


# Java tečaj

9. dio

Swing (2)

# Teme

- ♦ Izrada uređivača teksta, ili kako se radi s:
    - Izbornicima,
    - Toolbarima,
    - Tipkovničkim kraticama,
    - Dokumentima, ...
    - Lokalizacija (Internacionalizacija, i18n)
- 
- A stylized, dark teal mountain range graphic is located in the bottom right corner of the slide, extending from the right edge towards the center.

# Poučan tekst...

Preporučam pročitati:

## A Swing Architecture Overview

<http://www.oracle.com/technetwork/java/architecture-142923.html>



# Praktičan problem

- ◆ Problem:  
želim napraviti da kada korisnik  
zatvori prozor, provjerim je li OK  
zatvoriti aplikaciju

npr. Zatvarate uređivač dokumenta, on vidi  
da ima promjena koje nisu snimljene i  
nudi snimanje, ignoriranje izmjena ili  
otkazivanje zatvaranja

# Praktičan problem

## ◆ Rješenje:

- Postaviti `defaultCloseOperation` na `DO_NOTHING_ON_CLOSE`
- Registrirati `WindowListener` i u metodi `onWindowClosing(...)` napraviti provjeru; ako je OK zatvoriti prozor, pozvati metodu `dispose()`.

Naime, kad kliknemo na X-ić kojim želimo zatvoriti prozor, generira se događaj `windowClosing`, pa ako se prozor stvarno i zatvori, događaj `windowClosed`.

# UT - organizacija kôda

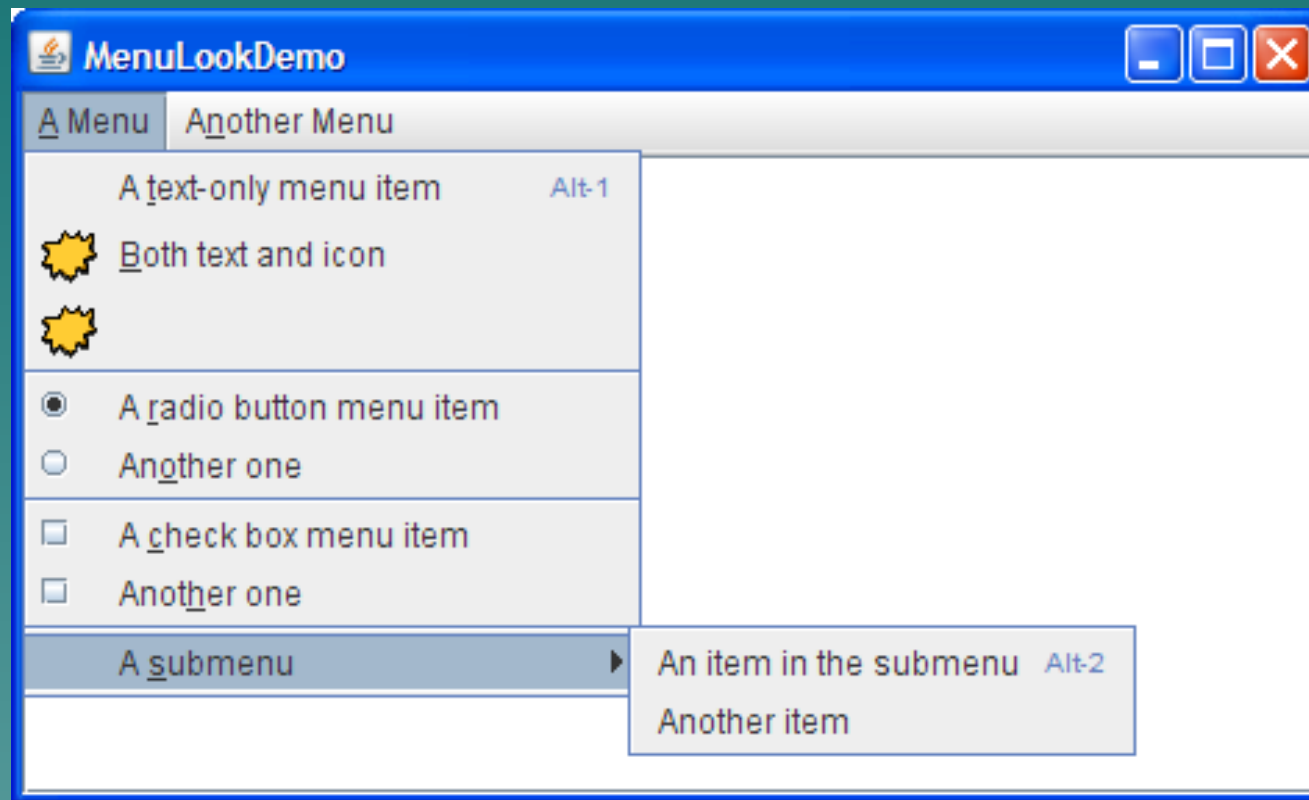
- ◆ Uređivač teksta možemo napraviti na više načina
  - Jedna komponenta tipa `JTextArea`,
  - Jedna komponenta tipa `JTextPane`,
  - Više uređivača unutar `JTabbedPane`-a
- ◆ Želimo:
  - Funkcionalan GUI!

# Pregled potrebnoga...

- ◆ Da bismo izgradili jednu takvu aplikaciju, trebat ćemo:
  - Izbornike
  - Alatne trake
  - Uporabu tipkovničkih kratica
  - I18n

# Izbornici

## ◆ Izbornici:





# Izbornici

- ◆ Izbornici se grade uporabom:
  - `JMenuBar` (sustav izbornika; u `JFrame` ga dodajemo sa `setJMenuBar(...)`)
  - `JMenu` (jedan "izbornik"; npr. File, Edit i slično)
  - `JMenuItem` (jedna izbornička stavka; npr. "Paste")
  - Stavke se gnijezde
  - Moguća izgradnja hijerarhije

# Izbornici

- ◆ Dodavanje kôda:
  - Kod koji želimo pokrenuti kada se klikne na konkretnu stavku (`JMenuItem`) toj stavki možemo dodati uporabom metode `addActionListener(...)`
  - Stavki možemo definirati naziv, mnemonik te akceleratorSKU kombinaciju

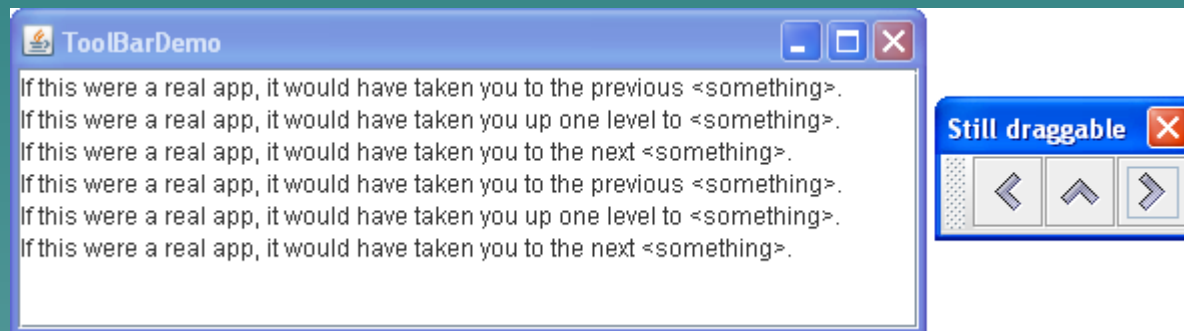
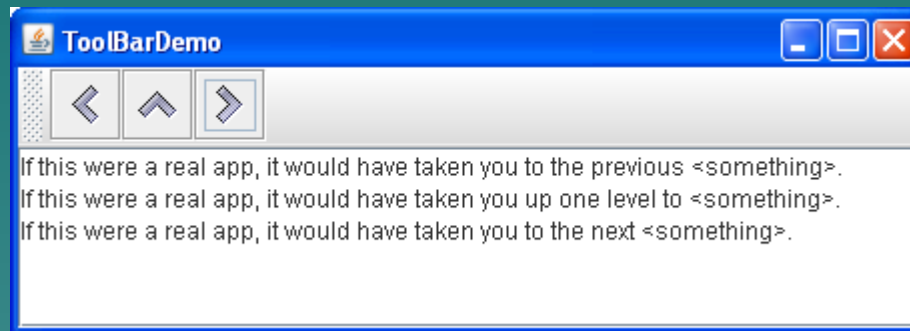


Both text and icon

An item in the submenu Alt-2

# Alatne trake

## ◆ Alatna traka:



# Alatne trake

- ◆ Alatna traka gradi se uporabom komponente `JToolBar`
  - Komponenta može "floatati"
    - ◆ Tada mora biti u kontejneru koji koristi `BorderLayout`, gdje ona nije `CENTER` i gdje postoji još samo jedna komponenta koja je `CENTER`
  - U alatnu traku možemo dodavati svašta: gumbe, separatore, `JTextField`-ove itd.

# Zadavanje akcija

- ◆ Dodavanje kôda:
  - Kod koji želimo pokrenuti kada se klikne na konkretni gumb (`JButton`) tom gumbu možemo dodati uporabom metode `addActionListener(...)`

# Zadavanje akcija

- ◆ Uporaba tipkovničkih kratica – dodavanje kôda:
  - Kod koji želimo pokrenuti kada se pritisne određena kombinacija tipki možemo dodati uporabom metode `addKeyListener(...)`

# Zadavanje akcija

- ◆ Masovna redundancija!
  - Na više načina možemo pokretati konceptualno iste poslove
  - Ako loše organiziramo kôd, postoji čak mogućnost da više puta pišemo identičan kôd!
  - Puno bolje rješenje: izdvojiti kôd koji predstavlja pojedine poslove u zasebne "obogaćene" objekte
  - Osloniti se na oblikovni obrazac *Naredbu*

# Oblikovni obrazac *Naredba*

- ◆ Naredba (engl. *Command*)
  - Ideja je razdvojiti implementacijske detalje poslova od pozivatelja: nužna uporaba apstrakcije
  - Pozivatelj poslove vidi preko generičkog sučelja
  - Pristup omogućava implementaciju prikaza raspoloživih poslova na generički način (primjerice, možemo graditi izborničke strukture koje ništa ne znaju o detaljima poslova)

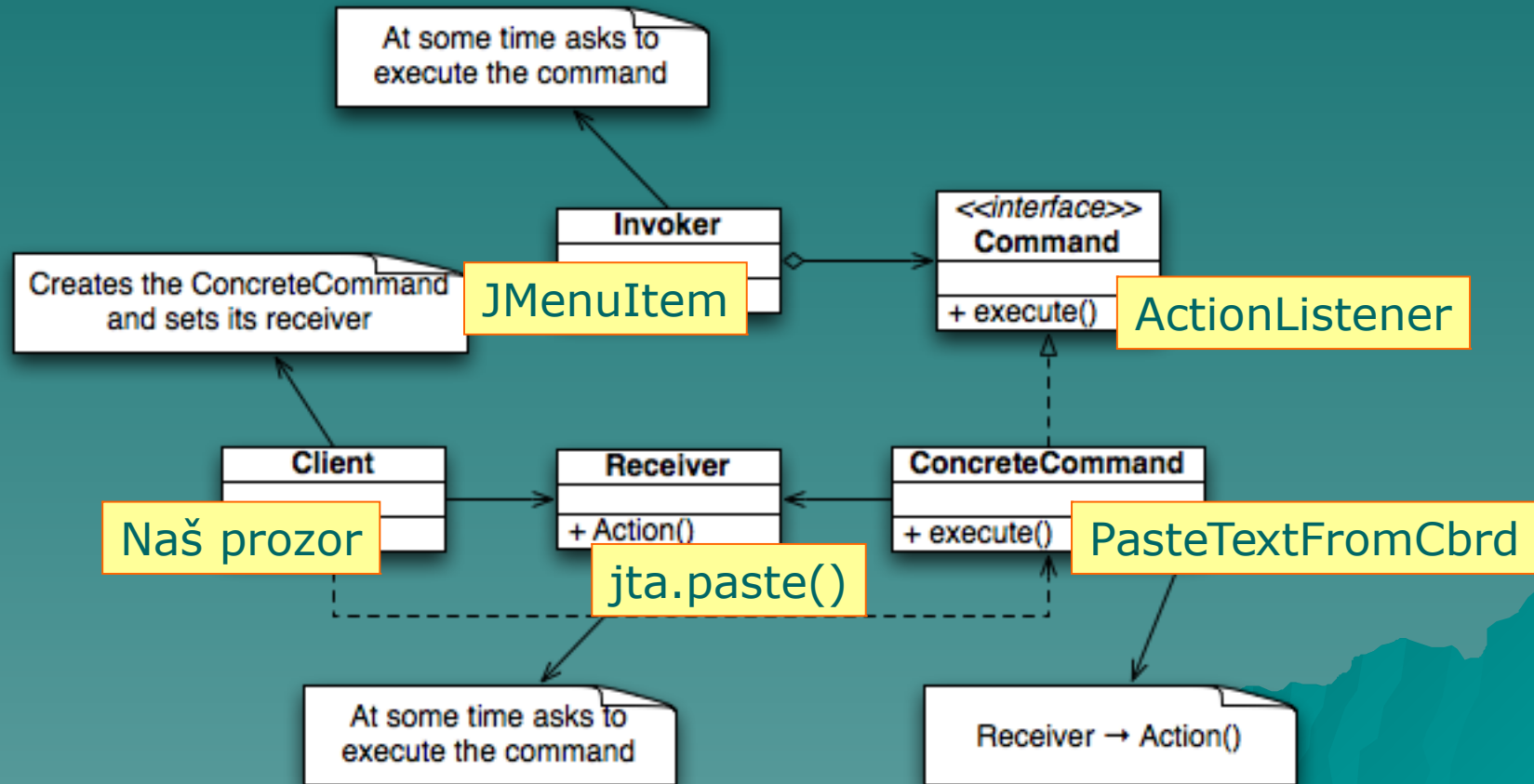


# Oblikovni obrazac *Naredba*

- ◆ Naredba (engl. *Command*)
  - Jednostavan primjer: posao se modelira kao razred koji implementira sučelje `ActionListener` i takav se može dodati na različite GUI-komponente koje ne znaju implementacijske detalje, ali ipak mogu pozivati taj posao
  - Zašto i kako? Tj. Što sve pozivatelj treba znati o poslu?

# Oblikovni obrazac *Naredba*

## ◆ Naredba (engl. *Command*)



# Poopćenje akcija

- ◆ Kako bi dodatno pospješio dijeljenje i višestruku iskoristivost kôda, u Swingu je napravljeno daljnje poopćenje ovog rješenja: sučelje `Action`

# Poopćenje akcija

## ◆ Sučelje `Action`

- To je proširenje sučelja `ActionListener`
- Dodaje mogućnost pohrane raznih informacija o samoj akciji: naziv, mnemonik, akcelerator, hint, ikonu
- Nudi mogućnost omogućavanja / onemogućavanja akcije

# Poopćenje akcija

## ◆ Sučelje Action

- Konceptualno sadrži mapu koja mapira nazive svojstava na trenutne vrijednosti
- Ključevi su statičke konstante sučelja Action:

NAME, SHORT\_DESCRIPTION,  
ACTION\_COMMAND\_KEY, MNEMONIC\_KEY,  
DISPLAYED\_MNEMONIC\_INDEX\_KEY,  
LARGE\_ICON\_KEY, SMALL\_ICON,  
ACCELERATOR\_KEY, SELECTED\_KEY

# Poopćenje akcija

## ◆ Sučelje `Action`

- Koristeći oblikovni obrazac Promatrač (engl. *Observer*) nudi klijentima mogućnost pretplate na dojavu o promjenama pohranjenih informacija
- Objekti tipa `Action` su subjekti u oblikovnom obrascu Promatrač
- Promatrači su modelirani sučeljem `PropertyChangeListener`

# Poopćenje akcija

## ◆ Sučelje `Action`

```
Interface Action extends ActionListener {  
    public void actionPerformed(ActionEvent e);  
    public Object getValue(String key);  
    public void putValue(String key, Object value);  
    public void setEnabled(boolean b);  
    public boolean isEnabled();  
    public void addPropertyChangeListener(  
        PropertyChangeListener listener);  
    public void removePropertyChangeListener(  
        PropertyChangeListener listener);  
}
```

# Poopćenje akcija

- ◆ Apstraktni razred `AbstractAction` implementira svu potrebnu logiku (pamćenje podataka, rad s promatračima) osim metode `actionPerformed`
  - Stoga poslove modeliramo upravo temeljeći se na tom razredu



# Poopćenje akcija

- ◆ Najvažnije Swing komponente imaju konstruktore koji primaju referencu na `Action`
  - Inicijaliziraju se iz pohranjenih parametara (npr. `Action.NAME` za tekst)
  - Registriraju se kao listener za promjenu podataka i automatski mijenjaju tekst, omogućenost/onemogućenost, mnemonike, ...

# Povezivanje kratica i akcija

- ◆ Swing dodatno omogućava razdvajanje tipkovničkih kratica i akcija koje poziva na razini samih komponenti
- ◆ Koristi se spoj dviju mapa
  - `InputMap` ima ključeve tipkovničke kratice a vrijednosti nazive akcija; postoje tri vrste ove mape (sljedeći slide)
  - `ActionMap` ima ključeve nazive akcija a vrijednosti reference na same akcije

# Povezivanje kratica i akcija

- ◆ Svaka komponenta nudi pristup do tri ulazne mape (`InputMap` objekta)
- ◆ Koju mapu želimo, određujemo parametrom `JComponent.getInputMap(vrsta)`
  - `JComponent.WHEN_FOCUSED`  
(vrijedi kada komponenta ima fokus)
  - `JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT`  
(vrijedi kada je roditelj fokusirane komponente)
  - `JComponent.WHEN_IN_FOCUSED_WINDOW`  
(vrijedi kada je u prozoru koji je fokusiran)


# Povezivanje kratica i akcija

- ◆ Početne vrijednosti se pune pretpostavljenim vrijednostima
  - Nekako očekujemo da *cut*, *copy* i *paste* rade na svim tekstovnim komponentama
  - `DefaultEditorKit` sadrži standardizirane nazive za najčešće akcije
    - ◆ `DefaultEditorKit.copyAction`,  
`.cutAction`, ...

# Povezivanje kratica i akcija

## ◆ Primjer:

Bez argumenata:  
vraća mapu za  
WHEN\_FOCUSED



```
InputMap imap = jta.getInputMap();
ActionMap amap = jta.getActionMap();

// Prebaci "paste" na F2 tipku:
Object actionKey =
    imap.get(KeyStroke.getKeyStroke(KeyEvent.VK_V,
        KeyEvent.CTRL_DOWN_MASK));

imap.put(
    KeyStroke.getKeyStroke("control V"), "none");
imap.put(KeyStroke.getKeyStroke("F2"), actionKey);

// Zamijeni "copy" svojom akcijom:
Action dummy = new ...;
amap.put(DefaultEditorKit.copyAction, dummy);
```

# Povezivanje kratica i akcija

- ◆ Dodatni primjer: načini stvaranja **KeyStroke** objekata

```
// pritiskanje CTRL+ALT+V (treći argument je false!)
KeyStroke k1 = KeyStroke.getKeyStroke(
    KeyEvent.VK_V,
    InputEvent.CTRL_DOWN_MASK |
    InputEvent.ALT_DOWN_MASK,
    false);

// Pritiskanje CTRL+SHIFT+V; "released" za otpuštanje
KeyStroke k2 = KeyStroke.getKeyStroke(
    "control shift pressed V");

// Otpuštanje tipke F2 (treći argument je true!)
KeyStroke k3 = KeyStroke.getKeyStroke(
    KeyEvent.VK_F2, 0, true);
```

# Povezivanje kratica i akcija

- ◆ Više informacija dostupno je na:

How to Use Key Bindings

<https://docs.oracle.com/javase/tutorial/uiswing/misc/keybinding.html>

- ◆ Više informacija za izravnu uporabu KeyListener:

How to Write a Key Listener

<https://docs.oracle.com/javase/tutorial/uiswing/events/keylistener.html>

# Jednostavan uređivač teksta

- ◆ Idemo napraviti uređivač teksta
  - Koristimo komponentu `JTextArea`
- ◆ Dodati izbornike, toolbar, tipkovničke kratice
  - CTRL+F2 briše označeni dio teksta
  - CTRL+F3 radi toggle case-a na označenom dijelu teksta
- ◆ Proučiti:
  - `Document`, `Caret`



# Internacionalizacija

- ♦ Želimo mogućnost internacionalizacije (famozni i18n)
  - Pisati GUI koji će se potom lagano prikazivati u odabranim jezicima
- ♦ Java ima podršku:
  - Razred `ResourceBundle`
  - Ideja: za svaki jezik imamo po jednu lokalizacijsku datoteku u kojoj su podatci oblika *ključ = prijevod*
  - Tekstove po potrebi vučemo iz bundle-a

# Internacionalizacija

## ◆ Primjer:

```
Locale locale = Locale.forLanguageTag("en");  
ResourceBundle bundle =  
    ResourceBundle.getBundle("hr.fer.nazivi", locale);  
String ime = bundle.getString("ime");
```

```
hr/fer/nazivi_en.properties  
ime = User name
```

```
hr/fer/nazivi_de.properties  
ime = Benutzername
```

# Internacionalizacija

- ♦ Važno: od Jave 9 `ResourceBundle` očekuje da su `x.properties` datoteke pohranjene uporabom kodne stranice UTF-8 pa napravite tako (prije je default bio ISO 8859-1 uz escapeanje nekodirljivih simbola)

# Internacionalizacija

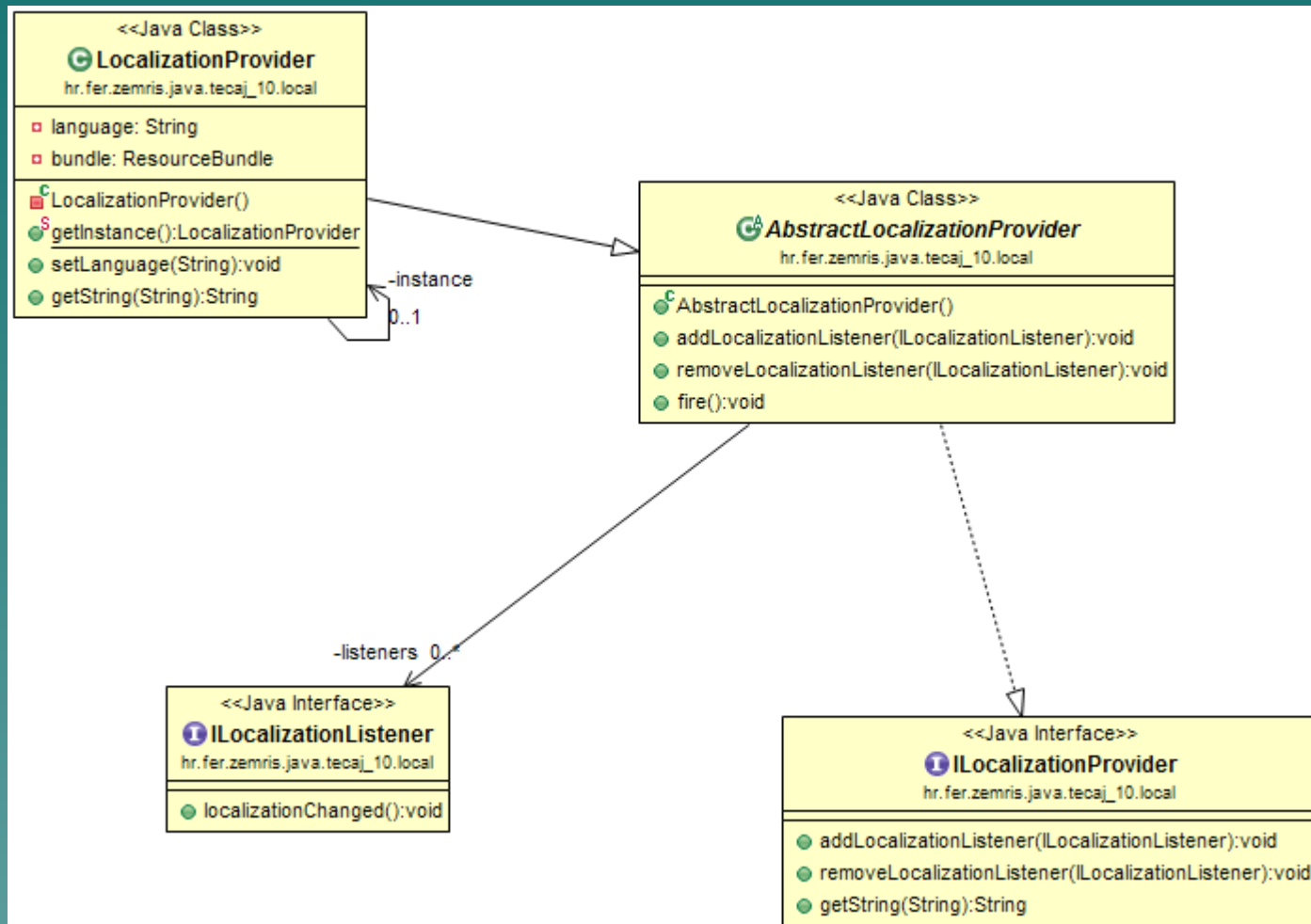
- ♦ Želimo mogućnost internacionalizacije (takozvani *i18n*)
  - Još bismo, dakako, željeli da korisnik može lokalizaciju promijeniti dinamički, dok je aplikacija otvorena, i da se sve prekonfigurira dinamički
  - Za to trebamo ipak još malo pisati...

# Internacionalizacija

- ◆ Dinamička promjena lokalizacije
  - Netko treba znati koji je trenutni jezik
  - Sve komponente koje su lokalizirane trebaju imati mogućnost prijaviti se za dojavu informacija o promjeni trenutnog jezika kako bi mogle podesiti nazive koje prikazuju

# Internacionalizacija

## ◆ Idemo napraviti sljedeće:



# Pojašnjenje

## ◆ `ILocalizationProvider`

- Sučelje prema objektu koji nudi uslugu dohvaćanja lokaliziranih tekstova
- Subjekt u oblikovnom obrascu  
Promatrač za dojavu promjene trenutnog jezika
- (dopuna) treba još dodati metodu:  
`String getCurrentLanguage()` ;

# Pojašnjenje

## ◆ `ILocalizationListener`

- Sučelje kojim su definirani promatrači koji žele biti obaviješteni da je došlo do promjene trenutnog jezika



# Pojašnjenje

- ◆ **AbstractLocalizationProvider**
  - Apstraktni razred koji implementira pamćenje zainteresiranih promatrača, njihovu prijavu i odjavu te slanje obavijesti da je došlo do promjene jezika

# Pojašnjenje

## ◆ `LocalizationProvider`

- Konkretna implementacija jednog pružatelja usluge dohvata lokaliziranih tekstova
- Ostvaren oblikovnim obrascem  
Jedinstveni objekt

# Oblikovni obrazac: Jedinstveni objekt

- ◆ Rješava problem gdje ne smije postojati više od jednog primjerka nekog razreda
- ◆ Tipično rješenje:
  - Definiramo razred
  - Definiramo mu konstruktor, ali privatni!
  - Definiramo privatnu statičku varijablu koja čuva referencu na jedan primjerak tog razreda
  - Definiramo javni statički getter

# Oblikovni obrazac: Jedinstveni objekt

- ◆ Rješava problem gdje ne smije postojati više od jednog primjerka nekog razreda
- ◆ Tipično rješenje:
  - ...
  - Moguć je i scenarij gdje javni statički getter pri prvom pozivu inicijalizira tu privatnu statičku varijablu tako da se primjerak uopće ne stvara ako ga nitko ne treba: *lijena inicijalizacija*

# Oblikovni obrazac: Jedinstveni objekt

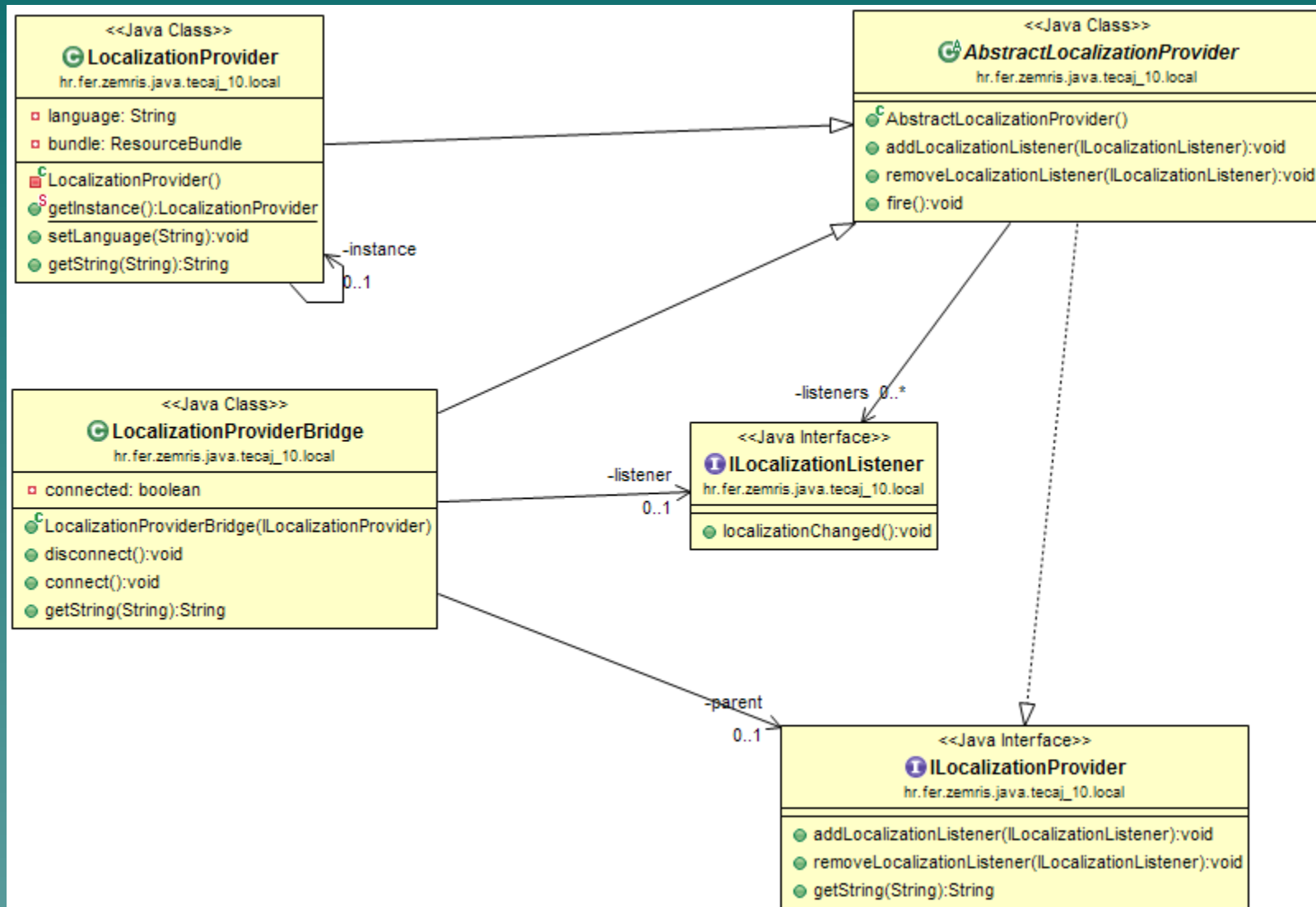
- ◆ engl. *Singleton*
- ◆ Rješava problem gdje ne smije postojati više od jednog primjerka nekog razreda

Singleton	
-	<u>singleton : Singleton</u>
-	Singleton()
+	<u>getInstance() : Singleton</u>

# Internacionalizacija - nastavak

- ◆ Imamo problem s `JFrame`-ovima
  - Naš `LocalizationProvider` drži reference na labele, gumbe i slično, koji pak drže reference na prozor u kojem su, što znači da i nakon zatvaranja prozora garbage collector nikada neće uspjeti pokupiti taj prozor
- velik problem ako aplikacija otvara i zatvara više prozora

# Internacionalizacija - nastavak



◆ Dodajmo još "bridge":

# Pojašnjenje

## ◆ `LocalizationProviderBridge`

- Posrednik (proxy) između izvornog objekta koji čuva lokalizirane tekstova i komponenata koje ih trebaju
- Na prethodnom dijagramu sadrži primjerak ugniježđenog anonimnog razreda koji je promatrač i koji se može prijaviti/odjaviti na izvorni objekt



# Pojašnjenje

## ◆ `LocalizationProviderBridge`

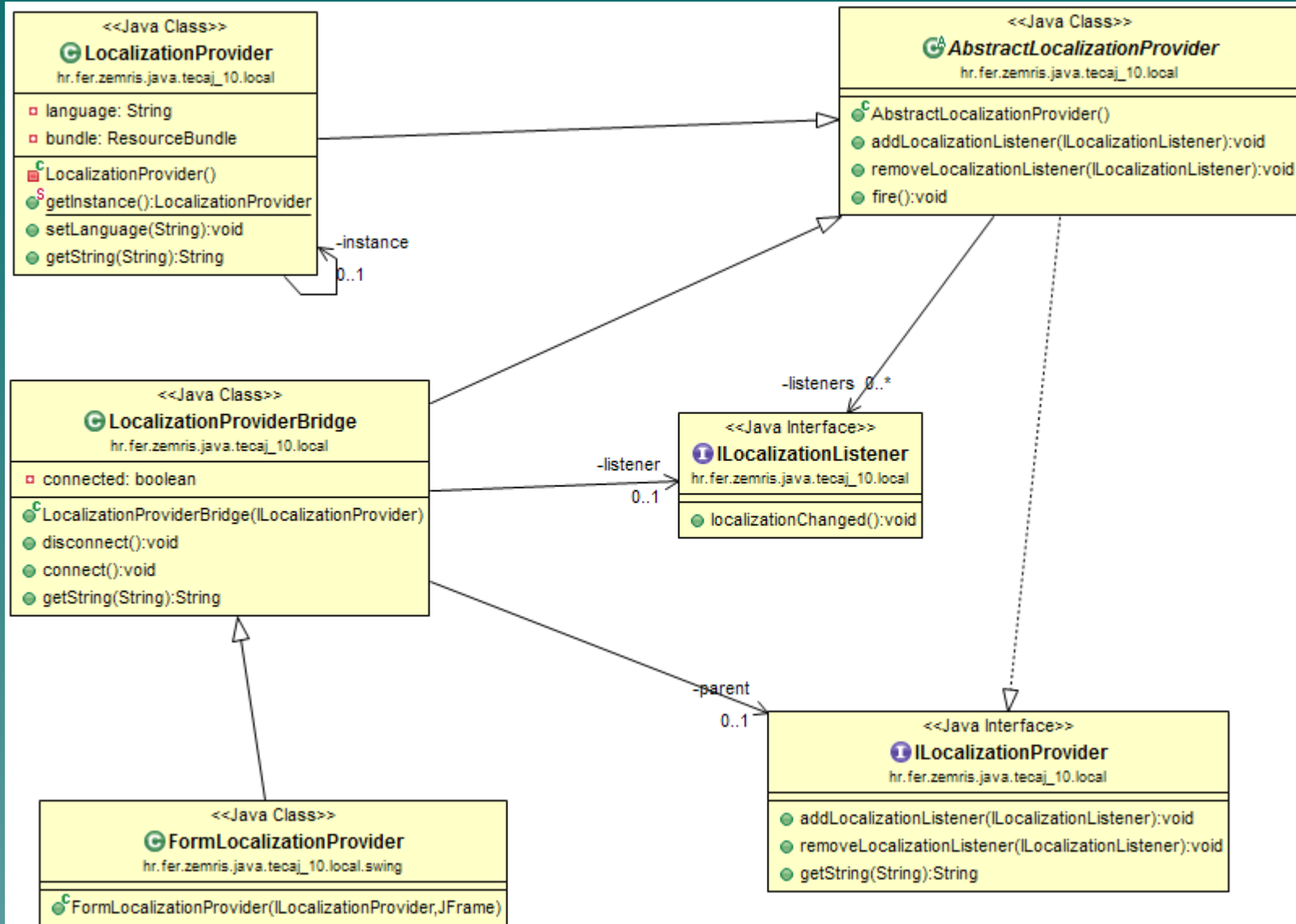
- Prosljeđuje događaje svojim promatračima
- Ideja je da se komponente prijave na njega a on (kada je `connected`) na izvorni objekt
- Kada se roditelj komponenata uništava, dovoljno je da se ovaj objekt odjavi od izvornog objekta i nema problema s "curenjem" memorije

# Pojašnjenje

## ◆ `LocalizationProviderBridge`

- Uz pretpostavku da je dodan `getCurrentLanguage()`, u trenutku `disconnect()` treba zapamtiti trenutni jezik (u pomoćnoj privatnoj članskoj varijabli) kako bi u trenutku novog `connect()` provjerio je li došlo do promjene jezika (pa obavijestio svoje promatrače i ažurirao jezik koji pamti)

# Internacionalizacija - nastavak

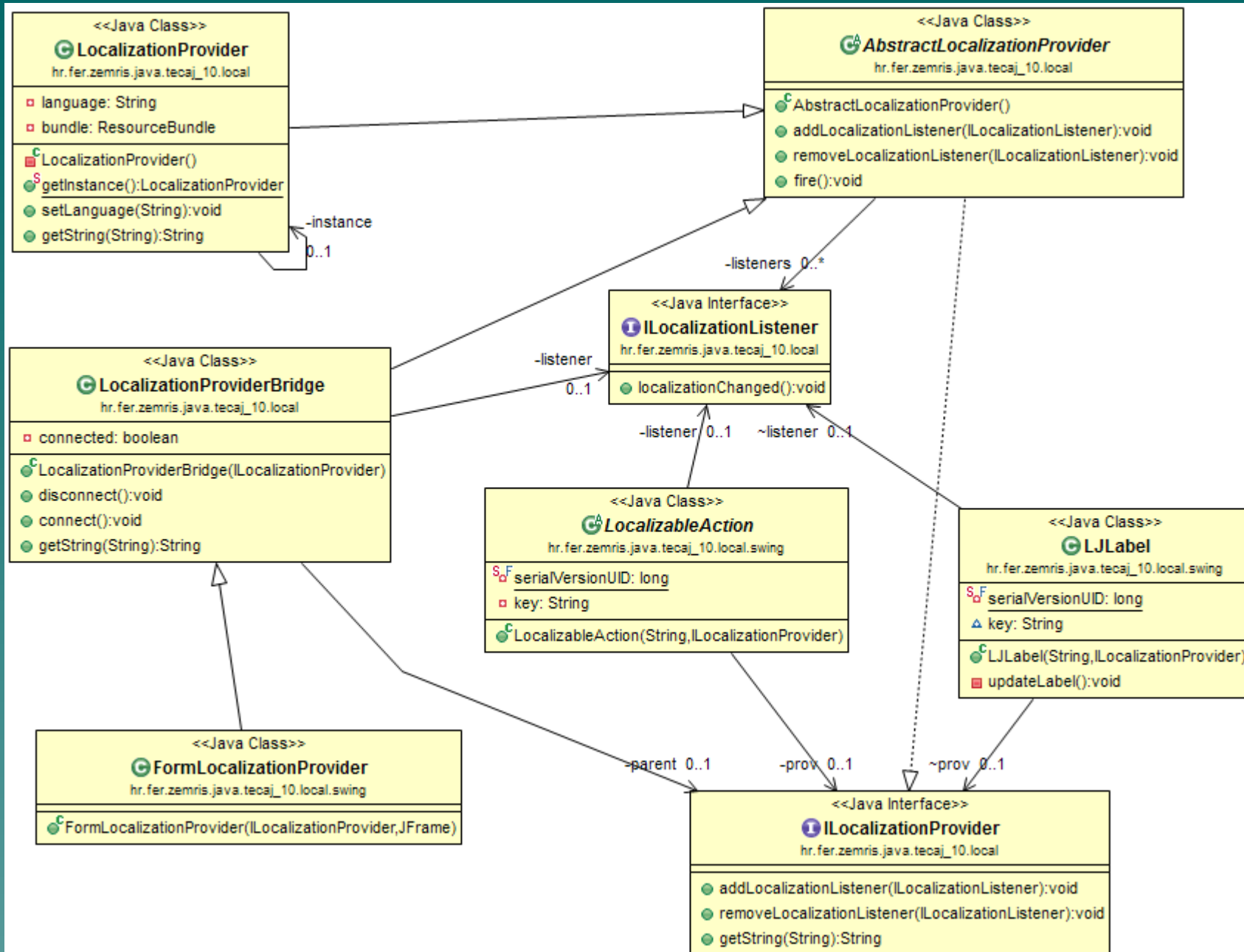


◆ Te podršku za forme:

# Pojašnjenje

## ◆ `FormLocalizationProvider`

- Implementacija mosta koja u konstruktoru dobiva referencu na prozor te interno nad njim stvara `windowClose`-promatrača koji automatski poziva `disconnect()` kada se prozor zatvara
- Time ovaj kod nije potrebno duplicirati u svim prozorima koji trebaju ovu uslugu



# Pojašnjenje

## ◆ LocalizableAction

- Apstraktna akcija koja u konstruktoru prima ključ i referencu na objekt za lokalizaciju
- Trenutno *ime* postavlja na vrijednost lokaliziranog ključa
- Registrira anonimnog promatrača na lokalizacijski objekt i pri promjeni jezika automatski dohvaća i postavlja novi prijevod kao *ime*

# Internacionalizacija - nastavak

- ◆ Implementacijom `LocalizableAction`:
  - Besplatno smo dobili funkcionalnu lokalizaciju svih komponenti koje se inicijaliziraju iz akcija (izbornici, gumbi)
- ◆ Međutim, ima stvari koje još nismo riješili
  - Npr. `TableModel` između ostaloga definira metode za dohvat naziva stupaca → njih bismo možda htjeli lokalizirati!

