Софтуерни шаблони за проектиране (Software Design Patterns)

**Ханойски кули Project**

Диньо Динев.

F№ M24133

22.06.2013

София

Съдържание

[1. Изисквания за приложението 2](#_Toc359614910)

[2. Избор на технологии 2](#_Toc359614911)

[3. Осигуряване на качеството 2](#_Toc359614912)

[3.1. Отчет за качеството 3](#_Toc359614913)

[4. Стартиране на приложението 3](#_Toc359614914)

[5. Дизайн на системата 3](#_Toc359614915)

[6. Реализация на приложението 9](#_Toc359614916)

[Използвана литература: 11](#_Toc359614917)

# 

# Изисквания за приложението

Заданието за разработка на приложението е дефинирано по следния начин:

Да се създаде прототип на играта Ханойски кули с персистентни данни (запазване на статуса на играта, текущ, краен за даден именуван играч), с един потребител (играч) и едно административно приложение.

Освен изисквания към приложението има и няколко допълнителни изисквания за начина, по който трябва да е направен дизайна на системата. Това са:

1. В приложението трябва да се използват поне 4 GoF шаблона за проектиране.
2. От всяка една група (създаващи, структурни и поведенчески шаблони) да има поне един използван шаблон.
3. Използваните шаблони да са свързани по подходящ начин като структура и поведение.

# Избор на технологии

След анализ на изискванията беше взето решение да се разработи desktop приложение. За език за програмиране е избран Java. Това ще направи по-лесно използването на изучаваните шаблони за проектиране. За запазване на персистентни данни ще се използва xml файл, който ще се записва в папката на приложението. Потребителския интерфейс в Java е реализиран посредством Java Swing компоненти.

За билдване на системата ще се използва **Gradle** приложението, което осигурява бърз, лесен и гъвкав начин за конфигуриране, тестване и изпълнение. Основния скрипт за изпълнение е разположен в главния **build.gradle** файл в коренната директория на приложението, докато допълнителния скриптов файл, отговарящ за конфигурирането на плъгините за осигуряване на статичен анализ на кода се намират в **\config\quality.gradle.**

# Осигуряване на качеството

В приложението за осигуряване на по-добро качество наред с използването на 5-т шаблона за проектиране се използват и 3-ри средства за статичен анализ на кода, които **не се** припокриват в своите функции, а се допълват. Това са:

* **PMD** – код анализатор, който открива често срещани потенциални пропуски в кода
* **findBugs** – приложение за статичен анализ на Java код, който открива потенциални грешки в кода
* **Checkstyle** – приложение, което прави проверка за спазването на общоприети стандарти за кодиране на Java приложения

Посочените по-горе средства за осигуряване на качеството на кода, се използват като плъгини, които са лесно достъпни чрез **Gradle build** тула. Правилата, чрез които се задава поведението на тези плъгини е избрано да бъде default-ното, т.е. не са създавани специални правила за проверка, а са използвани общоприети.

# Отчет за качеството

След изпълнение на проверките за качеството:

>gradle clean check

Се генерират съответно в \build\reports , съответно **pmd**, **findBugs**, checkstyle директории с отчетите за евентуални нередности в кода, които трябва да бъдат отстранени. След направен задълбочен анализ, голяма част от проблемите, които съм счел за такива са отстранени.

# Стартиране на приложението

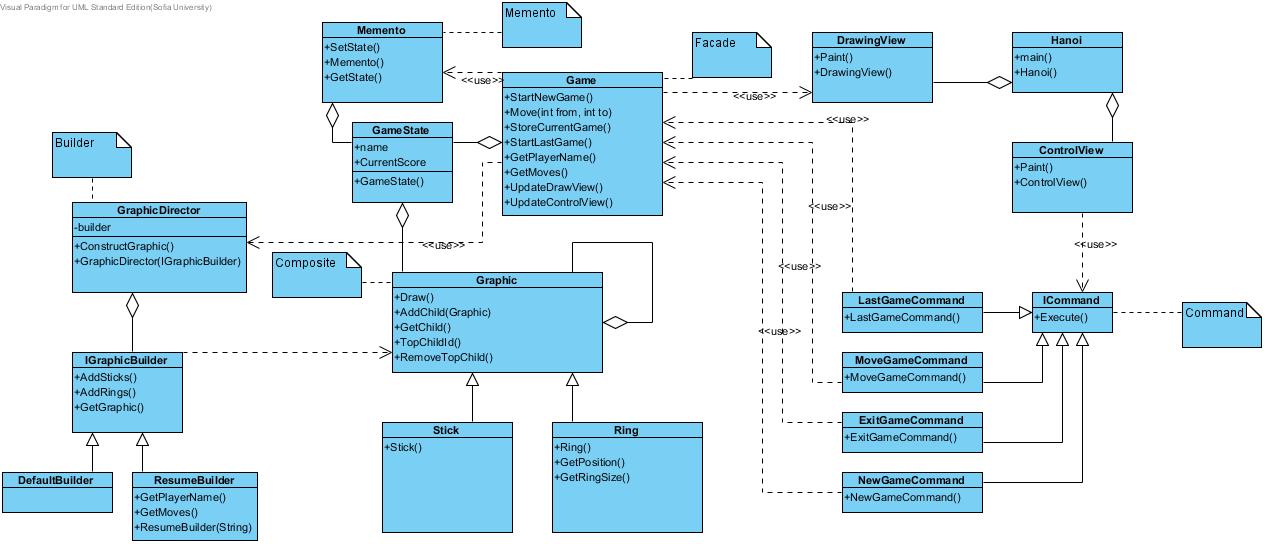
Необходимите конфигурации са отразени в скриптовете на проекта. За да се извърши компилиране и стартирането му е необходимо да се въведе следното в конзолен режим в кореннтата папка на проекта:

>gradle clean build main

След което се извършва зареждане на приложението.

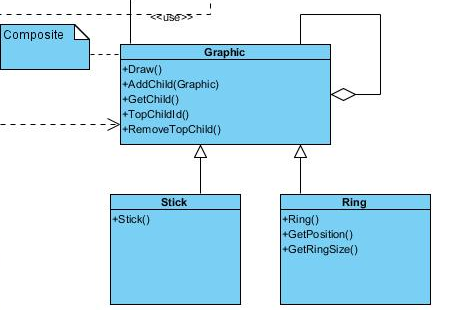
# Дизайн на системата

На *фигура 1* е показана клас диаграмата, която е реализирана в приложението:



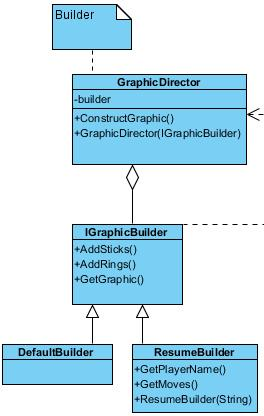
**Фиг. 1** Цялостна *клас* диаграма на приложението.

Изображенията в приложението се представят като йерархична структура, която е реализирана посредством шаблона **композиция (composite).** Най-отгоре в йерархията се намира класът Graphic. Той реализира някои базови методи като **addChild**, **removeChild** и др. Също така той съдържа структура с референции към същия обект. По този начин ще се реализира обект Graphic, който ще има 3 подобекта за стълбове (stick), които от своя страна ще съдържат кръгове (ring). По този начин с едно извикване на Draw метода на базовия клас ще се изчертае цялата структура. Частта от клас диаграмата, я която е реализиран шаблона композиция (composite) е показана на фигура 2.



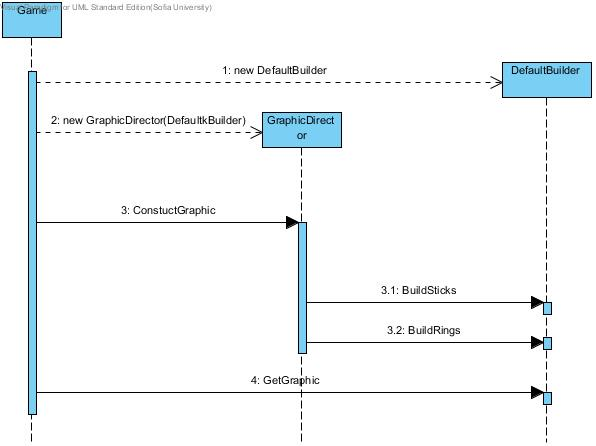
**Фиг. 2** Реализация на шаблона композиция.

За да се създаде тази сложна структура от данни е използват **шаблона строител**. Той има за цел да обособи създаването на структурата в отделен модул. Освен това шаблонът е реализиран посредством абстрактен клас за строител и два конкретни класа, които го наследяват. Това са **DefaultBuilder** и **ResumeBuilder**. Това кой от двата класа да се избере се прави по време на изпълнение и поради *тази причина* е използван този шаблон. Първият ще построи структурите в начално положение, докато втория ще изчете xml файл и ще създаде структура спрямо този файл. По този начин се зарежда последно, използваната игра. Частта от клас диаграмата, която реализира шаблонът строител е показана на Фиг. 3.



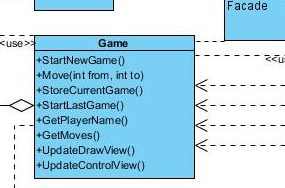
***Фиг. 3*** Реализация на шаблона строител.

Комуникацията между обектите при създаване е показана на фигура 4 посредством диаграма на последователността.



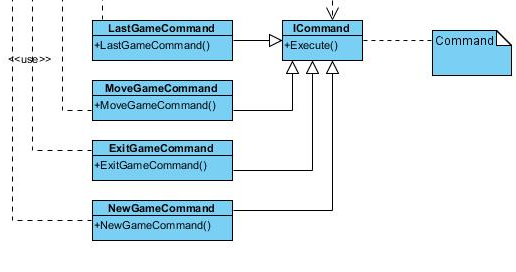
Фиг. 4 Диаграма на последователността при използване на строител.

Друг шаблон, който е използван в приложението е фасада. Класът Game служи като фасада, като за целта предоставя интерфейс от по-високо ниво, през които си комуникират компонентите отговарящи за потребителския интерфейс и останалите. Частта от клас диаграмата, която реализира фасада е показана на фигура 5.



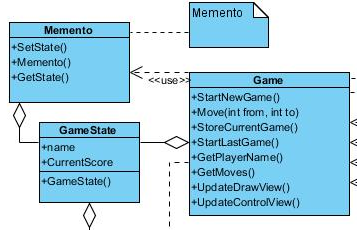
Фиг. 5 Реализация на фасада в приложението.

Поведенчески шаблони са използвани два. На първо място това е шаблонът команда. Той се използва за препредаване на командите по общ интерфейс към фасадата Game. Реализиран е абстрактен клас ICommand и четири команди наследници. Използването на шаблона се налага, защото при натискането на бутоните на приложението е необходимо да се изпращат заявки до обектите, без да се знае нищо за операцията. Реализацията на шаблона команда може да се види на Фиг. 6.



Фиг. 6 Реализация на команда в приложението.

Последния шаблон, който е реализиран в приложението е Memento (спомен). Той се използва, за да се удовлетвори изискването за персистентни данни. Чрез него се запазва текущото състояние на програмата в xml файл и след това може да се възстанови при повторно стартиране. Реализацията на спомен е показана на фигура 7.

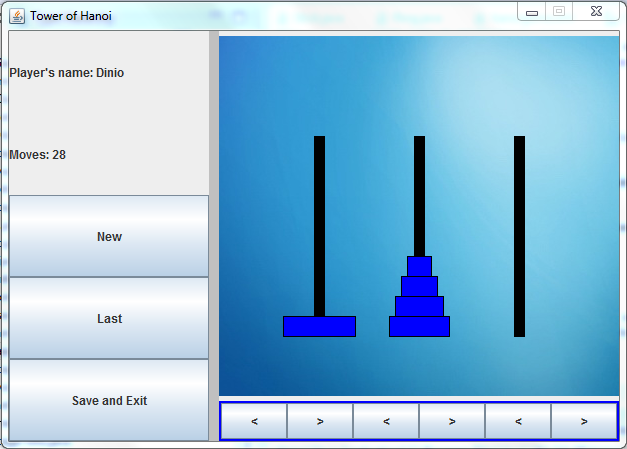


Фиг. 7 Реализация на шаблона спомен в приложението.

Това означава, че в цялото приложение са реализирани 5 шаблона за проектиране. Един за създаване, два структурни и два поведенчески.

# Реализация на приложението

Изглед на имплементираното приложението може да се види на фигура 8.



Фиг. 8 Общ вид на създаденото приложение.

Данните, които се записват в xml файл изглеждат по следния начин:

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<game>

<playerName>Player 1</playerName>

<moves>3</moves>

<stick>

<ring>5</ring>

<ring>4</ring>

<ring>3</ring>

</stick>

<stick/>

<stick>

<ring>2</ring>

<ring>1</ring>

</stick>

</game>

# Използвана литература:

1. <http://www.gradle.org/docs/current/userguide/pmd_plugin.html>
2. Gamma, Erich; Richard Helm, Ralph Johnson, and John Vlissides (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. ISBN 0-201-63361-2.
3. <http://findbugs.sourceforge.net/manual/index.html>
4. <http://www.gradle.org/docs/current/dsl/org.gradle.api.plugins.quality.Checkstyle.html>
5. Gabriel, Richard (1996). Patterns of Software: Tales From The Software Community (PDF). Oxford University Press. p. 235.ISBN 0-19-512123-6.