**НТУУ «КПІ» 2014**

**Інженерія ПЗ**

**Лабораторна робота №7:**

**Шаблони поведінки - 3.**

**Шаблони Memento, State, Command та Interpreter.**

**Факультет: ФІОТ**

**Група: ІО-34**

**Заліковка №3405**

**Виконав:**

**Власов М. Д.**

**Викладач:**

**Антонюк А.І.**

**Мета**

Вивчення шаблонів поведінки. Отримання базових навичок з застосування шаблонів Memento, State, Command та Interpreter.

**Завдання**

1. Повторити шаблони поведінки для проектування ПЗ. Знати загальну характеристику шаблонів поведінки та призначення кожного з них.

2. Детально вивчити шаблони поведінки для проектування ПЗ - Memento, State, Command та Interpreter. Для кожного з них:

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

3. В підготованому проекті (ЛР1) створити програмний пакет com.lab111.labwork7. В пакеті розробити інтерфейси і класи, що реалізують завдання (згідно варіанту) з застосуванням одного чи декількох шаблонів (п.2). 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){

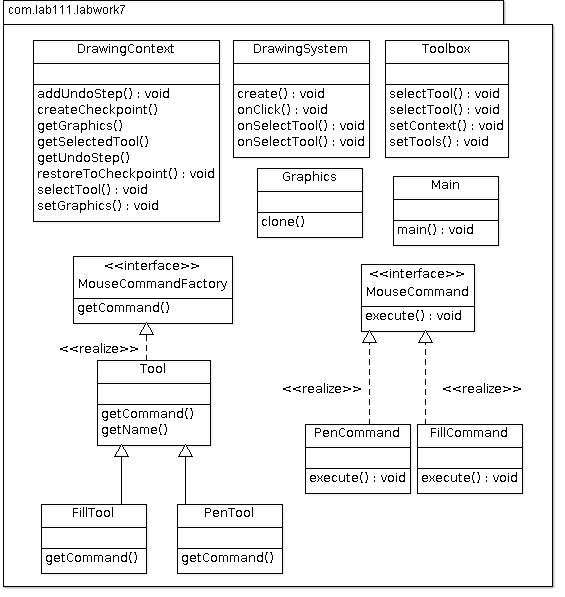
System.out.println(“Метод draw з параметрами x=”+x+” y=”+y);

}

4. 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 3405 mod 8**

5. Визначити специфікації класів, що подають інструменти малювання та робочий простір в графічному редакторі. Реалізувати механізм зміни реакції на натискання кнопки миші в залежності від вибраного інструменту. Уникати використання громіздких умовних конструкцій.



**Код**

**package** com.lab111.labwork7;

**import** java.util.Stack;

/\*\*

\* Context that keeps current drawing editor state. In this case tool property

\* used as state object thus taking its part in a 'state' pattern Also, it is

\* able to

\*

\* **@author** Maxym Vlasov

\* **@version** 1.0

\*/

**public** **class** DrawingContext {

**private** Tool selectedTool;

**private** Graphics graphics;

**private** Stack<Object> undo;

**public** DrawingContext() {

System.***out***.println("Created new 'DrawingContext' state object");

undo = **new** Stack<Object>();

}

**public** **void** selectTool(Tool tool) {

selectedTool = tool;

System.***out***.println("Tool '" + tool.getName()

+ "' selected in drawing context");

}

**public** Tool getSelectedTool() {

**return** selectedTool;

}

/\*\*

\* **@return** the graphics

\*/

**public** Graphics getGraphics() {

**return** graphics;

}

/\*\*

\* **@param** graphics

\* the graphics to set

\*/

**public** **void** setGraphics(Graphics graphics) {

**this**.graphics = graphics;

}

/\*\*

\* Sample 'memento' object creation, thus implementing a 'memento' pattern

\*

\* **@return**

\*/

**public** Object createCheckpoint() {

**return** graphics.clone();

}

/\*\*

\* Restores previously saved graphics content thus implementing a 'memento'

\* pattern

\*

\* **@param** checkpoint

\*/

**public** **void** restoreToCheckpoint(Object checkpoint) {

**if** (checkpoint **instanceof** Graphics) {

setGraphics((Graphics) checkpoint);

}

}

**public** **void** addUndoStep(Object step) {

System.***out***.println("'Memento' object is stored in undo chain");

undo.push(step);

}

**public** Object getUndoStep() {

**return** undo.pop();

}

}

**package** com.lab111.labwork7;

/\*\*

\* A graphics editor facade acting as 'invoker' in 'command' pattern and as a

\* caretaker in 'memento' pattern

\*

\* **@author** Maxym Vlasov

\* **@version** 1.0

\*/

**public** **class** DrawingSystem {

**private** DrawingContext drawingContext;

**private** Toolbox toolbox;

**public** DrawingSystem() {

drawingContext = **new** DrawingContext();

toolbox = **new** Toolbox(**new** Tool[] { **new** PenTool(), **new** FillTool() });

toolbox.setContext(drawingContext);

}

**public** **void** create(**int** x, **int** y) {

Graphics graphics = **new** Graphics(x, y);

drawingContext.setGraphics(graphics);

}

/\*\*

\* Process particular mouse event. Invoker role implementation in a

\* 'command' pattern.

\*

\* **@param** x

\* **@param** y

\*/

**public** **void** onClick(**int** x, **int** y) {

Tool tool = drawingContext.getSelectedTool();

MouseCommand command = tool.getCommand();

// 'caretaker' role in 'memento' pattern

Object undoStep = drawingContext.createCheckpoint();

drawingContext.addUndoStep(undoStep);

// end of 'caretaker' role code

command.execute(drawingContext.getGraphics(), x, y);

}

**public** **void** onSelectTool(String name) {

toolbox.selectTool(name);

}

**public** **void** onSelectTool(**int** index) {

toolbox.selectTool(index);

}

}

**package** com.lab111.labwork7;

/\*\*

\* A concrete command implementation

\*

\* **@author** Maxym Vlasov

\* **@version** 1.0

\*/

**public** **class** FillCommand **implements** MouseCommand {

@Override

**public** **void** execute(Graphics graphics, **int** x, **int** y) {

System.***out***.format("Fill the area around (%d, %d)\n", x, y);

}

}

**package** com.lab111.labwork7;

/\*\*

\* Fill tool implementation, example of a concrete 'client' in 'command' pattern

\*

\* **@author** Maxym Vlasov

\* **@version** 1.0

\*/

**public** **class** FillTool **extends** Tool {

**public** FillTool() {

name = "Fill";

}

@Override

**public** MouseCommand getCommand() {

System.***out***.println("Command object created");

**return** **new** FillCommand();

}

}

**package** com.lab111.labwork7;

/\*\*

\* Graphics is a class that should represent edited graphics data.

\* Since we don't implement any business logic, it is empty.

\*

\* **@author** Maxym Vlasov

\* **@version** 1.0

\*/

**public** **class** Graphics **implements** Cloneable {

**public** Graphics() {

}

**public** Graphics(**int** x, **int** y) {

System.***out***.format("Created %dx%d pixels graphics\n", x, y);

}

@Override

**public** Object clone() {

System.***out***.println("Current state of graphics data was taken in a 'memento' object");

**return** **new** Graphics();

}

}

**package** com.lab111.labwork7;

/\*\*

\* **@author** Maxym Vlasov

\* **@version** 1.0

\*/

**public** **class** Main {

/\*\*

\* **@param** args

\*/

**public** **static** **void** main(String[] args) {

DrawingSystem drawingSystem = **new** DrawingSystem();

drawingSystem.create(400, 300);

drawingSystem.onSelectTool("Pen");

drawingSystem.onClick(100, 200);

drawingSystem.onClick(120, 180);

drawingSystem.onSelectTool("Fill");

drawingSystem.onClick(150, 160);

}

}

**package** com.lab111.labwork7;

/\*\*

\* Interface for commands

\*

\* **@author** Maxym Vlasov

\* **@version** 1.0

\*/

**public** **interface** MouseCommand {

/\*\*

\* Executes drawing command on given graphics at given position

\*

\* **@param** graphics

\* **@param** x

\* **@param** y

\*/

**public** **void** execute(Graphics graphics, **int** x, **int** y);

}

**package** com.lab111.labwork7;

/\*\*

\* Factory for mouse command interface. Also a way to introduce 'client' role in

\* a 'command' pattern

\*

\* **@author** Maxym Vlasov

\* **@version** 1.0

\*/

**public** **interface** MouseCommandFactory {

/\*\*

\* Instantiates a command that can be executed at a later time

\*

\* **@return**

\*/

MouseCommand getCommand();

}

**package** com.lab111.labwork7;

/\*\*

\* A concrete command implementation

\*

\* **@author** Maxym Vlasov

\* **@version** 1.0

\*/

**public** **class** PenCommand **implements** MouseCommand {

@Override

**public** **void** execute(Graphics graphics, **int** x, **int** y) {

System.***out***.format("Do pen drawing at (%d, %d)\n", x, y);

}

}

**package** com.lab111.labwork7;

/\*\*

\* Pen tool implementation, example of a concrete 'client' in 'command' pattern

\*

\* **@author** Maxym Vlasov

\* **@version** 1.0

\*/

**public** **class** PenTool **extends** Tool {

**public** PenTool() {

name = "Pen";

}

@Override

**public** MouseCommand getCommand() {

System.***out***.println("Command object created");

**return** **new** PenCommand();

}

}

**package** com.lab111.labwork7;

/\*\*

\* Base class for implementations of 'client' role in a 'command' pattern

\*

\* **@author** Maxym Vlasov

\* **@version** 1.0

\*/

**public** **abstract** **class** Tool **implements** MouseCommandFactory {

**protected** String name;

**public** String getName() {

**return** name;

}

@Override

**public** **abstract** MouseCommand getCommand();

}

**package** com.lab111.labwork7;

**import** java.util.HashMap;

**import** java.util.Map;

/\*\*

\* Class that implements a tool box in a graphics editor. Acts as 'client' in a

\* command pattern scenario

\*

\* **@author** Maxym Vlasov

\* **@version** 1.0

\*/

**public** **class** Toolbox {

**private** Tool tools[];

**private** Map<String, Tool> toolsByName;

**private** DrawingContext context;

**public** Toolbox() {

toolsByName = **new** HashMap<String, Tool>();

}

**public** Toolbox(Tool tools[]) {

**this**();

**this**.tools = tools;

**for** (Tool tool : tools) {

toolsByName.put(tool.getName(), tool);

}

}

**public** **void** selectTool(**int** index) {

**if** (index < 0 || index >= tools.length)

**return**;

context.selectTool(tools[index]);

}

**public** **void** selectTool(String name) {

Tool tool = toolsByName.get(name);

**if** (tool == **null**) {

**return**;

}

context.selectTool(tool);

}

**public** **void** setTools(Tool tools[]) {

**this**.tools = tools;

**for** (Tool tool : tools) {

toolsByName.put(tool.getName(), tool);

}

}

/\*\*

\* **@param** context

\* the context to set

\*/

**public** **void** setContext(DrawingContext context) {

**this**.context = context;

}

}