МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ КОМП’ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра

комп’ютерних наук

Комплексне практичне індивідуальне завдання

з дисципліни

«Конструювання програмного забезпечення»

**Виконав:**

студент групи ПЗАС-32

Макаренко В.В.

**Перевірив:**

викладач

Сирник О. Й.

Тернопіль

2015

**План**

Вступ

1. Основи Git
2. Створення Git-репозиторію
3. Робота зі змінами
4. Лістинги програми

**Вступ**

Контроль версій - це система, котра веде протокол змін в файлі або кількох файлах увесь час для того, щоб ви мали можливість викликати певні версії пізніше. Незважаючи на те, що в якості прикладів файлів під контролем версій в цій книзі використовується вихідний код програмного забезпечення, насправді, будь-який тип файлів на ПК може бути розташований під контролем версій.

Якщо ви графічний або веб-дизайнер і ви хочете зберегти кожну версію малюнку або шару (, що безсумнівно вам хочеться,), то використання системи контролю версій (надалі - СКВ) буде дуже розсудливим в цьому випадку. СКВ надає можливість робити вам наступне: повертати файли до попереднього вигляду, повертати цілий проект до попереднього вигляду, переглядати зміни, зроблені за увесь час, дивитися, хто востаннє змінював щось, що могло призвести до проблеми, хто вирішив задачу та коли й багато іншого. Використання СКВ також означає що, якщо ви щось зламаєте або загубите файли, ви зможете легко усе відновити. До того ж, ви все це отримуєте за дуже маленькі накладні витрати.

**Основи Git**

Так що ж таке Git? Git це система контролю версій. Головна відмінність Git від будь-яких інших СКВ (наприклад, Subversion і їй подібних) - це те, як Git дивиться на свої дані. У принципі, більшість інших систем зберігає інформацію як список змін (патчів) для файлів. Ці системи (CVS, Subversion, Perforce, Bazaar та інші) відносяться до збережених даних як до набору файлів і змін, зроблених для кожного з цих файлів в часі, як показано на рис. 1.1.



Рис. 1.1. Інші системи зберігають дані як зміни до базової версії для кожного файлу.

Git не зберігає свої дані в такому вигляді. Замість цього Git вважає збережені дані набором зліпків невеликої файлової системи. Кожен раз, коли ви фіксуєте поточну версію проекту, Git, по суті, зберігає зліпок того, як виглядають всі файли проекту на поточний момент. Заради ефективності, якщо файл не змінювався, Git не зберігається файл знову, а робить посилання на раніше збережений файл. Те, як Git підходить до зберігання даних, схоже на рис. 1.2.



Рис. 1.2 Git зберігає дані як зліпки станів проекту в часі.

Це важлива відмінність Git від практично всіх інших систем контролю версій. Через нього Git змушений переглянути практично всі аспекти контролю версій, які інші системи перейняли від своїх попередниць. Git більше схожий на невелику файлову систему з неймовірно потужними інструментами, що працюють поверх неї, ніж на просто ВКВ.

**Створення Git-репозиторію**

Для створення Git-репозиторію існують два основні підходи. Перший підхід - імпорт в Git вже існуючого проекту або каталогу. Другий - клонування вже існуючого репозиторію з сервера.

Для того, щоб почати використовувати Git для існуючого проекту, то вам необхідно перейти в проектний каталог і в командному рядку ввести

$ git init

Ця команда створює в поточному каталозі новий підкаталог з ім'ям .git містить всі необхідні файли репозиторію - основу Git-репозиторію.

Щоб додати під версійній контроль існуючі файли (на відміну від порожнього каталогу), вам варто проіндексувати ці файли і здійснити перші фіксацію змін. Здійснити це ви можете за допомогою декількох команд git add вказують індексовані файли, а потім commit:

$ git add \*.c

$ git add README

$ git commit -m 'initial project version'

Якщо ви бажаєте отримати копію існуючого репозиторію Git, наприклад, проекту, в якому ви хочете взяти участь, то вам потрібна команда git clone. Якщо ви знайомі з іншими системами контролю версій, такими як Subversion, то помітите, що команда називається clone, а не checkout. Це важлива відмінність - Git отримує копію практично всіх даних, що є на сервері. Кожна версія кожного файлу з історії проекту забирається (pulled) з сервера, коли ви виконуєте git clone. Фактично, якщо серверний диск вийде з ладу, ви можете використовувати будь-який з клонів на будь-якому з клієнтів, для того щоб повернути сервер в той стан, в якому він перебував у момент клонування (ви можете втратити частину серверних перехоплювачів (server-side hooks) і т.п., але всі дані, поміщені під версійність контроль, будуть збережені.

Клонування репозиторію здійснюється командою git clone. Наприклад, якщо ви хочете клонувати бібліотеку Ruby Git, відому як Grit, ви можете зробити це таким чином:

$ git clone git://github.com/schacon/grit.git

Ця команда створює каталог з ім'ям grit, ініціалізує в ньому каталог .git, викачує всі дані для цього репозиторію і створює (checks out) робочу копію останньої версії. Якщо ви зайдете в новий каталог grit, ви побачите в ньому проектні файли, придатні для роботи і використання.

**Робота зі змінами**

Отже, у вас є справжній Git-репозиторій і робоча копія файлів для деякого проекту. Вам потрібно робити деякі зміни і фіксувати "знімки" стану (snapshots) цих змін у вашому репозиторії кожен раз, коли проект досягає стану, який вам хотілося б зберегти.

Запам'ятайте, кожен файл у вашому робочому каталозі може перебувати в одному з двох станів: під версійність контролем (відстежувані) і ні (невідстежувані). Відстежувані файли - це ті файли, які були в останньому зліпку стану проекту (snapshot); вони можуть бути незміненими, зміненими або підготовленими до коммітов (staged). Невідстежувані файли - це все інше, будь-які файли у вашому робочому каталозі, які не входили до вашого останнього зліпку стану і не підготовлені до коммітов. Коли ви вперше клонуєте репозиторій, всі файли будуть відстежувати і незміненими, тому що ви тільки взяли їх зі сховища (checked them out) і нічого поки не редагували.

Як тільки ви відредагуєте файли, Git буде розглядати їх як змінені, тому ви змінили їх з моменту останнього коммітов. Ви індексуєте (stage) ці зміни і потім фіксуєте все індексовані зміни, а потім цикл повторюється.

Тепер ви можете робити зміни у проекті. Коли ви будете вважати, що проект потрібно зберегти, вам потрібно використати команду

$ git commit –m “message”

Замість “message” слід ввести повідомлення яке описує зміни, які ви зробили, або завдання, яке ви виконали. Після цього цей комміт можна надіслати до віддаленого репозиторію, тобто зробити push, за допомогою команди

$ git push

**Лістинги програми**

Нижче приведені лістинги програми для імітації музичного інструменту “гітара”

import java.awt.BorderLayout;

import java.awt.Graphics;

import java.awt.Image;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.Box;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JComponent;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.SwingUtilities;

public class Main {

final JFrame Frame = new JFrame("Guitar");

ImageIcon Img\_1 = new ImageIcon("src\\Guitars\\Fender.png");

ImageIcon Img\_2 = new ImageIcon("src\\Guitars\\Ibanez.png");

ImageIcon Img\_3 = new ImageIcon("src\\Guitars\\Jakson.png");

ImageIcon Img\_4 = new ImageIcon("src\\Guitars\\Yamaha.png");

ImageIcon Img\_5 = new ImageIcon("src\\Guitars\\1.png");

ImageIcon Img\_6 = new ImageIcon("src\\Guitars\\2.png");

ImageIcon Img\_7 = new ImageIcon("src\\Guitars\\3.png");

ImageIcon Img\_8 = new ImageIcon("src\\Guitars\\4.png");

final JButton One = new JButton(Img\_1);

final JButton Two = new JButton(Img\_2);

final JButton Thre = new JButton(Img\_3);

final JButton Four = new JButton(Img\_4);

final JButton Five = new JButton(Img\_5);

final JButton Six = new JButton(Img\_6);

final JButton Seven = new JButton(Img\_7);

final JButton Eigth = new JButton(Img\_8);

public Main() {

class JBackgroundPanell extends JPanel {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private BufferedImage img1;

public JBackgroundPanell() {

// load the background image

try {

img1 = ImageIO.read(new File("src//Fon.bmp"));

} catch(IOException e) {

e.printStackTrace();

}

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

// paint the background image and scale it to fill the entire space

g.drawImage(img1, 0, 0, getWidth(), getHeight(), this);

}

}

JBackgroundPanell Main\_Panel = new JBackgroundPanell();

Frame.setVisible(true);

Frame.setSize(840, 500);

Frame.setLayout(new BorderLayout());

Frame.setLocationRelativeTo(null);

Frame.setResizable(false);

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

Frame.add(Main\_Panel);

Main\_Panel.setLayout(null);

Main\_Panel.add(One);

Main\_Panel.add(Two);

Main\_Panel.add(Thre);

Main\_Panel.add(Four);

Main\_Panel.add(Five);

Main\_Panel.add(Six);

Main\_Panel.add(Seven);

Main\_Panel.add(Eigth);

One.setBounds(10, 80, 158, 65);

Two.setBounds(175, 80, 158, 65);

Thre.setBounds(340, 80, 158, 65);

Four.setBounds(506, 80, 158, 65);

Five.setBounds(10, 200, 156, 236);

Six.setBounds(175, 200, 156, 236);

Seven.setBounds(340, 200, 156, 236);

Eigth.setBounds(506, 200, 156, 236);

One.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

new Fender();

}

});

Two.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

new Ibanez();

}

});

Thre.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

new Jackson ();

}

});

Four.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

new Yamaha ();

}

});

Five.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

new Acoustic();

}

});

Six.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

new Bass();

}

});

Seven.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

new Electric();

}

});

Eigth.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

new Aco();

}

});

}

public static void main(String[] args)

{

// TODO Auto-generated method stub

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

// TODO Auto-generated method stub

new Main();

}

});

}

}

import java.awt.BorderLayout;

import java.awt.Graphics;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class Aco {

JFrame aco = new JFrame ();

Aco ()

{

class JBackgroundPanell extends JPanel {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private BufferedImage img1;

public JBackgroundPanell() {

// load the background image

try {

img1 = ImageIO.read(new File("src//Guitar//acous.png"));

} catch(IOException e) {

e.printStackTrace();

}

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

// paint the background image and scale it to fill the entire space

g.drawImage(img1, 0, 0, getWidth(), getHeight(), this);

}

}

JBackgroundPanell Main\_Panel = new JBackgroundPanell();

aco.setVisible(true);

aco.setSize(1135, 616);

aco.setResizable(false);

aco.setLocationRelativeTo(null);

aco.setLayout(new BorderLayout());

aco.add(Main\_Panel);

}

}

import java.awt.BorderLayout;

import java.awt.Graphics;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class Acoustic {

JFrame acoustic = new JFrame ();

Acoustic ()

{

class JBackgroundPanell extends JPanel {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private BufferedImage img1;

public JBackgroundPanell() {

// load the background image

try {

img1 = ImageIO.read(new File("src//Guitar//Frecnbxyf.png"));

} catch(IOException e) {

e.printStackTrace();

}

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

// paint the background image and scale it to fill the entire space

g.drawImage(img1, 0, 0, getWidth(), getHeight(), this);

}

}

JBackgroundPanell Main\_Panel = new JBackgroundPanell();

acoustic.setVisible(true);

acoustic.setSize(1135, 616);

acoustic.setResizable(false);

acoustic.setLocationRelativeTo(null);

acoustic.setLayout(new BorderLayout());

acoustic.add(Main\_Panel);

}

}

import java.awt.BorderLayout;

import java.awt.Graphics;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class Bass {

JFrame bass = new JFrame ();

Bass ()

{

class JBackgroundPanell extends JPanel {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private BufferedImage img1;

public JBackgroundPanell() {

// load the background image

try {

img1 = ImageIO.read(new File("src//Guitar//Bass.png"));

} catch(IOException e) {

e.printStackTrace();

}

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

// paint the background image and scale it to fill the entire space

g.drawImage(img1, 0, 0, getWidth(), getHeight(), this);

}

}

JBackgroundPanell Main\_Panel = new JBackgroundPanell();

bass.setVisible(true);

bass.setSize(1135, 616);

bass.setResizable(false);

bass.setLocationRelativeTo(null);

bass.setLayout(new BorderLayout());

bass.add(Main\_Panel);

}

}

import java.awt.BorderLayout;

import java.awt.Graphics;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class Electric {

JFrame electric = new JFrame ();

Electric ()

{

class JBackgroundPanell extends JPanel {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private BufferedImage img1;

public JBackgroundPanell() {

// load the background image

try {

img1 = ImageIO.read(new File("src//Guitar//Electro.png"));

} catch(IOException e) {

e.printStackTrace();

}

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

// paint the background image and scale it to fill the entire space

g.drawImage(img1, 0, 0, getWidth(), getHeight(), this);

}

}

JBackgroundPanell Main\_Panel = new JBackgroundPanell();

electric.setVisible(true);

electric.setSize(1135, 616);

electric.setResizable(false);

electric.setLocationRelativeTo(null);

electric.setLayout(new BorderLayout());

electric.add(Main\_Panel);

}

}import java.awt.BorderLayout;

import java.awt.Graphics;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class Fender {

JFrame fender = new JFrame ();

Fender ()

{

class JBackgroundPanell extends JPanel {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private BufferedImage img1;

public JBackgroundPanell() {

// load the background image

try {

img1 = ImageIO.read(new File("src//Mrk//Fender.bmp"));

} catch(IOException e) {

e.printStackTrace();

}

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

// paint the background image and scale it to fill the entire space

g.drawImage(img1, 0, 0, getWidth(), getHeight(), this);

}

}

JBackgroundPanell Main\_Panel = new JBackgroundPanell();

fender.setVisible(true);

fender.setSize(1135, 616);

fender.setResizable(false);

fender.setLocationRelativeTo(null);

fender.setLayout(new BorderLayout());

fender.add(Main\_Panel);

}

}

import java.awt.BorderLayout;

import java.awt.Graphics;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class Ibanez {

JFrame ibanez = new JFrame ();

Ibanez ()

{

class JBackgroundPanell extends JPanel {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private BufferedImage img1;

public JBackgroundPanell() {

// load the background image

try {

img1 = ImageIO.read(new File("src//Mrk//ibanez.bmp"));

} catch(IOException e) {

e.printStackTrace();

}

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

// paint the background image and scale it to fill the entire space

g.drawImage(img1, 0, 0, getWidth(), getHeight(), this);

}

}

JBackgroundPanell Main\_Panel = new JBackgroundPanell();

ibanez.setVisible(true);

ibanez.setSize(1135, 616);

ibanez.setResizable(false);

ibanez.setLocationRelativeTo(null);

ibanez.setLayout(new BorderLayout());

ibanez.add(Main\_Panel);

}

}

import java.awt.BorderLayout;

import java.awt.Graphics;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class Jackson {

JFrame jackson = new JFrame ();

Jackson ()

{

class JBackgroundPanell extends JPanel {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private BufferedImage img1;

public JBackgroundPanell() {

// load the background image

try {

img1 = ImageIO.read(new File("src//Mrk//jackson.bmp"));

} catch(IOException e) {

e.printStackTrace();

}

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

// paint the background image and scale it to fill the entire space

g.drawImage(img1, 0, 0, getWidth(), getHeight(), this);

}

}

JBackgroundPanell Main\_Panel = new JBackgroundPanell();

jackson.setVisible(true);

jackson.setSize(1135, 616);

jackson.setResizable(false);

jackson.setLocationRelativeTo(null);

jackson.setLayout(new BorderLayout());

jackson.add(Main\_Panel);

}

}

import java.awt.BorderLayout;

import java.awt.Graphics;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class Yamaha {

JFrame yamaha = new JFrame ();

Yamaha ()

{

class JBackgroundPanell extends JPanel {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private BufferedImage img1;

public JBackgroundPanell() {

// load the background image

try {

img1 = ImageIO.read(new File("src//Mrk//Yamaha.bmp"));

} catch(IOException e) {

e.printStackTrace();

}

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

// paint the background image and scale it to fill the entire space

g.drawImage(img1, 0, 0, getWidth(), getHeight(), this);

}

}

JBackgroundPanell Main\_Panel = new JBackgroundPanell();

yamaha.setVisible(true);

yamaha.setSize(1135, 616);

yamaha.setResizable(false);

yamaha.setLocationRelativeTo(null);

yamaha.setLayout(new BorderLayout());

yamaha.add(Main\_Panel);

}

}