Харківський національний університет імені В. Н. Каразіна
Факультет комп'ютерних наук
Кафедра штучного інтелекту та програмного забезпечення

ЗВІТ
З РОЗРАХУНКОВО-ГРАФІЧНОЇ РОБОТИ №2

дисципліна: «Крос-платформне
програмування»

Виконав: студент групи КС23
Травченко Сергій Миколайович

Перевірив: викладач кафедри ШІтаПЗ
Споров Олександр Євгенович

Харків
2024

Розрахунково-графічна робота №2 Java Beans

На даному занятті необхідно ознайомитися з основами компонентної технології JavaBeans, з правилами створення JavaBeans – компонентів, з основами їх налаштування та застосування в середовищах розробки.

Основні завдання

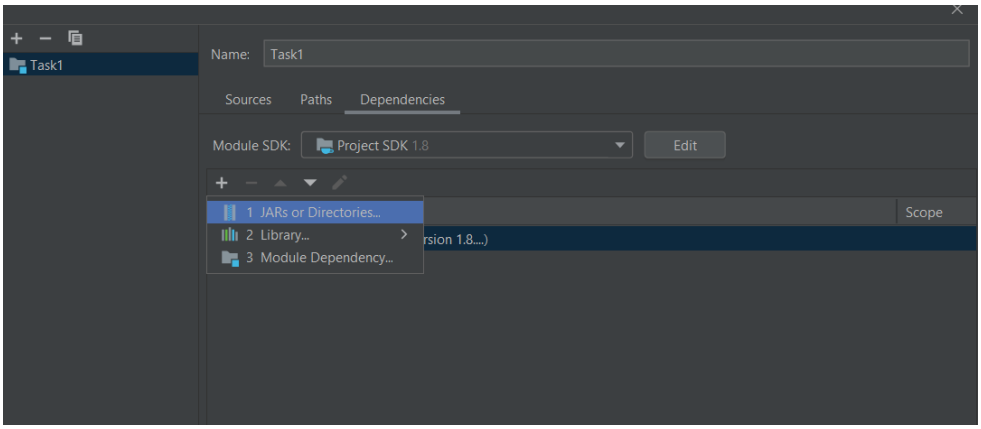
Завдання №1 Створити два компоненти JavaBeans для представлення зашумленого набору експериментальних даних з попереднього За в дан н я: компонент для табличного представлення набору даних та роботи з ним; компонент для графічного представлення набору даних. Використовуючи створені компоненти, написати додаток з графічним інтерфейсом користувача, призначений для перегляду та редагування зазначених наборів даних, збережених у XML-файлах. Під час створення додатку скористатися візуальним редактором графічного інтерфейсу з вибраного середовища розробки. Для того, щоб вирішити задачу, у новому проєкті Java спочатку створимо пакет для компонентів JavaBeans. Цей пакет можна назвати mybeans. У першу чергу, в цьому пакеті можна розмістити класи, які представляють дані з XML-файлу (клас Data для представлення рядка з даними та клас DataSheet, який представляю весь набір даних).

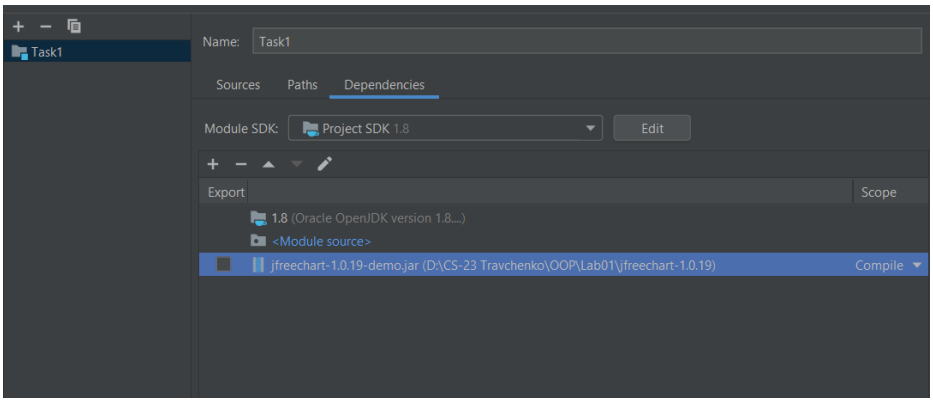
Результати виконання завдання №1 наведено:

1. У лістингу 1 – вихідний код програми;
2. На малюнку 1.1, 1.2 – процесс додавання бібліотеки JFreeChart;
3. На малюнку 1.3 – результат виконання програми;

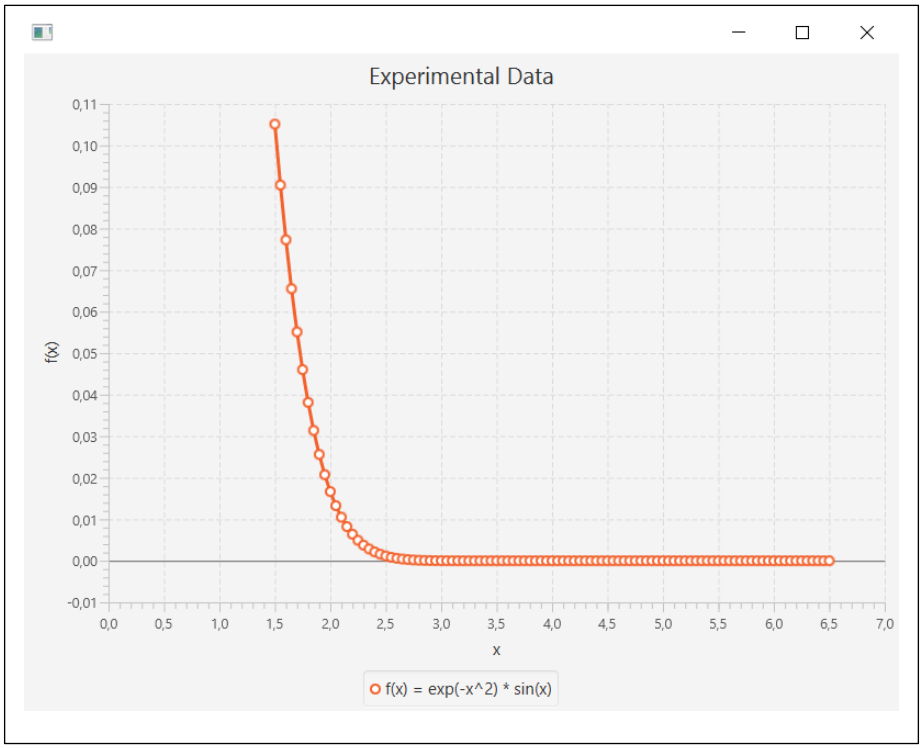
Лістинг 1. Вихідний код програми

```
@Override
public void start(Stage stage) {
    // Define the axes
    final NumberAxis xAxis = new NumberAxis();
    final NumberAxis yAxis = new NumberAxis();
    xAxis.setLabel("x");
    yAxis.setLabel("f(x)");

    // Create the line chart
    final LineChart<Number, Number> lineChart = new LineChart<>(xAxis, yAxis);
    lineChart.setTitle("Experimental Data");

    // Create a series for the data
    XYChart.Series series = new XYChart.Series();
    series.setName("f(x) = exp(-x^2) * sin(x)");

    // Generate experimental data and add it to the series
    double x = 1.5;
    while (x <= 6.5) {
        double y = Math.exp(-x * x) * Math.sin(x);
        series.getData().add(new XYChart.Data(x, y));
        x += 0.05;
    }

    // Add the series to the chart
    lineChart.getData().add(series);

    // Create the scene and add the chart to it
    Scene scene = new Scene(lineChart,  width: 800,  height: 600);
    stage.setScene(scene);
    stage.show();
```





Малюнок 1.1, 1.2 – додавання бібліотеки JFreeChart



Малюнок 1.3, – результат виконання завдання

Тепер перейдемо до створення першого компонента JavaBean, призначеного для
табличного відображення даних і для роботи з ними. Для цього створимо візуальний клас
Java (наприклад, DataSheetTable), який розширює стандартний клас панелі JPanel.
Зовнішній вигляд і склад компонента у середовищі розробки представлені на рисунку вище.
Для зручності розміщення компонентів замінимо стандартний менеджер компонування панелі на менеджер компонування BorderLayout. У південну область
(BorderLayout.SOUTH) додамо панель panelButtons для кнопок додавання (addButton) та
видалення (delButton) рядків таблиці. Після цього (спочатку необхідно впевнитись, що для
панелі panelButtons встановлений стандартний менеджер компонування FlowLayout) на
панель слід додати необхідні кнопки і налаштувати їх властивості. Крім того, у менеджері
компонування можна налаштувати бажану горизонтальну відстань між компонентами (у
додатку на рисунку цей параметр дорівнює 25). У центральну область панелі
DataSheetTable за необхідністю помістити панель прокрутки (JScrollPane scrollPane) і
встановити відображення вертикальних та горизонтальних смуг прокрутки:

```
scrollPane.setHorizontalScrollBarPolicy(
ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
scrollPane.setVerticalScrollBarPolicy(
ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED);
```

На панель прокрутки слід помістити таблицю JTable table. Як вже було вивчено у
попередньому семестрі, компонент JTable, який використовується для табличного
представлення і редагування даних, сам ці дані не містить, а лише відображає їх. У випадку,
коли необхідно організувати редагування даних, таблиця JTable отримує дані від об’єкта, що
реалізує інтерфейс TableModel. Для цього об’єкта слід створити окремий клас, в якому
повинна міститися структура даних, яка включає вміст клітинок таблиці. Для цього у
поточному пакеті створимо новий не візуальний клас моделі (DataSheetTableModel), який

розширює клас AbstractTableModel. Під час створення класу можна обрати опцію
Inherited abstract methods. Слід додати потрібні поля (поле для серіалізації serialVersionUID, поля для зберігання кількості рядків та стовпців у таблиці columnCount,
rowCount, «сховище даних» dataSheet) і створити необхідний набір методів доступу. Після
цього слід написати код перевизначених методів. (Слід продивитись документацію по
кожному перевизначеному методу). У результаті може вийти приблизно такий код: public class DataSheetTableModel extends AbstractTableModel {
private static final long serialVersionUID = 1L;
private int columnCount = 3;
private int rowCount = 1;
private DataSheet dataSheet = null;
String[] columnNames = {"Date", "X Value", "Y Value"};
public DataSheet getDataSheet() {
return dataSheet;
}
public void setDataSheet(DataSheet dataSheet) {
this.dataSheet = dataSheet;
rowCount = this.dataSheet.size();
}
@Override
public int getColumnCount() {
return columnCount;
}
@Override
public int getRowCount() {
return rowCount;
}
@Override
public String getColumnName(int column) {
return columnNames[column];
}
@Override
public boolean isCellEditable(int rowIndex, int columnIndex) {
return columnIndex >= 0;
}
@Override
public void setValueAt(Object value, int rowIndex, int columnIndex) {
try {
double d;
if (dataSheet != null) {
if (columnIndex == 0) {

```
dataSheet.getDataItem(rowIndex).setDate((String) value);
} else if (columnIndex == 1) {
d = Double.parseDouble((String) value);
dataSheet.getDataItem(rowIndex).setX(d);
} else if (columnIndex == 2) {
d = Double.parseDouble((String) value);
dataSheet.getDataItem(rowIndex).setY(d);
}
}
} catch (Exception ex) {}
}
@Override
public Object getValueAt(int rowIndex, int columnIndex) {
// TODO Auto-generated method stub
if (dataSheet != null) {
if (columnIndex == 0)
return dataSheet.getDataItem(rowIndex).getDate();
else if (columnIndex == 1)
return dataSheet.getDataItem(rowIndex).getX();
else if (columnIndex == 2)
return dataSheet.getDataItem(rowIndex).getY();
}
return null;
}
public void setRowCount(int rowCount) {
if (rowCount > 0)
this.rowCount = rowCount;
}
}
```

Після цього можна перейти до класу DataSheetTable і під’єднати до таблиці її
модель. Це можна зробити в палітрі налаштування компонента, а можна вказати рядки прямо
у коді. В результаті у конструкторі DataSheetTable з’явиться приблизно такий фрагмент:

```
table = new JTable();
tableModel = new DataSheetTableModel();
table.setModel(tableModel);
```

Для зручності роботи з компонентом DataSheetTable можна додати методи доступу
до моделі таблиці та метод, що оновлює таблицю.

```
public DataSheetTableModel getTableModel() {
return tableModel;
}
public void setTableModel(DataSheetTableModel tableModel) {
```

```
this.tableModel = tableModel;
table.revalidate();
}
public void revalidate() {
if (table != null) table.revalidate();
}
```

Оскільки даний компонент в подальшому може бути упакований в jar архів і розповсюджений як компонент JavaBeans без доступу до вихідного коду, слід якось передати зв'язаним із ним компонентам, якщо в результаті дій користувача зміниться стан «сховища даних», яке знаходиться у моделі таблиці DataSheetTableModel tableModel. Для цього можна скористатися готовими подіями, але ми створимо свою власну подію.

Спочатку створимо клас події. Цей клас, згідно зі специфікацією JavaBeans, повинен бути нащадком класу java.util.EventObject і мати назву ABCEvent, де ABC — назва події. Для нашої події, пов'язаної зі зміною стану «сховища даних» оберемо ім'я DataSheetChangeEvent.

```
public class DataSheetChangeEvent extends EventObject {
private static final long serialVersionUID = 1L;
public DataSheetChangeEvent(Object source) {
super(source);
}
}
```

Клас події не буде зберігати ніякої додаткової інформації: подія буде просто маркером того, що «сховище» якось змінилось. У конструкторі класу слід вказати джерело події — компонент, в якому ця подія виникла.

Далі слід описати інтерфейс слухача нашої події. Даний інтерфейс повинен бути реалізований клієнтами, зацікавленими у відстеженні цієї події. Згідно зі специфікацією JavaBeans, цей інтерфейс повинен бути нащадком інтерфейсу java.util.EventListener, який не має жодного метода. Інтерфейс нашої події може бути таким:

```
public interface DataSheetChangeListener extends EventListener {
public void dataChanged(DataSheetChangeEvent e);
}
```

В інтерфейсі був визначений лише один метод, який буде викликатись при зміні стану
«сховища». Цьому методу в якості параметра буде передано об’єкт нашої події
DataSheetChangeEvent. Таким чином, програміст, який реалізує інтерфейс слухача, зможе
дізнатись необхідні подробиці про подію.
Тепер залишається включити підтримку події в класі компонента, що генерує цю
подію. У нашому випадку цим класом буде клас моделі таблиці DataSheetTableModel.
У цьому класі створимо список, в якому будуть зберігатись всі слухачі, під’єднані до
компоненту. У разі виникнення події всі слухачі з цього списку отримають про нього
необхідну інформацію. Таким чином, до класу DataSheetTableModel додамо наступний код:

```
// список слухачів
private ArrayList<DataSheetChangeListener> listenerList = new
ArrayList<DataSheetChangeListener>();
// один універсальний об’єкт-подія
private DataSheetChangeEvent event = new DataSheetChangeEvent(this);
// метод, що приєднує слухача події
public void addDataSheetChangeListener(DataSheetChangeListener l) {
listenerList.add(l);
}
// метод, що від’єднує слухача події
public void removeDataSheetChangeListener(DataSheetChangeListener l) {
listenerList.remove(l);
}
// метод, що оповіщає зареєстрованих слухачів про подію
protected void fireDataSheetChange() {
Iterator<DataSheetChangeListener> i = listenerList.iterator();
while ( i.hasNext() )
(i.next()).dataChanged(event);
}
```

Крім цього, слід виправити методи, які змінюють «сховище даних» dataSheet для
того, щоб усі зацікавлені компоненти могли відреагувати на ці зміни:

```
public void setDataSheet(DataSheet dataSheet) {
this.dataSheet = dataSheet;
rowCount = this.dataSheet.size();
fireDataSheetChange();
}
@Override
```

```
public void setValueAt(Object value, int rowIndex, int columnIndex) {
// TODO Auto-generated method stub
//super.setValueAt(value, rowIndex, columnIndex);
try {
double d;
if (dataSheet != null) {
if (columnIndex == 0) {
dataSheet.getDataItem(rowIndex).setDate((String) value);
} else if (columnIndex == 1) {
d = Double.parseDouble((String) value);
dataSheet.getDataItem(rowIndex).setX(d);
} else if (columnIndex == 2) {
d = Double.parseDouble((String) value);
dataSheet.getDataItem(rowIndex).setY(d);
}
}
fireDataSheetChange();
} catch (Exception ex) {}
}
```

Після цього можна повернутися до класу нашого компонента DataSheetTable і створити оброблювачі натискання кнопок addButton та delButton. У цих оброблювачах слід вказати дії, які необхідно виконати для додавання та видалення рядка в таблиці (і, відповідно, у «сховищі даних»), а також оповістити зацікавлені компоненти у змінах, які виникли. Код оброблювачів може бути приблизно таким:

```
addButton.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
tableModel.setRowCount(tableModel.getRowCount()+1);
tableModel.getDataSheet().addDataItem(new Data());
table.revalidate();
tableModel.fireDataSheetChange();
}
});
... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
delButton.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
if (tableModel.getRowCount() > 1) {
tableModel.setRowCount(tableModel.getRowCount() - 1);
tableModel.getDataSheet().removeDataItem(
tableModel.getDataSheet().size()-1);
table.revalidate();
tableModel.fireDataSheetChange();
```

```
} else {
tableModel.getDataSheet().getDataItem(0).setDate("");
tableModel.getDataSheet().getDataItem(0).setX(0);
tableModel.getDataSheet().getDataItem(0).setY(0);
table.revalidate();
table.repaint();
tableModel.fireDataSheetChange();
}
}
});
```

Наш перший компонент практично готовий.

Другий компонент JavaBean

У цьому пакеті створимо другий компонент, призначений для графічного виводу
інформації про точки. Для цього створимо клас DataSheetGraph, похідний від класу JPanel.

У цьому класі створимо поле, яке контролює процес серіалізації:

```
private static final long serialVersionUID = 1L;
```

Крім того, слід вказати деякі властивості:

```
private DataSheet dataSheet = null;
private boolean isConnected;
private int deltaX;
private int deltaY;
transient private Color color;
```

з відповідними методами доступу (гетери та сетери, згідно зі специфікацією JavaBeans).

```
public DataSheet getDataSheet() {
return dataSheet;
}
public void setDataSheet(DataSheet dataSheet) {
this.dataSheet = dataSheet;
}
public boolean isConnected() {
return isConnected;
}
public void setConnected(boolean isConnected) {
this.isConnected = isConnected;
repaint();
}
public int getDeltaX() {
return deltaX;
}
public int getDeltaY() {
return deltaY;
}
```

```
public void setDeltaX(int deltaX) {
this.deltaX = deltaX;
repaint();
}
public void setDeltaY(int deltaY) {
this.deltaY = deltaY;
repaint();
}
public Color getColor() {
return color;
}
public void setColor(Color color) {
this.color = color;
repaint();
}
```

Крім того, створимо конструктор і метод ініціалізації властивостей:

```
public DataSheetGraph() {
super();
initialize();
}
private void initialize() {
isConnected = false;
deltaX = 5;
deltaY = 5;
color = Color.red;
this.setSize(300, 400);
}
```

Для зручності роботи слід визначити методи, що обчислюють максимальне і
мінімальне значення величин X та Y для даних, що знаходяться у сховищі. Наприклад, так:

```
private double minX() {
// Вісь повинна відображатись
double result = 0;
// Визначаємо мінімальне значення X
if (dataSheet != null) {
int size = dataSheet.size();
for (int i = 0; i < size; i++)
if (dataSheet.getDataItem(i).getX() < result)
result = dataSheet.getDataItem(i).getX();
}
return result;
}
private double maxX() {
... ... ... ... ... ... ... ... ... ... ... ...
```

```
}
private double minY() {
... ... ... ... ... ... ... ... ... ... ... ...
}
private double maxY() {
... ... ... ... ... ... ... ... ... ... ... ...
}
```
Далі слід перевизначити метод paintComponent класу JComponent:
```
@Override
protected void paintComponent(Graphics g) {
super.paintComponent(g);
Graphics2D g2 = (Graphics2D) g;
showGraph(g2);
}
```
Тепер слід створити метод showGraph, який власне й здійснює малювання графіка.
Схематично покажемо можливий код методу:
```
public void showGraph(Graphics2D gr) {
double xMin, xMax, yMin, yMax;
double width = getWidth();
double height = getHeight();
xMin = minX() - deltaX;
xMax = maxX() + deltaX;
yMin = minY() - deltaY;
yMax = maxY() + deltaY;
// Визначаємо коефіцієнти перетворення та положення початку координат
double xScale = width / (xMax - xMin);
double yScale = height / (yMax - yMin);
double x0 = -xMin*xScale;
double y0 = yMax*xScale;
// Заповнюємо область графіка білим кольором
Paint oldColor = gr.getPaint();
gr.setPaint(Color.WHITE);
gr.fill(new Rectangle2D.Double(0.0, 0.0, width, height));
Stroke oldStroke = gr.getStroke();
Font oldFont = gr.getFont();
// Створюємо лінії сітки
// Сітка для вісі X
loat[] dashPattern = {10, 10};
gr.setStroke(new BasicStroke(1.0f, BasicStroke.CAP_BUTT,
BasicStroke.JOIN_MITER, 10.0f, dashPattern, 0));
gr.setFont(new Font("Serif", Font.BOLD, 14));
// Взагалі слід створити метод для обчислення кроку сітки
double xStep = 1;
for (double dx = xStep; dx < xMax; dx += xStep) {
```

```
double x = x0 + dx*xScale;
gr.setPaint(Color.LIGHT_GRAY);
gr.draw(new Line2D.Double(x, 0, x, height));
gr.setPaint(Color.BLACK);
gr.drawString(Math.round(dx/xStep)*xStep+"", (int)x+2, 10);
}
for (double dx = -xStep; dx >= xMin; dx -= xStep) {
double x = x0 + dx*xScale;
gr.setPaint(Color.LIGHT_GRAY);
gr.draw(new Line2D.Double(x, 0, x, height));
gr.setPaint(Color.BLACK);
gr.drawString(Math.round(dx/xStep)*xStep+"", (int)x+2, 10);
}
// Сітка для вісі Y
// Взагалі слід створити метод для обчислення кроку сітки
double yStep = 1;
for (double dy = yStep; dy < yMax; dy += yStep) {
double y = y0 - dy*yScale;
gr.setPaint(Color.LIGHT_GRAY);
gr.draw(new Line2D.Double(0, y, width, y));
gr.setPaint(Color.BLACK);
gr.drawString(Math.round(dy/yStep)*yStep+"", 2, (int)y-2);
}
for (double dy = -yStep; dy >= yMin; dy -= yStep) {
double y = y0 - dy*yScale;
gr.setPaint(Color.LIGHT_GRAY);
gr.draw(new Line2D.Double(0, y, width, y));
gr.setPaint(Color.BLACK);
gr.drawString(Math.round(dy/yStep)*yStep+"", 2, (int)y-2);
}
// Вісі координат
gr.setPaint(Color.BLACK);
gr.setStroke(new BasicStroke(3.0f));
gr.draw(new Line2D.Double(x0, 0, x0, height));
gr.draw(new Line2D.Double(0, y0, width, y0));
gr.drawString("X", (int)width-10, (int)y0-2);
gr.drawString("Y", (int)x0+2, 10);
// Відображаємо точки, якщо визначено сховище
if (dataSheet != null) {
if (!isConnected) {
for (int i = 0; i < dataSheet.size(); i++) {
double x = x0 + (dataSheet.getDataItem(i).getX() * xScale);
double y = y0 - (dataSheet.getDataItem(i).getY() * yScale);
gr.setColor(Color.white);
gr.fillOval((int) (x - 5 / 2), (int) (y - 5 / 2), 5, 5);
```

```
gr.setColor(color);
gr.drawOval((int) (x - 5 / 2), (int) (y - 5 / 2), 5, 5);
}
} else {
gr.setPaint(color);
gr.setStroke(new BasicStroke(2.0f));
double xOld = x0 + dataSheet.getDataItem(0).getX() * xScale;
double yOld = y0 - dataSheet.getDataItem(0).getY() * yScale;
for (int i = 1; i < dataSheet.size(); i++) {
double x = x0 + dataSheet.getDataItem(i).getX() * xScale;
double y = y0 - dataSheet.getDataItem(i).getY() * yScale;
gr.draw(new Line2D.Double(xOld, yOld, (double) x, y));
xOld = x;
yOld = y;
}
}
// Відновляємо вихідні значення
gr.setPaint(oldColor);
gr.setStroke(oldStroke);
gr.setFont(oldFont);
}
```

Для того, щоб на панелі інструментів не відображались «зайві» властивості, створимо клас DataSheetGraphBeanInfo, який описує наш компонент JavaBean. Цей клас повинен бути нащадком класу SimpleBeanInfo і в ньому слід описати все те, що повинно відображатись на палітрі інструментів у середовищі розробки. У нашому випадку, просто обмежимо властивості відображення:

```
public class DataSheetGraphBeanInfo extends SimpleBeanInfo {
private PropertyDescriptor[] propertyDescriptors;
public DataSheetGraphBeanInfo() {
try {
propertyDescriptors = new PropertyDescriptor[]
{
new PropertyDescriptor("color",DataSheetGraph.class),
new PropertyDescriptor("filled",DataSheetGraph.class),
new PropertyDescriptor("deltaX",DataSheetGraph.class),
new PropertyDescriptor("deltaY",DataSheetGraph.class)
};
} catch (IntrospectionException e) {}
}
@Override
public PropertyDescriptor[] getPropertyDescriptors() {
```

return propertyDescriptors;
}
}
Оскільки наші властивості відносяться до достатньо простих типів, середовище
розробки автоматично надає їм відповідні редактори властивостей. Тому, спеціально
створювати їх ми не будемо.
Таким чином, були створені два необхідних компонента JavaBean. У NetBeans їх
можна просто помістити на панель інструментів, а у Eclipse ними можна скористатись за
допомогою кнопки Choose Bean на панелі компонентів візуального редактора. Також їх
можна експортувати в jar–файл компонентів (див. документацію).
Інші класи додатку
Після того, як компоненти створені, зробимо пакет xml, в якому розмістимо класи,
призначені для читання XML–документа SAX парсером зі створенням дерева об'єктів у
пам'яті (класи DataHandler, SAXRead), і клас, призначений для створення DOM–об'єкта по
структурі даних, і збережемо його в XML–файл (клас DataSheetToXML). Ці класи аналогічні
відповідним класам, розробленим на попередньому занятті.
Тепер можна переходити до створення головного вікна додатку. Створимо новий пакет
myApplication для класу Test, який є нащадком класу JFrame з автоматично згенерованим
методом main. Змінимо заголовок фрейму і вкажемо, що він не повинен змінювати свої
розміри:
setResizable(false);
Об'явимо поле
private DataSheetTable dataSheetTable = null;
Слід перевірити, що менеджером компонування даного компонента є менеджер
BorderLayout. У південній області компонента (BorderLayout.SOUTH) слід розмістити панель
JPanel panel, яка є контейнером для кнопок управління додатком.

Структура і розташування компонентів наведені на рисунку. У східну область
(BorderLayout.EAST) слід додати компонент для малювання графіка (DataSheetGraph

dataSheetGraph), а в західну (BorderLayout.WEST) — компонент для роботи з таблицею
(DataSheetTable dataSheetTable). При налаштуванні компонентів слід вказати, що вони
повинні бути не локальними змінними, а полями класу. Для зручності цим двом компонентам
можна встановити бажані розміри (setPreferredSize(new Dimension(200, 300)), і
відповідним чином змінити розміри фрейму. На самому початку конструктора фрейму слід
створити «сховище даних» з одним рядком:

```
dataSheet = new DataSheet();
dataSheet.addDataItem(new Data());
```

Далі, у конструкторі фрейму, після створення компонента dataSheetGraph, але перед
його підключенням до фрейму, слід встановити йому створене «сховище».

```
dataSheetGraph.setDataSheet(dataSheet);
```

Далі, після створення компонента dataSheetTable, але перед його підключенням до
фрейму, слід також встановити йому створене «сховище» і додати слухача подій:

```
dataSheetTable.getTableModel().setDataSheet(dataSheet);
dataSheetTable.getTableModel().addDataSheetChangeListener(
new DataSheetChangeListener() {
public void dataChanged(DataSheetChangeEvent e) {
dataSheetGraph.revalidate();
dataSheetGraph.repaint();
}
});
```

Для забезпечення можливості вибору файлу для читання або запису необхідно додати
компонент JfileChooser. Це робиться за допомогою функції Choose Bean панелі
компонентів. Слід відмітити, що компонент додається не в середину фрейму, а поряд із ним.
У результаті буде створено та ініціалізовано поле:

```
private final JFileChooser fileChooser = new JFileChooser();
```

Для того, щоб файли обирались з поточної папки, у конструкторі фрейму слід додати
код:

```
fileChooser.setCurrentDirectory(new java.io.File("."));
```

Тепер залишилось створити оброблювачі подій для кнопок та змінити їх автоматично
згенерований код.
Для кнопки завершення роботи додатку:

```
exitButton.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent arg0) {
dispose();
}
});
```
Для кнопки очистки таблиці з даними оброблювач може бути таким:
```
clearButton.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
dataSheet = new DataSheet();
dataSheet.addDataItem(new Data());
dataSheetTable.getTableModel().setDataSheet(dataSheet);
dataSheetTable.revalidate();
dataSheetGraph.setDataSheet(dataSheet);
}
});
```
Для кнопки збереження таблиці з даними код оброблювача може мати такий вигляд:
```
saveButton.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
if (JFileChooser.APPROVE_OPTION == fileChooser.showSaveDialog(null)) {
String fileName = fileChooser.getSelectedFile().getPath();
DataSheetToXML.saveXMLDoc(
DataSheetToXML.createDataSheetDOM(dataSheet), fileName);
JOptionPane.showMessageDialog(null,
"File " + fileName.trim() + " saved!", "Результати збережені",
JOptionPane.INFORMATION_MESSAGE);
}
}
});
```
Для кнопки відкриття файлу з даними код оброблювача може мати такий вигляд:
```
readButton.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
if (JFileChooser.APPROVE_OPTION == fileChooser.showOpenDialog(null)) {
String fileName = fileChooser.getSelectedFile().getPath();
dataSheet = SAXRead.XMLReadData(fileName);
dataSheetTable.getTableModel().setDataSheet(dataSheet);
dataSheetTable.revalidate();
dataSheetGraph.setDataSheet(dataSheet);
}
}
});
```
Можна перевірити роботу додатку і експортувати його у jar–файл, що запускається.

Такий файл можна створити за допомогою засобів інтегрованого середовища розробки. Всі
середовища роблять це достатньо адекватно.
Можна зробити це вручну — алгоритм є аналогічним тому, як компонент упаковується
у jar–файл (див. лекцію), тільки у файлі маніфесту замість заголовків Name і Java-Beans
необхідно вказати заголовок Main-Class: mypackage.StartClass
Manifest-Version: 1.0
Main-Class: mypackage.StartClass