Домашна работа бр. 5
Предмет: Шаблони за дизајн на софтвер

RPN Calculator

Изработил: Стефан Трајковиќ
индекс: 149019

Содржина:

Анекс:

1. Задача (опис на домашната работа)

Да се реализира програма која симулира работа на специфичен програмабилен стек калкулатор (работи во инверзна полска нотација 1). Калкулаторот работи со децимални броеви и поддржува:

- (најмалку) четирите основни математички операции (+, -, *, /),
- CHS - за промена на предзнакот на бројот кој е на врвот на стекот
- DROP – за бришење на елементот од врвот на стекот
- SWAP – за промена на местата на двата последни елементи на врвот на стекот
- операциите UNDO и REDO (со неограничен број на нивоа),
- десет мемориски регистри во кои може да се чува по еден број за кој се задолжени операциите STO и RCL (се внесува STO и еден од тастерите 0-9 за избор на регистарот, исто и за RCL)
- можност за сместување на барем една програма (не подолга од 20 чекори) кои може одеднаш да се извршат со притискање на тастерот EXE; за снимање на секвенцата е задолжен е посебен тастер PROG,
- калкулаторот поддржува прикажување на последните 4 вредности на врвот на стекот

ЗА БОНУС!

- Да се реализираат основните тригонометриски операции SIN, COS, TAN и нивните инверзни операции (Inv тастер) и можност да се работи со степени и радијани (RAD/DEG тастер чија состојба е прикажана на дисплејот) ;

или

- Да се реализира посебен мод за работа со цели броеви со основните аритметички (+,- ,/,*) и логички (AND, OR, XOR, NOT) со приказ во броен систем кој се избира со еден тастер кој циклично ги менува (DEC, HEX, BIN, OCT)

2. Анализа на проблемот (задачата)

Во овој дел ќе направиме анализа на она како би требало калкулаторот да работи. Ќе ги разгледаме секоја команда подетално, и на почетокот ќе дадеме една скица (mockup) како би изгледал калкулаторот.

2.1 Mockup за изгледот на апликацијата

2.2 Објаснување на статусните линии (статусен панел)

- **Работна линија** - во работната линија со притискање на тастерите (0-9 и '.') се внесуваат броевите (операндите). Со притискање на Enter или со притискање на некој тастер со кој што се извршува некоја операција, операндот од работната линија се поставува (push) на стекот.

- **Стек линии** - има 4 линии кои ги репрезентираат првите 4 вредности на стекот.

- **RPN MODE статусна линија** - ја прикажува состојбата во која што се наоѓа калкулаторот. Калкулаторот може да се најде во една од четирите состојби BASIC, PROG, STO, RCL.

- **RAD/DEG статусна линија** - ја прикажува репрезентацијата на аглите со кои што работи калкулаторот (радијани или степени).

- **TRIG. FUNC. MODE статусна линија** - ја прикажува состојбата на извршување на тригонометриските функции (нормални или инверзни тригонометриски функции).

- **ERROR статусна линија** - ја прикажува грешката настаната од притискање на последната команда (тастер).

2.3 Објаснување на командите (команден панел)

- **0..9** - со притискање на еден од тастерите од 0 до 9 се внесува притиснатата цифра на работната линија

- **.** - со притисканје на тастерот за децимала, се додава децимала на бројот во работната линија.

- **CHS** - со оваа команда се менува предзнакот на бројот (+/-). Доколку работната линија е полна (има цифри) предзнакот се менува на работната линија. Доколку работната линија е празна, се проверува дали има некои броеви на стекот, доколку има се менува предзнакот на бројот кој што се наоѓа на првата позиција на стекот. Доколку е празна работната линија, а воедно и стекот е празен, тогаш се јавува грашка (Too Few Arguments).

- **+, -, *, /** - при притискање на една од основните аритметички операции најпрво се проверува дали е работната линија полна, доколку е вредноста се поставува (push) на стекот. Потоа се проверува дали има доволен број на операнди на стекот (поточно дали има две вредности на стекот). Доколку нема се јавува грешката “Too Few Arguments”, а во спротивно се извршува операцијата на следниов начин:

  STACK[1] [*operation*] STACK[0]

- **ENTER** – команда со која вредноста во работната линијасе поставува (push) на стекот.

- **SWAP** – операција со која првата и втората вредност на стекот си ги заменуваат местата. Доколку работната линија е полна, најпрво вредноста од нејзе се поставува на стекот, па потоа се извршува командата SWAP.

- **DROP** – го отфрла првата вредност на стекот. Доколку реботната линија е полна, најпрво се поставува вредноста на стекот, па потоа се извршува командата DROP.

- **AC** – со оваа команда се бришат вредностите кои се наоѓаат на работната линија, стекот и регистрите.

- **UNDO/REDO** – команди кои служат за враќање на извршените команди наназад (UNDO) и повторно извршување на претходно вратените команди (REDO).

- **Sin/Cos/Tng** – команди со кои се извршуваат основните тригонометриски функции. Вредноста врз која ќе се изврши командата зависи од тоа дали работната линија е полна. Доколку реботната линија е полна, операцијата ќе се изврши врз вредноста од работната линија, а во спротивно операцијата ќе се изврши врз првата вредност од стекот. Резултатот од овие команди зависи од состојбата во која се наоѓа калкулаторот (дали е во INV или NORMAL, како и дали е во RAD или DEG репрезентација на агол (подолу се објаснети)).

- **INV** – ја менува состојбата на калкулаторот за пресметување на тригонометриските функции. Калкулаторот може да биде во една од две состојби за пресметување на тригонометриски функции и тоа:
  - INVERS – пресметка на arcsin (аркус синус), arccos (аркус косинус) и arctng (аркус тангенс).
  - NORMAL – пресметка на sin (синус), cos (косинус) и tng (тангенс).

- **RAD/DEG –** ја менува репрезентацијата на аглите при работа со тригонометриските функции.

- **STO** – со оваа команда се внесуваат вредности во 10-те предефинирани регистри од 0 до 9. Од како ќе биде притисната командата STO калкулаторот влегува во STORE состојба и може да прими само команда за цифра од 0 до 9 (која го означува регистарот во кој треба да се сочува вредноста). Вредноста која треба да биде сочувана во регистарот се зема од работната линија (доколку е таа полна), во спротивно се зема првата бројка на стекот. Доколку нема вредност ниту во работната линија, ниту на стекот, тогаш се јавува грешката (Too Few Arguments). Доколку пак додека калкулаторот се наоѓа во STO состојба, а корисникот притисне команда која не е цифра (од 0 до 9) се појавува грешката (Invalid Operation).

- **RCL** – оваа команда идентично функционира како и STO, само што оваа команда има за цел, вредноста што се наоѓа во некој од регистрите да ја постави за прва вредност на стекот.

- **PROG** – со притискање на оваа команда калкулаторот влегува во PROGRAMMING состојба со што секоја наредна команда (тастер) која што ќе биде притисната, ќе биде забележена (сочувана) во низа, за подоцна запаметените команди да можат да бидат на еднаш извршени.
  - Со притискање на тастерот PROG се влегува и се излегува од PROGRAMMING состојбата.
  - Додека калкулаторот се наоѓа во PROG состојбата не е дозволено извршување на командата EXE како не би настанал бесконечен циклус. Доколку се притисне тастерот се јавова грешката “Invalid Operation”.

- **EXE** – со притискање на оваа команда, однапред снимената низа од команди со помош на PPROGRAMMING состојбата, се извршува атомично (како една команда).

3. Дизајн и имплементација на решението

Во овој дел најпрво ќе ја објасниме глобалната архитектура на апликацијата, како е таа поделена слоевито, а потоа ќе ја разгледаме дел по дел апликацијата, преку искористените шаблони, како работи и како е изградена.

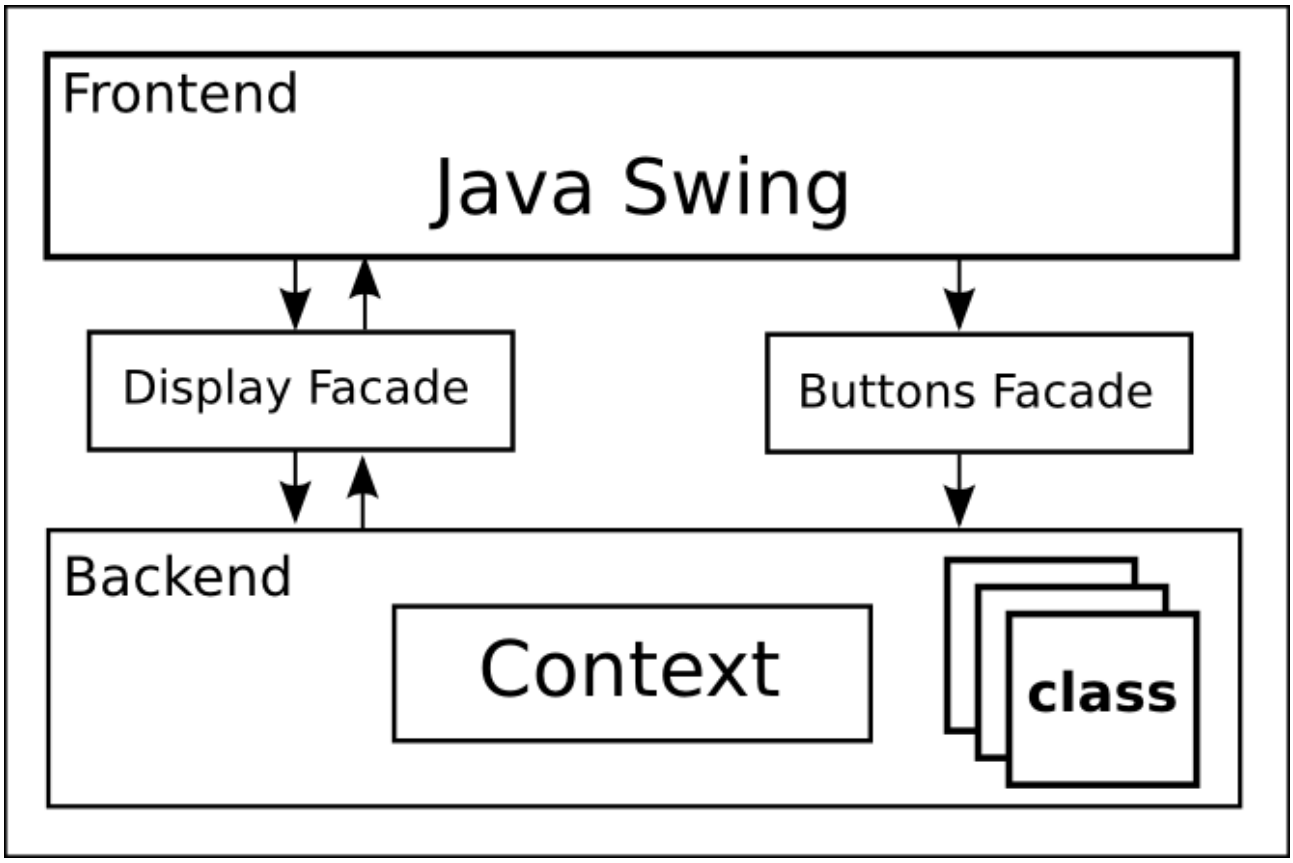
3.1 Архитектура на апликацијата

Поради барањето во задачата, корисничкиот интерфејс да биде независен од калкулаторот (да може лесно да се замени со друг), апликацијата е изградена слоевито, така што има 2 дела: Frontend – кој што е задолжен за изгледот и работата на корисничкиот интерфејс и Backend – кој што е задолжен за целата логика (работа) на калкулаторот.

Во сржта на Backend-от се наоѓа каласата Context која всушност ја чува моменталната состојба на сите делови на калкулаторот како што се: стекот, работната линија, глобалната состојба на калкулаторот (Basic, STO, RCL или PROG), репрезентацијата на аголот при извршување на тригонометриските функции итн.

Frontend-от работи со Backend-от преку две фасади:

- *Button Facade* – која што е задолжена за проследување на барањата (requests) кога ќе биде притиснато некое копче (тастер) и
- *Display Facade* – преку која може да се добијат информациите од Context класата како што се: стекот, работната линија, состојбите на калкулаторот итн.



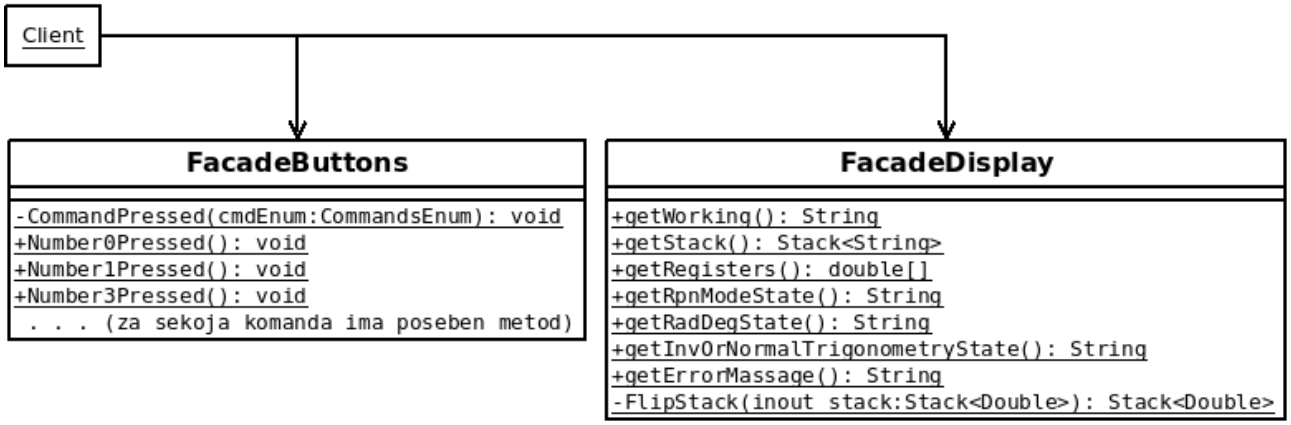
Моментално корисничкиот интерфејс (frontend-от) на апликацијата е направен со Java Swing, но многу лесно може да биде заменет со Java PrimeFaces или пак HTML и CSS.

3.2 Преглед на решението на апликацијата по шаблони

Во ова поглавје ќе бидат изложени класните дијаграми и објаснувањата на класите (која за што се користи). Приказот на класните дијаграми е поделен по искористените шаблони. Доколку сакате да го погледнете целосниот дијаграм на апликацијата, можете да го најдете во папката UML дијаграми. Објаснувањето за секој метод (за што се користи) може да се најде во изворниот код на апликацијата (во коментарите над секој метод).

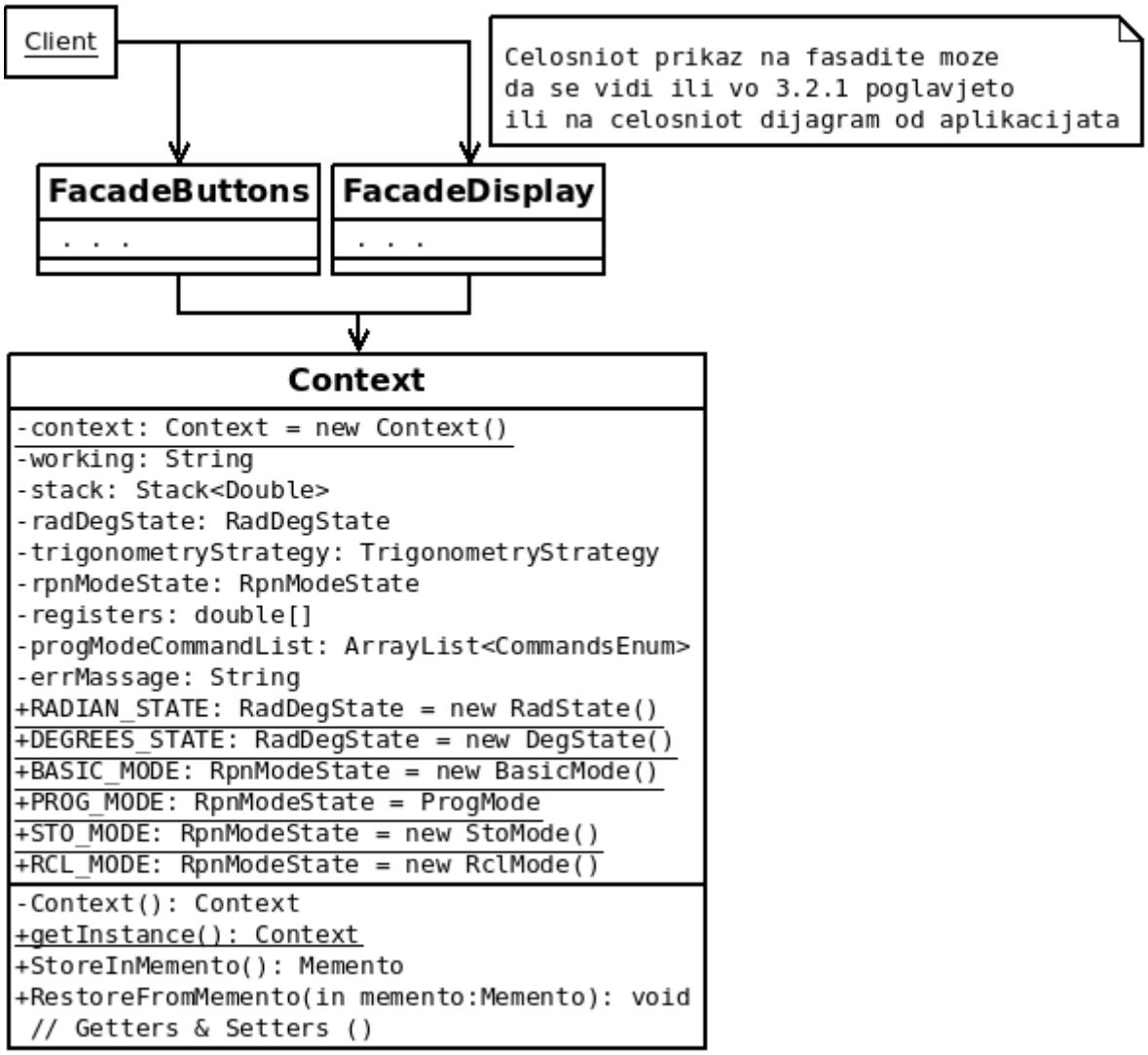
Најпрво ќе започнеме со FacadeButtons и FacadeDisplay класите (Facade Pattern) чие значење е објаснето погоре, потоа ќе следи Context класата (Singleton Pattern) која претставува сржта и главниот носител на апликацијата. Потоа ќе ги прикажеме главните состојби (RPN Modes) на калкулаторот: BASIC, PROG, STO и RCL (State Pattern). Командите кои што се извршуваат во зависност од состојбата (State Pattern-от) се превземаат (се градат) преку Abstract Factory Pattern-от. По генерирањето (превземањето) на командата, на ред доаѓа нејзиното извршување (Command Pattern), како и овозможувањето на враќање на состојбата на калкулаторот назад во состојбата пред да се изврши последната команда (undo), како и повторно нејзино извршување (redo) (Memento Pattern). На крај, приказ на Strategy Pattern-от кој се користи при извршување на тригонометриските функции за репрезентација на аглите (RAD/DEG) и самата функција (дали се работи за нормална тригонометриска функција или нејзината инверзна (пр. sin или arcsin)).

3.2.1 FacadeButton & FacadDisplay **(Facade Pattern)**



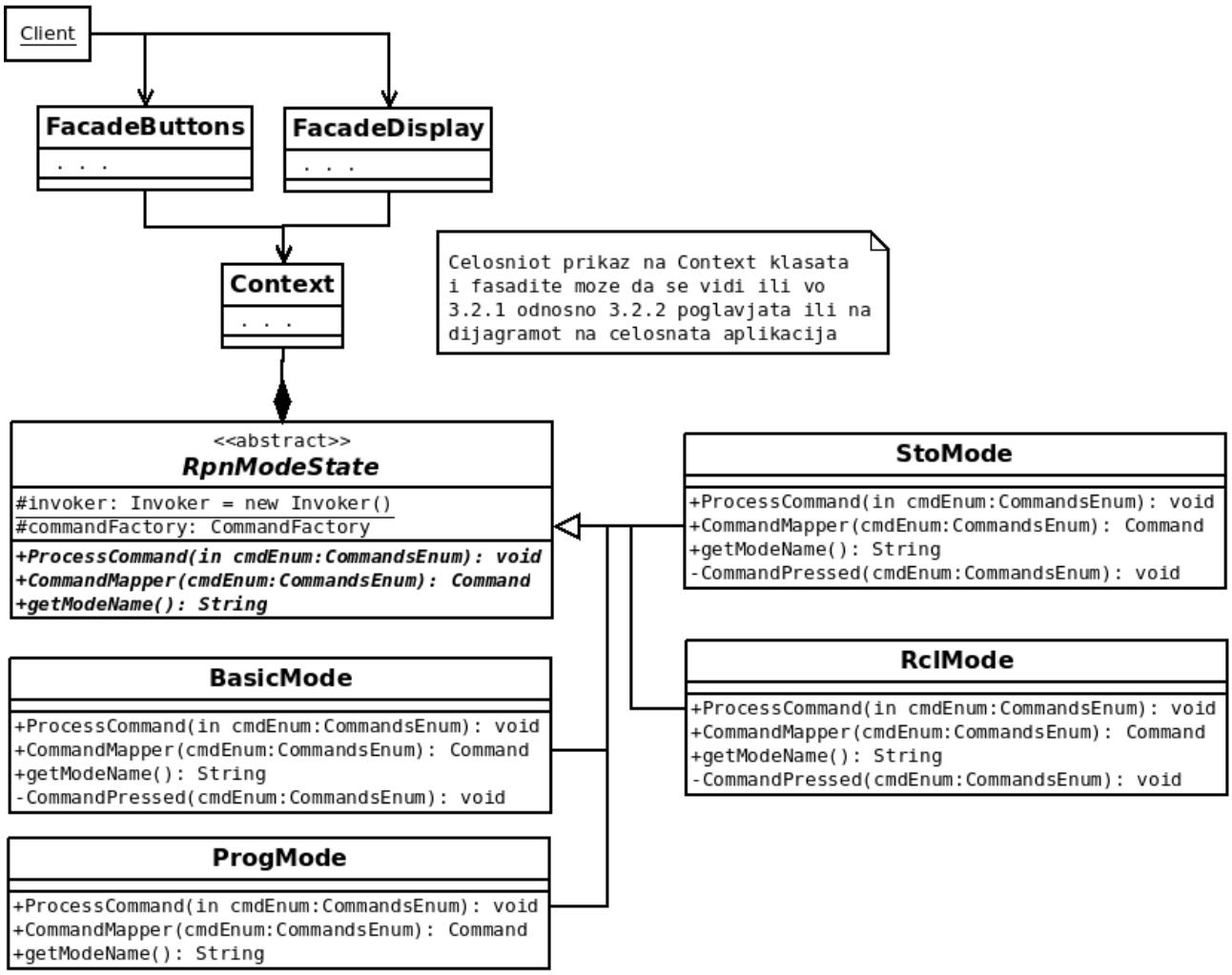
Име на класа	Опис на класа
FacadeButtons.java	Фасада која претставува интерфејс за работа со копчињата на апликацијата.
FacadeDisplay.java	Фасада која овозможува превземање на податоците, од Context класата (дефинирана подолу) , потребни за приказ на корисничкиот интерфејс.

3.2.2 Context **(Singleton Pattern)**



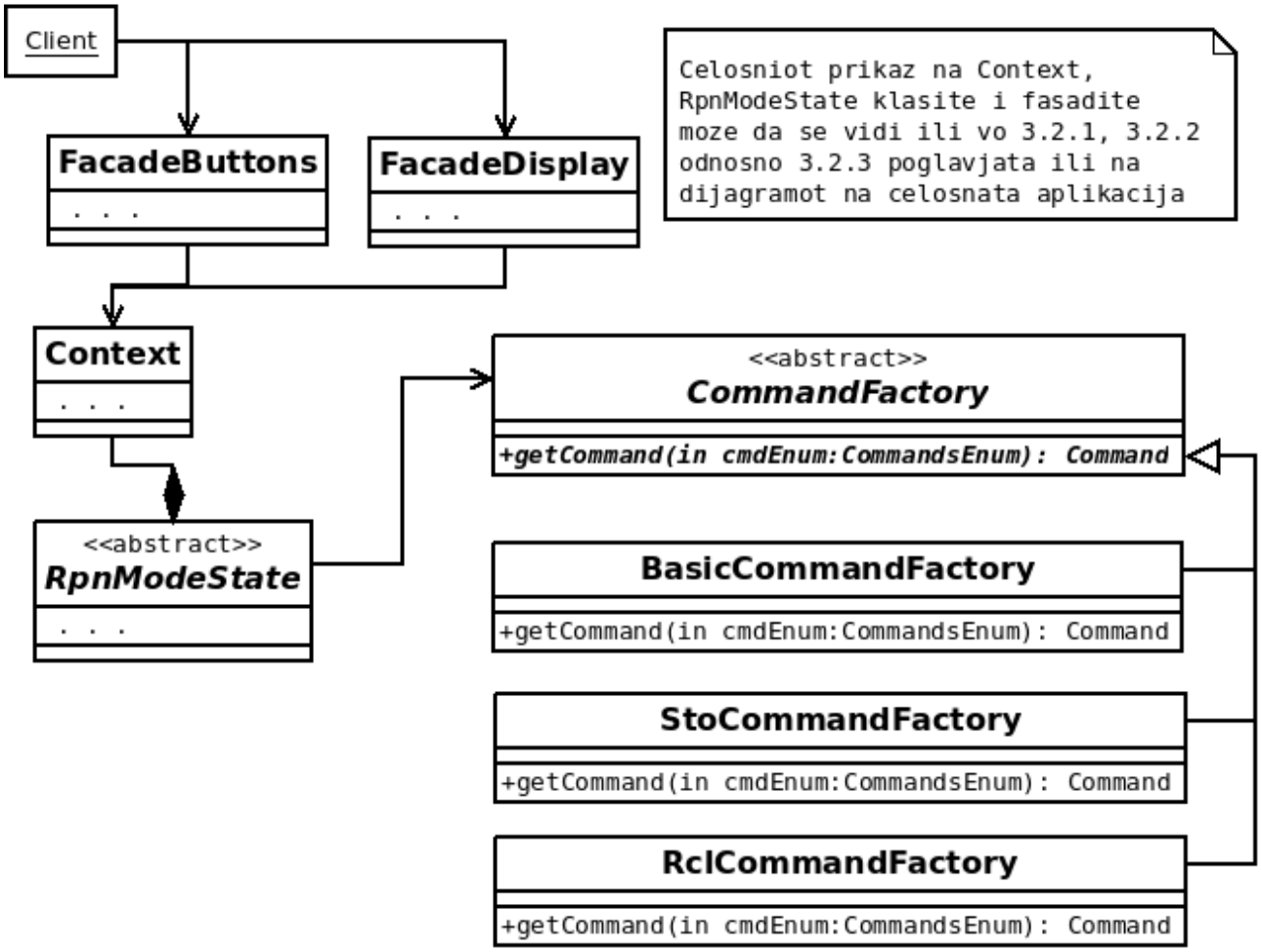
Име на класа	Опис на класа
Context.java	Оваа каласа ја претставува сржта на калкулаторот. Во нејзе се чува моменталната состојба на калкулаторот (целосната, глобалната состојба, се она што е потребно за да работи калкулаторот). Искористен е Singleton шаблонот бидејќи е потребно (смее) да има само една инстанца од оваа класа, бидејќи како што рековме погоре, таа е главниот носител на калкулаторот, односно врз нејзе се извршуваат сите операции.

3.2.3 RPN Mode - BASIC/PROG/STO/RCL **(State Pattern)**



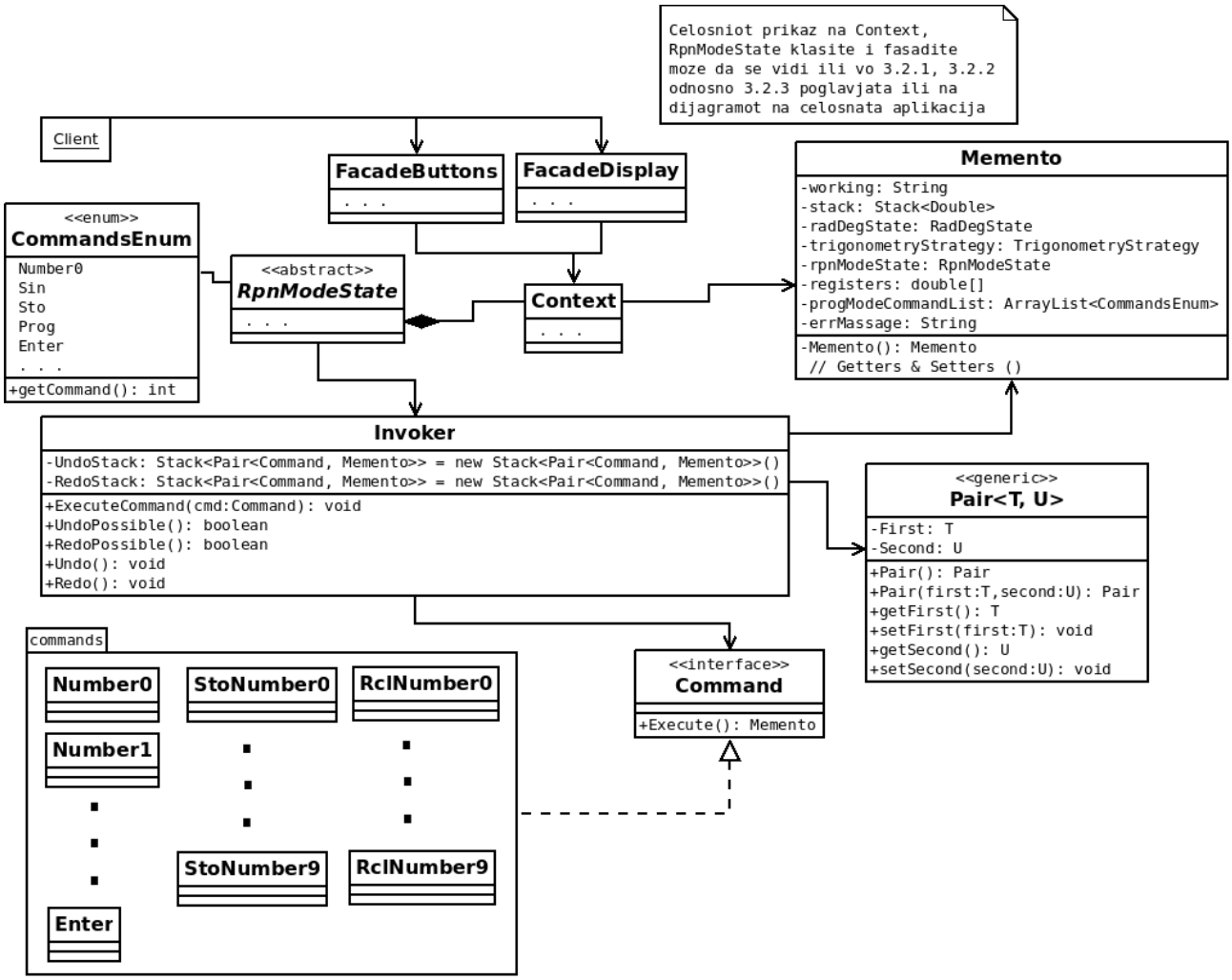
Име на класа	Опис на класа
RpnModeState.java	Ова е апстрактната класа кој ни служи полиморфички да ја менуваме глобалната состојбата на калкулаторот.
BasicMode.java	Оваа состојба ја претставува нормалната (базичната) состојба во која се наоѓа калкулаторот уште на самиот старт на апликацијата.
ProgMode.java	Оваа класа ја претставува состојбата во која калкулаторот влегува по притискање на копчето PROG т.е состојбата за програмирање.
StoMode.java	Оваа класа ја претставува состојбата во која калкулаторот влегува по притискање на копчето STO т.е состојбата за запишување на првата вредност од стекот во регистар.
RclMode.java	Оваа класа ја претставува состојбата во која калкулаторот влегува по притискање на копчето RCL т.е состојбата за отчитување на вредноста од некој регистар и нејзино запишување на врвот на стекот.

3.2.4 BASIC/PROG/STO/RCL - Command Factory **(Abstract Factory Pattern)**



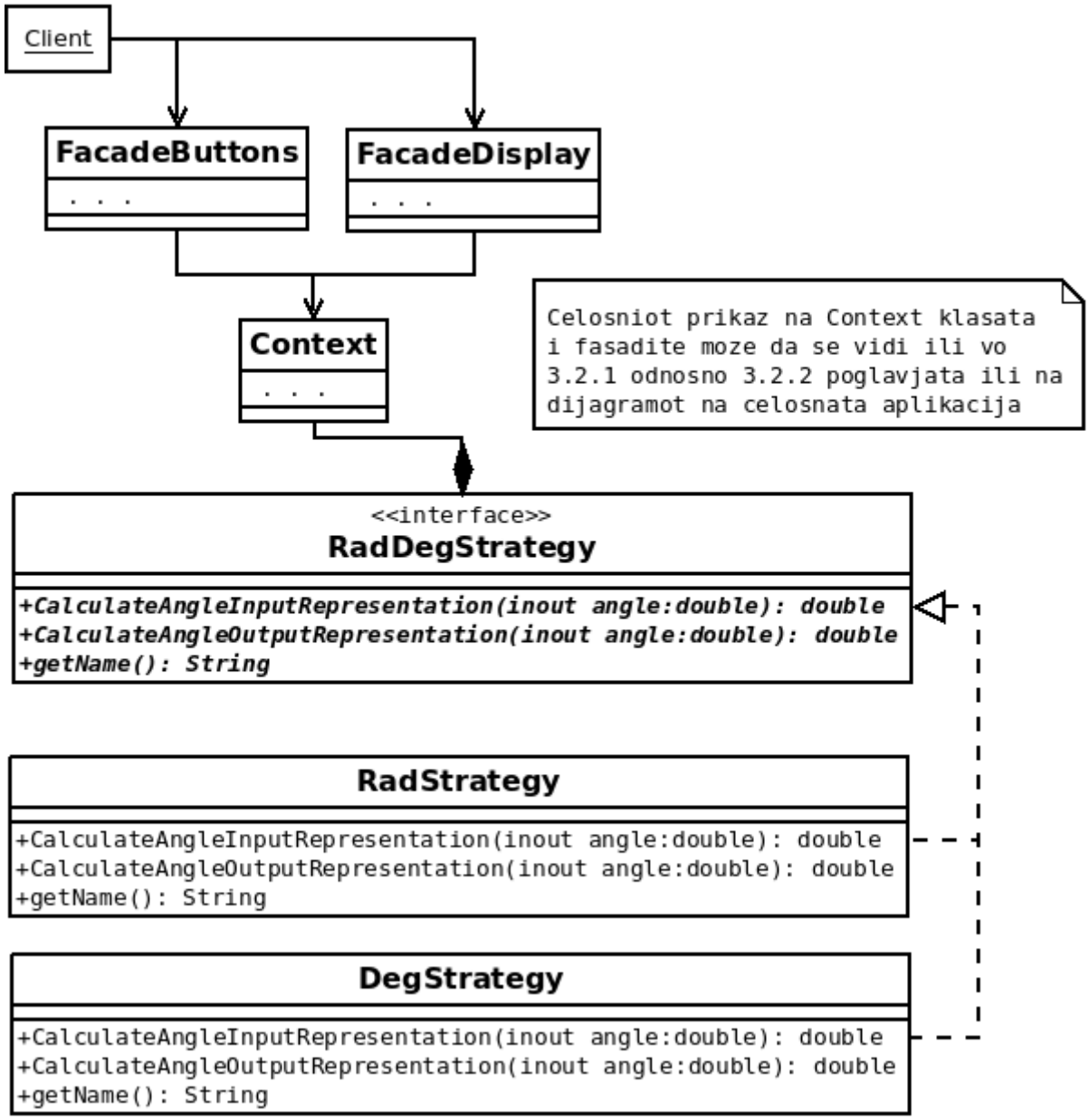
Име на класа	Опис на класа
CommandFactory.java	Апстрактна класа која ни овозможува полиморфно да ги менуваме “фабриките” за генерирање на командите.
BasicCommandFactory.java	“Фабрика” која ги генерира командите за калкулаторот кога се наоѓа во Basic состојбата.
StoCommandFactory.java	“Фабрика” која ги генерира командите за калкулаторот кога се наоѓа во Store состојбата.
RclCommandFactory.java	“Фабрика” која ги генерира командите за калкулаторот кога се наоѓа во Recall состојбата.

3.2.5 Command Execution & State **(Command/Memento Pattern)**



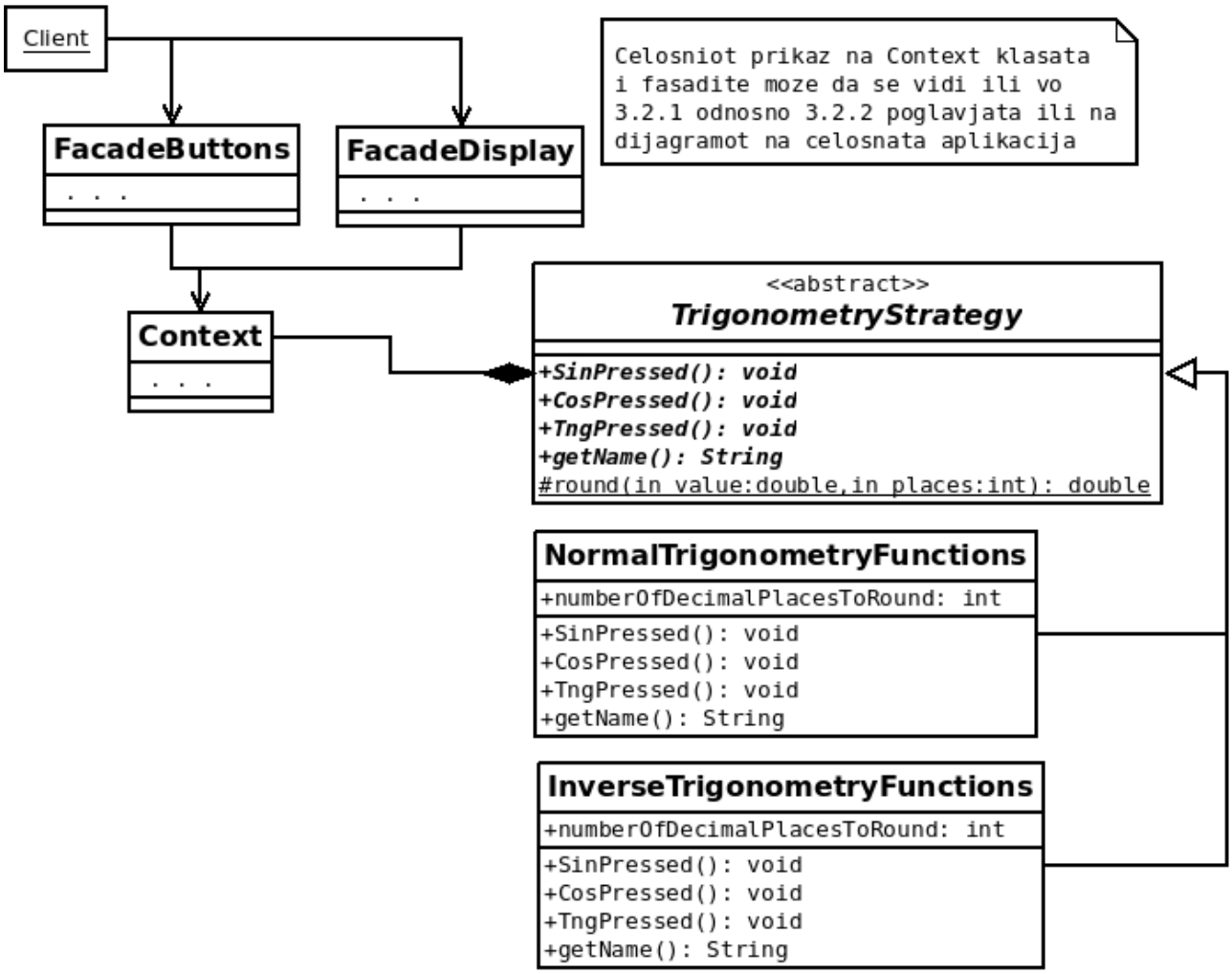
Име на класа	Опис на класа
Invoker.java	Invoker класата ја претставува инвокер класата од Command шаблонот. Таа е задолжена за извршување на командите, како и нивното чување во Undo и Redo стековите. Покрај тоа инвокер класата ја претставува и Caretaker класата од Memento шаблонот задолжена за чување на сите креирани мементо објекти.
Memento.java	Мементо класата ја претставува мементо класата од мементо шаблонот која претставува 'placeholder' класа за Context објектот.
Command.java	Ова е интерфејсот кој го наследуваат сите команди. Потребен е да можеме униформно да ги репрезентираме сите команди.
Pair.java	Pair класата претставува генеричка класа која овозможува да се градат објекти кои во себе ќе чуваат 2 објекта од две различни класи. Во нашиов случај потребно е да се чуваат парови од Command објект и Memento објект, и како такви да бидат зачувани во Undo и Redo стекот. Командата претставува последната команда извршена, а Memento-то претставува состојбата на калкулаторот пред да биде извршена командата.

3.2.6 (a) RAD/DEG **(Strategy Pattern)**



Име на класа	Опис на класа
RadDegStrategy.java	Ова е интерфејсот кој ни служи полиморфички да ја менуваме стратегијата за рапрезентација на аголот при извршување на тригонометриските функции.
RadStrategy.java	Оваа класа ја претставува стретегијата за пресметка на тригонометриските функции во радијани.
DegStrategy.java	Оваа класа ја претставува стретегијата за пресметка на тригонометриските функции во степени.

3.2.6 (b) NORMAL/INVERSE **(Strategy Pattern)**



Име на класа	Опис на класа
TrigonometryStrategy.java	Ова е апстрактната класа кој ни служи полиморфички да ја менуваме стратегијата за извршувањето на тригонометриските функции (нормална (пр. sin) и инверзна (пр. arcsin)).
NormalTrigonometryFunctions.java	Оваа класа ја претставува стретегијата за пресметка на нормалните тригонометриски функции.
InverseTrigonometryFunctions.java	Оваа класа ја претставува стретегијата за пресметка на инверзните тригонометриски функции.

4. Платформа и развојна околина

Платформа: Ubuntu 14.04 Linux OS
Развојна околина: Eclipse Luna
Програмски јазик: Java
Алатка за дизајнирање на UML дијаграми: Dia

Анекс:

А. Начин на стартување на апликацијата

За да се стартува апликацијата потребно е Executable JAR или Eclipce Project Zip, кои можете да ги најдете во папката 'Source'.

Постојат два начина на стартување на апликацијата:

- Стартување на Executable JAR
- Вметнување во Eclipse и стартување преку истиот

1. Стартување на Executable JAR :

Windows/Linux:

Во CommandLine се впишува : **java -jar <Pateka-do-JAR-Fajlot>**
пр. **java -jar SDSHomework5.jar**

Исто така може и на следниов начин:

Апликацијата за оваа домашна, која е изработена со Java Swing библиотеката за кориснички интерфејс, може да се стартува и со двоен клик на executable jar фајлот.

2. Вметнување во Eclipse и стартување преку истиот:

Windows/Linux/Mac:

Во Eclipse се притиска **File → Import → Existing Projects Into Workspace**

Потоа со десен клик на апликацијата се одбира **Run As → Java Application**