Java tečaj

7. dio (b)

Swing (2)

Teme

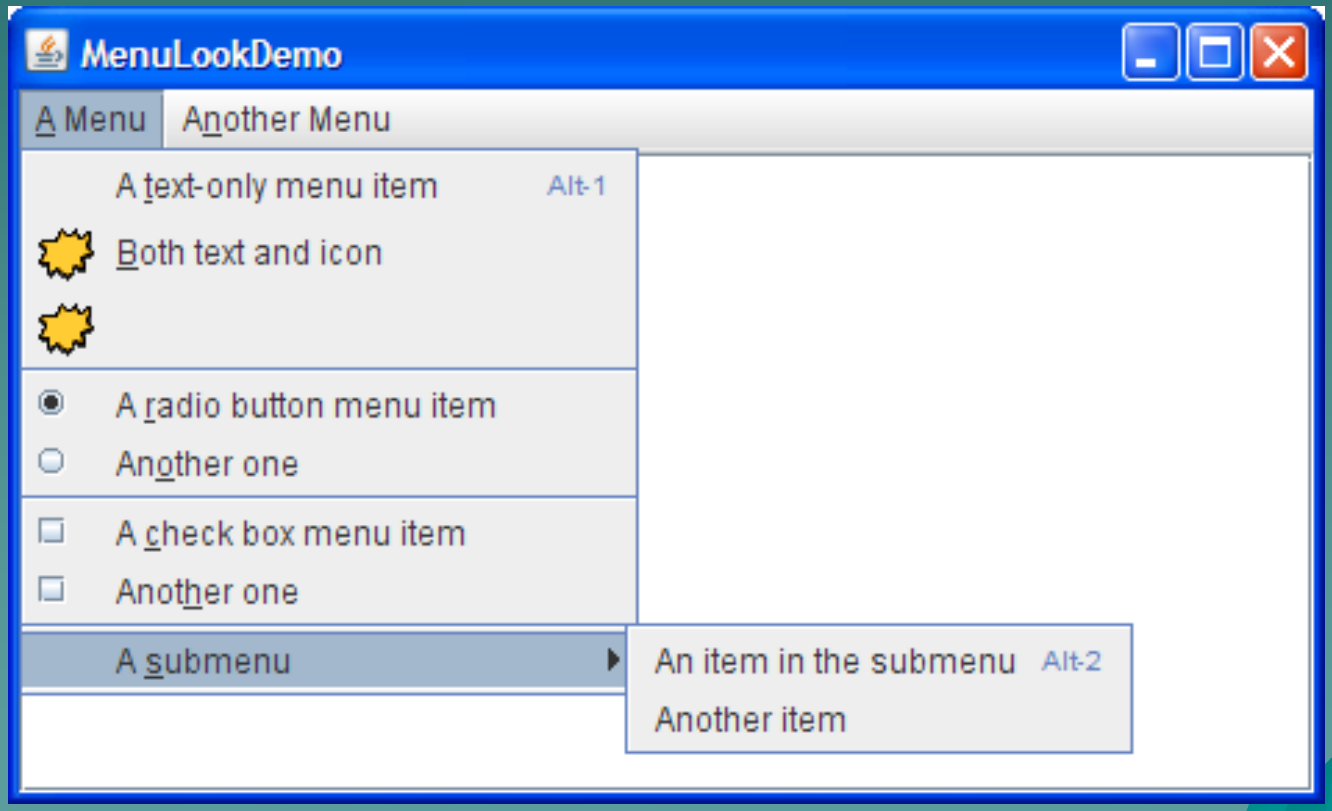
- Izrada uređivača teksta, ili kako se radi s:
  - Izbornicima,
  - Toolbarima,
  - Tipkovničkim kraticama,
  - Dokumentima, …
  - Lokalizacija (Internacionalizacija, i18n)
- Izrada vlastitih grafičkih komponenti

Teme: UT - organizacija kôda

- Postoji uređivač teksta
  - Jedna komponenta tipa **`JTextArea`**,
  - Jedna komponenta tipa **`JTextPane`**,
  - Više uređivača unutar **`JTabbedPane`**-a
- Želimo:
  - Funkionalan GUI!

Teme: UT - organizacija kôda

- Izbornici:

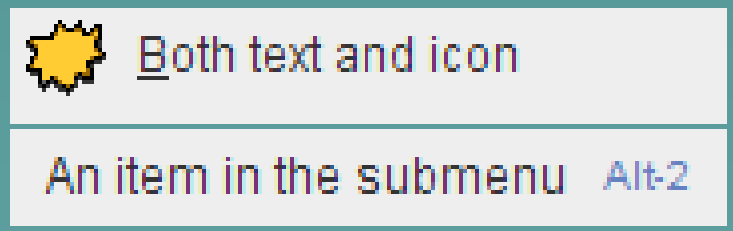


Teme: UT - organizacija kôda

- 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

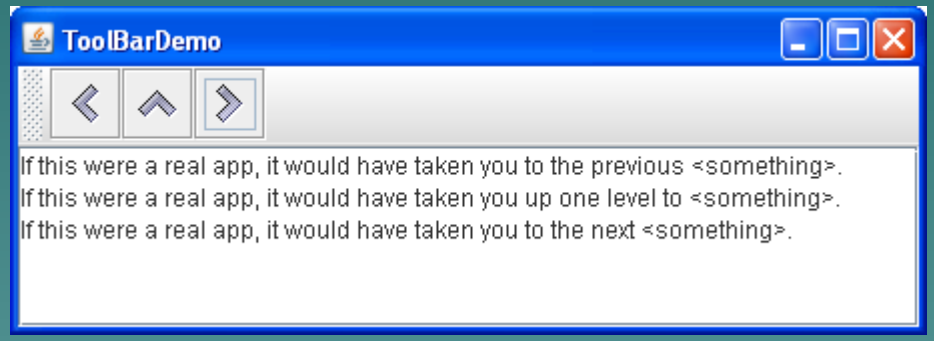
Teme: UT - organizacija kôda

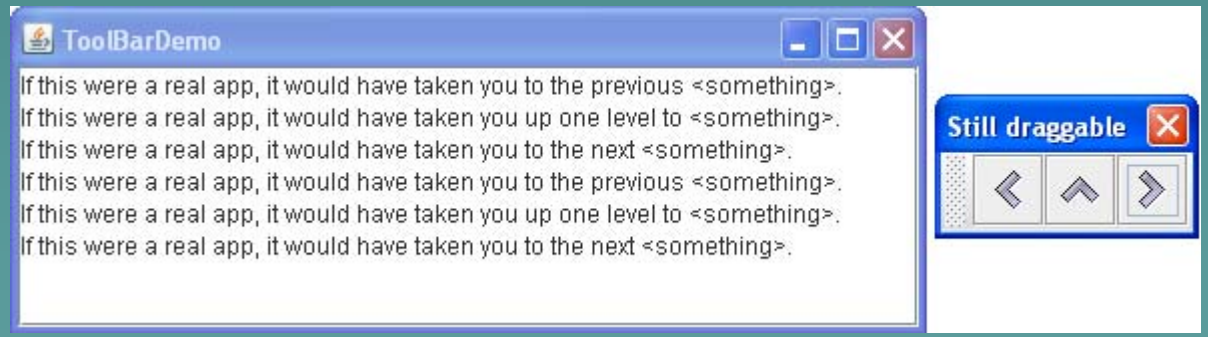
- Dodavanje kôda:
  - Kod koji želimo pokrenuti kada se klikne na konkretnu stavku (`JMenuItem`) toj stavki možemo dodati uporabom metode `addActionListener(…)`
  - Stavki možemo definirati naziv, mnemonik te akceleratorsku kombinaciju



Teme: UT - organizacija kôda

- Toolbar:





Teme: UT - organizacija kôda

- Toolbar se gradi 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 toolbar možemo dodavati svašta: gumbe, separatore, **`JTextField`**-ove itd.

Teme: UT - organizacija kôda

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

Teme: UT - organizacija kôda

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

Teme: UT – problem pristupa

- Masovna redundancija!
  - Na više načina možemo pokretati konceptualno iste poslove
  - Ako loše organiziramo kod, 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*

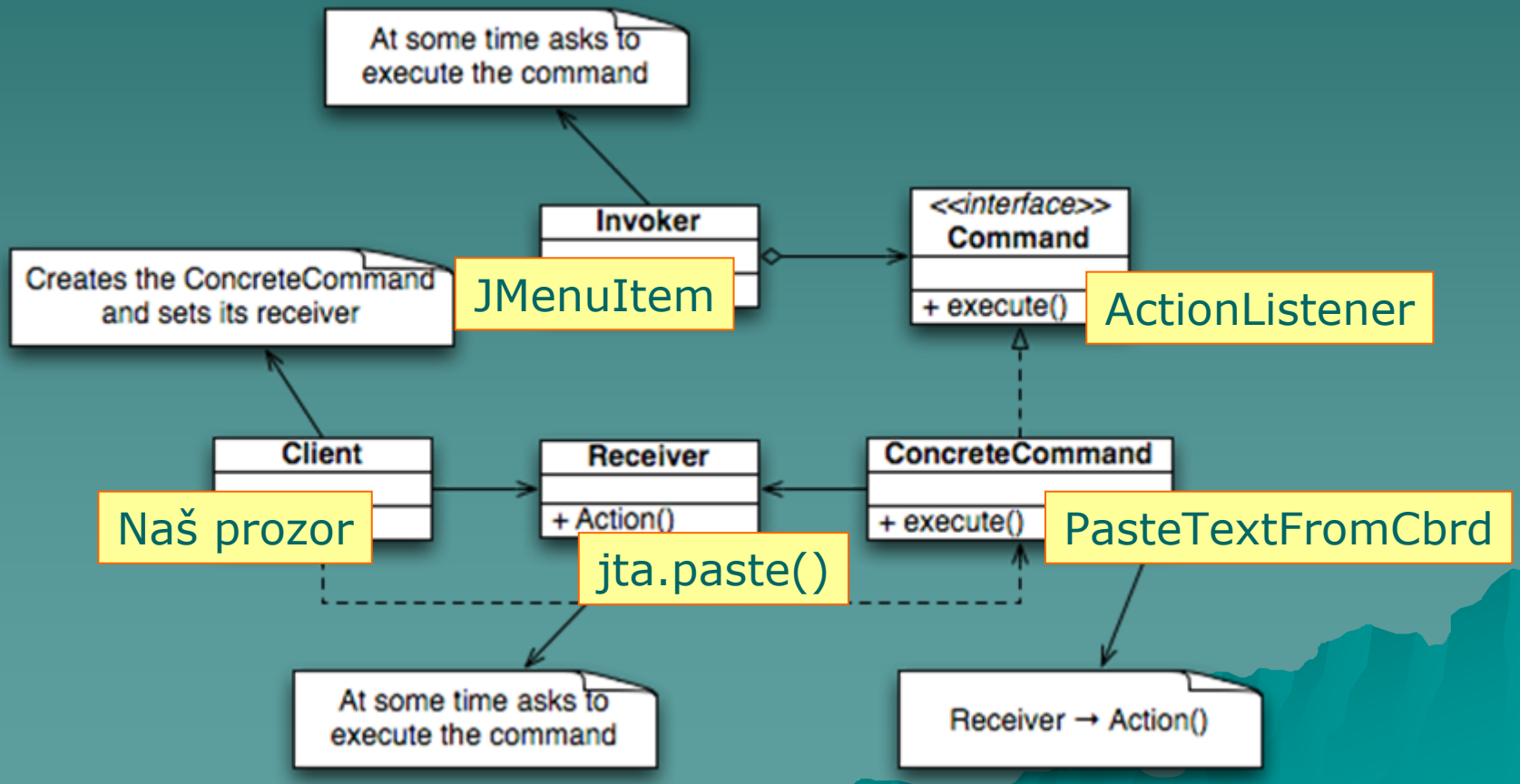
- Naredba (engl. Command)
  - Ideja je razdvojiti implementacijske detalje poslova od pozivatelja: nužna uporaba apstrakcije
  - Pozivatelj poslove vidi preko generičkog sučelja
  - Pristup omogućava implementaciju prikaza raspoloživih poslova na generički način (primjerice, možemo graditi izborničke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



Teme: UT – akcije

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

Teme: UT – akcije

- 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

Teme: UT – akcije

- Sučelje **`Action`**
  - Koristeći oblikovni obrazac Promatrač (engl. Observer) nudi klijentima mogućnost pretplate na dojavu o promjenama pohranjenih informacija

Teme: UT – akcije

- Sučelje **`Action`**

```
Interface Action extends ActionListener {
  public void actionPerformed(ActionEvent e);
  public Object getValue();
  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);
}
```

Teme: UT – akcije

- Dio ključeva:
  - `Action.NAME`, `Action.SHORT_DESCRIPTION`, `Action.ACCELERATOR_KEY`, `Action.MNEMONIC_KEY`, `Action.SMALL_ICON`, `Action.LARGE_ICON_KEY`
- Apstraktni razred `AbstractAction` implementira svu potrebnu logiku osim metode `actionPerformed`
  - Stoga poslove modeliramo upravo temeljeći se na tom razredu

Teme: UT – akcije

- 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, …

Teme: UT – razdvajanje

- 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

Teme: UT – razdvajanje

- 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, …

Teme: UT – razdvajanje

- 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);
```

Teme: UT – razdvajanje

- 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

Teme: UT – i18n

- 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

Teme: UT – razdvajanje

- 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
```

Teme: UT – i18n

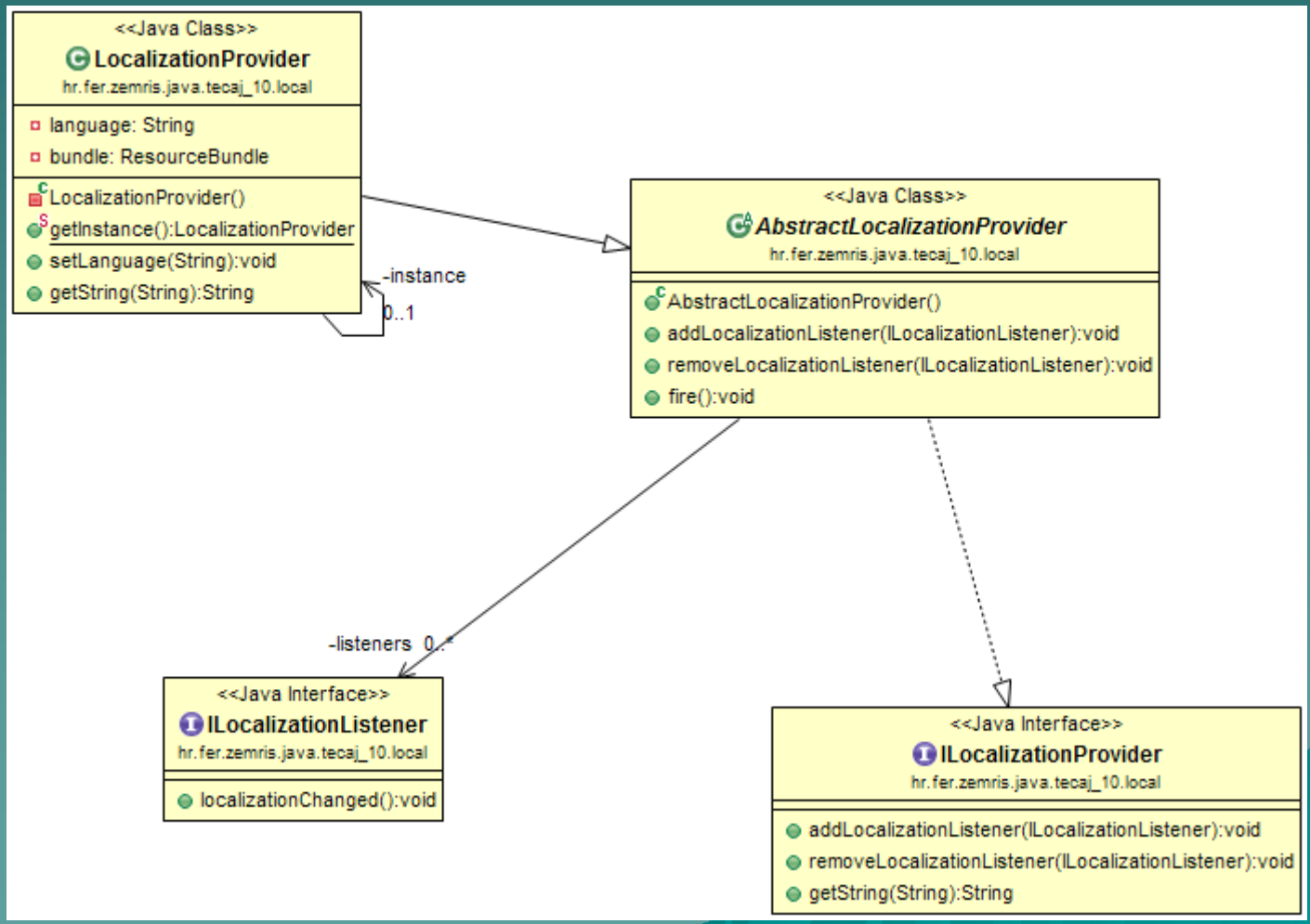
- Želimo mogućnost internacionalizacije (famozni i18n)
  - Još bismo, dakako, željeli da korisnik može lokalizaciju promijeniti dinamički, dok je aplikacija otvorena, i da se sve prekonfigurira
  - Za to trebamo ipak još malo pisati…

Teme: UT – i18n

- 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

Teme: UT – i18n

- Idemo napraviti sljedeće:



Oblikovni obrazac: Singleton

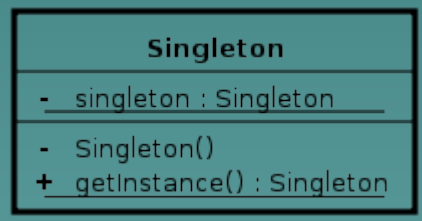
- Rješava problem gdje ne smije postojati više od jednog primjerka nekog razreda
- Tipično rješenje:
  - Definiramo razred
  - Definiramo mu konstruktor, ali privatni!
  - Definiramo privatnu statičku varijablu koja čuva referencu na jedan primjerak tog razreda
  - Definiramo javni statički getter

Oblikovni obrazac: Singleton

- 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: Singleton

- Rješava problem gdje ne smije postojati više od jednog primjerka nekog razreda

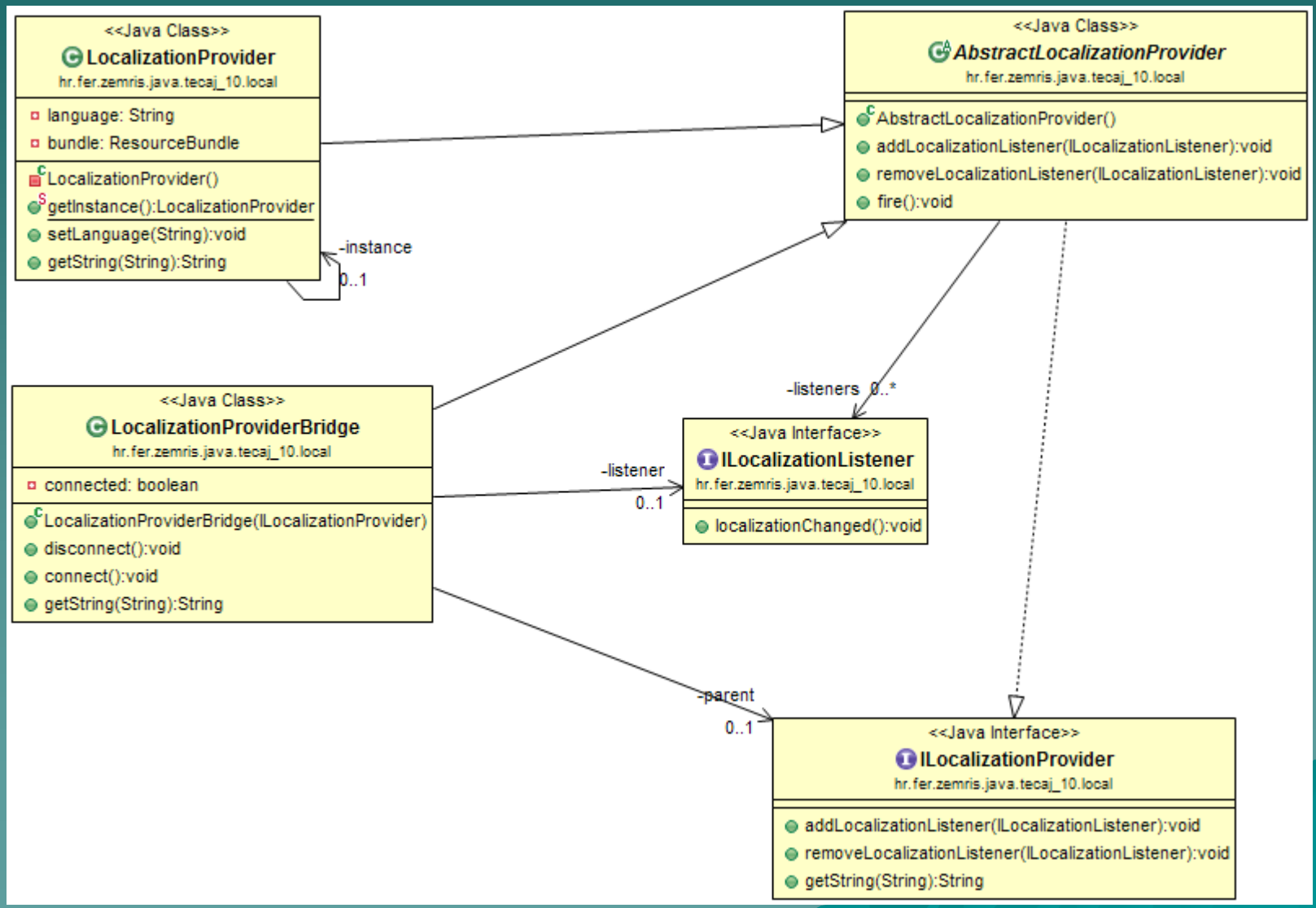


Teme: UT – i18n

- 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

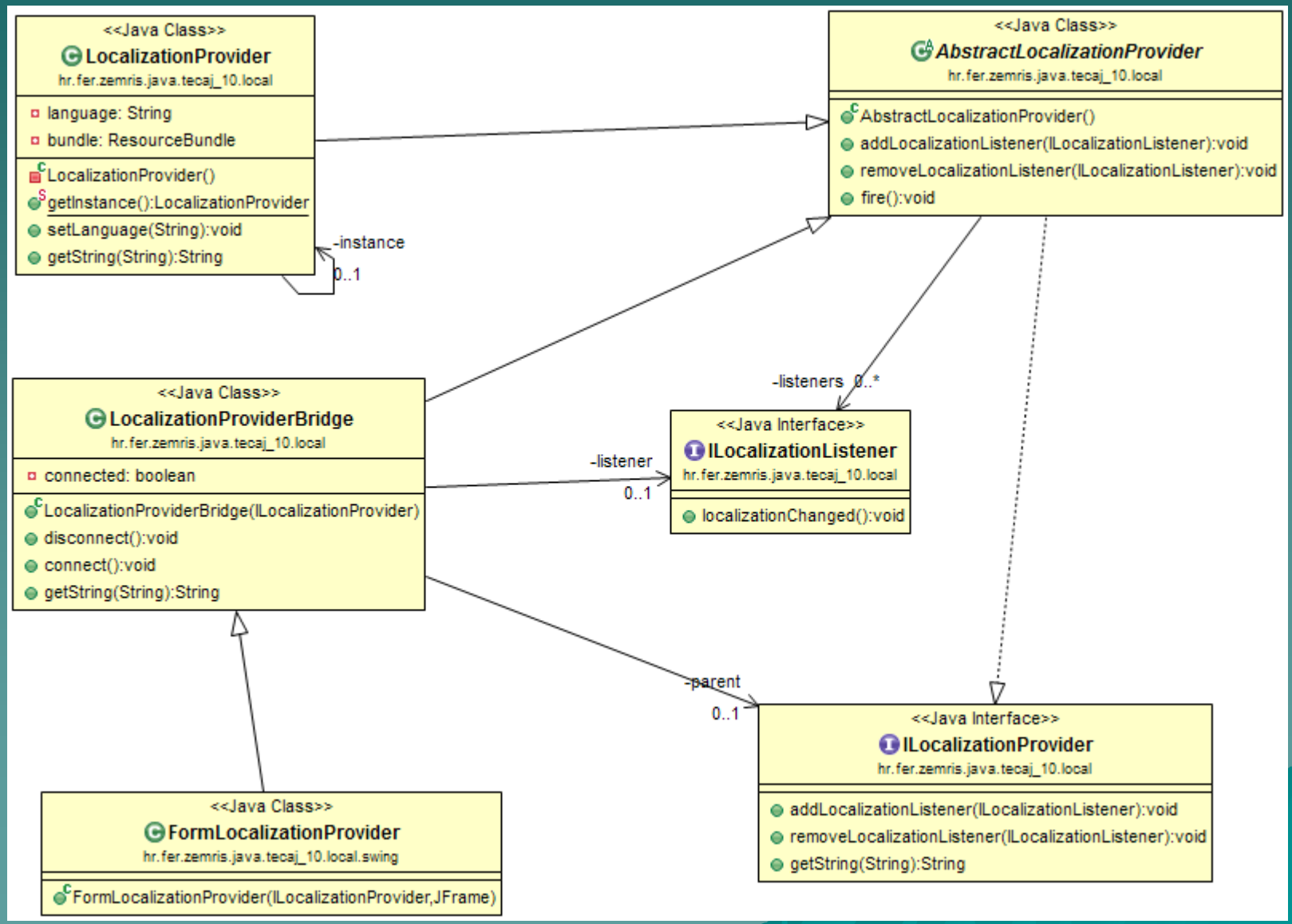
    → velik problem ako aplikacija otvara i zatvara više prozora

Teme: UT – i18n

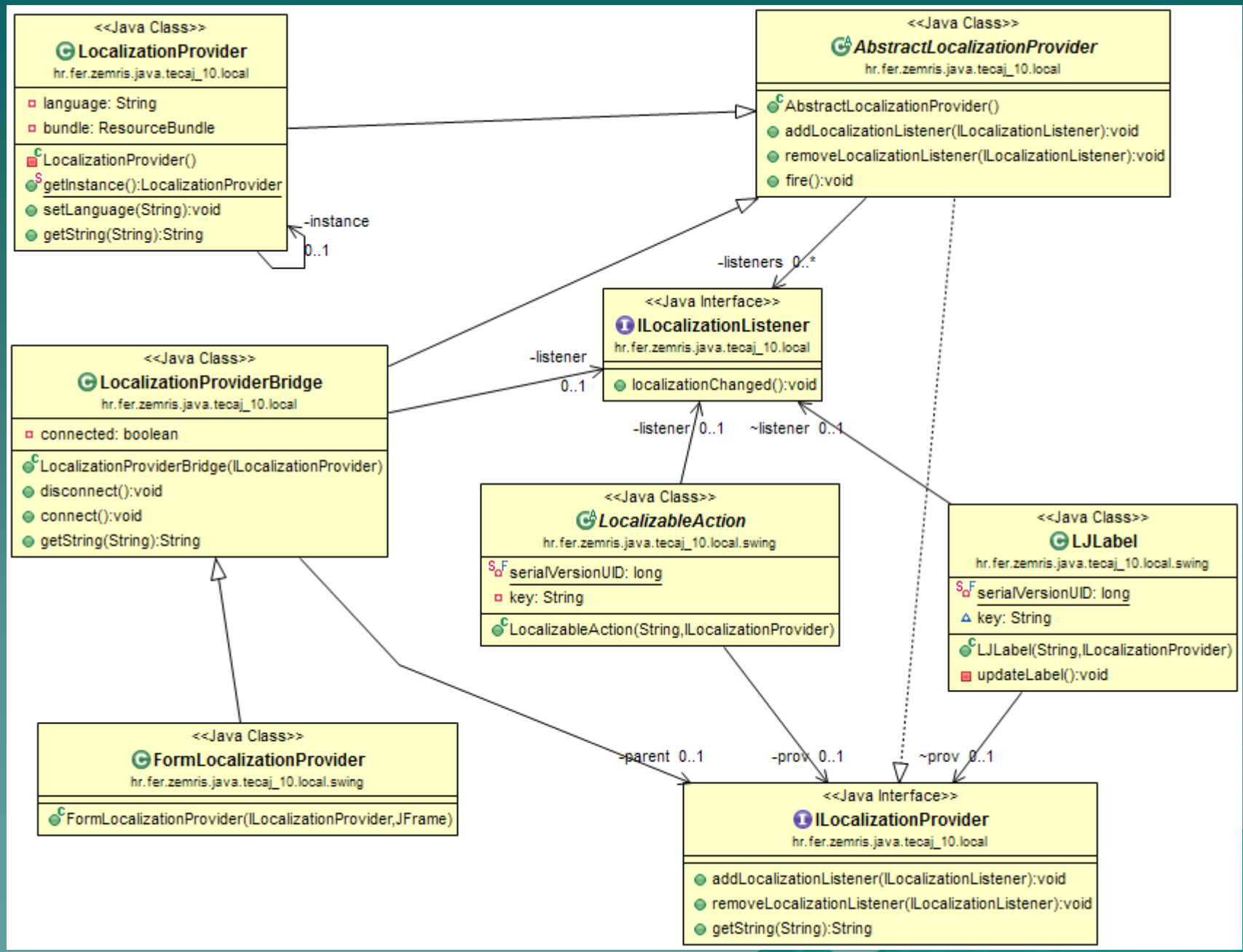


- Dodajmo još "bridge":

Teme: UT – i18n

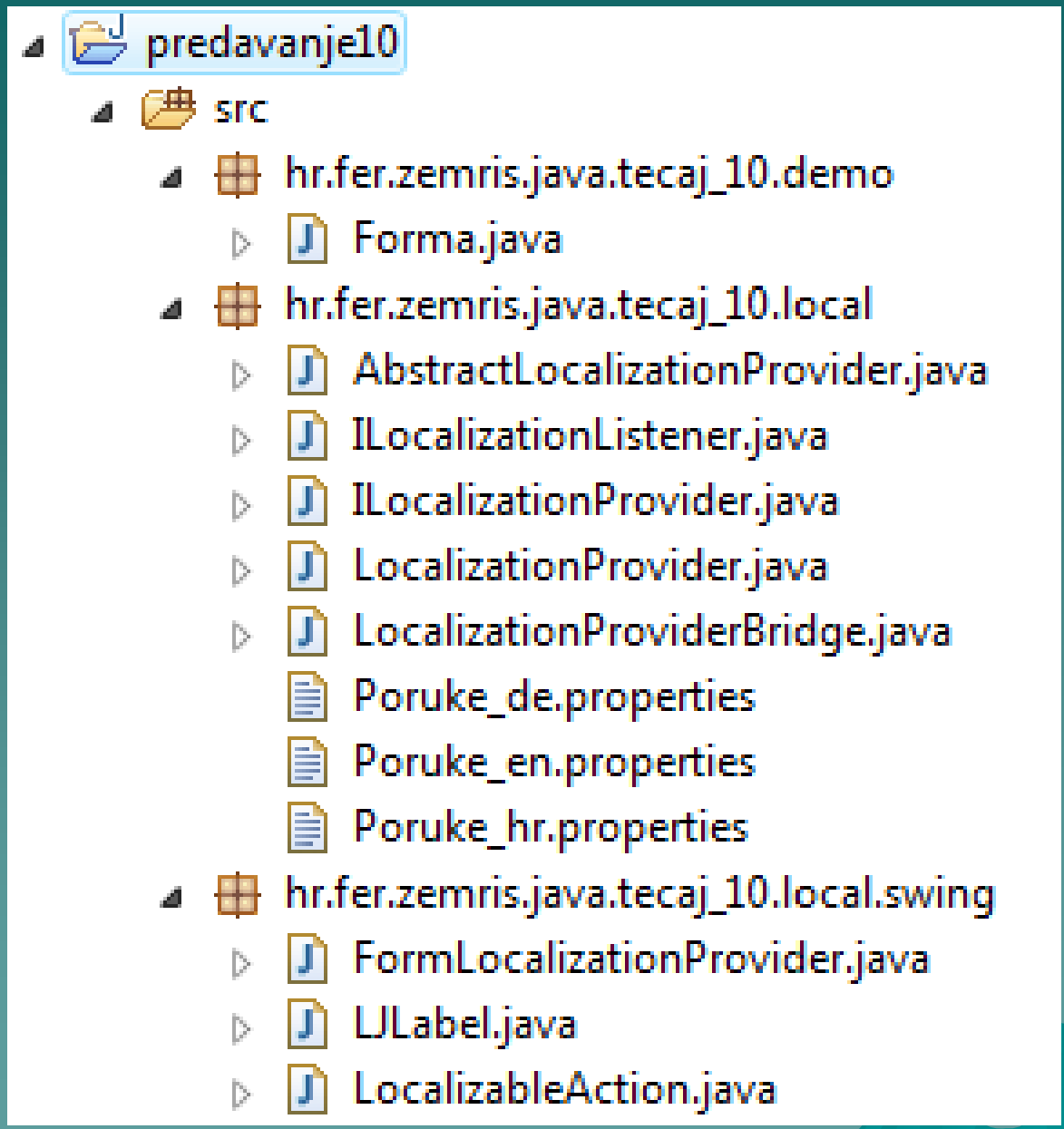


- Te podršku za forme:


<<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

Teme: UT – i18n

- 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

Teme: vlastite grafičke komponente

- Skroz jednostavno!
  - Svaka komponenta zadužena je za održavanje svojeg stanja (ne slike!)
  - Kad god se stanje promijeni, potrebno je nad komponentom pozvati **`repaint()`**: to je signal EDT-u da bi trebalo ponovno osvježiti prikaz te komponente (kad stigne!)
  - EDT će nad komponentom pozvati **`paint()`** koji dalje zove **`paintComponent()`** → to pregazimo!

Teme: vlastite grafičke komponente

- Skroz jednostavno!
  - Prilikom crtanja imamo na raspolaganju Graphics objekt (koji možemo kastati u Graphics2D) koji nudi hrpu različitih metoda
  - Samo jedna napomena: čitava površina komponente nije naša – treba poštivati insete jer je na tom području možda nacrtan border ili nešto slično