КИЇВСЬКИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

**Лабораторна робота №8**

З інженерії програмного забезпечення

**Тема**: «ПОРОДЖУВАЛЬНІ ШАБЛОНИ. ШАБЛОНИ PROTOTYPE, SINGLETON, FACTORY METHOD»

залікова книжка № 3223

Виконав:

студент ІІ курсу ФІОТ

група ІО-32

Попенко Руслан

Перевірив:

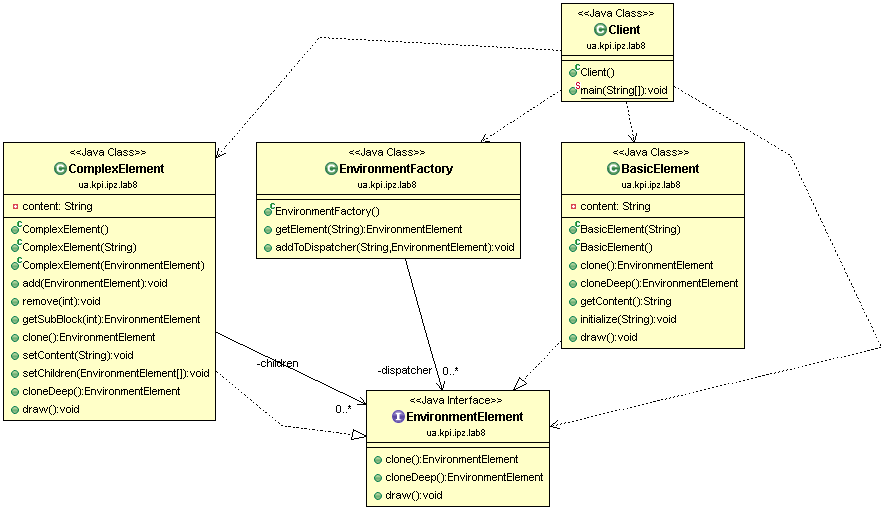
Антонюк А. І.

Київ-2014

**Мій варіант 0**

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

**Діаграма класів**

****

**Роздруківка тексту**

**package** ua.kpi.ipz.lab8;

/\*\*

\* A client that uses the constructed system.

\* **@author** Ruslan Popenko

\* **@version** 8.0

\*

\*/

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

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

//Creating a factory and setting basic prototypes to dispatcher.

**EnvironmentFactory** **factory**=**new** EnvironmentFactory();

factory.addToDispatcher("basic element", **new** BasicElement("Tree"));

factory.addToDispatcher("complex element", **new** ComplexElement("Field"));

//Creating clones of prototypes

EnvironmentElement **newTree**=factory.getElement("basic element");

EnvironmentElement **newField**=factory.getElement("complex element");

**System**.***out***.println("Prototypes are set, here are deep-cloned methods (postfix"+

" '\_cloned' added):");

newTree.draw();

newField.draw();

((**ComplexElement**)newField).add(newTree);

newField.draw();

factory.addToDispatcher("tree and field", newField);

EnvironmentElement **newField1**=factory.getElement("tree and field");

**System**.***out***.println("Here I added a new prototype, cloned it DEEPly and here is a proof");

newField.draw();

newField1.draw();

}

}

**package** ua.kpi.ipz.lab8;

/\*\*

\* The class represents a basic, simple and non-complex element of environment.

\* From a point of view of composite pattern, it's a simple leaf-class. It can be

\* added to a complex element of environment.

\* The class implements an "EnvironmentElement" interface, so it has a structure of

\* a typical prototype: methods "clone" and "cloneDeep" allow a client class to create

\* new instances of this class without using a straight-forward new()-declaration.

\* **@author** Ruslan Popenko

\* **@version** 8.0

\*

\*/

**public** **class** **BasicElement** **implements** EnvironmentElement{

**private** **String** content;

/\*\*

\* Constructor that sets a content of basic element. It's supposed that client-class

\* knows nothing of the way to create this class.

\* **@param** content String, content of the class.

\*/

**public** **BasicElement**(**String** content){

**this**.content=content;

}

/\*\*

\* Constructor by default.

\*/

**public** **BasicElement**(){}

/\*\*

\* A shallow-clone method. Clones only an object's basic elements, but

\* doesn't clone the objects that may be aggregated in it.

\*/

**public** EnvironmentElement **clone**(){

**BasicElement** **copied**=**new** BasicElement();

copied.initialize(**this**.getContent());

**return** (EnvironmentElement)copied;

}

/\*\*

\* A deep-clone method. Clones both object and the other objects that

\* are aggregated in it.

\*/

**public** EnvironmentElement **cloneDeep**(){

**BasicElement** **copied**=**new** BasicElement();

**String** **copiedString**=**new** String(**this**.getContent().concat("\_cloned!"));

copied.initialize(copiedString);

**return** copied;

}

/\*\*

\* Returns the content

\* **@return** content content of basic element, String.

\*/

**public** **String** **getContent**(){

**return** content;

}

/\*\*

\* Initializes a basic element with a given content

\* **@param** content a content to be added to basic element.

\*/

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

**this**.content=content;

}

/\*\*

\* Prints an internal state of the BasicElement-object.

\*/

**public** **void** **draw**(){

**System**.***out***.println(content);

}

}

**package** ua.kpi.ipz.lab8;

**import** javax.naming.OperationNotSupportedException;

/\*\*

\* The class represents a complex element of environment.

\* From a point of view of composite pattern, it's a composite-element. It can be

\* added to another complex element of environment and it can also add another elements.

\* The class implements an "EnvironmentElement" interface, so it has a structure of

\* a typical prototype: methods "clone" and "cloneDeep" allow a client class to create

\* new instances of this class without using a straight-forward new()-declaration.

\*

\* **@author** Ruslan Popenko

\* **@version** 8.0

\*

\*/

**public** **class** **ComplexElement** **implements** EnvironmentElement{

**private** EnvironmentElement[] children;

**private** **String** content;

/\*\*

\* Constructor by default. Sets values of all parameters to 0 and null.

\*/

**public** **ComplexElement**(){}

/\*\*

\* Constructor sets a content of complex element.

\* **@param** content String, content of complex element.

\*/

**public** **ComplexElement**(**String** content){

**this**.content=content;

}

/\*\*

\* Constructor sets a child of complex element.

\* **@param** e an element that has to be aggregated in complex element.

\*/

**public** **ComplexElement**(EnvironmentElement e){

children=**new** EnvironmentElement[1];

children[0]=e;

}

/\*\*

\* Method allows to add a new element to be aggregated in complex element.

\* **@param** e en element to be aggregateed

\*/

**public** **void** **add**(EnvironmentElement e){

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

EnvironmentElement[] **buffer**=**new** EnvironmentElement[children.length+1];

**for**(**int** **i**=0; i<children.length; i++){

buffer[i]=children[i];

}

buffer[buffer.length-1]=e;

children=**new** EnvironmentElement[buffer.length];

children=buffer;

}

**else**{

children=**new** EnvironmentElement[1];

children[0]=e;

}

}

/\*\*

\* Method allows to remove an element on the given position.

\* **@param** pos position of an element

\* **@throws** OperationNotSupportedException

\*/

**public** **void** **remove**(**int** pos) **throws** **OperationNotSupportedException**{

**if**(children==**null**){

**throw** **new** OperationNotSupportedException();

}

EnvironmentElement[] **buffer**=**new** EnvironmentElement[children.length-1];

**if**(pos==0){

**for**(**int** **i**=1; i<children.length; i++){

buffer[i-1]=children[i];

}

children=**new** EnvironmentElement[buffer.length];

children=buffer;

}

**else** **if**(pos==children.length-1){

**for**(**int** **i**=0; i<children.length-1; i++){

buffer[i]=children[i];

}

children=**new** EnvironmentElement[buffer.length];

children=buffer;

}

**else** {

**for**(**int** **i**=0; i<pos; i++){

buffer[i]=children[i];

}

**for**(**int** **i**=pos; i<children.length-1; i++){

buffer[i]=children[i+1];

}

children=**new** EnvironmentElement[buffer.length];

children=buffer;

}

}

/\*\*

\* The method returns block in the

\* given position

\*/

**public** EnvironmentElement **getSubBlock**(**int** pos){

**if**(pos<children.length && pos>=0){

**return** children[pos];

}

**throw** **new** IndexOutOfBoundsException();

}

/\*\*

\* A shallow-clone method. Clones only an object's basic elements, but

\* does nothing to the objects that may be aggregated in it.

\*/

**public** EnvironmentElement **clone**(){

**ComplexElement** **copied**=**new** ComplexElement();

copied.setChildren(children);

copied.setContent(content);

**return** (EnvironmentElement)copied;

}

/\*\*

\* Method sets the content of complex element

\* **@param** cntnt a content to be set

\*/

**public** **void** **setContent**(**String** cntnt){

content=cntnt;

}

/\*\*

\* Sets classes that have to be aggregated in complex element.

\* **@param** elmts elements that have to be aggregated

\*/

**public** **void** **setChildren**(EnvironmentElement[] elmts){

children=elmts;

}

/\*\*

\* A deep-clone method. Clones both object and the other objects that

\* are aggregated in it.

\*/

**public** EnvironmentElement **cloneDeep**(){

**ComplexElement** **copied**=**new** ComplexElement();

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

EnvironmentElement[] **copiedChldrn**=**new** EnvironmentElement[children.length];

**for**(**int** **i**=0; i<children.length; i++){

copiedChldrn[i]=children[i].cloneDeep();

}

copied.setChildren(copiedChldrn);

}

**String** **copiedContent**=**new** String(content.concat("\_cloned"));

copied.setContent(copiedContent);

**return** copied;

}

/\*\*

\* Prints an internal state of the ComplexElement-object.

\*/

**public** **void** **draw**(){

**System**.***out***.print(" "+content+" , contents: ");

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

**for**(**int** **i**=0; i<children.length; i++){

**this**.getSubBlock(i).draw();

}

}

**else**{

**System**.***out***.print("null");

}

**System**.***out***.println();

}

}

**package** ua.kpi.ipz.lab8;

/\*\*

\* An interface for any environment-element.

\* Basically, makes the classes that implement this interface, perfect prototypes.

\* This allows to implement a prototype-pattern in this application.

\*

\* **@author** Ruslan Popenko

\* **@version** 8.0

\*

\*/

**public** **interface** EnvironmentElement {

/\*\*

\* A shallow-clone method. Clones only an object's basic elements, but

\* doesn't clone the objects that may be aggregated in it.

\* **@return** EnvironmentElement an element of environment.

\*/

**public** EnvironmentElement **clone**();

/\*\*

\* A deep-clone method. Clones both object and the other objects that

\* are aggregated in it.

\* **@return** EnvironmentElement a cloned element of environment.

\*/

**public** EnvironmentElement **cloneDeep**();

/\*\*

\* Prints the internal state of an object.

\*/

**public** **void** **draw**();

}

**package** ua.kpi.ipz.lab8;

**import** java.util.HashMap;

/\*\*

\* A class represents a factory of different Products with a function of

\* dispatchering.

\* Client can get s apecific Product by knowing the key, that is set to the specific

\* prototype (or simply an order to create)

\*

\* **@author** Ruslan Popenko

\* **@version** 8.0

\*

\*/

**public** **class** **EnvironmentFactory** {

**private** **HashMap**<String,EnvironmentElement> dispatcher=**new** HashMap<String,

EnvironmentElement>();

/\*\*

\* Method allows to request a clone of a prototype with a given key.

\* **@param** key key that is set to a specific prototype

\* **@return** EnvironmentElement clone of a prototype that is being requested.

\*/

**public** EnvironmentElement **getElement**(**String** key){

EnvironmentElement **el**=dispatcher.get(key);

**return** el.cloneDeep();

}

/\*\*

\* Method allows to add prototypes to the dispatchering-list.

\* **@param** key a key that has to be assigned to specific prototype

\* **@param** e a prototype that has to be added to a list.

\*/

**public** **void** **addToDispatcher**(**String** key, EnvironmentElement e){

dispatcher.put(key, e);

}

}

Результат роботи програми

Prototypes are set, here are deep-cloned methods (postfix '\_cloned' added):

Tree\_cloned!

Field\_cloned , contents: null

Field\_cloned , contents: Tree\_cloned!

Here I added a new prototype, cloned it DEEPly and here is a proof

Field\_cloned , contents: Tree\_cloned!

Field\_cloned\_cloned , contents: Tree\_cloned!\_cloned!