**Форма отчета обучающегося о прохождении практики**

МИНОБРНАУКИ РОССИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**“ТВЕРСКОЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”**

**(ФГБОУ ВО «ТГУ»)**

Математический факультет

Кафедра Компьютерной безопасности и

математических методов управления

**ОТЧЕТ**

 по *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* практике

*указать вид практики*

студента \_\_ курса \_\_ группы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*ФИО (полностью)*

Специальность 10.05.01 Компьютерная безопасность

Специализация «Математические методы защиты информации»

Зав. кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_.\_\_.2017

*Подпись, расшифровка, ученая степень, звание*

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_.\_\_.2017

*Подпись, расшифровка подписи*

Руководитель ООП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.\_\_.2017

*Подпись, расшифровка подписи, ученая степень, звание*

Руководитель практики от предприятия

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_.\_\_.2017

*Подпись, расшифровка подписи, ученая степень, звание*

# Содержание

1. Введение……………………………………………………………………3
	1. Характеристика предприятия …………………………………………3
	2. Цели прохождения практики …………………………………….……4
2. Основная часть …………………………………………………………….5
	1. Задачи, решаемые в рамках практики………………………...……….5
	2. Теоретические и практические аспекты………………………………6
		1. Задание по SQL…………………………………………………………6
		2. Задание по Java ………………………………………………………....7
3. Заключение………………………………………………………………..10
4. Список источников…………………………………………………….....11
5. Приложения……………………………………………………………….12
	1. Приложение 1………………………………………………………….12
	2. Приложение 2………………………………………………………….15
	3. Приложение 3………………………………………………………….20

# 1. Введение

# 1.1. Характеристика предприятия

**Объектом производственной практики** является консалтинговая компания Accenture. Работая с клиентами в 120 странах мира, компания решает задачи в области стратегии, цифровизации бизнеса, технологий и операций, решает вопросы по оптимизации и организации аутсорсинга бизнес-процессов, управления взаимоотношениями с клиентами, управления логистическими процессами, управления персоналом. Штат компании на 2016 год составил 380 тыс. сотрудников.

За время существования Accenture в России было реализовано более 1000 проектов для лидеров российского бизнеса. Accenture работает во всех секторах экономики: нефть и газ, машиностроение, металлургия, электроэнергетика, телекоммуникации, потребительские товары и услуги, банковский и инвестиционный бизнес, пищевая, фармацевтическая, автомобильная промышленность и т.д. [1]

# 1.2. Цели прохождения практики

**Целями** прохождения практики являются:

* ознакомление с деятельностью организации и её структурными подразделениями;
* изучение особенностей работы в компании;
* приобретение опыта работы в коллективе;
* получение практического опыта работы;
* закрепление полученных знаний по общим и специальным дисциплинам.

**Дата начала практики:** 16 июня 2017 г.

**Дата окончания:** 30 июня 2017 г.

# 2. Основная часть

# 2.1. Задачи, решаемые в рамках практики

В рамках практики требовалось выполнить два практических задания.

Первое задание - по SQL - включает в себя установку базы данных Oracle XE, приложения SQL Developer, создание подключения из SQL Developer к базе данных и написание запросов к базе данных на SQL. Задание разделено на две части: установка необходимых компонентов базы данных и выполнение различных операций с ней.

Второе задание - по Java – тоже состоит из двух частей: установка интегрированной среды разработки Eclipse и написание простого оконного приложения на Java.

# 2.2. Теоретические и практические аспекты

# 2.2.1. Задание по SQL

В соответствии с требованиями, задание по SQL выполняется на основе Oracle Database 11*g* Express Edition (**Oracle Database XE**) - СУБД начального уровня, основанной на программном коде СУБД Oracle Database 11*g* Release 2.  Данная СУБД бесплатна для разработки, развертывания и продажи, быстро скачивается и проста в администрировании.

Встраивание в СУБД Oracle JavaVM и полномасштабная поддержка серверных технологий привели к стремительному росту популярности данной СУБД в среде Internet.

Еще одной составляющей успеха СУБД Oracle является кроссплатформенность, так как она поставляется практически для всех существующих на сегодня операционных систем. Таким образом, компаниям, начинающим работу с продуктами Oracle не приходится менять уже сложившееся сетевое окружение. [2]

Oracle Database XE может быть установлена на любой компьютер с любым количеством процессоров (одна база данных на машину), но с ограничением в 4Гб пользовательских данных, использует не более 1Гб оперативной памяти и только один из имеющихся процессоров. [3]

Также для выполнения запросов устанавливается **Oracle SQL Developer**— [среда разработки](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) на языках [SQL](https://ru.wikipedia.org/wiki/SQL) и [PL/SQL](https://ru.wikipedia.org/wiki/PL/SQL), специально ориентированная на применение в среде [Oracle Database](https://ru.wikipedia.org/wiki/Oracle_Database).

Установка всех элементов и создание подключений между ними производится в соответствии с официальными инструкциями (<http://docs.oracle.com/cd/E17781_01/install.112/e18803.pdf>, http://docs.oracle.com/cd/E17781\_01/admin.112/e18585.pdf).

Подключение к базе данных проводится через пользователя HR (Human Resources). У данного пользователя уже существует большое количество таблиц, заполненные данными для учебных целей.

Приложение 1 можно использовать в качестве образца для составления SQL-запросов к базе данных. Выполнение практического задания по SQL приведено в приложении 2.

# 2.2.1. Задание по Java

Задание выполнено в IDE **Eclipse**. Eclipse - свободная интегрированная среда разработки модульных кроссплатформенных приложений. Развивается и поддерживается Eclipse Foundation. Известна в качестве платформы для написания расширений (причём любой разработчик может расширить Eclipse своими модулями) и диспетчерами для работы с базами данных, серверами приложений и т.д.

Eclipse JDT (Java Development Tools) — наиболее известный модуль, нацеленный на групповую разработку: среда интегрирована с системами управления версиями — [CVS](https://ru.wikipedia.org/wiki/CVS), [GIT](https://ru.wikipedia.org/wiki/GIT), для других систем существуют плагины. Также предлагает поддержку связи между IDE и системой управления задачами (ошибками). В основной поставке включена поддержка трекера ошибок [Bugzilla](https://ru.wikipedia.org/wiki/Bugzilla), также имеется множество расширений для поддержки других трекеров ([Trac](https://ru.wikipedia.org/wiki/Trac), [Jira](https://ru.wikipedia.org/wiki/Jira) и др.). В силу бесплатности и высокого качества, Eclipse во многих организациях является корпоративным стандартом для разработки приложений. [4]

В этом задании была использована библиотека **Swing** – это модуль для создания [графического интерфейса](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F) для программ на языке [Java](https://ru.wikipedia.org/wiki/Java). Swing содержит в себе такие графические компоненты, как кнопки, поля ввода, таблицы, формы и т.д.

Swing относится к библиотеке классов JFC – это набор библиотек для разработки графических оболочек. К ним относятся Java 2D, Accessibility-API, Drag & Drop-API и AWT.

В сравнении с более ранней библиотекой AWT, Swing предоставляет более гибкие интерфейсные компоненты. В отличие от AWT, компоненты Swing разработаны для одинаковой кросс-платформенной работы, в то время как компоненты AWT повторяют интерфейс исполняемой платформы без изменений. AWT же использует только стандартные элементы ОС для отображения, то есть для каждого элемента создается отдельный объект ОС (окно), в связи с чем, AWT не позволяет создавать элементы произвольной формы (возможно использовать только прямоугольные компоненты), элементы управления на основе AWT всегда отображаются поверх Swing-элементов (так как все Swing компоненты отображаются на поверхности контейнера). [5]

Для группировки компонент интерфейса Swing использует контейнеры (Container). Для создания основного контейнера для приложения чаще всего используется контейнер JFrame (также присутствуют JWindows и JApplet). Поэтому для создания компонент монжо унаследовать класс от JFrame и тем самым получить доступ ко множеству методов, например:

setBounds(x, y, w, h) - указывает координаты верхней левой вершины окна, а также его ширину и высоту.

setResizable(bool) - указывает, можно ли изменять размер окна.

setTitle(str) - устанавливает название окна.

setVisible(bool) - отображает окно при выводе на экран.

setDefaultCloseOperation(operation) - указывает операцию, которая будет произведена при закрытии окна.

Основные элементы управления:

JLabel - элемент для отображения фиксированного текста;

JButton - обычная кнопка (button);

JCheckBox - элемент выбора (аналог checkbox);

JRadioButton - радиокнопка

При отображении элементов управления используются специальные менеджеры - LayoutManager. У всех LayoutManager'ов есть методы для добавления у удаления элементов.

FlowLayout - используется для последовательного отображения элементов. Если элемент не помещается в конкретную строку, он отображается в следующей.

GridLayout - отображения элементов в виде таблицы с одинаковыми размерами ячеек.

BorderLayout - используется при отображении не более 5 элементов. Эти элементы располагаются по краям фрейма и в центре: North, South, East, West, Center.

BoxLayout - отображает элементы в виде рядка или колонки.

GridBagLayout - позволяет назначать месторасположение и размер каждого виджета. Это самый сложный, но и самый эффективный вид отображения.

Для обработки событий используются так называемые Event Listeners. [6][7]

Программы по созданию окон и результаты их работы представлены в Приложении 3.

# 3. Заключение

В процессе прохождения практики было осуществлено знакомство с компанией Accenture, областями и направлениями её деятельности, структурой и организацией предприятия. Также были получены теоретические и практические знания по базе данных Oracle и работе с языком Java.

# 4. Список источников

1. Accenture . https://ru.wikipedia.org/wiki/Accenture
2. Характеристика СУБД Oracle. http://www.omega.ru/oracleinfo.html
3. http://www.oracle.com/technetwork/ru/database/express-edition/overview/index.html
4. Eclipse. https://ru.wikipedia.org/wiki/Eclipse \_(среда\_разработки)
5. Swing. https://ru.wikipedia.org/wiki/Swing
6. Графический интерфейс на Java Swing - GUI Tutorial. http://www.javenue.info/post/36
7. Иван Портянкин. Swing: Эффектные пользовательские интерфейсы (2005)

# 5. Приложения

# 5.1. Приложение 1

В данном приложении будут описаны основные команды SQL , использовавшиеся при выполнении задания и в ходе теоретической подготовки.

1. Посмотреть структуру таблицы:

DESC имя\_таблицы;

2. Создать таблицу table1:

CREATE TABLE table1 (
field1  VARCHAR2(25) NOT NULL,
field2 NUMBER(4,2) NOT NULL,
field3 DATE
);

3. Вставить данные в таблицу table1:

INSERT INTO table1 VALUES (‘Data1′, 1, ’5-NOV-2000′);
INSERT INTO table1 VALUES (‘Data2′, 2.5, ’29-JUN-2001′);
INSERT INTO table1 VALUES (‘Data3′, 50.75, ’10-DEC-2002′);
INSERT INTO table1 VALUES (‘Data4′, 1, NULL);

4. Выбрать все поля (столбцы)  из таблицы table1:

SELECT \* FROM table1;

5. Выбрать некоторые поля из таблицы table1:

SELECT field1, field2 FROM table1;

6. Удалить таблицу table1:

DROP TABLE table1;

7. Выбрать поля из таблицы table1 с условием:

SELECT столбцы FROM имя\_таблицы WHERE условие(я);

8. Отсортировать по отдельным столбцам:

SELECT \* FROM имя\_таблицы ORDER BY столбец\_сортировки ASC(DESC);

где, ASC — возрастающая сортировка, а DESC — убывающая сортировка.

9. Изменить данные в таблице table1:

UPDATE имя\_таблицы SET имя\_столбца = новое\_значение WHERE условие;

10. Удаление записей из таблицы table1:

DELETE FROM имя\_таблщы WHERE условие;

11. Удаление всех записей таблицы:

DELETE FROM имя\_таблщы WHERE условие;

или

TRUNCATE TABLE имя\_таблицы;

12. Откат одной и более транзакций:

ROLLBACK TO имя\_точки \_отката;

13. Откат в  предварительно установленную точку:

SAVEPOINT имя\_точки\_сохранения;

14. Выравнивания количества десятичных знаков:

COLUMN имя\_столбца FORMAT код\_формата;

15. Форматирование заголовков столбцов:

COLUMN имя\_столбца HEADING текст\_заголовка JUSTIFY LEFT;

или

COLUMN имя\_столбца HEADING текст\_заголовка JUSTIFY CENTER;

или

COLUMN имя\_столбца HEADING текст\_заголовка JUSTIFY RIGHT;

16. Отключения форматирования, заданного командой COLUMN:

COLUMN имя\_столбца OFF;

17. Буферизация выходных данных на диске (процесс записи информации в дисковый файл):

SPOOL имя\_буферного\_файла;

SPOOL c:\test1.sql
SELECT \* FROM table1;
SELECT \* FROM table2;
SPOOL OFF

# 5.2. Приложение 2

Запросы к базе данных.

Команда SQL, которая:

1. выведет всех сотрудников из отдела с ID = 90

select EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME

 FROM EMPLOYEES

 WHERE (DEPARTMENT\_ID=90);



2. выведет сотрудников с зарплатой больше 10000, и у которых есть комиссионный процент

select FIRST\_NAME, LAST\_NAME

 FROM EMPLOYEES

 WHERE (SALARY>10000) and (COMMISSION\_PCT is not NULL);



3. выведет всех сотрудников, имена которых начинаются на ‘Ma’

select FIRST\_NAME, LAST\_NAME

 FROM EMPLOYEES

 WHERE FIRST\_NAME LIKE 'Ma%'



4. выведет всех сотрудников с названиями отделов, в которых они работают

select FIRST\_NAME, LAST\_NAME,DEPARTMENTS.DEPARTMENT\_NAME

 FROM EMPLOYEES JOIN DEPARTMENTS ON

 (EMPLOYEES.DEPARTMENT\_ID=DEPARTMENTS.DEPARTMENT\_ID);



5. выведет всех сотрудников с сортировкой их по возрасту в порядке убывания

select FIRST\_NAME, LAST\_NAME, HIRE\_DATE

 FROM EMPLOYEES

 order by HIRE\_DATE desc;



6. выведет для каждого сотрудника название города и региона, в котором располагается его отдел

select EMPLOYEES.FIRST\_NAME, EMPLOYEES.LAST\_NAME, LOCATIONS.CITY, REGIONS.REGION\_ID

 FROM EMPLOYEES CROSS JOIN LOCATIONS CROSS JOIN DEPARTMENTS CROSS JOIN REGIONS CROSS JOIN COUNTRIES

 WHERE (EMPLOYEES.DEPARTMENT\_ID = DEPARTMENTS.DEPARTMENT\_ID)

 AND (DEPARTMENTS.LOCATION\_ID = LOCATIONS.LOCATION\_ID)

 AND (COUNTRIES.REGION\_ID = REGIONS.REGION\_ID)

 AND (LOCATIONS.COUNTRY\_ID = COUNTRIES.COUNTRY\_ID);



7. выведет количество сотрудников для каждого отдела (использовать group by)

select count(EMPLOYEE\_ID), DEPARTMENT\_ID

from EMPLOYEES

group by DEPARTMENT\_ID

order by DEPARTMENT\_ID;



8. создаст таблицу для хранения данных о проектах. У каждого проекта есть краткое описание и дата начала и завершения проекта

CREATE TABLE PROJECTS

(

PROJECT\_ID number(5) NOT NULL primary key,

PROJECTS\_DESCRIPTION VARCHAR(55) NOT NULL,

PROJECTS\_STARTDATE VARCHAR(25) NOT NULL,

PROJECTS\_ENDDATE VARCHAR(25) NOT NULL

);

9. добавит в таблицу с проектами новый столбец для хранения перечня технологий, которые используются на проекте

alter table PROJECTS add PROJECTS\_TECNOLOJIES VARCHAR(25);

10. удалит таблицу с проектами

drop table PROJECTS;

# 5.3. Приложение 3

## ConnectAndExecuteExample.java

package com.accenture.practice.jdbc;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.LinkedList;

import java.util.List;

import java.util.Locale;

public class ConnectAndExecuteExample {

 private static final int RESULT\_COLUMNS\_COUNT = 3;

 final private static String driverName = "oracle.jdbc.driver.OracleDriver";

 private static String url;

 final private static String server = "localhost";

 final private static String port = "1521";

 final private static String sid = "xe";

 final private static String username = "HR";

 final private static String password = "Qwerty1234";

 public static void main(String[] args) {

 Locale.setDefault(Locale.ENGLISH);

 Object[][] data = connectAndExecute(server, port, sid, username,

 password);

 System.out.println("Result data:");

 if (data != null) {

 for (int i = 0; i < data.length; i++) {

 System.out.println("firstName = " + data[i][0] + "; lastName = "

 + data[i][1] + "; departmentId = " + data[i][2]);

 }

 }

 }

 public static Object[][] connectAndExecute(String server, String port,

 String sid, String user, String password) {

 ResultSet rs = null;

 Statement st = null;

 Connection connection = null;

 List<Object[]> result = new LinkedList<Object[]>();

 try {

 url = "jdbc:oracle:thin:@" + server + ":" + port + ":" + sid;

 Class.forName(driverName);

 connection = DriverManager.getConnection(url, username, password);

 System.out.println("connecting: " + url);

 st = connection.createStatement();

 String sql = String.format("SELECT e.first\_name, e.last\_name, e.department\_id FROM employees e %s",

 "where e.department\_id = 90");

 System.out.println("SQL: " + sql);

 rs = st.executeQuery(sql);

 while (rs.next()) {

 Object[] resultRow = new Object[3];

 resultRow[0] = rs.getString(1);

 resultRow[1] = rs.getString(2);

 resultRow[2] = rs.getString(3);

 result.add(resultRow);

 }

 } catch (ClassNotFoundException e) {

 System.out.println("ClassNotFoundException");

 e.printStackTrace();

 } catch (SQLException e) {

 System.out.println("SQLException\n" + e.getMessage());

 e.printStackTrace();

 } catch(Exception e){

 e.printStackTrace();

 } finally {

 try {

 if (rs != null) {

 rs.close();

 }

 if (st != null) {

 st.close();

 }

 if (connection != null) {

 connection.close();

 }

 } catch (Exception e) {

 System.out.println("Other Exception on close: "

 + e.getMessage());

 e.printStackTrace();

 }

 }

 return result.toArray(new Object[result.size()][RESULT\_COLUMNS\_COUNT]);

 }

}

## QuitButtonExample.java

package com.accenture.practice.swing;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.SwingUtilities;

public class QuitButtonExample extends JFrame {

 private static final long serialVersionUID = -4882115815033643770L;

 public QuitButtonExample() {

 initUI();

 }

 private void initUI() {

 JPanel panel = new JPanel();

 getContentPane().add(panel);

 panel.setLayout(null);

 JButton quitButton = new JButton("Quit");

 quitButton.setBounds(50, 60, 80, 30);

 quitButton.addActionListener(new ActionListener() {

 @Override

 public void actionPerformed(ActionEvent event) {

 System.exit(0);

 }

 });

 panel.add(quitButton);

 setTitle("Quit button");

 setSize(300, 200);

 setLocationRelativeTo(null);

 setDefaultCloseOperation(EXIT\_ON\_CLOSE);

 }

 public static void main(String[] args) {

 SwingUtilities.invokeLater(new Runnable() {

 @Override

 public void run() {

 QuitButtonExample ex = new QuitButtonExample();

 ex.setVisible(true);

 }

 });

 }

}



## SimpleExample.java

package com.accenture.practice.swing;

import javax.swing.JFrame;

import javax.swing.SwingUtilities;

public class SimpleExample extends JFrame {

 private static final long serialVersionUID = -3911799410331330219L;

 public SimpleExample() {

 setTitle("Simple example");

 setSize(300, 200);

 setLocationRelativeTo(null);

 setDefaultCloseOperation(EXIT\_ON\_CLOSE);

 }

 public static void main(String[] args) {

 SwingUtilities.invokeLater(new Runnable() {

 @Override

 public void run() {

 SimpleExample ex;

 ex = new SimpleExample();

 ex.setVisible(true);

 }

 });

 }

}



## TooltipExample.java

package com.accenture.practice.swing;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.SwingUtilities;

public class TooltipExample extends JFrame {

 private static final long serialVersionUID = -4161843805224389004L;

 public TooltipExample() {

 initUI();

 }

 private void initUI() {

 JPanel panel = new JPanel();

 getContentPane().add(panel);

 panel.setLayout(null);

 panel.setToolTipText("A Panel container");

 JButton btn = new JButton("Button");

 btn.setBounds(100, 60, 100, 30);

 btn.setToolTipText("A Button component");

 panel.add(btn);

 setTitle("Tooltip");

 setSize(300, 200);

 setLocationRelativeTo(null);

 setDefaultCloseOperation(EXIT\_ON\_CLOSE);

 }

 public static void main(String[] args) {

 SwingUtilities.invokeLater(new Runnable() {

 @Override

 public void run() {

 TooltipExample ex = new TooltipExample();

 ex.setVisible(true);

 }

 });

 }

}



## ListExample.java

package com.accenture.practice.swing.components;

import java.awt.BorderLayout;

import java.awt.Dimension;

import java.awt.Font;

import java.awt.GraphicsEnvironment;

import javax.swing.BorderFactory;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JList;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.SwingUtilities;

import javax.swing.event.ListSelectionEvent;

import javax.swing.event.ListSelectionListener;

public class ListExample extends JFrame {

 private static final long serialVersionUID = -5016350577424621456L;

 private JLabel label;

 private JList<Object> list;

 public ListExample() {

 initUI();

 }

 private void initUI() {

 JPanel panel = new JPanel();

 panel.setLayout(new BorderLayout());

 panel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

 GraphicsEnvironment ge = GraphicsEnvironment

 .getLocalGraphicsEnvironment();

 String[] fonts = ge.getAvailableFontFamilyNames();

 list = new JList<Object>(fonts);

 list.addListSelectionListener(new ListSelectionListener() {

 @Override

 public void valueChanged(ListSelectionEvent e) {

 if (!e.getValueIsAdjusting()) {

 String name = (String) list.getSelectedValue();

 Font font = new Font(name, Font.PLAIN, 12);

 label.setFont(font);

 }

 }

 });

 JScrollPane pane = new JScrollPane();

 pane.getViewport().add(list);

 pane.setPreferredSize(new Dimension(250, 200));

 panel.add(pane);

 label = new JLabel("Aguirre, der Zorn Gottes");

 label.setFont(new Font("Serif", Font.PLAIN, 12));

 add(label, BorderLayout.SOUTH);

 add(panel);

 pack();

 setTitle("JList");

 setDefaultCloseOperation(EXIT\_ON\_CLOSE);

 setLocationRelativeTo(null);

 }

 public static void main(String[] args) {

 SwingUtilities.invokeLater(new Runnable() {

 @Override

 public void run() {

 ListExample ex = new ListExample();

 ex.setVisible(true);

 }

 });

 }

}



## TextAreaExample.java

package com.accenture.practice.swing.components;

import java.awt.BorderLayout;

import java.awt.Dimension;

import javax.swing.BorderFactory;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTextArea;

import javax.swing.SwingUtilities;

public class TextAreaExample extends JFrame {

 private static final long serialVersionUID = 8935491495025991259L;

 public TextAreaExample() {

 initUI();

 }

 private void initUI() {

 JPanel panel = new JPanel();

 panel.setLayout(new BorderLayout());

 panel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

 JScrollPane pane = new JScrollPane();

 JTextArea area = new JTextArea();

 area.setLineWrap(true);

 area.setWrapStyleWord(true);

 area.setBorder(BorderFactory.createEmptyBorder(8, 8, 8, 8));

 pane.getViewport().add(area);

 panel.add(pane);

 add(panel);

 setTitle("JTextArea");

 setSize(new Dimension(350, 300));

 setDefaultCloseOperation(EXIT\_ON\_CLOSE);

 setLocationRelativeTo(null);

 }

 public static void main(String[] args) {

 SwingUtilities.invokeLater(new Runnable() {

 @Override

 public void run() {

 TextAreaExample ex = new TextAreaExample();

 ex.setVisible(true);

 }

 });

 }

}



## ToggleButtonExample.java

package com.accenture.practice.swing.components;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.BorderFactory;

import javax.swing.Box;

import javax.swing.BoxLayout;

import javax.swing.JDialog;

import javax.swing.JPanel;

import javax.swing.JToggleButton;

import javax.swing.SwingUtilities;

import javax.swing.border.LineBorder;

public class ToggleButtonExample extends JDialog implements ActionListener {

 private static final long serialVersionUID = 6853574057205125099L;

 private JToggleButton redButton;

 private JToggleButton greenButton;

 private JToggleButton blueButton;

 private JPanel display;

 public ToggleButtonExample() {

 initUI();

 }

 private void initUI() {

 JPanel bottom = new JPanel();

 bottom.setLayout(new BoxLayout(bottom, BoxLayout.X\_AXIS));

 bottom.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

 JPanel leftPanel = new JPanel();

 leftPanel.setLayout(new BoxLayout(leftPanel, BoxLayout.Y\_AXIS));

 redButton = new JToggleButton("red");

 redButton.addActionListener(this);

 greenButton = new JToggleButton("green");

 greenButton.addActionListener(this);

 blueButton = new JToggleButton("blue");

 blueButton.addActionListener(this);

 blueButton.setMaximumSize(greenButton.getMaximumSize());

 redButton.setMaximumSize(greenButton.getMaximumSize());

 leftPanel.add(redButton);

 leftPanel.add(Box.createRigidArea(new Dimension(25, 7)));

 leftPanel.add(greenButton);

 leftPanel.add(Box.createRigidArea(new Dimension(25, 7)));

 leftPanel.add(blueButton);

 bottom.add(leftPanel);

 bottom.add(Box.createRigidArea(new Dimension(20, 0)));

 display = new JPanel();

 display.setPreferredSize(new Dimension(110, 110));

 display.setBorder(LineBorder.createGrayLineBorder());

 display.setBackground(Color.black);

 bottom.add(display);

 add(bottom);

 pack();

 setTitle("JToggleButton");

 setResizable(false);

 setLocationRelativeTo(null);

 setDefaultCloseOperation(JDialog.DISPOSE\_ON\_CLOSE);

 }

 @Override

 public void actionPerformed(ActionEvent e) {

 Color color = display.getBackground();

 int red = color.getRed();

 int green = color.getGreen();

 int blue = color.getBlue();

 if (e.getActionCommand().equals("red")) {

 if (red == 0) {

 red = 255;

 } else {

 red = 0;

 }

 }

 if (e.getActionCommand().equals("green")) {

 if (green == 0) {

 green = 255;

 } else {

 green = 0;

 }

 }

 if (e.getActionCommand().equals("blue")) {

 if (blue == 0) {

 blue = 255;

 } else {

 blue = 0;

 }

 }

 Color setCol = new Color(red, green, blue);

 display.setBackground(setCol);

 }

 public static void main(String[] args) {

 SwingUtilities.invokeLater(new Runnable() {

 @Override

 public void run() {

 ToggleButtonExample ex = new ToggleButtonExample();

 ex.setVisible(true);

 }

 });

 }

}

 

## RefreshableTableDemo.java

package com.accenture.practice.swing.table;

import java.awt.Dimension;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.table.AbstractTableModel;

public class RefreshableTableDemo extends JPanel {

 private static final long serialVersionUID = 7753397844373862032L;

 private boolean DEBUG = true;

 Object[][] gData = null;

 JTable table = null;

 MyTableModel model = null;

 String[] columnNames = { "First Name", "Last Name", "Sport", "Index" };

 public RefreshableTableDemo() {

 super(new GridLayout(1, 0));

 gData = getActualData();

 model = new MyTableModel(columnNames, gData);

 table = new JTable(model);

 table.setPreferredScrollableViewportSize(new Dimension(500, 70));

 table.setFillsViewportHeight(true);

 if (DEBUG) {

 table.addMouseListener(new MouseAdapter() {

 public void mouseClicked(MouseEvent e) {

 printDebugData(table);

 }

 });

 }

 // Create the scroll pane and add the table to it.

 JScrollPane scrollPane = new JScrollPane(table);

 JButton btn = new JButton("Refresh");

 btn.setBounds(100, 60, 100, 30);

 btn.addActionListener(new ActionListener() {

 @Override

 public void actionPerformed(ActionEvent event) {

 changeTableContent();

 }

 });

 // Add the scroll pane to this panel.

 add(scrollPane);

 add(btn);

 }

 private void printDebugData(JTable table) {

 int numRows = table.getRowCount();

 int numCols = table.getColumnCount();

 javax.swing.table.TableModel model = table.getModel();

 System.out.println("Value of data: ");

 for (int i = 0; i < numRows; i++) {

 System.out.print(" row " + i + ":");

 for (int j = 0; j < numCols; j++) {

 System.out.print(" " + model.getValueAt(i, j));

 }

 System.out.println();

 }

 System.out.println("--------------------------");

 }

 int index = 0;

 private Object[][] getActualData() {

 Object[][] result = { { "Kathy", "Smith", "Snowboarding", index },

 { "John", "Doe", "Rowing", index },

 { "Sue", "Black", "Knitting", index },

 { "Jane", "White", "Speed reading", index },

 { "Joe", "Brown", "Pool", index } };

 index++;

 return result;

 }

 private void changeTableContent() {

 model.setData(columnNames, getActualData());

 table.repaint();

 msgbox("Table refreshed successfully");

 }

 class MyTableModel extends AbstractTableModel {

 private String[] columnNames = null;

 private Object[][] data = null;

 public MyTableModel(String[] columnNames, Object[][] data) {

 super();

 setData(columnNames, data);

 }

 public void setData(String[] columnNames, Object[][] data) {

 this.columnNames = columnNames;

 this.data = data;

 }

 public int getColumnCount() {

 return columnNames.length;

 }

 public int getRowCount() {

 return data.length;

 }

 public String getColumnName(int col) {

 return columnNames[col];

 }

 public Object getValueAt(int row, int col) {

 return data[row][col];

 }

 /\*

 \* JTable uses this method to determine the default renderer/ editor for

 \* each cell. If we didn't implement this method, then the last column

 \* would contain text ("true"/"false"), rather than a check box.

 \*/

 public Class getColumnClass(int c) {

 return getValueAt(0, c).getClass();

 }

 /\*

 \* Don't need to implement this method unless your table's editable.

 \*/

 public boolean isCellEditable(int row, int col) {

 // Note that the data/cell address is constant,

 // no matter where the cell appears onscreen.

 if (col < 2) {

 return false;

 } else {

 return true;

 }

 }

 public void setValueAt(Object value, int row, int col) {

 if (DEBUG) {

 System.out.println("Setting value at " + row + "," + col

 + " to " + value + " (an instance of "

 + value.getClass() + ")");

 }

 data[row][col] = value;

 if (DEBUG) {

 System.out.println("New value of data:");

 printDebugData();

 }

 }

 private void printDebugData() {

 int numRows = getRowCount();

 int numCols = getColumnCount();

 for (int i = 0; i < numRows; i++) {

 System.out.print(" row " + i + ":");

 for (int j = 0; j < numCols; j++) {

 System.out.print(" " + data[i][j]);

 }

 System.out.println();

 }

 System.out.println("--------------------------");

 }

 }

 /\*\*

 \* Create the GUI and show it. For thread safety, this method should be

 \* invoked from the event-dispatching thread.

 \*/

 private static void createAndShowGUI() {

 // Create and set up the window.

 JFrame frame = new JFrame("SimpleTableDemo");

 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

 // Create and set up the content pane.

 RefreshableTableDemo newContentPane = new RefreshableTableDemo();

 newContentPane.setOpaque(true); // content panes must be opaque

 frame.setContentPane(newContentPane);

 // Display the window.

 frame.pack();

 frame.setVisible(true);

 }

 public static void main(String[] args) {

 // Schedule a job for the event-dispatching thread:

 // creating and showing this application's GUI.

 javax.swing.SwingUtilities.invokeLater(new Runnable() {

 public void run() {

 createAndShowGUI();

 }

 });

 }

 private void msgbox(String s) {

 JOptionPane.showMessageDialog(null, s);

 }

}



## SimpleTableDemo.java

package com.accenture.practice.swing.table;

import java.awt.Dimension;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.table.AbstractTableModel;

public class SimpleTableDemo extends JPanel {

 private static final long serialVersionUID = 7753397844373862032L;

 private boolean DEBUG = true;

 Object[][] gData = null;

 JTable table = null;

 MyTableModel model = null;

 String[] columnNames = { "First Name", "Last Name", "Sport", "Index" };

 public SimpleTableDemo() {

 super(new GridLayout(1, 0));

 gData = getActualData();

 model = new MyTableModel(columnNames, gData);

 table = new JTable(model);

 table.setPreferredScrollableViewportSize(new Dimension(500, 70));

 table.setFillsViewportHeight(true);

 if (DEBUG) {

 table.addMouseListener(new MouseAdapter() {

 public void mouseClicked(MouseEvent e) {

 printDebugData(table);

 }

 });

 }

 // Create the scroll pane and add the table to it.

 JScrollPane scrollPane = new JScrollPane(table);

 JButton btn = new JButton("Refresh");

 btn.setBounds(100, 60, 100, 30);

 btn.addActionListener(new ActionListener() {

 @Override

 public void actionPerformed(ActionEvent event) {

 changeTableContent();

 }

 });

 // Add the scroll pane to this panel.

 add(scrollPane);

 add(btn);

 }

 private void printDebugData(JTable table) {

 int numRows = table.getRowCount();

 int numCols = table.getColumnCount();

 javax.swing.table.TableModel model = table.getModel();

 System.out.println("Value of data: ");

 for (int i = 0; i < numRows; i++) {

 System.out.print(" row " + i + ":");

 for (int j = 0; j < numCols; j++) {

 System.out.print(" " + model.getValueAt(i, j));

 }

 System.out.println();

 }

 System.out.println("--------------------------");

 }

 int index = 0;

 private Object[][] getActualData() {

 Object[][] result = { { "Kathy", "Smith", "Snowboarding", index },

 { "John", "Doe", "Rowing", index },

 { "Sue", "Black", "Knitting", index },

 { "Jane", "White", "Speed reading", index },

 { "Joe", "Brown", "Pool", index } };

 index++;

 return result;

 }

 private void changeTableContent() {

 model.setData(columnNames, getActualData());

 table.repaint();

 msgbox("Table refreshed successfully");

 }

 class MyTableModel extends AbstractTableModel {

 private String[] columnNames = null;

 private Object[][] data = null;

 public MyTableModel(String[] columnNames, Object[][] data) {

 super();

 setData(columnNames, data);

 }

 public void setData(String[] columnNames, Object[][] data) {

 this.columnNames = columnNames;

 this.data = data;

 }

 public int getColumnCount() {

 return columnNames.length;

 }

 public int getRowCount() {

 return data.length;

 }

 public String getColumnName(int col) {

 return columnNames[col];

 }

 public Object getValueAt(int row, int col) {

 return data[row][col];

 }

 /\*

 \* JTable uses this method to determine the default renderer/ editor for

 \* each cell. If we didn't implement this method, then the last column

 \* would contain text ("true"/"false"), rather than a check box.

 \*/

 public Class getColumnClass(int c) {

 return getValueAt(0, c).getClass();

 }

 /\*

 \* Don't need to implement this method unless your table's editable.

 \*/

 public boolean isCellEditable(int row, int col) {

 // Note that the data/cell address is constant,

 // no matter where the cell appears onscreen.

 if (col < 2) {

 return false;

 } else {

 return true;

 }

 }

 public void setValueAt(Object value, int row, int col) {

 if (DEBUG) {

 System.out.println("Setting value at " + row + "," + col

 + " to " + value + " (an instance of "

 + value.getClass() + ")");

 }

 data[row][col] = value;

 if (DEBUG) {

 System.out.println("New value of data:");

 printDebugData();

 }

 }

 private void printDebugData() {

 int numRows = getRowCount();

 int numCols = getColumnCount();

 for (int i = 0; i < numRows; i++) {

 System.out.print(" row " + i + ":");

 for (int j = 0; j < numCols; j++) {

 System.out.print(" " + data[i][j]);

 }

 System.out.println();

 }

 System.out.println("--------------------------");

 }

 }

 /\*\*

 \* Create the GUI and show it. For thread safety, this method should be

 \* invoked from the event-dispatching thread.

 \*/

 private static void createAndShowGUI() {

 // Create and set up the window.

 JFrame frame = new JFrame("SimpleTableDemo");

 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

 // Create and set up the content pane.

 SimpleTableDemo newContentPane = new SimpleTableDemo();

 newContentPane.setOpaque(true); // content panes must be opaque

 frame.setContentPane(newContentPane);

 // Display the window.

 frame.pack();

 frame.setVisible(true);

 }

 public static void main(String[] args) {

 // Schedule a job for the event-dispatching thread:

 // creating and showing this application's GUI.

 javax.swing.SwingUtilities.invokeLater(new Runnable() {

 public void run() {

 createAndShowGUI();

 }

 });

 }

 private void msgbox(String s) {

 JOptionPane.showMessageDialog(null, s);

 }

}



## SimpleTableDemo.java

package com.accenture.practice.swing.table;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.table.AbstractTableModel;

import java.awt.Dimension;

import java.awt.GridLayout;

public class TableSortDemo extends JPanel {

 private boolean DEBUG = true;

 public TableSortDemo() {

 super(new GridLayout(1, 0));

 JTable table = new JTable(new MyTableModel());

 table.setPreferredScrollableViewportSize(new Dimension(500, 70));

 table.setFillsViewportHeight(true);

 table.setAutoCreateRowSorter(true);

 // Create the scroll pane and add the table to it.

 JScrollPane scrollPane = new JScrollPane(table);

 // Add the scroll pane to this panel.

 add(scrollPane);

 }

 class MyTableModel extends AbstractTableModel {

 private String[] columnNames = { "First Name", "Last Name", "Sport",

 "# of Years", "Vegetarian" };

 private Object[][] data = {

 { "Kathy", "Smith", "Snowboarding", new Integer(5),

 new Boolean(false) },

 { "John", "Doe", "Rowing", new Integer(3), new Boolean(true) },

 { "Sue", "Black", "Knitting", new Integer(2),

 new Boolean(false) },

 { "Jane", "White", "Speed reading", new Integer(20),

 new Boolean(true) },

 { "Joe", "Brown", "Pool", new Integer(10), new Boolean(false) } };

 public int getColumnCount() {

 return columnNames.length;

 }

 public int getRowCount() {

 return data.length;

 }

 public String getColumnName(int col) {

 return columnNames[col];

 }

 public Object getValueAt(int row, int col) {

 return data[row][col];

 }

 /\*

 \* JTable uses this method to determine the default renderer/ editor for

 \* each cell. If we didn't implement this method, then the last column

 \* would contain text ("true"/"false"), rather than a check box.

 \*/

 public Class getColumnClass(int c) {

 return getValueAt(0, c).getClass();

 }

 /\*

 \* Don't need to implement this method unless your table's editable.

 \*/

 public boolean isCellEditable(int row, int col) {

 // Note that the data/cell address is constant,

 // no matter where the cell appears onscreen.

 if (col < 2) {

 return false;

 } else {

 return true;

 }

 }

 /\*

 \* Don't need to implement this method unless your table's data can

 \* change.

 \*/

 public void setValueAt(Object value, int row, int col) {

 if (DEBUG) {

 System.out.println("Setting value at " + row + "," + col

 + " to " + value + " (an instance of "

 + value.getClass() + ")");

 }

 data[row][col] = value;

 // Normally, one should call fireTableCellUpdated() when

 // a value is changed. However, doing so in this demo

 // causes a problem with TableSorter. The tableChanged()

 // call on TableSorter that results from calling

 // fireTableCellUpdated() causes the indices to be regenerated

 // when they shouldn't be. Ideally, TableSorter should be

 // given a more intelligent tableChanged() implementation,

 // and then the following line can be uncommented.

 // fireTableCellUpdated(row, col);

 if (DEBUG) {

 System.out.println("New value of data:");

 printDebugData();

 }

 }

 private void printDebugData() {

 int numRows = getRowCount();

 int numCols = getColumnCount();

 for (int i = 0; i < numRows; i++) {

 System.out.print(" row " + i + ":");

 for (int j = 0; j < numCols; j++) {

 System.out.print(" " + data[i][j]);

 }

 System.out.println();

 }

 System.out.println("--------------------------");

 }

 }

 /\*\*

 \* Create the GUI and show it. For thread safety, this method should be

 \* invoked from the event-dispatching thread.

 \*/

 private static void createAndShowGUI() {

 // Create and set up the window.

 JFrame frame = new JFrame("TableSortDemo");

 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

 // Create and set up the content pane.

 TableSortDemo newContentPane = new TableSortDemo();

 newContentPane.setOpaque(true); // content panes must be opaque

 frame.setContentPane(newContentPane);

 // Display the window.

 frame.pack();

 frame.setVisible(true);

 }

 public static void main(String[] args) {

 // Schedule a job for the event-dispatching thread:

 // creating and showing this application's GUI.

 javax.swing.SwingUtilities.invokeLater(new Runnable() {

 public void run() {

 createAndShowGUI();

 }

 });

 }

}

 