Student : Tomislav Dundović

Profesor : Doc.dr.sc Ante Panjkota

Fakultet : Sveučilište u Zadru

Studij : Stručni Studij Informacijskih Tehnologija

Kolegij : Napredno Objektno Orijentirano Programiranje

Popis Studenata - Projekt NOOP Tomislav

Aplikacija se sastoji od panela podPloca te ploPres koji su smješteni unutar glavnoga framea.

Unutar panela podPloca su dostupne komponente uz pomoč kojih korisnik može definirati koje će biti studentove informacije.

Komponente koje se nalazu su JLabel, JTextField, JFormattedTextField, JComboBox, JRadioBox, JList te JButton.

Uz pomoć ovih komponenti korisniku se nudi razni broj opcija uz pomoć kojih će kreirati studenta.

Definira se njegovo ime, prezime, datum rođenja, odabir fakulteta te studija, da li je izvanredan ili redoviti student, na kojoj godini je fakulteta,

da li ima pravo na x-icu te da li je rekreativan ili natjecateljski sportaš.

Nakon što je korisnik unio sve podatke te pritisnuo botun Potvrda, sav output od studenta će se prikazati na prezentacijskom panelu unutar JTablea sa

svojim jedistveni id-om. Unutar tablice korisnik može unositi onoliko broj studenata koliko on želi.

Sa gornje strane aplikacije se nalazi traka ili tkzv. JMenuBar koji sadrži odabir "Opcija" i "Server".

Unutar odabira Opcija korisnik može novonastale studente spremiti unutar svoga računala u obliku datoteke na bilo kojoj lokaciji.

Ekstenzija sa kojom će se datoteka spremiti jest .stud. Osim spremnja datoteke korisnik ju može i importati.

Unutar odabira Server studentski podatci se mogu direktno spremiti te importati unutar online baze podataka.

Ova baza podataka se nalazi na stranici db4free.com te je potrebna registracija korisnika.

Unutar ove baze podataka je kreirana SQL tablica koja sadrži iste stupce kao što ima i aplikacija unutar JTable-a (id, ime, prezime, datum rođenja etc...).

Prilikom spremanja studenata sa aplikacije na bazu podataka svi njegovi atributi će se poredati u svoje stupce te će se ispisati njegov sadržaj.

Ova aplikacija je napravljena prema MVC(Model, View, Controller) arhitekturi te je dostupna JavaDoc Api dokumentacija.

Osim JavaDoc-a unutar aplikacije su zapisane sve biblioteke koje su korištenje unutar ovoga projekta te sa njihovim značenjima.

Unutar projekta je dostupan UML dijagram koji je napravljen uz pomoč ObjectAid UML Explorera.

Izgled aplikacije te objašnjavanje njihovih elemenata se nalazi unutar projektne mape "wireframing-tutorial"

Za izgled korisničkoga sučelja se koristio open source alat Pencil koji je instaliran unutar Eclipse-a.

Popis biblioteka te importa

.

.

.

-----------import java.sql.SQLException;-----------

public class SQLException

extends Exception

implements Iterable<Throwable>

An exception that provides information on a database access error or other errors.

Each SQLException provides several kinds of information:

a string describing the error. This is used as the Java Exception message, available via the method getMessage.

a "SQLstate" string, which follows either the XOPEN SQLstate conventions or the SQL:2003 conventions. The values of the SQLState string are described in the appropriate spec. The DatabaseMetaData method getSQLStateType can be used to discover whether the driver returns the XOPEN type or the SQL:2003 type.

an integer error code that is specific to each vendor. Normally this will be the actual error code returned by the underlying database.

a chain to a next Exception. This can be used to provide additional error information.

the causal relationship, if any for this SQLException.

Since:

1.1

-----------import java.io.File;-----------

public class File

extends Object

implements Serializable, Comparable<File>

An abstract representation of file and directory pathnames.

User interfaces and operating systems use system-dependent pathname strings to name files and directories. This class presents an abstract, system-independent view of hierarchical pathnames. An abstract pathname has two components:

An optional system-dependent prefix string, such as a disk-drive specifier, "/" for the UNIX root directory, or "\\\\" for a Microsoft Windows UNC pathname, and

A sequence of zero or more string names.

The first name in an abstract pathname may be a directory name or, in the case of Microsoft Windows UNC pathnames, a hostname. Each subsequent name in an abstract pathname denotes a directory; the last name may denote either a directory or a file. The empty abstract pathname has no prefix and an empty name sequence.

-----------import java.io.IOException;-----------

public class IOException

extends Exception

Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.

Since:

JDK1.0

-----------import java.util.List;-----------

public interface List<E>

extends Collection<E>

An ordered collection (also known as a sequence). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list.

Unlike sets, lists typically allow duplicate elements. More formally, lists typically allow pairs of elements e1 and e2 such that e1.equals(e2), and they typically allow multiple null elements if they allow null elements at all. It is not inconceivable that someone might wish to implement a list that prohibits duplicates, by throwing runtime exceptions when the user attempts to insert them, but we expect this usage to be rare.

-----------import java.io.FileInputStream;-----------

public class FileInputStream

extends InputStream

A FileInputStream obtains input bytes from a file in a file system. What files are available depends on the host environment.

FileInputStream is meant for reading streams of raw bytes such as image data. For reading streams of characters, consider using FileReader.

Since:

JDK1.0

-----------import java.io.FileOutputStream;-----------

public class FileOutputStream

extends OutputStream

A file output stream is an output stream for writing data to a File or to a FileDescriptor. Whether or not a file is available or may be created depends upon the underlying platform. Some platforms, in particular, allow a file to be opened for writing by only one FileOutputStream (or other file-writing object) at a time. In such situations the constructors in this class will fail if the file involved is already open.

FileOutputStream is meant for writing streams of raw bytes such as image data. For writing streams of characters, consider using FileWriter.

Since:

JDK1.0

-----------import java.io.ObjectInputStream;-----------

public class ObjectInputStream

extends InputStream

implements ObjectInput, ObjectStreamConstants

An ObjectInputStream deserializes primitive data and objects previously written using an ObjectOutputStream.

ObjectOutputStream and ObjectInputStream can provide an application with persistent storage for graphs of objects when used with a FileOutputStream and FileInputStream respectively. ObjectInputStream is used to recover those objects previously serialized. Other uses include passing objects between hosts using a socket stream or for marshaling and unmarshaling arguments and parameters in a remote communication system.

-----------import java.io.ObjectOutputStream;-----------

public class ObjectOutputStream

extends OutputStream

implements ObjectOutput, ObjectStreamConstants

An ObjectOutputStream writes primitive data types and graphs of Java objects to an OutputStream. The objects can be read (reconstituted) using an ObjectInputStream. Persistent storage of objects can be accomplished by using a file for the stream. If the stream is a network socket stream, the objects can be reconstituted on another host or in another process.

Only objects that support the java.io.Serializable interface can be written to streams. The class of each serializable object is encoded including the class name and signature of the class, the values of the object's fields and arrays, and the closure of any other objects referenced from the initial objects.

-----------import java.sql.Connection;-----------

public interface Connection

extends Wrapper, AutoCloseable

A connection (session) with a specific database. SQL statements are executed and results are returned within the context of a connection.

A Connection object's database is able to provide information describing its tables, its supported SQL grammar, its stored procedures, the capabilities of this connection, and so on. This information is obtained with the getMetaData method.

-----------import java.sql.DriverManager;-----------

public class DriverManager

extends Object

The basic service for managing a set of JDBC drivers.

NOTE: The DataSource interface, new in the JDBC 2.0 API, provides another way to connect to a data source. The use of a DataSource object is the preferred means of connecting to a data source.

As part of its initialization, the DriverManager class will attempt to load the driver classes referenced in the "jdbc.drivers" system property. This allows a user to customize the JDBC Drivers used by their applications. For example in your ~/.hotjava/properties file you might specify:

jdbc.drivers=foo.bah.Driver:wombat.sql.Driver:bad.taste.ourDriver

-----------import java.sql.PreparedStatement;-----------

public interface PreparedStatement

extends Statement

An object that represents a precompiled SQL statement.

A SQL statement is precompiled and stored in a PreparedStatement object. This object can then be used to efficiently execute this statement multiple times.

Note: The setter methods (setShort, setString, and so on) for setting IN parameter values must specify types that are compatible with the defined SQL type of the input parameter. For instance, if the IN parameter has SQL type INTEGER, then the method setInt should be used.

If arbitrary parameter type conversions are required, the method setObject should be used with a target SQL type.

In the following example of setting a parameter, con represents an active connection:

PreparedStatement pstmt = con.prepareStatement("UPDATE EMPLOYEES

SET SALARY = ? WHERE ID = ?");

pstmt.setBigDecimal(1, 153833.00)

pstmt.setInt(2, 110592)

-----------import java.sql.ResultSet;-----------

public interface ResultSet

extends Wrapper, AutoCloseable

A table of data representing a database result set, which is usually generated by executing a statement that queries the database.

A ResultSet object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The next method moves the cursor to the next row, and because it returns false when there are no more rows in the ResultSet object, it can be used in a while loop to iterate through the result set.

A default ResultSet object is not updatable and has a cursor that moves forward only. Thus, you can iterate through it only once and only from the first row to the last row. It is possible to produce ResultSet objects that are scrollable and/or updatable. The following code fragment, in which con is a valid Connection object, illustrates how to make a result set that is scrollable and insensitive to updates by others, and that is updatable. See ResultSet fields for other options.

Statement stmt = con.createStatement(

ResultSet.TYPE\_SCROLL\_INSENSITIVE,

ResultSet.CONCUR\_UPDATABLE);

ResultSet rs = stmt.executeQuery("SELECT a, b FROM TABLE2");

// rs will be scrollable, will not show changes made by others,

// and will be updatable

-----------import java.util.Arrays;-----------

public class Arrays

extends Object

This class contains various methods for manipulating arrays (such as sorting and searching). This class also contains a static factory that allows arrays to be viewed as lists.

The methods in this class all throw a NullPointerException, if the specified array reference is null, except where noted.

The documentation for the methods contained in this class includes briefs description of the implementations. Such descriptions should be regarded as implementation notes, rather than parts of the specification. Implementors should feel free to substitute other algorithms, so long as the specification itself is adhered to. (For example, the algorithm used by sort(Object[]) does not have to be a MergeSort, but it does have to be stable.)

This class is a member of the Java Collections Framework.

Since:

1.2

-----------import java.util.LinkedList;-----------

public class LinkedList<E>

extends AbstractSequentialList<E>

implements List<E>, Deque<E>, Cloneable, Serializable

Doubly-linked list implementation of the List and Deque interfaces. Implements all optional list operations, and permits all elements (including null).

All of the operations perform as could be expected for a doubly-linked list. Operations that index into the list will traverse the list from the beginning or the end, whichever is closer to the specified index.

-----------import java.io.Serializable;-----------

public interface Serializable

Serializability of a class is enabled by the class implementing the java.io.Serializable interface. Classes that do not implement this interface will not have any of their state serialized or deserialized. All subtypes of a serializable class are themselves serializable. The serialization interface has no methods or fields and serves only to identify the semantics of being serializable.

To allow subtypes of non-serializable classes to be serialized, the subtype may assume responsibility for saving and restoring the state of the supertype's public, protected, and (if accessible) package fields. The subtype may assume this responsibility only if the class it extends has an accessible no-arg constructor to initialize the class's state. It is an error to declare a class Serializable if this is not the case. The error will be detected at runtime.

-----------import javax.swing.SwingUtilities;-----------

public class SwingUtilities

extends Object

implements SwingConstants

A collection of utility methods for Swing.

Since:

1.2

-----------import java.awt.BorderLayout;-----------

public class BorderLayout

extends Object

implements LayoutManager2, Serializable

A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center. Each region may contain no more than one component, and is identified by a corresponding constant: NORTH, SOUTH, EAST, WEST, and CENTER. When adding a component to a container with a border layout, use one of these five constants, for example:

Panel p = new Panel();

p.setLayout(new BorderLayout());

p.add(new Button("Okay"), BorderLayout.SOUTH);

As a convenience, BorderLayout interprets the absence of a string specification the same as the constant CENTER:

Panel p2 = new Panel();

p2.setLayout(new BorderLayout());

p2.add(new TextArea()); // Same as p.add(new TextArea(), BorderLayout.CENTER);

-----------import java.awt.Color;-----------

public class Color

extends Object

implements Paint, Serializable

The Color class is used to encapsulate colors in the default sRGB color space or colors in arbitrary color spaces identified by a ColorSpace. Every color has an implicit alpha value of 1.0 or an explicit one provided in the constructor. The alpha value defines the transparency of a color and can be represented by a float value in the range 0.0 - 1.0 or 0 - 255. An alpha value of 1.0 or 255 means that the color is completely opaque and an alpha value of 0 or 0.0 means that the color is completely transparent. When constructing a Color with an explicit alpha or getting the color/alpha components of a Color, the color components are never premultiplied by the alpha component.

-----------import java.awt.event.ActionEvent;-----------

public class ActionEvent

extends AWTEvent

A semantic event which indicates that a component-defined action occurred. This high-level event is generated by a component (such as a Button) when the component-specific action occurs (such as being pressed). The event is passed to every ActionListener object that registered to receive such events using the component's addActionListener method.

Note: To invoke an ActionEvent on a Button using the keyboard, use the Space bar.

The object that implements the ActionListener interface gets this ActionEvent when the event occurs. The listener is therefore spared the details of processing individual mouse movements and mouse clicks, and can instead process a "meaningful" (semantic) event like "button pressed".

An unspecified behavior will be caused if the id parameter of any particular ActionEvent instance is not in the range from ACTION\_FIRST to ACTION\_LAST.

Since:

1.1

-----------import java.awt.event.ActionListener;-----------

public interface ActionListener

extends EventListener

The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method. When the action event occurs, that object's actionPerformed method is invoked.

Since:

1.1

-----------import java.awt.event.KeyEvent;-----------

public class KeyEvent

extends InputEvent

An event which indicates that a keystroke occurred in a component.

This low-level event is generated by a component object (such as a text field) when a key is pressed, released, or typed. The event is passed to every KeyListener or KeyAdapter object which registered to receive such events using the component's addKeyListener method. (KeyAdapter objects implement the KeyListener interface.) Each such listener object gets this KeyEvent when the event occurs.

-----------import javax.swing.JFileChooser;-----------

public class JFileChooser

extends JComponent

implements Accessible

JFileChooser provides a simple mechanism for the user to choose a file. For information about using JFileChooser, see How to Use File Choosers, a section in The Java Tutorial.

The following code pops up a file chooser for the user's home directory that sees only .jpg and .gif images:

JFileChooser chooser = new JFileChooser();

FileNameExtensionFilter filter = new FileNameExtensionFilter(

"JPG & GIF Images", "jpg", "gif");

chooser.setFileFilter(filter);

int returnVal = chooser.showOpenDialog(parent);

if(returnVal == JFileChooser.APPROVE\_OPTION) {

System.out.println("You chose to open this file: " +

chooser.getSelectedFile().getName());

}

-----------import javax.swing.JFrame;-----------

public class JFrame

extends Frame

implements WindowConstants, Accessible, RootPaneContainer

An extended version of java.awt.Frame that adds support for the JFC/Swing component architecture. You can find task-oriented documentation about using JFrame in The Java Tutorial, in the section How to Make Frames.

The JFrame class is slightly incompatible with Frame. Like all other JFC/Swing top-level containers, a JFrame contains a JRootPane as its only child. The content pane provided by the root pane should, as a rule, contain all the non-menu components displayed by the JFrame.

-----------import javax.swing.JMenu;-----------

public class JMenu

extends JMenuItem

implements Accessible, MenuElement

An implementation of a menu -- a popup window containing JMenuItems that is displayed when the user selects an item on the JMenuBar. In addition to JMenuItems, a JMenu can also contain JSeparators.

In essence, a menu is a button with an associated JPopupMenu. When the "button" is pressed, the JPopupMenu appears. If the "button" is on the JMenuBar, the menu is a top-level window. If the "button" is another menu item, then the JPopupMenu is "pull-right" menu.

Menus can be configured, and to some degree controlled, by Actions. Using an Action with a menu has many benefits beyond directly configuring a menu. Refer to Swing Components Supporting Action for more details, and you can find more information in How to Use Actions, a section in The Java Tutorial.

-----------import javax.swing.JMenuBar;-----------

public class JMenuBar

extends JComponent

implements Accessible, MenuElement

An implementation of a menu bar. You add JMenu objects to the menu bar to construct a menu. When the user selects a JMenu object, its associated JPopupMenu is displayed, allowing the user to select one of the JMenuItems on it.

For information and examples of using menu bars see How to Use Menus, a section in The Java Tutorial.

Warning: Swing is not thread safe. For more information see Swing's Threading Policy.

Warning: Serialized objects of this class will not be compatible with future Swing releases. The current serialization support is appropriate for short term storage or RMI between applications running the same version of Swing. As of 1.4, support for long term storage of all JavaBeansTM has been added to the java.beans package. Please see XMLEncoder.

See Also:

JMenu, JPopupMenu, JMenuItem

-----------import javax.swing.JMenuItem;-----------

public class JMenuItem

extends AbstractButton

implements Accessible, MenuElement

An implementation of an item in a menu. A menu item is essentially a button sitting in a list. When the user selects the "button", the action associated with the menu item is performed. A JMenuItem contained in a JPopupMenu performs exactly that function.

Menu items can be configured, and to some degree controlled, by Actions. Using an Action with a menu item has many benefits beyond directly configuring a menu item. Refer to Swing Components Supporting Action for more details, and you can find more information in How to Use Actions, a section in The Java Tutorial.

-----------import javax.swing.JOptionPane;-----------

public class JOptionPane

extends JComponent

implements Accessible

JOptionPane makes it easy to pop up a standard dialog box that prompts users for a value or informs them of something. For information about using JOptionPane, see How to Make Dialogs, a section in The Java Tutorial.

-----------import javax.swing.KeyStroke;-----------

public class KeyStroke

extends AWTKeyStroke

A KeyStroke represents a key action on the keyboard, or equivalent input device. KeyStrokes can correspond to only a press or release of a particular key, just as KEY\_PRESSED and KEY\_RELEASED KeyEvents do; alternately, they can correspond to typing a specific Java character, just as KEY\_TYPED KeyEvents do. In all cases, KeyStrokes can specify modifiers (alt, shift, control, meta, altGraph, or a combination thereof) which must be present during the action for an exact match.

KeyStrokes are used to define high-level (semantic) action events. Instead of trapping every keystroke and throwing away the ones you are not interested in, those keystrokes you care about automatically initiate actions on the Components with which they are registered.

-----------import javax.swing.filechooser.FileNameExtensionFilter;-----------

public final class FileNameExtensionFilter

extends FileFilter

An implementation of FileFilter that filters using a specified set of extensions. The extension for a file is the portion of the file name after the last ".". Files whose name does not contain a "." have no file name extension. File name extension comparisons are case insensitive.

The following example creates a FileNameExtensionFilter that will show jpg files:

FileFilter filter = new FileNameExtensionFilter("JPEG file", "jpg", "jpeg");

JFileChooser fileChooser = ...;

fileChooser.addChoosableFileFilter(filter);

Since:

1.6

-----------import java.awt.Dimension;-----------

public class Dimension

extends Dimension2D

implements Serializable

The Dimension class encapsulates the width and height of a component (in integer precision) in a single object. The class is associated with certain properties of components. Several methods defined by the Component class and the LayoutManager interface return a Dimension object.

Normally the values of width and height are non-negative integers. The constructors that allow you to create a dimension do not prevent you from setting a negative value for these properties. If the value of width or height is negative, the behavior of some methods defined by other objects is undefined.

Since:

1.0

-----------import javax.swing.JPanel;-----------

public class JPanel

extends JComponent

implements Accessible

JPanel is a generic lightweight container. For examples and task-oriented documentation for JPanel, see How to Use Panels, a section in The Java Tutorial.

Warning: Swing is not thread safe. For more information see Swing's Threading Policy.

-----------import javax.swing.JScrollPane;-----------

public class JScrollPane

extends JComponent

implements ScrollPaneConstants, Accessible

Provides a scrollable view of a lightweight component. A JScrollPane manages a viewport, optional vertical and horizontal scroll bars, and optional row and column heading viewports.

-----------import javax.swing.JTable;-----------

public class JTable

extends JComponent

implements TableModelListener, Scrollable, TableColumnModelListener, ListSelectionListener, CellEditorListener, Accessible, RowSorterListener

The JTable is used to display and edit regular two-dimensional tables of cells. See How to Use Tables in The Java Tutorial for task-oriented documentation and examples of using JTable.

The JTable has many facilities that make it possible to customize its rendering and editing but provides defaults for these features so that simple tables can be set up easily. For example, to set up a table with 10 rows and 10 columns of numbers:

TableModel dataModel = new AbstractTableModel() {

public int getColumnCount() { return 10; }

public int getRowCount() { return 10;}

public Object getValueAt(int row, int col) { return new Integer(row\*col); }

};

JTable table = new JTable(dataModel);

JScrollPane scrollpane = new JScrollPane(table);

-----------import javax.swing.JTextPane;-----------

public class JTextPane

extends JEditorPane

A text component that can be marked up with attributes that are represented graphically. You can find how-to information and examples of using text panes in Using Text Components, a section in The Java Tutorial.

This component models paragraphs that are composed of runs of character level attributes. Each paragraph may have a logical style attached to it which contains the default attributes to use if not overridden by attributes set on the paragraph or character run. Components and images may be embedded in the flow of text.

-----------import javax.swing.table.AbstractTableModel;-----------

public abstract class AbstractTableModel

extends Object

implements TableModel, Serializable

This abstract class provides default implementations for most of the methods in the TableModel interface. It takes care of the management of listeners and provides some conveniences for generating TableModelEvents and dispatching them to the listeners. To create a concrete TableModel as a subclass of AbstractTableModel you need only provide implementations for the following three methods:

public int getRowCount();

public int getColumnCount();

public Object getValueAt(int row, int column);

-----------import javax.swing.text.BadLocationException;-----------

public class BadLocationException

extends Exception

This exception is to report bad locations within a document model (that is, attempts to reference a location that doesn't exist).

Warning: Serialized objects of this class will not be compatible with future Swing releases. The current serialization support is appropriate for short term storage or RMI between applications running the same version of Swing. As of 1.4, support for long term storage of all JavaBeans™ has been added to the java.beans package. Please see XMLEncoder.

-----------import javax.swing.text.SimpleAttributeSet;-----------

public class SimpleAttributeSet

extends Object

implements MutableAttributeSet, Serializable, Cloneable

A straightforward implementation of MutableAttributeSet using a hash table.

Warning: Serialized objects of this class will not be compatible with future Swing releases. The current serialization support is appropriate for short term storage or RMI between applications running the same version of Swing. As of 1.4, support for long term storage of all JavaBeans™ has been added to the java.beans package. Please see XMLEncoder.

-----------import javax.swing.text.StyleConstants;-----------

public class StyleConstants

extends Object

A collection of well known or common attribute keys and methods to apply to an AttributeSet or MutableAttributeSet to get/set the properties in a typesafe manner.

The paragraph attributes form the definition of a paragraph to be rendered. All sizes are specified in points (such as found in postscript), a device independent measure.

-----------import javax.swing.text.StyledDocument;-----------

public interface StyledDocument

extends Document

Interface for a generic styled document.

-----------import java.awt.Component;-----------

public abstract class Component

extends Object

implements ImageObserver, MenuContainer, Serializable

A component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. Examples of components are the buttons, checkboxes, and scrollbars of a typical graphical user interface.

The Component class is the abstract superclass of the nonmenu-related Abstract Window Toolkit components. Class Component can also be extended directly to create a lightweight component. A lightweight component is a component that is not associated with a native window. On the contrary, a heavyweight component is associated with a native window. The isLightweight() method may be used to distinguish between the two kinds of the components.

-----------import java.awt.GridBagConstraints;-----------

public class GridBagConstraints

extends Object

implements Cloneable, Serializable

The GridBagConstraints class specifies constraints for components that are laid out using the GridBagLayout class.

Since:

JDK1.0

-----------import java.awt.GridBagLayout;-----------

public class GridBagLayout

extends Object

implements LayoutManager2, Serializable

The GridBagLayout class is a flexible layout manager that aligns components vertically, horizontally or along their baseline without requiring that the components be of the same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells, with each component occupying one or more cells, called its display area.

Each component managed by a GridBagLayout is associated with an instance of GridBagConstraints. The constraints object specifies where a component's display area should be located on the grid and how the component should be positioned within its display area. In addition to its constraints object, the GridBagLayout also considers each component's minimum and preferred sizes in order to determine a component's size.

-----------import java.awt.Insets;-----------

public class Insets

extends Object

implements Cloneable, Serializable

An Insets object is a representation of the borders of a container. It specifies the space that a container must leave at each of its edges. The space can be a border, a blank space, or a title.

Since:

JDK1.0

-----------import java.awt.event.KeyAdapter;-----------

public abstract class KeyAdapter

extends Object

implements KeyListener

An abstract adapter class for receiving keyboard events. The methods in this class are empty. This class exists as convenience for creating listener objects.

Extend this class to create a KeyEvent listener and override the methods for the events of interest. (If you implement the KeyListener interface, you have to define all of the methods in it. This abstract class defines null methods for them all, so you can only have to define methods for events you care about.)

Create a listener object using the extended class and then register it with a component using the component's addKeyListener method. When a key is pressed, released, or typed, the relevant method in the listener object is invoked, and the KeyEvent is passed to it.

Since:

1.1

-----------import java.awt.event.KeyEvent;-----------

public class KeyEvent

extends InputEvent

An event which indicates that a keystroke occurred in a component.

This low-level event is generated by a component object (such as a text field) when a key is pressed, released, or typed. The event is passed to every KeyListener or KeyAdapter object which registered to receive such events using the component's addKeyListener method. (KeyAdapter objects implement the KeyListener interface.) Each such listener object gets this KeyEvent when the event occurs.

-----------import javax.swing.BorderFactory;-----------

public class BorderFactory

extends Object

Factory class for vending standard Border objects. Wherever possible, this factory will hand out references to shared Border instances. For further information and examples see How to Use Borders, a section in The Java Tutorial.

-----------import javax.swing.Box;-----------

public class Box

extends JComponent

implements Accessible

A lightweight container that uses a BoxLayout object as its layout manager. Box provides several class methods that are useful for containers using BoxLayout -- even non-Box containers.

The Box class can create several kinds of invisible components that affect layout: glue, struts, and rigid areas. If all the components your Box contains have a fixed size, you might want to use a glue component (returned by createGlue) to control the components' positions. If you need a fixed amount of space between two components, try using a strut (createHorizontalStrut or createVerticalStrut). If you need an invisible component that always takes up the same amount of space, get it by invoking createRigidArea.

-----------import javax.swing.ButtonGroup;-----------

public class ButtonGroup

extends Object

implements Serializable

This class is used to create a multiple-exclusion scope for a set of buttons. Creating a set of buttons with the same ButtonGroup object means that turning "on" one of those buttons turns off all other buttons in the group.

A ButtonGroup can be used with any set of objects that inherit from AbstractButton. Typically a button group contains instances of JRadioButton, JRadioButtonMenuItem, or JToggleButton. It wouldn't make sense to put an instance of JButton or JMenuItem in a button group because JButton and JMenuItem don't implement the selected state.

-----------import javax.swing.DefaultComboBoxModel;-----------

public class DefaultComboBoxModel<E>

extends AbstractListModel<E>

implements MutableComboBoxModel<E>, Serializable

The default model for combo boxes.

Since:

1.2

See Also:

Serialized Form

-----------import javax.swing.DefaultListModel;-----------

public class DefaultListModel<E>

extends AbstractListModel<E>

This class loosely implements the java.util.Vector API, in that it implements the 1.1.x version of java.util.Vector, has no collection class support, and notifies the ListDataListeners when changes occur. Presently it delegates to a Vector, in a future release it will be a real Collection implementation.

Warning: Serialized objects of this class will not be compatible with future Swing releases. The current serialization support is appropriate for short term storage or RMI between applications running the same version of Swing. As of 1.4, support for long term storage of all JavaBeans™ has been added to the java.beans package. Please see XMLEncoder.

Since:

1.2

-----------import javax.swing.JButton;-----------

public class JButton

extends AbstractButton

implements Accessible

An implementation of a "push" button.

Buttons can be configured, and to some degree controlled, by Actions. Using an Action with a button has many benefits beyond directly configuring a button. Refer to Swing Components Supporting Action for more details, and you can find more information in How to Use Actions, a section in The Java Tutorial.

-----------import javax.swing.JComboBox;-----------

public class JComboBox<E>

extends JComponent

implements ItemSelectable, ListDataListener, ActionListener, Accessible

A component that combines a button or editable field and a drop-down list. The user can select a value from the drop-down list, which appears at the user's request. If you make the combo box editable, then the combo box includes an editable field into which the user can type a value.

-----------import javax.swing.JLabel;-----------

public class JLabel

extends JComponent

implements SwingConstants, Accessible

A display area for a short text string or an image, or both. A label does not react to input events. As a result, it cannot get the keyboard focus. A label can, however, display a keyboard alternative as a convenience for a nearby component that has a keyboard alternative but can't display it.

A JLabel object can display either text, an image, or both. You can specify where in the label's display area the label's contents are aligned by setting the vertical and horizontal alignment. By default, labels are vertically centered in their display area. Text-only labels are leading edge aligned, by default; image-only labels are horizontally centered, by default.

-----------import javax.swing.JPanel;-----------

public class JPanel

extends JComponent

implements Accessible

JPanel is a generic lightweight container. For examples and task-oriented documentation for JPanel, see How to Use Panels, a section in The Java Tutorial.

Warning: Swing is not thread safe. For more information see Swing's Threading Policy.

Warning: Serialized objects of this class will not be compatible with future Swing releases. The current serialization support is appropriate for short term storage or RMI between applications running the same version of Swing. As of 1.4, support for long term storage of all JavaBeansTM has been added to the java.beans package. Please see XMLEncoder.

-----------import javax.swing.JRadioButton;-----------

public class JRadioButton

extends JToggleButton

implements Accessible

An implementation of a radio button -- an item that can be selected or deselected, and which displays its state to the user. Used with a ButtonGroup object to create a group of buttons in which only one button at a time can be selected. (Create a ButtonGroup object and use its add method to include the JRadioButton objects in the group.)

-----------import javax.swing.JTextField;-----------

public class JTextField

extends JTextComponent

implements SwingConstants

JTextField is a lightweight component that allows the editing of a single line of text. For information on and examples of using text fields, see How to Use Text Fields in The Java Tutorial.

JTextField is intended to be source-compatible with java.awt.TextField where it is reasonable to do so. This component has capabilities not found in the java.awt.TextField class. The superclass should be consulted for additional capabilities.

JTextField has a method to establish the string used as the command string for the action event that gets fired. The java.awt.TextField used the text of the field as the command string for the ActionEvent. JTextField will use the command string set with the setActionCommand method if not null, otherwise it will use the text of the field as a compatibility with java.awt.TextField.

The method setEchoChar and getEchoChar are not provided directly to avoid a new implementation of a pluggable look-and-feel inadvertently exposing password characters. To provide password-like services a separate class JPasswordField extends JTextField to provide this service with an independently pluggable look-and-feel.

-----------import java.util.EventObject;-----------

public class EventObject

extends Object

implements Serializable

The root class from which all event state objects shall be derived.

All Events are constructed with a reference to the object, the "source", that is logically deemed to be the object upon which the Event in question initially occurred upon.

Since:

JDK1.1

-----------import java.util.EventListener;-----------

public interface EventListener

A tagging interface that all event listener interfaces must extend.

Since:

JDK1.1

-----------import java.text.DateFormat;---------------

public abstract class DateFormat

extends Format

DateFormat is an abstract class for date/time formatting subclasses which formats and parses dates or time in a language-independent manner. The date/time formatting subclass, such as SimpleDateFormat, allows for formatting (i.e., date -> text), parsing (text -> date), and normalization. The date is represented as a Date object or as the milliseconds since January 1, 1970, 00:00:00 GMT.

DateFormat provides many class methods for obtaining default date/time formatters based on the default or a given locale and a number of formatting styles. The formatting styles include FULL, LONG, MEDIUM, and SHORT. More detail and examples of using these styles are provided in the method descriptions.

DateFormat helps you to format and parse dates for any locale. Your code can be completely independent of the locale conventions for months, days of the week, or even the calendar format: lunar vs. solar.

To format a date for the current Locale, use one of the static factory methods:

myString = DateFormat.getDateInstance().format(myDate);

-----------import java.text.SimpleDateFormat;---------------

public class SimpleDateFormat

extends DateFormat

SimpleDateFormat is a concrete class for formatting and parsing dates in a locale-sensitive manner. It allows for formatting (date -> text), parsing (text -> date), and normalization.

SimpleDateFormat allows you to start by choosing any user-defined patterns for date-time formatting. However, you are encouraged to create a date-time formatter with either getTimeInstance, getDateInstance, or getDateTimeInstance in DateFormat.

Each of these class methods can return a date/time formatter initialized with a default format pattern. You may modify the format pattern using the applyPattern methods as desired. For more information on using these methods, see DateFormat.

-----------import java.util.logging.Level;---------------

public class Level

extends Object

implements Serializable

The Level class defines a set of standard logging levels that can be used to control logging output.

The logging Level objects are ordered and are specified by ordered integers.

Enabling logging at a given level also enables logging at all higher levels.

Clients should normally use the predefined Level constants such as Level.SEVERE.

-----------import java.util.logging.Logger;---------------

public class Logger

extends Object

A Logger object is used to log messages for a specific system or application component.

Loggers are normally named, using a hierarchical dot-separated namespace.

Logger names can be arbitrary strings, but they should normally be based on the package name or class name of the logged component, such as java.net or javax.swing.

In addition it is possible to create "anonymous" Loggers that are not stored in the Logger namespace.

Logger objects may be obtained by calls on one of the getLogger factory methods. These will either create a new Logger or return a suitable existing Logger.

It is important to note that the Logger returned by one of the getLogger factory methods may be garbage collected at any time if a strong reference to the Logger is not kept.

Logging messages will be forwarded to registered Handler objects, which can forward the messages to a variety of destinations, including consoles, files, OS logs, etc.

Each Logger keeps track of a "parent" Logger, which is its nearest existing ancestor in the Logger namespace.

Each Logger has a "Level" associated with it. This reflects a minimum Level that this logger cares about.

If a Logger's level is set to null, then its effective level is inherited from its parent, which may in turn obtain it recursively from its parent, and so on up the tree.