**ŽILINSKÁ UNIVERZITA V ŽILINE**  
 FAKULTA RIADENIE A INFORMATIKY



**Semestrálna práca 1**Predmet: Diskrétna simuláciaŠtudijný program: Informačné systémy

Vypracoval: Bc. Dávid Zimen Školský rok: 2023/2024   
Cvičiaci: Ing. Andrea Galadíková Študijná skupina: 5ZIS12

**OBSAH**

[1 Úloha 3](#_Toc161078813)

[2 Implementácia 3](#_Toc161078814)

[2.1 Generátory 3](#_Toc161078815)

[2.1.1 DiscreteUniformGenerator 4](#_Toc161078816)

[2.1.2 DeterministicGenerator 4](#_Toc161078817)

[2.1.3 ContinuousUniformGenerator 4](#_Toc161078818)

[2.1.4 ContinuousEmpiricalGenerator 4](#_Toc161078819)

[2.2 Jadro aplikácie 5](#_Toc161078820)

[2.2.1 SimulationCore 6](#_Toc161078821)

[2.2.2 MonteCarloCore 6](#_Toc161078822)

[2.2.3 MortgageMonteCarlo 6](#_Toc161078823)

[2.3 Užívateľské rozhranie 6](#_Toc161078824)

[3 Interpretácia výsledkov 7](#_Toc161078825)

# Úloha

Úlohou prvej semestrálnej práce bolo vytvoriť simulačný model s využitím metódy Monte Carlo. S týmto modelom bolo následne potrebné vykonať experimenty ohľadom hodnosti jednotlivých stratégií na doby fixácií pre hypotéku.

Medzi potrebné náležitosti taktiež patrilo vytvoriť všeobecné simulačné jadro pre metódu Monte Carlo a naprogramovať vlastný flexibilný generátor spojitého empirického rozdelenia.

Stratégie pre fixáciu hypotéky v hodnote 100 000€ na intervale 10 rokov boli nasledovné:

* A – fixácia 5 rokov, následne na 3 a potom dvakrát na 1 rok
* B – fixácia trikrát na 3 roky, následne na 1 rok
* C – fixácia na 3 roky, následne na 1 rok, potom na 5 rokov a ešte na 1 rok

# Implementácia

Pre implementáciu zadania bol zvolený programovací jazyk Kotlin, ktorý je postavený na platforme JVM a má schopnosť využívať knižnice naprogramované v jazyku Java, ale ponúka jednoduchšiu syntax a možnosť funkčného programovania.

## Generátory

Na nasledujúcom obrázku sú znázornené všetky druhy rozdelení pravdepodobností, ktoré bolo potrebné využiť v semestrálnej práci.

Obrázok, na ktorom je text, snímka obrazovky, číslo, písmo

Automaticky generovaný popisVlastný generátor bolo potrebné naprogramovať iba pre spojité empirické rozdelenie pravdepodobnosti. Avšak pre uľahčenie následnej implementácie, sme vytvorili nasledovnú štruktúru tried.

Obrázok 1.1 Potrebné typy rozdelení pravdepodobností

Obrázok, na ktorom je text, snímka obrazovky, písmo, diagram

Automaticky generovaný popisVšetky triedy implementujú spoločné rozhranie Generator, ktoré teda sprístupňuje iba metódu sample(), ktorá má vrátiť iba vygenerovanú hodnotu typu Double.

Obrázok 1.2 Diagram tried pre generátory

### DiscreteUniformGenerator

Pomocou atribútu generator, vygeneruje celé číslo z intrevalu <minValue; maxValue).

### DeterministicGenerator

V metóde sample() vracia exaktnú hodnotu zadanú užívateľom.

### ContinuousUniformGenerator

Vracia hodnotu z intervalu <minValue; maxValue) ako reálne číslo.

### ContinuousEmpiricalGenerator

Atribút mainGenerator vygeneruje hodnotu z intervalu <0; 1) a na základe tejto hodnoty sa vyberie z mapy generators príslušný objekt IntervalRandom, ktorý obsahuje príslušné hodnoty intervalu a inštanciu triedy Random a vráti vygenerovanú hodnotu z intervalu.

Pre overenie korektného naprograovania bolo vygenerovaných 500 000 hodnôt, ktoré boli spracované a pre jednotlivé intervaly boli porovnané teoretické a skutočné pravdepodobnosti.

Tabuľka 1 Porovnanie pravdepodobností

|  |  |  |  |
| --- | --- | --- | --- |
| **Interval** | **Počet vygenerovaných hodnôt** | **Skutočná pravdepodobnosť** | **Teoretická pravdepodobnosť** |
| <0,1 - 0,3) | 100103 | 0,100103 | 0,1 |
| <0,3 - 0,8) | 349288 | 0,349288 | 0,35 |
| <0,8 - 1,2) | 200670 | 0,20067 | 0,2 |
| <1,2 - 2,5) | 150362 | 0,150362 | 0,15 |
| <2,5 - 3,8) | 149541 | 0,149541 | 0,15 |
| <3,8 - 4,8) | 50036 | 0,050036 | 0,05 |

Obrázok 1.3 Graf porovnania pravdepodobností

Ako vyplýva z uvedenej tabuľky a grafu, tak rozdiel medzi teoretickou a skutočnou pravdepodobnosťou je na úrovní tisícin až desaťtisícin, takže generátor sa môže považovať za korektný.

## Jadro aplikácie

Obrázok, na ktorom je text, snímka obrazovky, písmo, list

Automaticky generovaný popisNa nasledujúcom obrázku je zobrazený diagram tried pre backend aplikácie.

Obrázok 1.4 Jadro aplikácie

### SimulationCore

Všeobecné simulačné jadro, ktoré poskytuje metódy runSimulation, stopSimulation, pauseSimulation a resumeSimulation. Trieda nie je abstraktná, avšak všetky metódy sú prázdne, takže pre konkrétnu implementáciu je potrebné ich prekrytie.

### MonteCarloCore

Všeobecné jadro pre statické simulovanie metódou Monte Carlo. Dedí od triedy SimulationCore, ale prekrýva iba metódy runSimulation a stopSimulation. PauseSimulation a resumeSimulation nie sú v našom prípade potrebné.

Samotná simulácia sa spúšťa zavolaním metódy runSimulation, kde v jej tele sa najskôr vykoná metóda beforeReplications, v cykle od 0 po hodnotu replicationsCount následne vykonáva metódy beforeReplication, replication a afterReplication. Po prejdení cyklu ešte vykoná metódu afterReplications.

Zastavenie sa vykoná zavolaním stopSimulation, kedy sa atribút simulationRunning nastaví false a metóda runSimulation na základe hodnoty atribútu preruší vykonávanie simulácie.

Metódy runSimulation a stopSimulation sú označené ako final, takže nie sú dostupné pre prekrytie. Na konkrétnu implementáciu si používateľ môže prekryť metódy beforeReplications, beforeReplication, replication, afterReplication a afterReplications.

Ponúka aj atribút typu Observable<Any>, kde prípadná implementácia môže poskytovať aktuálny stav, ktorý môžu následne užívatelia spracovávať, tak ako im je vhodné. Využitý je tu návrhový vzor Observer.

### MortgageMonteCarlo

Konkrétna implementácia simulácie pre potreby zadania. Dedí od triedy a prekrýva všetky dostupné metódy.

* BeforeReplications – sú vynulované všetky hodnoty a vytvorené generátory
* BeforeReplication – stav simulácie sa pripraví na rátanie
* Replication – výpočet zaplatenej sumy pre danú replikáciu
* AfterReplication – aktualizácia stavu simulácie
* AfterReplications – poskytnutie výsledku

## Užívateľské rozhranie

Na tvorbu užívateľského rozhrania bola použitá technológia JavaFX. Pre účely vykresľovania grafov sa využila knižnica JFreeChart.

K dispozícií sú 3 grafy, pričom každý z nich predstavuje jeden scenár zo zadania. Vpravo hore je možné zadať počet replikácii a výšku istiny pre hypotéku. Naľavo od týchto polí sa následne nachádzajú tlačidlá na spustenie, zastavenie simulácie a odstrihnutie prvých 10% hodnôt z grafu, ktoré môžu pre fázu zahrievania ovplyvniť čitateľnosť grafu.

# Obrázok, na ktorom je text, snímka obrazovky, diagram, písmo Automaticky generovaný popisInterpretácia výsledkov

Obrázok 1.5 Ukážka užívateľského rozhrania

Simuláciu sme spustili s nastavením na 50 000 000 replikácií a dostali sme nasledovné výsledky:

* Stratégia A = 111 657,74392€
* Stratégia B = 112 367,30672€
* Stratégia C = 111 052,38993€

Na základe takto dosiahnutých výsledkov môžeme skonštatovať, že najvhodnejšie je si vybrať stratégiu C, teda fixáciu na 3 roky, následne na 1 rok, potom na 5 rokov a ešte na 1 rok