**НТУУ «КПІ» 2014**

**Інженерія ПЗ**

**Лабораторна робота №3:**

**Структурні шаблони проектування ПЗ.**

**Шаблони Composite. Decorator. Proxy.**

**Факультет: ФІОТ**

**Група: ІО-34**

**Виконав:**

**Власов М. Д.**

**Викладач:**

**Антонюк А.І.**

**Мета**

Ознайомлення з видами шаблонів проектування ПЗ. Вивчення структурних шаблонів. Отримання базових навичок з застосування шаблонів Composite, Decorator та Proxy.

**Завдання**

1. Ознайомитись з призначенням та видами шаблонів проектування ПЗ. Вивчити класифікацію шаблонів проектування ПЗ. Знати назви шаблонів, що відносяться до певного класу.

2. Вивчити структурні шаблонів проектування ПЗ. Знати загальну характеристику структурних шаблонів та призначення кожного з них.

3. Детально вивчити структурні шаблони проектування Composite, Decorator та Proxy. Для кожного з них:

* вивчити Шаблон, його призначення, альтернативні назви, мотивацію, випадки коли його застосування є доцільним та результати такого застосування;
* знати особливості реалізації Шаблону, споріднені шаблони, відомі випадки його застосування в програмних додатках;
* вільно володіти структурою Шаблону, призначенням його класів та відносинами між ними;
* вміти розпізнавати Шаблон в UML діаграмі класів та будувати сирцеві коди Java-класів, що реалізують шаблон.

4. В підготованому проекті (ЛР1) створити програмний пакет com.lab111.labwork3. В пакеті розробити інтерфейси і класи, що реалізують завдання (згідно варіанту) з застосуванням одного чи декількох шаблонів (п.3). В розроблюваних класах повністю реалізувати методи, пов'язані з функціюванням Шаблону. Методи, що реалізують бізнес-логіку закрити заглушками з виводом на консоль інформації про викликаний метод та його аргументи. Приклад реалізації бізнес-методу:

void draw(int x, int y){

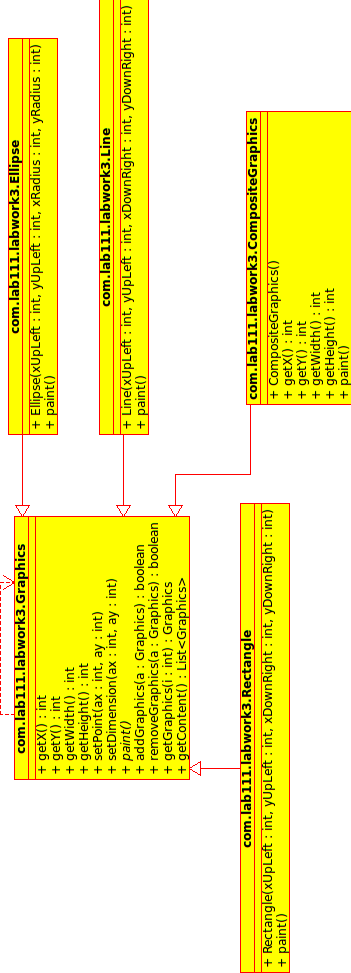
System.out.println(“Метод draw з параметрами x=”+x+” y=”+y);

}

5. За допомогою автоматизованих засобів виконати повне документування розроблених класів (також методів і полів), при цьому документація має в достатній мірі висвітлювати роль певного класу в загальній структурі Шаблону та особливості конкретної реалізації.

**Варіант**

Визначити специфікації класів, які подають графічні примітиви та їх композиції у редакторі векторної графіки. Кожний примітив має атрибути розміщення - позиція (координати x та y) і розмір (ширина та висота). Реалізувати бізнес-метод відображення таких атрибутів розміщення для примітивів (задаються в конструкторі) і композицій (динамічно обчислюються).



**Код**

**package com.lab111.labwork3;**

**import** java.util.List;

/\*\*

\* Composite class,

\* unbreakable group of graphics

\*

\* **@author** Maksym Vlasov

\*/

**public** **class** CompositeGraphics **extends** Graphics {

/\*\*

\* Constructor, create object and make possible to add graphics

\*/

**public** CompositeGraphics() {

initContent();

}

/\*\*

\* Generate new x coordinate, as the most left point

\*

\* **@return** new x coordinate

\*/

@Override

**public** **int** getX() {

List<Graphics> list = getContent();

**if** (list == **null**) {

**return** -1;

}

**else** {

**if** (list.isEmpty()) {

**return** -1;

}

**int** ax = list.get(0).getX();

**for** (**int** i = 1; i < list.size(); i++) {

**if** (list.get(i).getX() < ax) {

ax = list.get(i).getX();

}

}

x = ax;

**return** x;

}

}

/\*\*

\* Generate new y coordinate, as the most top point

\*

\* **@return** new y coordinate

\*/

@Override

**public** **int** getY() {

List<Graphics> list = getContent();

**if** (list == **null**) {

**return** -1;

}

**else** {

**if** (list.isEmpty()) {

**return** -1;

}

**int** ay = list.get(0).getY();

**for** (**int** i = 1; i < list.size(); i++) {

**if** (list.get(i).getY() < ay) {

y = list.get(i).getY();

}

}

y = ay;

**return** y;

}

}

/\*\*

\* Generate new width, as difference between x and most right point

\*

\* **@return** new width

\*/

@Override

**public** **int** getWidth() {

List<Graphics> list = getContent();

**if** (list == **null**) {

**return** 0;

}

**else** {

**if** (list.isEmpty()) {

**return** 0;

}

**int** ax = list.get(0).getX() + list.get(0).getWidth();

**for** (**int** i = 1; i < list.size(); i++) {

**if** ((list.get(i).getX() + list.get(i).getWidth()) > ax) {

ax = list.get(i).getX() + list.get(i).getWidth();

}

}

width = ax - x;

**return** width;

}

}

/\*\*

\* Generate new height, as difference between y and most down point

\*

\* **@return** new height

\*/

@Override

**public** **int** getHeight() {

List<Graphics> list = getContent();

**if** (list == **null**) {

**return** 0;

}

**else** {

**if** (list.isEmpty()) {

**return** 0;

}

**int** ay = list.get(0).getY() + list.get(0).getHeight();

**for** (**int** i = 1; i < list.size(); i++) {

**if** ((list.get(i).getY() + list.get(i).getHeight()) > ay) {

ay = list.get(i).getY() + list.get(i).getHeight();

}

}

height = ay - y;

**return** height;

}

}

/\*\*

\* Paint all included graphics

\*/

@Override

**public** **void** paint() {

List<Graphics> list = getContent();

**for** (**int** i = 0; i < list.size(); i++) {

list.get(i).paint();

}

}

}

**package com.lab111.labwork3;**

/\*\*

\* Leaf class,

\* Ellipse line

\*

\* **@author** Maksym Vlasov

\*/

**public** **class** Ellipse **extends** Graphics {

/\*\*

\* Constructor, create size and place of graphics

\*

\* **@param** xUpLeft

\* is coordinate x of top left corner

\* **@param** yUpLeft

\* is coordinate y of top left corner

\* **@param** xRadius

\* is max radius along x

\* **@param** yRadius

\* is max radius along y

\*/

**public** Ellipse(**int** xUpLeft, **int** yUpLeft, **int** xRadius, **int** yRadius) {

x = xUpLeft;

y = yUpLeft;

width = 2 \* xRadius;

height = 2 \* yRadius;

}

/\*\*

\* Paint ellipse

\*/

@Override

**public** **void** paint() {

System.***out***.println("Ellipse: x=" + x + ", y=" + y + ", width=" + width

+ ", height=" + height);

}

}

**package com.lab111.labwork3;**

**import** java.util.ArrayList;

**import** java.util.List;

/\*\*

\* General abstract component class, generalize all possible type of vector

\* graphis

\*

\* **@author** Maksym Vlasov

\*/

**public** **abstract** **class** Graphics {

**protected** **int** x; //Coordinate x, top left corner

**protected** **int** y; //Coordinate y, top left corner

**protected** **int** width; //Width of graphic

**protected** **int** height; //Height of graphic

**private** List<Graphics> content; //List of included graphics

/\*\*

\* **@return** coordinate x, top left corner

\*/

**public** **int** getX() {

**return** x;

}

/\*\*

\* **@return** coordinate y, top left corner

\*/

**public** **int** getY() {

**return** y;

}

/\*\*

\* **@return** width of graphic

\*/

**public** **int** getWidth() {

**return** width;

}

/\*\*

\* **@return** height of graphic

\*/

**public** **int** getHeight() {

**return** height;

}

/\*\*

\* Set coordinate of top left corner of graphics

\*

\* **@param** ax

\* is new coordinate x

\* **@param** ay

\* is new coordinate y

\*/

**public** **void** setPoint(**int** ax, **int** ay) {

x = ax;

y = ay;

}

/\*\*

\* Set size of graphics

\*

\* **@param** ax

\* is new width

\* **@param** ay

\* is new height

\*/

**public** **void** setDimension(**int** ax, **int** ay) {

width = ax;

height = ay;

}

/\*\*

\* Paint graphics, must be realise

\*/

**public** **abstract** **void** paint();

/\*\*

\* Manage structure, add new graphic to the composite graphics

\*

\* **@param** a

\* is new graphic

\* **@return** true if adding was successfull

\*/

**public** **boolean** addGraphics(Graphics a) {

**if** (content != **null**) {

**return** content.add(a);

}

**return** **false**;

}

/\*\*

\* Manage structure, remove some graphic from the composite graphics

\*

\* **@param** a

\* is removing graphic

\* **@return** true if removing was successfull

\*/

**public** **boolean** removeGraphics(Graphics a) {

**if** (content != **null**) {

**return** content.remove(a);

}

**return** **false**;

}

/\*\*

\* Manage structure, get graphics by it's index

\*

\* **@param** i

\* is index of graphics

\* **@return** graphics, placed on index-parameter

\*/

**public** Graphics getGraphics(**int** i) {

**if** (content == **null**) {

**return** **null**;

}

**return** content.get(i);

}

/\*\*

\* Make new list, if graphics can include other graphics

\*/

**protected** **void** initContent() {

content = **new** ArrayList<Graphics>();

}

/\*\*

\* **@return** list of included graphics

\*/

**public** List<Graphics> getContent() {

**return** content;

}

}

**package com.lab111.labwork3;**

/\*\*

\* Leaf class,

\* Straight line

\*

\* **@author** Maksym Vlasov

\*/

**public** **class** Line **extends** Graphics {

/\*\*

\* Constructor, create size and place of graphics

\*

\* **@param** xUpLeft

\* is coordinate x of left end

\* **@param** yUpLeft

\* is coordinate y of left end

\* **@param** xDownRight

\* is coordinate x of right end

\* **@param** yDownRight

\* is coordinate x of right end

\*/

**public** Line(**int** xUpLeft, **int** yUpLeft, **int** xDownRight, **int** yDownRight) {

x = xUpLeft;

y = yUpLeft;

width = xDownRight - xUpLeft;

height = yDownRight - yUpLeft;

}

/\*\*

\* Paint line

\*/

@Override

**public** **void** paint() {

System.***out***.println("Line: x=" + x + ", y=" + y + ", width=" + width

+ ", height=" + height);

}

}

**package com.lab111.labwork3;**

/\*\*

\* Leaf class, Rectangle line

\*

\* **@author** Maksym Vlasov

\*/

**public** **class** Rectangle **extends** Graphics {

/\*\*

\* Constructor, create size and place of graphics

\*

\* **@param** xUpLeft

\* is coordinate x of top left corner

\* **@param** yUpLeft

\* is coordinate y of top left corner

\* **@param** xDownRight

\* is coordinate x of low right corner

\* **@param** yDownRight

\* is coordinate x of low right corner

\*/

**public** Rectangle(**int** xUpLeft, **int** yUpLeft, **int** xDownRight, **int** yDownRight) {

x = xUpLeft;

y = yUpLeft;

width = xDownRight - xUpLeft;

height = yDownRight - yUpLeft;

}

/\*\*

\* Paint rectangle

\*/

@Override

**public** **void** paint() {

System.***out***.println("Rectangle: x=" + x + ", y=" + y + ", width="

+ width + ", height=" + height);

}

}

## Package com.lab111.labwork3

|  |  |
| --- | --- |
| Class Summary | |
| [CompositeGraphics](file:///home/vm/Документи/02%20course%20(Паливо)/01%20Semester/01Palevo_3%20семестр/Инженерия%20ПЗ/Лаб%203/template/doc/com/lab111/labwork3/CompositeGraphics.html) | Composite class, unbreakable group of graphics |
| [Ellipse](file:///home/vm/Документи/02%20course%20(Паливо)/01%20Semester/01Palevo_3%20семестр/Инженерия%20ПЗ/Лаб%203/template/doc/com/lab111/labwork3/Ellipse.html) | Leaf class, Ellipse line |
| [Graphics](file:///home/vm/Документи/02%20course%20(Паливо)/01%20Semester/01Palevo_3%20семестр/Инженерия%20ПЗ/Лаб%203/template/doc/com/lab111/labwork3/Graphics.html) | General abstract component class, generalize all possible type of vector graphiсs |
| [Line](file:///home/vm/Документи/02%20course%20(Паливо)/01%20Semester/01Palevo_3%20семестр/Инженерия%20ПЗ/Лаб%203/template/doc/com/lab111/labwork3/Line.html) | Leaf class, Straight line |
| [Rectangle](file:///home/vm/Документи/02%20course%20(Паливо)/01%20Semester/01Palevo_3%20семестр/Инженерия%20ПЗ/Лаб%203/template/doc/com/lab111/labwork3/Rectangle.html) | Leaf class, Rectangle line |