Multi-device Content Display & Smart Use of Annotation Processing

@gdigugli
@dbaeli

Speakers

@dbaeli - Dimitri BAELI

- Java developer since 1999
- R&D Team Mentor at

LesFurets.com
Comparez futé

- Coder, DevOps, Agile Coach
  - From idea to production
- eXo Platform
  - VP Quality

@gdigugli - Gilles Di Guglielmo

- Java developer since 1999
- Software architect at

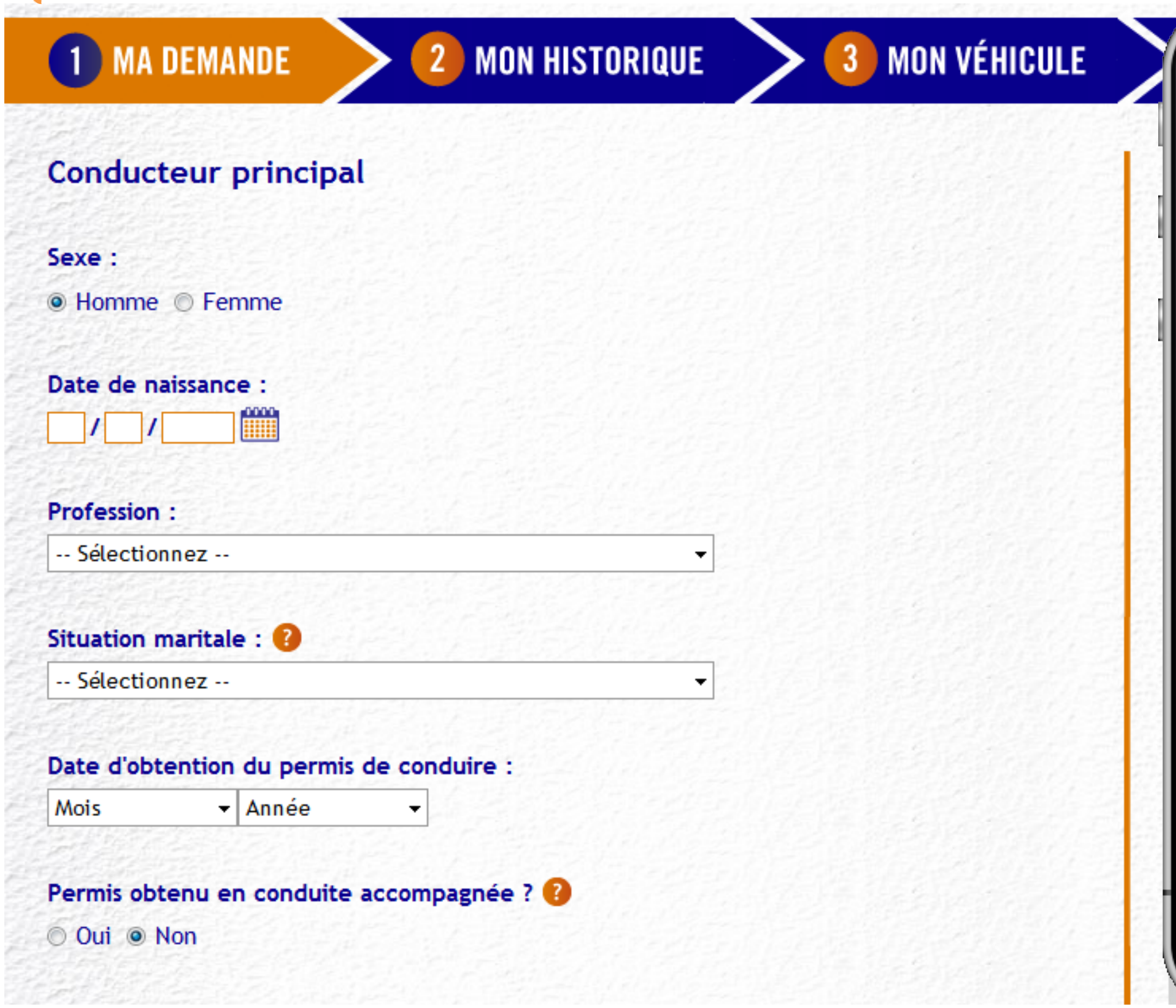
LesFurets.com
Comparez futé

- ILOG - IBM
  - ✓ 2D graphic toolkit
  - ✓ Rule engine
- Prima-Solutions
  - ✓ Services platform for J2EE
  - ✓ Domain models code generators

We're working for
A French Insurance Aggregator

LesFurets.com
Comparez futé

LesFurets.com mobile & desktop



1 MA DEMANDE
2 MON HISTORIQUE
3 MON VÉHICULE

Conducteur principal

Sexe :
[✓ Homme] Femme

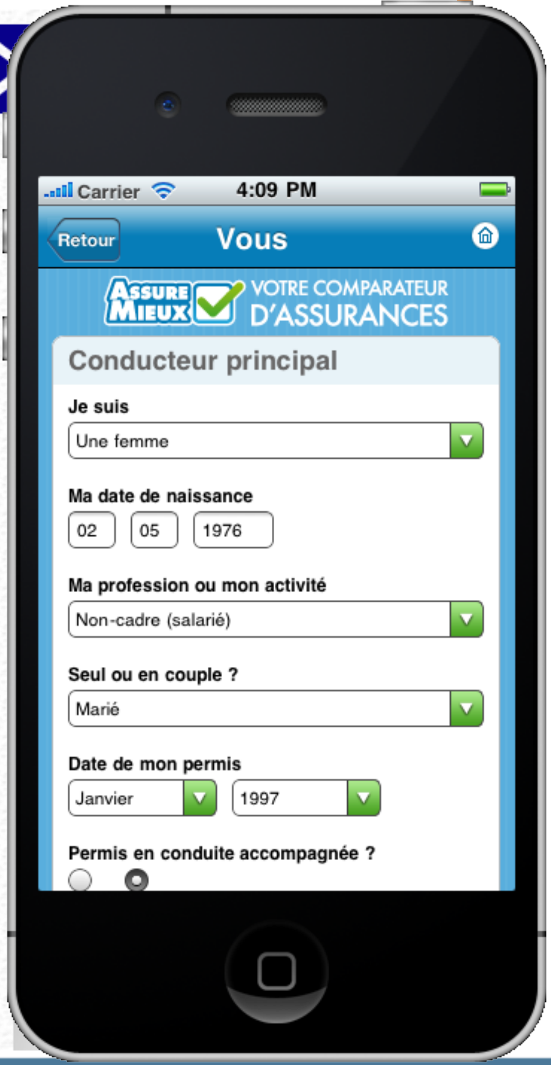
Date de naissance :
/ /

Profession :
-- Sélectionnez --

Situation maritale : ?
-- Sélectionnez --

Date d'obtention du permis de conduire :
Mois | Année

Permis obtenu en conduite accompagnée ? ?
Oui [✓ Non]



Carrier 4:09 PM

Retour — Vous

ASSURE MIEUX — VOTRE COMPARATEUR D'ASSURANCES

Conducteur principal

Je suis
Une femme

Ma date de naissance
02 05 1976

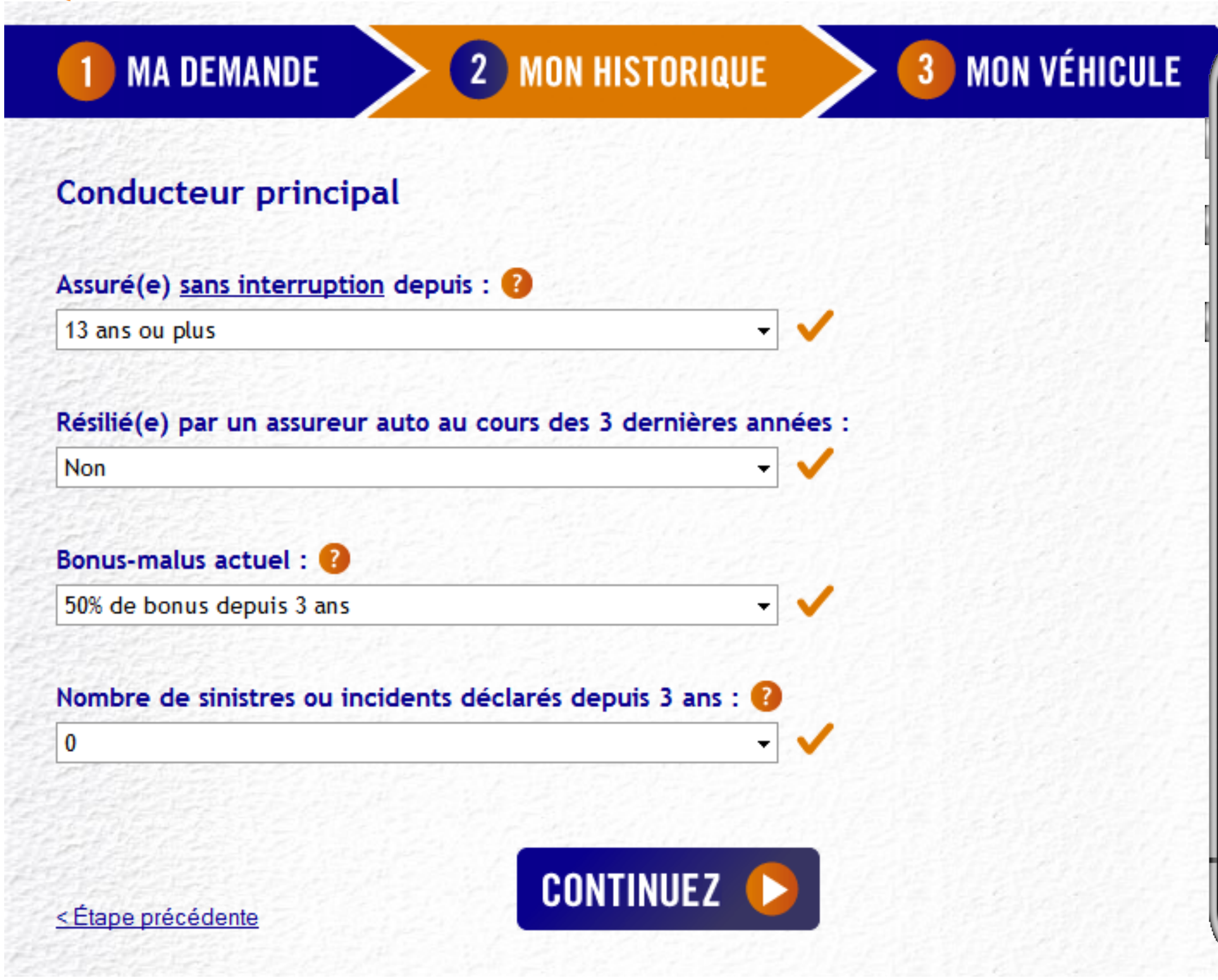
Ma profession ou mon activité
Non-cadre (salarié)

Seul ou en couple ?
Marié

Date de mon permis
Janvier 1997

Permis en conduite accompagnée ?

LesFurets.com mobile & desktop



1 MA DEMANDE | 2 MON HISTORIQUE | 3 MON VÉHICULE

Conducteur principal

Assuré(e) sans interruption depuis : ?

13 ans ou plus ✓

Résilié(e) par un assureur auto au cours des 3 dernières années :

Non ✓

Bonus-malus actuel : ?

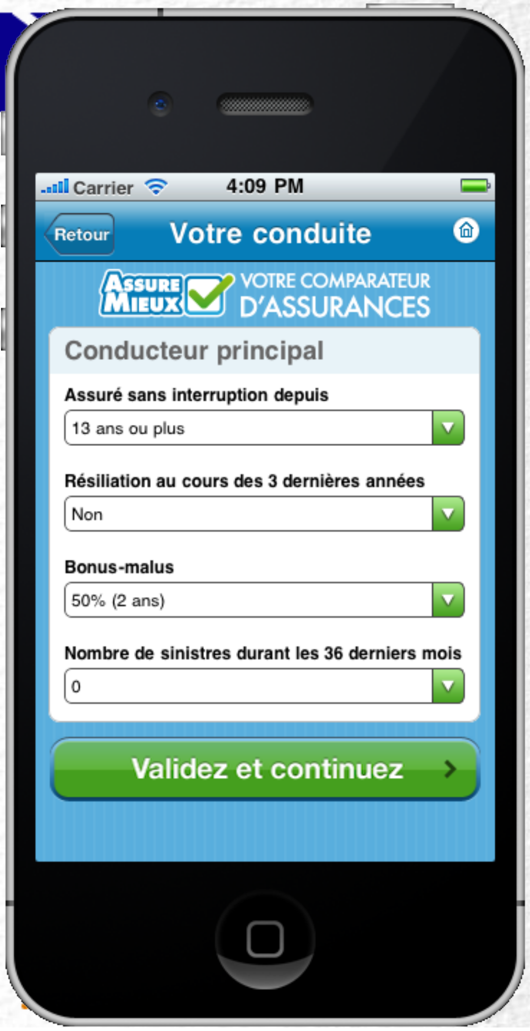
50% de bonus depuis 3 ans ✓

Nombre de sinistres ou incidents déclarés depuis 3 ans : ?

0 ✓

< Étape précédente

CONTINUEZ



Carrier 4:09 PM

Retour | Votre conduite

ASSURE MIEUX VOTRE COMPARATEUR D'ASSURANCES

Conducteur principal

Assuré sans interruption depuis

13 ans ou plus

Résiliation au cours des 3 dernières années

Non

Bonus-malus

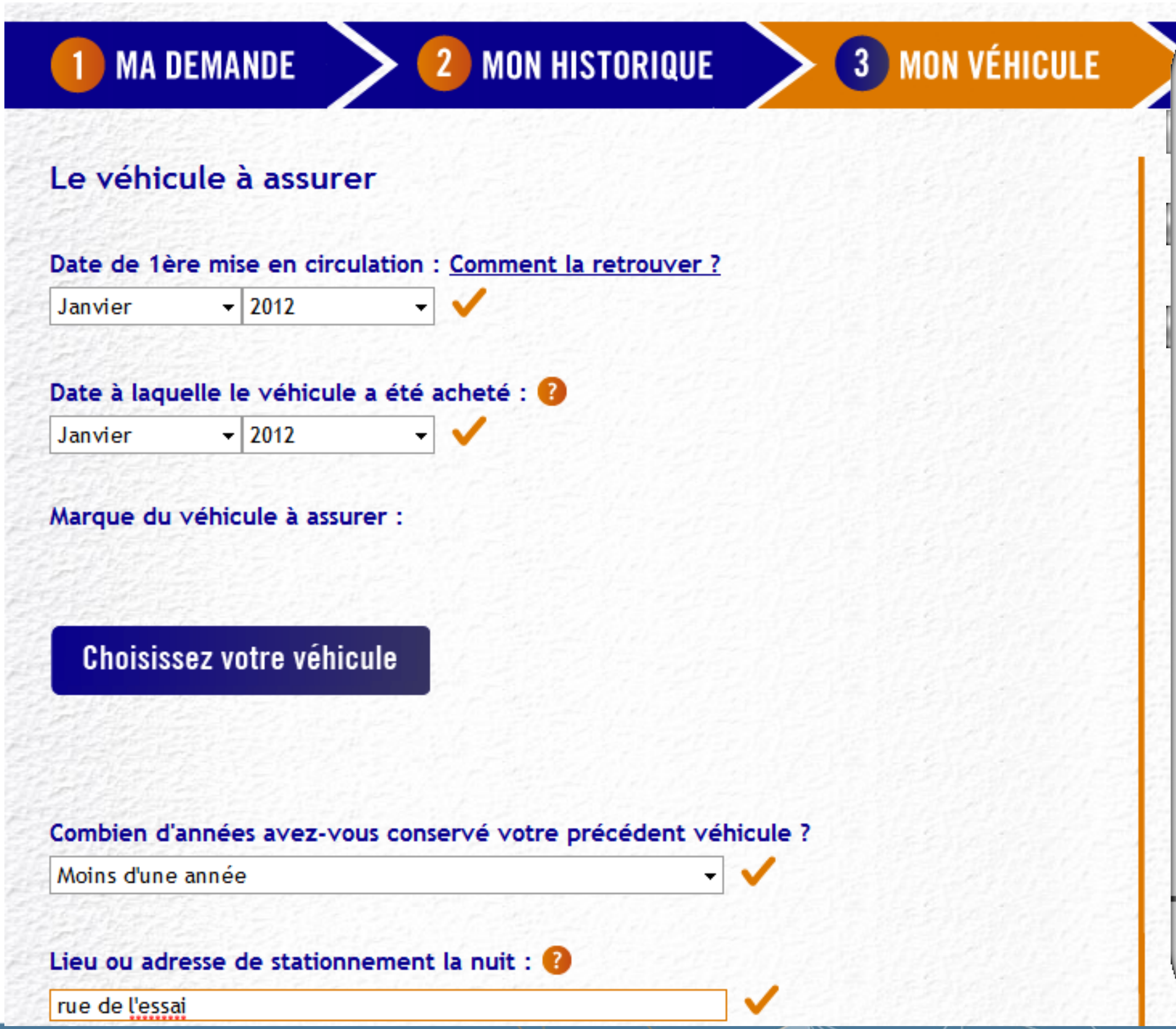
50% (2 ans)

Nombre de sinistres durant les 36 derniers mois

0

Validez et continuez

LesFurets.com mobile & desktop



1 MA DEMANDE | 2 MON HISTORIQUE | 3 MON VÉHICULE

Le véhicule à assurer

Date de 1ère mise en circulation : Comment la retrouver ?

Janvier | 2012 ✓

Date à laquelle le véhicule a été acheté : ?

Janvier | 2012 ✓

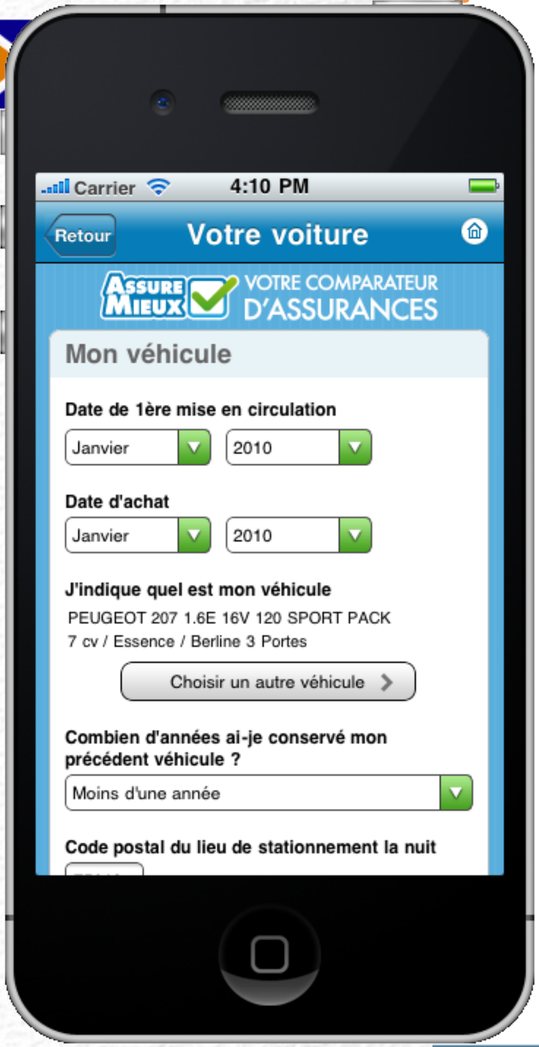
Marque du véhicule à assurer :

Combien d'années avez-vous conservé votre précédent véhicule ?

Moins d'une année ✓

Lieu ou adresse de stationnement la nuit : ?

rue de l'essai ✓



Carrier 4:10 PM
Retour
Votre voiture
ASSURE MIEUX VOTRE COMPARATEUR D'ASSURANCES

Mon véhicule

Date de 1ère mise en circulation

Janvier | 2010

Date d'achat

Janvier | 2010

J'indique quel est mon véhicule

PEUGEOT 207 1.6E 16V 120 SPORT PACK
7 cv / Essence / Berline 3 Portes

Choisir un autre véhicule

Combien d'années ai-je conservé mon précédent véhicule ?

Moins d'une année

Code postal du lieu de stationnement la nuit

In the search of
Effective content display

The story

Effective Content Display

- Multi device & languages
  - Labels
  - Layout & images
- Clean code
  - Strong Quality
  - Easy Maintenance

using APT Tooling

- APT Engine
- APT Processors
  - Generate technical code
  - Generate reports
  - Generate patterns

based on i18n

- @Message
- @MessageBundle
- Dedicated APT Processors

https://github.com/dbaeli/ez18n

Improved i18n for text display

Java i18n pattern

- The JDK default tooling to:
  - Dynamically bind the content
  - Usable for Texts, but also CSS and images (urls)
- Tooling :
  - java.util.ResourceBundle : for .properties reading
  - java.util.MessageFormat : tiny templating
  - .properties files with naming pattern

java.util.ResourceBundle

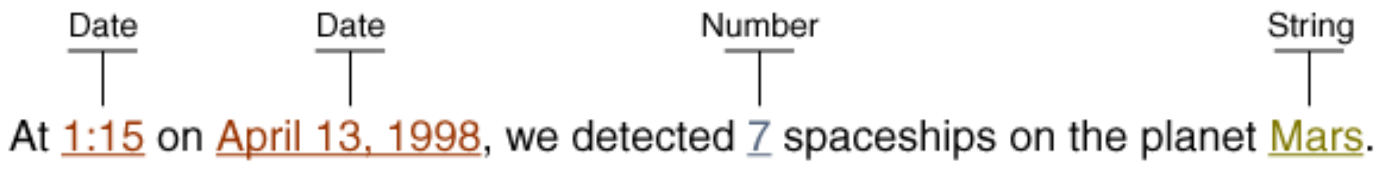
- The .properties loader for a given Locale
- Key / Value in .properties
- Naming convention for the storage

`Messages_en_EN.properties`

Language Country

```
ResourceBundle myResources =
     ResourceBundle.getBundle("MyResources", currentLocale);
```

java.util.MessageFormat



At 1:15 on April 13, 1998, we detected 7 spaceships on the planet Mars.

```
template = At {2,time,short} on {2,date,long}, \
    we detected {1,number,integer} spaceships on \
    the planet {0}.
```

```
currentLocale = en_US
At 10:16 AM on July 31, 2009, we detected 7
spaceships on the planet Mars.
currentLocale = de_DE
Um 10:16 am 31. Juli 2009 haben wir 7 Raumschiffe
auf dem Planeten Mars entdeckt.
```

- Tiny templating
- format(“<pattern>”, args)
- Date, numbers are formatted according to the Locale
- Options, conditional values easy to use

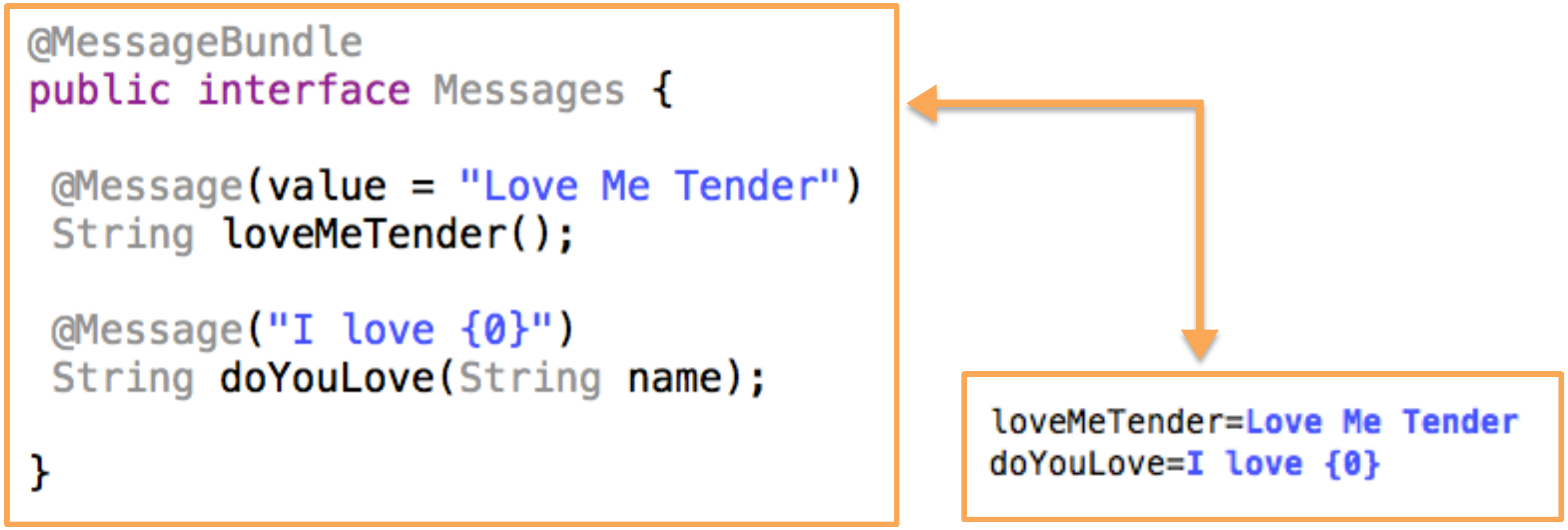
.properties issues

- Low quality control
  - – Keys are strings in the code
  - – Poor IDE support
    - No warning on unused or wrong keys
  - – Encoding Hell
    - use \uxxxx or you're in trouble
- Forces you to maintain two files in sync
  - – key declaration / value in .properties
  - – Key usage in the .java files

Improved i18n

Ez18n : improved i18n

- Interfaces representing each .properties
- The methods acts as keys



```java
@MessageBundle
public interface Messages {

 @Message(value = "Love Me Tender")
 String loveMeTender();

 @Message("I love {0}")
 String doYouLove(String name);

}
```

```properties
loveMeTender=Love Me Tender
doYouLove=I love {0}
```

Messages.java

Messages.properties

Annotations and Code generation

- Same pattern as in GWT, but for J2SE
- New Annotations in the code :
  - @MessageBundle to mark interfaces

    ➔ represents a ResourceBundle
  - @Message to mark methods

    ➔represents a localization key
- Generate :
  - .properties file (for ‘default’)
  - A ResourceBundle for each .properties
  - Manage other languages out-side your code

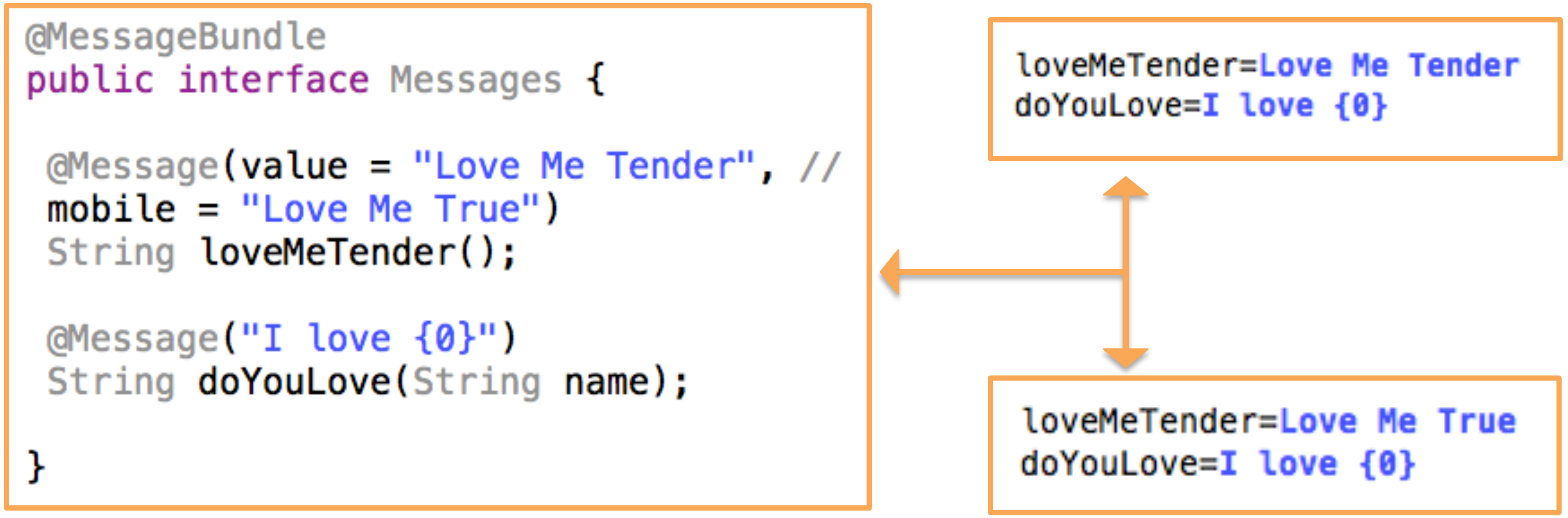
Improved i18n benefits

- Now you can
  - Refactor your keys
  - Maintain the ‘default’ in Java
  - Never change a .properties file for default locale
- And use it with other libs:
  - GWT (done on GitHub)
  - Even JQuery, Dojo, CoffeeScript (planned)
- We called that ez18n

Extend this pattern for Multi-display

Extended to displays

- Add mobile support in @Message declaration



DesktopMessages.properties

```
loveMeTender=Love Me Tender
doYouLove=I love {0}
```

```
@MessageBundle
public interface Messages {

 @Message(value = "Love Me Tender", //
 mobile = "Love Me True")
 String loveMeTender();

 @Message("I love {0}")
 String doYouLove(String name);

}
```

```
loveMeTender=Love Me True
doYouLove=I love {0}
```

Messages.java

MobileMessages.properties

Multiple kind of displays

- One ResourceBundle by kind of display
- All driven by @MessageBundle annotation
- Fallback on the default display
- Keep the plumbing generated

APT to generate .properties and ResourceBundle classes from annotations

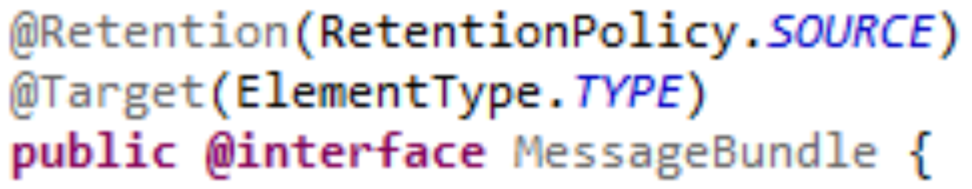
Behind the scene
How APT works

APT basics

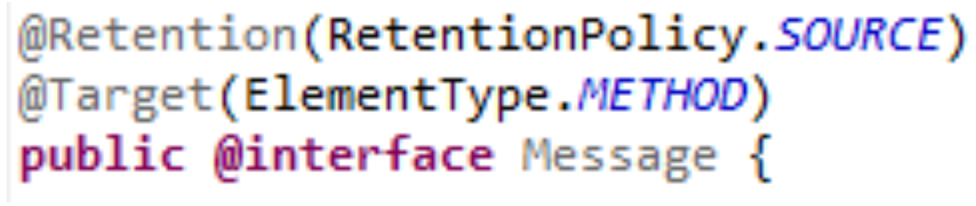
- APT - Annotation Processing Tool
- Kind of old-school pre-processing
- Standard in JDK6+ (JSR 269)
- No runtime overload
- Based on annotations in source code
- Standard since JDK 1.6 (available in Sun JDK 1.5)

APT annotations

- Use @Retention, @Target



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



```java
@Retention(RetentionPolicy.SOURCE)
@Target(ElementType.METHOD)
public @interface Message {
```

APT Processors

- javax.annotation.processing.Processor
- Code parsing similar to Reflection
  - No need of compiled code
  - Some limitations
- 2 key elements :
  - @SupportedAnnotationTypes to declare the matching annotations
  - FileObject : the future generated file

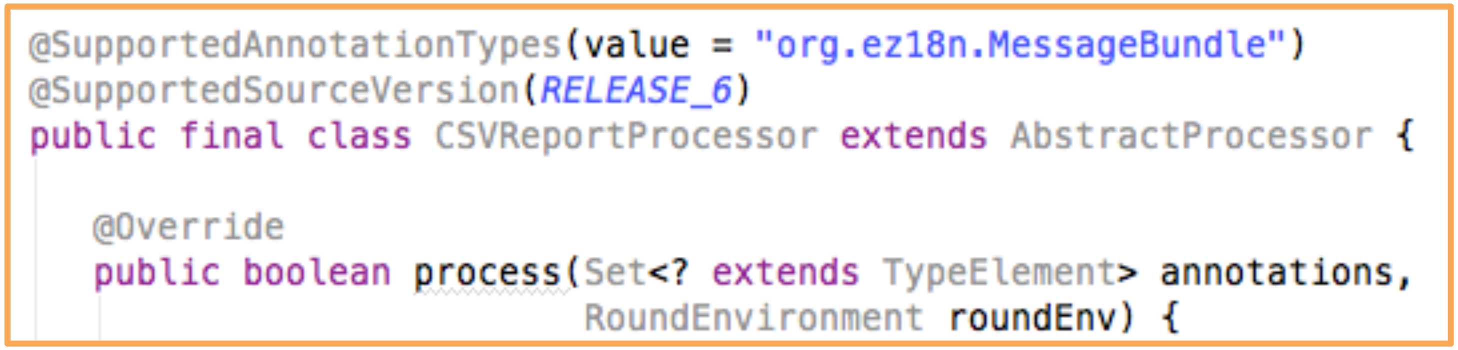
Similarities with java.lang.reflect

Java.lang.reflect	Javax.annotation.processing
java.lang.Class	TypeElement
Constructor	ExecutableElement
Field, Parameter	VariableElement
Method	ExecutableElement
java.lang.Package	PackageElement

- NO Class.newInstance()
- NO instanceOf, NO isAssignable()
- NO getConstructor, getMethod, …
- Weak inheritance support

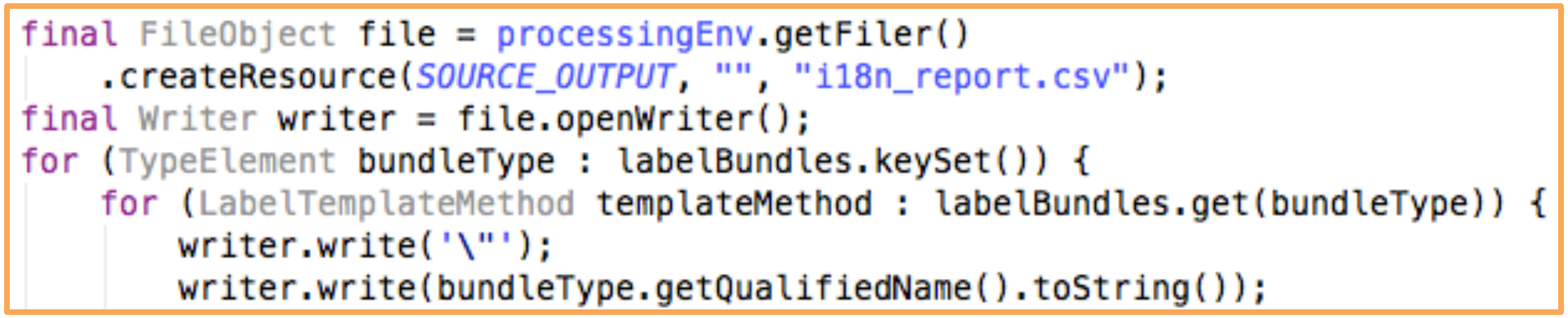
Processor code sample

- Processor declaration



```java
@SupportedAnnotationTypes(value = "org.ez18n.MessageBundle")
@SupportedSourceVersion(RELEASE_6)
public final class CSVReportProcessor extends AbstractProcessor {

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

- Use a FileObject to generate the content



```java
final FileObject file = processingEnv.getFiler()
    .createResource(SOURCE_OUTPUT, "", "i18n_report.csv");
final Writer writer = file.openWriter();
for (TypeElement bundleType : labelBundles.keySet()) {
    for (LabelTemplateMethod templateMethod : labelBundles.get(bundleType)) {
        writer.write('\"');
        writer.write(bundleType.getQualifiedName().toString());
```

APT command line

```
javac
   -cp $CLASSPATH
   -proc:only                          Or -proc:none
   -encoding UTF-8
   -processor $PROCESSOR                 processors fqcn list
   -d $PROJECT_HOME\target\classes
   -s $PROJECT_HOME\target\generated-sources\apt
   -sourcepath $SOURCE_PATH
   -verbose
   $FILES                                    optional
```

APT tooling

- Maven integration
  - maven-processor-plugin (google-code)
- Ant integration
  - javac
- IDE integration
  - Extend the JDK compilation options

APT usages

- Generate required repetitive code :
  - Not always possible at runtime
  - Unit tests, JMX declarations
  - Utility code with coverage and debug
- Build your reports on your code
  - Your metrics without runtime overload
  - Even fail the build if you want !

One or Two phase compilation

- One phase :
  - APT runs during the compilation
  - Generated code is directly produced as bytecode (.class)
  - Harder to debug (no .java created)
- Two phases : “proc:only”
  - javac with proc:only then with proc:none
  - Creates .java files in the sourcepath

Problems with APT

- Beware of the “Generate” golden hammer
  - generate needed code
- APT Processors can be tricky:
  - hard to test / maintain
  - bad error management (hidden errors !)
  - Not really (well) documented
- No built-in templating mechanism
- Enforced file path creation
- Beware of maven parallel builds
  - Because javac is not thread safe

It’s time to convince your team

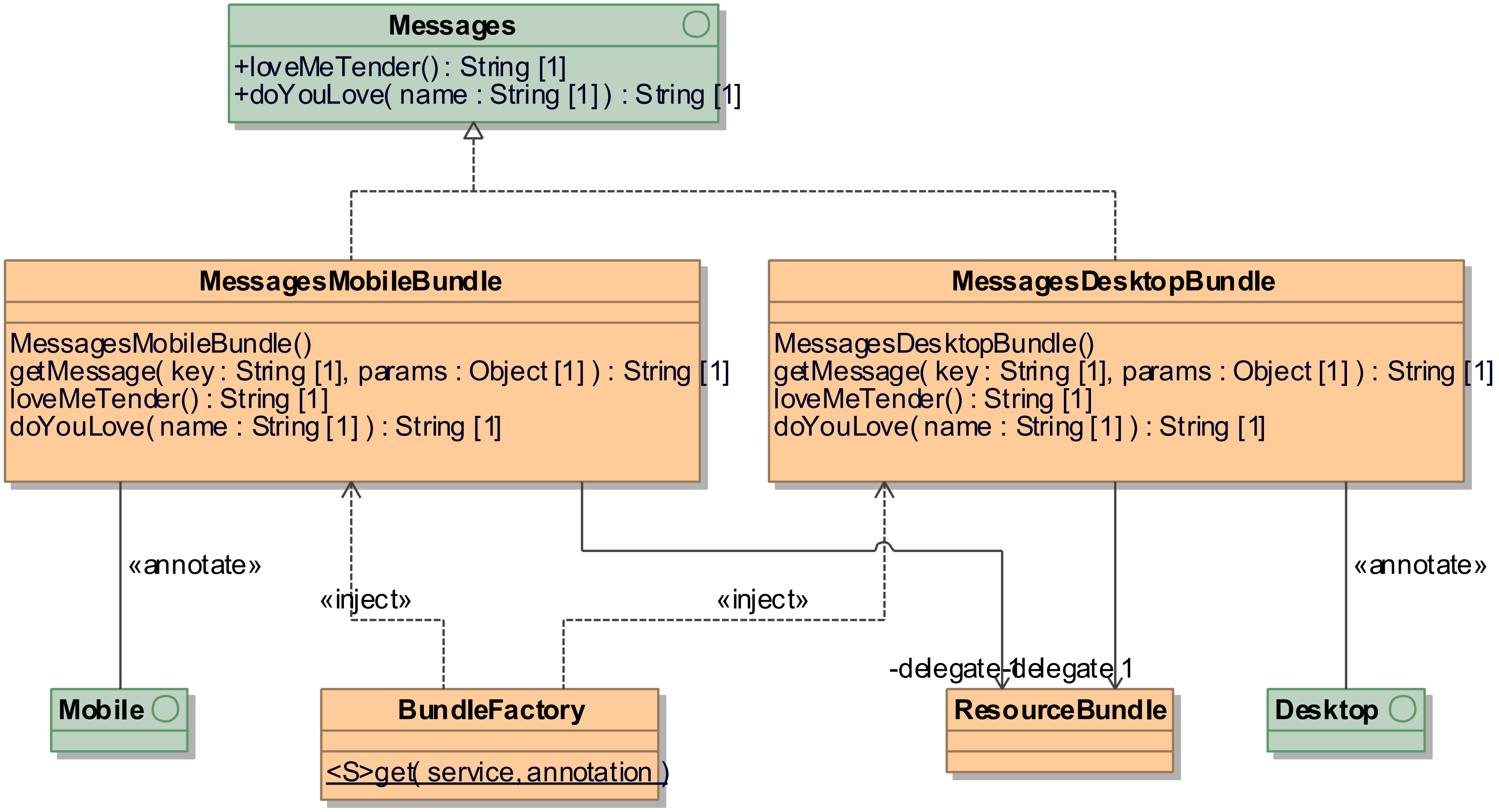
- APT parses the source code to generate
  - Java Files & .class, Reports (.csv, …)
  - Build log information or even build failures
- It allows you to have a source level DSL
  - Annotate your code & Generate the plumbing
  - Compile / Debug the generated code
- APT framework is compact

Go deep in APT usage with Ez18n

Demo

- The Stock-watcher available on
  - http://github.com/dbaeli/ez18n
  - In the ez18n-webapp module
- With a desktop browser
- With a mobile browser

Ez18n - Big picture


Messages
+loveMeTender() : String [1]
+doYouLove(name : String [1]) : String [1]
MessagesMobileBundle
MessagesMobileBundle()
getMessage(key : String [1], params : Object [1]) : String [1]
loveMeTender() : String [1]
doYouLove(name : String [1]) : String [1]
MessagesDesktopBundle
MessagesDesktopBundle()
getMessage(key : String [1], params : Object [1]) : String [1]
loveMeTender() : String [1]
doYouLove(name : String [1]) : String [1]
«annotate»
«inject»
«inject»
«annotate»
-delegate 1
-delegate 1
Mobile
BundleFactory
<S>get(service, annotation)
ResourceBundle
Desktop

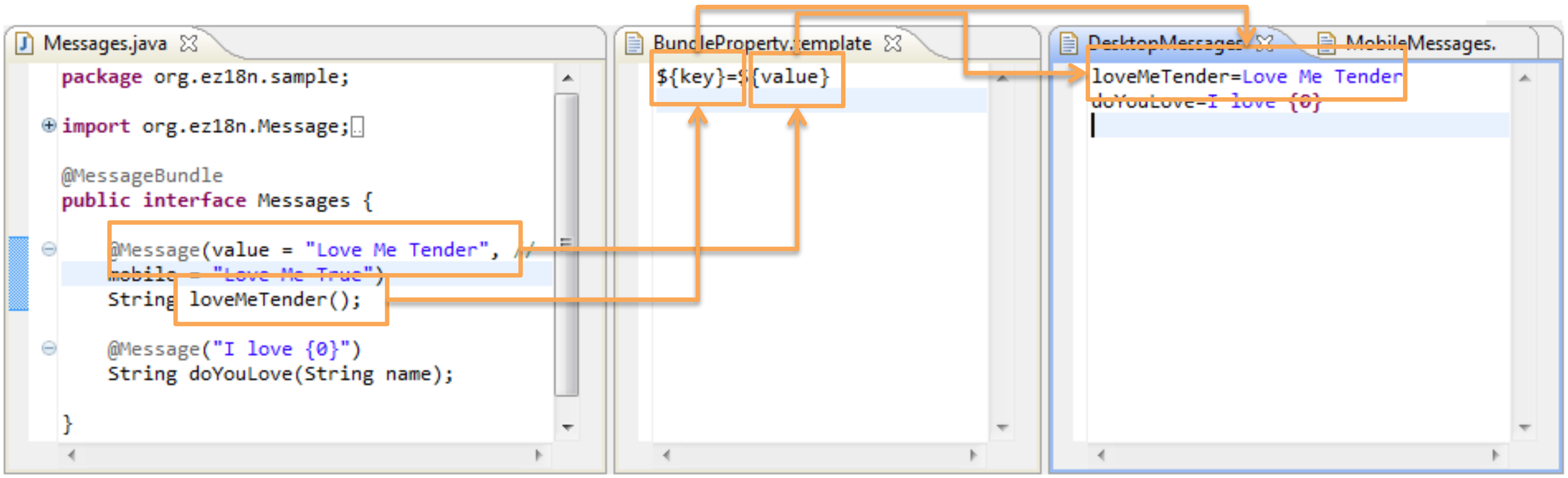
Ez18n - APT chaining

```xml
<plugin>
  <groupId>org.bsc.maven</groupId>
  <artifactId>maven-processor-plugin</artifactId>
  <executions>
    <execution>
      <id>generate-i18n-source</id>
      <goals>
        <goal>process</goal>
      </goals>
      <phase>generate-sources</phase>
      <configuration>
        <compilerArguments>-encoding UTF-8</compilerArguments>
        <outputDirectory>${project.build.directory}/generated-sources/apt</outputDirectory>
        <processors>
          <processor>org.ez18n.apt.processor.MobileBundleProcessor</processor>
          <processor>org.ez18n.apt.processor.MobileBundlePropertiesProcessor</processor>
          <processor>org.ez18n.apt.processor.DesktopBundleProcessor</processor>
          <processor>org.ez18n.apt.processor.DesktopBundlePropertiesProcessor</processor>
          <processor>org.ez18n.apt.processor.CSVReportProcessor</processor>
          <processor>org.ez18n.apt.processor.MetaInfServicesProcessor</processor>
        </processors>
      </configuration>
    </execution>
```

- 5 APT processors to obtain the default pattern
- Optional CSV files for analysis/tooling

From Messages to DesktopMessages.properties

- One property file per interface with **@MessageBundle**
- One property entry per method with **@Message**



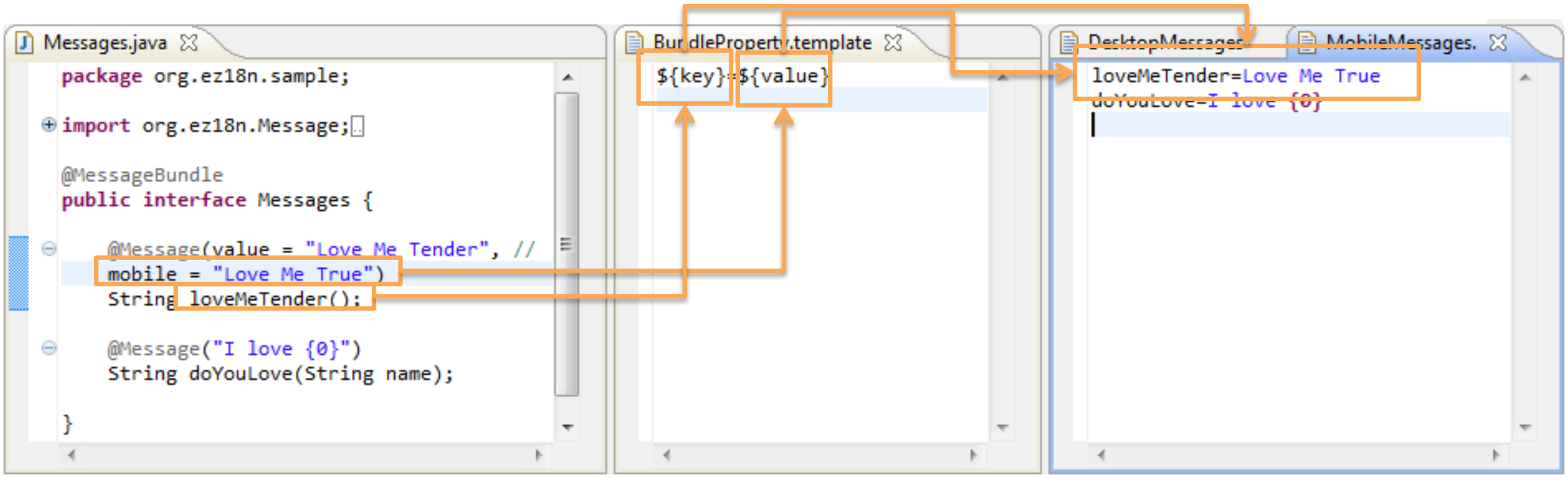
```
package org.ez18n.sample;

import org.ez18n.Message;

@MessageBundle
public interface Messages {

    @Message(value = "Love Me Tender", //
    mobile = "Love Me True")
    String loveMeTender();

    @Message("I love {0}")
    String doYouLove(String name);

}
```

```
${key}=${value}
```

```
loveMeTender=Love Me Tender
doYouLove=I love {0}
```

From Messages to MobileMessages.properties

- Another property file is generated for the mobile content
- If **@Message#mobile** is empty, the **@Message#value** is used as fallback

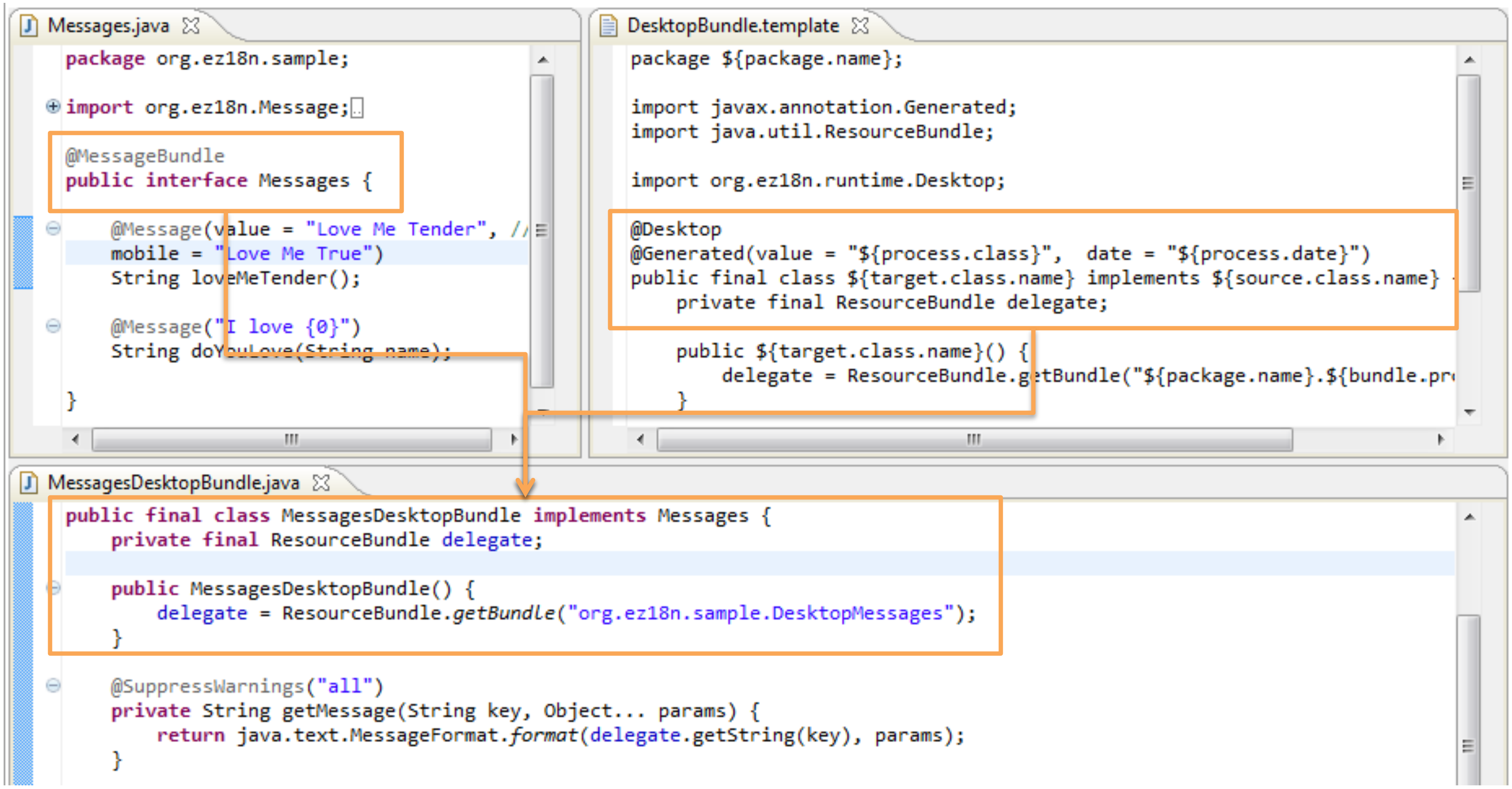


```
package org.ez18n.sample;

import org.ez18n.Message;

@MessageBundle
public interface Messages {

    @Message(value = "Love Me Tender", //
    mobile = "Love Me True")
    String loveMeTender();

    @Message("I love {0}")
    String doYouLove(String name);

}
```

```
${key}=${value}
```

```
loveMeTender=Love Me True
doYouLove=I love {0}
```

From Messages to MessagesDesktopBundle.java (1/2)



Messages.java

```
package org.ez18n.sample;

import org.ez18n.Message;

@MessageBundle
public interface Messages {

    @Message(value = "Love Me Tender", //
    mobile = "Love Me True")
    String loveMeTender();

    @Message("I love {0}")
    String doYouLove(String name);

}
```

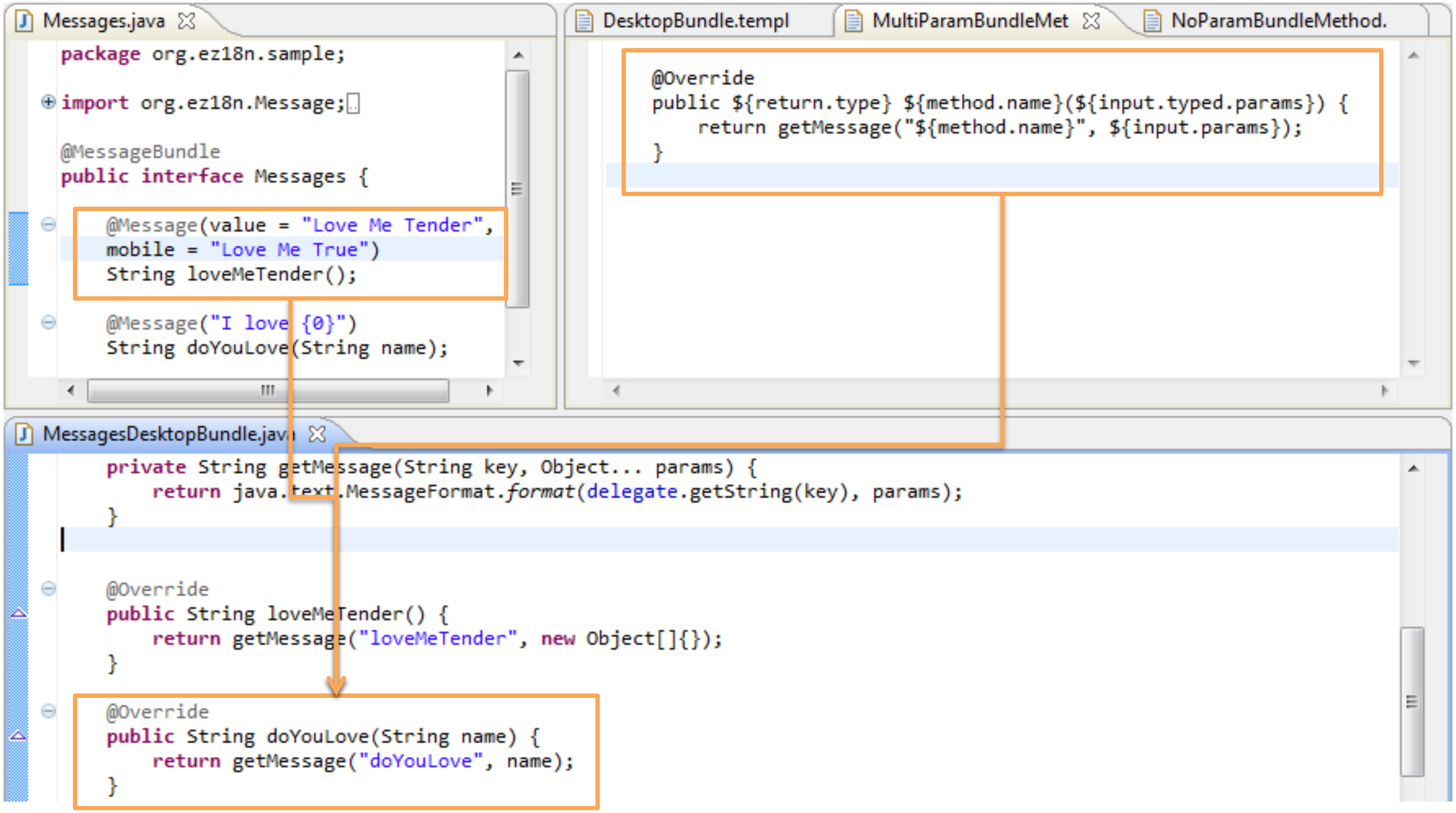
DesktopBundle.template

```
package ${package.name};

import javax.annotation.Generated;
import java.util.ResourceBundle;

import org.ez18n.runtime.Desktop;

@Desktop
@Generated(value = "${process.class}",  date = "${process.date}")
public final class ${target.class.name} implements ${source.class.name}
    private final ResourceBundle delegate;

    public ${target.class.name}() {
        delegate = ResourceBundle.getBundle("${package.name}.${bundle.pr
    }
```

MessagesDesktopBundle.java

```
public final class MessagesDesktopBundle implements Messages {
    private final ResourceBundle delegate;

    public MessagesDesktopBundle() {
        delegate = ResourceBundle.getBundle("org.ez18n.sample.DesktopMessages");
    }

    @SuppressWarnings("all")
    private String getMessage(String key, Object... params) {
        return java.text.MessageFormat.format(delegate.getString(key), params);
    }
```

From Messages to MessagesDesktopBundle.java (2/2)



Messages.java

```
package org.ez18n.sample;

import org.ez18n.Message;

@MessageBundle
public interface Messages {

    @Message(value = "Love Me Tender",
    mobile = "Love Me True")
    String loveMeTender();

    @Message("I love {0}")
    String doYouLove(String name);
```

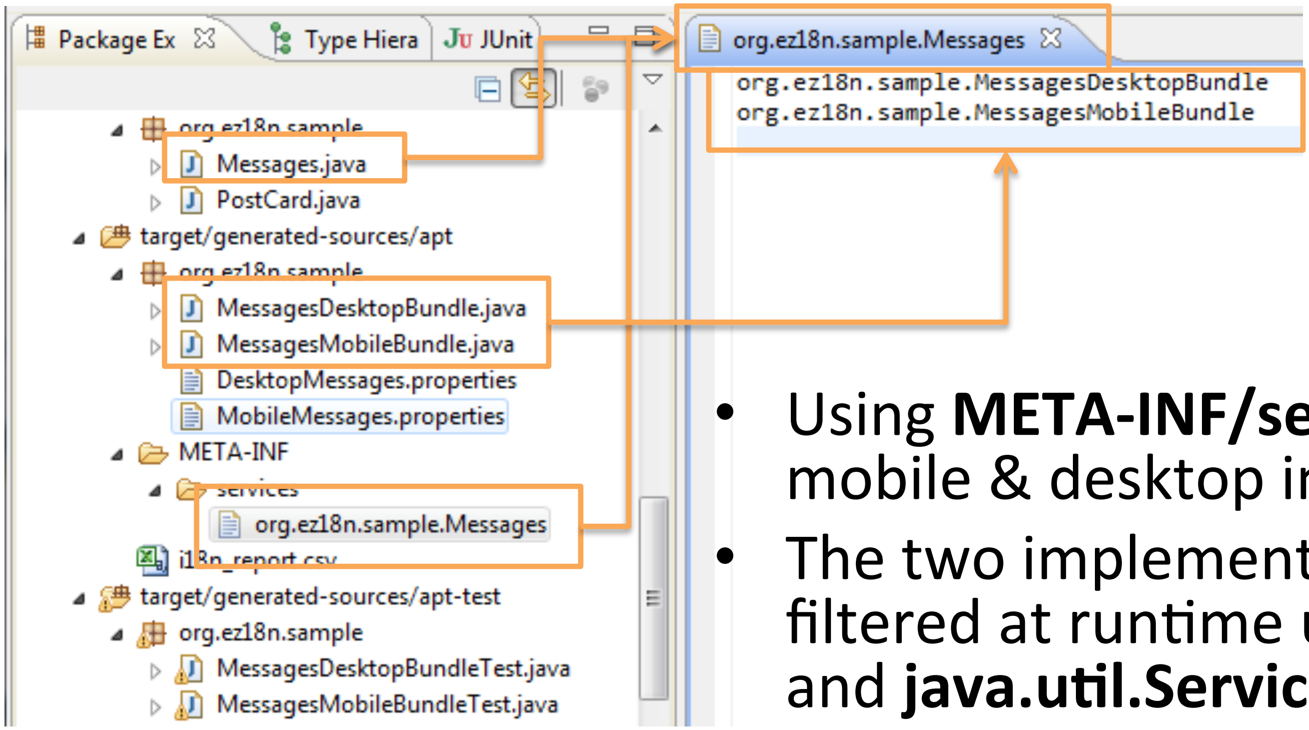
DesktopBundle.templ | MultiParamBundleMet | NoParamBundleMethod.

```
@Override
public ${return.type} ${method.name}(${input.typed.params}) {
    return getMessage("${method.name}", ${input.params});
}
```

MessagesDesktopBundle.java

```
    private String getMessage(String key, Object... params) {
        return java.text.MessageFormat.format(delegate.getString(key), params);
    }


    @Override
    public String loveMeTender() {
        return getMessage("loveMeTender", new Object[]{});
    }

    @Override
    public String doYouLove(String name) {
        return getMessage("doYouLove", name);
    }
```

From Messages to META-INF/services/org.ez18n.sample.Messages

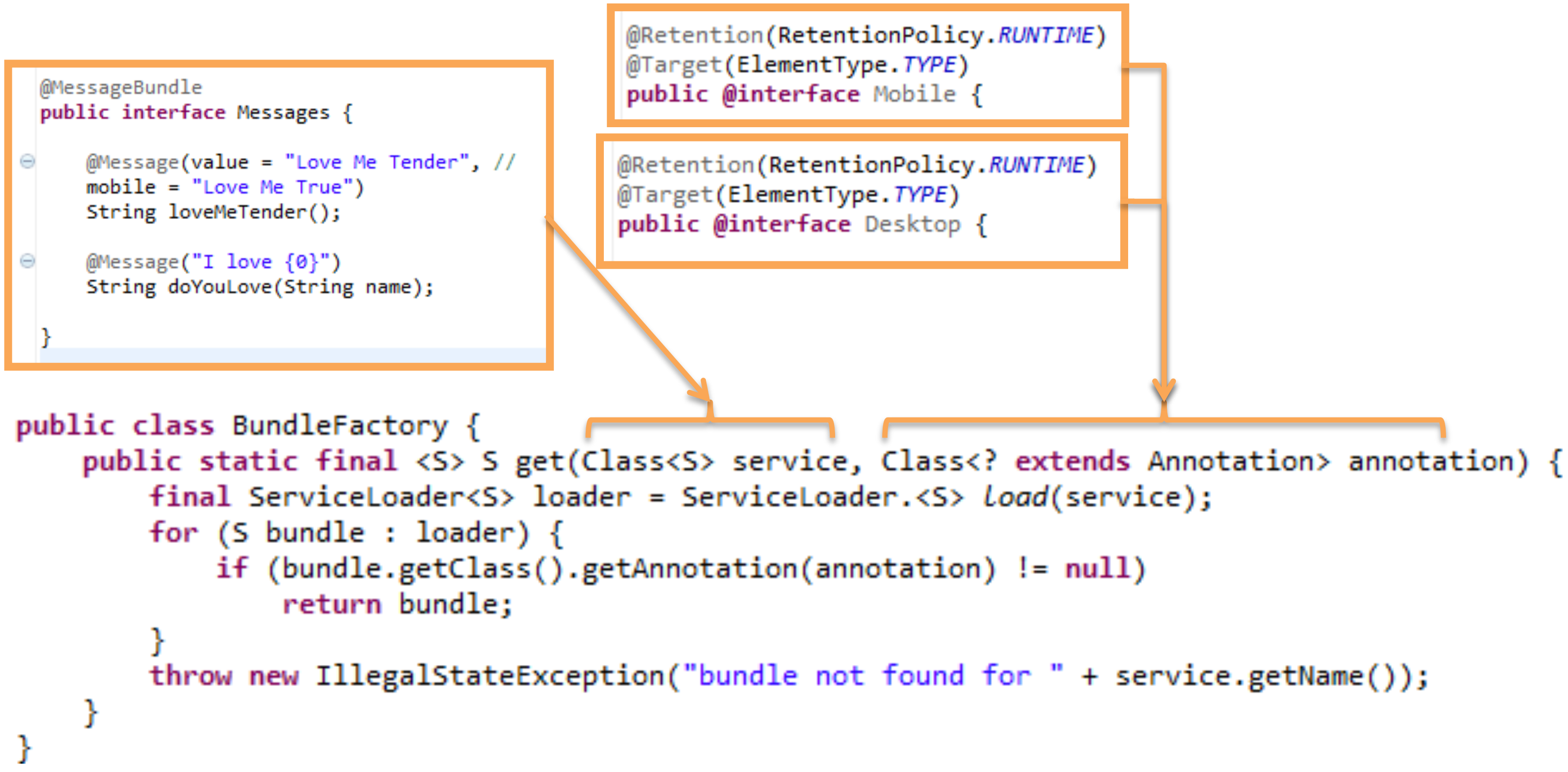

Package Ex
Type Hiera
JUnit
org.ez18n.sample
Messages.java
PostCard.java
target/generated-sources/apt
org.ez18n.sample
MessagesDesktopBundle.java
MessagesMobileBundle.java
DesktopMessages.properties
MobileMessages.properties
META-INF
services
org.ez18n.sample.Messages
i18n_report.csv
target/generated-sources/apt-test
org.ez18n.sample
MessagesDesktopBundleTest.java
MessagesMobileBundleTest.java
org.ez18n.sample.Messages

```
org.ez18n.sample.MessagesDesktopBundle
org.ez18n.sample.MessagesMobileBundle
```

- Using **META-INF/services** to inject the mobile & desktop implementation
- The two implementations could be filtered at runtime using **annotations** and **java.util.ServiceLoader**
  - **@Mobile**
  - **@Desktop**

A factory for the Messages implementations

- Using **java.util.ServiceLoader** to inject the interface with **@MessageBundle**
- **@Desktop** and **@Mobile** used to filter the injection result



```
@MessageBundle
public interface Messages {

    @Message(value = "Love Me Tender", //
    mobile = "Love Me True")
    String loveMeTender();

    @Message("I love {0}")
    String doYouLove(String name);

}
```

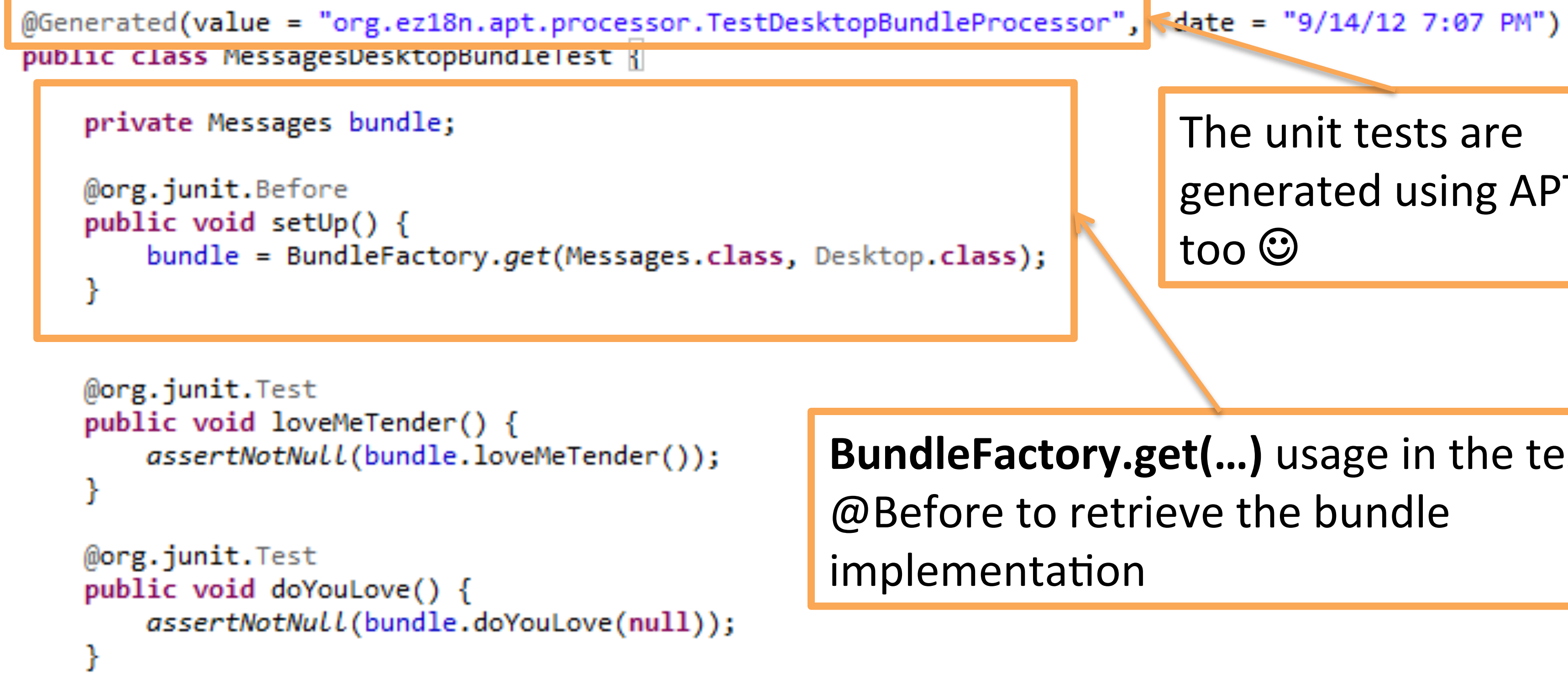
```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface Mobile {
```

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface Desktop {
```

```
public class BundleFactory {
    public static final <S> S get(Class<S> service, Class<? extends Annotation> annotation) {
        final ServiceLoader<S> loader = ServiceLoader.<S> load(service);
        for (S bundle : loader) {
            if (bundle.getClass().getAnnotation(annotation) != null)
                return bundle;
        }
        throw new IllegalStateException("bundle not found for " + service.getName());
    }
}
```

Client code sample with JUnit

- Some basic JUnit test using the API

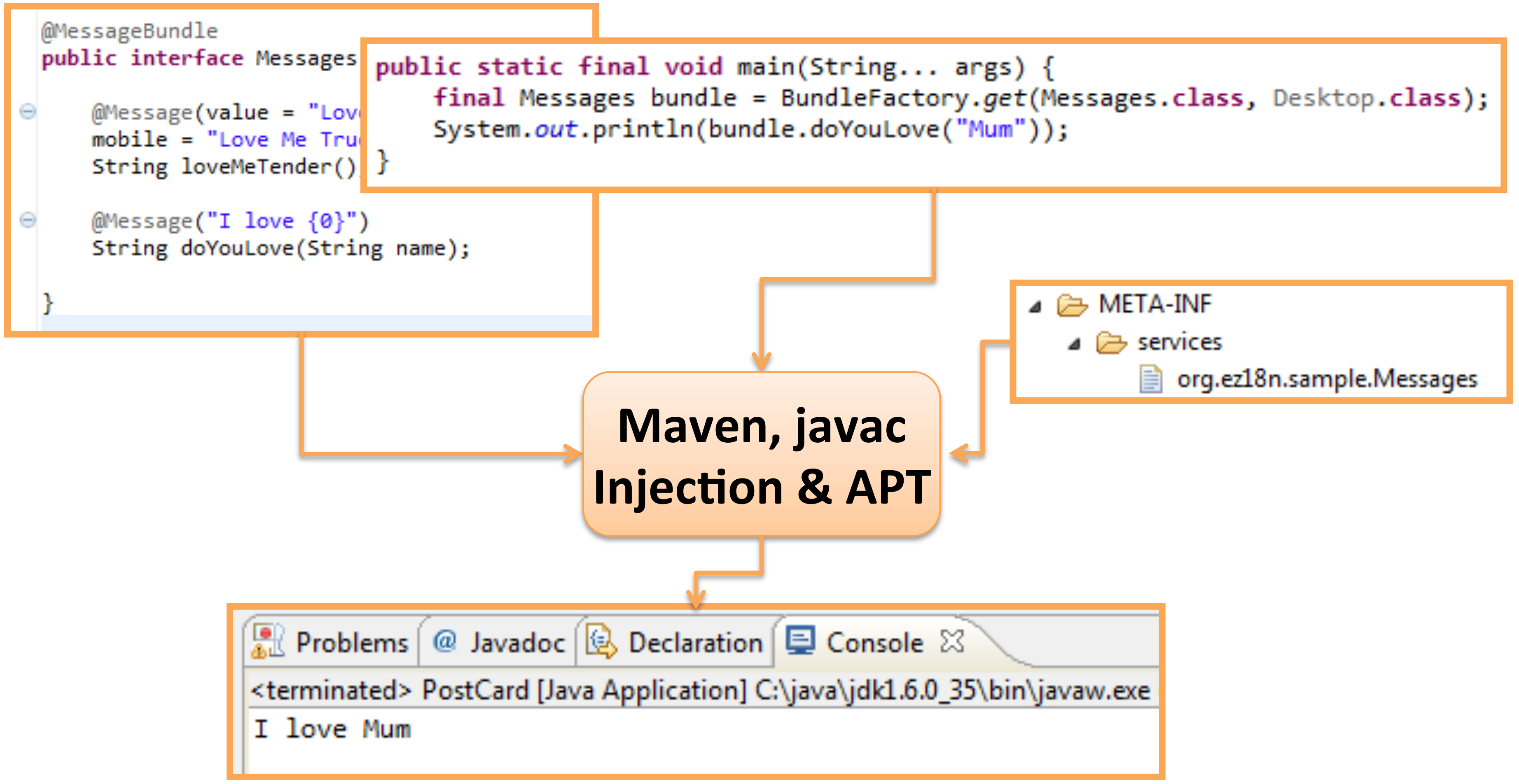


```
@Generated(value = "org.ez18n.apt.processor.TestDesktopBundleProcessor", date = "9/14/12 7:07 PM")
public class MessagesDesktopBundleTest {

    private Messages bundle;

    @org.junit.Before
    public void setUp() {
        bundle = BundleFactory.get(Messages.class, Desktop.class);
    }

    @org.junit.Test
    public void loveMeTender() {
        assertNotNull(bundle.loveMeTender());
    }

    @org.junit.Test
    public void doYouLove() {
        assertNotNull(bundle.doYouLove(null));
    }
```

The unit tests are generated using APT too ☺

BundleFactory.get(…) usage in the test @Before to retrieve the bundle implementation

Ez18n - Summary



```
@MessageBundle
public interface Messages

    @Message(value = "Love
    mobile = "Love Me Tru
    String loveMeTender()

    @Message("I love {0}")
    String doYouLove(String name);

}
```

```
public static final void main(String... args) {
    final Messages bundle = BundleFactory.get(Messages.class, Desktop.class);
    System.out.println(bundle.doYouLove("Mum"));
}
```

META-INF
services
org.ez18n.sample.Messages

Maven, javac
Injection & APT

Problems | @ Javadoc | Declaration | Console

```
<terminated> PostCard [Java Application] C:\java\jdk1.6.0_35\bin\javaw.exe
I love Mum
```

If you'd like a JSR for ez18n please tell us !

Ez18n =
@Message
@MessageBundle
Set of Processors

APT virtual mini-track

Sessions

- **CON3372 - Advanced Annotation Processing with JSR 269**
  - Jaroslav Tulach
- **CON6489 - Build Your Own Type System for Fun and Profit**
  - Werner Dietl and Michael Ernst
- **CON4469 - Annotations and Annotation Processing: What's New in JDK 8?**
  - Joel Borggrén-Franck
- **HOL3925 - Hack into Your Compiler!**
  - Jaroslav Tulach
- **CON7928 - Writing Annotation Processors to Aid Your Development Process**
  - Ian Robertson

Thanks to

- Joseph D. Darcy (APT spec lead) - https://blogs.oracle.com/darcy/

"As the lead engineer on JSR 269 in JDK 6, I'd be heartened to see greater adoption and use of annotation processing by Java developers." **Joseph D. Darcy (Oracle)**

Thank you !

Ez18n is on GitHub.
Just fork it !

https://github.com/dbaeli/ez18n