Java tečaj

7. dio (b)

Swing (2)

Teme

- Look&Feel
- Izrada uređivača teksta, ili kako se radi s:
  - Izbornicima,
  - Toolbarima,
  - Tipkovničkim kraticama,
  - Dokumentima, …
  - Lokalizacija (Internacionalizacija, i18n)

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`.

Skinability (izmjena kože?)

- Swing aplikacije podržavaju definiranje načina prikaza tipičnih komponenata (*"htio bih da moja Java aplikacija izgleda kao Windows program"*)
- Standardne komponente (`JLabel`, `JButton`, …) stoga se same ne crtaju već to povjeravaju (delegiraju) iz svoje metode `paint(g)` drugom objektu koji je primjerak razreda izvedenog iz `ComponentUI`

Skinability (izmjena kože?)

- Razred `ComponentUI` deklarira metode za izračun dimenzija komponente (i još neke druge) te:

```
void update(Graphics g, JComponent c);
void paint(Graphics g, JComponent c);
void installUI(JComponent c);
void uninstallUI(JComponent c);
```

- Razred `ComponentUI` dakle zna kako crtati jednu specifičnu vrstu komponente (on je *strategija*!)

Skinability (izmjena kože?)

- `paint(...)` od komponente poziva svoj `paintComponent(g)` pa `paintBorder(g)`
- `paintComponent(g)` nad ui-delegatom poziva metodu `update(g, this)`
- `update(g,c)` oboji pozadinu (ako je `opaque=true`) i zove svoj `paint(...)`

Skinability (izmjena kože?)

- Svaka "tema" (jezikom jave: svaki *Look&Feel*) za svaku vrstu standardne komponente nudi razred koji je ui-delegat za tu vrstu komponente
- Java standardno dolazi s tri “teme”: Windows, višeplatformska (Metal) te unixoidna (Motif)

Skinability (izmjena kože?)

- UI-delegati za `JButton` su razredi:
  - MetalButtonUI (višeplatformski)
  - WindowsButtonUI (windowsoliki)
  - MotifButtonUI (unixoidni)
  - ...
- Konkretan delegat možemo stvoriti statičkom metodom `createUI(c)` nar razredom delegatom
- Delegata komponenti postavljamo sa `c.setUI(delegat)`

Skinability (izmjena kože?)

- Primjer:

```
JButton b1 = new JButton("OK");
JButton b2 = new JButton("Cancel");
ComponentUI ui1 =
            WindowsButtonUI.createUI(b1);
ComponentUI ui2 =
            WindowsButtonUI.createUI(b2);
b1.setUI(ui1);
b2.setUI(ui2);
```

Skinability (izmjena kože?)

- Specifični delegati za pojedine vrste komponenata izvedeni su iz razreda `ComponentUI`; tako imamo:
  - `ButtonUI`
  - `LabelUI`
  - `ListUI`
  - `TableUI`
  - `...`

Skinability (izmjena kože?)

- "Tema" je kolekcija razreda koji omogućavaju izradu delegata za standardne komponente (implementacija `ButtonUI`, `LabelUI`, ...) plus zajednički "ulazni" razred koji čuva i nudi zajednička svojstva teme te nudi metodu za dohvat primjeraka delegata za pojedine komponente
- Modelirana razredom `LookAndFeel`

Skinability (izmjena kože?)

- Razred `UIManager` pamti trenutno odabranu temu te nudi mogućnost dohvata informacija o svim instaliranim temama
- Standardne komponente, kada se stvaraju, pitaju `UIManager` za trenutni `LookAndFeel` i dohvaćaju delegat koji će koristiti

Skinability (izmjena kože?)

```
for(LookAndFeelInfo lafi :
            UIManager.getInstalledLookAndFeels()) {
    System.out.println(lafi.getClassName());
}
```

```
javax.swing.plaf.metal.MetalLookAndFeel
javax.swing.plaf.nimbus.NimbusLookAndFeel
com.sun.java.swing.plaf.motif.MotifLookAndFeel
com.sun.java.swing.plaf.windows.WindowsLookAndFeel
com.sun.java.swing.plaf.windows.WindowsClassicLookAndFeel
```

Skinability (izmjena kože?)

- Temu koju aplikacija treba koristiti možemo podesiti statičkom metodom razreda `UIManager`:

```
UIManager.setLookAndFeel(LookAndFeel laf);
UIManager.setLookAndFeel(String classname);
```

- Ako je promjena napravljena nakon što su komponente stvorene, nad svakom se mora pozvati `c.updateUI();`
  ona pita `UIManager` za novi delegat i instalira ga sa `setUI(...)`

Skinability (izmjena kože?)

- Ako imamo stvorenu hijerarhiju komponenata, možemo iskoristiti metodu `SwingUtilities.updateComponentTreeUI(comp);` koja će protrčati kroz sve komponente hijerarhije i pozivati `updateUI()` nad njima

Skinability (izmjena kože?)

- `UIManager` nudi metodu `UIManager.getSystemLookAndFeelClassName();` koja vraća nayiv razreda za `LookAndFeel` koji odgovara operacijskom sustavu na kojem je program pokrenut (ili višeplatformski `LookAndFeel` ako takav specifični ne postoji); ovaj posljednji direktno vraća `UIManager.getCrossPlatformLookAndFeelClassName()`

UT - organizacija kôda

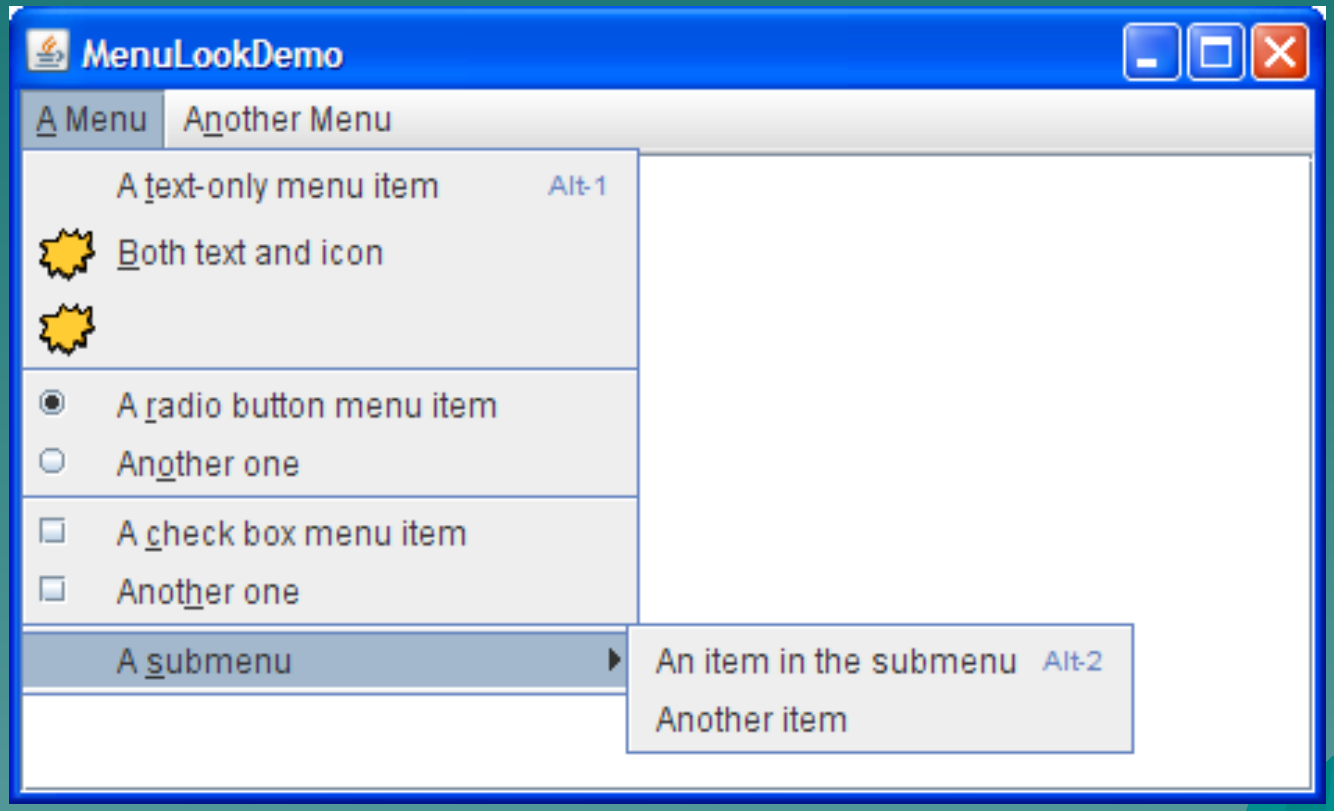
- Uređivač teksta možemo napraviti na više načina
  - Jedna komponenta tipa `JTextArea`,
  - Jedna komponenta tipa `JTextPane`,
  - Više uređivača unutar `JTabbedPane`-a
- Želimo:
  - Funkionalan GUI!

Pregled potrebnoga...

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

Izbornici

- Izbornici:

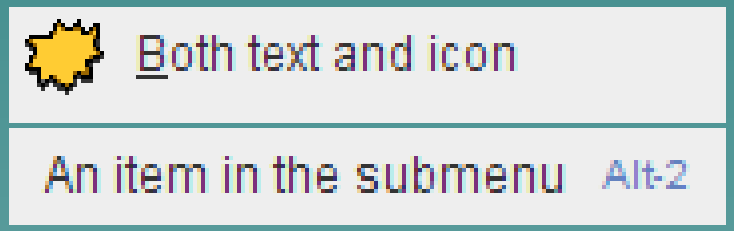


Izbornici

- 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

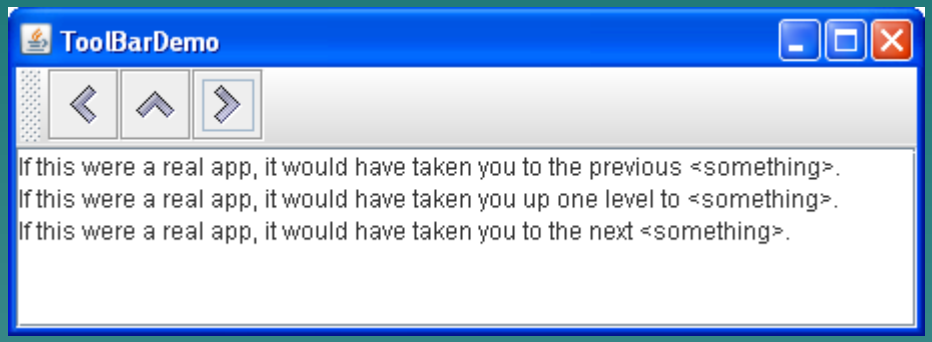
Izbornici

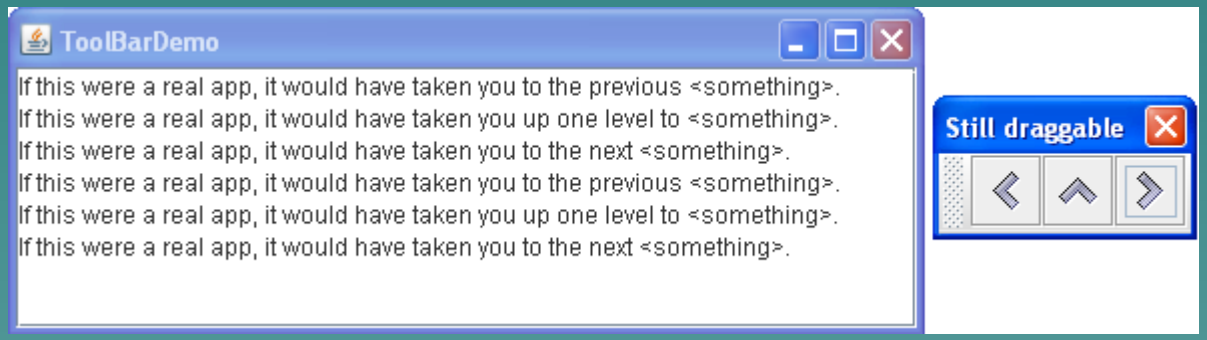
- Dodavanje kôda:
  - Kod koji želimo pokrenuti kada se klikne na konkretnu stavku (`JMenuItem`) toj stavki možemo dodati uporabom metode `addActionListener(…)`
  - Stavki možemo definirati naziv, mnemonik te akceleratorsku kombinaciju



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*

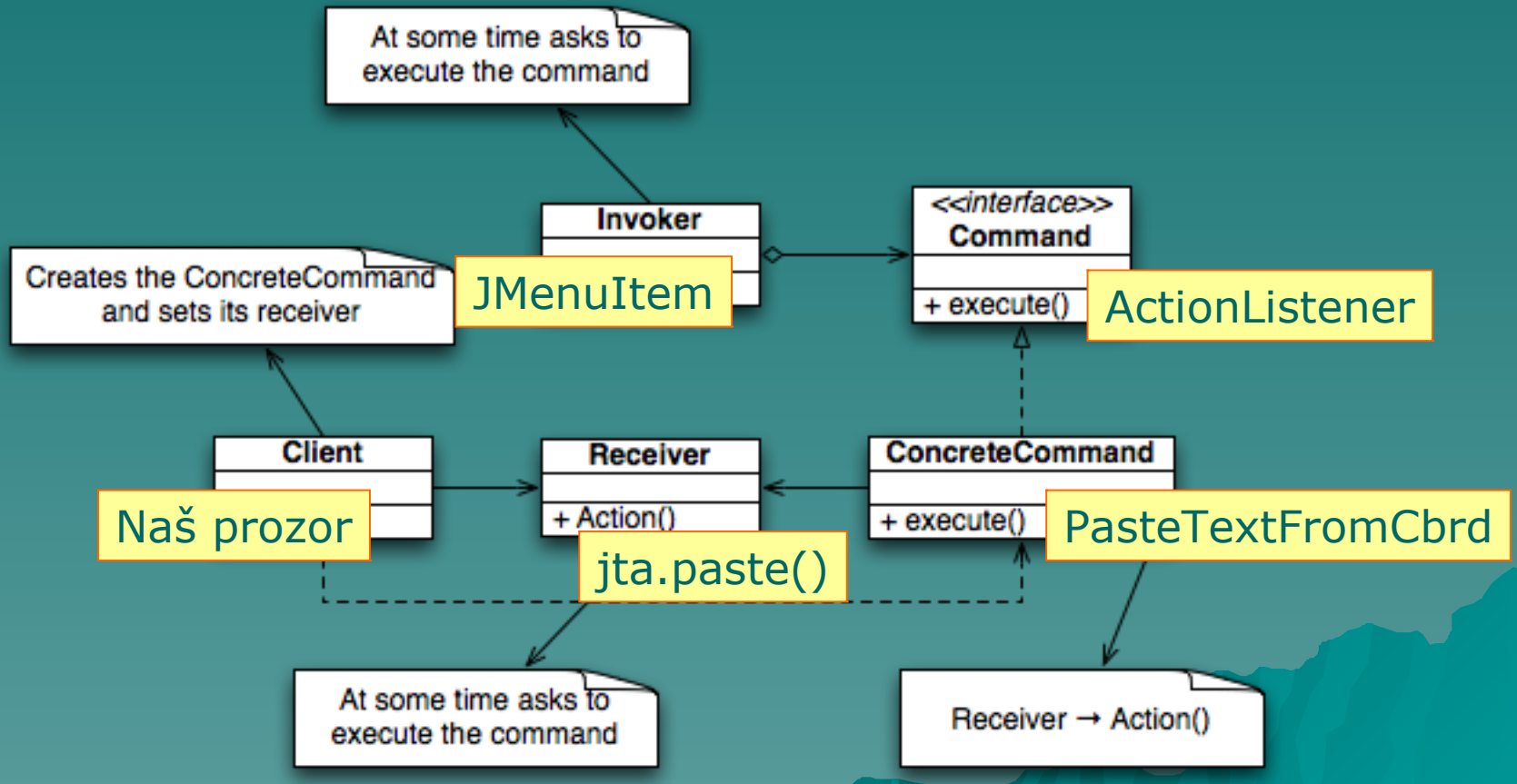
- Naredba (engl. *Command*)
  - Ideja je razdvojiti implementacijske detalje poslova od pozivatelja: nužna uporaba apstrakcije
  - Pozivatelj poslove vidi preko generičkog sučelja
  - Pristup omogućava implementaciju prikaza raspoloživih poslova na generički način (primjerice, možemo graditi izborničke 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
  - 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
  - `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 nazive za najčešće akcije
    - DefaultEditorKit.copyAction, .cutAction, …

Povezivanje kratica i akcija

- Primjer:

```
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);
```

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

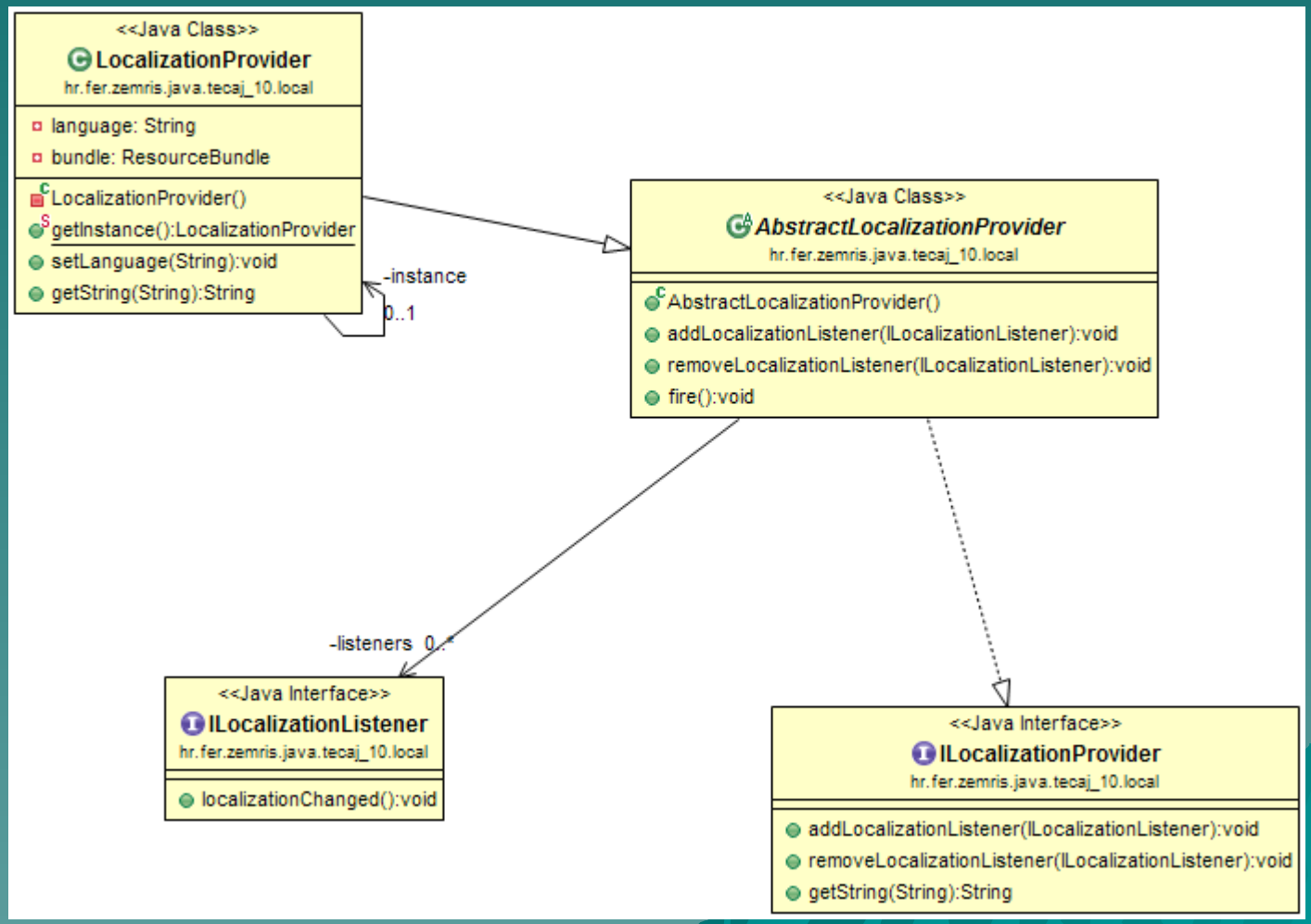
- Želimo mogućnost internacionalizacije (takozvani *i18n*)
  - Još bismo, dakako, željeli da korisnik može lokalizaciju promijeniti dinamički, dok je aplikacija otvorena, i da se sve prekonfigurira dinamički
  - Za to trebamo ipak još malo pisati…

Internacionalizacija

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

Internacionalizacija

- Idemo napraviti sljedeće:



Oblikovni obrazac: Jedinstveni objekt

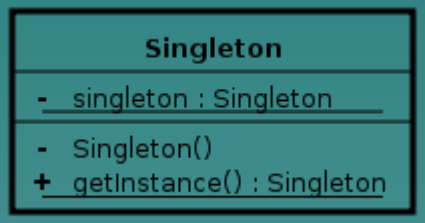
- Rješava problem gdje ne smije postojati više od jednog primjerka nekog razreda
- Tipično rješenje:
  - Definiramo razred
  - Definiramo mu konstruktor, ali privatni!
  - Definiramo privatnu statičku varijablu koja čuva referencu na jedan primjerak tog razreda
  - Definiramo javni statički getter

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



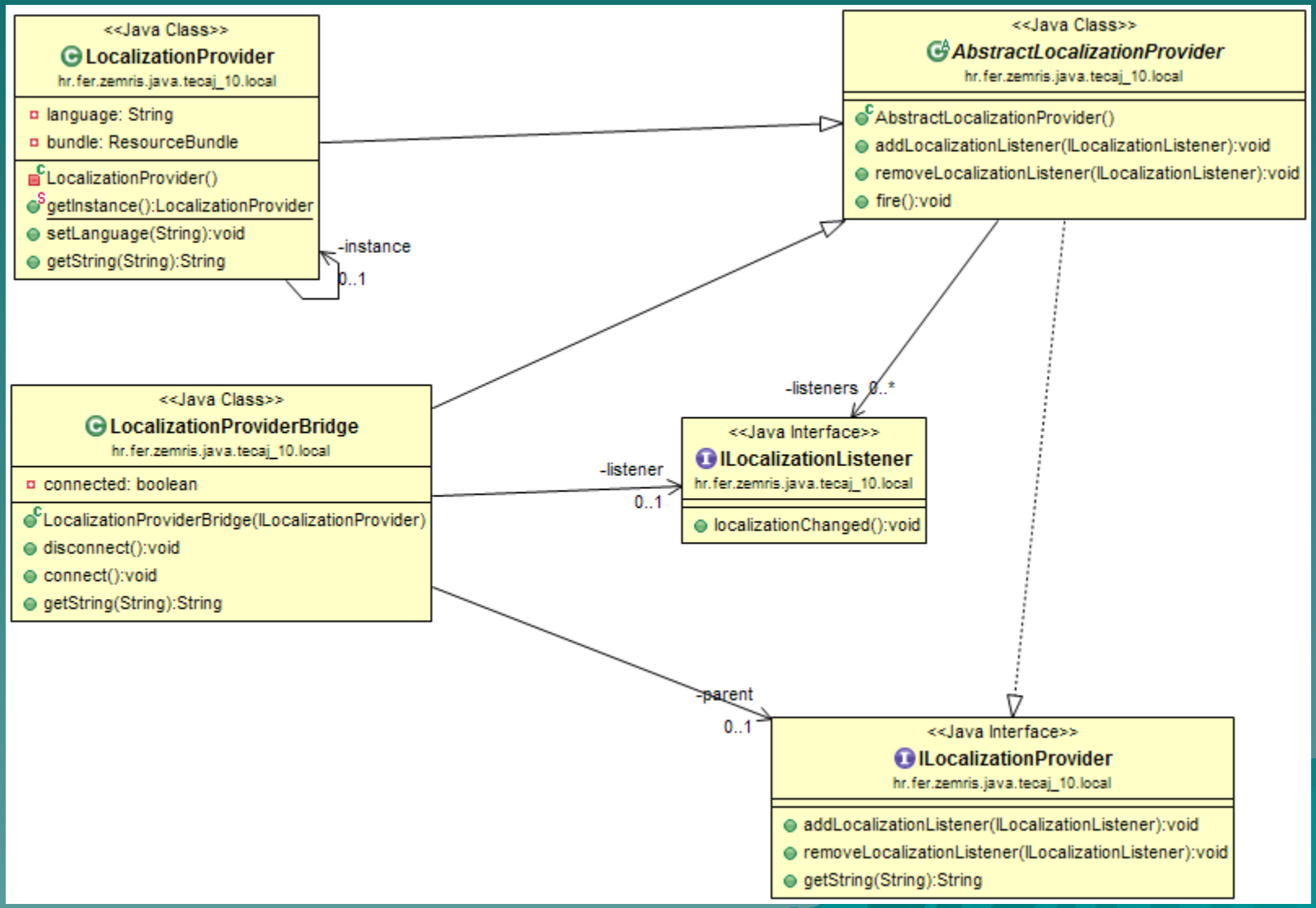
Internacionalizacija - nastavak

- 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

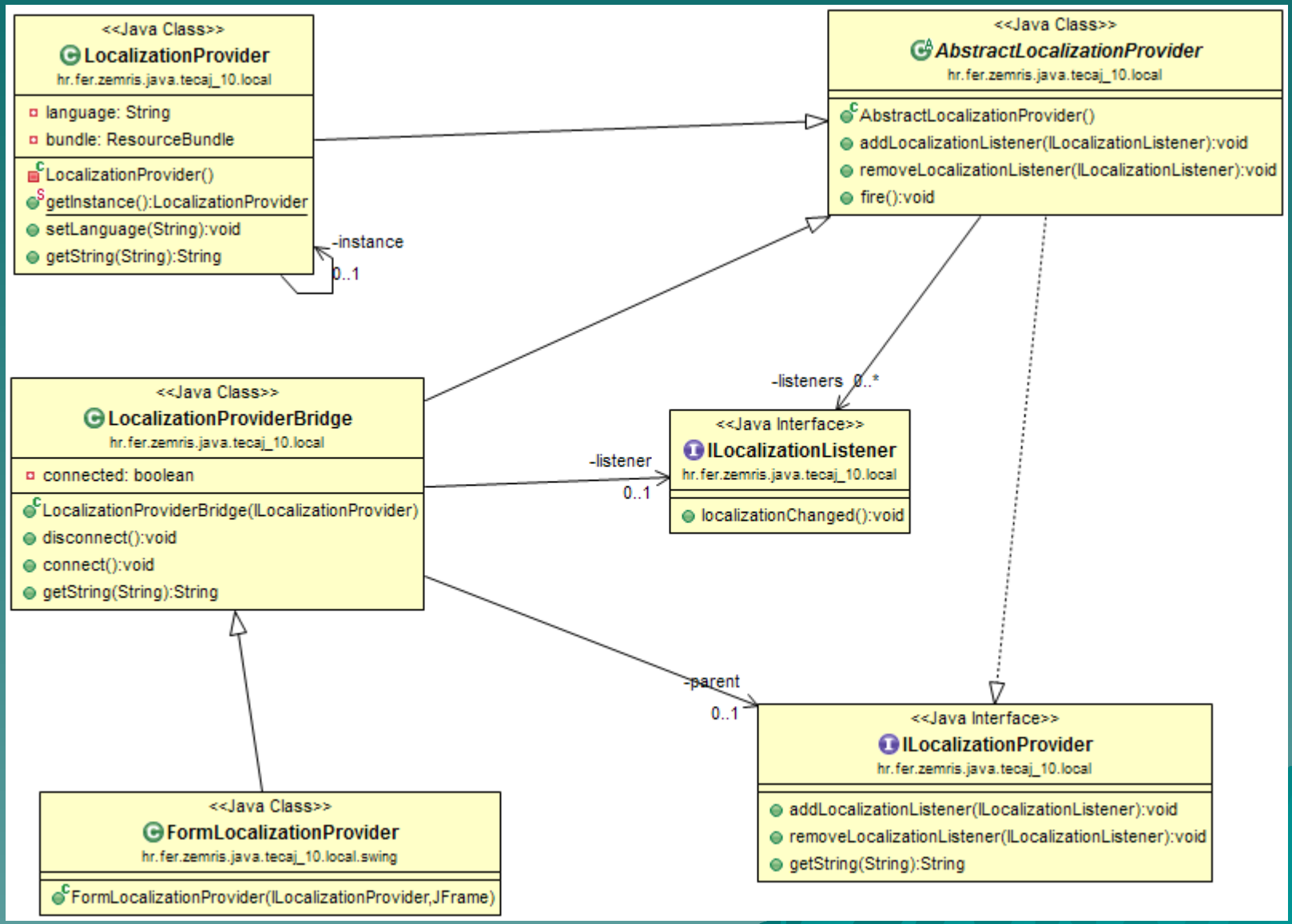
Internacionalizacija - nastavak

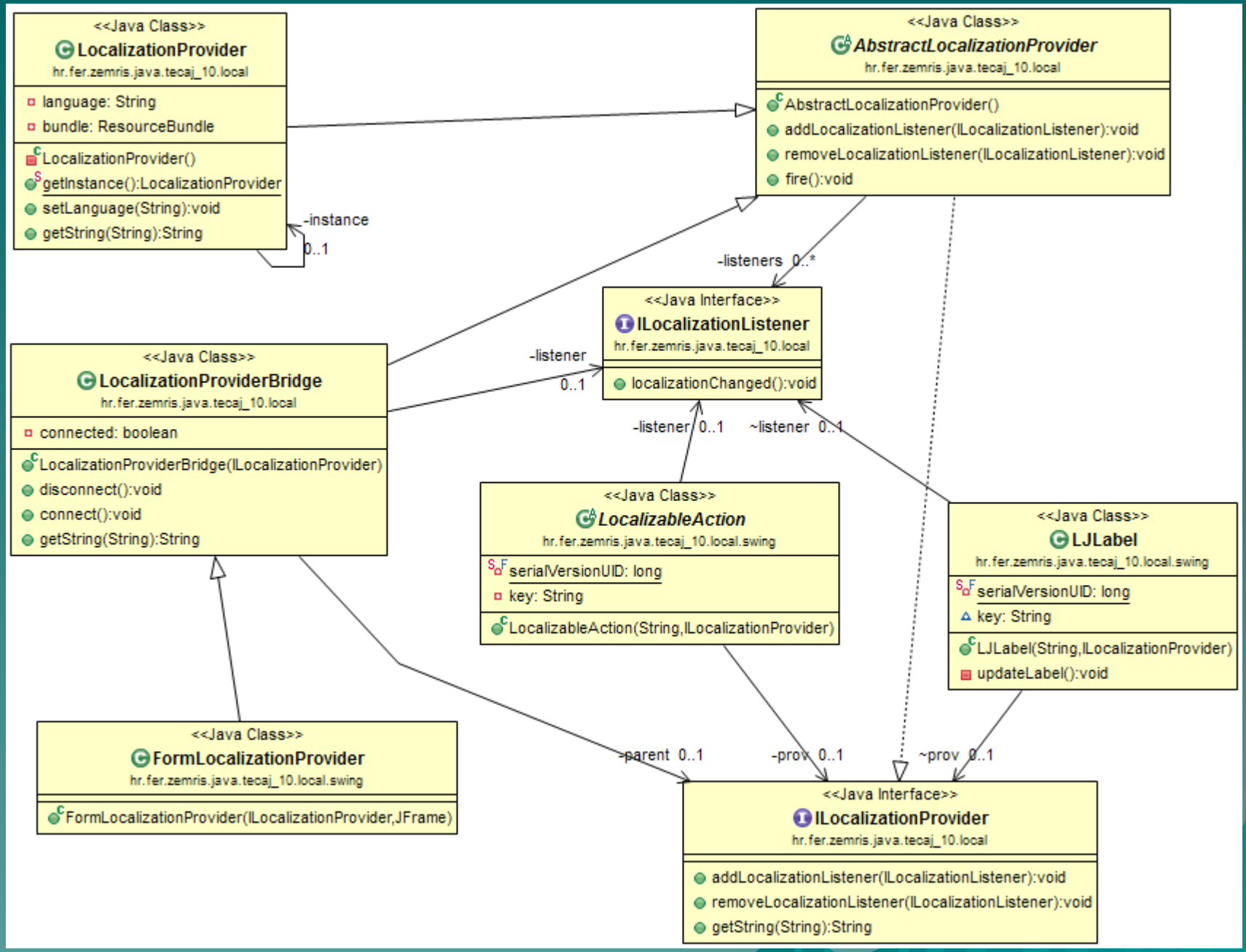
- Dodajmo još "bridge":



Internacionalizacija - nastavak

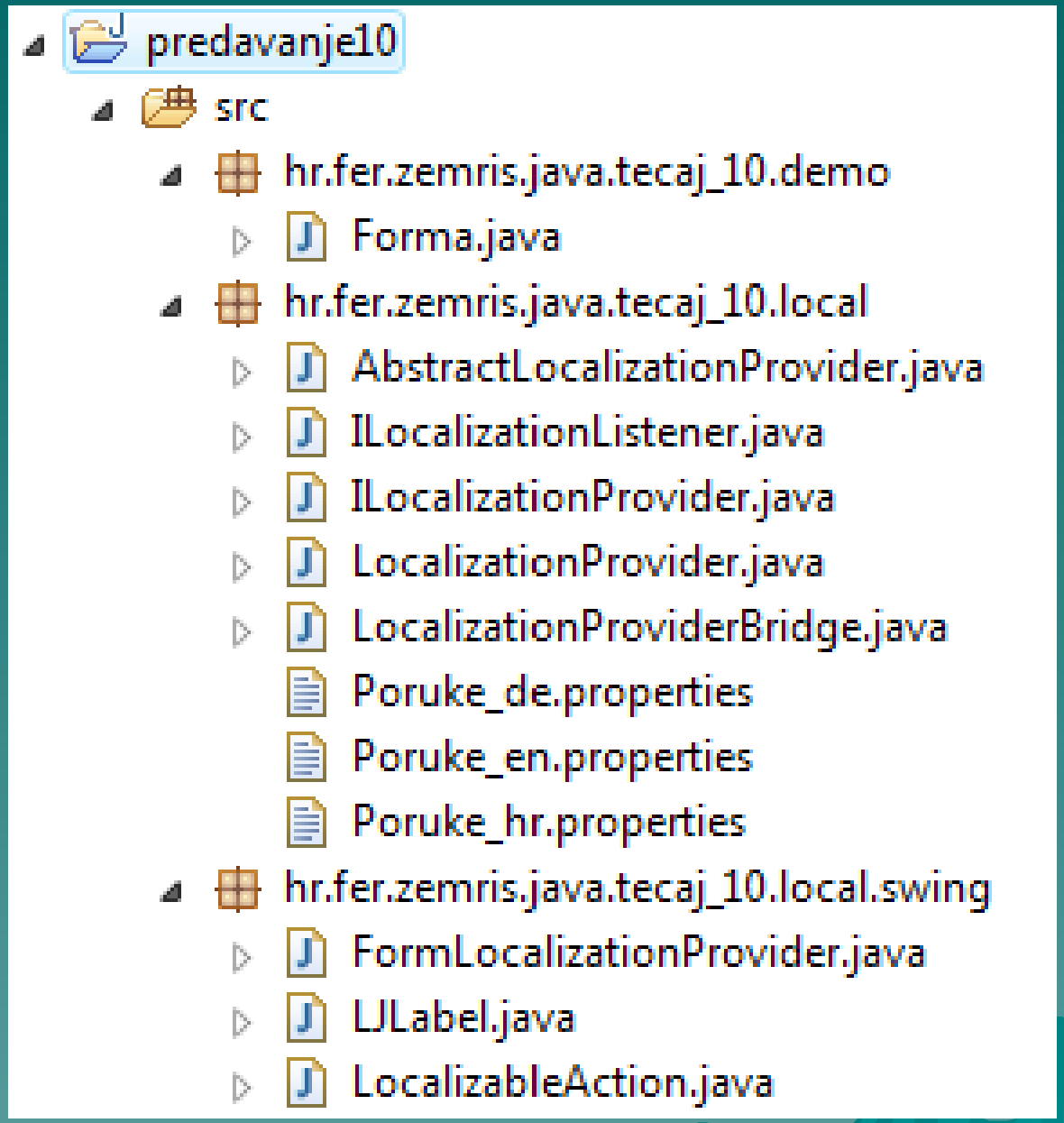
- Te podršku za forme:




<<Java Class>>
LocalizationProvider
hr.fer.zemris.java.tecaj_10.local
language: String
bundle: ResourceBundle
LocalizationProvider()
getInstance():LocalizationProvider
setLanguage(String):void
getString(String):String
-instance
0..1
<<Java Class>>
AbstractLocalizationProvider
hr.fer.zemris.java.tecaj_10.local
AbstractLocalizationProvider()
addLocalizationListener(ILocalizationListener):void
removeLocalizationListener(ILocalizationListener):void
fire():void
-listeners 0..*
<<Java Interface>>
ILocalizationListener
hr.fer.zemris.java.tecaj_10.local
localizationChanged():void
-listener
0..1
<<Java Class>>
LocalizationProviderBridge
hr.fer.zemris.java.tecaj_10.local
connected: boolean
LocalizationProviderBridge(ILocalizationProvider)
disconnect():void
connect():void
getString(String):String
-listener 0..1
~listener 0..1
<<Java Class>>
LocalizableAction
hr.fer.zemris.java.tecaj_10.local.swing
serialVersionUID: long
key: String
LocalizableAction(String,ILocalizationProvider)
<<Java Class>>
LJLabel
hr.fer.zemris.java.tecaj_10.local.swing
serialVersionUID: long
key: String
LJLabel(String,ILocalizationProvider)
updateLabel():void
<<Java Class>>
FormLocalizationProvider
hr.fer.zemris.java.tecaj_10.local.swing
FormLocalizationProvider(ILocalizationProvider,JFrame)
-parent 0..1
-prov 0..1
~prov 0..1
<<Java Interface>>
ILocalizationProvider
hr.fer.zemris.java.tecaj_10.local
addLocalizationListener(ILocalizationListener):void
removeLocalizationListener(ILocalizationListener):void
getString(String):String

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!


predavanje10
src
hr.fer.zemris.java.tecaj_10.demo
Forma.java
hr.fer.zemris.java.tecaj_10.local
AbstractLocalizationProvider.java
ILocalizationListener.java
ILocalizationProvider.java
LocalizationProvider.java
LocalizationProviderBridge.java
Poruke_de.properties
Poruke_en.properties
Poruke_hr.properties
hr.fer.zemris.java.tecaj_10.local.swing
FormLocalizationProvider.java
LJLabel.java
LocalizableAction.java