Міністерство освіти і науки, молоді та спорту України

Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №8**

По дисципліні «Інженерія програмного забезпечення»

Виконав: Перевірив:

Студент групи ІО-21 Викладач

Коноз А.О. Абу Усбах О. Н.

Дата здачі\_\_\_\_\_\_\_\_\_\_\_\_\_

Захищено з балом\_\_\_\_\_

Київ 2013

**Завдання**

1. Вивчити шаблони, що породжують. Знати загальну характеристику шаблонів, що породжують та призначення кожного з них.

2. Детально вивчити шаблони, що породжують - Prototype, Singleton та Factory Method. 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.

3. В підготованому проекті (ЛР1) створити програмний пакет com.lab111.labwork8. 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){

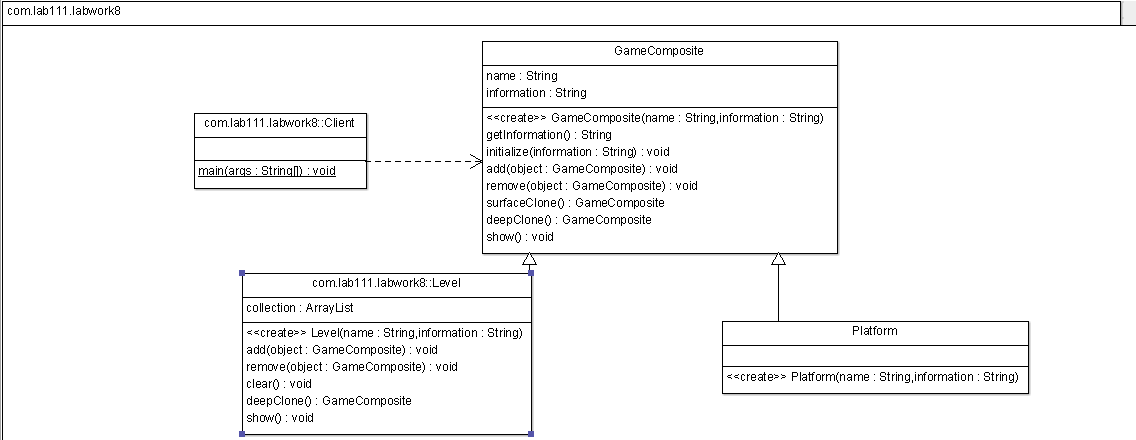
System.out.println(“Метод draw з параметрами x=”+x+” y=”+y);

}

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.

2112 mod 8 = 0

Визначити специфікації класів для подання композитної структури ігрового простору. Реалізувати глибоке та поверхневе клонування такої структури.



**package** com.lab111.labwork8;

/\*\*

\* **@author** User

\* Main class

\*/

**public** **class** Client {

/\*\*

\* **@param** args

\*/

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

GameComposite mainLevel = **new** Level("level1", "information of level");

Level level2 = **new** Level("level2", "information of leve2");

Platform platform1 = **new** Platform("platform1", "information of platform1");

Platform platform2 = **new** Platform("platform2", "information of platform2");

mainLevel.add(platform1);

mainLevel.add(level2);

level2.add(platform2);

mainLevel.show();

System.*out*.println();

System.*out*.println("Surface Clone");

GameComposite surfaceClon = mainLevel.surfaceClone();

surfaceClon.show();

System.*out*.println();

System.*out*.println("Deep Clone");

GameComposite deepClon = mainLevel.deepClone();

deepClon.show();

}

}

**package** com.lab111.labwork8;

/\*\*

\* **@author** User

\* The class defines the interface for the composite.

\*/

**public** **class** GameComposite **implements** Cloneable {

/\*\*

\* **@param** name is name of part

\* **@param** information is information for part.

\*/

**protected** String name;

**protected** String information;

/\*\*

\* Constructor initialize field of object

\* **@param** name is name of part

\* **@param** information is informationf for part

\*/

**public** GameComposite(String name, String information){

**this**.name = name;

**this**.information = information;

}

/\*\*

\* **@return** information for part

\*/

**public** String getInformation(){

**return** information;

}

/\*\*

\* Method set new information

\* **@param** information new information

\*/

**public** **void** initialize(String information){

**this**.information = information;

}

/\*\*

\* Method add new part to the composite structure.

\* **@param** object is object to added

\*/

**public** **void** add(GameComposite object){

System.*out*.println("Illegal operation");

}

/\*\*

\* Method remove part from the composite structure.

\* **@param** object part for removing

\*/

**public** **void** remove(GameComposite object){

System.*out*.println("Illegal operation");

}

/\*\*

\* **@return** surfase clone for the composite structure

\* **@throws** CloneNotSupportedException

\*/

**public** GameComposite surfaceClone() **throws** CloneNotSupportedException{

GameComposite var = (GameComposite) **this**.clone();

var.initialize("Clone " + var.getInformation());

**return** var;

}

/\*\*

\* **@return** deep clone for the composite structure

\* **@throws** CloneNotSupportedException

\*/

**public** GameComposite deepClone() **throws** CloneNotSupportedException{

GameComposite var = (GameComposite) **this**.clone();

var.initialize("Clone " + var.getInformation());

**return** var;

}

/\*\*

\* Method show information for part

\*/

**public** **void** show(){

System.*out*.println(name + " " + information);

}

}

**package** com.lab111.labwork8;

**import** java.util.ArrayList;

**import** java.util.Iterator;

/\*\*

\* **@author** User

\* This class override methods for composite part

\*/

**public** **class** Level **extends** GameComposite {

/\*\*

\* Constructor initialize field of object

\* **@param** name is name of part

\* **@param** information is informationf for part

\*/

**public** Level(String name, String information){

**super**(name, information);

}

/\*\*

\* **@param** collection is ctorage for child of this composite part

\*/

**private** ArrayList<GameComposite> collection = **new** ArrayList<GameComposite>();

/\* (non-Javadoc)

\* @see com.lab111.labwork8.GameComposite#add(com.lab111.labwork8.GameComposite)

\*/

**public** **void** add(GameComposite object){

collection.add(object);

}

/\* (non-Javadoc)

\* @see com.lab111.labwork8.GameComposite#remove(com.lab111.labwork8.GameComposite)

\*/

**public** **void** remove(GameComposite object){

collection.remove(object);

}

/\* (non-Javadoc)

\* @see com.lab111.labwork8.GameComposite#deepClone()

\*/

**public** GameComposite deepClone() **throws** CloneNotSupportedException{

GameComposite result = **new** Level(name, "Clone " + information);

Iterator<GameComposite> iter = collection.iterator();

**while**(iter.hasNext()){

result.add(iter.next().deepClone());

}

**return** result;

}

/\* (non-Javadoc)

\* @see com.lab111.labwork8.GameComposite#show()

\*/

**public** **void** show(){

System.*out*.println(name + " " + information);

Iterator<GameComposite> iter = collection.iterator();

**while**(iter.hasNext()){

iter.next().show();

}

}

}

**package** com.lab111.labwork8;

/\*\*

\* **@author** User

\* This class for leaf. He just extends methods of futher.

\*/

**public** **class** Platform **extends** GameComposite{

/\*\*

\* Constructor initialize field of object

\* **@param** name is name of part

\* **@param** information is informationf for part

\*/

**public** Platform(String name, String information) {

**super**(name, information);

}