Objektno Orijentisano Programiranje 1

Komponente

Komponente

JButton	**Dugme**
JTextField	**Jednolinijsko Edit polje**
JTextArea	**Višelinijsko Edit polje (Memo)**
JLabel	**Labela**
JCheckBox	**CheckBox**
ButtonGroup	**Kontejner za Radio Button-e**
JRadioButton	**Radio Button**
JComboBox	**ComboBox**
JList	**ListBox**
JTabbedPane	**Kartice**
JOptionPane	**MessageBox**
JMenu, JMenuItem	**Meniji**
JDialog	**DialogBox**
JFrame	**Prozor**

Standardne komponente

- Primeri:
  - JTextField i KeyEvent (primer01)
  - JTextArea i CaretEvent (primer02)
  - JCheckBox, JRadioButton i ItemEvent (primer03)

JComboBox

- Karakteriše je prisustvo dva koncepta:
  - koncept modela podataka i
  - koncept renderera.
- Najjednostavniji primer podrazumeva da su oba koncepta implementirana default klasama:

```
JComboBox comboBox = new JComboBox(items);
```

  - gde je:
    - items – niz (ili vektor) podataka.

JComboBox model

- Ako je potrebno modelovati podatke na specifičan način, implementira se interfejs **`ComboBoxModel`** i redefinisati metode:
  - **`Object getSelectedItem()`** i
  - **`void setSelectedItem(Object item)`**,
- Interfejs **`ComboBoxModel`** nasleđuje **`ListModel`** interfejs, pa je potrebno redefinisati i:
  - **`Object getElementAt(int i)`**,
  - **`int getSize()`**, kao i
  - **`add/removeListDataListener(ListDataListener l)`**.
- Postoji već implementiran interfejs u vidu **`AbstractListModel`** klase, koju potrebno samo naslediti i redefinisati prve četiri navedene metode metode
  - ona dodaje podršku za odgovarajuće osluškivače.
- Klasa **`DefaultComboBoxModel`** nasleđuje **`AbstractListModel`** i implementira smeštaj podataka preko vektora.
- **`ComboBoxModel`** je uveden da bi se istakao koncept selektovane stavke, koja se prikazuje u ComboBox-u na poseban način.

primer04 JCombobox model renderer

ComputerComponent.java

```
public class ComputerComponent {
  private String name;
  private ImageIcon icon;
  public ComputerComponent(String name) {
    this.name = name;
    try {
      icon = new ImageIcon(ComputerComponent.class.getResource("/img/" + name + ".png"));
    } catch (Exception e) {
      this.icon = null;
    }
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public ImageIcon getIcon() {
    return icon;
  }
  public void setIcon(ImageIcon icon) {
    this.icon = icon;
  }
}
```

JComboBox renderer

- Ako je potrebno prikazati podatke iz ćelije na specifičan način, implementira se interfejs **`ListCellRenderer`**.
- Renderer je vezan za stavku koja se prikazuje.
- Postoji već implementiran renderer u vidu klase **`DefaultListCellRenderer`** koja podatke prikazuje upotrebom **`JLabel`** komponente.

`Component getListCellRendererComponent(JList list, Object value, int index, boolean isSelected, boolean cellHasFocus)`	**Vraća komponentu kojom se prikazuje data stavka.**

JList

- Karakteriše je prisustvo dva koncepta:
  - koncept modela podataka i
  - koncept renderera.
- Najjednostavniji primer podrazumeva da su oba koncepta implementirana default klasama:

```
JList list = new JList(items);
```

  - gde je:
    - items – niz (ili vektor) podataka.

JList model

- Ako je potrebno modelovati podatke na specifičan način, implementira se interfejs **`ListModel`**:
  - **`Object getElementAt(int i)`**,
  - **`int getSize()`**, kao i
  - **`add/removeListDataListener(ListDataListener l)`**.
- Postoji već implementiran interfejs u vidu **`AbstractListModel`** klase, koju potrebno samo naslediti i redefinisati prve dve navedene metode metode
  - ona dodaje podršku za odgovarajuće osluškivače.
- Klasa **`DefaultListModel`** nasleđuje **`AbstractListModel`** i implementira smeštaj podataka preko vektora.

primer05 JList model renderer

JList renderer

- Ako je potrebno prikazati podatke iz ćelije na specifičan način, implementira se interfejs **`ListCellRenderer`**.
- Renderer je vezan za stavku koja se prikazuje.
- Postoji već implementiran renderer u vidu klase **`DefaultTableCellRenderer`** koja podatke prikazuje upotrebom **`JLabel`** komponente.

`Component getListCellRendererComponent(JList list, Object value, int index, boolean isSelected, boolean cellHasFocus)`	**Vraća komponentu kojom se prikazuje data stavka.**

JTabbedPane

- Reprezentuje sistem međusobno preklopljenih prozora do kojih se dolazi klikom po odgovarajućoj “kartici”.
- Kartice su reprezentovane panelima (JPanel) i dodaju se metodom add:

```
tabbedPane.addTab(
  “Naslov na kartici", icon, panel,
  “Tooltip tekst");
```

JMenu

- Reprezentuje pull-down meni.
- Proces kreiranja menija počinje kreiranjem trake menija:
  - **`menuBar = new JMenuBar();`**
- Opcije menija su reprezentovane klasom JMenu:
  - **`menu = new JMenu(“Edit");`**
  - **`menu.setMnemonic(KeyEvent.VK_E);`**
  - **`menuBar.add(menu);`**
- Opcije menija su reprezentovane klasom JMenuItem:

```
menuItem = new JMenuItem(“Copy", KeyEvent.VK_C);
// menuItem.setMnemonic(KeyEvent.VK_C);
menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_C, ActionEvent.CTRL_MASK));
menu.add(menuItem);
```

JPopupMenu

Primer07

- Reprezentuje popup meni.
- Kreiranje:

```
JPopupMenu popup = new JPopupMenu();
menuItem = new JMenuItem("A popup menu item");
menuItem.addActionListener(this);
popup.add(menuItem);
menuItem = new JMenuItem("Another popup menu item");
menuItem.addActionListener(this);
popup.add(menuItem);

MouseListener popupListener = new
  PopupListener(popup);
onaj_na_koji_treba_kliknuti_za_popup.addMouseListener(
    popupListener);
```

JTable

Primer08

- Reprezentuje tabelu.
- Karakteriše je prisustvo tri koncepta:
  - koncept modela podataka,
  - koncept editora i
  - koncept renderera.
- Najjednostavniji primer podrazumeva da su sva tri koncepta implementirana default klasama:
  - **`JTable jTable1 = new JTable(data, columnNames);`**
  - gde su:
    - data – dvodimenzionalni niz podataka,
    - columnNames – niz stringova koji reprezentuju nazive kolona (zaglavlje).

Model podataka (TableModel)

- Ako je potrebno modelovati podatke na specifičan način, implementira se interfejs **`TableModel`**.
- Postoji već implementiran interfejs u vidu **`AbstractTableModel`** klase, koju potrebno samo naslediti i redefinisati odgovarajuće metode (minimalno **`getRowCount()`**, **`getColumnCount()`** i **`getValueAt()`**).
- Klasa **`DefaultTableModel`** nasleđuje **`AbstractTableModel`** i implementira smeštaj podataka preko vektora.

`Class getColumnClass(int columnIndex)`	Vraća klasu koja reprezentuje zadatu kolonu. Na osnovu ove informacije, tabela određuje koji će editor i renderer da koristi za kolonu.
`int getColumnCount()`	Vraća broj kolona u tabeli.
`String getColumnName(int columnIndex)`	Vraća naziv zadate kolone.
`int getRowCount()`	Vraća broj vrsta u tabeli.
`Object getValueAt(int rowIndex, int columnIndex)`	Vraća vrednost iz tabele za zadatu vrstu i kolonu.
`boolean isCellEditable(int rowIndex, int columnIndex)`	Vraća true ako zadata ćelija može da se menja (edituje).
`void setValueAt(Object aValue, int rowIndex, int columnIndex)`	Postavlja vrednost podatka u ćeliju datu vrstom i kolonom.

Prikaz ćelije (TableCellRenderer)

- Ako je potrebno prikazati podatke iz ćelije na specifičan način, implementira se interfejs **`TableCellRenderer`**.
- Renderer je vezan za kolonu.
- Postoji već implementiran renderer u vidu klase **`DefaultTableCellRenderer`** koja podatke prikazuje upotrebom **`JLabel`** komponente.

`Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column)`	**Vraća komponentu kojom se prikazuje data ćelija.**

Editovanje ćelije (TableCellEditor)

- Ako je potrebno editovati podatke iz ćelije na specifičan način, implementira se interfejs **`TableCellEditor`**.
- Editor je vezan za kolonu.
- Postoji već implementiran editor u vidu klase **`AbstractCellEditor`**.
- Klasa **`DefaultCellEditor`** nasleđuje ovu klasu i implementira elementarno editovanje ćelija (**`JTexField`**, **`JCheckBox`** i **`JComboBox`** komponente).

`Component getTableCellEditorComponent(JTable table, Object value, boolean isSelected, int row, int column)`	Vraća komponentu kojom se edituje dati tip ćelije.
`Object getCellEditorValue()`	Ova metoda se automatski poziva kada editor celije završi editovanje (nakon što je završetak potvrđen metodom stopCellEditing()). Posao ove metode je da prihvati vrednost iz editora i da je prosledi u tabelu.
`boolean stopCellEditing()`	Ova metoda se automatski poziva kada korisnik proba da zavrsi editovanje. Vraća true ako dozvoljava završetak editovanja.
	Primer11

JTree

- Služi za hijerarhijski prikaz (stablo).
- Stablo se kreira dodavanjem čvorova, počev od prvog (root).
  - Čvorovi su reprezentovani klasama koje implementiraju interfejs **`TreeNode`** – osnovna klasa koja implementira ovaj interfejs je **`DefaultMutableTreeNode`**.
  - dodavanje podčvora – metoda **`add(newChild)`**.
- Reakcija na klik mišem – **`TreeSelectionListener`**.

JTree model

- Ako je potrebno modelovati podatke na specifičan način, implementira se interfejs **`TreeModel`**.
- Ovaj model je proširen interfejsom **`MutableTreeNode`**, koji dodaje svojstvo izmene čvora preko dodavanja i uklanjanja čvorova, kao i izmene objekta smeštenog u čvor.
- Klasa **`DefaultTreeModel`** implementira **`TreeModel`** i implementira smeštaj podataka preko hijerarhije **`TreeNode`** objekata.

`public Object getChild(Object parent, int index)`	Vraća čvor zadat indeksom.
`public int getChildCount(Object parent)`	Vraća broj direktnih potomaka.
`public int getIndexOfChild(Object parent, Object child)`	Vraća redni broj zadatog čvora.
`Object getRoot()`	Vraća koren.
`public boolean isLeaf(Object node)`	Vraća true ako je list.
`void addTreeModelListener(TreeModelListener l)`	Dodaje osluškivač promena strukture stabla.
`void removeTreeModelListener(TreeModelListener l)`	Uklanja osluškivač promena strukture stabla.
`void valueForPathChanged(TreePath path, Object newValue)`	Poziva se kada je promenjena vrednost čvora (identifikovanog putanjom) na novu vrednost

TreeNode interfejs

- Definiše metode koje bi trebalo da implementira objekat koji se smešta u stablo:
  - **`Enumeration children();`**
  - **`boolean getAllowsChildren();`**
  - **`int getChildCount();`**
  - **`int getIndex(TreeNode node);`**
  - **`TreeNode getParent();`**
  - **`boolean isLeaf();`**

MutableTreeNode interfejs

- Definiše izmenjiv element stabla, u smislu da je moguće dodavanje i uklanjanje čvorova, kao i izmena sadržaja čvora:
  - **`void insert(MutableTreeNode node);`**
  - **`void remove(int index);`**
  - **`void remove(MutableTreeNode node);`**
  - **`void removeFromParent();`**
  - **`void setParent(MutableTreeNode node);`**
  - **`void setUserObject(Object object);`**

JTree renderer

- Ako je potrebno prikazati podatke iz čvora na specifičan način, implementira se interfejs **`TreeCellRenderer`**.
- Renderer je vezan za čvor stabla.
- Postoji već implementiran renderer u vidu klase **`DefaultTreeCellRenderer`** koja podatke prikazuje upotrebom **`JLabel`** komponente.

`Component getTableCellRendererComponent(JTree tree, boolean selected, boolean expanded, boolean leaf, int row, boolean hasFocus)`	**Vraća komponentu kojom se prikazuje dati čvor stabla.**

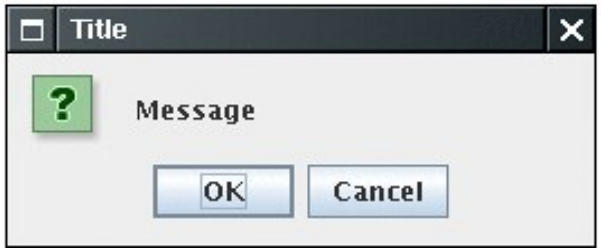
Dijalozi

- Služe za interakciju sa korisnikom
- Mogu biti:
  - modalni (ne može se preći na glavni ekran dok se ne ugasi dijalog)
  - nemodalni (može se preći na glavni ekran).
- Postoje
  - predefinisani dijalozi (JOptionPane)
  - dijalozi koje kreira korisnik (JDialog)
- Dijalozi kreirani pomoću JOptionPane su uvek modalni (blokiraju izvršavanje dok se dijalog ne zatvori)
- Naslednici klase JDialog mogu a i ne moraju biti modalni, u zavisnosti od parametra konstruktora:

```
JDialog(Dialog owner, String title, boolean modal)
```

Klasa JOptionPane

- Najčešće korišćene metode:
  - **`showMessageDialog`** – koristi se za prikaz poruke, sadrži samo OK dugme
  - **`showConfirmDialog`** – prikazuje poruku i traži odgovor od korisnika (može da sadrži dugmad OK, Cancel, Yes, No)
  - **`showInputDialog`** – prikazuje poruku i komponentu korisničkog interfejsa u okviru koje korisnik unosi svoj odgovor (linija za unos ili combo-box)



Klasa JOptionPane

- Izgled ikone zavisi od prosleđenog tipa poruke (konstante definisane u okviru klase JOptionPane):
  - ERROR_MESSAGE
  - INFORMATION_MESSAGE
  - WARNING_MESSAGE
  - QUESTION_MESSAGE
  - PLAIN_MESSAGE
- Raspoloživa dugmad zavise od prosleđene opcije
  - DEFAULT_OPTION
  - YES_NO_OPTION
  - YES_NO_CANCEL_OPTION
  - OK_CANCEL_OPTION

Klasa JOptionPane

- Metoda **`showConfirmDialog`** vraća integer koji označava izabrano dugme:
  - OK_OPTION
  - CANCEL_OPTION
  - YES_OPTION
  - NO_OPTION
  - CLOSED_OPTION (dijalog zatvoren bez izbora dugmića)
- Metoda **`showInputDialog`** vraća String koji je korisnik uneo

Primeri korišćenja klase JOptionPane

```
import javax.swing.*;

class MessageDialogExample {
  public static void main(String[] args) {
       JOptionPane.showMessageDialog(null,
      "First message");
  }
}
```

Primer14

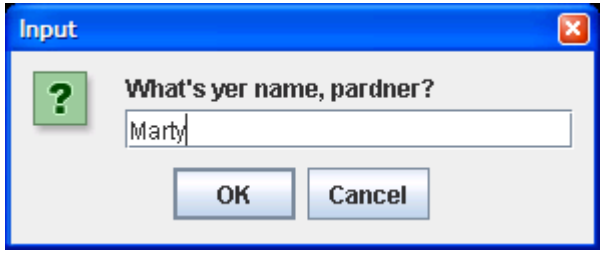
Primeri korišćenja klase JOptionPane

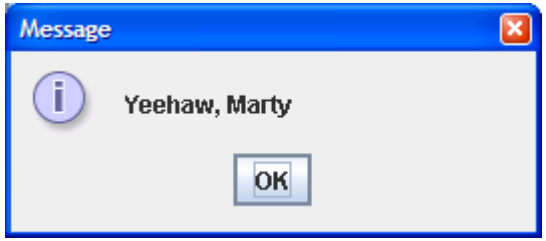
```
import javax.swing.*;

class ConfirmDialogExample {
  public static void main(String[] args) {
    int choice = JOptionPane.showConfirmDialog(null,
                   "Erase your hard disk?");
    if (choice == JOptionPane.YES_OPTION) {
      JOptionPane.showMessageDialog(null, "Disk erased!");
    } else {
      JOptionPane.showMessageDialog(null, "Cancelled.");
    }
  }
}
```

Primer15

Primeri korišćenja klase JOptionPane

```
import javax.swing.*;

class InputDialogExample {
  public static void main(String[] args) {
    String name = JOptionPane.showInputDialog(null,
                        "What's yer name, pardner?");
    JOptionPane.showMessageDialog(null, "Yeehaw, " +
   name);
  }
}
```





Primer16

Raspoložive metode
`showMessageDialog`

- **`static void showMessageDialog(Component parent, Object message, String title, int messageType, Icon icon)`**
- **`static void showMessageDialog(Component parent, Object message, String title, int messageType)`**
- **`static void showMessageDialog(Component parent, Object message)`**

Parametri:

- **`parent`** - “roditelj” dijaloga (može biti null)
- **`message`** - Poruka koja se prikazuje u okviru dijaloga
- **`title`** - Naslov dijaloga
- **`messageType`** - ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, QUESTION_MESSAGE, PLAIN_MESSAGE
- **`icon`** - Ikona koja se prikazuje umesto DEFAULT ikone za taj tip poruke

Raspoložive metode
`showConfirmDialog`

- **`static int showConfirmDialog(Component parent, Object message, String title, int optionType, int messageType, Icon icon)`**
- **`static int showConfirmDialog(Component parent, Object message, String title, int optionType, int messageType)`**
- **`static int showConfirmDialog(Component parent, Object message, String title, int optionType)`**
- **`static int showConfirmDialog(Component parent, Object message)`**

Parametri:

- **`parent`** - “roditelj” dijaloga (može biti null)
- **`message`** - Poruka koja se prikazuje u okviru dijaloga
- **`title`** - Naslov dijaloga
- **`messageType`** - ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, QUESTION_MESSAGE, PLAIN_MESSAGE
- **`optionType`** - DEFAULT_OPTION, YES_NO_OPTION, YES_NO_CANCEL_OPTION, OK_CANCEL_OPTION
- **`icon`** - Ikona koja se prikazuje umesto DEFAULT ikone za taj tip poruke

Raspoložive metode
showInputDialog

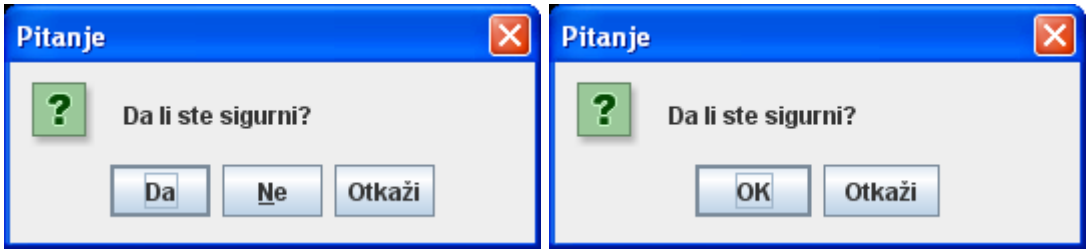
- **`static Object showInputDialog(Component parent, Object message, String title, int messageType, Icon icon, Object[] values, Object default)`**
- **`static String showInputDialog(Component parent, Object message, String title, int messageType)`**
- **`static String showInputDialog(Component parent, Object message)`**
- **`static String showInputDialog(Object message)`**
- **`static String showInputDialog(Component parent, Object message, Object default)`**
- **`static String showInputDialog(Object message, Object default)`**

Parametri:

- **`parent`** - “roditelj” dijaloga (može biti null)
- **`message`** - Poruka koja se prikazuje u okviru dijaloga
- **`title`** - Naslov dijaloga
- **`messageType`** - ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, QUESTION_MESSAGE, PLAIN_MESSAGE
- **`values`** - Niz vrednosti koje se prikazuju u combo-boxu
- **`default`** - Default vrednost koja se prikazuje u komponenti za unos
- **`icon`** - Ikona koja se prikazuje umesto DEFAULT ikone za taj tip poruke

Primer lokalizovanja JOptionPane

```
public class UIExample {
public static void main(String[] args) {
    // tekst na default dugmicima
    UIManager.put("OptionPane.yesButtonText", "Da");
    UIManager.put("OptionPane.noButtonText", "Ne");
    UIManager.put("OptionPane.cancelButtonText", "Otka\u017ei");
    UIManager.put("OptionPane.okButtonText", "OK");

    int answer1 = JOptionPane.showConfirmDialog(null, "Da li ste sigurni?",
        "Pitanje", JOptionPane.YES_NO_CANCEL_OPTION);

    int answer2 = JOptionPane.showConfirmDialog(null, "Da li ste sigurni?",
        "Pitanje", JOptionPane.OK_CANCEL_OPTION);
  }
}
```



Unicode karakteri slova sa dijakriticima

- Za ispisivanje na menijima, dugmićima i labelama radi prenosivosti koristiti unicode kodove za slova sa dijakriticima ("naša" slova):
  - Š 0160
  - š 0161
  - Ć 0106
  - ć 0107
  - Ž 017d
  - ž 017e
  - Č 010c
  - č 010d
  - Đ 0110
  - đ 0111

JDialog

- Reprezentuje dijalog ekran
- Dijalog može biti:
  - modalan (ne može se preći na glavni ekran dok se ne ugasi dijalog)
  - nemodalan (može se preći na glavni ekran).
- Obično u okviru aplikacije postoji jedna instanca klase JFrame (osnovni prozor), a ostalo su dijalozi, mada je moguće da postoji i više instanci klase JFrame ako je potrebno
- Obično se kreira u glavnom prozoru, pa se poziva metodom **`setVisible(true)`**, a gasi metodom **`setVisible(false)`**

Primer18

LookAndFeel

- Java poseduje mehanizam definisanja izgleda GUI komponenti.
- **`LookAndFeel`** mehanizam omogućuje promenu izgleda aplikacije bez restarta.
- Postoje ugrađene **`LookAndFeel`** klase.
- Korisnički definisane **`LookAndFeel`** klase.
- Dekoracija prozora upotrebom:
  - **`JFrame.setDefaultLookAndFeelDecorated(true);`**

Primer19