

# Multi-device Content Display & Smart Use of Annotation Processing

@gdigugli

@dbaeli

# Speakers

**@dbaeli**

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



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

**@gdigugli**

- Java developer since 1999
- Software architect at



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

Let's talk about visible quality

## Effective Content Display

# The case

## Effective Content Display

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

## with APT Tooling

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

## based on i18n

- @Message
- @MessageBundle



<https://github.com/dbaeli/ez18n>



# LesFurets.com mobile & desktop

**1 MA DEMANDE** **2 MON HISTORIQUE** **3 MON VÉHICULE**

## Conducteur principal

**Sexe :**  
☒ Homme ☐ Femme

**Date de naissance :**  
[ ] / [ ] / [ ] [calendar icon]

**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 :

Choisissez votre véhicule

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

# How to manage text display ?



# Java i18n pattern

- The provided tooling for :
  - Dynamically bind the content
  - 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

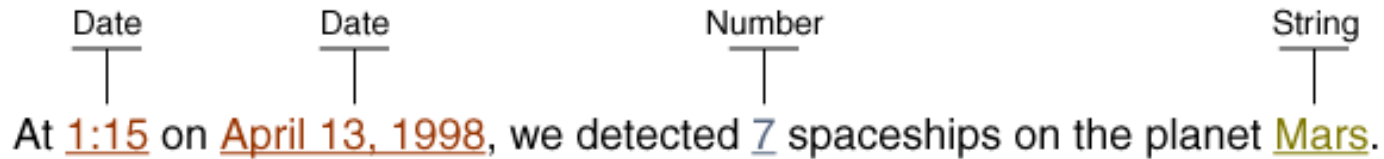
Language

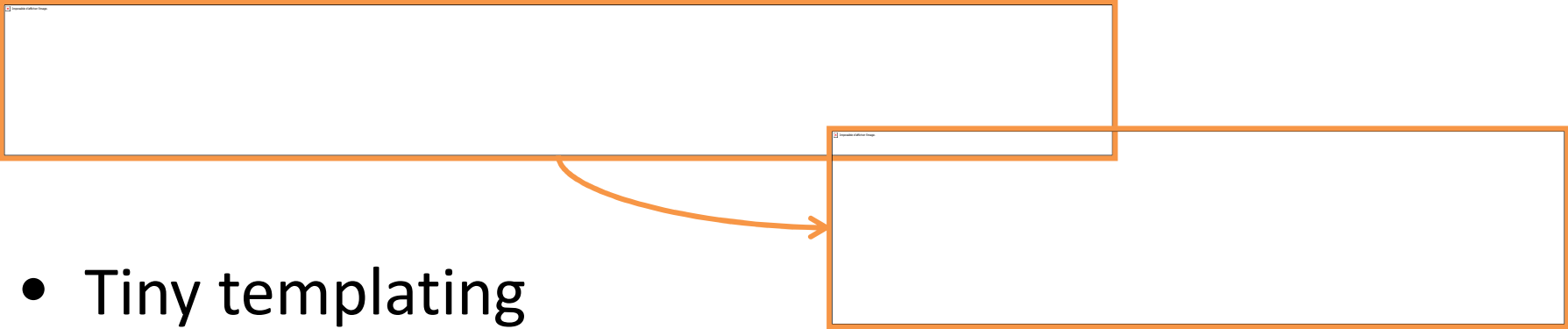
Country

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

# java.util.MessageFormat

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



- 
- 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

# Improved i18n

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

```
@MessageBundle
public interface Messages {

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

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

}
```

Messages.java

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

Messages.properties



# Annotations and Code generation

- Same pattern as in GWT, but for J2SE
- Annotate your 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)

# Extend this pattern for Multi-display

# Extended to displays

- Add mobile support in @Message declaration

```
@MessageBundle
public interface Messages {

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

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

Messages.java

DesktopMessages.properties

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

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

MobileMessages.properties

# Extended to 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
- But not on the file it-self
- 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

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

```
@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

# Processor code sample

- Processor declaration

```
@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

```
final FileObject file = processingEnv.getFile()
    .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());
```

# Similarities with `java.lang.reflect`

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

- NO `Class.newInstance()`
- NO `instanceOf`, NO `isAssignable()`
- NO `getConstructor`, `getMethod`, ...
- Weak inheritance management



# APT command line

javac

-cp \$CLASSPATH

-proc:only

→ -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
  - javac with proc:only then with proc:none
  - Creates .java files in the sourcepath
  - Not really supported by IDEs, ok with maven.

# 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 // builds
  - javac is not thread safe

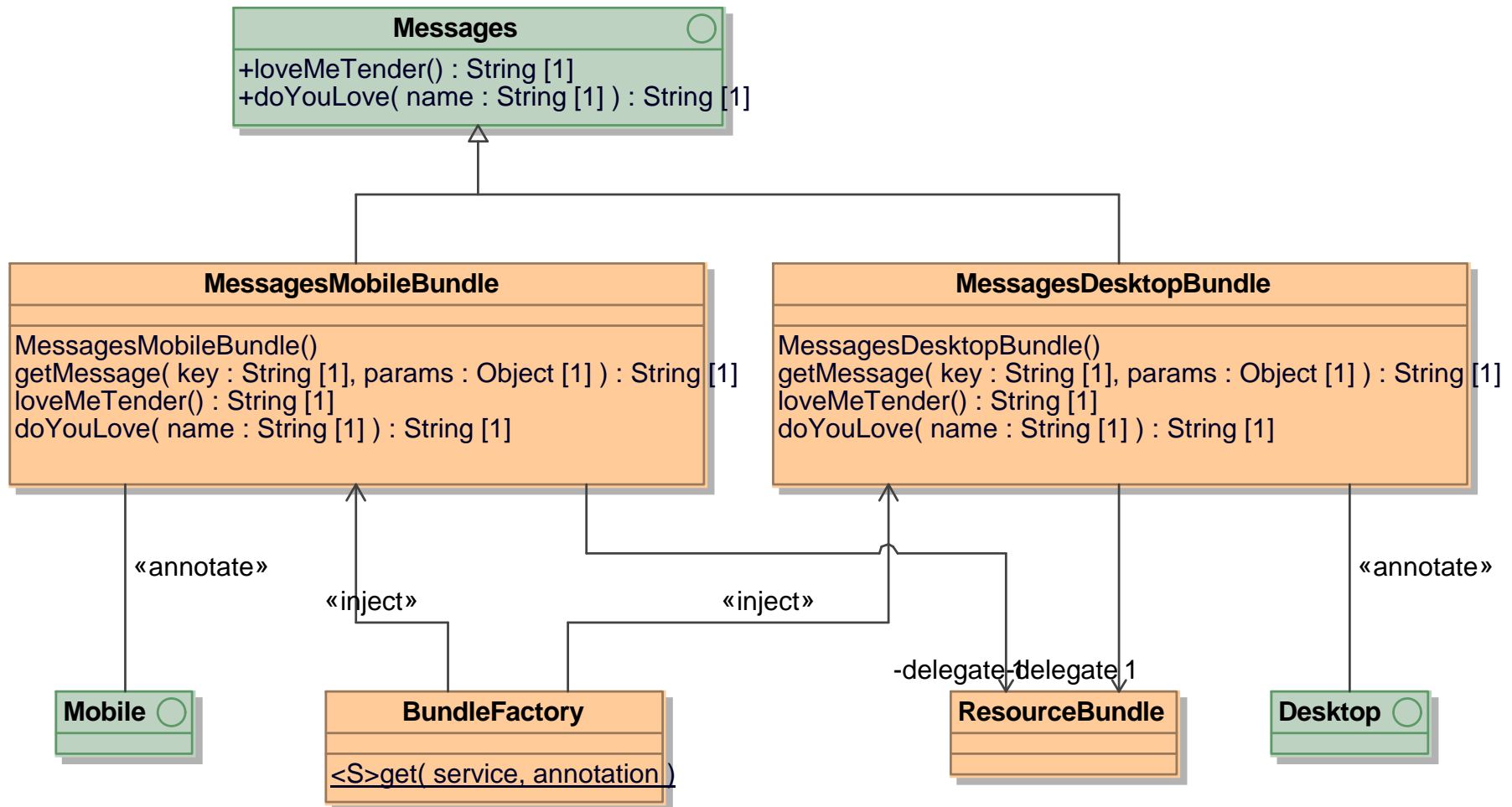
# It's time to convince your team

- 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 real code
- APT framework is compact
- Learning curve is low



# Go deep in Ez18n

# Ez18n - Big picture



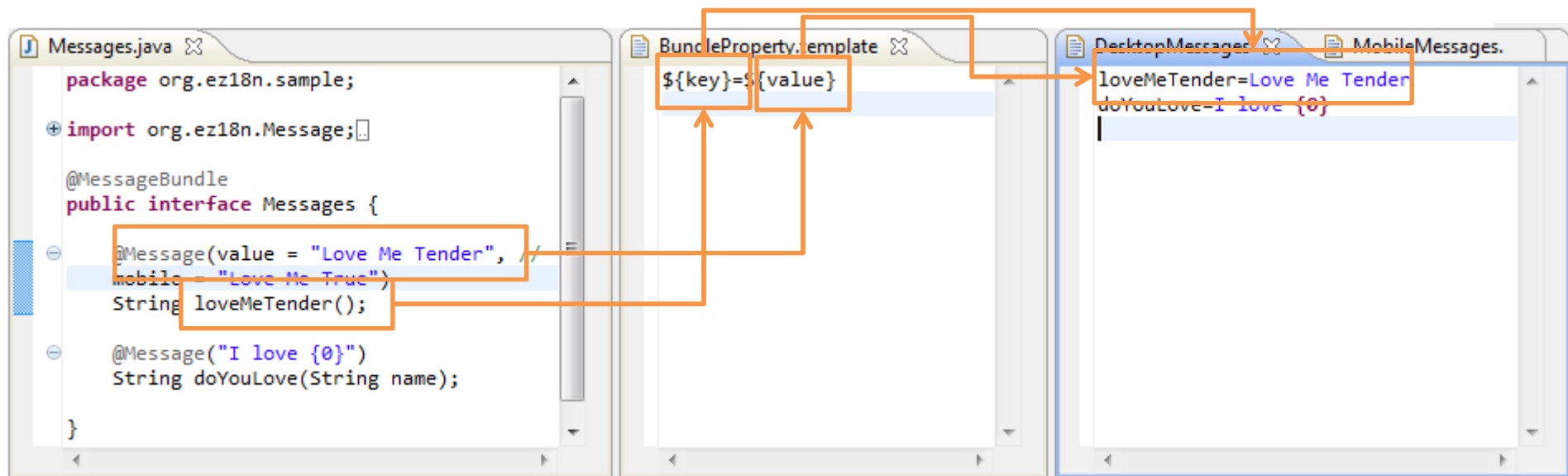
# Ez18n - APT chaining

```
<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.apr.processor.MobileBundleProcessor</processor>
          <processor>org.ez18n.apr.processor.MobileBundlePropertiesProcessor</processor>
          <processor>org.ez18n.apr.processor.DesktopBundleProcessor</processor>
          <processor>org.ez18n.apr.processor.DesktopBundlePropertiesProcessor</processor>
          <processor>org.ez18n.apr.processor.CSVReportProcessor</processor>
          <processor>org.ez18n.apr.processor.MetaInfServicesProcessor</processor>
        </processors>
      </configuration>
    </execution>
  </executions>
</plugin>
```

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

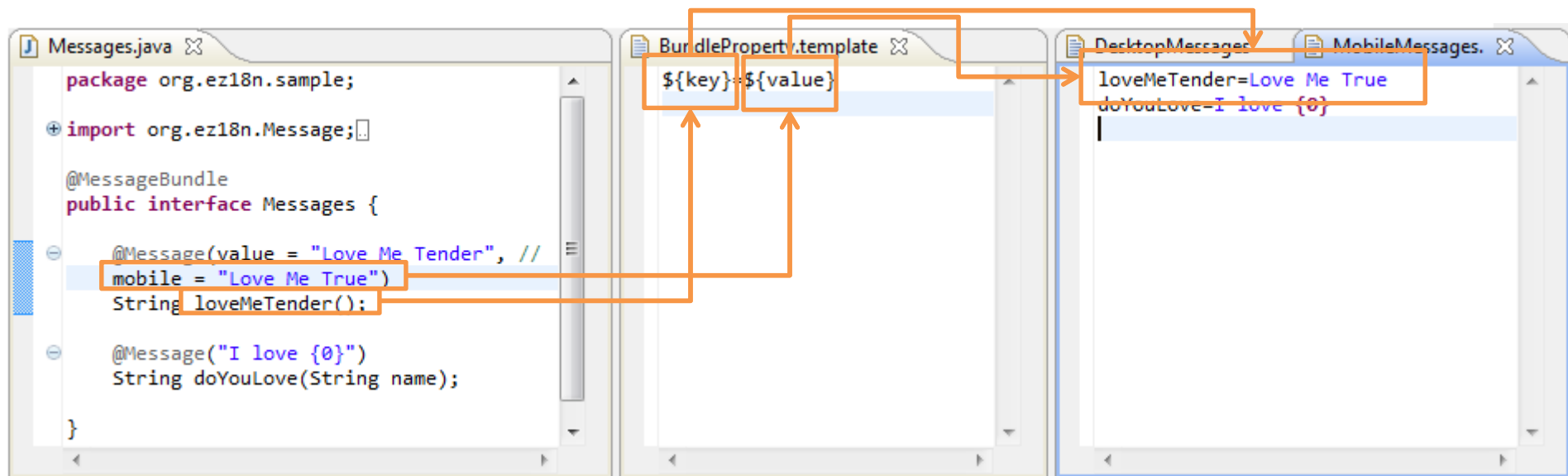
# From Messages to DesktopMessages.properties

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



# 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



# From Messages to MessagesDesktopBundle.java (1/2)

The screenshot displays three Java files in an IDE:

- Messages.java**:
  - Package: `org.ez18n.sample;`
  - Import: `import org.ez18n.Message;`
  - Annotation: `@MessageBundle`
  - Interface: `public interface Messages {`
  - Message 1: `@Message(value = "Love Me Tender", // mobile = "Love Me True") String loveMeTender();`
  - Message 2: `@Message("I love {0}") String doYouLove(String name);`
- DesktopBundle.template**:
  - Package: `package ${package.name};`
  - Imports: `import javax.annotation.Generated;` and `import java.util.ResourceBundle;`
  - Import: `import org.ez18n.runtime.Desktop;`
  - Annotation: `@Desktop`
  - Annotation: `@Generated(value = "${process.class}", date = "${process.date}")`
  - Class Declaration: `public final class ${target.class.name} implements ${source.class.name}`
  - Field: `private final ResourceBundle delegate;`
  - Constructor: `public ${target.class.name}() { delegate = ResourceBundle.getBundle("${package.name}.${bundle.pr`
- MessagesDesktopBundle.java**:
  - Class Declaration: `public final class MessagesDesktopBundle implements Messages {`
  - Field: `private final ResourceBundle delegate;`
  - Constructor: `public MessagesDesktopBundle() { delegate = ResourceBundle.getBundle("org.ez18n.sample.DesktopMessages"); }`
  - Annotation: `@SuppressWarnings("all")`
  - Method: `private String getMessage(String key, Object... params) { return java.text.MessageFormat.format(delegate.getString(key), params); }`

An orange box in **Messages.java** highlights the `@MessageBundle` annotation and the `Messages` interface. An orange box in **DesktopBundle.template** highlights the `@Desktop` annotation, the `@Generated` annotation, the class declaration, and the `delegate` field. An orange box in **MessagesDesktopBundle.java** highlights the class declaration, the `delegate` field, and the constructor. An orange arrow points from the `@MessageBundle` annotation in **Messages.java** to the `MessagesDesktopBundle` class in **MessagesDesktopBundle.java**.



# From Messages to MessagesDesktopBundle.java (2/2)

The screenshot shows three files in an IDE:

- 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**:

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

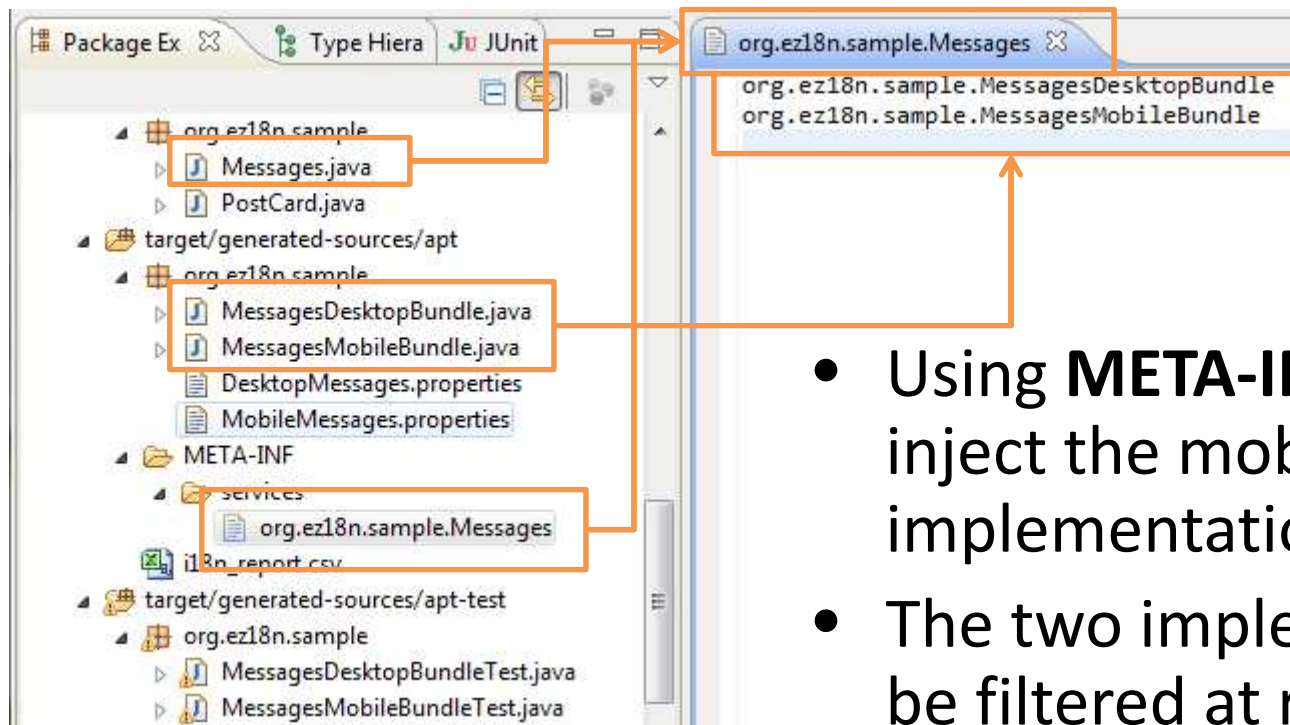
```
private String getMessage(String key, Object... params) {
    return java.text.MessageFormat.format(delegate.getString(key), params);
}

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

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

Orange boxes highlight the message definitions in `Messages.java`, the `getMessage` call in the `DesktopBundle.template`, and the `getMessage` calls in `MessagesDesktopBundle.java`. An arrow points from the `loveMeTender` message definition in `Messages.java` to its implementation in `MessagesDesktopBundle.java`.

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

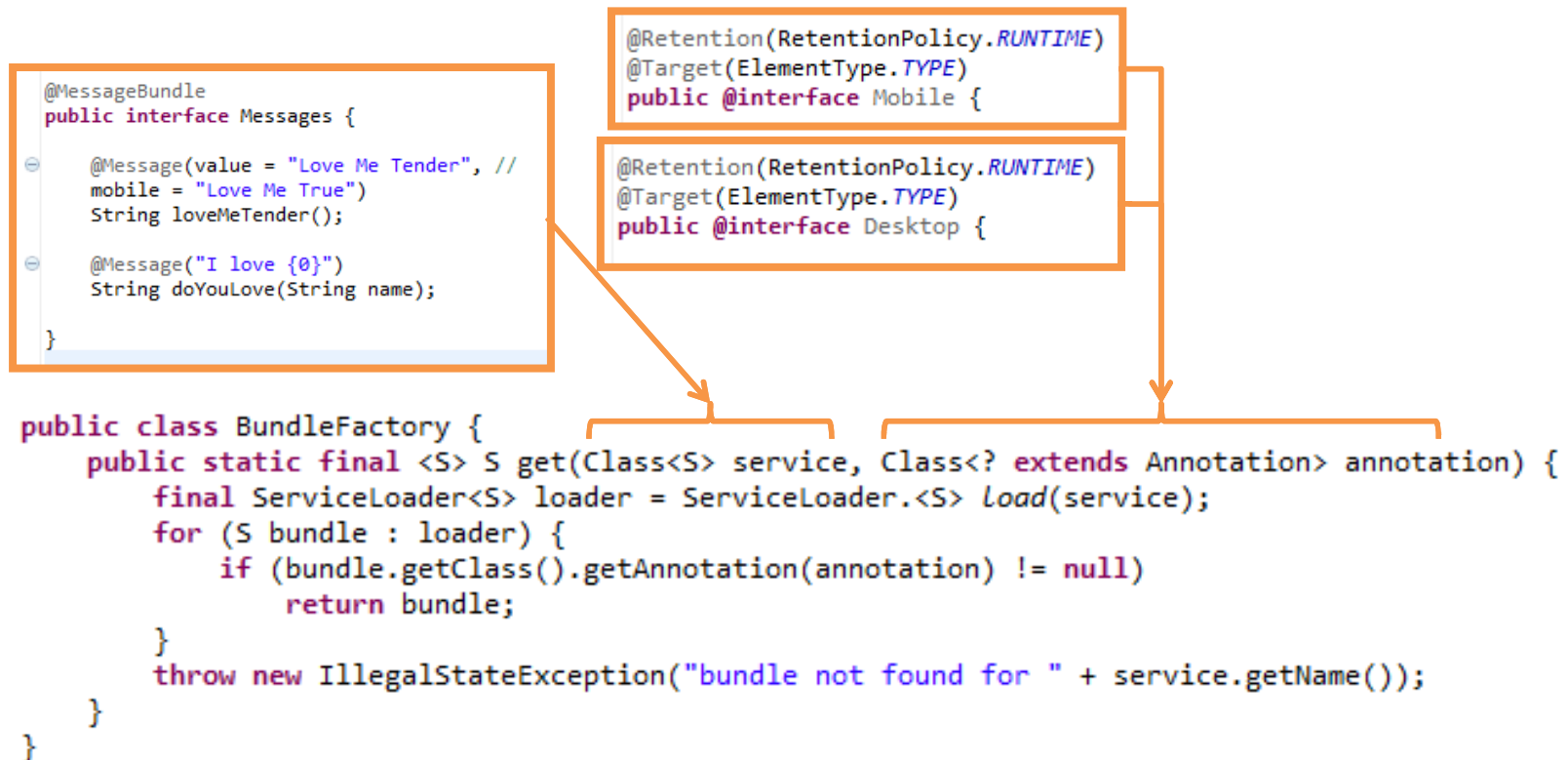


- Using **META-INF/services** to inject the mobile & desktop implementation
- The two implementation could be filtered at runtime using annotations and **ServiceLoader**
  - **@Mobile**
  - **@Desktop**



# 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



# Client code sample with JUnit

- Some basic JUnit test using the API

```
@Generated(value = "org.ez18n.apr.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);  
    }
```

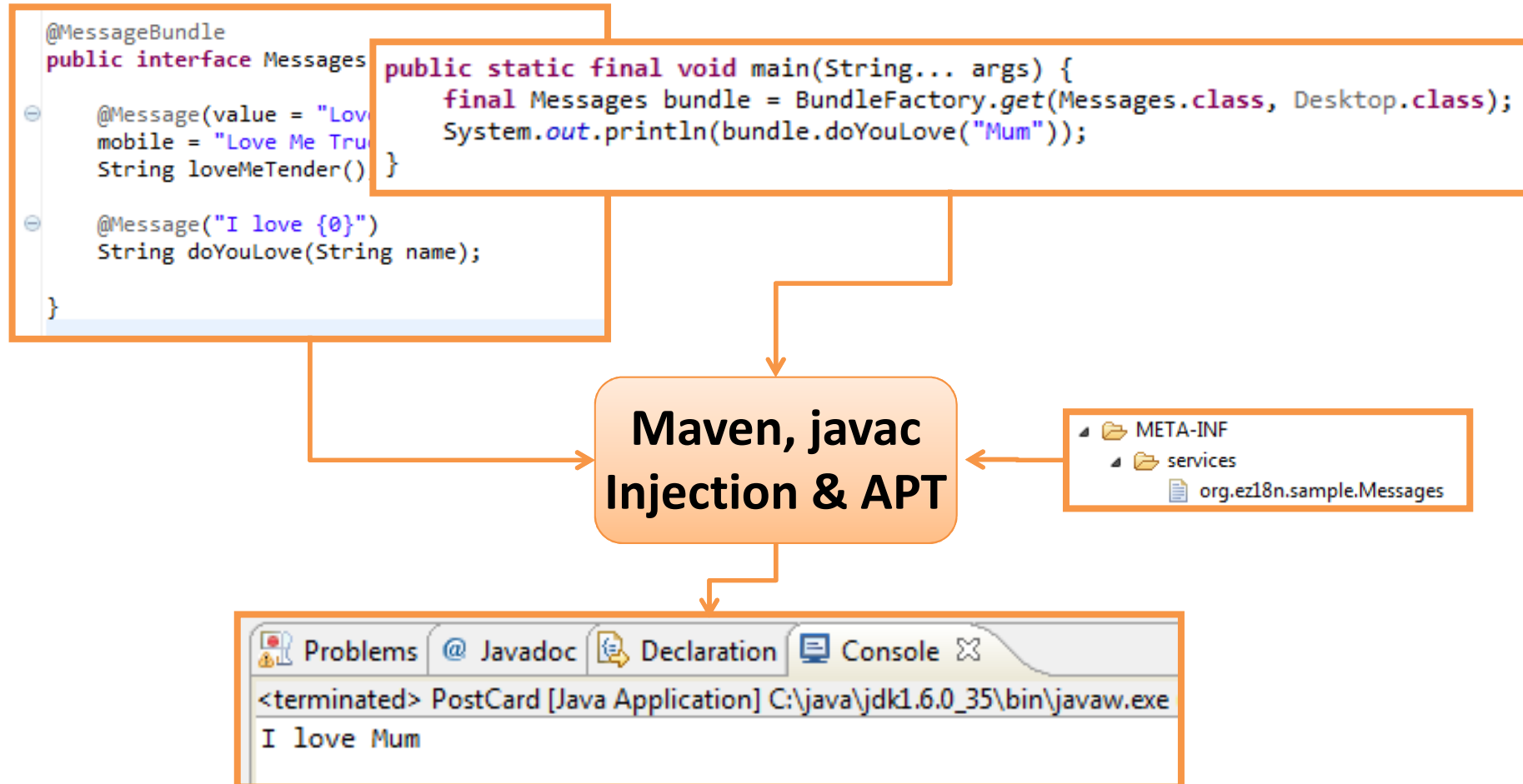
```
    @org.junit.Test  
    public void loveMeTender() {  
        assertNotNull(bundle.loveMeTender());  
    }
```

```
    @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



**If you think there  
is room for a JSR  
Please tell us.**



# Thank you !

## Ez18n is on GitHub. Just fork it !



<https://github.com/dbaeli/ez18n>

