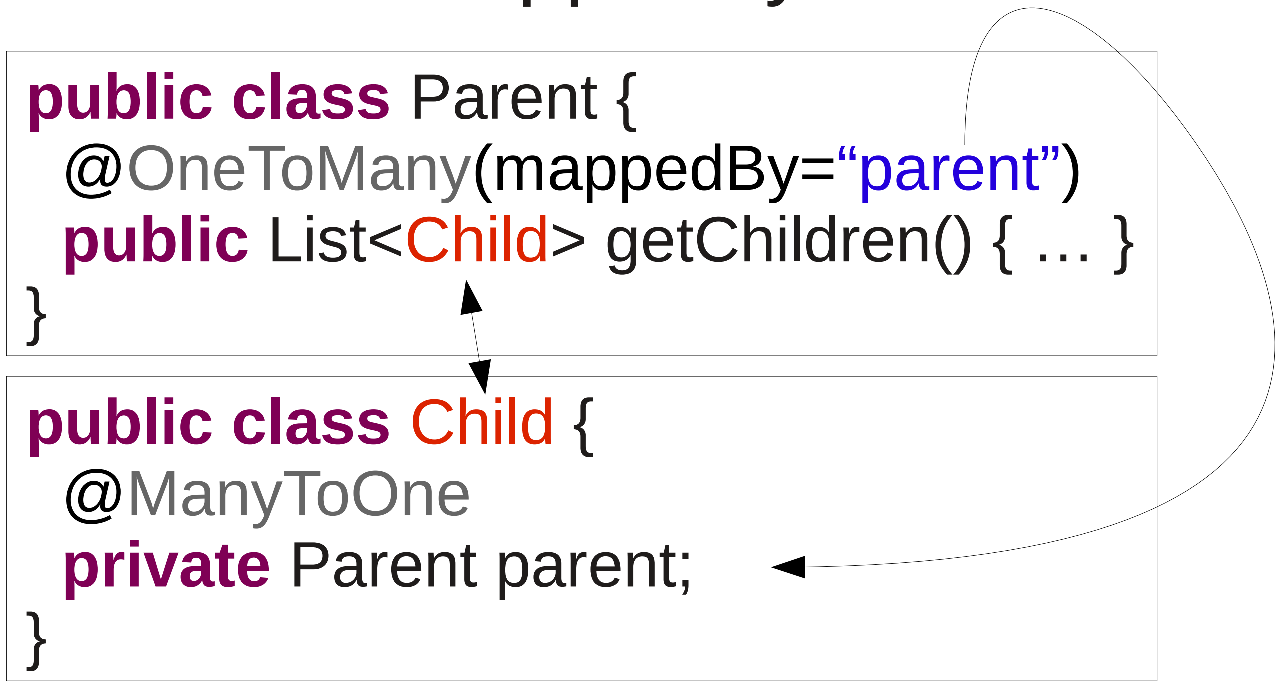
```
TypeElement oneToManyElement = processingEnv.getElementUtils()
    .getTypeElement(“javax.persistence.OneToMany”);

DeclaredType oneToManyType = processingEnv.getTypeUtils()
    .getDeclaredType(oneToManyElement);

Set<? extends Element> oneToManyAnnotated =
    roundEnv.getElementsAnnotatedWith(oneToManyTypeElement);
for (Element element : oneToManyAnnotated) {
    checkForBiDirectionalMapping(element);
}
```

Will be of type
ExecutableElement (method)
or VariableElement (field)

overstock.com®

checkForBiDirectionalMapping

- Get the child type from the collection type
- Find the element for the child type
- Find the parent property in the child type
- Check the mappedBy attribute on OneToMany



```java
public class Parent {
  @OneToMany(mappedBy="parent")
  public List<Child> getChildren() { … }
}
```

```java
public class Child {
  @ManyToOne
  private Parent parent;
}
```

Finding Property Type

```
private void checkForBiDirectionalMapping(Element childProperty) {
   TypeMirror propertyType = getPropertyType(childProperty);
```

```
public class Parent {
  @OneToMany(mappedBy="parent")
  public List<Child> getChildren() { … }
}
```

overstock.com®

Finding Property Type

```
private void checkForBiDirectionalMapping(Element childProperty) {
    TypeMirror propertyType = getPropertyType(childProperty);
```

```
private TypeMirror getPropertyType(Element element) {
    switch (element.getKind()) {
        case FIELD:
            return ((VariableElement) element).asType();
        case METHOD:
            return ((ExecutableElement) element).getReturnType();
        default:
            throw new IllegalArgumentException();
    }
}
```

Finding Property Type

```
private void checkForBiDirectionalMapping(Element childProperty) {
   TypeMirror propertyType = getPropertyType(childProperty);

   DeclaredType childType = getCollectionType(propertyType);
```

```
public class Parent {
  @OneToMany(mappedBy="parent")
  public List<Child> getChildren() { … }
}
```

overstock.com®

Type paremeters

```
private void checkForBiDirectionalMapping(Element childProperty) {
    TypeMirror propertyType = getPropertyType(childProperty);

    DeclaredType childType = getCollectionType(propertyType);
```

```
private DeclaredType getCollectionType(TypeMirror type) {

    DeclaredType collectionType = …; // java.util.Collection

    if(processingEnv.getTypeUtils().isAssignable(type, collectionType)) {

        List<? extends TypeMirror> typeArguments =
            ((DeclaredType) type).getTypeArguments();

        return (DeclaredType) typeArguments.get(0);
    }
    // else, raise an error
}
```

overstock.com®

Navigating to classes

```
private void checkForBiDirectionalMapping(Element childProperty) {
   TypeMirror propertyType = getPropertyType(childProperty);

   DeclaredType childType = getCollectionType(propertyType);
   Element childElement = childType.asElement();

   Element enclosingElement = childProperty.getEnclosingElement();

   DeclaredType parentType = processingEnv.getTypeUtils()
      .getDeclaredType((TypeElement) enclosingElement);
```

```
public class Parent {
  @OneToMany(mappedBy="parent")
  public List<Child> getChildren() { … }
}
```

```
public class Child {
  @ManyToOne
  private Parent parent;
}
```

overstock.com®

Clean up...

```
private void checkForBiDirectionalMapping(Element childProperty) {
    TypeMirror propertyType = getPropertyType(childProperty);

    DeclaredType childType = getCollectionType(propertyType);
    Element childElement = childType.asElement();

    Element enclosingElement = childProperty.getEnclosingElement();

    DeclaredType parentType = processingEnv.getTypeUtils()
        .getDeclaredType((TypeElement) enclosingElement);
```

overstock.com®

Clean up...

```
private void checkForBiDirectionalMapping(Element childProperty) {
    Element childElement = …;
    DeclaredType parentType = …;
```

```
public class Parent {
  @OneToMany(mappedBy="parent")
  public List<Child> getChildren() { … }
}
```

```
public class Child {
  @ManyToOne
  private Parent parent;
}
```

Finding fields & methods

```
private void checkForBiDirectionalMapping(Element childProperty) {
    Element childElement = …;
    DeclaredType parentType = …;
```

```
public class Parent {
  @OneToMany(mappedBy="parent")
  public List<Child> getChildren() { … }
}
```

```
public class Child {
  @ManyToOne
  private Parent parent;
}
```

overstock.com®

Finding Fields & Methods

```
private void checkForBiDirectionalMapping(Element childProperty) {
  Element childElement = …;
  DeclaredType parentType = …;
  Element parentPropertyInChild =
    findParentReferenceInChildType(parentType, childElement);
```

```
private Element findParentReferenceInChildType(
  TypeMirror parentType, Element childType) {
  for (Element element: childType.getEnclosedElements()) {
    if (element.getKind() == ElementKind.FIELD
        || element.getKind() == ElementKind.METHOD) {
      if (element.getAnnotation(ManyToOne.class) != null) {
        if (processingEnv.getTypeUtils().isSameType(
              parentType, getPropertyType(element))) }
          return element;
      }}}
  return null;
}
```

overstock.com®

Finding annotations

```
private void checkForBiDirectionalMapping(Element childProperty) {
    Element childElement = …;
    DeclaredType parentType = …;
    Element parentPropertyInChild =
        findParentReferenceInChildType(parentType, childElement);
    AnnotationMirror oneToManyAnnotation =
        getAnnotation(childProperty, oneToManyType);
```

```
private AnnotationMirror getAnnotation(Element element,
                                        DeclaredType annotationType) {
    for (AnnotationMirror mirror : element.getAnnotationMirrors()) {
      if (processingEnv.getTypeUtils()
            .isSameType(mirror.getAnnotationType(), annotationType)) {
        return mirror;
      }
      return null;
    }
  }
```

overstock.com®

So far...

```
private void checkForBiDirectionalMapping(Element childProperty) {
    Element childElement = …;
    DeclaredType parentType = …;
    Element parentPropertyInChild =
        findParentReferenceInChildType(parentType, childElement);
    AnnotationMirror oneToManyAnnotation =
        getAnnotation(childProperty, oneToManyType);
```

```
public class Parent {
  @OneToMany(mappedBy="parent")
  public List<Child> getChildren() { … }
}
```

```
public class Child {
  @ManyToOne
  private Parent parent;
}
```

overstock.com®

Clean up...

```
private void checkForBiDirectionalMapping(Element childProperty) {
  Element childElement = …;
  DeclaredType parentType = …;
  Element parentPropertyInChild =
    findParentReferenceInChildType(parentType, childElement);
  AnnotationMirror oneToManyAnnotation =
    getAnnotation(childProperty, oneToManyType);
```

Verify @ManyToOne

```
private void checkForBiDirectionalMapping(Element childProperty) {
    Element childElement = …;
    Element parentPropertyInChild = …;
    AnnotationMirror oneToManyAnnotation = …;
    if (parentPropertyInChild == null) {
      processingEnv.getMessager().printMessage(
        Kind.ERROR,
        "No matching @ManyToOne annotation on " +
          childElement.getSimpleName(),
        childProperty,
        oneToManyAnnotation);
    }
```

overstock.com®

Annotation Values

```
private void checkForBiDirectionalMapping(Element childProperty) {
    Element childElement = …;
    Element parentPropertyInChild = …;
    AnnotationMirror oneToManyAnnotation = …;
    if (parentPropertyInChild == null) { … }
    else {
      AnnotationValue mappedByValue =
        getMappedByValue(oneToManyAnnotation);
```

```
  private AnnotationValue getMappedByValue(
      AnnotationMirror oneToManyAnnotation) {
    Map<? extends ExecutableElement, ? extends AnnotationValue>
        elementValues = oneToManyAnnotation.getElementValues();
    return elementValues.get(mappedByAttribute);
  }
```

overstock.com®

Executable Elements

```
mappedByAttribute = getMethod(oneToManyTypeElement, "mappedBy");
```

```
private ExecutableElement getMethod(
  Element element, String methodName) {
  for (ExecutableElement executable:
      ElementFilter.methodsIn(element.getEnclosedElements())) {
    if (executable.getSimpleName().toString().equals(methodName)) {
      return executable;
    } }  throw new IllegalArgumentException(); }
```

```
private AnnotationValue getMappedByValue(
    AnnotationMirror oneToManyAnnotation) {
  Map<? extends ExecutableElement, ? extends AnnotationValue>
      elementValues = oneToManyAnnotation.getElementValues();
  return elementValues.get(mappedByAttribute);
}
```

Annotation Values

```
private void checkForBiDirectionalMapping(Element childProperty) {
    Element childElement = …;
    Element parentPropertyInChild = …;
    AnnotationMirror oneToManyAnnotation = …;
    if (parentPropertyInChild == null) { … }
    else {
      AnnotationValue mappedByValue =
        getMappedByValue(oneToManyAnnotation);
```

overstock.com®

Verify mappedBy

```
private void checkForBiDirectionalMapping(Element childProperty) {
    Element childElement = …;
    Element parentPropertyInChild = …;
    AnnotationMirror oneToManyAnnotation = …;
    if (parentPropertyInChild == null) { … }
    else {
      AnnotationValue mappedByValue = …;
      if (mappedByValue == null) {
        processingEnv.getMessager().printMessage(
          Kind.ERROR,
          "Missing mappedBy attribute",
          childProperty,
          oneToManyAnnotation);
      }
```

overstock.com®

Property names

```
private void checkForBiDirectionalMapping(Element childProperty) {
    Element childElement = …;
    Element parentPropertyInChild = …;
    AnnotationMirror oneToManyAnnotation = …;
    if (parentPropertyInChild == null) { … }
    else {
      AnnotationValue mappedByValue = …;
      if (mappedByValue == null) { … }
      else {
        String mappedBy = (String) mappedBy.getValue();
        String expected  = getPropertyName(parentPropertyInChild);
```

```
private String getPropertyName(Element propertyElement) {
  switch (propertyElement.getKind()) {
    case FIELD: return propertyElement.getSimpleName().toString();
    case METHOD: ... // remove leading get, decapitalize
}
```

overstock.com®

Precise error placement

```
private void checkForBiDirectionalMapping(Element childProperty) {
    Element childElement = …;
    Element parentPropertyInChild = …;
    AnnotationMirror oneToManyAnnotation = …;
    if (parentPropertyInChild == null) { … }
    else {
      AnnotationValue mappedByValue = ...;
      if (mappedByValue == null) { … }
      else {
        String mappedBy = (String) mappedBy.getValue();
        String expected  = getPropertyName(parentPropertyInChild);
        if (! mappedBy.equals(expected) {
          processingEnv.getMessager().printMessage(
            Kind.ERROR,
            "mappedBy attribute should be " + expected,
            childProperty,
            oneToManyAnnotation,
            mappedBy);
        }
```

overstock.com®

Done!!!

```
private void checkForBiDirectionalMapping(Element childProperty) {
    Element childElement = …;
    Element parentPropertyInChild = …;
    AnnotationMirror oneToManyAnnotation = …;
    if (parentPropertyInChild == null) { … }
    else {
      AnnotationValue mappedByValue = ...;
      if (mappedByValue == null) { … }
      else {
        String mappedBy = (String) mappedBy.getValue();
        String expected  = getPropertyName(parentPropertyInChild);
        if (! mappedBy.equals(expected) { … }
      }
    }
}
```

overstock.com®

Testing

- Use javax.tools.JavaCompiler
- Not only allows automated tests, but also debugging of annotation processors
- To verify messages, wrap your processor in one which wraps ProcessingEnvironment
  - Use the wrapped processingEnv to mock out Messager

Handling all annotations

- @SupportedAnnotationTypes("*")
- Can be useful even for non-annotated code
- Start with roundEnv.getRootElements()
- Recurse from there by getEnclosedElements()
  - Except for packages
- Also useful if the compiler fails to see inherited annotations...

overstock.com®

Advice

- For annotations with non-trivial default values, use Elements#getElementValuesWithDefaults
- To see inherited annotations, use Elements#getAllAnnotationMirrors
- Classpath includes the jar the processor is in, but not necessarily other jars
- When dealing with inner classes, pay attention to binary versus qualified names (A.B vs A$B)
- Define DeclaredTypes and the like in init()

Sample code

https://github.com/irobertson/jpa-annotation-processor

overstock.com®

Questions

?