МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

МЕТОДИЧНІ ВКАЗІВКИ
ЩОДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ
З НАВЧАЛЬНОЇ ДИСЦИПЛІНИ
«ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»
ДЛЯ СТУДЕНТІВ ДЕННОЇ ФОРМИ НАВЧАННЯ
ЗІ СПЕЦІАЛЬНОСТІ 123 – «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ».
ЧАСТИНА ІІ

КРЕМЕНЧУК 2019

Методичні вказівки щодо виконання лабораторних робіт з навчальної дисципліни «Інженерія програмного забезпечення» (Частина ІІ) для студентів денної форми навчання зі спеціальності 123 – «Комп’ютерні системи та мережі»

Укладач к. т. н., доц. О. Г. Славко

Рецензент к. т. н., доц. В. М. Сидоренко

Кафедра комп’ютерних та інформаційних систем

Затверджено методичною радою КрНУ імені Михайла Остроградського
Протокол № 4 від 9 грудня 2018 р.

Голова методичної ради _____ проф. В. В. Костін

ЗМІСТ

ВСТУП

Навчальна дисципліна є логічним продовженням навчального курсу «Програмування», «Комп'ютерні мережі», «Архітектура комп'ютерів». Предметом вивчення навчальної дисципліни є фундаментальні основи, практичні методи та засоби підтримки життєвого циклу продуктів програмного забезпечення та інформаційних систем, основи та методи менеджменту створення програмних продуктів інформаційних технологій, основи та методи управління якістю програмних продуктів.

У межах навчальної дисципліни розглядаються питання проблем та напрямків розвитку технології програмування; основні методи і засоби автоматизації проектування, виробництва, випробувань та оцінки якості програмного забезпечення; напрямки розвитку методів і програмних засобів колективної розробки програмного забезпечення; стандартні підходи до розробки програмних проектів, який полягає у використанні моделей життєвого циклу, в процеси яких вбудовані методи проектування, верифікації, тестування та оцінювання проміжних робочих продуктів, а також перевірки планів та часу виконання робіт на цих процесах для того, щоб регулювати строки та витрати, а також ймовірні ризики та недоліки.

Мета і завдання навчальної дисципліни: ознайомлення студентів з процесами, методами та засобами підтримки життєвого циклу продуктів програмного забезпечення та інформаційних систем, основам менеджменту створення програмних продуктів інформаційних технологій, основам управління якістю програмних продуктів.

Місце навчальної дисципліни у навчальному процесі: навчальна дисципліна «Інженерія програмного забезпеченння» тісно пов'язана з такими дисциплінами, як «Програмування», «Комп'ютерні мережі», «Архітектура комп'ютерів» та іншими спеціальними дисциплінами навчального плану.

У результаті вивчення навчальної дисципліни студент повинен:

знати:

– методи проектування та виробництва програмного продукту;

– методи організації роботи в колективах розробників програмного забезпечення;

– основні етапи створення програмного забезпечення, моделі життєвого циклу програмного забезпечення, принципи побудови, проектування і тестування програм, програмних систем і комплексів;

уміти:

– самостійно вибирати потрібну для розробки модель життєвого циклу;

– самостійно застосовувати методи організації роботи в колективах розробників програмного забезпечення;

– самостійно застосовувати отримані знання при виконанні проектів і випускних кваліфікаційних робіт, а також у ході наукових досліджень;

– самостійно проектувати, тестувати та оцінювати якість та аналізувати ефективність програмного забезпечення;

– розробляти програмне забезпечення при роботі в команді.

1 ПЕРЕЛІК ЛАБОРАТОРНИХ РОБІТ

Лабораторна робота № 1

Тема. Середовище розробки Eclipse та архітектури вводу-виводу Java

Мета: ознайомлення та набуття практичних навичок роботи в середовищі Eclipse та організацією вводу-виводу Java.

Короткі теоретичні відомості

Eclipse – це розширюване середовище розробки (Integrated Development Environment, далее IDE) з відкритим вихідним кодом.

Платформа Eclipse у поєднанні з JDT (Java Development Tools) має багато можливостей, які включаються в комерційні IDE: редактор з підсвічуванням синтаксису, інкрементальна компіляція коду, потокобезпечний відладчик, навігатор по класах, менеджери файлів і проектів, а також інтерфейси до стандартних систем контролю вихідних текстів, таким як CVS і ClearCase.

Інтерфейс середовища Eclipse

Eclipse має стандартне меню та панель інструментів, а також багатосторінковий редактор з закладками.

Для вивчення середовища Eclipse необхідно створити простий застосунок – калькулятора на базі Swing. В майстрі створення застосунків в Eclipse можна вказати місце для збереження вихідних і скомпільованих файлів, створити окремі каталоги для таких файлів, вказати місце для зберігання вихідних і скомпільованих файлів, а також задати будь-які підпроекти і бібліотеки, які можуть знадобитися для роботи і компонування проекту.

Після створення простого застосунку в Eclipse новий зовнішній вигляд носить назву «Перспективи Java».

Перспектива, в термінах Eclipse, це зовнішній вигляд вікна, що зберігається та включає будь-яку кількість редакторів (editors) і уявлень (views). У Eclipse входить декілька перспектив за замовчанням (Java, Debug, Resource і

т. д.), які можна налаштовувати. Можна створювати нові перспективи. Перспективи управляються за допомогою елементів меню Window або панелі інструментів, яка зазвичай розташовується уздовж лівої межі вікна Eclipse.

Створення великої кількості папок з використанням діалогу New Folder може здатися дещо незручним. На щастя, Eclipse дозволяє зробити цю роботу, використовуючи методи, прийняті у операційній системі: командну оболонку, Windows Explorer і т. д.

Зауваження щодо створення файлів: на ранніх етапах розробки проекту ви можете зіткнутися з необхідністю часто запускати код на виконання, створювати нові файли або папки, додавати новий код, компілювати і тестувати його знову. Важливо пам'ятати, що меню Eclipse відображається по-різному залежно від поточної перспективи. Таким чином, якщо ви переключаєтеся в перспективу ресурсів для створення нових файлів або папок через уявлення Навігатора, ви виявите, що, наприклад, меню Run помітно відрізняється від аналогічного меню в перспективі Java. Одним з варіантів розв'язання проблеми може стати застосування пункту меню Window-> Show View і відображення уявлення Навігатора в перспективі Java. Щоб закріпити цю зміну, використовуйте команду меню Window-> Save Perspective As ..., що дозволяє зберегти настройки перспективи під новим ім'ям або замінити настройки існуючої перспективи Java.

Додавання коду

Нехай код програми калькулятора, що буде створюватись в лабораторній роботі, буде розділений на три окремих файли: *CalcModel.java*, *CalcPanel.java* і *Calculator.java*.

CalcModel.java

```
package com.devious.calculator;
 import java.util.Observable;
 public class CalcModel extends Observable
 {
    public static final int OP_NONE = 0;
    public static final int OP_ADD  = 1;
    public static final int OP_SUB  = 2;
    public static final int OP_MUL  = 3;
```

```
public static final int OP_DIV  = 4;

private int pendingOp = 0;
private int previousValue = 0;
private int currentValue = 0;

public String getValue()
{
   return Integer.toString(currentValue);
}
public void addDigit(String s)
{
   char c = s.charAt(0);
   String val = getValue() + c;
   setDisplay(Integer.parseInt(val));
}
public void addDigit(char c)
{
  String val = getValue() + c;
  setDisplay(Integer.parseInt(val));
}
public void setOperation(int op)
{
   if (pendingOp != OP_NONE)
      calculate();
   previousValue = currentValue;
   this.pendingOp = op;
   currentValue = 0;
}
public void setDisplay(int value)
{
   currentValue = value;
   setChanged();
   notifyObservers();
}
public void clear()
{
   this.pendingOp = OP_NONE;
   previousValue = 0;
   setDisplay(0);
}
public void calculate()
{
   switch (pendingOp)
   {
   case OP_ADD:
      setDisplay(previousValue + currentValue);
      break;
   case OP_SUB:
      setDisplay(previousValue + currentValue);
      break;
   case OP_MUL:
```

```
            setDisplay(previousValue * currentValue);
            break;
        case OP_DIV:
            setDisplay(previousValue / currentValue);
            break;
        }
        pendingOp = OP_NONE;
        previousValue = 0;
    }
}
```

CalcPanel.java

```
package com.devious.calculator;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class CalcPanel
     extends JPanel
     implements ActionListener, Observer
{
    private CalcModel model;
    private JLabel display = new JLabel("0");
    public CalcPanel(CalcModel model)
    {
        super();
        this.model = model;
        model.addObserver(this);
    }
    protected JButton addButton(String label)
    {
        JButton b = new JButton(label);
        b.addActionListener(this);
        return b;
    }
    protected JPanel createButtonPanel()
    {
        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(0, 4));

        panel.add(addButton("1"));
        panel.add(addButton("2"));
        panel.add(addButton("3"));
        panel.add(addButton("+"));

        panel.add(addButton("4"));
        panel.add(addButton("5"));
        panel.add(addButton("6"));
        panel.add(addButton("-"));
```

```
    panel.add(addButton("7"));
    panel.add(addButton("8"));
    panel.add(addButton("9"));
    panel.add(addButton("*"));

    panel.add(addButton("C"));
    panel.add(addButton("0"));
    panel.add(addButton("="));
    panel.add(addButton("/"));

    return panel;
}
public void init()
{
    this.setLayout(new BorderLayout());
    display.setVerticalTextPosition(JLabel.CENTER);
    display.setHorizontalTextPosition(JLabel.RIGHT);
    display.setVerticalAlignment(JLabel.TOP);
    display.setHorizontalAlignment(JLabel.TRAILING);
    display.setBorder(BorderFactory.createLineBorder(Color.black));

    this.add(display, BorderLayout.NORTH);
    this.add(createButtonPanel(), BorderLayout.CENTER);
}
public void actionPerformed(ActionEvent evt)
{
    System.out.println(evt.paramString());
    try {
        switch (evt.getActionCommand().charAt(0)) {
        case '+':
            model.setOperation(CalcModel.OP_ADD);
            break;
        case '-':
            model.setOperation(CalcModel.OP_SUB);
            break;
        case '*':
            model.setOperation(CalcModel.OP_MUL);
            break;
        case '/':
            model.setOperation(CalcModel.OP_DIV);
            break;
        case '=':
            model.calculate();
            break;
        case 'C':
            model.clear();
            break;
        default:
            model.addDigit(evt.getActionCommand());
            break;
        }
    } catch (NumberFormatException ex) {}
```

```
    }
    public void update(Observable o, Object arg)
    {
       display.setText(model.getValue());
    }
   }
```

Calculator.java

```
package com.devious.calculator;
 import javax.swing.*;
 public class Calculator
{
   public static void main(String[] args)
   {
      try {
         UIManager.setLookAndFeel(
            UIManager.getCrossPlatformLookAndFeelClassName());
      } catch (Exception e) {}

      JFrame calcFrame = new JFrame("Calculator");

      CalcModel model = new CalcModel();
      CalcPanel calcPanel = new CalcPanel(model);
      calcPanel.init();

      calcFrame.setContentPane(calcPanel);
      calcFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      calcFrame.pack();
      calcFrame.setVisible(true);
   }
}
```

CalcPanel.java обробляє велику частину призначеного для користувача інтерфейсу калькулятора. Calculator.java відповідає за настройку графічного інтерфейсу та його відображення. Велика частина важливої роботи забезпечується файлом CalcModel.java, який реалізує безпосередньо «механіку» калькулятора: реакцію на події, числові обчислення, оновлення вікна і т. д. Найпростіший шлях дослідження коду полягає в перемиканні в перспективу Java або в перспективу Перегляду Java (Java Browsing). Використовуйте елемент меню Window -> Open Perspective або натисніть кнопку Open на «перспективній» панелі інструментів.

При використанні Package Explorer для перегляду вихідного коду можна помітити, що уявлення угруповання коду є надмірним. Щоб його закрити, слід

натиснути на системний значок у заголовку його вікна. При необхідності відобразити його знову, слід використовувати команду меню Window -> Show View -> Outline.

Після того, як створено конфігурацію запуску додатка, можна запускати його багаторазово з використанням елемента меню Run -> Run History або кнопки Run на панелі інструментів. Для пошуку помилки (***в наведеному коді не виконується операція вирахування)*** необхідно використовувати відладчик вихідного коду Eclipse. Можна налагодити додаток в тій конфігурації запуску, яка була створена раніше.

Засоби форматованого введення /виводу Java

Як засіб форматованого введення / виведення в Java можна використовувати клас StringTokenizer, який схожий за назвою і призначенням на клас StreamTokenizer, однак працює не з потоками, а з текстовими рядками. Можна використовувати клас StringTokenizer для виділення елементів рядків класу String.

У класі StringTokenizer передбачено три конструктора:

```
public StringTokenizer(String str);
public StringTokenizer(String str,
  String delim);
public StringTokenizer(String str,
  String delim, boolean returnTokens);
```

Як перший параметр конструкторам передається посилання на рядок, що розбирається. Другий параметр delim, якщо він є, задає роздільники, з використанням яких у рядку будуть виділятися елементи. Параметр returnTokens визначає, чи треба повертати виявлені роздільники як елементи рядка, що розглядається.

Методи класу StringTokenizer перераховані нижче:

```
public String nextToken();
public String nextToken(String delim);
public int countTokenss();
public boolean hasMoreElements();
public boolean hasMoreTokenss();
public Object nextElement();
```

Для розбору рядка додаток зазвичай створює цикл, викликаючи в ньому метод nextToken. Умовою завершення циклу може бути або виникнення виключення NoSuchElementException, або повернення значення false методами hasMoreElements або hasMoreTokens.

Метод countTokens дозволяє визначити, скільки разів був викликаний метод nextToken перед виникненням виключення NoSuchElementException.

```
String s = "Строка, которую мы хотим разобрать на слова";
StringTokenizer st = new StringTokenizer(s, " \t\n\r,.");
while(st.hasMoreTokens()){
// Получаем слово и что-нибудь делаем с ним, например,
// просто выводим на экран
System.out.println(st.nextToken()) ;
}
```

Отримані слова заносяться в який-небудь клас-колекцію: Vector, Stack або інший контейнер, який найбільше підходить для подальшої обробки тексту.

Порядок виконання роботи

1. Ознайомитися з матеріалом, викладеним у коротких теоретичних відомостях.

2. Створити новий проект: вибрати File->New->Project..., у лівому списку майстра, що з’явився, вибрати Java Project, натиснути кнопку Next. Як назву проекту необхідно ввести Calculator і знову натиснути Next. Клацніть на кнопці Finish. У вікні, що з’явилося та пропонує перемкнути перспективу, натисніть Yes. Eclipse створить новий проект.

3. Створити папки, в яких буде міститися вихідний код. Для цього перейти у перспективу ресурсів (Resource) за допомогою Window -> Open Perspective -> Other. У поданні Навігатора розкрити дерево папок і знайти вузол src. Вибрати цей вузол, а потім пункт меню File -> New -> Folder. У діалозі переконатися, що вибрана папка src, і ввести com у поле Folder Name. Створити папки devious і calculator у папці com і вибрати будь-яку папку в поданні Навігатора, а потім вибрати пункт меню File -> Refresh. Eclipse перегляне файлову систему та приведе проект у відповідність до внесених змін.

4. Додати код до проекту (коди файлів *CalcMode.java*, *CalcPanel.java* і *Calculator.java* наведені в теоретичних відомостях). Для цього необхідно помістити указані файли у підпапку com / devious / calculator, використовуючи команду File -> Refresh для оновлення проекту Eclipse.

5. Запуск коду та налагодження:

– з меню Run вибрати елемент Run ...;

– у діалозі зі списку Змін Запуску (Launch Configurations) вибрати «Java Application», потім натиснути на кнопку New;

– увести Calculator як ім’я (Name) нової конфігурації;

– натиснути кнопку Search ... і вибрати Calculator як головний клас (Main Class), ввести у текстовому полі com.delirious.calculator.Calculator;

– натиснути кнопку Run для збереження конфігурації та запуску додатка.

6. Запустити застосунок у відладчику (для цього вибрати у меню Run -> Debug History -> Calculator) і знайти помилку в коді (як зазначалось у теоретичних відомостях, ***не виконується операція вирахування)***.

Зміст звіту

1. Назва та мета роботи.
2. Методика проведення роботи з графічними результатами.
3. Письмові відповіді на контрольні питання.

Контрольні питання

1. Що таке архітектура програмного забезпечення?
2. Для чого повторне використання коду у формуванні архітектури?
3. Навести приклади типових рішень, шаблонів та їх типів при проектуванні програмного забезпечення.
4. Навести структуру опису шаблона.

Література: [1–3, 15].

Лабораторна робота № 2

Тема. Розробка клієнта COM-серверу

Мета: ознайомлення та набуття практичних навичок розробки клієнту COM-серверу.

Короткі теоретичні відомості

Сервер COM (Component Object Model, модель компонентного об’єкту) – додаток або бібліотека, які надають сервіс клієнтському додатку (або бібліотеці). Сервер включає принаймні один об’єкт COM, який у свою чергу, являє собою сукупність методів і властивостей. Клієнт не зобов’язаний знати, де в пам’яті розташовуються об’єкти COM.

Коли клієнт запитує сервіс у об’єкта COM, він (клієнт) повинен передати ідентифікатор класу CLSID (class identifier). Ідентифікатор класу CLSID створюється на основі GUID інтерфейсу об’єкта COM.

За ідентифікатором класу CLSID COM визначає відповідний сервер, завантажує його в пам’ять і сервер створює екземпляр об’єкта COM. Примірники об’єктів COM створює фабрика класів (class factory), до якої звертається сервер. Фабрика класів має свій інтерфейс IClassFactory.

Фабрика класів і клас CoClass. COM об’єкт є екземпляром класу CoClass, в якому реалізовані один або більше інтерфейсів COM. Об’єкт COM забезпечує ті сервіси, які визначені в інтерфейсах класу CoClass.

Екземпляри класу CoClass створюються спеціальним типом об’єкта, який називається фабрикою класу. Коли клієнт звертається до COM об’єкту, фабрика класу створює екземпляр об’єкта та реєструє екземпляр об’єкта для цього конкретного клієнта. Якщо в цей час інший клієнт звертається до об’єкта, фабрика класів для нього також створює (інший) екземпляр об’єкта.

Будь-клас CoClass повинен мати фабрику класів і ідентифікатор класу CLSID, отже екземпляр COM об’єкта цього класу може бути створений ззовні, тобто з іншого модуля. Завдяки наявності у класів CoClass унікального ідентифікатора CLSID, вони можуть бути оновлені в будь-який час, як тільки

для класу розроблений новий інтерфейс. Новий інтерфейс може використовувати модифіковані або нові методи і це не чинить ніякого впливу на колишні версії.

Способи реалізації COM-серверів

При роботі з об'єктами COM клієнт не знає, де знаходиться об'єкт. Він просто звертається до об'єкта за допомогою його інтерфейсу. Далі бібліотека COM виконує всі необхідні кроки для задоволення виклику клієнта.

Ці кроки залежать від конкретного місцезнаходження об'єкта COM: у тому ж процесі, що і клієнт, в іншому процесі на машині клієнта або на іншій машині в мережі. Залежно від цього розрізняють три види серверів COM.

Сервер у клієнтському процесі (In-process server). Це бібліотека DLL, яка виконується в адресному просторі процесу клієнта. Наприклад, елемент ActiveX, впроваджений у web-сторінку, виконується в Chrome або іншому браузері. Це означає, що об'єкт ActiveX завантажується на машину клієнта та виконується в тому ж процесі, що і web браузер. Клієнт звертається до об'єкту COM шляхом прямого виклику інтерфейсу COM.

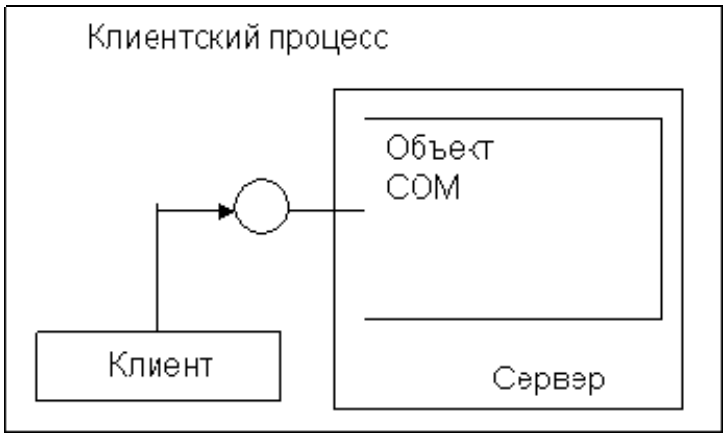

Рисунок 1 – Сервер в клієнтському процесі

Локальний сервер (local server) являє собою інший застосунок (файл *.exe), який виконується в іншому адресному просторі, але на тому ж комп'ютері, що й клієнтська програма. Наприклад, обробка таблиці Excel, впровадженої в документ Word, виконується додатком Excel. Локальний сервер зв'язується з клієнтом за допомогою COM

бібліотек.

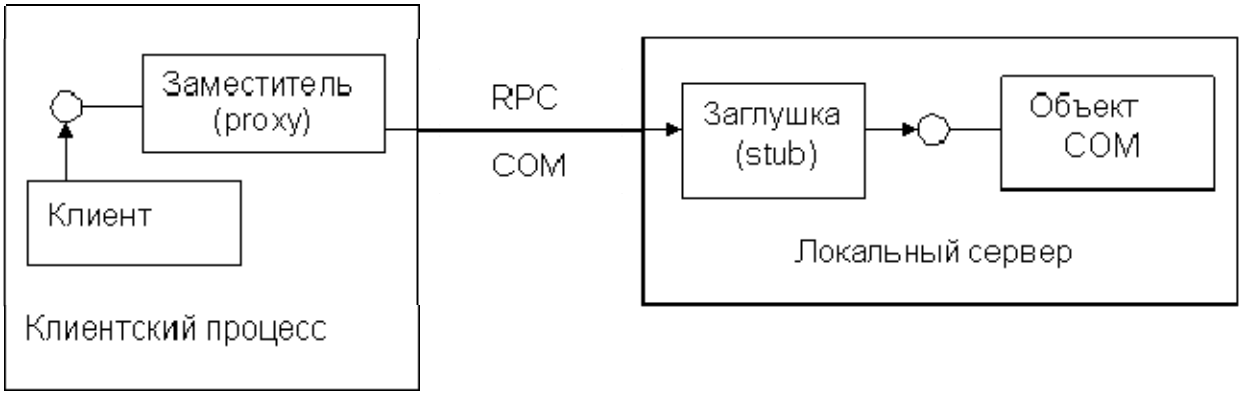

Рисунок 2 – Локальний сервер

Коли об’єкт COM належить іншому процесу на тому ж комп’ютері, що й клієнтська програму, або зовсім на іншому комп’ютері (в мережі), COM використовує так званого «замісник» (proxy) для того, щоб ініціювати віддалений виклик процедури (remote procedure call – RPC). Оскільки замісник знаходиться в клієнтському процесі, то з точки зору клієнта звернення до інтерфейсів виглядають так само, як і для випадку розміщення сервера в клієнтському процесі. Замісник перехоплює виклики клієнта та відправляє їх туди, де знаходиться сервер. Механізм, який дозволяє клієнту отримати доступ до об’єкта, що знаходиться в іншому процесі або на іншому комп’ютері (невидимо для клієнта), називається ***маршалінг*** або ***маршалізація***.

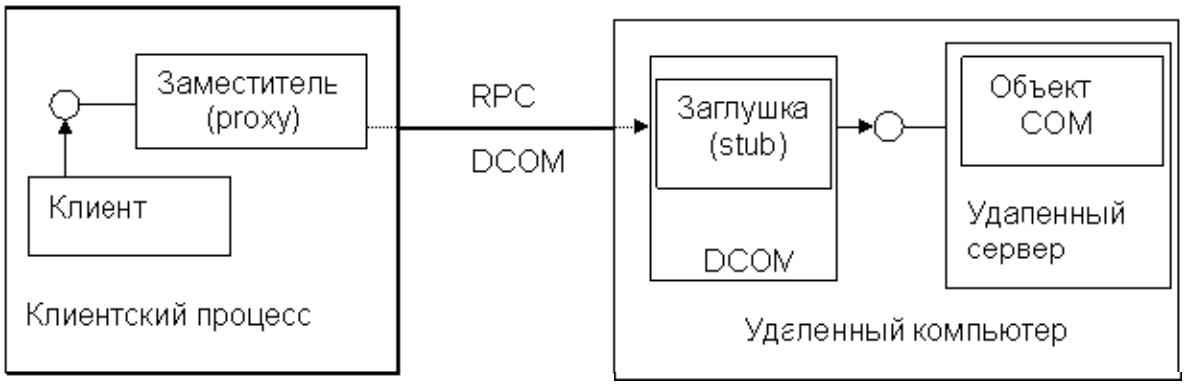

Рисунок 3 – Віддалений сервер

Різниця між локальним і віддаленим сервером полягає в способі (і інструментальних засобах) взаємодії клієнта і сервера, що застосовується: у першому випадку використовується COM, а в другому – DCOM.

Якщо сервер реалізований як бібліотека (а бібліотека не може виконуватися як самостійний застосунок), то COM створює спеціальний застосунок-сурогат (surrogate), в рамках якого виконується бібліотека.

Маршалинг. Механізм маршалінгу дозволяє клієнтському застосунку робити виклики методів інтерфейсу для об'єкта COM, який знаходиться в іншому процесі або на іншому комп'ютері.

При будь-якому виклику функцій за допомогою інтерфейсу клієнтська програма поміщає фактичні параметри функції в стек і виконує її виклик. Якщо функція не перебуває у клієнтському процесі, то виклик передається заміснику. Замісник упаковує параметри у пакет і передає їх віддаленому об'єкту. Заглушка COM об'єкта розпаковує пакет, поміщає аргументи у стек і викликає об'єкт COM. Таким чином об'єкт COM обробляє запит клієнта в своєму власному адресному просторі.

Який саме маршалінг буде реалізований, залежить від реалізації COM об'єкта. Стандартний механізм реалізований інтерфейсом IDispatch. Крім того, об'єкти можуть самостійно виконувати маршалінг, проте це досить складно.

Порядок виконання роботи

1. Реалізація та використання COM-класу в C++:

- Опис класу IUnknown:

```
struct IUnknown
{
virtual HRESULT STDMETHODCALLTYPE QueryInterface(
/* [in] */ REFIID riid,
/* [iid_is][out] */ void __RPC_FAR *__RPC_FAR *ppvObject)=0 ;
virtual ULONG STDMETHODCALLTYPE AddRef( void) =0;
virtual ULONG STDMETHODCALLTYPE Release( void) =0;
};
```

Смысл использованных макросов:
HRESULT – long;
STDMETHODCALLTYPE – _stdcall

REFIID – const GUID &

- Опис базового класу IAccount (нащадка IUnknown) як абстрактного класу:

```
class IAccount : public IUnknown
{
public:
// IAccount methods
STDMETHOD(GetBalance)(int* pBalance) = 0;
STDMETHOD(Deposit)(int amount) = 0;
};
```

Тут STDMETHOD – virtual HRESULT _stdcall.

- Опис конкретного класу CAccount:

```
class CAccount : public IAccount
{
public:
CAccount() // конструктор
{
m_nRef = 0;
m_nBalance = 100;
}
// методи IUnknown
STDMETHOD(QueryInterface)(REFIID, void**);
STDMETHOD_(ULONG, AddRef)();
STDMETHOD_(ULONG, Release)();
// IAccount methods
STDMETHOD(GetBalance)(int* pBalance);
STDMETHOD(Deposit)(int amount);
protected:
ULONG m_nRef; // число ссылок
int m_nBalance; // счет в банке
};
```

Використані макроси:

REFIID – GUID (структура из 16 байт);

STDMETHODIMP – HRESULT _stdcall.

- Реалізація методів класу CAccount:

```
STDMETHODIMP
CAccount::QueryInterface(REFIID iid, void** ppv)
{
if (iid == IID_IUnknown)
*ppv = (IAccount*) this;
else if (iid == IID_IAccount)
*ppv = (IAccount*) this;
```

```
else
{
*ppv = NULL;
return E_NOINTERFACE;
}
AddRef();
return NOERROR;   }
```

Метод QueryInterface в кінцевому підсумку повертає посилання на таблицю vtable, яка створюється компілятором C ++ автоматично, якщо в класі визначено віртуальні функції.

```
STDMETHODIMP_(ULONG) CAccount :: AddRef()
{   return ++m_nRef;   }
STDMETHODIMP_(ULONG) CAccount :: Release()
{   if(--m_nRef == 0)
{   delete this; // уничтожение экземпляра класса
   // Trace("Object destroyed", "CAccount");
    return 0;   }
return m_nRef;   }
STDMETHODIMP CAccount :: Deposit(int amount)
{  m_nBalance += amount;
   return S_OK;   }
STDMETHODIMP CAccount::GetBalance(int* pBalance)
{  *pBalance = m_nBalance;
    return S_OK;   }
```

- описати екземпляр класу IAccount для практичного використання класу CAccount: IAccount * m_pAccount=NULL;
- отримати посилання на інтерфейс. Це завдання вирішує (звичайна) функція CreateAccount:

```
BOOL CreateAccount(IAccount ** ppAccount)
{  HRESULT hr;
if (ppAccount == NULL)
return FALSE;
// Create object
CAccount* pAccount = new CAccount;
if (pAccount == NULL)
return FALSE;
// получить интерфейс. При этом в QueryInterface вызывается метод AddRef
hr = pAccount->QueryInterface(IID_IAccount, (void**) ppAccount);
if (SUCCEEDED(hr))
{    Trace("Object created", "CAccount");
     return TRUE;   }
else
{   delete pAccount;
    return FALSE;   }  }
```

- отримати ідентифікатор інтерфейсу за допомогою GuidGen і описати його як константу, наприклад:

```
static const IID IID_IDisplay =
{ 0x5723b700, 0x2878, 0x11d1, { 0xa0, 0x1b, 0x0, 0xa0, 0x24, 0xd0, 0x66, 0x32 } };
```

Ідентифікатор для інтерфейсу IUnknown визначений наперед.

Не дивлячись на те, що у функції CreateAccount покажчик pAccount на екземпляр класу створюється локально, об’єкт може бути зруйнований, так як посилання на нього зберігається в параметрі ppAccount. Примірник COM класу знищується в методі Release.

Виклик цієї функції доцільно виконати наступним чином:

```
if (m_pAccount)
{    m_pAccount->Release();
     m_pAccount = NULL;   }
// создание экземпляра COM класса
if (!CreateAccount(&m_pAccount))
{   MessageBox("CreateAccount failed");
    return;  }
```

- після отримання посилання на інтерфейс нею можна користуватися для виклику методів класу, наприклад, так:

```
int balance;
HRESULT hr = m_pAccount->GetBalance(&balance);
if (FAILED(hr))
{
MessageBox("GetBalance failed");
return;   }
```

2. Виконати наступні кроки для додавання інших додаткових інтерфейсів:

- оголосити новий (абстрактний) клас як похідний від IUnknown;
- згенерувати для нього ідентифікатор IID за допомогою GuidGen.exe;
- оголосити конкретний клас як похідний від всіх підтримуваних об’єктом COM інтерфейсів.

Наприклад:

```
class IDisplay : public IUnknown
{
public:
// IDisplay methods
STDMETHOD(Show)() = 0;  };
```

- використати множинне успадкування й описати конкретний клас:

```
class NewCAccount : public IAccount, IDisplay
{
…
}
```

- оскільки з’явився додатковий інтерфейс, змінити реалізацію QueryInterface так, щоб він дозволяв отримувати будь-який з інтерфейсів IAccount і IDisplay. Приклад реалізації цього методу:

```
STDMETHODIMP
CAccount::QueryInterface(REFIID iid, void** ppv)
{
if (iid == IID_IUnknown)
*ppv = (IAccount*) this;
else if (iid == IID_IAccount)
*ppv = (IAccount*) this;
else if (iid == IID_IDisplay)
*ppv = (IDisplay*) this;
else
{
*ppv = NULL;
return E_NOINTERFACE;  }
AddRef();
return NOERROR;  }
```

Зміст звіту

1. Назва та мета роботи.

2. Методика проведення роботи з графічними результатами.

3. Письмові відповіді на контрольні питання.

Контрольні питання

1. Що таке шаблони аналізу при проектуванні програмного забезпечення?

2. Для чого потрібні діаграми класів при проектуванні програмного забезпечення?

3. Дати опис архітектурному стилю «Конвеєрна обробка даних».

4. Дати опис архітектурному стилю «Пакетна обробка даних».

5. Дати опис архітектурному стилю «Канали і фільтри».

Література: [4–6].

Лабораторна робота № 3

Тема. Використання технології XML у Java-застосунках

Мета: ознайомлення та набуття практичних навичок роботи з XML (eXtensible Markup Language) і засобами обробки XML-даних за допомогою парсерів SAX і DOM.

Короткі теоретичні відомості

Розширювана мова розмітки (Extensible Markup Language, XML) – це запропонований консорціумом World Wide Web Consortium (W3C) стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет. Є спрощеною підмножиною мови розмітки SGML. XML-документ складається з текстових знаків і придатний до читання людиною.

Стандарт XML визначає набір базових лексичних та синтаксичних правил для побудови мови описання інформації шляхом застосування простих тегів. Цей формат достатньо гнучкий для того, аби бути придатним для застосування в різних галузях. Іншими словами, запропонований стандарт визначає метамову, на основі якої шляхом запровадження обмежень на структуру та зміст документів визначаються специфічні, предметно-орієнтовані мови розмітки даних. Ці обмеження описуються мовами схем, такими як XML Schema (XSD), DTD або RELAX NG. Прикладами мов, заснованих на XML, є: XSLT, XAML, XUL, RSS, MathML, GraphML, XHTML, SVG, XML Schema.

Simple API for XML (простий програмний інтерфейс для роботи з XML, скорочено SAX) – специфікація прикладного програмного інтерфейсу для послідовного отримання даних із структурованих XML документів. Цей інтерфейс є поширеною альтернативою Document Object Model (DOM).

Аналізатор, який реалізує інтерфейс SAX (SAX Parser) обробляє

інформацію із XML документа як єдиний потік даних. Цей потік даних доступний лише в одному напрямі, тобто, раніш оброблені дані неможливо повторно прочитати без повторного аналізу.

Більшість програмістів XML технологій вважають, що обробка XML документів відповідно парадигмі SAX, в цілому, швидша, аніж при використанні DOM. Це пояснюється тим, що потік SAX потребує набагато меншого обсягу пам'яті у порівнянні із побудовою повного дерева DOM.

SAX аналізатори реалізують з використанням підходу передачі повідомлень, коли програмісту необхідно описати обробники подій, які викликаються аналізаторами під час обробки XML документа.

SAX було розроблено зусиллями спільноти списку розсилки xml-dev, без формальних комітетів, але він був швидко визнаний компаніями, які спеціалізуються на засобах обробки XML документів.

Об'єктна модель документа (Document Object Model, DOM) – специфікація прикладного програмного інтерфейсу для роботи зі структурованими документами (як правило, документів XML). Визначається ця специфікація консорціумом W3C.

З точки зору об'єктно-орієнтованого програмування, DOM визначає класи, методи та атрибути цих методів для аналізу структури документів і роботи з представленням документів у вигляді дерева. Все це призначено для того, аби надати можливість комп'ютерній програмі доступу та динамічної модифікації структури, змісту й оформлення документа.

Разом із поширенням і розвитком веб-технологій і веб-браузерів почали з'являтись різні, часто несумісні інтерфейси роботи з HTML документами в інтерпретаторах JavaScript, вбудованих у веб-браузери. Це спонукало World Wide Web Consortium (W3C) узгодити і визначити низку стандартів, які отримали назву W3C Document Object Model (W3C DOM). Специфікації W3C не залежать від платформи або мови програмування.

Через те, що структура документа представляється у вигляді дерева, повний зміст документа аналізується та зберігається в пам'яті комп'ютера.

Тому, DOM підходить для застосувань у програмах, які вимагають багаторазовий доступ до елементів документа в довільному порядку. В разі, якщо треба лише послідовний або одноразовий доступ до елементів документа, рекомендується, для пришвидшення переробки і зменшення обсягів необхідної пам'яті комп'ютера, використовувати послідовну модель роботи зі структурованими документами (SAX).

Порядок виконання роботи

Розробити програму згідно свого варіанту завдання з табл. 1 (номер за журналом mod кількість варіантів + 1), яка дозволяє маніпулювати даними за рахунок завантаження та вивантаження з XML-файлу.

Парсер **SAX** реалізують студенти з **парним** номером варіанту.

Парсер **DOM** реалізують студенти з **непарним** номером варіанту.

Таблиця 1 – Варіанти завдань

№ варіанту	Завдання
1	Телефонна книга
2	Адресна книга
3	Курс валют за тиждень
4	Прогноз погоди на тиждень
5	Бібліотека
6	Розклад руху транспортних засобів
7	Розклад занять ВНЗ

Зміст звіту

1. Назва та мета роботи.
2. Методика проведення роботи з графічними результатами.
3. Письмові відповіді на контрольні питання.

Контрольні питання

1. В чому полягає різниця між SAX і DOM?

2. Що таке XML? Як він використовується в web-сервісах?

3. Яке розширення мають файли web-сервісів?

4. Який формат даних, які повертаються, використовуються у web-сервісах (службах)?

5. Дати опис технології XML-RPC, SOAP і WSDL.

Література: [1–3, 7, 8].

Лабораторна робота № 4

Тема. Створення web-сервісів

Мета: ознайомлення та набуття практичних навичок зі створення web-сервісів і розподілених застосунків.

Короткі теоретичні відомості

Веб-служба, веб-сервіс (web service) – це програмна система, що ідентифікується URI, і публічні інтерфейси і прив'язки якої визначені та описані мовою XML. Опис цієї програмної системи може бути знайдено іншими програмними системами, які можуть взаємодіяти з нею відповідно до цього опису з використанням повідомлень, що базуються на XML та передаються за допомогою Інтернет-протоколів.

Термін «веб-служба» організацією W3C застосовується до багатьох різних систем, але в основному термін стосується клієнтів та серверів, що взаємодіють за допомогою повідомлень протоколу SOAP. В обох випадках припускається, що існує також опис доступних операцій у форматі WSDL. Хоча наявність цього опису не є вимогою SOAP, а радше передумовою для автоматичного генерування коду на платформах Java і .NET на стороні клієнта.

Стандарти, що використовуються веб-службами

XML – розширювана мова розмітки, призначена для зберігання та передачі структурованих даних;

SOAP – протокол обміну повідомленнями на базі XML;

WSDL – мова опису зовнішніх інтерфейсів веб-служби на базі XML;

UDDI – універсальний інтерфейс розпізнавання опису та інтеграції (Universal Discovery, Description, and Integration). Каталог веб-служб і даних про компанії, що надаються веб-служби для загального користування або конкретним компаніям.

Переваги web-служб:

- забезпечують взаємодію програмних систем незалежно від платформи;
- базуються на відкритих стандартах та протоколах. Завдяки використанню XML досягається простота розробки і відлагодження веб-служб;
- використання HTTP-протоколу забезпечує взаємодію програмних систем через міжмережевій екран.

Порядок виконання роботи

1. Створити і протестувати пошуковий web-сервіс з конкретної предметної області. Для цього необхідно:

- реалізувати передачу користувальницької частини пошукового запиту в якості параметра у web-метод (і) web-сервісу;
- реалізувати валідацію (перевірка на порожнечу та наявність прогалин) вхідного пошукового запиту;
- реалізувати повернення результатів пошуку в якості результатів виконання web-методу (і) web-сервісу, обґрунтувати тип значення.

2. Створити і протестувати клієнт, який використовує web-сервіс. Для цього необхідно:

- реалізувати введення користувальницької частини пошукового запиту;
- результати роботи web-сервісу відобразити у вигляді списку перших 10 релевантних посилань;

- при виклику web-методів web-сервісу використовувати механізм асинхронного виклику;

3. Навести в звіті наступне:

- приклад SOAP і HTTP POST запитів і відповідей для будь-якого web-методу web-сервісу, що приймає параметри і повертає результат;
- опис методу рішення задачі, вхідних і вихідних даних;
- лістинг вихідного коду файлу класу web-сервісу з коментарями;
- лістинг вихідного коду клієнта, призначеного для тестування web-сервісу, з коментарями;
- екранну форму клієнту в процесі тестування web-сервісу.

Вибір варіанту по кожному з параметрів здійснюється за формулою:

Номер залікової книжки MOD кількість варіантів + 1

Таблиця 2 – Варіанти завдань

№ варіанта	**Предметна область**	**Клієнт (на вибір студента)**
1	Пошук по сайтам, що входять у доменну зону **com.ua**	Windows Forms Application
2	Пошук по сайтам, що входять у доменну зону **gov.ua**	ASP .NET Web Application
3	Пошук по сайту **https://dou.ua/**	
4	Пошук по сайту **https://habr.com/**	ASP .NET Web Site
5	Пошук по сайту **https://msdn.microsoft.com/ru-ru/**	

Зміст звіту

1. Назва та мета роботи.
2. Методика проведення роботи з графічними результатами.
3. Письмові відповіді на контрольні питання.

Контрольні питання

1. Що називається web-сервісом?
2. Перерахуйте стандарти для web-сервісів.
3. Що таке SOAP, WSDL, UDDI?
4. Чи можна використовувати доступ до БД з web-сервісів?
5. Яку роль відіграє ім’'я посилання на службу в Windows-застосунку?
6. Перерахувати порядок реєстрації web-сервісу.
7. Перерахувати порядок використання web-сервісу.
8. Наведіть приклади, де можна використовувати Web-сервіси.
9. Перерахуйте переваги і недоліки Web-сервісів.

Література: [9–11].

Лабораторна робота № 5

Тема. Централізована система керування версіями Subversion

Мета: ознайомлення та набуття практичних навичок роботи з централізованою системою контролю версій Subversion.

Короткі теоретичні відомості

Для організації колективної роботи над програмним проектом розробниками, як правило, використовується певна система контролю версій. Призначення такої системи – дати змогу названому колективу здійснити за потреби повернення до певної попередньої версії проекту. Передумовами до цього можуть бути помилки, виявлені в результаті внесення змін до проекту.

Існує чимало різноманітних систем контролю версій. Найпоширеніші із них – Subversion (SVN) та Git. Названі системи є демонстраціями двох принципово різних підходів до здійснення контролю версій: SVN –

централізованого, а Git – децентралізованого підходу [12]. Кожен із підходів має як переваги, так і недоліки. Git, наприклад, характеризується більшими (у порівнянні із SVN) витратами обчислювальних ресурсів (пам'яті), проте його децентралізованість сприяє стійкості до виходу із ладу окремих обчислювальних вузлів мережі з розміщеними на них версіями проекту, використовуваних колективом розробників.

Ця лабораторна робота присвячена централізованій системі контролю версій SVN [13]. Централізованість полягає у виділенні окремого вузла – серверу, на якому розміщується репозитарій проекту. Проект при цьому розглядається як деревовидна структура, елементами якої є файли і каталоги. На клієнтських вузлах при цьому зберігаються копії (робочі копії) репозитарію.

Використання SVN дає можливість відслідковувати історію внесення змін до файлів, каталогів розробниками та, за потреби (виникнення помилок), виконувати скасування внесених змін.

Розробником SVN є компанія CollabNet. В цій лабораторній роботі використовувалась версія Subversion 1.8.10. Актуальну версію можна завантажити за посиланням https://subversion.apache.org/.

Базові поняття SVN:

– *репозитарій* – центр збереження інформації, розміщений на сервері; контент репозитарію (файли, каталоги) представлений у вигляді деревовидної структури; репозитарій можна розглядати як різновид файлового сервера;

– *робоча копія (РК)* – копія репозитарію, розміщена на клієнтському вузлі. Робочу копію можна розглядати як локальне робоче середовище клієнта.

РК – дерево каталогів із набором файлів [14]. Кожна РК містить службовий каталог .svn, що містить дані про зміни у РК.

Архітектуру SVN, що ґрунтується на поняттях репозитарію та РК, наведено на рис. 4.

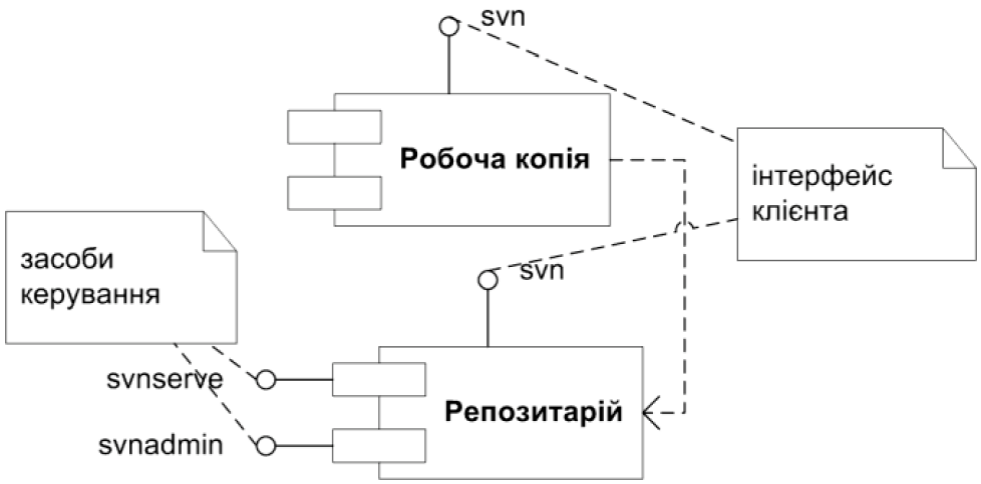

Рисунок 4 – Компоненти та утиліти SVN

Ключові утиліти SVN – svn, svnserve, svnadmin – представлено на рис. 4. Їх призначення роз'яснено в табл. 3. Користування наведеними утилітами можливе, наприклад, за рахунок інсталяції збірки CollabNetSubversion-client-1.7.8-1-Win32.

Таблиця 3 – Утиліти SVN

Утиліти	**Призначення**
svn	Консольний інтерфейс клієнтського командного рядку
svnserve	Відкриття доступу до створеного репозитарію
svnadmin	Створення та налаштування репозитарію
svnversion	Одержання даних про поточну версію РК
svnlook	Перегляд структури та вмісту репозитарію

Утиліта `svnserve` – це спеціалізований сервер, що забезпечує доступ до репозитарію через протокол TCP/IP. Взаємодія з репозитарієм можлива із використанням протоколів, наведених в табл. 4.

Таблиця 4 – Протоколи взаємодії з репозитарієм

Протоколи	**Призначення**
`file:///`	Доступ до локального репозитарію
`svn://`	Спеціалізований спрощений протокол доступу до svnserve-репозитарію
`svn+ssh://`	Взаємодія із шифруванням трафіку
`http://`	Доступ з використанням протоколу WebDAV
`https://`	Доступ на основі SSL-шифрування

При виконанні роботи будуть використовуватися протоколи file:/// і svn://.

Взаємодія з репозитарієм може здійснюватися згідно однієї з наступних моделей:

– блокування/модифікація/розблокування – одночасна взаємодія з репозитарієм декількох клієнтів (розробників) неможлива;

– копіювання/модифікація/злиття – на клієнтських вузлах створюються РК репозитарію; кожний із розробників працює із власною копією, якими потім модифікується репозитарій. Ця модель буде використовуватися в роботі.

Модель з блокуванням доречно використовувати при роботі із бінарними файлами (звукові, графічні), де неможливо об’єднувати конфліктні зміни [13].

Контроль версій у SVN здійснюється на основі віртуальної файлової системи, що дозволяє відслідковувати зміни ієрархічних структур каталогів у часі. Файлам та каталогам ставляться у відповідність множини метаданих (даних про дані), що дозволяє фіксувати відмінності поточної ревізії (версії) проекту від попередньої.

У випадку використання протоколу `svn://` (табл. 4) потрібно здійснювати запуск серверу `svnserve` – за допомогою однойменної утиліти. Правила взаємодії з репозитарієм при цьому регламентуються вмістом конфігураційного файлу *svnserve.conf*, розміщеного в каталозі *conf* репозитарію

Лістинг 1 – Вміст конфігураційного файлу репозитарію

```
[general]
anon-access = write
auth-access = write
password-db = passwd
realm = repository No1
```

Модифікація файлу *svnserve.conf* виконується шляхом зміни значень конфігураційних параметрів (табл. 5).

Таблиця 5 – Параметри конфігураційного файлу *svnserve.conf*

Параметри	**Призначення**
`anon-access`	Регламентація прав доступу для неавторизованих користувачів: `none` – доступ заборонено, `read` – внесення змін заборонено, `write` – повний доступ

Продовження таблиці 5

`auth-access`	Регламентація прав доступу для авторизованих користувачів. Допустимі значення ті самі
`password-db`	Задає назву файлу, що містить рядки вигляду `user_name = password`
`realm`	Унікальний ідентифікатор репозитарію, значенням якого є рядок

Типовий сценарій роботи з репозитарієм можна представити наступним чином:

– створення репозитарію – `svnadmin create \path`, де `\path` – шлях до каталогу, в якому планується розмістити репозитарій;

– імпортування структури файлів і каталогів до створеного репозитарію – `svn import source destination –m mark`, де source – шлях до кореневого каталогу деревовидної структури файлів і каталогів, призначених для імпорту, `destination` – шлях до каталогу з репозитарієм із зазначенням одного з

протоколів, наведених в табл. 4, `mark` – рядок, що унікальним чином ідентифікує здійснювану операцію;

– перегляд вмісту репозитарію, використовуючи команду `svn list` або svnlook tree. У першому випадку вміст буде виведено у вигляді списку, у другому – у вигляді деревовидної структури. Шаблони використання названих команд наступні: `svn list file:///path` і `svnlook tree \path`, відповідно. Для команди `svn list`, на відміну від команди `svnlook tree`, слід вказувати абсолютний шлях до репозитарію;

– опціональний запуск серверу svnserve, що дасть змогу надалі користуватися спеціалізованим протоколом доступу `svn://`. Формат відповідної команди наступний: `svnserve –d –r \path`;

– створення РК репозитарію, що дасть змогу організувати взаємодію з останнім згідно моделі *копіювання/модифікація/злиття*. Для цього використовується команда `svn checkout` або команда `svn co`. Наприклад, наступним чином можна створити РК, яка міститиметься у каталозі *repo_copy*, з використанням протоколу `svn://`:

`>>svn co svn://127.0.0.1/E:/.../repo E:\...\repo_copy`;

– модифікація РК;

– внесення змін із РК до репозитарію – виконання операції злиття, коли до репозитарію додаються нові файли і каталоги, зокрема здійснюється заміщення файлів і каталогів новими ревізіями. Для цього виконується команда `svn commit`, разом із зазначенням рядка-ідентифікатора операції злиття. Наприклад, команда `svn commit –m "1st_commit"` здійснює злиття РК і репозитарію, що ідентифікується рядком `"1st_commit"`.

Модифікація РК полягає у виконанні наступних дій (табл. 6):

– оновлення РК до актуальної версії репозитарію – командою `svn update`. Потреба в цьому ґрунтується на колективному використанні репозитарію – коли певний користувач здійснює `commit`-операцію, інші РК перестають бути актуальними;

– внесення змін до РК – за рахунок виконання наступних команд: `svn add`, `svn delete`, `svn copy` і `svn move`;

– перегляд змін в РК – команди `svn status` і `svn diff`;

– виправлення помилок, зроблених при внесенні змін до РК – команда `svn revert`;

– вирішення локальних конфліктів файлів і каталогів, обумовлених внесенням змін до РК – команда `svn resolve`;

Таблиця 6 – Деякі команди модифікації РК

Команди	**Призначення**
`svn add`	Додавання нового файлу (каталогу) до РК
`svn info`	Переглянути інформацію про поточну версію РК
`svn revert`	Скасувати внесені зміни
`svn diff`	Порівняння двох версій РК. Наприклад: `svn diff -r1:2` – визначити відмінність 1-ї версії від 2-ї. Додані рядки файлу позначатимуться символом «+», а видалені – символом «–«
`svn status`	Одержання станів файлів та каталогів РК. Станам відповідають такі прапорці: A (*added*) – файл (каталог) додано; D (*deleted*) – файл (каталог) видалено; M (*modified*) – файл (каталог) модифіковано; R (*replaced*) – файл (каталог) заміщено.

Порядок виконання роботи

1. Перейти до власного робочого каталогу, де створити нову директорію *logs* – для збереження лістингів про виконання роботи.

2. Відкрити Far Manager (або Total Commander), за допомогою якого перейти до власного робочого каталогу, де переглянути інформацію про версію

SVN і доступні команди:

```
>>svn help > 1.02_svn_help.log
```

Зауваження: >> – символ запиту командного рядка, що не підлягає набору.

3. Знаходячись у власному робочому каталозі, створити новий репозитарій *repo*:

```
>>svnadmin create repo
```

4. Переглянути вміст створеного однойменного каталогу *repo*. Замістити вміст конфігураційного файлу *conf\svnserve.conf* вмістом лістингу 1.

5. Імпортувати до репозитарію проект, створений при виконанні лабораторної роботи № 1. Для цього слід скористатися протоколом `file:///` згідно наступного шаблону:

```
>>svn import f:\path\to\project file:///f:/path/to/repo
-m "first import" > logs\1.05_import.log
```

6. Переглянути лістинг про результати імпорту проекту до репозитарію у створеному файлі *import_log.txt*. У випадку успіху лістинг міститиме інформацію про файли і каталоги імпортованого проекту, додані до репозитарію. Лістинг має завершуватись рядком `Committed revision 1`, де 1 – початковий номер версії проекту, що надалі буде інкрементуватися.

7. Переглянути вміст репозитарію проекту у вигляді деревовидної структури:

```
>>svnlook tree repo > logs\1.07_svnlook_tree.log
```

8. Переглянути вміст створеного репозитарію у вигляді списку. Для цього слід зазначати повний шлях до каталогу репозитарію, з використанням протоколу file:///:

```
>>svn list file:///f:/path/to/repo logs\1.08.log
```

або

```
>>svn list file://localhost/f:/path/to/repo
```

9. Виконати дії попереднього кроку 8 з використанням протоколу `svn://` замість протоколу `file:///`:

```
>>svn list svn://127.0.0.1/f:/path/to/repo
```

Виникне помилка – неможливість підключення до репозитарію. Аби позбутися її, необхідно спочатку виконати крок 10.

10. Для використання протоколу `svn://`, що надасть змогу користуватися репозитарієм з іншого комп'ютера (наприклад, для створення робочої копії) слід попередньо відкрити доступ до репозитарію:

```
>>svnserve -d -r f:\path\to\repo
```

Зауваження: для зазначеної команди `svnserve` зручно відкрити окреме консольне вікно, яке має залишатися відкритим упродовж подальшого виконання роботи.

За успішного відкриття доступу до репозитарію результат виконання завдання 9 буде ідентичним лістингу *1.8_log.txt*.

11. Занести до звіту інформацію про призначення прапорців `-d` та `-r` використаної команди `svnserve`.

12. У власному робочому каталозі на одному рівні ієрархії з каталогом *repo* створити нову директорію *repo_copy* – контейнер робочої копії.

13. Для роботи з репозитарієм згідно моделі *копіювання/модифікація/злиття*, створити робочу копію *repo_copy* репозитарію *repo* шляхом використання команди `svn checkout` або еквівалентної команди `svn co`:

```
>>svn checkout svn://localhost/f:/path/to/repo
f:\path\to\repo_copy > logs\1.13_repo_copy_log.txt
```

За успішного створення РК каталог *repo_copy* міститиме проект, імпортований до репозитарію. У лістингу *1.12_repo_copy_log.txt* міститимуться дані про додані файли і каталоги проекту, на що вказуватимуть прапорці A. Завершальний рядок міститиме інформацію про поточну ревізію РК – `Checked out revision 1`, номер якої наразі співпадатиме з номером версії репозитарію. Подальші маніпуляції мають здійснюватися безпосередньо над РК.

14. Модифікувати конфігураційний файл `svnserve.conf` репозитарію (лістинг 1), замінивши значення `write` поля `anon-access` значенням none.

15. Спробувати створити нову РК *repo_copy_2*. Прокоментувати одержуваний при цьому лістинг.

Зауваження: для успішного виконання завдання 15 необхідно створити обліковий запис у файлі *repo\conf\passwd* у форматі логін = пароль.

16. Внести до конфігураційного файлу *svnserve.conf* зворотні зміни.

17. Створити у каталозі *test* робочої копії *repo_copy* файл *test\test\Tester.java*, який містить код з файлу *CalcPanel.java* з лабораторної роботи № 1, і новий файл модульного тесту *PiTester.java* (лістинг 2).

Лістинг 2 – Клас модульного тесту

```
package test;
import org.junit.Test;
import calc.*;
import static org.junit.Assert.*;
public class PiTester {
@Test
27
public void test() {
    PiCalc calc = new PiCalc();
    assertEquals(calc.getResult(10), pi, 0);
}
public static final double pi = 3.14159265;
}
```

Призначення тесту – перевірка роботи методу `getResult()` обчислення значення π . Додати створений файл до РК:

```
>>svn add PiTester.java
```

Про успіх додавання свідчитиме прапорець A поряд з назвою доданого файлу: `A PiTester.java`.

18. Модифікувати файл *Tester.java*, додавши до нього *javadoc*-документацію перед визначенням класу `Tester`:

```
/**
* unit test container for getResult() method
*/
```

19. Переглянути стан робочої копії *repo_copy*:

```
>>svn st –u –v repo_copy > logs\1.19_svn_st.log
```

У списку-лістингу навпроти доданого та модифікованого файлів РК зазначатимуться, відповідно, прапорці A та M (*modified*).

20. Скопіювати вміст файлу *Tester.java* до нового файлу *PiTester.java*.

21. Видалити модульний тест *Tester.java* з РК:

```
>>svn del --force Tester.java
```

Індикатором успіху буде прапорець D поряд з назвою видаленого файлу, який з'явиться замість прапорця M: `D Tester.java`. При цьому файл буде видалено не лише з РК, а й взагалі.

22. Відновити втрачений файл Tester.java, скориставшись командою `svn revert`:

```
>>svn revert Tester.java
```

Видалений файл буде відновлено до початкового стану, визначеного або командою `svn co`, або останньою командою `svn update`. Це означає, що додана *javadoc*-документація буде відсутня.

23. Знову виконати завдання 21, після чого створити новий файл *Tester.java* із копією вмісту файлу *PiTester.java*, додавши його до РК. При перегляді стану РК навпроти файлу *Tester.java* має з'явиться новий прапорець R, що означає заміщений (*replaced*).

24. Здійснити злиття модифікованої РК з репозитарієм. Для цього слід виконати команду `commit`:

```
>>svn commit --force-log -F *.java
```

У результаті злиття буде сповіщено про додавання нового файлу *PiTester.java* з робочої копії *repo_copy* до репозитарію *repo* і про заміщення вже існуючого файлу *Tester.java*. Номер версії репозитарію при цьому інкрементується:

```
Committed revision 2.
```

25. Оновити раніше створену другу робочу копію *repo_copy_2* внесеними до репозитарію модифікаціями, перейшовши до однойменного каталогу та виконавши команду

```
>>svn update
```

У відповідному лістингу, попри інформацію про доданий файл, зазначатиметься також оновлена версія РК:

```
Updated to revision 2.
```

26. Переглянути детальнішу інформацію про поточну версію РК *repo_copy_2*:

```
>>svn info
```

27. Відновити попередню версію РК і переглянути інформацію про неї:

```
>>svn update –r1
>>svn info
```

28. Переглянути відмінності між версіями 1 і 2 РК. Для цього у власному робочому каталозі слід створити і виконати командний файл *1.28_svn_diff.cmd* з наступним контентом:

```
>>cd repo_copy
>>svn diff -r1:2 > f:\path\to\...\logs\1.27_diff.log
```

Результатом виконання командного файлу має бути вміст лістингу *1.27_diff.log*, де символом «+» позначатимуться додані рядки *javadoc*-документації завдання 18.

29. Змінити формат команди `svn diff` створеного командного файлу, у кості аргументу якої зазначити рядок `-r2:1` замість рядку `-r1:2`. Проаналізувати новий результат виконання командного файлу. Коментарі занести до звіту.

Зміст звіту

1. Назва та мета роботи.
2. Методика проведення роботи з графічними результатами.
3. Письмові відповіді на контрольні питання.

Контрольні питання

1. Дати визначення поняттям репозитарій і робоча копія.
2. Охарактеризувати моделі контролю версій.
3. Навести і прокоментувати архітектуру SVN.

4. Навести призначення утиліт *svn.exe*, *svnadmin.exe* і *svnserve.exe*.

5. Навести призначення та параметри конфігураційного файлу *svnserve.conf*.

6. Прокоментувати протоколи доступу до репозитарію.

7. Навести призначення та приклади команд `svn commit` і `svn checkout`.

8. Команда `svn status`. Прапорці станів файлів та каталогів.

9. Команди `svn list` і `svnlook tree`. Навести призначення та приклади.

10. Навести послідовність кроків для імпорту файлів та каталогів з репозитарію з використанням NetBeans IDE.

Література: [12–14].

Лабораторна робота № 6

Тема. Основи роботи з Atlassian Bitbucket для управління кодом у Git

Мета: ознайомлення та набуття практичних навичок роботи з таким інструментом управління кодом у Git для команд, як Atlassian Bitbucket.

Короткі теоретичні відомості

Git – це набір консольних утиліт, які відстежують і фіксують зміни у файлах (найчастіше мова йде про вихідний код програм, але можна використовувати його для будь-яких файлів). З його допомогою можна відкотитися на більш стару версію проекту, порівнювати, аналізувати, зливати зміни і багато іншого. Цей процес називається контролем версій. Існують різні системи для контролю версій: SVN, Mercurial, Perforce, CVS, Bitkeeper та ін.

Git є розподіленим, тобто не залежить від одного центрального сервера, на якому зберігаються файли. Замість цього він працює повністю локально, зберігаючи дані в папках на жорсткому диску, які називаються репозитарієм. Тим не менш, можна зберігати копію репозитарія онлайн, що полегшує роботу над одним проектом для кількох людей. Для цього використовуються сайти на

кшталт github і bitbucket.

Bitbucket – веб-сервіс для хостингу проектів та їх спільної розробки, заснований на системі контролю версій Mercurial і Git. За призначенням і основним пропонованим функцій аналогічний GitHub, від якого відрізняється, з одного боку, меншою користувальницької базою, а з іншого – має певні переваги у плані розміщення непублічних репозиторіїв – можливістю їх безкоштовного хостингу з обмеженням на розмір команди не більше п’яти осіб і менша орендна плата при більшому розмірі команди, а також управління правами доступу на рівні окремих гілок проекту.

Якщо необхідно працювати над одним проектом на двох комп’ютерах (наприклад, на роботі та вдома), то в цьому випадку необхідно створити репозиторій на Bitbucket, завантажити туди проект. Після чого вдома ***склонувати*** його, іншими словами приєднатись до цього проекту.

У git, як і в інших системах контролю версій, є можливість створювати розгалуження версій – ***branch***. Розгалуження корисно, наприклад, коли необхідно проробити якийсь функціонал свого проекту, але розробник не впевнений в тому, що результат буде правильним.

Розгалудження корисно також у випадку, коли необхідно виправити помилку в проекті або дописати певний функціонал.

Класичний приклад використання гілок, ви вирішуєте кілька проблем одночасно. Наприклад, ви доповнюєте сайт новим функціоналом і вас просять терміново вирішити баг на поточній версії сайту. В такому випадку ви створюєте ще одну гілку від master для рішення бага, переходите на неї і працюєте на ній, до тих пір, поки не виправите баг. Після чого поєднуєте гілка master з гілкою де виправляли баг, далі переходите на колишню гілка де доповнювали функціонал і працюєте далі. Важливо, розуміти, що гілки об'єднуються вдало тільки тоді, коли правки або в різних файлах, а якщо в одному файлі, то в різних його місцях.

Корисні команди:

Дізнатися про підключених видалених репозиторіїв

```
git remote -v
```

Видалити віддалений репозиторій з ім’ям origin

```
git remote rm origin
```

Список commit до поточного HEAD (ваше поточне положення на гілці)

```
git log
```

Список всіх commit-ів

```
git log --all
```

Поточний стан, git укаже поточні змінені файли і проблеми

```
git status
```

Перехід на інший commit

```
git checkout xxxx
```

xxxx це частина номера commit-a, дізнатися список номерів можна за допомогою `git log`

Повернутися на верхівку гілки (за замовчуванням ***master***)

```
git checkout master
```

Видалити файл з індексу

```
git rm ім’я_файлу
```

Якщо ви фізично видалили файл і захотіли зробити commit вашого проекту, git видасть попередження.

Для того, щоб commit був вдалий, слід видалити файл index.php з індексу за допомогою `git rm index.php`, після чого git не буде його відстежувати.

Видалити останній комміт, невідправлений на Bitbucket, і перейти на стан попереднього

```
git reset -hard HEAD
```

Порядок виконання роботи

Створити проект на Bitbucket:

1.1. Завантажити git клієнт (http://git-scm.com/download/win), встановити його.

1.2. Запустити консольний git, задати своє ім’я та e-mail, за допомогою команди:

```
git config --global user.name "FIRST_NAME LAST_NAME"
git config --global user.email "MY_NAME@example.com"
```

1.3. За допомогою команди cd перейти у директорію, де буде знаходитись проект. Ініціалізувати дану директорію, іншими словами, сповістити git, що працювати треба саме в цій директорії.

```
git init
```

1.4. Додати усі файли у директорії в індекс, іншими словами указати git, у яких файлів (директорій) він повинен відслідковувати зміни.

```
git add *
```

Якщо потрібно видалити файл з індексу, тобто фізично він залишиться, але git не буде фіксувати зміни в ньому, потрібно ввести `git rm ім'я_файлу`.

1.5. Зробити перший commit, іншими словами указати git, що потрібно зафіксувати зміни.

```
git commit -m 'коментар коммітів'
```

Коментар до commit-у писати обов'язково.

1.6. Підключити git до сервера bitbucket. Для цього зареєструватись на **bitbucket** і створити репозиторій, із запропонованих варіантів вибираємо «I have an existing project» і вводимо команди, запропоновані на рис. 5, у git.

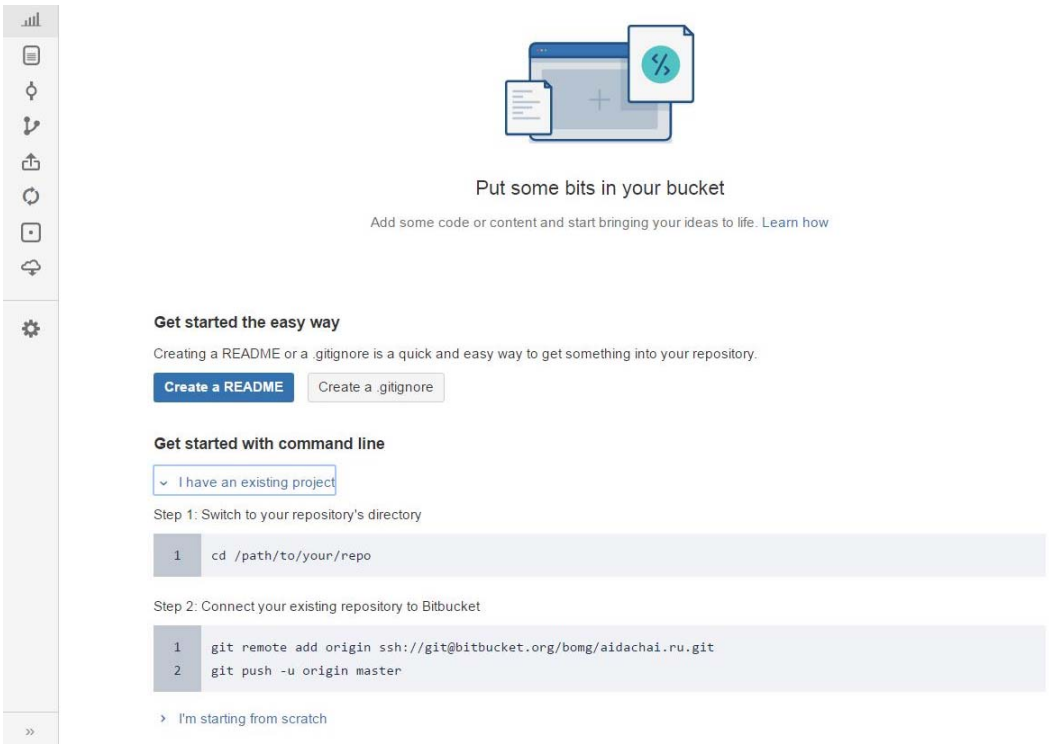

Рисунок 5 – Команди для реєстрації на bitbucket і створення депозиторію

Якщо виникає помилка, спробуйте замість ssh адреси на репозиторій використовувати https. Для цього переведіть у верхньому правому куті вкладку з ssh на https і в команді `git remote add origin` використовуйте цю адресу.

Замість origin можна використовувати будь-яке інше ім'я, це просто ім'я репозиторію.

P.S. На поточний момент на сайті bitbucket немає можливості перемкнути адресу з ssh на https у верхньому правому куті. Тому формуємо https адресу самостійно.

Було: `git remote add origin git@bitbucket.org: accountname / reponame.git`

Стало: `https://bitbucket.org/accountname/reponame.git`

Повинно з'явитися діалогове вікно, де необхідно ввести логін і пароль, який ви використовуєте на bitbucket.

2. Приєднатися до вже існуючого проекту

2.1 Завантажити git клієнт (http://git-scm.com/download/win), встановити його.

2.2 Запустити консольний git, задати своє ім'я та e-mail, за допомогою команди:

```
git config --global user.name "FIRST_NAME LAST_NAME"
git config --global user.email "MY_NAME@example.com"
```

2.3 За допомогою команди cd перейти у директорію, де буде знаходитись проект. Ініціалізувати цю директорію, іншими словами, сповістити git про те, що працювати треба саме в цій директорії:

```
git init
```

2.4 Клонувати репозиторій за допомогою команди, іншими словами, приєднатися до проекту.

```
git clone https: //bomg@bitbucket.org/bomg/aidachai.ru.git
```

Посилання на репозиторій можна дізнатися на сайті Bitbucket, при виборі Clone, під логотипом репозитарія (рис. 6).

2.5 Повинно з'явитися діалогове вікно, де необхідно ввести логін і пароль, який використовується в Bitbucket.

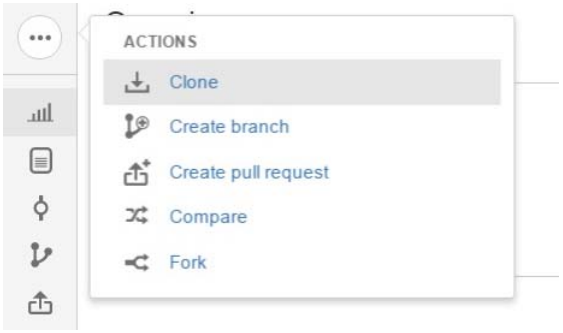

Рисунок 6 – Посилання на репозиторій

2.6 Завантажити проект с bitbucket

```
git pull origin master
```

3. Проведення робочих маніпуляцій.

3.1 Перед роботою над проектом, перш за все, необхідно оновити свою робочу копію, яка зберігається на локальному комп'ютері, тому що можливо, що на сервері хтось вже попрацював над проектом. Для цього необхідно запустити git, перейти за допомогою cd у робочу директорію та виконати:

```
git pull origin master
```

Origin це ім’я репозитарія, а master – ім’я гілки.

3.2 Виконати необхідні дії над проектом як зазвичай на своєму локальному комп’ютері.

3.3 Якщо необхідно зафіксувати проміжний етап своєї роботи, перше, слід підготувати файли до commit-у (фіксації результату), іншими словами проіндексувати їх:

```
git add *
```

Якщо файл було видалено з проекту, то його потрібно видалити з індексу за допомогою `git rem ім’я_файлу` або використовувати `git add -u *` для ігнорування віддалених файлів.

Далі виконати саму фіксацію:

```
git commit -m 'коментар'
```

3.4 Після декількох commit-ів, результат останнього з них можна завантажити на BitBucket:

```
git push origin master
```

У цей момент на Bitbucket з’являтися нова версія проекту, яку можна

подивитися на сайті.

4. Робота з гілками

4.1 Створити окрему гілку та виконати перехід на неї за допомогою команд:

```
git branch new_fork
git checkout new_fork
```

4.2 Виконати необхідні дії над проектом.

4.3 Поєднати нову гілку зі старою за допомогою команди `merge`. Для цього необхідно перейти на головну гілку *master* і об'єднати з нею гілку *new_fork*:

```
git checkout master
git merge new_fork -m 'ваш коментар'
```

У результаті буде створений новий commit, який об'єднує дві гілки (master і new_fork). Якщо щось не вийшло, то можна повернутись до *master* і видалити гілку *new_fork*:

```
git checkout master
git branch -d new_fork
```

Зміст звіту

1. Назва та мета роботи.
2. Методика проведення роботи з графічними результатами.
3. Письмові відповіді на контрольні питання.

Контрольні питання

1. Які альтернативи Bitbucket вам відомі?
2. Перелічити основні функції, що виконує Bitbucket.
3. Що таке комміт? Які дії можна виконувати над ними?
4. Перелічити основні команди Git.

Література: [16, 17].

2 КРИТЕРІЇ ОЦІНЮВАННЯ ЯКОСТІ ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ СТУДЕНТАМИ

У 8-му семестрі студенти виконують 6 лабораторних робіт. Загальна кількість балів, яку отримують студенти за виконання лабораторних робіт, складає 32,5 балів – сума за захист виконаних лабораторних робіт (максимально по 5,4 балів на кожну лабораторну роботу).

Шкала оцінювання: національна та ECTS

<table>
<tr><td rowspan="2">Сума балів за всі види навчальної діяльності</td><td rowspan="2">Оцінка ECTS</td><td colspan="2">Оцінка за національною шкалою</td></tr>
<tr><td>Для екзамену, курсового проекту (роботи), практики</td><td>Для заліку</td></tr>
<tr><td>90–100</td><td>A</td><td>Відмінно</td><td rowspan="5">Зараховано</td></tr>
<tr><td>82–89</td><td>B</td><td rowspan="2">Добре</td></tr>
<tr><td>74–81</td><td>C</td></tr>
<tr><td>64–73</td><td>D</td><td rowspan="2">Задовільно</td></tr>
<tr><td>60–63</td><td>E</td></tr>
<tr><td>35–59</td><td>FX</td><td>Незадовільно з можливістю повторного складання</td><td>Не зараховано з можливістю повторного складання</td></tr>
<tr><td>0–34</td><td>F</td><td>Незадовільно з обов’язковим повторним вивченням навчальної дисципліни</td><td>Не зараховано з обов’язковим повторним вивченням навчальної дисципліни</td></tr>
</table>

СПИСОК ЛІТЕРАТУРИ

1. Шилдт Г. Java 8: руководство для начинающих / Г. Шилдт. – М. : «Вильямс», 2015. – 720 с.
2. Кузен К. Современный Java. Рецепты программирования / К. Кузен. – М. : «ДМК Пресс», 2016. – 274 с.
3. Седжвик Р. Computer Science. Основы программирования на Java, ООП, алгоритмы и структуры данных / Р. Седжвик, К. Уэйн. – СПб. : «Питер», 2016. – 1072 с.
4. Оберг Р. Дж. Технология COM+. Основы и программирование / Р. Дж. Оберг. – М. : «Вильямс», 2000. – 480 с.
5. Артемов М.А. Основы COM-технологий : учебно-методическое пособие / М.А. Артемов и др. – Воронеж : ИПЦ ВГУ, 2007. – 84 с.
6. Бокс Д. Сущность технологии COM. Библиотека программиста / Д. Бокс. – СПб. : Питер, 2001. – 400 с.
7. Одиночкина С. В. Основы технологий XML / С. В. Одиночкина. – СПб. : НИУ ИТМО, 2013. – 56 с.
8. Холзнер С. XML. Энциклопедия / С. Холзнер. – СПб. : «Питер», 2004. – 1101 с.
9. Ньюкомер Э. Веб-сервисы. Для профессионалов / Э. Ньюкомер. – СПб. : «Питер», 2003. – 256 с.
10. Шапошников И. Web-сервисы Microsoft .NET / И. Шапошников. – СПб. : «БХВ-Петербург», 2002. –334 с.
11. Машнин Т. Web-сервисы Java / Т. Машнин. – СПб. : «БХВ-Петербург», 2012. – 560 с.
12 Chacon S. Pro Git / Scott Chacon. – NY.: Apress, 2009. – 288 p.
13 Collins-Sussman B. Version Control with Subversion / B. Collins-Sussman, B.W. Fitzpatrick, C.M. Pilato. – [2nd ed.]. – CA.: O'Reilly Media, Inc, 2008. – 432 p.

14. Управление версиями в Subversion [Електронний ресурс]. – Режим доступу: http://svnbook.red-bean.com/nightly/ru/.
15. Гонсалвес Э. Изучаем Java EE 7 / Э. Гонсалвес; пер. с англ. Е. Зазноба [и др.]. – СПб.: Питер, 2014. – 640 с.
16. Обучающие материалы по Git [Электронный ресрус]. – Режим доступа: https://ru.atlassian.com/git/tutorials/atlassian-git-cheatsheet.
17. 6 секретов Bitbucket [Электронный ресрус]. – Режим доступа: https://habr.com/ru/post/278547/.

Методичні вказівки щодо виконання лабораторних робіт з навчальної дисципліни «Інженерія програмного забезпечення» (Частина ІІ) для студентів денної форми навчання зі спеціальності 123 – «Комп'ютерна інженерія»

Укладач к. т. н., доц. О. Г. Славко

Відповідальний за випуск зав. кафедри КІС А. В. Луговой

Підп. до др. _____. Формат 60×84 1/16. Папір тип. Друк ризографія.
Ум. друк. арк. ____. Наклад _____ прим. Зам. №_____. Безкоштовно.

Видавничий відділ
Кременчуцького національного університету
імені Михайла Остроградського
вул. Першотравнева, 20, м. Кременчук, 39600