Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут”
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Курсовий проект

з дисципліни

"Інженерія програмного забезпечення"

Виконав
[redacted]

Київ 2011 р.

1. ЗМІСТ:

2. ВСТУП

В курсовій роботі розробляється програма для побудови кругових діаграм з використанням пакету SWING. Необхідно було реалізувати в комфортному для користувача вигляді графічний інтерфейс, тобто з панелями меню або кнопками, для швидкої реалізації дій з боку користувача та зручності перегляду результатів візуалізації.

В роботі розглядаються принципи побудови графічного інтерфейсу користувача, основні елементи інтерфейсу, організація обробки подій, система виключень. Розроблено програмне забезпечення на мові Java в середовищі eclipse. Робота містить повну документацію для коду програми та інструкцію користувача.

3. ТЕХНІЧНЕ ЗАВДАННЯ

3.1. Загальне завдання

Необхідно побудувати програму візуалізації табличних даних у вигляді діаграми. В якості вхідних даних виступає таблиця, що знаходиться у CSV-файлі. Кожен з рядків таблиці є описом елементу діаграми – сектора з можливими додатковими атрибутами.

3.2. Функціональність

Програма візуалізації має містити наступну функціональність:

- можливість завантаження/збереження даних у табличний формі в форматі CSV;
- синтаксичний розбір і верифікація формату CSV з відбудовою внутрішньої моделі даних, в разі помилок - формування виключення (Exception);
- інтерфейс користувача (на основі компонентів бібліотеки SWING), який містить дві основні області - табличну і графічну, а також допоміжні компоненти - меню, панелі, кнопки, діалоги, тощо. Таблична область містить дані, що завантажені з CSV-файлу, а графічна – її інтерпретацію у вигляді діаграми згідно варіанту;
- можливість редагування даних (модифікація елемента, вставка/видалення елемента) в одній області з синхронною зміною іншої області.
- можливість збереження результату графічної інтерпретації - діаграми у вигляді файлу формату JPEG.

3.3. Вимоги до реалізації

- мова програмування Java з використанням бібліотеки SWING;
- заборонено використовувати спеціалізовані компоненти для побудови діаграм;

- інтерфейс користувача має забезпечувати доступ до всієї функціональності програми;
- проект має бути повністю задокументований за допомогою JavaDoc;
- проект має повністю відповідати правилам CheckStyle;
- обґрунтована насиченість інтерфейсу елементами позитивно впливає на оцінку.
- можливість роботи програми з більше ніж з однією серією даних (в одному чи різних CSV-файлах) позитивно впливає на оцінку

3.4. Варіант завдання

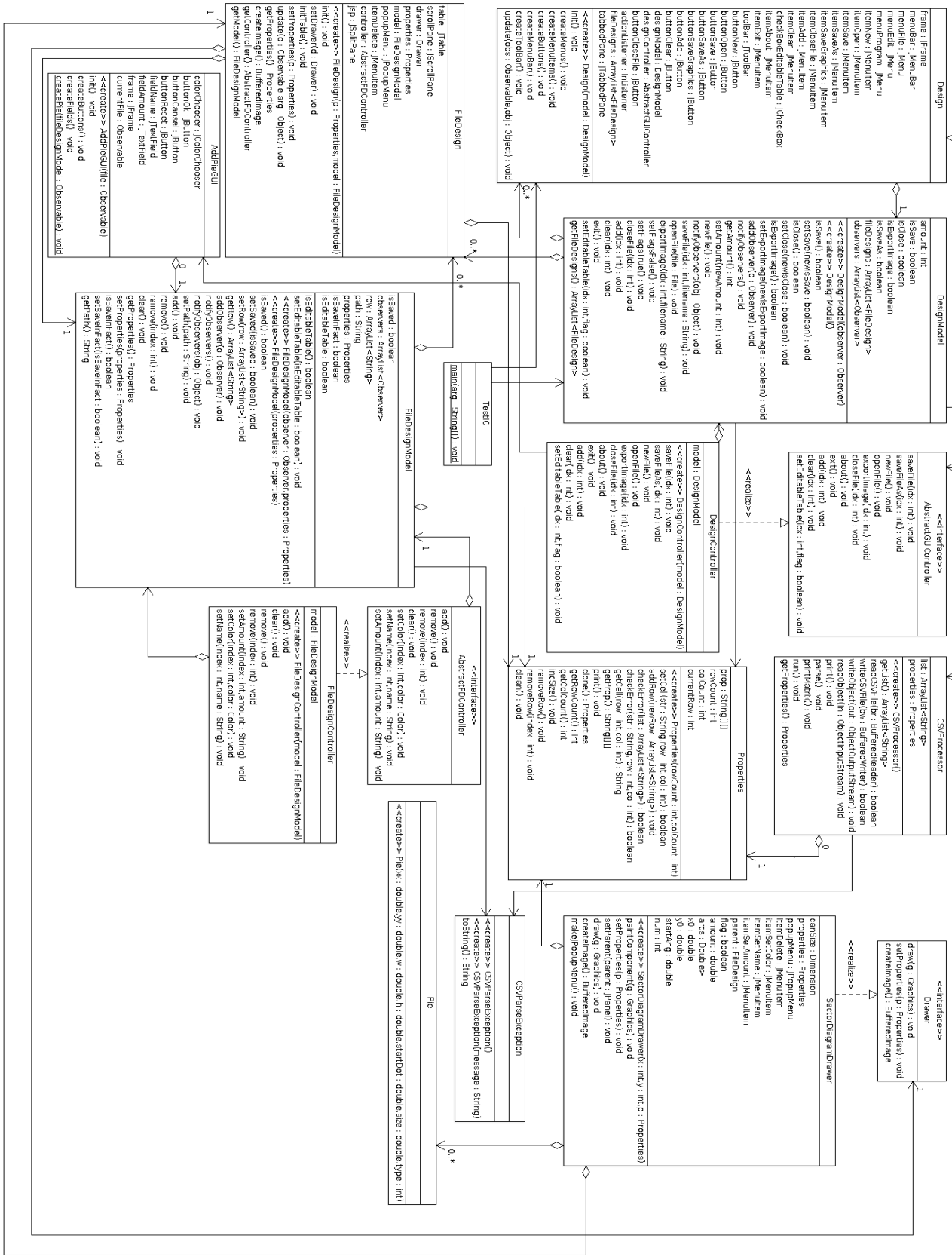
Назва діаграми – кругова.

Вхідні дані (елемент таблиці) — Значення (довжина серії дорівнює кількості секторів).

Вид редагування – редагування графічної області з синхронною модифікацією табличної області.

4. ПРОЕКТУВАННЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА

UML-діаграма класів



Початковий опис потоку подій роботи з програмою побудови діаграм:

1. Прецедент починається, коли користувач запускає програму.
2. Користувач може виконати одну з наступних дій:
   a) Створити файл.
   b) Зберегти файл.
   c) Зберегти файл в новому файлі.
   d) Відкрити збережений файл.
   e) Експортувати зображення діаграми.
   f) Додати сектор в діаграму.
   g) Видалити сектор з діаграми.
   h) Змінити колір сектора.
   i) Змінити назву сектора.
   j) Змінити розмір сектора.
   k) Закрити файл.
   l) Очистити діаграму.
4. Прецедент закінчується, коли користувач завершує роботу з програмою.

Розробка орієнтирів:

1. Прецедент починається, коли користувач запускає програму.
2. Користувач може виконати одну з наступних дій:
   a) Створити файл.
   b) Зберегти файл. [Користувач має зберігати файли в csv форматі]
   c) Зберегти файл в новому файлі. [Користувач має зберігати файли в csv форматі]
   d) Відкрити збережений файл. .[Користувач може відкривати один файл кільки разів]
   e) Експортувати зображення діаграми. [Користувач має експортувати зображення в форматі JPEG]
   f) Додати сектор в діаграму.

g) Видалити сектор з діаграми.

h) Змінити колір сектора.

i) Змінити назву сектора.

j) Змінити розмір сектора.

k) Закрити файл. [Користувач має отримати попередження, якщо файл не збережено]

l) Очистити діаграму.

4. Прецедент закінчується, коли користувач завершує роботу з програмою.

Розробка атрибутів:

1. Прецедент починається, коли користувач запускає програму.
2. Користувач може виконати одну з наступних дій:
   a) Створити файл.
   b) Зберегти файл. {Назва файлу в середньому 6 символів.}
   c) Зберегти файл в новому файлі. {Назва файлу в середньому 6 символів.}
   d) Відкрити збережений файл. {Назва файлу в середньому 6 символів.}
   e) Експортувати зображення діаграми. {Збереження у JPEG.}
   f) Додати сектор в діаграму. {Середній розмір сектора - 100}
   g) Видалити сектор з діаграми.
   h) Змінити колір сектора.
   i) Змінити назву сектора. {Середня довжина назви 6 символів.}
   j) Змінити розмір сектора. {Середній розмір сектора - 100}
   k) Закрити файл.
   l) Очистити діаграму.
4. Прецедент закінчується, коли користувач завершує роботу з програмою.

Розробка інтенсивності використання:

1. Прецедент починається, коли користувач запускає програму.
2. Користувач може виконати одну з наступних дій:
   a) Створити файл. (Використовується в 70% випадків)

b) Зберегти файл. (Використовується в 90% випадків)

c) Зберегти файл в новому файлі. (Використовується в 30% випадків)

d) Відкрити збережений файл. (Використовується в 30% випадків)

e) Експортувати зображення діаграми. (Використовується в 50% випадків)

f) Додати сектор в діаграму. (Використовується дуже часто)

g) Видалити сектор з діаграми. (Використовується часто)

h) Змінити колір сектора. (Використовується рідко)

i) Змінити назву сектора. (Використовується часто)

j) Змінити розмір сектора. (Використовується часто)

k) Закрити файл. (Використовується часто)

l) Очистити діараму. (Використовується рідко)

4. Прецедент закінчується, коли користувач завершує роботу з програмою.

Результуючий опис подій прикладу:

1. Прецедент починається, коли користувач запускає програму.
2. Користувач може виконати одну з наступних дій:
   a) Створити файл. (Використовується в 70% випадків)
   b) Зберегти файл. [Користувач має зберігати файли в csv форматі] {Назва файлу в середньому 6 символів.} (Використовується в 90% випадків)
   c) Зберегти файл в новому файлі. [Користувач має зберігати файли в csv форматі] {Назва файлу в середньому 6 символів.} (Використовується в 30% випадків)
   d) Відкрити збережений файл. .[Користувач може відкривати один файл кільки разів] {Назва файлу в середньому 6 символів.} (Використовується в 30% випадків)
   e) Експортувати зображення діаграми. [Користувач має експортувати зображення в форматі JPEG] {Назва має в середньому 6 символів } (Використовується в 50% випадків)
   f) Додати сектор в діаграму. {Середній розмір сектора - 100}

(Використовується дуже часто)

g) Видалити сектор з діаграми. (Використовується часто)

h) Змінити колір сектора. (Використовується рідко)

i) Змінити назву сектора. {Назва має в середньому 6 символів } (Використовується часто)

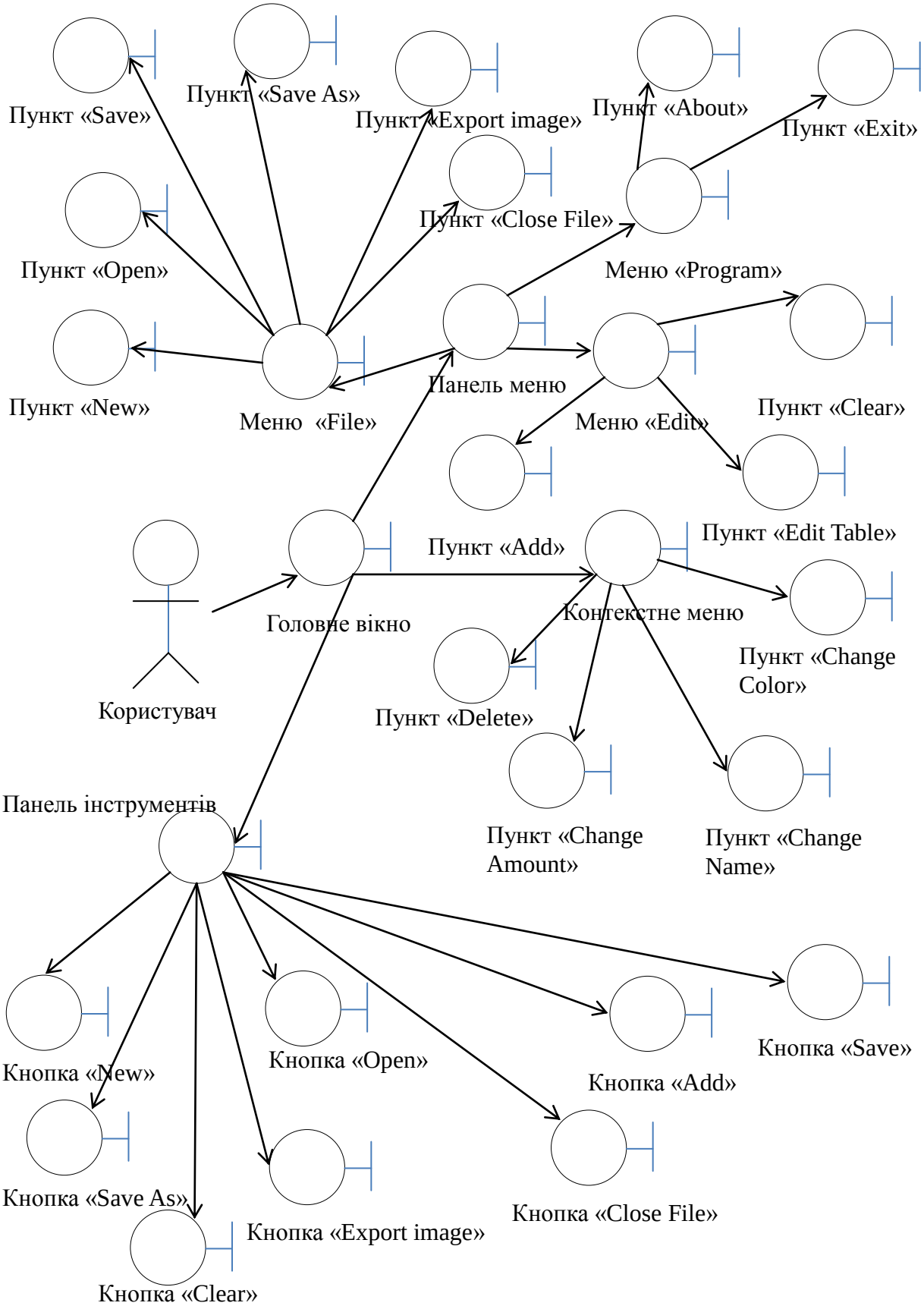
j) Змінити розмір сектора. {Середній розмір сектора - 100} (Використовується часто)

k) Закрити файл. [Користувач має отримати попередження, якщо файл не збережено] (Використовується часто)

l) Очистити діараму. (Використовується рідко)

4. Прецедент закінчується, коли користувач завершує роботу з програмою.

Діаграма граничних класів



Доповнення опису ілюстрованого сценарія об’єктами діаграм

1. Прецедент починається, коли користувач запускає програму.
2. Користувач може виконати одну з наступних дій:
   a) Створити файл. (Використовується в 70% випадків) <Пункт “New”> <Кнопка ”New”>
   b) Зберегти файл. [Користувач має зберігати файли в csv форматі] {Назва файлу в середньому 6 символів.} (Використовується в 90% випадків) <Пункт «Save»> <Кнопка «Save»>
   c) Зберегти файл в новому файлі. [Користувач має зберігати файли в csv форматі] {Назва файлу в середньому 6 символів.} (Використовується в 30% випадків) <Пункт «Save As»> <Кнопка «Save As»>
   d) Відкрити збережений файл. .[Користувач може відкривати один файл кільки разів] {Назва файлу в середньому 6 символів.} (Використовується в 30% випадків) <Пункт «Open»> <Кнопка «Open»>
   e) Експортувати зображення діаграми. [Користувач має експортувати зображення в форматі JPEG] {Назва має в середньому 6 символів } (Використовується в 50% випадків) <Пункт «Export image»> <Кнопка «Export image»>
   f) Додати сектор в діаграму. {Середній розмір сектора - 100} (Використовується дуже часто) <Пункт «Add»> <Кнопка «Add»>
   g) Видалити сектор з діаграми. (Використовується часто) <Пункт «Delete»>
   h) Змінити колір сектора. (Використовується рідко) <Пункт «Change Color»>
   i) Змінити назву сектора. {Назва має в середньому 6 символів } (Використовується часто) <Пункт «Change Name»>
   j) Змінити розмір сектора. {Середній розмір сектора - 100} (Використовується часто) <Пункт «Amount»>

k) Закрити файл. [Користувач має отримати попередження, якщо файл не збережено] (Використовується часто) <Пункт «Close File»> <Кнопка «Close File»>

l) Очистити діараму. (Використовується рідко) <Пункт «Clear»> <Кнопка «Clear»>

4. Прецедент закінчується, коли користувач завершує роботу з програмою.

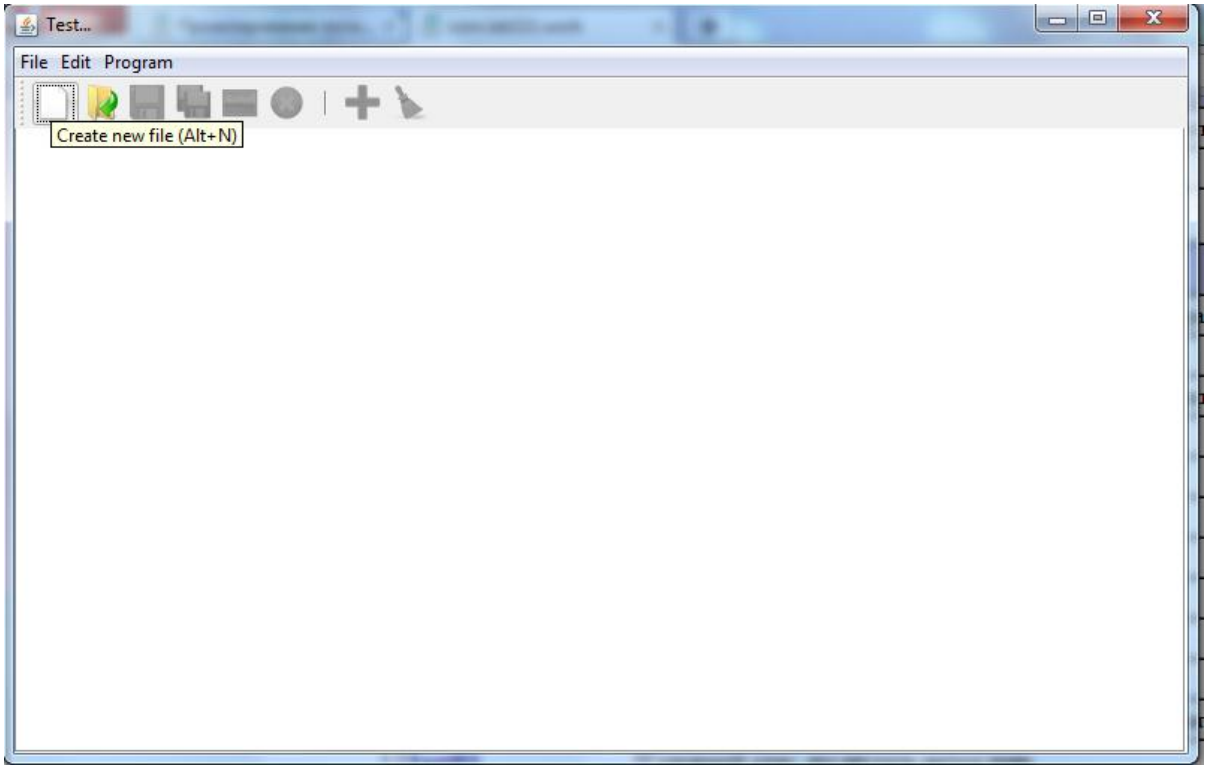
4. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Таблиця відповідності елементів дизайну

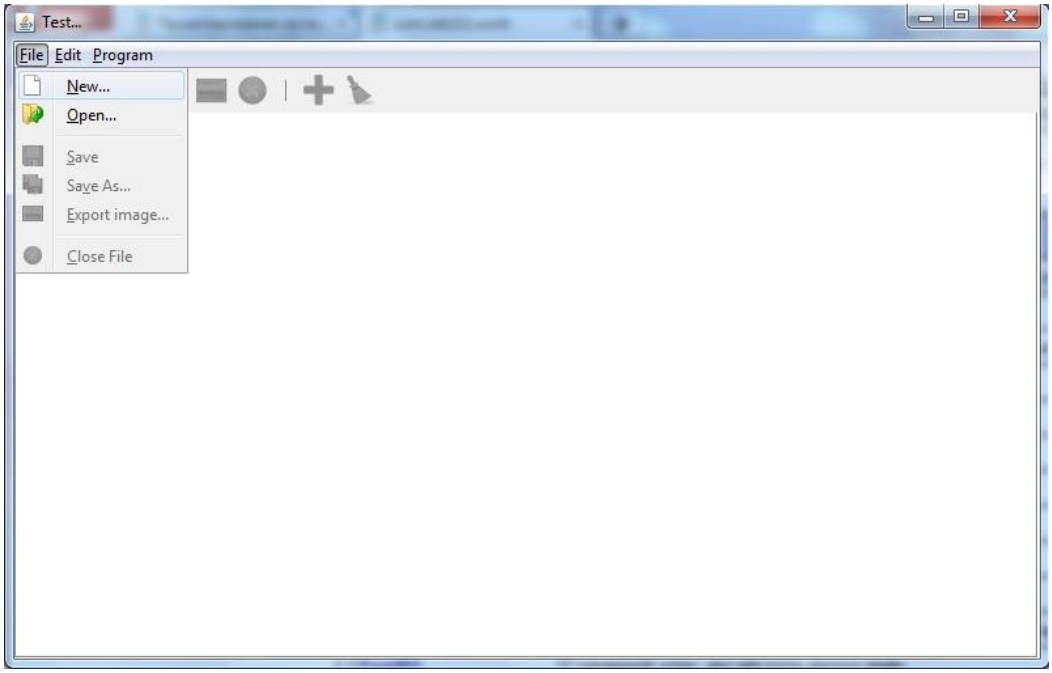
Пункт “New”	javax.swing.JMenuItem;
Пункт “Open”	javax.swing.JMenuItem;
Пункт “Save”	javax.swing.JMenuItem;
Пункт “Save As”	javax.swing.JMenuItem;
Пункт “Export image”	javax.swing.JMenuItem;
Пункт “Close File”	javax.swing.JMenuItem;
Кнопка “New”	javax.swing.JButton;
Кнопка “Open”	javax.swing.JButton;
Кнопка “Save”	javax.swing.JButton;
Кнопка “Save As”	javax.swing.JButton;
Кнопка “Export image”	javax.swing.JButton;
Кнопка “CloseFile”	javax.swing.JButton;
Панель Меню	javax.swing.JMenuBar;
Меню “File”	javax.swing.JMenu;
Меню “Edit”	javax.swing.JMenu;
Меню “Program”	javax.swing.JMenu;
Пункт “Edit Table”	javax.swing.JCheckBox
Пункт “Add”	javax.swing.JMenuItem;
Пункт “Clear”	javax.swing.JMenuItem;
Пункт “About”	javax.swing.JMenuItem;
Пункт “Exit”	javax.swing.JMenuItem;
Кнопка “Add”	javax.swing.JButton;
Кнопка “Clear”	javax.swing.JButton;
Вспливаюче меню	javax.swing.JPopupMenu;
“Delete”	javax.swing.JMenuItem;
Пункт “Change Color ”	javax.swing. JMenuItem;
Пункт “Change Name”	javax.swing. JMenuItem;
Пункт “Change Amount”	javax.swing. JMenuItem;
Вкладки	javax.swing.JTabbedPane;

Інструкція для роботи з програмою

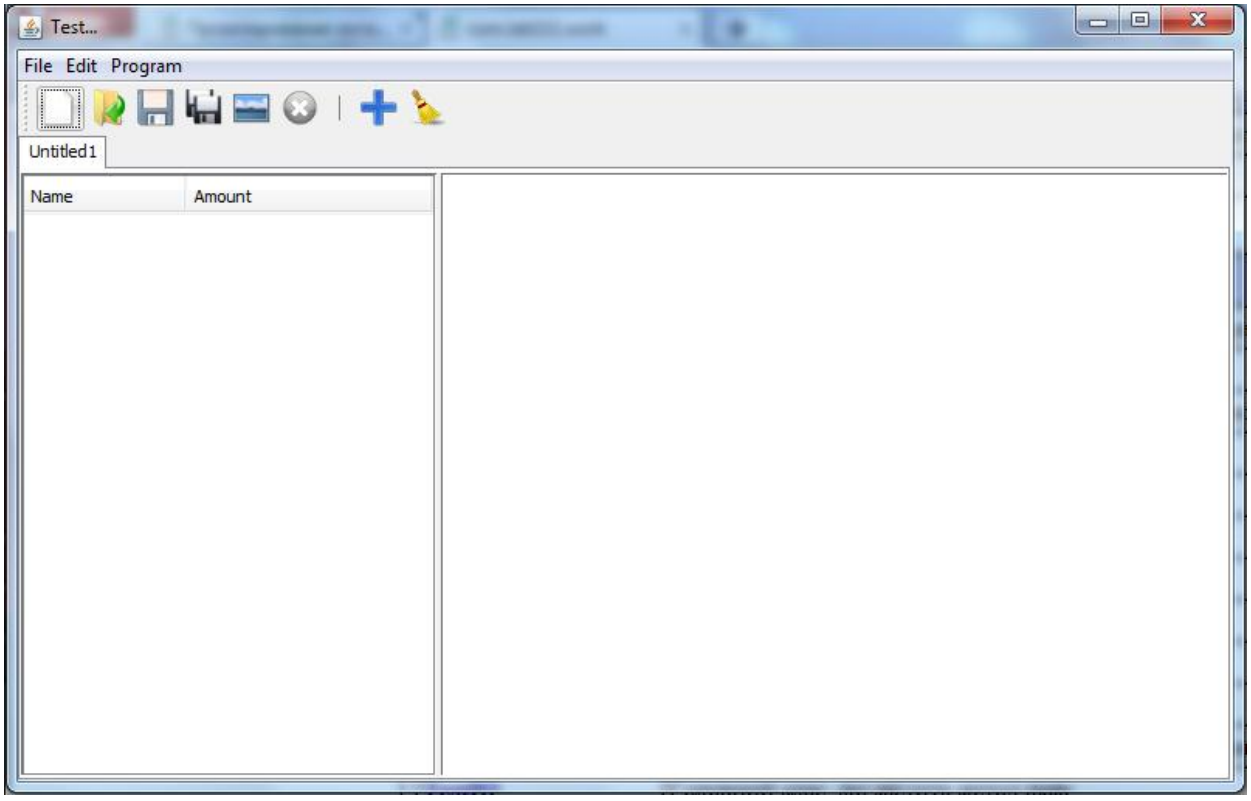
Створити файл можна натиснувши кнопку :



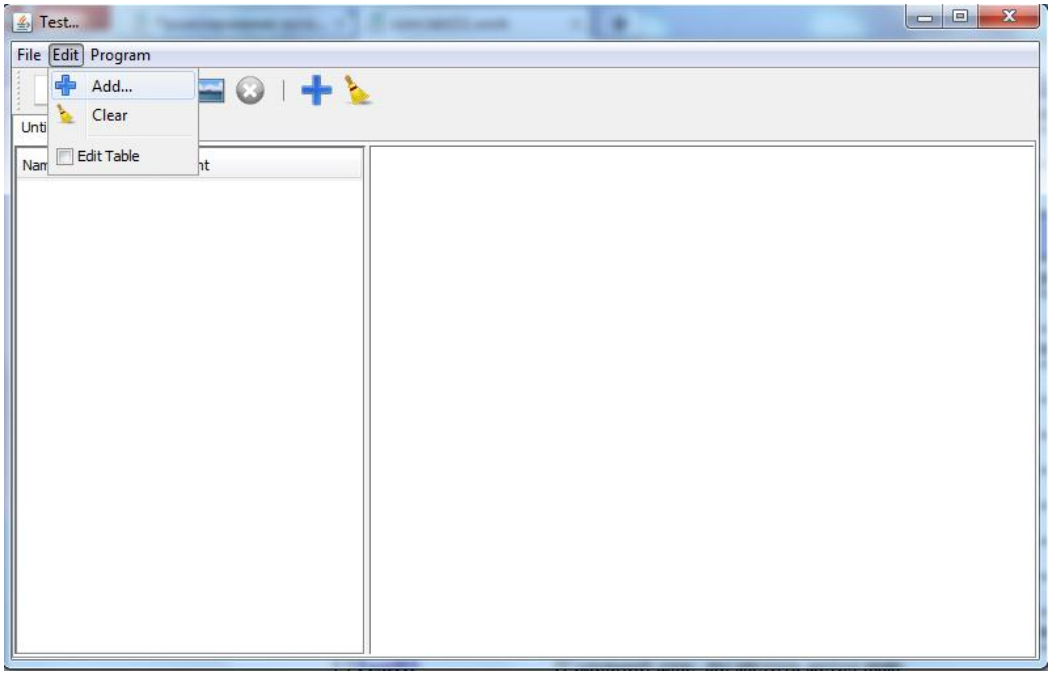
або натиснувши пункт меню

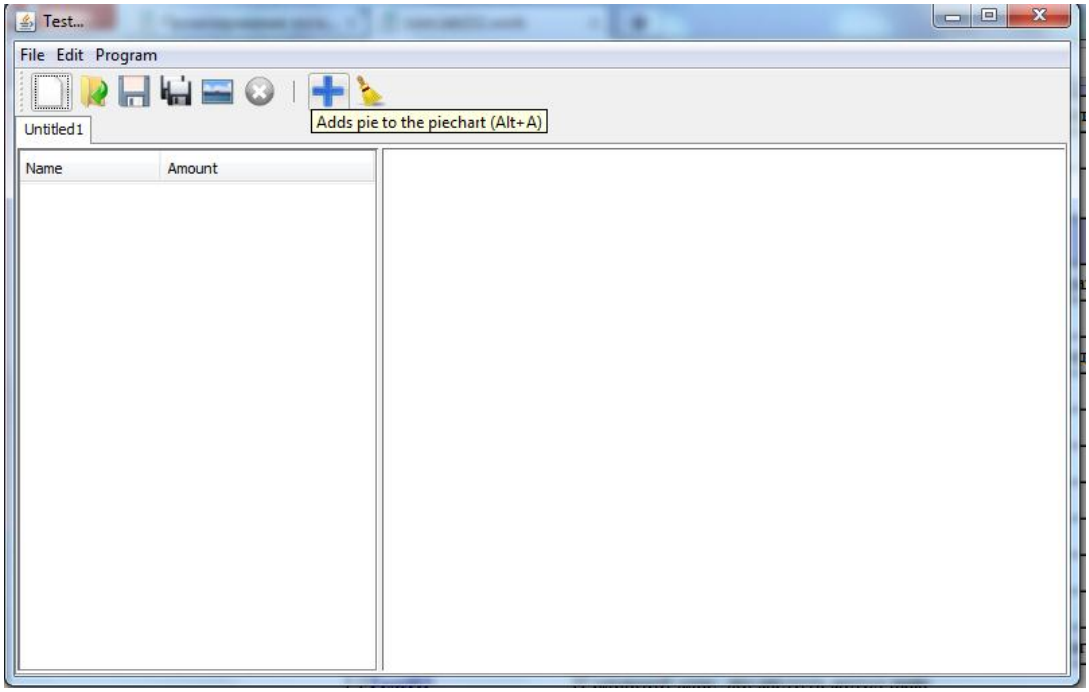


Створений файл має вигляд:

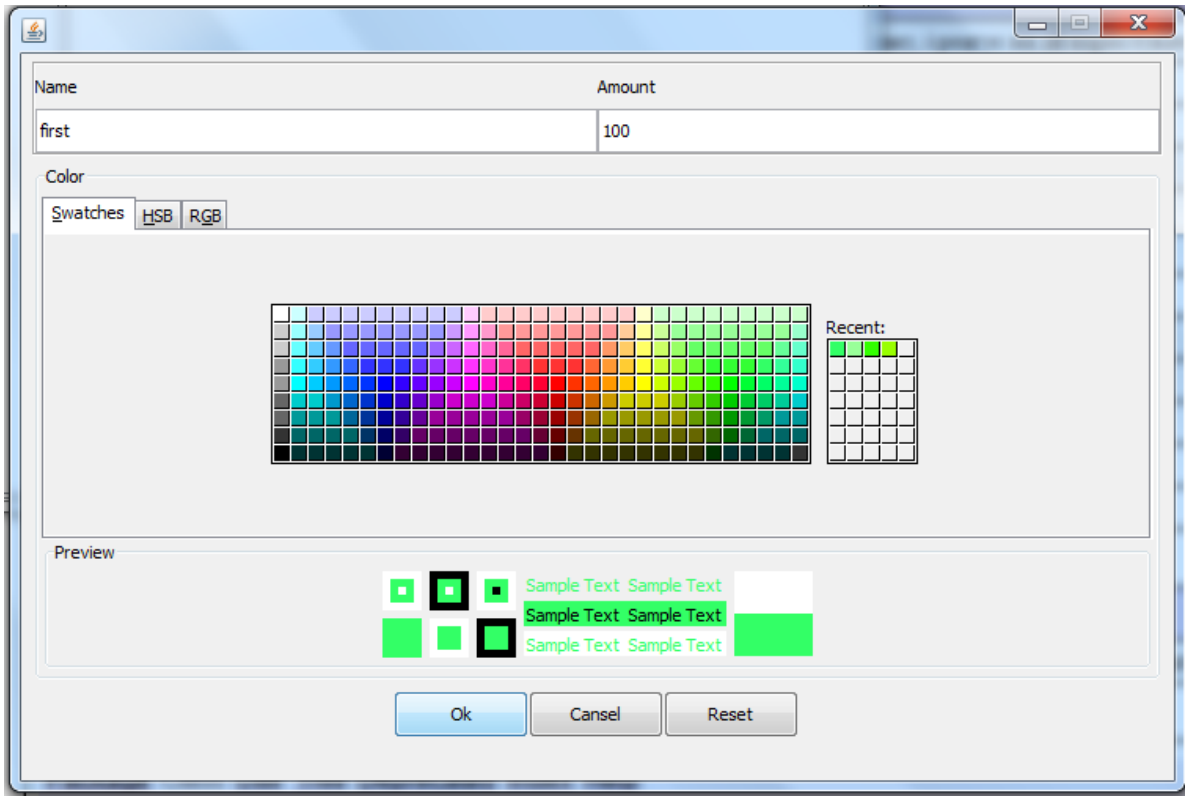


Додати сектор в діаграму можна натиснувши кнопку чи пункт меню :

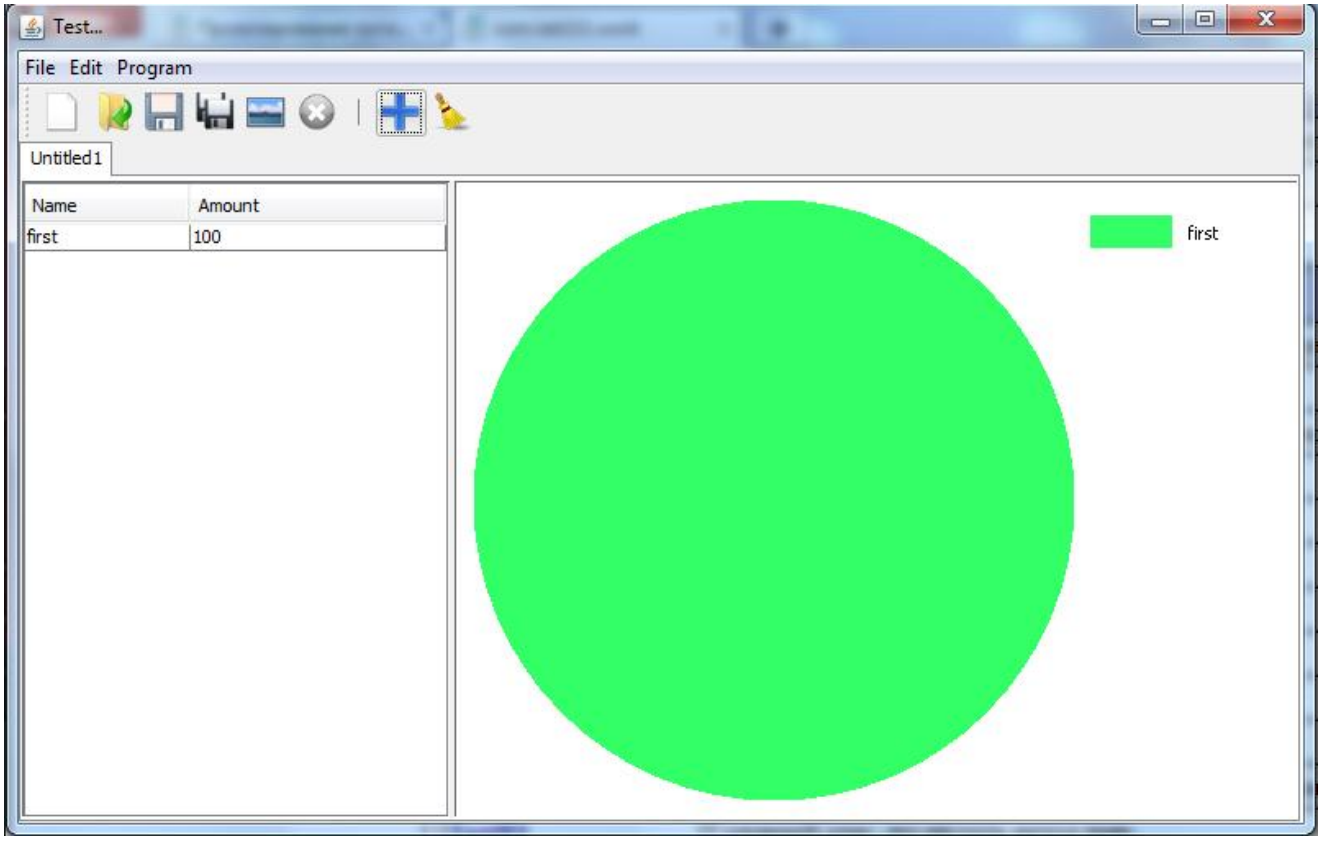




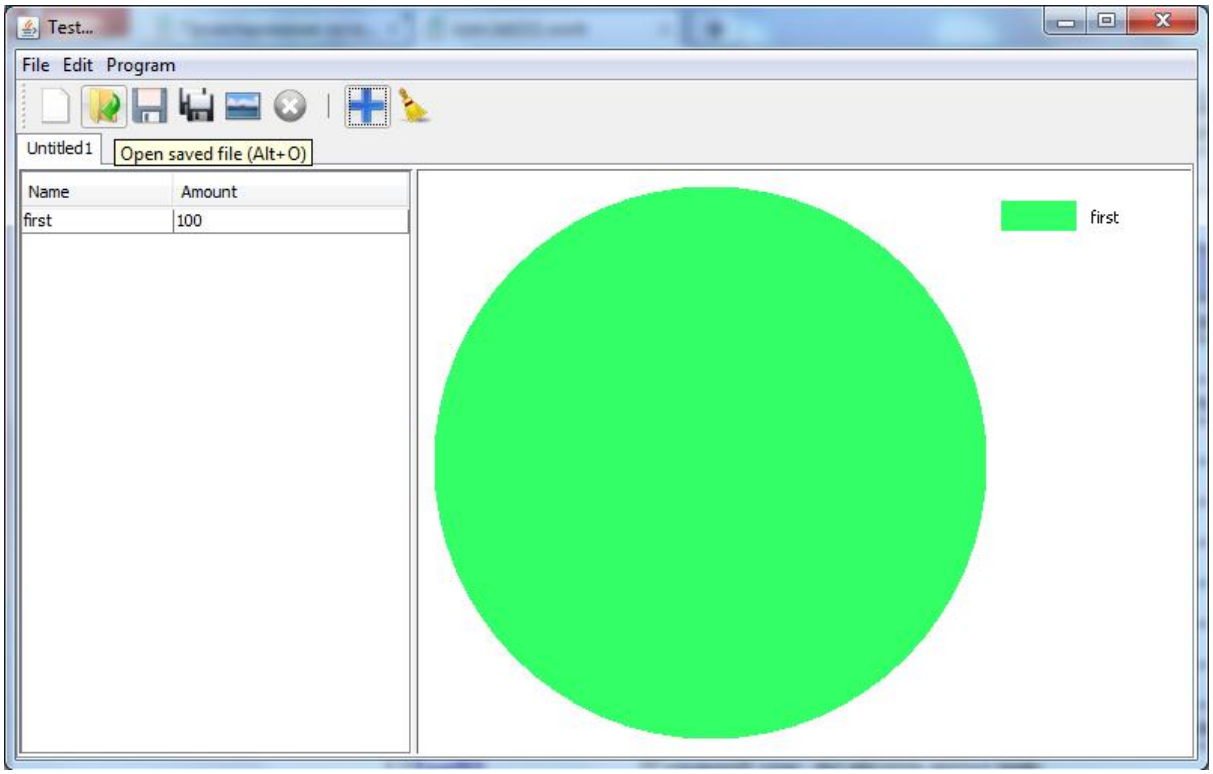
Після того з’являється вікно додавання сектору. Вводимо потрібні значення і тиснемо «Ok».

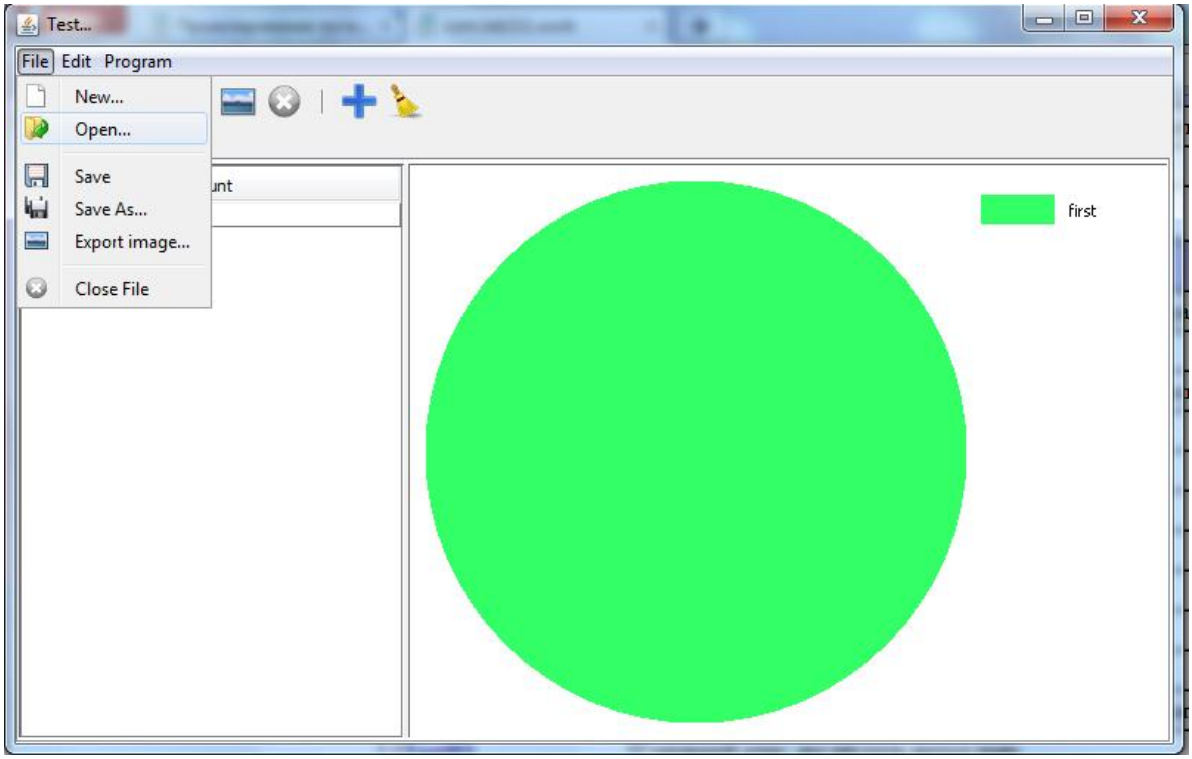


Після додавання даного сектору отримуємо :

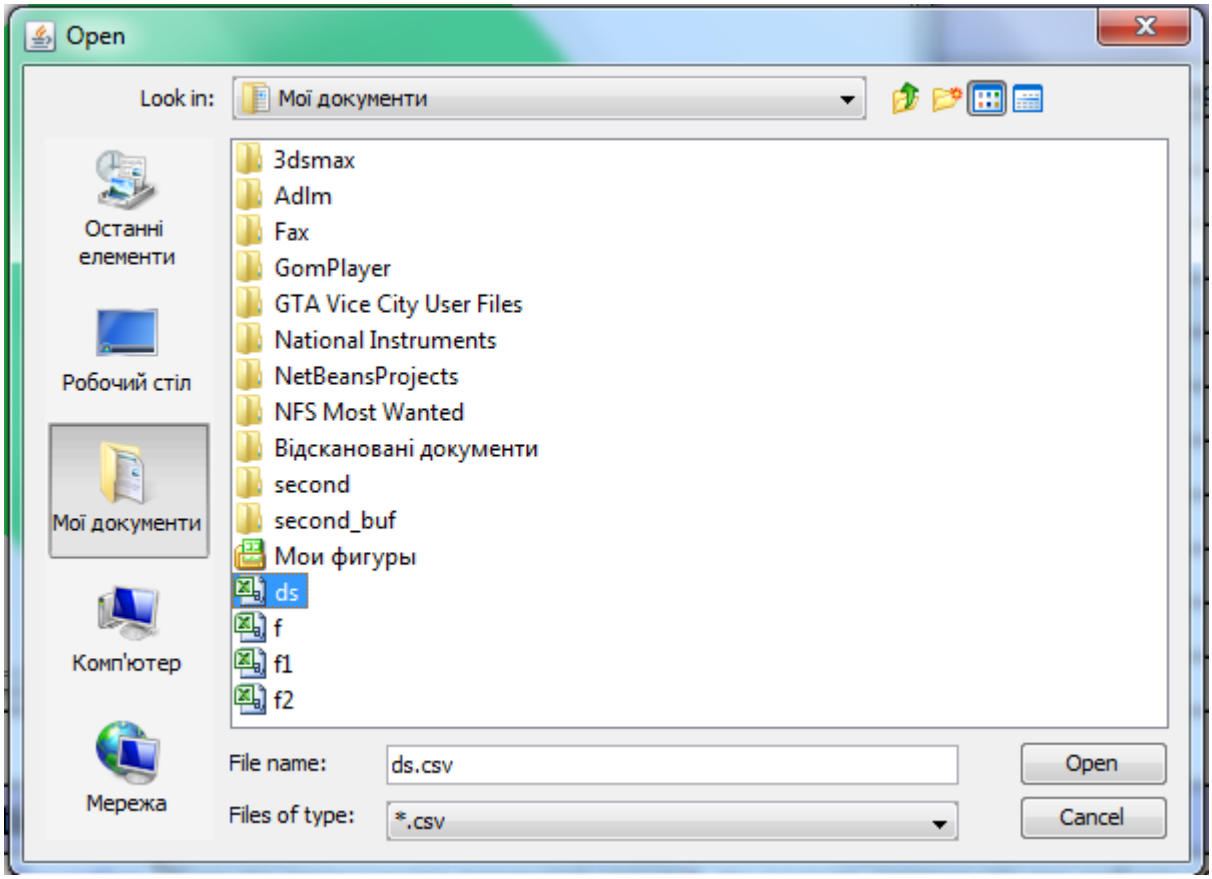


Відкрити файл можна натиснувши пункт меню чи кнопку:

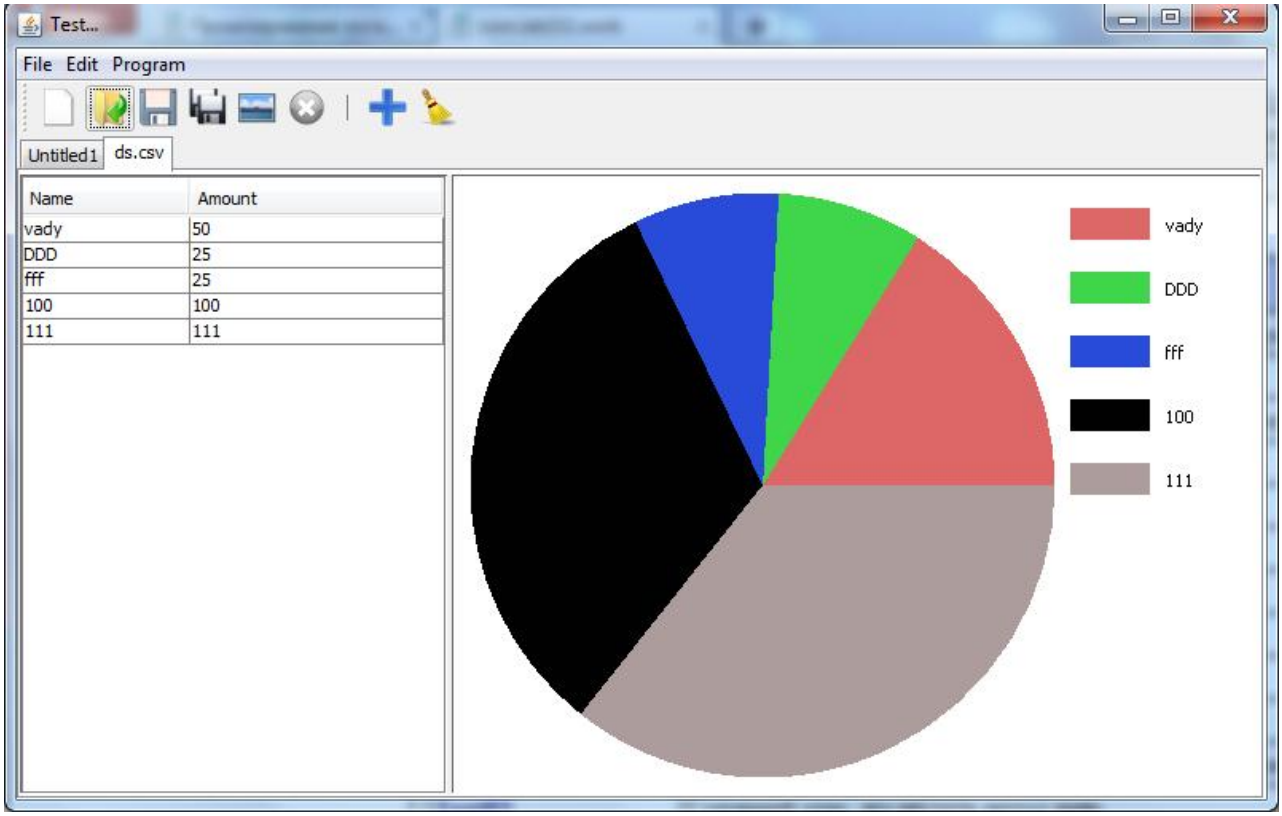




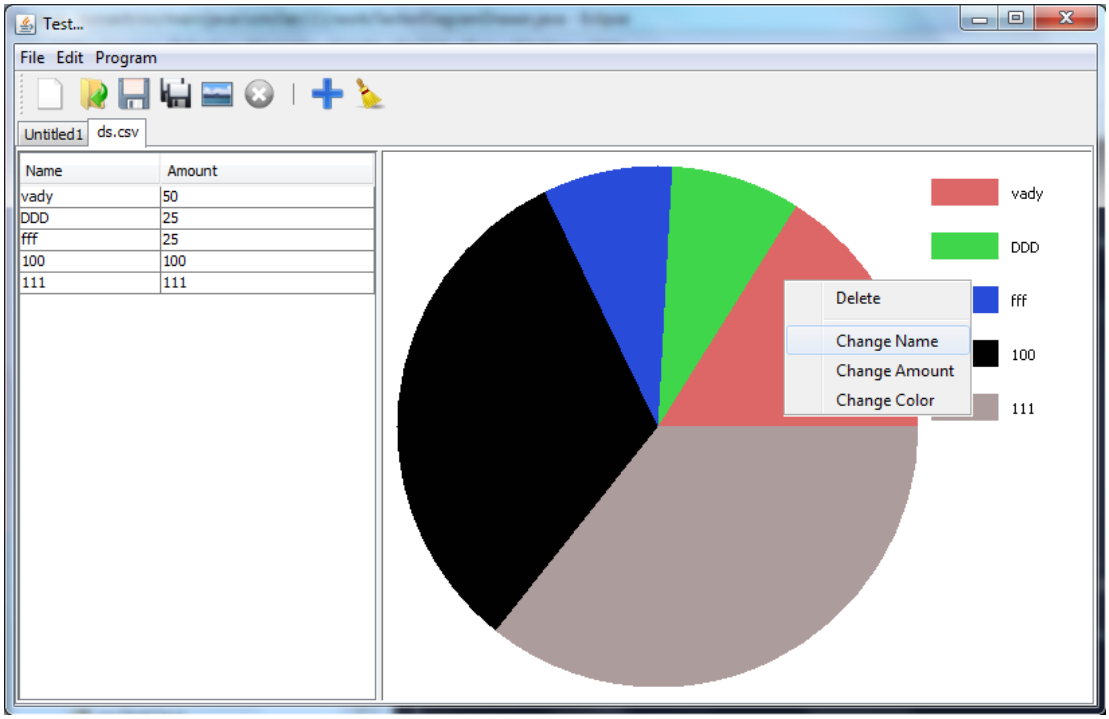
Відкриття файлу:



Файл відкрито :



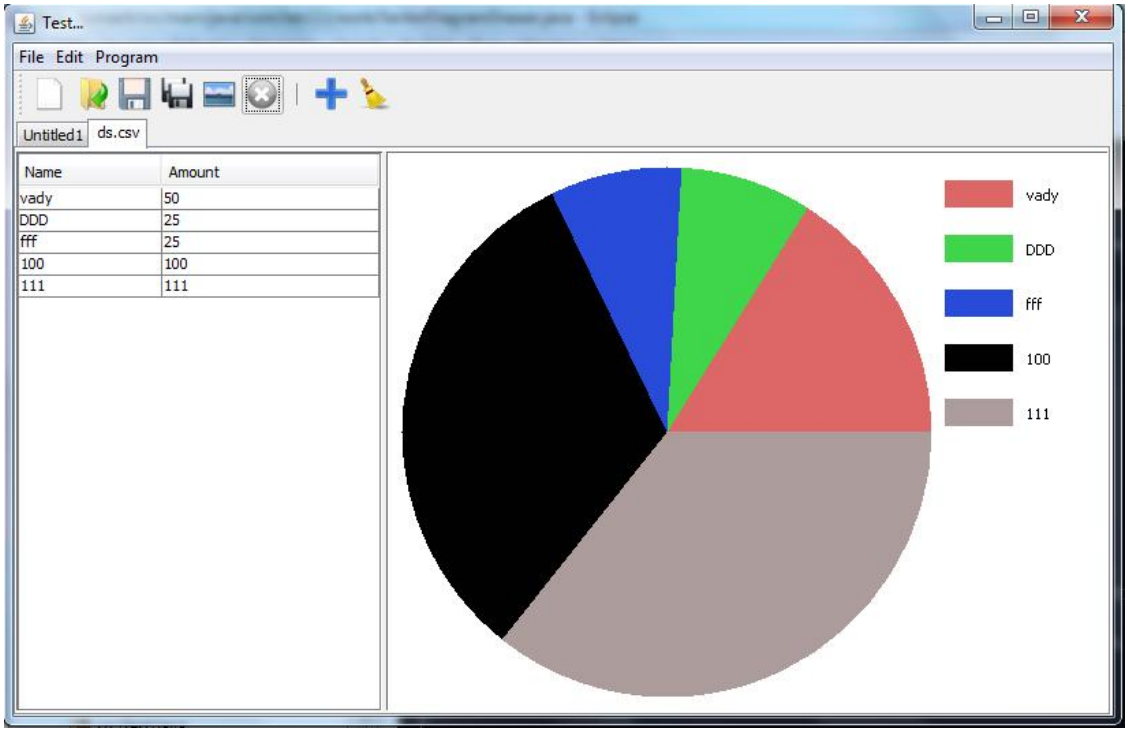
Натиснувши правою кнопкою миші на секторі отримаємо контекстне меню, що містить операції над сектором:



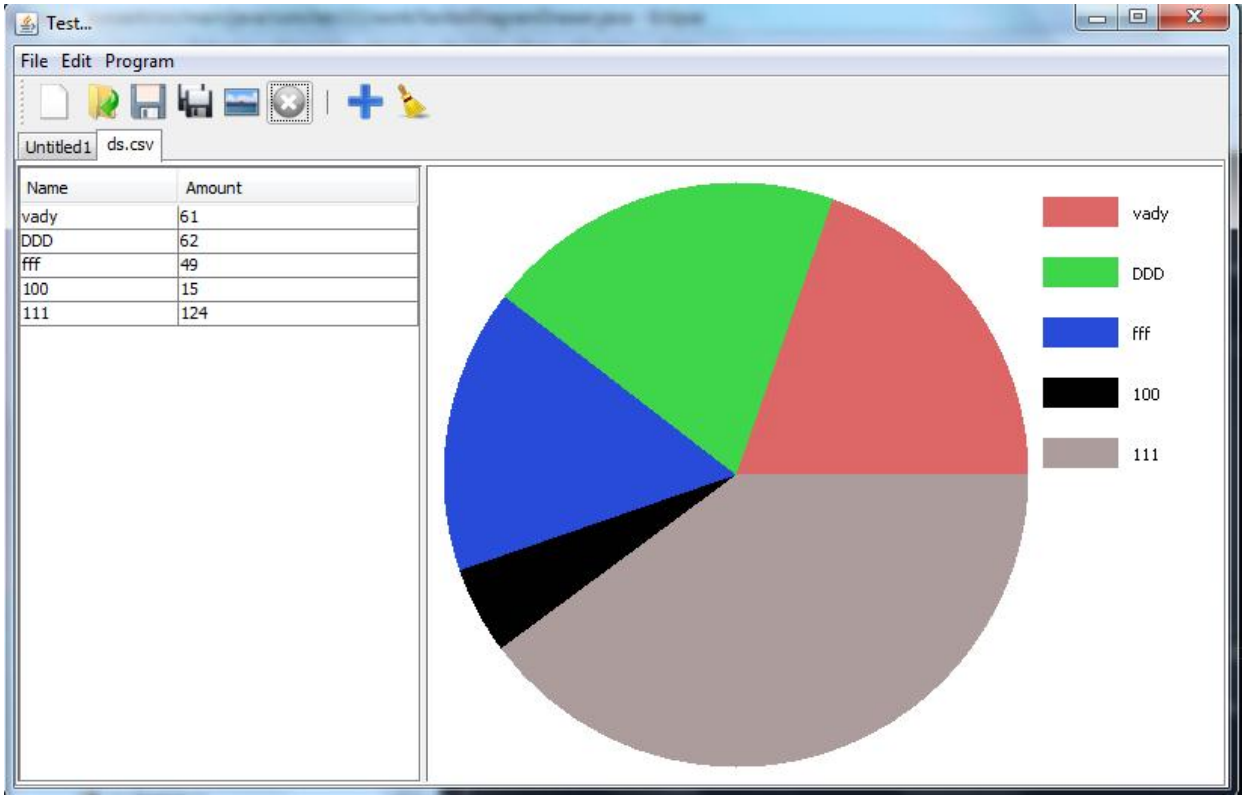
Можна відредагувати сектори з допомогою миші. Натиснувши і перетягуючи

мишу між сусідніми секторами змінюємо їх розмір.

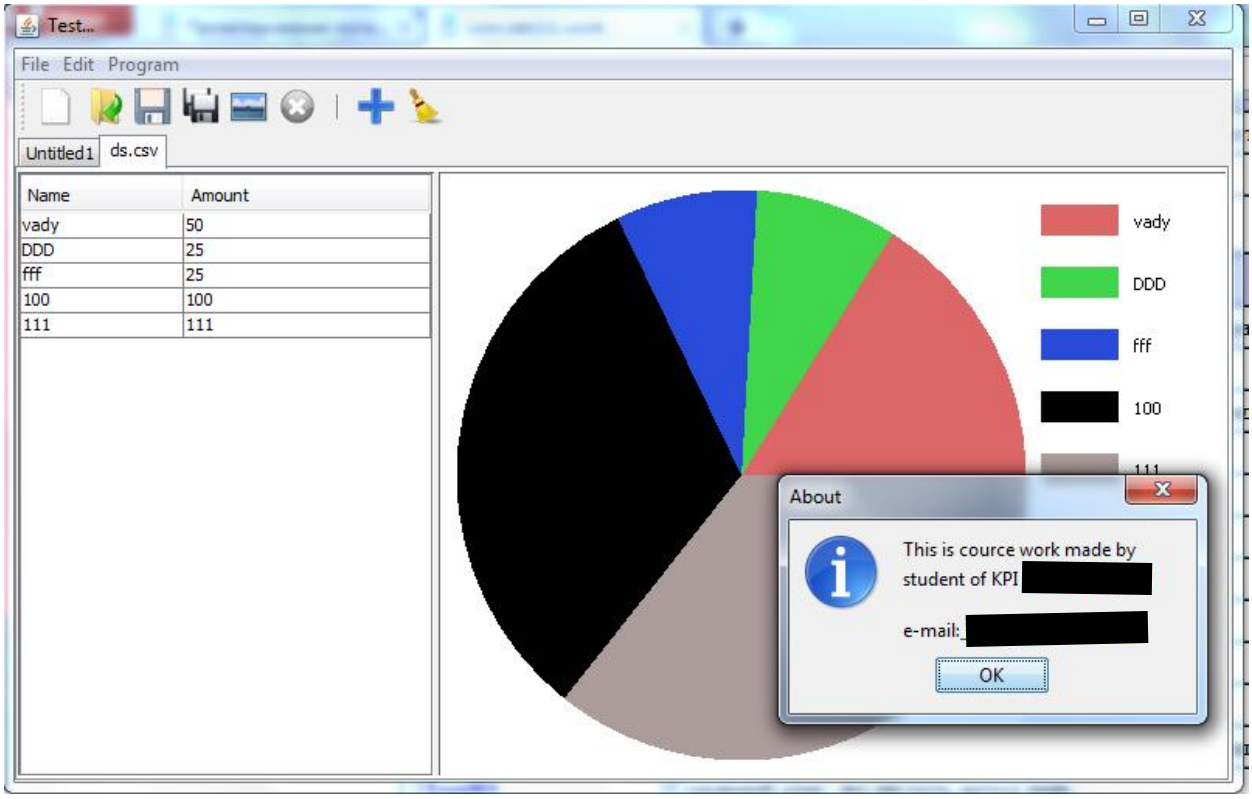
До перетягування:



Після перетягувань:



Пункт About :


Test...
File Edit Program
Untitled1
ds.csv
Name
Amount
vady
50
DDD
25
fff
25
100
100
111
111
vady
DDD
fff
100
About
This is cource work made by
student of KPI
e-mail:
OK

Специфікації класів

Package com.lab111.work

Interface Summary	
AbstractGUIController	Інтерфейс AbstractGUIController оголошує інтерфейс котролерів, що контролюють контролюють дії над класом `Design`.
Drawer	Інтерфейс для діаграм.

Class Summary	
AddPieGUI	Клас що визначає вигляд вікна додавання поля в діаграму, і реагує на дії користувача під час додавання поля.
CSVProcessor	Клас, що реалізує методи для роботи з CSV файлом.
Design	Клас, що відображає представлення програми, її вигляд.
DesignController	Клас - контролер для головного вікна.
DesignModel	Клас-модель для головного вікна.
FileDesign	Клас, що являє собою вигляд csv файлу.
FileDesignController	Клас контролер для класу FileDesign.
FileDesignModel	Клас, модель для класу FileDesign.
Pie	Клас що являє поле з секторної діаграми.
Properties	Клас, що репрезентує вміст csv файлу.
SectorDiagramDrawer	Метод, що являє полотно для малювання секторної діаграми.
TestIO	Головний клас, що містить метод main.

Exception Summary	
CSVParseException	Клас що репрезентує виключні ситуації, що можуть статися під час роботи з CSV файлом.

com.lab111.work
Interface AbstractGUIController

All Known Implementing Classes:

DesignController

public interface **AbstractGUIController**

Інтерфейс AbstractGUIController оголошує інтерфейс котролерів, що контролюють контролюють дії над класом `Design`.

Author:

rebelizant Гула Вадим

Method Summary	
void	**about**() Метод, що показує інформацію про програму та розробника.
void	**add**(int idx) Метод, що додає поле в діаграму поточного файлу.
void	**clear**(int idx) Метод, що видаляє всі поля з діаграми поточного файлу.
void	**closeFile**(int idx) Метод, що закриває поточний файл.
void	**exit**() Метод, що може призвести до виходу з програми.
void	**exportImage**(int idx) Метод для збереження зображення діаграми.
void	**newFile**() Метод створює новий файл.
void	**openFile**() Метод для відкриття збереженого файлу.
void	**saveFile**(int idx) Метод зберігає поточний файл в компоненті `JTabbedPane`.
void	**saveFileAs**(int idx) Метод, що зберігає поточний файл в новому файлі.
void	**setEditableTable**(int idx, boolean flag) Метод, що встановлює можливість редагування таблиці.

Method Detail

saveFile

void **saveFile**(int idx)

Метод зберігає поточний файл в компоненті `JTabbedPane`.

Parameters:

`idx` - - індекс файлу, що зберігається в компоненті `JTabbedPane`.

saveFileAs

void **saveFileAs**(int idx)

Метод, що зберігає поточний файл в новому файлі.

Parameters:

`idx` - - індекс поточного файлу в компоненті `JTabbedPane`.

newFile

void **newFile**()

Метод створює новий файл.

openFile

`void` **`openFile`**`()`

Метод для відкриття збереженого файлу.

---

exportImage

`void` **`exportImage`**`(int idx)`

Метод для збереження зображення діаграми.

Parameters:

`idx` - - індекс поточного файлу в компоненті `JTabbedPane`.

---

closeFile

`void` **`closeFile`**`(int idx)`

Метод, що закриває поточний файл.

Parameters:

`idx` - - індекс поточного файлу в компоненті `JTabbedPane`.

---

about

`void` **`about`**`()`

Метод, що показує інформацію про програму та розробника.

---

exit

`void` **`exit`**`()`

Метод, що може призвести до виходу з програми.

---

add

`void` **`add`**`(int idx)`

Метод, що додає поле в діаграму поточного файлу.

Parameters:

`idx` - - індекс поточного файлу в компоненті `JTabbedPane`.

---

clear

`void` **`clear`**`(int idx)`

Метод, що видаляє всі поля з діаграми поточного файлу.

Parameters:

`idx` - - індекс поточного файлу в компоненті `JTabbedPane`.

---

setEditableTable

```
void setEditableTable(int idx,
                      boolean flag)
```

Метод, що встановлює можливість редагування таблиці.

Parameters:

`idx` - - індекс поточного файлу в компоненті `JTabbedPane`.

`flag` - - true або false. True - таблицю можна редагувати, false - не можна.

---

com.lab111.work
Interface AbstractGUIController

All Known Implementing Classes:
DesignController

public interface **AbstractGUIController**
Інтерфейс AbstractGUIController оголошує інтерфейс котролерів, що контролюють контролюють дії над класом `Design`.

Author: [redacted]

Method Summary	
void	**about**() Метод, що показує інформацію про програму та розробника.
void	**add**(int idx) Метод, що додає поле в діаграму поточного файлу.
void	**clear**(int idx) Метод, що видаляє всі поля з діаграми поточного файлу.
void	**closeFile**(int idx) Метод, що закриває поточний файл.
void	**exit**() Метод, що може призвести до виходу з програми.
void	**exportImage**(int idx) Метод для збереження зображення діаграми.
void	**newFile**() Метод створює новий файл.
void	**openFile**() Метод для відкриття збереженого файлу.
void	**saveFile**(int idx) Метод зберігає поточний файл в компоненті `JTabbedPane`.
void	**saveFileAs**(int idx) Метод, що зберігає поточний файл в новому файлі.
void	**setEditableTable**(int idx, boolean flag) Метод, що встановлює можливість редагування таблиці.

Method Detail

saveFile

void **saveFile**(int idx)

Метод зберігає поточний файл в компоненті `JTabbedPane`.

Parameters:

`idx` - - індекс файлу, що зберігається в компоненті `JTabbedPane`.

saveFileAs

`void` **`saveFileAs`**`(int idx)`

Метод, що зберігає поточний файл в новому файлі.

Parameters:

`idx` - - індекс поточного файлу в компоненті `JTabbedPane`.

---

newFile

`void` **`newFile`**`()`

Метод створює новий файл.

---

openFile

`void` **`openFile`**`()`

Метод для відкриття збереженого файлу.

---

exportImage

`void` **`exportImage`**`(int idx)`

Метод для збереження зображення діаграми.

Parameters:

`idx` - - індекс поточного файлу в компоненті `JTabbedPane`.

---

closeFile

`void` **`closeFile`**`(int idx)`

Метод, що закриває поточний файл.

Parameters:

`idx` - - індекс поточного файлу в компоненті `JTabbedPane`.

---

about

`void` **`about`**`()`

Метод, що показує інформацію про програму та розробника.

---

exit

`void` **`exit`**`()`

Метод, що може призвести до виходу з програми.

---

add

`void` **`add`**`(int idx)`

Метод, що додає поле в діаграму поточного файлу.

Parameters:

`idx` - - індекс поточного файлу в компоненті `JTabbedPane`.

---

clear

`void` **`clear`**`(int idx)`

Метод, що видаляє всі поля з діаграми поточного файлу.

Parameters:

`idx` - - індекс поточного файлу в компоненті `JTabbedPane`.

---

setEditableTable

`void` **`setEditableTable`**`(int idx,`

boolean flag)

Метод, що встановлює можливість редагування таблиці.

Parameters:

`idx` - - індекс поточного файлу в компоненті `JTabbedPane`.

`flag` - - true або false. True - таблицю можна редагувати, false - не можна.

Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

com.lab111.work

Class AddPieGUI

```
java.lang.Object
  └ com.lab111.work.AddPieGUI
```

```
public class AddPieGUI
extends java.lang.Object
```

Клас що визначає вигляд вікна додавання поля в діаграму, і реагує на дії користувача під час додавання поля.

Author: [redacted]

Method Summary	
static void	**createPie**(java.util.Observable fileDesignModel) Метод, що створює новий діалог додавання поля в діаграму.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

createPie

```
public static void createPie(java.util.Observable fileDesignModel)
```

Метод, що створює новий діалог додавання поля в діаграму.

Parameters:

`fileDesignModel` - - модель файлу, в який додається нове поле діаграми.

Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

com.lab111.work
Class CSVProcessor

```
java.lang.Object
  └ com.lab111.work.CSVProcessor
```

All Implemented Interfaces:
java.io.Serializable, java.lang.Runnable

```
public class CSVProcessor
extends java.lang.Object
implements java.io.Serializable, java.lang.Runnable
```

Клас, що реалізує методи для роботи з CSV файлом.

Author:
[redacted]

See Also:
Serialized Form

Constructor Summary
CSVProcessor() Конструктор класу CSVProcessor.

Method Summary	
java.util.ArrayList<java.lang.String>	**getList**() Метод, що повертає масив-список рядків CSV файлу.
Properties	**getProperties**() Метод, що повертає об'єкт Properties.
void	**parse**() Метод, що виділяє з рядків list класу CSVProcessor поля.
void	**print**() Метод, що виводить вміст поля list класу CSVProcessor.
void	**printMatrix**() Метод, що виводить двовимірний масив String[][].
boolean	**readCSVFile**(java.io.BufferedReader br) Метод, що зчитує рядки з CSV файлу в поле list класу CSVProcessor.
void	**run**()
boolean	**writeCSVFile**(java.io.BufferedWriter bw) Метод, що записує рядки в файл з поля list класу CSVProcessor.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

CSVProcessor

```
public CSVProcessor()
```

Конструктор класу CSVProcessor.

Method Detail

getList

`public final java.util.ArrayList<java.lang.String> getList()`

Метод, що повертає масив-список рядків CSV файлу.

Returns:

list.

---

readCSVFile

`public final boolean readCSVFile(java.io.BufferedReader br)`

Метод, що зчитує рядки з CSV файлу в поле list класу CSVProcessor.

Parameters:

`br` - - буферизований потік зчитування файлу.

Returns:

- true у випадку вдалого зчитування, інакше - false.

---

writeCSVFile

`public final boolean writeCSVFile(java.io.BufferedWriter bw)`

Метод, що записує рядки в файл з поля list класу CSVProcessor.

Parameters:

`bw` - - буферизований потік запису в потрібний файл.

Returns:

- true, якщо запис виконано вдало, інакше - false.

---

print

`public final void print()`

Метод, що виводить вміст поля list класу CSVProcessor.

---

parse

`public final void parse()`

Метод, що виділяє з рядків list класу CSVProcessor поля.

---

printMatrix

`public final void printMatrix()`

Метод, що виводить двовимірний масив String[][].

---

run

`public final void run()`

Specified by:

`run` in interface `java.lang.Runnable`

---

getProperties

`public final Properties getProperties()`

Метод, що повертає об'єкт `Properties`.

Returns:

поле properties.

---

com.lab111.work
Class Design

```
java.lang.Object
  └ com.lab111.work.Design
```

All Implemented Interfaces:
java.util.Observer

```
public class Design
extends java.lang.Object
implements java.util.Observer
```

Клас, що відображає представлення програми, її вигляд. Виступає в ролі представлення в шаблоні MVC.

Author:
[redacted]

Constructor Summary
Design(DesignModel model) Конструктор класу.

Method Summary	
void	**update**(java.util.Observable obs, java.lang.Object obj)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Design

```
public Design(DesignModel model)
```

Конструктор класу.

Parameters:
`model` - - модель для даного класу.

Method Detail

update

```
public void update(java.util.Observable obs,
                   java.lang.Object obj)
```

Specified by:
`update` in interface `java.util.Observer`

com.lab111.work
Class DesignController

```
java.lang.Object
  └ com.lab111.work.DesignController
```

All Implemented Interfaces:
AbstractGUIController

```
public class DesignController
extends java.lang.Object
implements AbstractGUIController
```

Клас - контролер для головного вікна.

Author:
[redacted]

Constructor Summary
DesignController(DesignModel model) Конструктор класу.

Method Summary	
void	**about**() Метод, що показує інформацію про програму та розробника.
void	**add**(int idx) Метод, що додає поле в діаграму поточного файлу.
void	**clear**(int idx) Метод, що видаляє всі поля з діаграми поточного файлу.
void	**closeFile**(int idx) Метод, що закриває поточний файл.
void	**exit**() Метод, що може призвести до виходу з програми.
void	**exportImage**(int idx) Метод для збереження зображення діаграми.
void	**newFile**() Метод створює новий файл.
void	**openFile**() Метод для відкриття збереженого файлу.
void	**saveFile**(int idx) Метод зберігає поточний файл в компоненті JTabbedPane.
void	**saveFileAs**(int idx) Метод, що зберігає поточний файл в новому файлі.
void	**setEditableTable**(int idx, boolean flag) Метод, що встановлює можливість редагування таблиці.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

DesignController

```
public DesignController(DesignModel model)
```

Конструктор класу.

Parameters:

`model` - - модель класу-представлення.

Method Detail

saveFile

```
public void saveFile(int idx)
```

Description copied from interface: AbstractGUIController

Метод зберігає поточний файл в компоненті `JTabbedPane`.

Specified by:

`saveFile` in interface `AbstractGUIController`

Parameters:

`idx` - - індекс файлу, що зберігається в компоненті `JTabbedPane`.

---

saveFileAs

```
public final void saveFileAs(int idx)
```

Description copied from interface: AbstractGUIController

Метод, що зберігає поточний файл в новому файлі.

Specified by:

`saveFileAs` in interface `AbstractGUIController`

Parameters:

`idx` - - індекс поточного файлу в компоненті `JTabbedPane`.

---

newFile

```
public final void newFile()
```

Description copied from interface: AbstractGUIController

Метод створює новий файл.

Specified by:

`newFile` in interface `AbstractGUIController`

---

openFile

```
public final void openFile()
```

Description copied from interface: AbstractGUIController

Метод для відкриття збереженого файлу.

Specified by:

`openFile` in interface `AbstractGUIController`

---

exportImage

```
public final void exportImage(int idx)
```

Description copied from interface: AbstractGUIController

Метод для збереження зображення діаграми.

Specified by:

`exportImage` in interface `AbstractGUIController`

Parameters:

`idx` - - індекс поточного файлу в компоненті `JTabbedPane`.

---

closeFile

```
public final void closeFile(int idx)
```

Description copied from interface: AbstractGUIController

Метод, що закриває поточний файл.

Specified by:

closeFile in interface AbstractGUIController

Parameters:

idx - - індекс поточного файлу в компоненті JTabbedPane.

---

about

```
public final void about()
```

Description copied from interface: AbstractGUIController

Метод, що показує інформацію про програму та розробника.

Specified by:

about in interface AbstractGUIController

---

exit

```
public final void exit()
```

Description copied from interface: AbstractGUIController

Метод, що може призвести до виходу з програми.

Specified by:

exit in interface AbstractGUIController

---

add

```
public final void add(int idx)
```

Description copied from interface: AbstractGUIController

Метод, що додає поле в діаграму поточного файлу.

Specified by:

add in interface AbstractGUIController

Parameters:

idx - - індекс поточного файлу в компоненті JTabbedPane.

---

clear

```
public final void clear(int idx)
```

Description copied from interface: AbstractGUIController

Метод, що видаляє всі поля з діаграми поточного файлу.

Specified by:

clear in interface AbstractGUIController

Parameters:

idx - - індекс поточного файлу в компоненті JTabbedPane.

---

setEditableTable

```
public final void setEditableTable(int idx,
                                   boolean flag)
```

Description copied from interface: AbstractGUIController

Метод, що встановлює можливість редагування таблиці.

Specified by:

setEditableTable in interface AbstractGUIController

Parameters:

idx - - індекс поточного файлу в компоненті JTabbedPane.

flag - - true або false. True - таблицю можна редагувати, false - не можна.

Package Class Use Tree Deprecated Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

com.lab111.work
Class DesignModel

```
java.lang.Object
  └java.util.Observable
      └com.lab111.work.DesignModel
```

```
public class DesignModel
extends java.util.Observable
```

Клас-модель для головного вікна. Зберігає стан головного вікна.

Author: [redacted]

Constructor Summary
DesignModel() Конструктор класу.
DesignModel(java.util.Observer observer) Конструктор класу.

Method Summary	
void	**add**(int idx) Метод для додавання поля в діаграму.
void	**addObserver**(java.util.Observer o)
void	**clear**(int idx) Метод для видалення всіх полів з діаграми.
void	**closeFile**(int idx) Метод для закриття файлу.
void	**exit**() Метод для виходу з програми.
void	**exportImage**(int idx, java.lang.String filename) Метод для збереження зображення.
int	**getAmount**() Метод, що повертає кількість створених файлів в даній сесії.
java.util.ArrayList<FileDesign>	**getFileDesigns**() Метод, що повертає масив-список FileDesign.
boolean	**isClose**() Метод, що повертає можливість закривання.
boolean	**isExportImage**() Метод, що повертає можливість збереження зображення.

boolean	**isSave**() Метод, що повертає можливість зберігання.
void	**newFile**() Метод для створення нового файлу.
void	**notifyObservers**()
void	**notifyObservers**(java.lang.Object obj)
void	**openFile**(java.io.File file) Метод для відкриття збереженого файлу.
void	**saveFile**(int idx, java.lang.String filename) Метод для збереження файлу.
void	**setAmount**(int newAmount) Метод для зміни кількості створених файлів.
void	**setClose**(boolean newIsClose) Метод, що змінює можливість закривання.
void	**setEditableTable**(int idx, boolean flag) Метод для зміни можливості редагування таблиці.
void	**setExportImage**(boolean newIsExportImage) Метод, що змінює можливість збереження зображення.
void	**setSave**(boolean newIsSave) Метод, що змінює можливість зберігання.

Methods inherited from class java.util.Observable
countObservers, deleteObserver, deleteObservers, hasChanged

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

DesignModel

public **DesignModel**(java.util.Observer observer)

Конструктор класу.

Parameters:

observer - - спостерігач за моделю.

DesignModel

public **DesignModel**()

Конструктор класу.

Method Detail

isSave

public boolean **isSave**()

Метод, що повертає можливість зберігання.

Returns:

- поле isSave.

---

setSave

`public final void setSave(boolean newIsSave)`

Метод, що змінює можливість зберігання.

Parameters:

`newIsSave` - - новий стан можливості зберігання.

---

isClose

`public final boolean isClose()`

Метод, що повертає можливість закривання.

Returns:

- поле isClose.

---

setClose

`public final void setClose(boolean newIsClose)`

Метод, що змінює можливість закривання.

Parameters:

`newIsClose` - - новий стан можливості закривання.

---

isExportImage

`public final boolean isExportImage()`

Метод, що повертає можливість збереження зображення.

Returns:

- поле isExportImage.

---

setExportImage

`public final void setExportImage(boolean newIsExportImage)`

Метод, що змінює можливість збереження зображення.

Parameters:

`newIsExportImage` - - новий стан можливості збереження зображення.

---

addObserver

`public final void addObserver(java.util.Observer o)`

Overrides:

`addObserver` in class `java.util.Observable`

---

notifyObservers

`public final void notifyObservers()`

Overrides:

`notifyObservers` in class `java.util.Observable`

---

getAmount

`public final int getAmount()`

Метод, що повертає кількість створених файлів в даній сесії.

Returns:

- поле amount.

---

setAmount

`public final void setAmount(int newAmount)`

Метод для зміни кількості створених файлів.

Parameters:
`newAmount` - - нова кількість створених файлів.

---

newFile

```
public final void newFile()
```

Метод для створення нового файлу.

---

notifyObservers

```
public final void notifyObservers(java.lang.Object obj)
```

Overrides:
`notifyObservers` in class `java.util.Observable`

---

saveFile

```
public final void saveFile(int idx,
                           java.lang.String filename)
```

Метод для збереження файлу.
Parameters:
`idx` - - індекс вкладки в панелі вкладок.
`filename` - - ім'я файлу.

---

openFile

```
public final void openFile(java.io.File file)
```

Метод для відкриття збереженого файлу.
Parameters:
`file` - - шлях до файлу.

---

exportImage

```
public final void exportImage(int idx,
                              java.lang.String filename)
```

Метод для збереження зображення.
Parameters:
`idx` - - індекс вкладки файла в панелі вкладок.
`filename` - - шлях до зображення.

---

closeFile

```
public final void closeFile(int idx)
```

Метод для закриття файлу.
Parameters:
`idx` - - індекс файлу в панелі вкладок.

---

add

```
public final void add(int idx)
```

Метод для додавання поля в діаграму.
Parameters:
`idx` - - індекс файлу в панелі вкладок.

---

clear

```
public final void clear(int idx)
```

Метод для видалення всіх полів з діаграми.

Parameters:
`idx` - - індекс файлу в панелі вкладок.

---

exit

```
public final void exit()
```

Метод для виходу з програми.

---

setEditableTable

```
public final void setEditableTable(int idx,
                                   boolean flag)
```

Метод для зміни можливості редагування таблиці.
Parameters:
`idx` - - індекс файлу в панелі вкладок.
`flag` - - новий стан можливості редагування таблиці..

---

getFileDesigns

```
public final java.util.ArrayList<FileDesign> getFileDesigns()
```

Метод, що повертає масив-список FileDesign.
Returns:
- поле fileDesigns.

Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

---

com.lab111.work
Class FileDesign

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └com.lab111.work.FileDesign
```

All Implemented Interfaces:
java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.Observer, javax.accessibility.Accessible

---

```
public class FileDesign
extends javax.swing.JPanel
implements java.util.Observer
```

Клас, що являє собою вигляд csv файлу.
Author:
[redacted]
See Also:
Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JComponent
javax.swing.JComponent.AccessibleJComponent

Nested classes/interfaces inherited from class java.awt.Component
java.awt.Component.BaselineResizeBehavior

Field Summary

Fields inherited from class javax.swing.JComponent
TOOL_TIP_TEXT_KEY, UNDEFINED_CONDITION, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED, WHEN_IN_FOCUSED_WINDOW

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary
FileDesign(Properties p, FileDesignModel model) Конструктор класу.

Method Summary	
java.awt.image.BufferedImage	**createImage**() Метод для створення зображення.
com.lab111.work.AbstractFDController	**getController**() Метод, що повертає контролер даного класу.
FileDesignModel	**getModel**() Метод, що повертає модель класу.
Properties	**getProperties**() Метод для отримання об'єкту properties.
void	**initTable**() Метод для створення таблиці.
void	**setDrawer**(Drawer d) Метод для зміни drawer.
void	**setProperties**(Properties p) Метод для зміни properties.
void	**update**(java.util.Observable o, java.lang.Object arg)

Methods inherited from class javax.swing.JPanel
getAccessibleContext, getUI, getUIClassID, setUI, updateUI

Methods inherited from class javax.swing.JComponent

addAncestorListener, addNotify, addVetoableChangeListener, computeVisibleRect, contains, createToolTip, disable, enable, firePropertyChange, firePropertyChange, firePropertyChange, getActionForKeyStroke, getActionMap, getAlignmentX, getAlignmentY, getAncestorListeners, getAutoscrolls, getBaseline, getBaselineResizeBehavior, getBorder, getBounds, getClientProperty, getComponentPopupMenu, getConditionForKeyStroke, getDebugGraphicsOptions, getDefaultLocale, getFontMetrics, getGraphics, getHeight, getInheritsPopupMenu, getInputMap, getInputMap, getInputVerifier, getInsets, getInsets, getListeners, getLocation, getMaximumSize, getMinimumSize, getNextFocusableComponent, getPopupLocation, getPreferredSize, getRegisteredKeyStrokes, getRootPane, getSize, getToolTipLocation, getToolTipText, getToolTipText, getTopLevelAncestor, getTransferHandler, getVerifyInputWhenFocusTarget, getVetoableChangeListeners, getVisibleRect, getWidth, getX, getY, grabFocus, isDoubleBuffered, isLightweightComponent, isManagingFocus, isOpaque, isOptimizedDrawingEnabled, isPaintingForPrint, isPaintingTile, isRequestFocusEnabled, isValidateRoot, paint, paintImmediately, paintImmediately, print, printAll, putClientProperty, registerKeyboardAction, registerKeyboardAction, removeAncestorListener, removeNotify, removeVetoableChangeListener, repaint, repaint, requestDefaultFocus, requestFocus, requestFocus, requestFocusInWindow, resetKeyboardActions, reshape, revalidate, scrollRectToVisible, setActionMap, setAlignmentX, setAlignmentY, setAutoscrolls, setBackground, setBorder, setComponentPopupMenu, setDebugGraphicsOptions, setDefaultLocale, setDoubleBuffered, setEnabled, setFocusTraversalKeys, setFont, setForeground, setInheritsPopupMenu, setInputMap, setInputVerifier, setMaximumSize, setMinimumSize, setNextFocusableComponent, setOpaque, setPreferredSize, setRequestFocusEnabled, setToolTipText, setTransferHandler, setVerifyInputWhenFocusTarget, setVisible, unregisterKeyboardAction, update

Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, addPropertyChangeListener, addPropertyChangeListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder, getContainerListeners, getFocusTraversalKeys, getFocusTraversalPolicy, getLayout, getMousePosition, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusCycleRoot, isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paintComponents, preferredSize, printComponents, remove, remove, removeAll, removeContainerListener, setComponentZOrder, setFocusCycleRoot, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setLayout, transferFocusDownCycle, validate

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, contains, createImage, createImage, createVolatileImage, createVolatileImage, dispatchEvent, enable, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusCycleRootAncestor, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getForeground, getGraphicsConfiguration, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputContext, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocale, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMousePosition, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getToolkit, getTreeLock, gotFocus, handleEvent, hasFocus, hide,

imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet, isPreferredSizeSet, isShowing, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, remove, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, resize, resize, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setFocusable, setFocusTraversalKeysEnabled, setIgnoreRepaint, setLocale, setLocation, setLocation, setName, setSize, setSize, show, show, size, toString, transferFocus, transferFocusBackward, transferFocusUpCycle

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

FileDesign

```
public FileDesign(Properties p,
                  FileDesignModel model)
```

Конструктор класу.

Parameters:

p - - властивості csv файлу.

model - - модель файлу.

Method Detail

setDrawer

```
public void setDrawer(Drawer d)
```

Метод для зміни drawer.

Parameters:

d - - новий drawer.

---

initTable

```
public void initTable()
```

Метод для створення таблиці.

---

setProperties

```
public void setProperties(Properties p)
```

Метод для зміни properties.

Parameters:

p - - об’єкт класу Properties.

---

update

```
public void update(java.util.Observable o,
                   java.lang.Object arg)
```

Specified by:

update in interface java.util.Observer

---

getProperties

`public Properties getProperties()`

Метод для отримання об'єкту properties.

Returns:

- поле properties.

---

createImage

`public java.awt.image.BufferedImage createImage()`

Метод для створення зображення.

Returns:

- нове зображення.

---

getController

`public com.lab111.work.AbstractFDController getController()`

Метод, що повертає контролер даного класу.

Returns:

- поле controller.

---

getModel

`public FileDesignModel getModel()`

Метод, що повертає модель класу.

Returns:

- поле model.

Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

com.lab111.work

Class FileDesignController

```
java.lang.Object
  └ com.lab111.work.FileDesignController
```

`public class FileDesignController`
`extends java.lang.Object`

Клас контролер для класу FileDesign. Є підкласом абстрактного класу, AbstractFDController

Author:

[redacted]

Constructor Summary
FileDesignController(FileDesignModel model) Конструктор класу.

Method Summary	
void	**add**() Метод для додавання поля в кінець діаграми.

void	**clear**() Метод для видалення всіх полів з діаграми.
void	**remove**() Метод для видалення поля з кінця діаграми.
void	**remove**(int index) Метод для видалення поля з діаграми за переданим індексом.
void	**setAmount**(int index, java.lang.String amount) Метод для зміни розміру поля за переданим номером.
void	**setColor**(int index, java.awt.Color color) Метод для зміни кольору поля за переданим номером.
void	**setName**(int index, java.lang.String name) Метод для зміни назви поля за переданим номером.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

FileDesignController

public **FileDesignController**(FileDesignModel model)

Конструктор класу.

Method Detail

add

public void **add**()

Метод для додавання поля в кінець діаграми.

---

clear

public void **clear**()

Метод для видалення всіх полів з діаграми.

---

remove

public void **remove**()

Метод для видалення поля з кінця діаграми.

---

remove

public void **remove**(int index)

Метод для видалення поля з діаграми за переданим індексом.

Parameters:

index - - номер поля, що видаляється.

---

setAmount

public void **setAmount**(int index,
java.lang.String amount)

Метод для зміни розміру поля за переданим номером.

Parameters:

index - - номер поля діаграми, що змінюється.

amount - - новий розмір поля діаграми.

setColor

```
public void setColor(int index,
                     java.awt.Color color)
```

Метод для зміни кольору поля за переданим номером.

Parameters:

`index` - - номер поля діаграми, що змінюється.

`color` - - новий колір поля діаграми.

---

setName

```
public void setName(int index,
                    java.lang.String name)
```

Метод для зміни назви поля за переданим номером.

Parameters:

`index` - - номер поля діаграми, що змінюється.

`name` - - ова назва поля діаграми.

Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

com.lab111.work

Class FileDesignModel

```
java.lang.Object
  └java.util.Observable
      └com.lab111.work.FileDesignModel
```

```
public class FileDesignModel
extends java.util.Observable
```

Клас, модель для класу FileDesign.

Author:

[redacted]

Constructor Summary
FileDesignModel(java.util.Observer observer, Properties properties) Конструктор класу.
FileDesignModel(Properties properties) Конструктор класу.

Method Summary	
void	**add**() Метод для додавання поля в діаграму.
void	**addObserver**(java.util.Observer o)
void	**clear**()

	Метод для видалення всіх полів діаграми.
java.lang.String	**getPath**() Метод для отримання шляху до файла.
Properties	**getProperties**() Метод для отримання properties.
java.util.ArrayList<java.lang.String>	**getRow**() Метод, що повертає row.
boolean	**isEditableTable**() Метод, що повертає можливість редагування таблиці.
boolean	**isSaved**() Метод, що повертає стан збереження файлу.
boolean	**isSaveInFact**() Метод, що показує чи файл збережено на диск.
void	**notifyObservers**()
void	**notifyObservers**(java.lang.Object obj)
void	**remove**() Метод для видалення поля з кінця діаграми.
void	**remove**(int index) Метод для видалення поля з діаграми за індексом.
void	**setEditableTable**(boolean isEditableTable) Метод, що встановлює можливість редагування таблиці.
void	**setPath**(java.lang.String path) Метод для зміни шляху до файлу.
void	**setProperties**(Properties properties) Метод для зміни properties.
void	**setRow**(java.util.ArrayList<java.lang.String> row) Метод, що змінює row.
void	**setSaveInFact**(boolean isSaveInFact) Метод, що встановлює чи файл збережено на диск.

Methods inherited from class java.util.Observable
countObservers, deleteObserver, deleteObservers, hasChanged

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

FileDesignModel

```
public FileDesignModel(java.util.Observer observer,
                       Properties properties)
```

Конструктор класу.

Parameters:

`observer` - - спостерігач за моделю.

`properties` - - вміст csv файлу.

FileDesignModel

`public FileDesignModel(Properties properties)`

Конструктор класу.

Parameters:

`properties` - - вміст csv файлу.

Method Detail

isEditableTable

`public boolean isEditableTable()`

Метод, що повертає можливість редагування таблиці.

Returns:

- поле isEditableTable.

---

setEditableTable

`public void setEditableTable(boolean isEditableTable)`

Метод, що встановлює можливість редагування таблиці.

Parameters:

`isEditableTable` - - новий стан можливості редагування.

---

isSaved

`public boolean isSaved()`

Метод, що повертає стан збереження файлу.

Returns:

- поле isSaved.

---

setRow

`public void setRow(java.util.ArrayList<java.lang.String> row)`

Метод, що змінює row.

Parameters:

`row` - - новий рядок.

---

getRow

`public java.util.ArrayList<java.lang.String> getRow()`

Метод, що повертає row.

Returns:

- поле row.

---

addObserver

`public void addObserver(java.util.Observer o)`

Overrides:

`addObserver` in class `java.util.Observable`

---

notifyObservers

`public void notifyObservers()`

Overrides:

`notifyObservers` in class `java.util.Observable`

---

notifyObservers

`public void notifyObservers(java.lang.Object obj)`

Overrides:

`notifyObservers` in class `java.util.Observable`

---

setPath

`public void setPath(java.lang.String path)`

Метод для зміни шляху до файлу.

Parameters:

`path` - - новий шлях.

---

add

`public void add()`

Метод для додавання поля в діаграму.

---

remove

`public void remove()`

Метод для видалення поля з кінця діаграми.

---

remove

`public void remove(int index)`

Метод для видалення поля з діаграми за індексом.

Parameters:

`index` - - індекс поля, що видаляється.

---

clear

`public void clear()`

Метод для видалення всіх полів діаграми.

---

getProperties

`public Properties getProperties()`

Метод для отримання properties.

Returns:

- поле properties.

---

setProperties

`public void setProperties(Properties properties)`

Метод для зміни properties.

Parameters:

`properties` - - новий об’єкт properties.

---

isSaveInFact

`public boolean isSaveInFact()`

Метод, що показує чи файл збережено на диск.

Returns:

- true, якщо збережено; інакше - false.

---

setSaveInFact

`public void` **`setSaveInFact`**`(boolean isSaveInFact)`

Метод, що встановлює чи файл збережено на диск.

Parameters:

`isSaveInFact` - - новий стан.

---

getPath

`public java.lang.String` **`getPath`**`()`

Метод для отримання шляху до файла.

Returns:

- шлях до файлу. Поле path.

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**

PREV CLASS **NEXT CLASS** **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

---

com.lab111.work

Class Pie

```
java.lang.Object
  └java.awt.geom.RectangularShape
      └java.awt.geom.Arc2D
          └java.awt.geom.Arc2D.Double
              └com.lab111.work.Pie
```

All Implemented Interfaces:

java.awt.Shape, java.io.Serializable, java.lang.Cloneable

`public class` **`Pie`**
`extends java.awt.geom.Arc2D.Double`

Клас що являє поле з секторної діаграми.

Author:

[redacted]

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class java.awt.geom.Arc2D
java.awt.geom.Arc2D.Double, java.awt.geom.Arc2D.Float

Field Summary

Fields inherited from class java.awt.geom.Arc2D.Double
extent, height, start, width, x, y

Fields inherited from class java.awt.geom.Arc2D

CHORD, OPEN, PIE

Constructor Summary

Pie(double xx, double yy, double w, double h, double startDot, double size, int type)
Конструктор класу.

Method Summary

Methods inherited from class java.awt.geom.Arc2D.Double

getAngleExtent, getAngleStart, getHeight, getWidth, getX, getY, isEmpty, setAngleExtent, setAngleStart, setArc

Methods inherited from class java.awt.geom.Arc2D

contains, contains, contains, containsAngle, equals, getArcType, getBounds2D, getEndPoint, getPathIterator, getStartPoint, hashCode, intersects, setAngles, setAngles, setAngleStart, setArc, setArc, setArc, setArcByCenter, setArcByTangent, setArcType, setFrame

Methods inherited from class java.awt.geom.RectangularShape

clone, contains, getBounds, getCenterX, getCenterY, getFrame, getMaxX, getMaxY, getMinX, getMinY, getPathIterator, intersects, setFrame, setFrame, setFrameFromCenter, setFrameFromCenter, setFrameFromDiagonal, setFrameFromDiagonal

Methods inherited from class java.lang.Object

getClass, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Pie

```
public Pie(double xx,
           double yy,
           double w,
           double h,
           double startDot,
           double size,
           int type)
```

Конструктор класу.

Parameters:

xx - - верхня ліва координата.

yy - - верхня права координата.

w - - ширина.

h - - висота.

startDot - - початковий кут.

size - - розмір кута.

type - - тип.

com.lab111.work
Class Properties

```
java.lang.Object
  └ com.lab111.work.Properties
```

```
public class Properties
extends java.lang.Object
```

Клас, що репрезентує вміст csv файлу.

Author: [redacted]

Constructor Summary
Properties(int rowCount, int colCount) Конструктор класу.

Method Summary	
void	**addRow**(java.util.ArrayList<java.lang.String> newRow) Метод для додавання рядку в масив.
boolean	**checkError**(java.util.ArrayList<java.lang.String> list) Метод для перевірки рядка на відповідність формату.
boolean	**checkError**(java.lang.String str, int row, int col) Метод для перевірки потенційного вмісту комірки.
void	**clean**() Метод для видалення всіх рядків.
Properties	**clone**() Метод для клонування об'єкту Properties.
java.lang.String	**getCell**(int row, int col) Метод, що повертає вміст комірки.
int	**getColCount**() Метод для отримання кількості колонок.
java.lang.String[][]	**getProp**() Метод, що повертає масив csv файлу.
int	**getRowCount**() Метод для отримання кількості рядків.
void	**incSize**() Мето для збільшення кількості рядків на 1.
void	**print**() Метод для виведення матриці.
void	**removeRow**() Метод для видалення останього рядка з матриці.
void	**removeRow**(int index) Метод для видалення рядка за індексом.
boolean	**setCell**(java.lang.String str, int row, int col)

	Метод для зміни вмісту комірки.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Properties

```
public Properties(int rowCount,
                  int colCount)
```

Конструктор класу.

Parameters:

`rowCount` - - кількість рядків.

`colCount` - - кількість стовпчиків.

Method Detail

setCell

```
public boolean setCell(java.lang.String str,
                       int row,
                       int col)
```

Метод для зміни вмісту комірки.

Parameters:

`str` - - новий вміст комірки.

`row` - - рядок.

`col` - - стовпчик.

Returns:

- true, якщо комірку відредаговано вдало.

---

addRow

```
public void addRow(java.util.ArrayList<java.lang.String> newRow)
            throws CSVParseException
```

Метод для додавання рядку в масив.

Parameters:

`newRow` - - новий рядок.

Throws:

`CSVParseException` - - якщо неправильний формат рядка.

---

checkError

```
public boolean checkError(java.util.ArrayList<java.lang.String> list)
```

Метод для перевірки рядка на відповідність формату.

Parameters:

`list` - - рядок, що перевіряється.

Returns:

- true, якщо рядок відповідає формату, інакше - false.

---

checkError

```
public boolean checkError(java.lang.String str,
                          int row,
                          int col)
```

Метод для перевірки потенційного вмісту комірки.

Parameters:

`str` - - новий вміст.

row - - рядок.
col - - колонка.
Returns:
- true, якщо слово відповідає формату, інакше - false.

---

getCell

```
public java.lang.String getCell(int row,
                                 int col)
```

Метод, що повертає вміст комірки.
Parameters:
row - - рядок.
col - - колонка.
Returns:
- вміст комірки.

---

getProp

```
public java.lang.String[][] getProp()
```

Метод, що повертає масив csv файлу.
Returns:
- поле prop клоноване.

---

print

```
public void print()
```

Метод для виведення матриці.

---

clone

```
public Properties clone()
```

Метод для клонування об'єкту Properties.
Overrides:
clone in class java.lang.Object

---

getRowCount

```
public int getRowCount()
```

Метод для отримання кількості рядків.
Returns:
- кількість рядків.

---

getColCount

```
public int getColCount()
```

Метод для отримання кількості колонок.
Returns:
- кількість колонок.

---

incSize

```
public void incSize()
```

Мето для збільшення кількості рядків на 1.

---

removeRow

```
public void removeRow()
```

Метод для видалення останього рядка з матриці.

removeRow

`public void removeRow(int index)`

Метод для видалення рядка за індексом.

Parameters:

`index` - - індекс рядка, що видаляється.

---

clean

`public void clean()`

Метод для видалення всіх рядків.

Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

com.lab111.work

Class SectorDiagramDrawer

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └com.lab111.work.SectorDiagramDrawer
```

All Implemented Interfaces:

Drawer, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible

---

`public class SectorDiagramDrawer`
`extends javax.swing.JPanel`
`implements Drawer`

Метод, що являє полотно для малювання секторної діаграми.

Author:

[redacted]

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JComponent
javax.swing.JComponent.AccessibleJComponent

Nested classes/interfaces inherited from class java.awt.Component
java.awt.Component.BaselineResizeBehavior

Field Summary

Fields inherited from class javax.swing.JComponent
TOOL_TIP_TEXT_KEY, UNDEFINED_CONDITION, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED, WHEN_IN_FOCUSED_WINDOW

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary
SectorDiagramDrawer(int x, int y, Properties p) Конструктор класу.

Method Summary	
java.awt.image.BufferedImage	**createImage**() Метод для створення зображення.
void	**draw**(java.awt.Graphics g) Метод для малювання діаграми.
void	**makeJPopupMenu**() Метод для створення меню, що випливає.
void	**paintComponent**(java.awt.Graphics g)
void	**setParent**(javax.swing.JPanel parent) Метод для зміни панелі в якій міститься drawer.
void	**setProperties**(Properties p) Метод для зміни поля кламу Properties.

Methods inherited from class javax.swing.JPanel
getAccessibleContext, getUI, getUIClassID, setUI, updateUI

Methods inherited from class javax.swing.JComponent
addAncestorListener, addNotify, addVetoableChangeListener, computeVisibleRect, contains, createToolTip, disable, enable, firePropertyChange, firePropertyChange, firePropertyChange, getActionForKeyStroke, getActionMap, getAlignmentX, getAlignmentY, getAncestorListeners, getAutoscrolls, getBaseline, getBaselineResizeBehavior, getBorder, getBounds, getClientProperty, getComponentPopupMenu, getConditionForKeyStroke, getDebugGraphicsOptions, getDefaultLocale, getFontMetrics, getGraphics, getHeight, getInheritsPopupMenu, getInputMap, getInputMap, getInputVerifier, getInsets, getInsets, getListeners, getLocation, getMaximumSize, getMinimumSize, getNextFocusableComponent, getPopupLocation, getPreferredSize, getRegisteredKeyStrokes, getRootPane, getSize, getToolTipLocation, getToolTipText, getToolTipText, getTopLevelAncestor, getTransferHandler, getVerifyInputWhenFocusTarget, getVetoableChangeListeners, getVisibleRect, getWidth, getX, getY, grabFocus, isDoubleBuffered, isLightweightComponent, isManagingFocus, isOpaque, isOptimizedDrawingEnabled, isPaintingForPrint, isPaintingTile, isRequestFocusEnabled, isValidateRoot, paint, paintImmediately, paintImmediately, print, printAll, putClientProperty,

registerKeyboardAction, registerKeyboardAction, removeAncestorListener, removeNotify, removeVetoableChangeListener, repaint, repaint, requestDefaultFocus, requestFocus, requestFocus, requestFocusInWindow, resetKeyboardActions, reshape, revalidate, scrollRectToVisible, setActionMap, setAlignmentX, setAlignmentY, setAutoscrolls, setBackground, setBorder, setComponentPopupMenu, setDebugGraphicsOptions, setDefaultLocale, setDoubleBuffered, setEnabled, setFocusTraversalKeys, setFont, setForeground, setInheritsPopupMenu, setInputMap, setInputVerifier, setMaximumSize, setMinimumSize, setNextFocusableComponent, setOpaque, setPreferredSize, setRequestFocusEnabled, setToolTipText, setTransferHandler, setVerifyInputWhenFocusTarget, setVisible, unregisterKeyboardAction, update

Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, addPropertyChangeListener, addPropertyChangeListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder, getContainerListeners, getFocusTraversalKeys, getFocusTraversalPolicy, getLayout, getMousePosition, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusCycleRoot, isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paintComponents, preferredSize, printComponents, remove, remove, removeAll, removeContainerListener, setComponentZOrder, setFocusCycleRoot, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setLayout, transferFocusDownCycle, validate

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, contains, createImage, createImage, createVolatileImage, createVolatileImage, dispatchEvent, enable, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusCycleRootAncestor, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getForeground, getGraphicsConfiguration, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputContext, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocale, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMousePosition, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getToolkit, getTreeLock, gotFocus, handleEvent, hasFocus, hide, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet, isPreferredSizeSet, isShowing, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, remove, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, resize, resize, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setFocusable, setFocusTraversalKeysEnabled, setIgnoreRepaint, setLocale, setLocation, setLocation, setName, setSize, setSize, show, show, size, toString, transferFocus, transferFocusBackward, transferFocusUpCycle

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

SectorDiagramDrawer

```
public SectorDiagramDrawer(int x,
                           int y,
                           Properties p)
```

Конструктор класу.

Parameters:

x - - ширина панелі.

y - - висота панелі.

p - - об’єкт класу Properties.

Method Detail

paintComponent

```
public void paintComponent(java.awt.Graphics g)
```

Overrides:

paintComponent in class javax.swing.JComponent

---

setProperties

```
public void setProperties(Properties p)
```

Description copied from interface: Drawer

Метод для зміни поля кламу Properties.

Specified by:

setProperties in interface Drawer

Parameters:

p - - об’єкт класу Properties.

---

setParent

```
public void setParent(javax.swing.JPanel parent)
```

Метод для зміни панелі в якій міститься drawer.

Parameters:

parent - - нова панель.

---

draw

```
public void draw(java.awt.Graphics g)
```

Description copied from interface: Drawer

Метод для малювання діаграми.

Specified by:

draw in interface Drawer

Parameters:

g - - контекст малювання.

---

createImage

```
public java.awt.image.BufferedImage createImage()
```

Description copied from interface: Drawer

Метод для створення зображення.

Specified by:

createImage in interface Drawer

Returns:

- нове зображення.

makeJPopupMenu

`public void` **`makeJPopupMenu`**`()`

Метод для створення меню, що випливає.

Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

com.lab111.work
Class TestIO

```
java.lang.Object
  └ com.lab111.work.TestIO
```

`public class` **`TestIO`**
`extends java.lang.Object`

Головний клас, що містить метод main.

Author:

[redacted]

Constructor Summary
`TestIO``()`

Method Summary	
`static void`	**`main`**`(java.lang.String[] args)` Головний метод.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

TestIO

`public` **`TestIO`**`()`

Method Detail

main

`public static void` **`main`**`(java.lang.String[] args)`

Головний метод.

Parameters:

`args` - - додаткові параметри.

com.lab111.work
Class CSVParseException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └com.lab111.work.CSVParseException
```

All Implemented Interfaces:
java.io.Serializable

```
public class CSVParseException
extends java.lang.Exception
```

Клас що репрезентує виключні ситуації, що можуть статися під час роботи з CSV файлом.

Author:
[redacted]

See Also:
Serialized Form

Constructor Summary
CSVParseException() Конструктор класу CSVParseException.
CSVParseException(java.lang.String message) Конструктор класу CSVParseException.

Method Summary	
java.lang.String	**toString**()

Methods inherited from class java.lang.Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

CSVParseException

```
public CSVParseException()
```

Конструктор класу CSVParseException.

CSVParseException

```
public CSVParseException(java.lang.String message)
```

Конструктор класу CSVParseException.

Parameters:
`message` - - повідомлення, що супроводжує виключення.

Method Detail

toString

```
public final java.lang.String toString()
```

Overrides:
`toString` in class `java.lang.Throwable`

---

6. ВИСНОВКИ

Використовуючи пакети AWT та SWING я розробив графічний інтерфейс користувача. В основу архітектури покладено шаблон MVC.Також я використав шаблони Strategy і Observer. Також я використав механізм потоків java. Програма виконується в потоці обробки подій, а не в головному потоці.

Шаблон Observer дозволив забезпечити слабкий зв’язок між моделю та виглядом з шаблону MVC. Шаблон Strategy дозоляє динамічно змінювати діаграму.

Для реакції на дії користувача я використав внутрішні та анонімні класи, потім про подію повідомлявся контролер, який в свою чергу повідомляв модель про зміну.

В результаті я навчився проектувати та розробляти графічний інтерфейс. Програма дозволяє редагувати дані графічно, а також й таблично. Можна водночас працювати з багатьма файлами.

З допомогою зручного контекстного меню можна легко змінити колір, назву, розмір сектора, а також видалити його, якщо в цьому є потреба.

Інтерфей моєї програми є інтуїтивно зрозумілим, я використав іконки, аби надати програмі привабливості.

7.СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Герберт Шилдт

«Swing: руководство для начинающих», «В. Д. Вільямс», 2007.-704с

2. Екель Б

Философия Java. Библиотека программиста. 4-е езд. – СПБ.:Питер, 2009 – 640с.

3. Гама Е., Хелм Р., Джонсон Р., Влісідес Дж.

   Приемы объектно-ориентированого проектирования. Паттерны проектирования – СПб.: Питер, 2010 – 368с

4. К. Хорстманн, Г. Корнелл. «Java 2. Библиотека профессионала».
5. http://www.java2s.com/Tutorial/Java/CatalogJava.htm
6. http://sourcemaking.com/
7. Стивен Стелтинг, Олав Маасен - Применение шаблонов Java. Библиотека профессионала. : Пер. с англ. — М.:Издательский дом "Вильямс"
8. Кей С. Хорстманн, Гари Корнелл - Библиотека профессионала. Java 2. Том
9. Java API
10. Фленаган Д.

    Java. Справочник, 4-е издание – Пер. с англ. - СПб.: Символ-Плюс, 2004 – 1040 с

ДОДАТОК А. ПРОГРАМНИЙ КОД ПРОЕКТУ

A.1 AbstractFDController

```
package com.lab111.work;

import java.awt.Color;
/**
 * Інтерфейс AbstractFDController оголошує методи,
 * що контролюють дії над файлом.
 * @author [redacted]
 *
 */
interface AbstractFDController {

    /**
     * Метод для додавання поля в кінець діаграми.
     */
 void add();
 /**
  * Метод для видалення поля з кінця діаграми.
  */
 void remove();
 /**
  * Метод для видалення поля з діаграми
  * за переданим індексом.
  * @param index - номер поля, що видаляється.
  */
 void remove(int index);
 /**
  * Метод для видалення всіх полів з діаграми.
  */
 void clear();
 /**
  * Метод для зміни кольору поля за
  * переданим номером.
  * @param index - номер поля діаграми, що змінюється.
  * @param color - новий колір поля діаграми.
  */
 void setColor(int index, Color color);
 /**
  * Метод для зміни назви поля за
  * переданим номером.
  * @param index - номер поля діаграми, що змінюється.
  * @param name - ова назва поля діаграми.
  */
 void setName(int index, String name);
 /**
  * Метод для зміни розміру поля за
  * переданим номером.
  * @param index - номер поля діаграми, що змінюється.
  * @param amount - новий розмір поля діаграми.
  */
 void setAmount(int index, String amount);
}
```

A.2 AbstractGUIController

```
package com.lab111.work;
/**
 * Інтерфейс AbstractGUIController оголошує інтерфейс
 * котролерів, що контролюють контролюють дії
 * над класом {@link Design}.
 * @author r[redacted]
 *
 */
public interface AbstractGUIController {

 /**
  * Метод зберігає поточний файл в компоненті {@link JTabbedPane}.
  * @param idx - індекс файлу,
  * що зберігається в компоненті {@link JTabbedPane}.
  */
 void saveFile(int idx);
 /**
  * Метод, що зберігає поточний файл в новому файлі.
  * @param idx - індекс поточного файлу в компоненті {@link JTabbedPane}.
```

```
     */
    void saveFileAs(int idx);
    /**
     * Метод створює новий файл.
     */
    void newFile();
    /**
     * Метод для відкриття збереженого файлу.
     */
    void openFile();
    /**
     * Метод для збереження зображення діаграми.
     * @param idx - індекс поточного файлу в компоненті {@link JTabbedPane}.
     */
    void exportImage(int idx);
    /**
     * Метод, що закриває поточний файл.
     * @param idx - індекс поточного файлу в компоненті {@link JTabbedPane}.
     */
    void closeFile(int idx);
    /**
     * Метод, що показує інформацію про програму та розробника.
     */
    void about();
    /**
     * Метод, що може призвести до виходу з програми.
     */
    void exit();
    /**
     * Метод, що додає поле в діаграму поточного файлу.
     * @param idx - індекс поточного файлу в компоненті {@link JTabbedPane}.
     */
    void add(int idx);
    /**
     * Метод, що видаляє всі поля з діаграми поточного файлу.
     * @param idx - індекс поточного файлу в компоненті {@link JTabbedPane}.
     */
    void clear(int idx);
    /**
     * Метод, що встановлює можливість редагування таблиці.
     * @param idx - індекс поточного файлу в компоненті {@link JTabbedPane}.
     * @param flag - true або false. True - таблицю можна редагувати,
     * false - не можна.
     */
    void setEditableTable(int idx, boolean flag);
}
```

A.3 AddPieGUI

```
package com.lab111.work;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Observable;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JColorChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;

/**
 * Клас що визначає вигляд вікна додавання поля в діаграму,
 * і реагує на дії користувача під час додавання поля.
 * @author [redacted]
 *
 */
public class AddPieGUI {

    /**
```

```
 * Панель вибору кольору поля діаграми.
 */
private JColorChooser colorChooser;
/**
 * Кнопка підтвердження додавання поля в діаграму.
 */
private JButton buttonOk;
/**
 * Кнопка виходу з діалогу без додавання поля в діаграму.
 */
private JButton buttonCansel;
/**
 * Кнопка, що стирає введенні значеняв поля діалогу.
 */
private JButton buttonReset;
/**
 * Текстове поле для введення назви нового поля.
 */
private JTextField fieldName;
/**
 * Текстове поле для введення розміру нового поля.
 */
private JTextField fieldAmount;
/**
 * Фрейм діалогу додавання нового поля в діаграму.
 */
private JFrame frame;
/**
 * Модель поточного файлу в який додається нове поле.
 */
private Observable currentFile;
/**
 * Конструктор класу AddPieGUI.
 * @param file - модель файлу, в який додається нове поле.
 */
protected AddPieGUI(final Observable file) {
    currentFile = file;
    frame = new JFrame();
    frame.setSize(new Dimension(740, 490));
    frame.setResizable(false);
    frame.setLayout(new FlowLayout());
    frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    frame.setLocation((int) frame.getToolkit().
            getScreenSize().getWidth() / 4,
            (int) frame.getToolkit().
            getScreenSize().getHeight() / 4);
    init();
    frame.setVisible(true);
}
/**
 * Метод, що ініціалізує діалог
 * і додає в головний фрейм всі потрібні компоненти.
 */
private void init() {
    createButtons();
    createFields();
    colorChooser = new JColorChooser();
    JPanel panel1 = new JPanel();
    panel1.setLayout(new GridLayout(2, 2));
    panel1.add(new JLabel("Name"));
    panel1.add(new JLabel("Amount"));
    panel1.add(fieldName);
    panel1.add(fieldAmount);
    panel1.setBorder(BorderFactory.createEtchedBorder());
    panel1.setPreferredSize(
            new Dimension(frame.getWidth() - 20, 60));
    JPanel panel2 = new JPanel();
    panel2.setLayout(new GridLayout(1, 3));
    panel2.setPreferredSize(new Dimension(260,  30));
    panel2.add(buttonOk);
    panel2.add(buttonCansel);
    panel2.add(buttonReset);
    JPanel panel3 = new JPanel();
    panel3.setPreferredSize(
            new Dimension(frame.getWidth() - 20,  330));
    panel3.setLayout(new BorderLayout());
    panel3.setBorder(BorderFactory.createTitledBorder("Color"));
```

```
        panel3.add(colorChooser);
        frame.add(panel1);
        frame.add(panel3);
        frame.add(panel2);
    }
    /**
     * Метод, що ініціалізує поля кнопок
     * з відповідними налаштуваннями.
     */
    private void createButtons() {
        buttonCansel = new JButton("Cansel");
        buttonOk = new JButton("Ok");
        buttonOk.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(final ActionEvent e) {
                ArrayList<String> row = new ArrayList<String>();
                row.add(fieldName.getText());
                row.add(fieldAmount.getText());
                row.add("" + colorChooser.getColor().getRGB());
                ((FileDesignModel) currentFile).setRow(row);
                ((FileDesignModel) currentFile).add();
                frame.dispose();
            }
        });
        buttonCansel.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(final ActionEvent e) {
                frame.dispose();
            }
        });
        buttonReset = new JButton("Reset");
        buttonReset.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(final ActionEvent e) {
                fieldAmount.setText(null);
                fieldName.setText(null);
            }
        });
    }
    /**
     * Метод, що ініціалізує текстові поля призначенні
     * для введення параметрів.
     */
    private void createFields() {
        fieldName = new JTextField(20);
        fieldAmount = new JTextField(20);
    }
    /**
     * Метод, що створює новий діалог додавання поля в діаграму.
     * @param fileDesignModel - модель файлу, в який додається
     * нове поле діаграми.
     */
    public static void createPie(final Observable fileDesignModel) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new AddPieGUI(fileDesignModel);
            }
        });
    }
}
```

A.4 CSVParseException

```
package com.lab111.work;
/**
 * Клас що репрезентує виключні ситуації,
 * що можуть статися під час роботи з CSV файлом.
 * @author [redacted]
 *
 */
public class CSVParseException extends Exception {

    /**
     * Конструктор класу CSVParseException.
     */
    public CSVParseException() {
        super();
```

```
}
/**
 * Конструктор класу CSVParseException.
 * @param message - повідомлення, що супроводжує виключення.
 */
public CSVParseException(final String message) {
    super(message);
}
@Override
public final String toString() {
    return "SCVParseException";
}
}
```

A.5 CSVProcessor

```
package com.lab111.work;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.StringTokenizer;
/**
 * Клас, що реалізує методи для роботи з CSV файлом.
 * @author [redacted]
 *
 */
public class CSVProcessor implements Serializable, Runnable {

/**
 * Масив-список в якому зберігаються рядки CSV файлу.
 */
private ArrayList<String> list;
/**
 * Об'єкт, що зберігає вміст CSV файлу в двовимірному масиві String[][].
 */
private Properties properties;
/**
 * Конструктор класу CSVProcessor.
 */
public CSVProcessor() {
    list = new ArrayList<String>();
}
/**
 * Метод, що повертає масив-список рядків CSV файлу.
 * @return list.
 */
public final ArrayList<String> getList() {
    return (ArrayList<String>) this.list.clone();
}
/**
 * Метод, що зчитує рядки з CSV файлу в поле list класу CSVProcessor.
 * @param br - буферизований потік зчитування файлу.
 * @return - true у випадку вдалого зчитування, інакше - false.
 */
public final boolean readCSVFile(final BufferedReader br) {
    String str = "";
    try {
        while ((str = br.readLine()) != null) {
            this.list.add(str);
        }
    } catch (IOException ex) {
        ex.printStackTrace();
    } finally {
        try {
            br.close();
        } catch (IOException e) {
            e.printStackTrace();
            return false;
        }
    }
    return true;
}
/**
```

```
 * Метод, що записує рядки в файл з поля list класу CSVProcessor.
 * @param bw - буферизований потік запису в потрібний файл.
 * @return - true, якщо запис виконано вдало, інакше - false.
 */
public final boolean writeCSVFile(final BufferedWriter bw) {
    for (int i = 0; i < this.list.size(); i++) {
        try {
            bw.write((String) this.list.get(i) + "\n");
        } catch (IOException e) {
            e.printStackTrace();
            return false;
        } finally {
            try {
                bw.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    return true;
}
/**
 * Метод для запису вмісту поля list класу CSVProcessor
 * в файл в серіалізованому вигляді.
 * @param out - потік запису в файл.
 * @throws IOException - якщо сталася виключна ситуація запису.
 */
private void writeObject(final ObjectOutputStream out)
throws IOException {
    out.writeObject(this.list);
}
/**
 * Метод, що зчитує серіалізований файл в поле list класу CSVProcessor.
 * @param in - потік зчитування файлу.
 * @throws IOException - якщо відбулася виключна ситуація
 * стосовно зчитування.
 * @throws ClassNotFoundException - якщо зчитаний об'єкт з файлу
 * не належить до жодного відомого класу.
 */
@SuppressWarnings("unchecked")
private void readObject(final ObjectInputStream in) throws IOException,
    ClassNotFoundException {
    this.list = (ArrayList<String>) in.readObject();
}
/**
 * Метод, що виводить вміст поля list класу CSVProcessor.
 */
public final void print() {
    for (int i = 0; i < this.getList().size(); i++) {
        System.out.println(getList().get(i));
    }
}
/**
 * Метод, що виділяє з рядків list класу CSVProcessor поля.
 */
public final synchronized void parse() {
    ArrayList<String> buffer = new ArrayList<String>();
    StringTokenizer tok;
    if (this.list.size() != 0) {
        tok = new StringTokenizer(
                (String) this.list.get(0), ";");
        while (tok.hasMoreTokens()) {
            buffer.add(tok.nextToken());
        }
    }
    properties = new Properties(0, 3);
    for (int i = 0; i < list.size(); i++) {
        tok = new StringTokenizer(
                (String) this.list.get(i), ";");
        buffer = new ArrayList<String>();
        while (tok.hasMoreTokens()) {
            buffer.add(tok.nextToken());
        }
        try {
            properties.addRow(buffer);
        } catch (CSVParseException e) {
            e.printStackTrace();
```

```
            }
        }
    }
    /**
     * Метод, що виводить двовимірний масив String[][].
     */
    public final void printMatrix() {
        properties.print();
    }
    @Override
    public final void run() {
        parse();
    }
    /**
     * Метод, що повертає об’єкт {@link Properties}.
     * @return поле properties.
     */
    public final Properties getProperties() {
        return properties;
    }

}
```

A.6 Design

```
package com.lab111.work;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.KeyEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.File;
import java.util.ArrayList;
import java.util.Observable;
import java.util.Observer;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JTabbedPane;
import javax.swing.JToolBar;
/**
 * Клас, що відображає представлення програми,
 * її вигляд. Виступає в ролі представлення
 * в шаблоні MVC.
 * @author [redacted]
 *
 */
public class Design implements Observer {
    /**
     * Головне вікно програми.
     */
    private JFrame frame = new JFrame("Test...");
    /**
     * Панель меню програми.
     */
    private JMenuBar menuBar = new JMenuBar();
    /**
     * Підменю, що пов’язане з роботою з файлами.
     */
    private JMenu menuFile = new JMenu("File");
    /**
     * Підменю, що пов’язане з редагуванням файлів.
     */
    private JMenu menuEdit = new JMenu("Edit");
    /**
     * Підменю для роботи з програмою.
     */
    private JMenu menuProgram = new JMenu("Program");
```

```
/**
 * Пункт меню menuFile, що створює новий файл.
 */
private JMenuItem itemNew;
/**
 * Пункт меню menuFile, що відкриває файл.
 */
private JMenuItem itemOpen;
/**
 * Пункт меню menuFile, що зберігає файл.
 */
private JMenuItem itemSave;
/**
 * Пункт меню menuFile, що зберігає файл у новому файлі.
 */
private JMenuItem itemSaveAs;
/**
 * Пункт меню menuFile, що зберіє зображення діаграми.
 */
private JMenuItem itemSaveGraphics;
/**
 * Пункт меню menuFile, що закриває файл.
 */
private JMenuItem itemCloseFile;
/**
 * Пункт меню menuEdit, що додає нове поле в діаграму.
 */
private JMenuItem itemAdd;
/**
 * Пункт меню menuEdit, що видаляє всі поля з діаграми.
 */
private JMenuItem itemClear;
/**
 * Елемент вибору редагування таблиці.
 */
private JCheckBox checkBoxEditableTable;
/**
 * Пункт меню menuProgram, що показує вікно з інформацією про програму.
 */
private JMenuItem itemAbout;
/**
 * Пункт меню menuEdit, що закриває програму.
 */
private JMenuItem itemExit;
/**
 * Панель інструментів для роботи з файлами.
 */
private JToolBar toolBar;
/**
 * Кнопка створення нового файлу.
 */
private JButton buttonNew;
/**
 * Кнопка відкриття збереженого файлу.
 */
private JButton buttonOpen;
/**
 * Кнопка збереження поточного файлу.
 */
private JButton buttonSave;
/**
 * Кнопка збереження поточного файлу в новому файлі.
 */
private JButton buttonSaveAs;
/**
 * Кнопка збереження зображення діаграми поточного файлу.
 */
private JButton buttonSaveGraphics;
/**
 * Кнопка додавання поля в діаграму.
 */
private JButton buttonAdd;
/**
 * Кнопка видалення всіх полів з діаграми.
 */
private JButton buttonClear;
/**
```

```
 * Модель даного представлення.
 */
private DesignModel designModel;
/**
 * Контролер для даного представлення.
 */
private AbstractGUIController designController;
/**
 * Кнопка закриття поточного файлу.
 */
private JButton buttonCloseFile;
/**
 * Внутрішній слухач подій, призначений для реакції
 * програми на дії користувача.
 */
private InListener actionListener = new InListener();
/**
 * Масив-список дизайну файлів, що відкриті в програмі.
 */
private ArrayList<FileDesign> fileDesigns = new ArrayList<FileDesign>();
/**
 * Панель вкладок для файлів.
 */
private JTabbedPane tabbedPane;
/**
 * Конструктор класу.
 * @param model - модель для даного класу.
 */
public Design(final DesignModel model) {
    ImageIcon icon = new ImageIcon("images/diagram.png");
    frame.setIconImage(icon.getImage());
    designModel = model;
    model.addObserver(this);
    designController = new DesignController(model);
    frame.setSize(new Dimension(800,  600));
    init();
}
/**
 * Ініціалізація і відображення головного вікна.
 */
private void init() {
    frame.setLayout(new BorderLayout());
    createButtons();
    createMenuItems();
    createMenus();
    createMenuBar();
    createToolBar();
    tabbedPane = new JTabbedPane();
        frame.add(tabbedPane, BorderLayout.CENTER);
        frame.add(toolBar, BorderLayout.NORTH);
    frame.setJMenuBar(menuBar);
    frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    frame.addWindowListener(actionListener);
    frame.pack();
    frame.setMinimumSize(new Dimension(frame.getWidth(),
            frame.getHeight()));
    frame.setVisible(true);

}
/**
 * Метод, що ініціалізує меню.
 */
private void createMenus() {
    menuFile.setMnemonic('F');
    menuFile.add(itemNew);
    menuFile.add(itemOpen);
    menuFile.addSeparator();
    menuFile.add(itemSave);
    menuFile.add(itemSaveAs);
    menuFile.add(itemSaveGraphics);
    menuFile.addSeparator();
    menuFile.add(itemCloseFile);
    //меню редагування
    menuEdit.setMnemonic('E');
    menuEdit.add(itemAdd);
    menuEdit.add(itemClear);
    menuEdit.addSeparator();
```

```
    menuEdit.add(checkBoxEditableTable);
    //меню program
    menuProgram.setMnemonic('P');
    menuProgram.add(itemAbout);
    menuProgram.addSeparator();
    menuProgram.add(itemExit);
}
/**
 * Метод, що ініціалізує пункти меню і налаштовує їх.
 */
private void createMenuItems() {
    //------menu File----------
    itemNew = new JMenuItem("  New...", KeyEvent.VK_N);
    itemNew.setIcon(new ImageIcon("images/new_16.png"));
    itemNew.addActionListener(actionListener);
    itemOpen = new JMenuItem("  Open...", KeyEvent.VK_O);
    itemOpen.setIcon(new ImageIcon("images/open_16.png"));
    itemOpen.addActionListener(actionListener);
    itemSave = new JMenuItem("  Save", KeyEvent.VK_S);
    itemSave.setIcon(new ImageIcon("images/save_16.png"));
    itemSave.addActionListener(actionListener);
    itemSave.setEnabled(false);
    itemSaveAs = new JMenuItem("  Save As...", KeyEvent.VK_V);
    itemSaveAs.setIcon(new ImageIcon("images/save_as_16.png"));
    itemSaveAs.addActionListener(actionListener);
    itemSaveAs.setEnabled(false);
    itemSaveGraphics = new JMenuItem("  Export image...",
           KeyEvent.VK_E);
    itemSaveGraphics.setIcon(new ImageIcon(
           "images/picture_16.png"));
    itemSaveGraphics.addActionListener(actionListener);
    itemSaveGraphics.setEnabled(false);
    itemCloseFile = new JMenuItem("  Close File", KeyEvent.VK_C);
    itemCloseFile.setIcon(new ImageIcon("images/close_16.png"));
    itemCloseFile.addActionListener(actionListener);
    itemCloseFile.setEnabled(false);
    //------menu File ENDS----------
    //------menu Edit----------
    itemAdd = new JMenuItem("Add...", KeyEvent.VK_A);
    itemAdd.setIcon(new ImageIcon("images/add_16.png"));
    itemAdd.addActionListener(actionListener);
    itemAdd.setEnabled(false);

    itemClear = new JMenuItem("Clear", KeyEvent.VK_L);
    itemClear.setIcon(new ImageIcon("images/clear_16.png"));
    itemClear.addActionListener(actionListener);
    itemClear.setEnabled(false);

    checkBoxEditableTable = new JCheckBox("Edit Table");
    checkBoxEditableTable.addItemListener(actionListener);
    checkBoxEditableTable.setEnabled(false);
    //------menu Edit ENDS----------

    //------menu Program----------
    itemAbout = new JMenuItem("  About", KeyEvent.VK_W);
    itemAbout.setIcon(new ImageIcon("images/information_16.png"));
    itemAbout.addActionListener(actionListener);

    itemExit = new JMenuItem("  Exit", KeyEvent.VK_X);
    itemExit.setIcon(new ImageIcon("images/log_out_16.png"));
    itemExit.addActionListener(actionListener);
    //------menu Program ENDS----------
}
/**
 * Метод для ініціалізації кнопок і їх налаштування.
 */
private void createButtons() {
    buttonNew = new JButton();
    buttonNew.setIcon(new ImageIcon("images/new_24.png"));
    buttonNew.setToolTipText("Create new file (Alt+N)");
    buttonNew.setMnemonic('N');
    buttonNew.addActionListener(actionListener);

    buttonOpen = new JButton();
    buttonOpen.setIcon(new ImageIcon("images/open_24.png"));
    buttonOpen.setToolTipText("Open saved file (Alt+O)");
    buttonOpen.setMnemonic('O');
```

```
    buttonOpen.addActionListener(actionListener);

    buttonSave = new JButton();
    buttonSave.setIcon(new ImageIcon("images/save_24.png"));
    buttonSave.setToolTipText("Save current file (Alt+S)");
    buttonSave.setMnemonic('S');
    buttonSave.setEnabled(false);
    buttonSave.addActionListener(actionListener);

    buttonSaveAs = new JButton();
    buttonSaveAs.setIcon(new ImageIcon("images/save_as_24.png"));
    buttonSaveAs.setToolTipText("Save current file as... (Alt+E)");
    buttonSaveAs.setMnemonic('E');
    buttonSaveAs.setEnabled(false);
    buttonSaveAs.addActionListener(actionListener);

    buttonSaveGraphics = new JButton();
    buttonSaveGraphics.setIcon(new ImageIcon(
           "images/picture_24.png"));
    buttonSaveGraphics.setToolTipText("Export to image (Alt+G)");
    buttonSaveGraphics.setMnemonic('G');
    buttonSaveGraphics.setEnabled(false);
    buttonSaveGraphics.addActionListener(actionListener);

    buttonCloseFile = new JButton();
    buttonCloseFile.setIcon(new ImageIcon("images/close_24.png"));
    buttonCloseFile.setToolTipText("Close current file (Alt+C)");
    buttonCloseFile.setMnemonic('C');
    buttonCloseFile.setEnabled(false);
    buttonCloseFile.addActionListener(actionListener);

    buttonAdd = new JButton();
    buttonAdd.setIcon(new ImageIcon("images/add_24.png"));
    buttonAdd.setToolTipText("Adds pie to the piechart (Alt+A)");
    buttonAdd.setMnemonic('A');
    buttonAdd.setEnabled(false);
    buttonAdd.addActionListener(actionListener);

    buttonClear = new JButton();
    buttonClear.setIcon(new ImageIcon("images/clear_24.png"));
    buttonClear.setToolTipText("Cleares the piechart (Alt+L)");
    buttonClear.setMnemonic('L');
    buttonClear.setEnabled(false);
    buttonClear.addActionListener(actionListener);
}
/**
 * Метод для ініціалізації меню.
 */
private void createMenuBar() {
    menuBar.add(menuFile);
    menuBar.add(menuEdit);
    menuBar.add(menuProgram);
}
/**
 * Метод для ініціалізації і налаштування панелі інструментів.
 */
private void createToolBar() {
    this.toolBar = new JToolBar("File Operations...");
    toolBar.add(buttonNew);
    toolBar.add(buttonOpen);
    toolBar.add(buttonSave);
    toolBar.add(buttonSaveAs);
    toolBar.add(buttonSaveGraphics);
    toolBar.add(buttonCloseFile);
    toolBar.addSeparator(new Dimension(20,  20));
    toolBar.add(buttonAdd);
    toolBar.add(buttonClear);
}
@Override
public void update(final Observable obs, final Object obj) {
    buttonCloseFile.setEnabled(((DesignModel) obs).isClose());
    buttonSave.setEnabled(((DesignModel) obs).isSave());
    buttonSaveGraphics.setEnabled(((DesignModel) obs).
           isExportImage());
    buttonSaveAs.setEnabled(((DesignModel) obs).isExportImage());
    itemCloseFile.setEnabled(((DesignModel) obs).isClose());
    itemSave.setEnabled(((DesignModel) obs).isSave());
```

```
    itemSaveGraphics.setEnabled(((DesignModel) obs).
            isExportImage());
    itemSaveAs.setEnabled(((DesignModel) obs).isExportImage());
    buttonAdd.setEnabled(buttonCloseFile.isEnabled());
    buttonClear.setEnabled(buttonCloseFile.isEnabled());
    itemAdd.setEnabled(itemCloseFile.isEnabled());
    itemClear.setEnabled(itemCloseFile.isEnabled());
    checkBoxEditableTable.setEnabled(itemClear.isEnabled());
    if (obj != null) {
        if (fileDesigns.size() > ((ArrayList<FileDesign>) obj).
                size()) {
            FileDesign fd = null;
            for (int i = 0; i
            < fileDesigns.size() - 1; i++) {
                if (fileDesigns.get(i) != ((ArrayList
                        <FileDesign>) obj).
                        get(i)) {
                    fd = fileDesigns.get(i);
                    break;
                }
            }
            if (fd == null) {
                fd = fileDesigns.get(fileDesigns.
                        size() - 1);
            }
            tabbedPane.remove(fd);
            fileDesigns.remove(fd);
        } else {
            if (fileDesigns.size()
                    < ((ArrayList<FileDesign>) obj).
                    size()) {
                fileDesigns.
                add(((ArrayList<FileDesign>) obj).
                    get(fileDesigns.size()));
                if (((ArrayList<FileDesign>) obj).
                    get(fileDesigns.size() - 1).
                    getModel().getPath() == null) {
                    tabbedPane.add(fileDesigns.
                        get(fileDesigns.size() - 1),
                            "Untitled"+designModel.getAmount());
                } else {
                    File f = new File(fileDesigns.get(fileDesigns.
                            size()-1).getModel().getPath());
                    tabbedPane.add(fileDesigns.
                            get(fileDesigns.size() - 1),
                            f.getName());
                }
            }
        }
    } else {
        for (int i = 0; i < fileDesigns.size(); i++) {
            FileDesignModel fdm =
                fileDesigns.get(i).getModel();
            if (fdm.getPath() != null) {
                File f = new File(fdm.getPath());
                tabbedPane.setTitleAt(i, f.getName());
            }
        }
    }
}
/**
 * Внутрішній слухач подій для головного вікна.
 * @author rebelizant Гула Вадим.
 *
 */
class InListener extends WindowAdapter
implements ActionListener, ItemListener {
    @Override
    public void windowClosing(final WindowEvent e) {
        designController.exit();
        return;
    }
    @Override
    public void actionPerformed(final ActionEvent event) {
        Object source = event.getSource();
        if (source == itemNew || source == buttonNew) {
            designController.newFile();
```

```
            return;
        }
        if (source == itemOpen || source == buttonOpen) {
            designController.openFile();
            return;
        }
        if (source == itemSave || source == buttonSave) {
            designController.saveFile(tabbedPane.
                    getSelectedIndex());
            return;
        }
        if (source == itemSaveGraphics
                || source == buttonSaveGraphics) {
            designController.exportImage(tabbedPane.
                    getSelectedIndex());
            return;
        }
        if (source == itemCloseFile
                || source == buttonCloseFile) {
            designController.closeFile(tabbedPane.
                    getSelectedIndex());
            return;
        }
        if (source == itemAbout) {
            designController.about();
            return;
        }
        if (source == itemExit) {
            designController.exit();
            return;
        }
        if (source == itemSaveAs ||  source == buttonSaveAs) {
            designController.saveFileAs(tabbedPane.
                    getSelectedIndex());
            return;
        }
        if (source == itemAdd || source == buttonAdd) {
            designController.add(tabbedPane.
                    getSelectedIndex());
            return;
        }
        if (source == itemClear || source == buttonClear) {
            designController.clear(tabbedPane.
                    getSelectedIndex());
            return;
        }
    }

    @Override
    public void itemStateChanged(final ItemEvent arg0) {
        boolean flag = false;
        if (arg0.getStateChange() == ItemEvent.SELECTED) {
            flag = true;
        }
        designController.setEditableTable(tabbedPane.
                getSelectedIndex(),
                flag);
    }
 }
}
```

A.7 DesignController

```
package com.lab111.work;

import java.io.File;

import javax.swing.ImageIcon;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import javax.swing.SwingUtilities;
import javax.swing.filechooser.FileFilter;

/**
 * Клас - контролер для головного вікна.
 * @author [redacted]
 *
 */
```

```
public class DesignController implements AbstractGUIController {
 /**
  * Модель головного вікна.
  */
 private DesignModel model;
 /**
  * Конструктор класу.
  * @param model - модель класу-представлення.
  */
 public DesignController(final DesignModel model) {
     this.model = model;
 }
 @Override
 public void saveFile(final int idx) {
     if (model.getFileDesigns().get(idx).getModel().
             isSaveInFact() == true) {
         File file = new File(model.getFileDesigns().
                 get(idx).getModel().getPath());
         String str = file.getPath();
         str = str.substring(0, str.indexOf('.'));
         model.saveFile(idx, str);
     } else {
         JFileChooser jfc = new JFileChooser();
         jfc.setFileFilter(new FileFilter() {
             @Override
             public String getDescription() {
                 return "*.csv";
             }
             @Override
             public boolean accept(final File file) {
                 if (file.getName().endsWith(".csv")) {
                     return true;
                 }
                 if (file.isDirectory()) {
                     return true;
                 }
                 return false;
             }
         });
         int choise = jfc.showSaveDialog(null);
         if (choise == JFileChooser.APPROVE_OPTION) {
             File file = jfc.getSelectedFile();
             model.saveFile(idx, file.getPath());
         }
     }
 }

 @Override
 public final void saveFileAs(final int idx) {
     JFileChooser jfc = new JFileChooser();
     jfc.setFileFilter(new FileFilter() {

         @Override
         public String getDescription() {
             return "*.csv";
         }

         @Override
         public boolean accept(final File file) {
             if (file.getName().endsWith(".csv")) {
                 return true;
             }
             if (file.isDirectory()) {
                 return true;
             }
             return false;
         }
     });
     int choise = jfc.showSaveDialog(null);
     if (choise == JFileChooser.APPROVE_OPTION) {
         File file = jfc.getSelectedFile();
         model.saveFile(idx, file.getPath());
     }
 }
 @Override
 public final void newFile() {
     model.newFile();
```

```
}
@Override
public final void openFile() {
    JFileChooser jfc = new JFileChooser();
    jfc.setFileFilter(new FileFilter() {

        @Override
        public String getDescription() {
            return "*.csv";
        }

        @Override
        public boolean accept(final File file) {
            if (file.getName().endsWith(".csv")) {
                return true;
            }
            if (file.isDirectory()) {
                return true;
            }
            return false;
        }
    });
    int choose = jfc.showOpenDialog(null);
    if (choose == JFileChooser.CANCEL_OPTION) {
        return;
    }
    String str = jfc.getSelectedFile().getName();
    int n = str.length() - 3;
    str = str.substring(n, str.length());
    if (str.equals("csv")) {
        if (choose == JFileChooser.APPROVE_OPTION) {
            model.openFile(jfc.getSelectedFile());
        }
    } else {
        JOptionPane.showMessageDialog(null,
                "Illegal file format."
                 + "\nPlease, try again.",
                "Error", JOptionPane.ERROR_MESSAGE);
    }
}
@Override
public final void exportImage(final int idx) {
    JFileChooser jfc = new JFileChooser();
    jfc.setFileFilter(new FileFilter() {
        @Override
        public String getDescription() {
            return "*.jpeg";
        }

        @Override
        public boolean accept(final File file) {
            if (file.getName().endsWith(".jpeg")) {
                return true;
            }
            if (file.isDirectory()) {
                return true;
            }
            return false;
        }
    });
    int choise = jfc.showSaveDialog(null);
    if (choise == JFileChooser.APPROVE_OPTION) {
        File file = jfc.getSelectedFile();
        model.exportImage(idx, file.getPath());
    }
}
@Override
public final void closeFile(final int idx) {
    model.closeFile(idx);
}
@Override
public final void about() {
    JOptionPane.showMessageDialog(null,
            "<html>This is cource work"
            + " made by<br> student of KPI"
            + " <i><FONT size=4 color=\"red\">Gula Vadym!"
            + "<FONT></i><br><br>e-mail:"
```

```
            + "<u><FONT color"
            + "=\"green\"> braingvs@gmail.com ",
            "About", JOptionPane.INFORMATION_MESSAGE,
            new ImageIcon("images/information_48.png"));
    }
    @Override
    public final void exit() {
        model.exit();
    }
    @Override
    public final void add(final int idx) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                AddPieGUI.createPie(
                        model.getFileDesigns().
                        get(idx).getModel());
            }
        });

    }
    @Override
    public final void clear(final int idx) {
        model.clear(idx);
    }
    @Override
    public final void setEditableTable(final int idx, final boolean flag) {
        model.setEditableTable(idx, flag);
    }
}
```

A.8 DesignModel

```
package com.lab111.work;

import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Observable;
import java.util.Observer;

import javax.imageio.ImageIO;
import javax.swing.JOptionPane;
/**
 * Клас-модель для головного вікна. Зберігає стан головного вікна.
 * @author [redacted]
 *
 */
public class DesignModel extends Observable {
    /**
     * Кількість створених вкладок.
     */
    private int amount = 0;
    /**
     * Прапорець, що відображає можливість збереження.
     */
    private boolean isSave = false;
    /**
     * Прапорець, що відображає можливість закриття.
     */
    private boolean isClose = false;
    /**
     * Прапорець, що відображає можливість збереження зображення.
     */
    private boolean isExportImage = false;
    /**
     * Прапорець, що відображає можливість збереження в новому файлі.
     */
    private boolean isSaveAs = false;
    /**
     * Масив-список вигляду файлів.
     */
```

```
private ArrayList<FileDesign> fileDesigns =
    new ArrayList<FileDesign>();
/**
 * Масив-список спостерігачів за моделю.
 */
private ArrayList<Observer> observers;
/**
 * Конструктор класу.
 * @param observer - спостерігач за моделю.
 */
public DesignModel(final Observer observer) {
    observers = new ArrayList<Observer>();
    addObserver(observer);
}
/**
 * Конструктор класу.
 */
public DesignModel() {
    observers = new ArrayList<Observer>();
}
/**
 * Метод, що повертає можливість зберігання.
 * @return - поле isSave.
 */
public boolean isSave() {
    return isSave;
}
/**
 * Метод, що змінює можливість зберігання.
 * @param newIsSave - новий стан можливості зберігання.
 */
public final void setSave(final boolean newIsSave) {
    this.isSave = newIsSave;
}
/**
 * Метод, що повертає можливість закривання.
 * @return - поле isClose.
 */
public final boolean isClose() {
    return isClose;
}
/**
 * Метод, що змінює можливість закривання.
 * @param newIsClose - новий стан можливості закривання.
 */
public final void setClose(final boolean newIsClose) {
    this.isClose = newIsClose;
}
/**
 * Метод, що повертає можливість збереження зображення.
 * @return - поле isExportImage.
 */
public final boolean isExportImage() {
    return isExportImage;
}
/**
 * Метод, що змінює можливість збереження зображення.
 * @param newIsExportImage - новий стан можливості
 * збереження зображення.
 */
public final void setExportImage(final boolean newIsExportImage) {
    this.isExportImage = newIsExportImage;
}
@Override
public final void addObserver(final Observer o) {
    observers.add(o);
}
@Override
public final void notifyObservers() {
    for (int i = 0; i < observers.size(); i++) {
        observers.get(i).update(this, null);
    }
}
/**
 * Метод, що повертає кількість створених файлів в даній сесії.
 * @return - поле amount.
 */
```

```
public final int getAmount() {
    return amount;
}
/**
 * Метод для зміни кількості створених файлів.
 * @param newAmount - нова кількість створених файлів.
 */
public final void setAmount(int newAmount) {
    this.amount = newAmount;
}
/**
 * Метод для створення нового файлу.
 */
public final void newFile() {
    setAmount(getAmount() + 1);
    Properties prop = new Properties(0, 3);
    FileDesignModel fdm = new FileDesignModel(prop);
    FileDesign fd = new FileDesign(prop, fdm);
    fileDesigns.add(fd);
    fdm.addObserver(fd);
    setFlagsTrue();
    notifyObservers(fileDesigns);
}
@Override
public final void notifyObservers(final Object obj) {
    for (int i = 0; i < observers.size(); i++) {
        observers.get(i).update(this, obj);
    }
}
/**
 * Метод для збереження файлу.
 * @param idx - індекс вкладки в панелі вкладок.
 * @param filename - ім'я файлу.
 */
public final void saveFile( final int idx, String filename) {
    try {
        filename += ".csv";
        File file = new File(filename);
        file.createNewFile();
        Properties pr = fileDesigns.get(idx).getProperties();
        FileWriter fw = new FileWriter(file);
        BufferedWriter bw = new BufferedWriter(fw);
        for (int i = 0; i < pr.getRowCount(); i++) {
            for (int j = 0; j < pr.getColCount(); j++) {
                bw.write(pr.getCell(i, j));
                bw.write(";");
            }
            bw.write("\n");
        }
        bw.flush();
        bw.close();
        getFileDesigns().get(idx).getModel().
        setPath(file.getPath());
        getFileDesigns().get(idx).getModel().
        setSaveInFact(true);
        getFileDesigns().get(idx).getModel().setSaved(true);
        notifyObservers();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
/**
 * Метод для відкриття збереженого файлу.
 * @param file - шлях до файлу.
 */
public final void openFile(final File file) {
    setAmount(getAmount() + 1);
    setSave(true);
    setClose(true);
    setExportImage(true);
    CSVProcessor csvp = new CSVProcessor();
    FileReader fr;
    Thread t = new Thread(csvp);
    try {
        fr = new FileReader(file);
        BufferedReader br1 = new BufferedReader(fr);
        csvp.readCSVFile(br1);
```

```
            t.start();
            try {
                t.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        FileDesignModel fdm = new FileDesignModel(csvp.getProperties());
        FileDesign fd = new FileDesign(csvp.getProperties(), fdm);
        fileDesigns.add(fd);
        fdm.addObserver(fd);
        setFlagsTrue();
        fdm.setPath(file.getPath());
        fdm.setSaveInFact(true);
        fdm.setSaved(true);
        notifyObservers(fileDesigns);
    }
    /**
     * Метод для збереження зображення.
     * @param idx - індекс вкладки файла в панелі вкладок.
     * @param filename - шлях до зображення.
     */
    public final void exportImage(final int idx, String filename) {
        FileDesign fd = fileDesigns.get(idx);
        BufferedImage image = fd.createImage();
        filename+=".jpeg";
        File file = new File(filename);
        try {
            ImageIO.write(image, "jpeg", file);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    /**
     * Метод, що встановлює мітки в false.
     */
    private void setFlagsFalse() {
        this.isClose = false;
        this.isExportImage = false;
        this.isSave = false;
    }
    /**
     * Метод, що встановлює мітки в true.
     */
    private void setFlagsTrue() {
        this.isClose = true;
        this.isExportImage = true;
        this.isSave = true;
        this.isSaveAs = true;
    }
    /**
     * Метод для закриття файлу.
     * @param idx - індекс файлу в панелі вкладок.
     */
    public final void closeFile(final int idx) {
        int choose = 0;
        if (!fileDesigns.get(idx).getModel().isSaved()) {
            choose = JOptionPane.showConfirmDialog(null,
                    "<html>File "
                    + " is not saved!<br>"
                    + "Are you sure?");
        }
        if (choose == JOptionPane.YES_OPTION) {
            fileDesigns.set(idx, null);
            fileDesigns.remove(idx);
            if (fileDesigns.size() == 0) {
                setFlagsFalse();
            }
            notifyObservers(fileDesigns);
        }
    }
    /**
     * Метод для додавання поля в діаграму.
     * @param idx - індекс файлу в панелі вкладок.
     */
```

```
 public final void add(final int idx) {
     fileDesigns.get(idx).getModel().add();
 }
 /**
  * Метод для видалення всіх полів з діаграми.
  * @param idx - індекс файлу в панелі вкладок.
  */
 public final void clear(final int idx) {
     fileDesigns.get(idx).getModel().clear();
 }
 /**
  * Метод для виходу з програми.
  */
 public final void exit() {
     int choose = 0;
     if (fileDesigns.size() > 0) {
         choose = JOptionPane.showConfirmDialog(null,
                 "<html>Some files "
                 + " may be not saved!<br>"
                 + "Are you sure?");
         if (choose == JOptionPane.YES_OPTION) {
             System.exit(0);
         }
     } else {
         System.exit(0);
     }
 }
 /**
  * Метод для зміни можливості редагування таблиці.
  * @param idx - індекс файлу в панелі вкладок.
  * @param flag - новий стан можливості редагування таблиці..
  */
 public final void setEditableTable(final int idx, final boolean flag) {
     fileDesigns.get(idx).getModel().setEditableTable(flag);
 }
 /**
  * Метод, що повертає масив-список FileDesign.
  * @return - поле fileDesigns.
  */
 public final ArrayList<FileDesign> getFileDesigns() {
     return this.fileDesigns;
 }

}
```

A.9 Drawer

```
package com.lab111.work;

import java.awt.Graphics;
import java.awt.image.BufferedImage;
/**
 * Інтерфейс для діаграм.
 * @author [redacted]
 *
 */
public interface Drawer {
 /**
  * Метод для малювання діаграми.
  * @param g - контекст малювання.
  */
 void draw(Graphics g);
 /**
  * Метод для зміни поля кламу Properties.
  * @param p - об'єкт класу {@link Properties}.
  */
 void setProperties(Properties p);
 /**
  * Метод для створення зображення.
  * @return - нове зображення.
  */
 BufferedImage createImage();
}
```

A.10 FileDesign

```
package com.lab111.work;
import java.awt.BorderLayout;
```

```
import java.awt.Component;
import java.awt.Dimension;
import java.awt.image.BufferedImage;
import java.util.Observable;
import java.util.Observer;

import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;
import javax.swing.JTable;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
import javax.swing.table.TableColumn;
/**
 * Клас, що являє собою вигляд csv файлу.
 * @author [redacted]
 */
public class FileDesign extends JPanel implements Observer {
 /**
  * Таблиця для відображення csv файлу.
  */
 private JTable table;
 /**
  * Компонент для прокручування таблиці.
  */
 private JScrollPane scrollPane;
 /**
  * Об’єкт для відображення csv файлу в вигляді діаграми.
  */
 private Drawer drawer;
 /**
  * Вміст csv файлу.
  */
 private Properties properties;
 /**
  * Модель для представлення файлу.
  */
 private FileDesignModel model;
 /**
  * Спливаюче меню.
  */
 private JPopupMenu popupMenu;
 /**
  * Пункт видалення поля з діаграми.
  */
 private JMenuItem itemDelete;
 /**
  * Контролер представлення файлу.
  */
 private AbstractFDController controller;
 /**
  * Панель розколу.
  */
 private JSplitPane jsp;
 /**
  * Конструктор класу.
  * @param p - властивості csv файлу.
  * @param model - модель файлу.
  */
 public FileDesign(Properties p, FileDesignModel model) {
     this.model = model;
     setProperties(p);
     this.setSize(new Dimension(800,  600));
     this.drawer = new SectorDiagramDrawer(getWidth() / 2,
             getHeight() / 2, p);
     controller = new FileDesignController(model);
     ((SectorDiagramDrawer) drawer).setParent(this);
     init();
 }
 /**
  * Створення GUI.
  */
 private void init() {
     setLayout(new BorderLayout());
     initTable();
```

```
    ((Component) drawer).
    setPreferredSize(new Dimension(getWidth() / 2,
            getHeight() / 2));
    jsp = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, true,
            scrollPane, ((Component) drawer));
        add(jsp);
    setMinimumSize(new Dimension(getWidth(), getHeight()));
    table.getModel().
    addTableModelListener(new TableModelListener() {
        @Override
        public void tableChanged(TableModelEvent e) {
            if (e.getType() == TableModelEvent.UPDATE) {
                String str = (String) table.getModel().
                getValueAt(e.getFirstRow(),
                        e.getColumn());
                properties.setCell(str, e.getFirstRow(),
                        e.getColumn());
                model.setProperties(properties);
            }
        }
    });
    setVisible(true);
}
/**
 * Метод для зміни drawer.
 * @param d - новий drawer.
 */
public void setDrawer(Drawer d) {
    this.drawer = d;
}
/**
 * Метод для створення таблиці.
 */
public void initTable() {
    String[] headings = new String[2];
    headings[0] = "Name";
    headings[1] = "Amount";
    Object[][] obj =
        new Object[properties.getRowCount()]
                  [properties.getColCount()];
    for (int i = 0; i < properties.getRowCount(); i++) {
        for (int j = 0; j < properties.getColCount(); j++) {
            obj[i][j] = properties.getCell(i, j);
        }
    }
    table = new JTable(obj, headings);
    TableColumn t;
    for (int i = 0; i < table.getColumnCount() - 1; i++) {
        t = table.getColumnModel().getColumn(i);
        t.setPreferredWidth(20);
    }
        table.setEnabled(false);
    table.setAutoResizeMode(JTable.AUTO_RESIZE_LAST_COLUMN);
    table.setFillsViewportHeight(true);
    table.setCellSelectionEnabled(true);
    table.getModel().addTableModelListener(
            new TableModelListener() {
        @Override
        public void tableChanged(TableModelEvent e) {
            if (e.getType() == TableModelEvent.UPDATE) {
                String str =
                    (String) table.getModel().
                    getValueAt(e.getFirstRow(),
                            e.getColumn());
                properties.setCell(str, e.getFirstRow(),
                        e.getColumn());
                model.setProperties(properties);
            }
        }
    });
    scrollPane = new JScrollPane(table);
    scrollPane.setPreferredSize(
            new Dimension(getWidth() / 3, getHeight() / 2));
}
/**
 * Метод для зміни properties.
 * @param p - об'єкт класу {@link Properties}.
```

```
     */
    public void setProperties(Properties p) {
        this.properties = p;
    }
    @Override
    public void update(Observable o, Object arg) {
        Properties prop = ((Properties)arg);
        this.properties = prop;
        drawer.setProperties(prop);
        remove(jsp);
        initTable();
        jsp = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                true, scrollPane, ((Component) drawer));
        add(jsp);
        table.setEnabled(model.isEditableTable());
        setVisible(false);
        setVisible(true);
    }
    /**
     * Метод для отримання об'єкту properties.
     * @return - поле properties.
     */
    public Properties getProperties () {
        return this.properties.clone();
    }
    /**
     * Метод для створення зображення.
     * @return - нове зображення.
     */
    public BufferedImage createImage() {
        SectorDiagramDrawer sdd = (SectorDiagramDrawer) drawer;
        return drawer.createImage();
    }
    /**
     * Метод, що повертає контролер даного класу.
     * @return - поле controller.
     */
    public AbstractFDController getController() {
        return this.controller;
    }
    /**
     * Метод, що повертає модель класу.
     * @return - поле model.
     */
    public FileDesignModel getModel() {
        return this.model;
    }
}
```

A.11 FileDesignController

```
package com.lab111.work;

import java.awt.Color;
import java.util.Observable;
import java.util.Observer;

import javax.swing.SwingUtilities;

/**
 * Клас контролер для класу FileDesign. Є підкласом абстрактного класу,
 * AbstractFDController
 * @author [redacted]
 *
 */
public class FileDesignController implements AbstractFDController {
    /**
     * Модель для класу, що контролюється.
     */
    private FileDesignModel model;
    /**
     * Конструктор класу.
     */
    public FileDesignController(FileDesignModel model) {
        this.model = model;
    }
    @Override
    public void add() {
```

```
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                AddPieGUI.createPie(model);
            }
        });
    }
    @Override
    public void clear() {
        model.clear();
    }
    @Override
    public void remove() {
        model.remove();
    }
    @Override
    public void remove(int index) {
        model.remove(index);
    }
    @Override
    public void setAmount(int index, String amount) {
        model.getProperties().setCell(amount, index, 1);
        model.notifyObservers(model.getProperties());
    }
    @Override
    public void setColor(int index, Color color) {
        model.getProperties().setCell("" + color.getRGB(), index, 2);
        model.notifyObservers(model.getProperties());
    }
    @Override
    public void setName(int index, String name) {
        model.getProperties().setCell(name, index, 0);
        model.notifyObservers(model.getProperties());
    }
}
```

A.12 FileDesignModel

```
package com.lab111.work;

import java.util.ArrayList;
import java.util.Observable;
import java.util.Observer;

/**
 * Клас, модель для класу FileDesign.
 * @author [redacted]
 */
public class FileDesignModel extends Observable {
    /**
     * Мітка - чи збережений файл.
     */
    private boolean isSaved = false;
    /**
     * Спостерігачі за даною моделю.
     */
    private ArrayList<Observer> observers;
    /**
     * Рядок, що додається в properties.
     */
    private ArrayList<String> row;
    /**
     * Шлях до файлу.
     */
    private String path = null;
    /**
     * Вміст csv файлу.
     */
    private Properties properties;
    /**
     * Чи збережений файл на диску.
     */
    private boolean isSaveInFact = false;
    /**
     * Чи можна редагувати таблицю.
     */
    private boolean isEditableTable = false;
```

```
/**
 * Метод, що повертає можливість редагування таблиці.
 * @return - поле isEditableTable.
 */
public boolean isEditableTable() {
    return isEditableTable;
}
/**
 * Метод, що встановлює можливість редагування таблиці.
 * @param isEditableTable - новий стан можливості редагування.
 */
public void setEditableTable(boolean isEditableTable) {
    this.isEditableTable = isEditableTable;
    notifyObservers(properties);
}
/**
 * Конструктор класу.
 * @param observer - спостерігач за моделю.
 * @param properties - вміст csv файлу.
 */
public FileDesignModel(Observer observer, Properties properties) {
    this.properties = properties;
    observers = new ArrayList<Observer>();
    addObserver(observer);
}
/**
 * Конструктор класу.
 * @param properties - вміст csv файлу.
 */
public FileDesignModel(Properties properties) {
    this.properties = properties;
    observers = new ArrayList<Observer>();
}
/**
 * Метод, що повертає стан збереження файлу.
 * @return - поле isSaved.
 */
public boolean isSaved() {
    return isSaved;
}
/**
 * Метод, що встановлює стан збереження файлу.
 * @param isSaved - новий стан збереження.
 */
protected void setSaved(boolean isSaved) {
    this.isSaved = isSaved;
}
/**
 * Метод, що змінює row.
 * @param row - новий рядок.
 */
public void setRow(ArrayList<String> row) {
    this.row = row;
}
/**
 * Метод, що повертає row.
 * @return - поле row.
 */
public ArrayList<String> getRow() {
    return this.row;
}
@Override
public void addObserver(Observer o) {
    observers.add(o);
}
@Override
public void notifyObservers() {
    for(int i = 0; i < observers.size(); i++) {
        observers.get(i).update(this, null);
    }
}
@Override
public void notifyObservers(Object obj) {
    for(int i = 0; i < observers.size(); i++) {
        observers.get(i).update(this, obj);
    }
}
```

```
/**
 * Метод для зміни шляху до файлу.
 * @param path - новий шлях.
 */
public void setPath( String path) {
    this.path = path;
}
/**
 * Метод для додавання поля в діаграму.
 */
public void add() {
    try {
        properties.addRow(getRow());
        setSaved(false);
        notifyObservers(properties);
    } catch (CSVParseException e) {
        e.printStackTrace();
    }
}
/**
 * Метод для видалення поля з кінця діаграми.
 */
public void remove() {
    setSaved(false);
    properties.removeRow();
    notifyObservers(properties);
}
/**
 * Метод для видалення поля з діаграми за індексом.
 * @param index - індекс поля, що видаляється.
 */
public void remove(int index) {
    setSaved(false);
    properties.removeRow(index);
    notifyObservers(properties);
}
/**
 * Метод для видалення всіх полів діаграми.
 */
public void clear() {
    properties.clean();
    setSaved(false);
    notifyObservers(properties);
}
/**
 * Метод для отримання properties.
 * @return - поле properties.
 */
public Properties getProperties() {
    return properties;
}
/**
 * Метод для зміни properties.
 * @param properties - новий об’єкт properties.
 */
public void setProperties(Properties properties) {
    this.properties = properties;
    notifyObservers(this.properties);
}
/**
 * Метод, що показує чи файл збережено на диск.
 * @return - true, якщо збережено; інакше - false.
 */
public boolean isSaveInFact() {
    return isSaveInFact;
}
/**
 * Метод, що встановлює чи файл збережено на диск.
 * @param isSaveInFact - новий стан.
 */
public void setSaveInFact(boolean isSaveInFact) {
    this.isSaveInFact = isSaveInFact;
}
/**
 * Метод для отримання шляху до файла.
 * @return - шлях до файлу. Поле path.
 */
```

```
 public String getPath() {
     return path;
 }
}
```

A.13 Pie

```
package com.lab111.work;

import java.awt.geom.Arc2D;

/**
 * Клас що являє поле з секторної діаграми.
 * @author [redacted]
 */
public class Pie extends Arc2D.Double {
    /**
     * Конструктор класу.
     * @param xx - верхня ліва координата.
     * @param yy - верхня права координата.
     * @param w - ширина.
     * @param h - висота.
     * @param startDot - початковий кут.
     * @param size - розмір кута.
     * @param type - тип.
     */
    public Pie(double xx, double yy, double w, double h,
        double startDot, double size, int type) {
        super(xx, yy, w, h, startDot, size, type);
    }

}
```

A.14 Properties

```
package com.lab111.work;

import java.util.ArrayList;

import javax.swing.JOptionPane;
import javax.swing.SwingUtilities;

/**
 * Клас, що репрезентує вміст csv файлу.
 * @author [redacted]
 *
 */
public class Properties {
 /**
  * Матриця, що містить слова з csv файлу.
  */
 private String[][] prop;
 /**
  * Кількість рядків.
  */
 private int rowCount;
 /**
  * Кількість стовпчиків.
  */
 private int colCount = 3;
 /**
  * Поточний рядок.
  */
 private int currentRow;
 /**
  * Конструктор класу.
  * @param rowCount - кількість рядків.
  * @param colCount - кількість стовпчиків.
  */
 public Properties(int rowCount, int colCount) {
     this.rowCount = rowCount;
     this.colCount = colCount;
     prop = new String[this.rowCount][this.colCount];
     currentRow = 0;
 }
 /**
  * Метод для зміни вмісту комірки.
  * @param str - новий вміст комірки.
```

```
     * @param row - рядок.
     * @param col - стовпчик.
     * @return - true, якщо комірку відредаговано вдало.
     */
    public boolean setCell(String str, int row, int col) {
        if (checkError(str, row, col)) {
            return false;
        } else {
            prop[row][col] = str;
            return true;
        }
    }
    /**
     * Метод для додавання рядку в масив.
     * @param newRow - новий рядок.
     * @throws CSVParseException - якщо неправильний формат рядка.
     */
    public void addRow(ArrayList<String> newRow) throws CSVParseException {
        if (checkError(newRow)) {
            SwingUtilities.invokeLater(new Runnable() {

                @Override
                public void run() {
                    JOptionPane.showMessageDialog(null,
                            "Wrong input."
                            + "\nPlease, try again.",
                            "Error", JOptionPane.
                            ERROR_MESSAGE);
                }
            });
        } else {
            incSize();
            for (int i = 0; i < colCount; i++) {
                prop[currentRow][i] = newRow.get(i);
            }
            currentRow++;
        }
    }
    /**
     * Метод для перевірки рядка на відповідність формату.
     * @param list - рядок, що перевіряється.
     * @return - true, якщо рядок відповідає формату, інакше - false.
     */
    public boolean checkError(ArrayList<String> list) {
        try {
            double n = Double.parseDouble(list.get(1));
            int d = Integer.parseInt(list.get(2));
            return false;
        } catch (Exception e) {
            return true;
        }
    }
    /**
     * Метод для перевірки потенційного вмісту комірки.
     * @param str - новий вміст.
     * @param row - рядок.
     * @param col - колонка.
     * @return - true, якщо слово відповідає формату, інакше - false.
     */
    public boolean checkError(String str, int row, int col) {
        if (col == 1 || col == 2) {
            try {
                double d = Double.parseDouble(str);
            } catch (NumberFormatException exc) {
                SwingUtilities.invokeLater(new Runnable() {
                    @Override
                    public void run() {
                        JOptionPane.showMessageDialog(
                                null,
                                "Illegal input."
                                + "\nPlease,"
                                + " try again.",
                                "Error",
                                JOptionPane.
                                ERROR_MESSAGE);
                    }
                });
```

```
            return true;
        }
    }
    if ((row > rowCount - 1) || (col > colCount - 1)) {
        return false;
    } else {
        if ((row  <  0) || (col < 0)) {
            return false;
        }
    }
    if (str == null) {
        return false;
    } else {
        prop[row][col] = str;
        return true;
    }
}
/**
 * Метод, що повертає вміст комірки.
 * @param row - рядок.
 * @param col - колонка.
 * @return - вміст комірки.
 */
public String getCell(int row, int col) {
    String str = null;
    try {
        str = new String(prop[row][col].substring(0));
    return str;
    } catch (ArrayIndexOutOfBoundsException e) {
        e.printStackTrace();
    }
    return str;
}
/**
 * Метод, що повертає масив csv файлу.
 * @return - поле prop клоноване.
 */
public String[][] getProp() {
    return prop.clone();
}
/**
 * Метод для виведення матриці.
 */
public void print() {
    for (int i = 0; i < rowCount; i++) {
        for (int j = 0; j < colCount; j++) {
            System.out.print(prop[i][j] + " ");
        }
        System.out.println();
    }
}
/**
 * Метод для клонування об'єкту {@link Properties}.
 */
public Properties clone() {
    Properties p = new Properties(rowCount, colCount);
    for (int i = 0; i < rowCount; i++) {
        for (int j = 0; j < colCount; j++) {
            p.setCell(this.getCell(i, j), i, j);
        }
    }
    return p;
}
/**
 * Метод для отримання кількості рядків.
 * @return - кількість рядків.
 */
public int getRowCount() {
    return rowCount;
}
/**
 * Метод для отримання кількості колонок.
 * @return - кількість колонок.
 */
public int getColCount() {
    return colCount;
}
```

```
    /**
     * Мето для збільшення кількості рядків на 1.
     */
    public void incSize() {
        String[][] buf =
            new String[this.getRowCount() + 1]
                      [this.getColCount()];
        for (int i = 0; i < getRowCount(); i++) {
            for (int j = 0; j < getColCount(); j++) {
                buf[i][j] = this.prop[i][j];
            }
        }
        this.prop = null;
        this.prop = buf;
        rowCount++;
    }
    /**
     * Метод для видалення останього рядка з матриці.
     */
    public void removeRow() {
        if (this.prop.length > 0) {
            String[][] buf =
                new String[this.getRowCount() - 1]
                          [this.getColCount()];
            for (int i = 0; i < getRowCount() - 1; i++) {
                for (int j = 0; j < getColCount(); j++) {
                    buf[i][j] = this.prop[i][j];
                }
            }
            this.prop = null;
            this.prop = buf;
            rowCount--;
            currentRow--;
        }
    }
    /**
     * Метод для видалення рядка за індексом.
     * @param index - індекс рядка, що видаляється.
     */
    public void removeRow(int index) {
        if (getRowCount() == 1) {
            this.prop = null;
            rowCount--;
            currentRow--;
            return;
        }
        String[][] buf =
            new String[this.getRowCount() - 1]
                      [this.getColCount()];
        for (int i = 0; i < index; i++) {
            for (int j = 0; j < getColCount(); j++) {
                buf[i][j] = this.prop[i][j];
            }
        }
        for (int i = index + 1; i < getRowCount(); i++) {
            for  (int j = 0; j < getColCount(); j++) {
                buf[i - 1][j] = this.prop[i][j];
            }
        }
        this.prop = null;
        this.prop = buf;
        rowCount--;
        currentRow--;
    }
    /**
     * Метод для видалення всіх рядків.
     */
    public void clean() {
        String[][] buf = new String[0][3];
        this.prop = null;
        this.prop = buf;
        rowCount = 0;
        currentRow = 0;
    }
}
```

A.15 SectorDiagramDrawer

```
package com.lab111.work;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.geom.Arc2D;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import java.awt.image.BufferedImage;
import java.util.ArrayList;

import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JColorChooser;
import javax.swing.JDialog;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;
import javax.swing.SwingUtilities;
import javax.swing.border.EtchedBorder;
/**
 * Метод, що являє полотно для малювання секторної діаграми.
 * @author [redacted]
 *
 */
public class SectorDiagramDrawer extends JPanel implements Drawer {
 /**
  * Розмір панелі.
  */
 private Dimension canSize;
 /**
  * Вміст csv файлу, що малюється.
  */
 private Properties properties;
 /**
  * Вспливаюче меню.
  */
 private JPopupMenu popupMenu;
 /**
  * Пункт видалення сектору.
  */
 private JMenuItem itemDelete;
 /**
  * Пункт зміни кольору сектору.
  */
 private JMenuItem itemSetColor;
 /**
  * Пункт зміни назви сектору.
  */
 private JMenuItem itemSetName;
 /**
  * Пункт зміни розміру сектору.
  */
 private JMenuItem itemSetAmount;
 /**
  * Панель в якій міститься цей drawer.
  */
 private FileDesign parent = null;
 /**
  *
  */
 private boolean flag = false;
 /**
  *
  */
 private double amount = 0;
 /**
  *
  */
```

```
private ArrayList<Arc2D.Double> arcs = new ArrayList<Arc2D.Double>();
/**
 * Координата x центра діаграми.
 */
private double x0 = 0;
/**
 * Координата y центра діаграми.
 */
private double y0 = 0;
/**
 * Стартовий кут діаграми.
 */
private double startAng = 0;
/**
 * Номер останього натиснутого сектору.
 */
private int num = -1;
/**
 * Конструктор класу.
 * @param x - ширина панелі.
 * @param y - висота панелі.
 * @param p - об'єкт класу {@link Properties}.
 */
public SectorDiagramDrawer(int x, int y, Properties p) {
    setProperties(p);
    makeJPopupMenu();
    this.properties = p;
    setLayout(new BorderLayout());
    canSize = new Dimension(x, y);
    this.setMinimumSize(new Dimension(800,  100));
    setBackground(Color.white);
    this.addMouseListener(new DrawerObserver());

}
@Override
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    setBackground(Color.WHITE);
    draw(g);
}
@Override
public void setProperties(Properties p) {
    this.properties = null;
    this.properties = p;
}
/**
 * Метод для зміни панелі в якій міститься drawer.
 * @param parent - нова панель.
 */
public void setParent(JPanel parent) {
    this.parent = (FileDesign) parent;
}
@Override
public void draw(Graphics g) {
    super.paintComponent(g);
    arcs = new ArrayList<Arc2D.Double>();
    setBackground(Color.WHITE);
    Graphics2D g2 = (Graphics2D) g;
    double lower = canSize.width/2;
    double next = 0;
    double n = startAng;
    double hw = getWidth();
    double hh = getHeight();
    if (hw > hh) {
        lower = hh;
    } else {
        lower = hw;
    }
    amount = 0;
    for (int i = 0; i < properties.getRowCount(); i++) {
        amount += Double.parseDouble(properties.getCell(i, 1));
    }
    x0 = (lower - 20) / 2 + 10;
    y0 = (lower - 20) / 2 + 10;
    double h = -20;
    Rectangle2D.Double r2d = null;
    for (int i = 0; i < properties.getRowCount(); i++) {
```

```
        double current  = Double.parseDouble(
                properties.getCell(i, 1));
        current = current * 100 / amount;
        Color c = new Color(
                Integer.parseInt(properties.
                        getCell(i, 2)));
        g2.setColor(c);
        next += (Double.parseDouble(
                properties.getCell(
                        i, 1)) / amount) * 360;
        Pie arc = new Pie(10, 10, lower - 20, lower - 20,
                n, next - n + startAng,
                Arc2D.Double.PIE);
        arcs.add(arc);
        g2.fill(arc);
        h += 40;
        r2d = new Rectangle2D.Double(lower, h, 50, 20);
        g2.fill(r2d);
        g2.setColor(Color.black);
        g2.drawString(properties.getCell(
                i, 0), (float) lower + 60,
                (float) h + 15);
        n = next + startAng;
    }
}
@Override
public BufferedImage createImage() {
    BufferedImage bi = new BufferedImage(this.getWidth(),
            this.getHeight(), BufferedImage.TYPE_INT_RGB);
    Graphics2D g2 = (Graphics2D) bi.getGraphics();
    this.draw(g2);
    g2.dispose();
    return bi;
}
/**
 * Метод для створення меню, що випливає.
 */
public void makeJPopupMenu() {
    DrawerObserver drawerObserver = new DrawerObserver();
    itemDelete = new JMenuItem("  Delete");
    itemDelete.addActionListener(drawerObserver);
    itemSetColor = new JMenuItem("  Change Color");
    itemSetColor.addActionListener(drawerObserver);
    itemSetName = new JMenuItem("  Change Name");
    itemSetName.addActionListener(drawerObserver);
    itemSetAmount = new JMenuItem("  Change Amount");
    itemSetAmount.addActionListener(drawerObserver);
    popupMenu = new JPopupMenu();
    popupMenu.setBorder(BorderFactory.
            createEtchedBorder(EtchedBorder.LOWERED));
    popupMenu.add(itemDelete);
    popupMenu.addSeparator();
    popupMenu.add(itemSetName);
    popupMenu.add(itemSetAmount);
    popupMenu.add(itemSetColor);
}
/**
 * Внутрішній слухач подій.
 * @author rebelizant Гула Вадим
 *
 */
private class DrawerObserver extends MouseAdapter
implements ActionListener {
    @Override
    public void mouseReleased(MouseEvent e) {
        setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
        if (flag) {
            flag = false;
            return;
        }
        Point2D.Double p = new Point2D.Double(
                e.getX(), e.getY());
        if (num != -1) {
            for (int i = 0; i < arcs.size(); i++) {
                if (arcs.get(i).contains(p)) {
                    if (num != i) {
                        int k = i - 1;
```

```
                    if (i - 1 == num) {
                        if (i - 1 == num) {
                            k = i - 1;
                        }
                    } else {
                        if (i + 1 == num) {
                            change(p, i);
                        }
                        return;
                    }
                    double angle =
                        Math.atan(Math.
                                abs((p.getY()-y0)/(p.getX()-x0)));
                    angle = angle * 180 / Math.PI;
                    double startAngle =
                        arcs.get(k).getAngleStart();
                    double oldExtent =
                        arcs.get(k).getAngleExtent();
                    if ((p.getX() - x0) > 0  && (p.getY() - y0) < 0) {
                        arcs.get(k).setAngleExtent(
                                Math.abs(angle-startAngle));
                    } else {
                        if ( (p.getX() - x0) < 0 && (p.getY() - y0) < 0 ) {
                            angle = Math.atan(Math.abs(
                                    (p.getX() - x0) / (p.getY() - y0)));
                            angle = angle * 180 / Math.PI;
                            arcs.get(k).setAngleExtent(
                                    90 - startAngle + angle);
                        } else {
                            if ( (p.getX() - x0) < 0 && (p.getY() - y0) > 0 ) {
                                angle = Math.atan(Math.abs(
                                        (p.getY() - y0) / (p.getX() - x0)));
                                angle = angle * 180 / Math.PI;
                                arcs.get(k).setAngleExtent(180 - startAngle +
angle);
                            } else {
                                arcs.get(k).setAngleExtent(360 - (angle +
startAngle));
                            }
                        }
                    }
                    double old = arcs.get(i).getAngleExtent();
                    arcs.get(i).setAngleExtent(Math.abs(
                            oldExtent - arcs.get(k).
                            getAngleExtent() + old));
                    properties.setCell("" + Math.round(
                            arcs.get(i).getAngleExtent()
                            * amount / 360), i, 1);
                    properties.setCell("" + Math.round(
                            arcs.get(k).getAngleExtent()
                            * amount / 360), k, 1);
                    parent.getModel().setSaved(false);
                    parent.getModel().notifyObservers(properties);
                    return;
                }
                double alpha = Math.atan(Math.abs(
                        (p.getY() - y0) / (p.getX() - x0)));
                alpha = alpha * 180 / Math.PI;
                double beta = arcs.get(i).
                getAngleStart();
                double oldExtent = arcs.
                get(i).getAngleExtent();
                if ( (p.getX() - x0) > 0  && (p.getY() - y0) < 0) {
                    arcs.get(i).setAngleExtent(
                            Math.abs(alpha - beta));
                } else {
                    if ((p.getX() - x0) < 0 && (p.getY() - y0) < 0 ) {
                        alpha = Math.atan(Math.abs(
                                (p.getX() - x0) / (p.getY() - y0)));
                        alpha = alpha * 180 / Math.PI;
                        arcs.get(i).setAngleExtent(
                                90 - beta + alpha);
                    } else {
                        if ((p.getX() - x0) < 0 && (p.getY() - y0) > 0 ) {
                            alpha = Math.atan(Math.abs(
                                    (p.getY() - y0) / (p.getX() - x0)));
                            alpha = alpha * 180 / Math.PI;
```

```
                        arcs.get(i).setAngleExtent(180 - beta + alpha);
                    } else {
                        arcs.get(i).setAngleExtent(
                                360 - alpha - beta);
                    }
                }
            }
            int j = i + 1;
            if (j == arcs.size()) {
                j = 0;
            }
            double old = arcs.get(j).getAngleExtent();
            arcs.get(j).setAngleExtent(Math.abs(
                    oldExtent - arcs.get(i).
                    getAngleExtent() + old));
            properties.setCell("" + Math.round(
                    arcs.get(i).getAngleExtent()
                    * amount / 360), i, 1);
            properties.setCell("" + Math.round(
                    arcs.get(j).getAngleExtent()
                    * amount / 360), j, 1);
            parent.getModel().setSaved(false);
            parent.getModel().notifyObservers(properties);
            break;
        }
    }
  }
}
/**
 * Метод для перетягування.
 * @param p - нова точка.
 * @param i - сектор в якому міститься нова точка.
 */
private void change(Point2D p, int i) {
    double angle = Math.atan(Math.abs(
            (p.getY() - y0) / (p.getX() - x0)));
    angle = angle * 180 / Math.PI;
    int k = i + 1;
    double startAngle = arcs.get(i).getAngleStart();
    double oldExtent = arcs.get(i).getAngleExtent();
    if ((p.getX() - x0) > 0  && (p.getY() - y0) < 0) {
        arcs.get(i).setAngleExtent(
                Math.abs(angle - startAngle));
    } else {
        if ((p.getX() - x0) < 0 && (p.getY() - y0) < 0 ) {
            angle = Math.atan(Math.abs(
                    (p.getX() - x0) / (p.getY() - y0)));
            angle = angle * 180 / Math.PI;
            arcs.get(i).setAngleExtent(
                    90 - startAngle + angle);
        } else {
            if ((p.getX() - x0) < 0 && (p.getY() - y0) > 0 ) {
                angle = Math.atan(Math.abs(
                        (p.getY() - y0) / (p.getX() - x0)));
                angle = angle * 180 / Math.PI;
                arcs.get(i).setAngleExtent(
                        180 - startAngle + angle);
            } else {
                arcs.get(i).setAngleExtent(
                        360 - (angle + startAngle));
            }
        }
    }
    double old = arcs.get(k).getAngleExtent();
    arcs.get(k).setAngleExtent(Math.abs(
            oldExtent - arcs.get(i).
            getAngleExtent() + old));
    properties.setCell("" + Math.round(
            arcs.get(i).getAngleExtent()
            * amount / 360), i, 1);
    properties.setCell("" + Math.round(
            arcs.get(k).getAngleExtent()
            * amount / 360), k, 1);
    parent.getModel().setSaved(false);
    parent.getModel().notifyObservers(properties);
}
@Override
```

```
    public void mousePressed(MouseEvent e) {
        Point2D.Double p = new Point2D.Double(
                e.getX(), e.getY());
        num = -1;
        for (int i = 0; i < arcs.size(); i++) {
            if (arcs.get(i).contains(p)) {
                setCursor(new Cursor(Cursor.
                        HAND_CURSOR));
                num = i;
                break;
            }
        }
        if (e.isPopupTrigger() && num != -1) {
            popupMenu.show(e.getComponent(),
                    e.getX(), e.getY());
            flag = true;
        }
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getSource() == itemSetColor) {
            SwingUtilities.invokeLater(new Runnable() {
                @Override
                public void run() {
                    Color color = JColorChooser.showDialog(
                            null, "Change Color", Color.WHITE);
                    if (color != null) {
                        parent.getController().setColor(num, color);
                    }
                }
            });
            return;
        }
        if(e.getSource() == itemDelete) {
            if (num == -1) {
                return;
            }
            parent.getController().remove(num);
            arcs.remove(num);
            num = -1;
            return;
        }
        if (e.getSource() == itemSetName) {
            SwingUtilities.invokeLater(new Runnable() {
                @Override
                public void run() {
                    String name = JOptionPane.showInputDialog(
                            "Input new name", "new name");
                    if (name != null) {
                        parent.getController().setName(num, name);
                    }
                }
            });
            return;
        }
        if (e.getSource() == itemSetAmount) {
            SwingUtilities.invokeLater(new Runnable() {
                @Override
                public void run() {
                    String amountStr =
                        JOptionPane.showInputDialog(
                            "Input new amount", "100");
                    if (amountStr != null) {
                        parent.getController().setAmount(num, amountStr);
                    }
                }
            });
            return;
        }
    }
 }
}
```

A.16 TestIO

```
package com.lab111.work;
import javax.swing.SwingUtilities;
```

```
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;
/**
 * Головний клас, що містить метод main.
 * @author [redacted]
 *
 */
public class TestIO {
 /**
  * Головний метод.
  * @param args - додаткові параметри.
  */
 public static void main(String args[]) {
     SwingUtilities.invokeLater(new Runnable() {
         @Override
         public void run() {
     /*      try {
                 UIManager.setLookAndFeel(
                         UIManager.
                         getSystemLookAndFeelClassName());
             } catch (ClassNotFoundException e1) {
                 e1.printStackTrace();
             } catch (InstantiationException e1) {
                 e1.printStackTrace();
             } catch (IllegalAccessException e1) {
                 e1.printStackTrace();
             } catch (UnsupportedLookAndFeelException e1) {
                 e1.printStackTrace();
             } */
             Design d = new Design(
                     new DesignModel());
         }
     });
 }
}
```

ДОДАТОК В. СТРУКТУРА ПРОЕКТУ