Bank Simulator

24067 박성현

목차

I. 주제 선정 동기

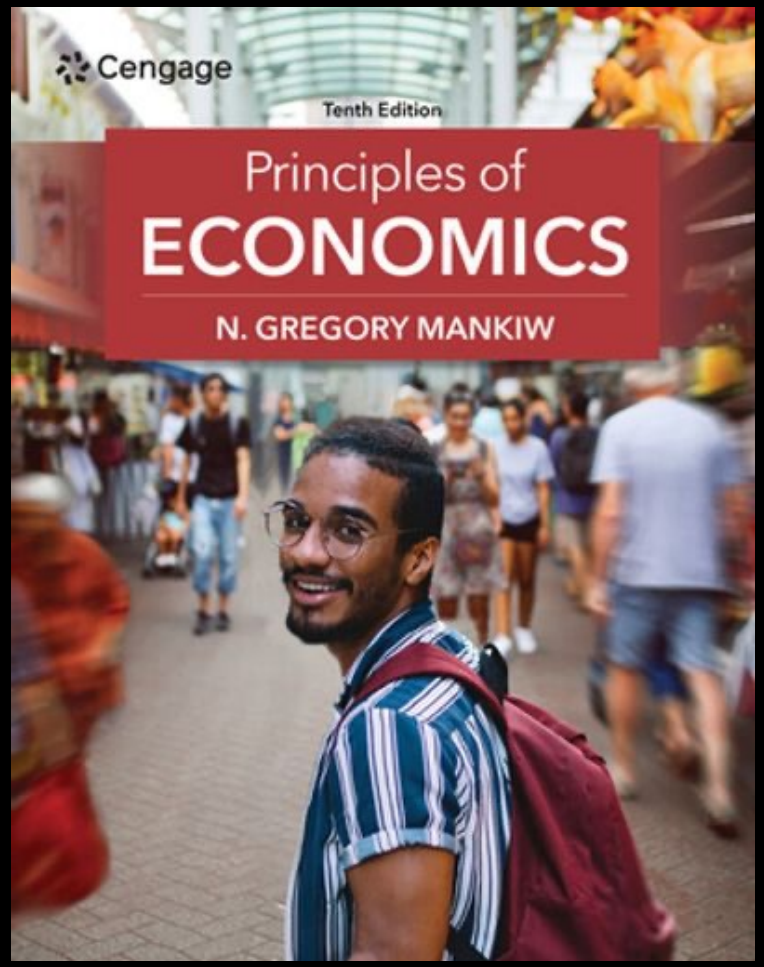
II. 핵심 코드 설명

III. 코드 시연

IV. 어려움 극복 사례 & 느낀점

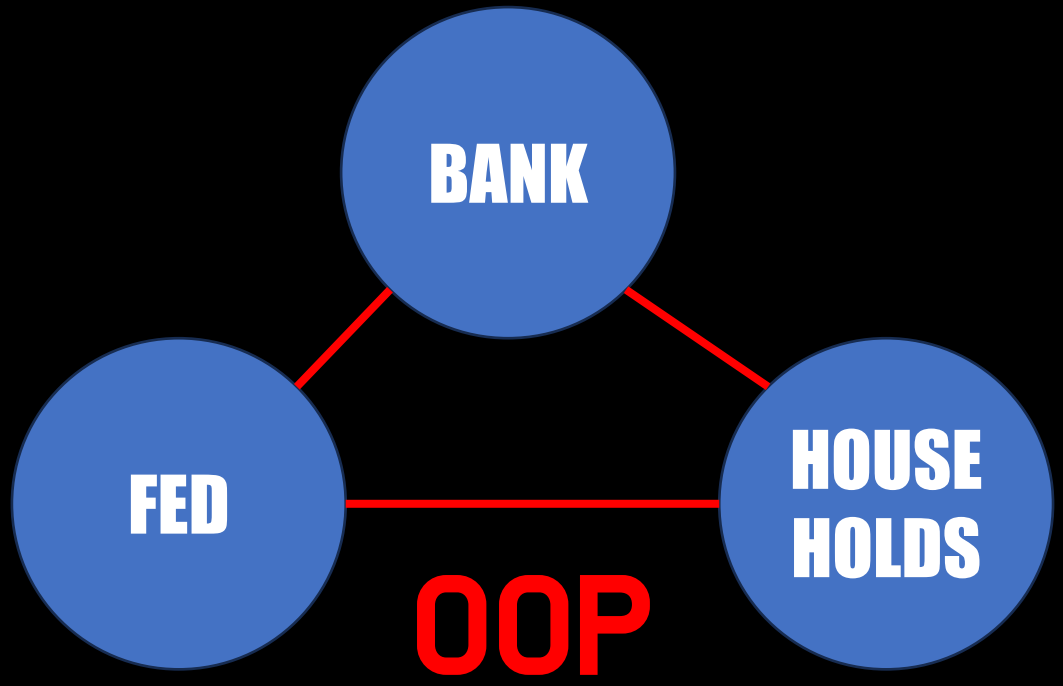
Picture Retrieved From https://tavexbullion.co.uk/what-makes-a-bullion-coin-rare/

I. 주제 선정 동기



Picture Retrieved From https://www.amazon.com/Principles-Economics-N-Gregory-Mankiw/dp/035772271X

AP MACROECONOMICS 중간고사



II. 핵심 코드 설명

부분지급준비제도

-> 은행이 예금의 일부(지급준비금)만 보유하고 나머지는 대출하는 제도.

지급준비금 = Reserve

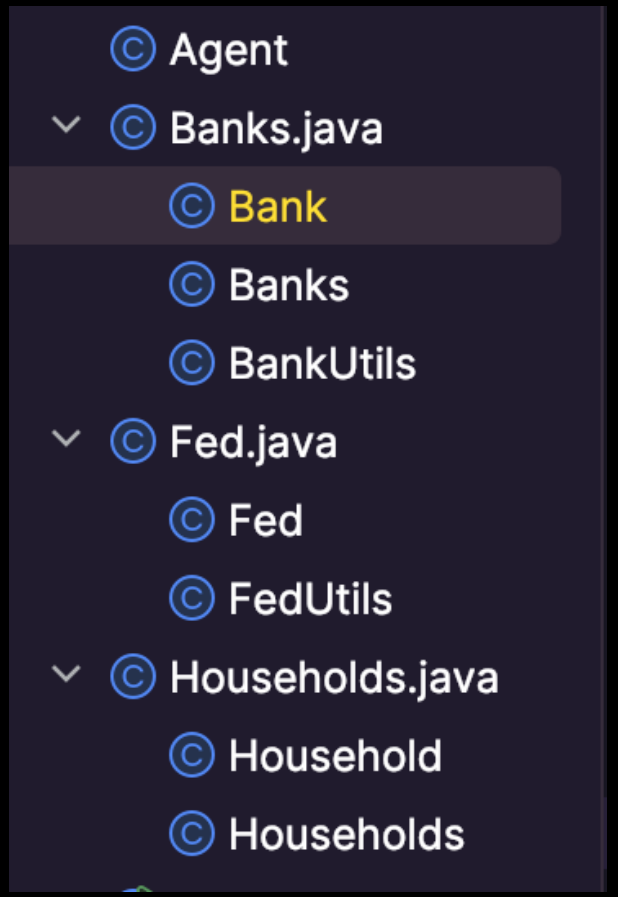
대출 = Loan

민간이 예금/적금 = Deposit

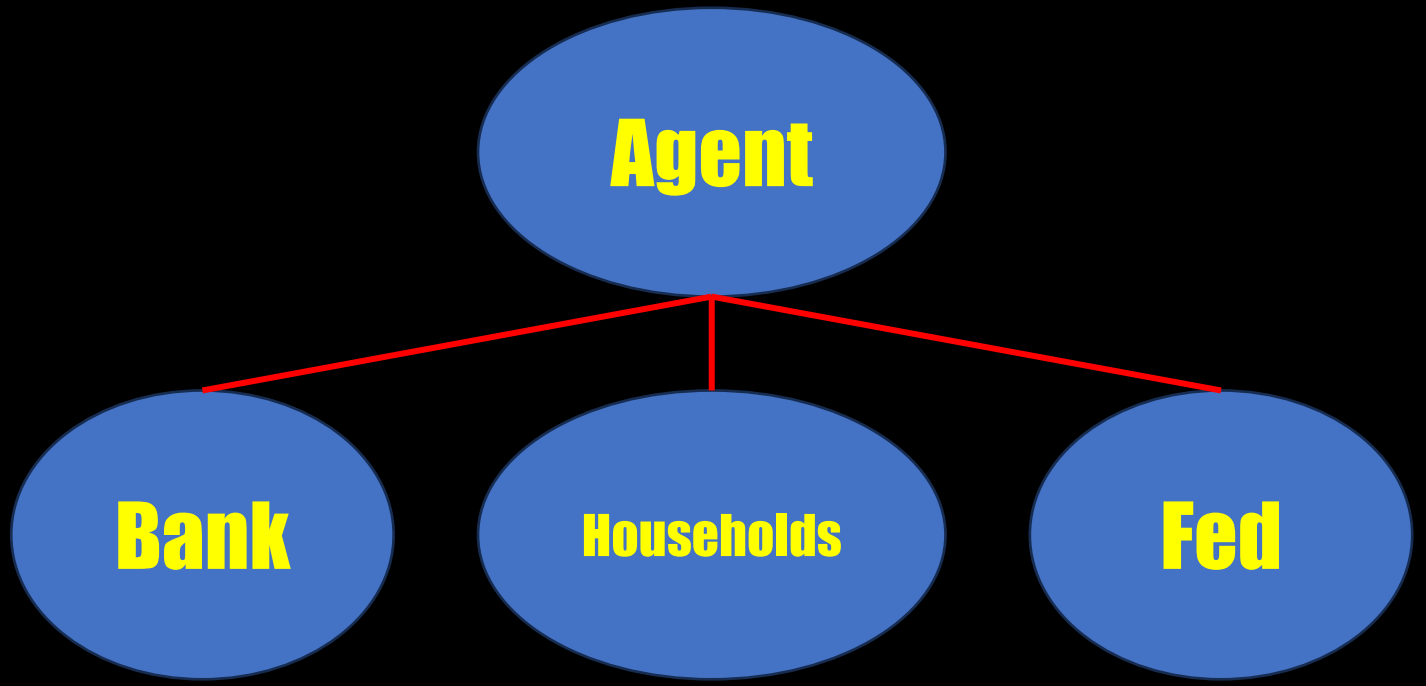
이에 대한 비율을
FED가 정함

HOUSEHOLD, FED, BANK
3개의 주체를 가진 부분준비지급제도
Simulation!

II. 핵심 코드 설명



〈상속 구조〉



II. 핵심 코드 설명

Agent

➔ 경제 주체의 기본 틀. 공통적인 정보를 담음
(직접 사용 X)

Bank

➔ 일반 은행. 가장 핵심적인 클래스이며,
Bank.Util에 여러 상황의 대출, 이자 갱신,
FED와의 상호작용을 넣음

II. 핵심 코드 설명

Agent

➔ 경제 주체의 기본 틀. 공통적인 정보를 담음
(직접 사용 X)

Bank

➔ 일반 은행. 가장 핵심적인 클래스이며,
Bank.Util에 여러 상황의 대출, 이자 갱신,
FED와의 상호작용을 넣음

II. 핵심 코드 설명

```java
public class Agent {  5 usages  3 inheritors
    String name;  5 usages
    int id;  46 usages
    float total_money = 1000000F;  3 usages

    Agent(int id){  no usages
        this.id = id;
    }
    Agent(int id, String name){  8 usages
        this.id = id;
        this.name = name;
    }

    Agent(int id, String name, float total_money){  3
        this.id = id;
        this.name = name;
        this.total_money = total_money;
    }
}
```

II. 핵심 코드 설명

```java
public class Fed extends Agent{ 4 usages
    float dRate = 0.04F; //Discount Rate, 0 <= <= 1  2 usages
    float rRatio = 0.03F; //Reserve Ratio, 0  <= <= 1  4 usages
    ArrayList<float[]> relations = new ArrayList<float[]>(); //{{Agent_id, 빌린값, 빌려준값}}  7 usages

    Fed(){  1 usage
        super( id: 0,  name: "Fed",  total_money: 6587000000000F); //Only one fed instance can exist.
    }

    JSONObject changeRR(float newRR) {  1 usage
        JSONObject jO = new JSONObject();
        if (newRR >= 0 || newRR <= 1) {
            jO.put("Success", true);
            jO.put("Message", "Reserve Ratio updated succesfully.");
            rRatio = newRR;
        }
        else{
            jO.put("Success", false);
            jO.put("Message", "Invalid Reserve Ratio.");
        }
        return jO;
    } //Changing reserve ratio

    JSONObject changeDR(float newDR) {  1 usage
        JSONObject jO = new JSONObject();
        if (newDR >= 0 || newDR <= 1) {
            jO.put("Success", true);
            jO.put("Message", "Discount Rate updated succesfully.");
            dRate = newDR;
        }
        else{
            jO.put("Success", false);
            jO.put("Message", "Invalid Discount Rate.");
        }
```

```java
class Bank extends Agent{  34 usages
    //id is Auto incremented, <= 500
    String name;  7 usages

    float reserve = 50000000000F; //$, default is $500B  26 usages
    float loans = 0;  //$, default is $0  16 usages
    float deposit = 50000000000F; //$, default is $500B  19 usages
    float equity = 100000000000F; //$, default is $1000B  22 usages
    float loanRate = 0.023F;  1 usage
    float depositRate = 0.03F;  1 usage
    float total_money = this.reserve + this.loans + this.deposit + this.equity;  22 usages
    ArrayList<float[]> relations = new ArrayList<float[]>();  //{{Agent_id, 빌린값, 빌려준값}}  29 usages

    Bank(String name){  1 usage
        super( id: -1, name);
    }
```

```java
class Household extends Agent{  14 usages
    float total_money = 1000000000000000F; //$  6 usages
    ArrayList<float[]> relations = new ArrayList<float[]>(); //{{Agent_id, 빌린값, 빌려준값}}  21 usages

    Household (String name) {  1 usage
        super( id: -1, name);
    }

    Household (int id, String name){  no usages
        super(id, name);
    }

    Household (String name, float total_money){  no usages
        super( id: -1, name, total_money);
    }

    Household (int id, String name, float total_money){  no usages
        super(id, name, total_money);
    }
}
```

II. 핵심 코드 설명

```
class BankUtils{  2 usages
    void deposit(float amount, Household house, Bank bank) { //deposit from households to bank  1 usage
        boolean house_exist = true;
        for (int i = 0; i < house.relations.size(); i++) { //check if bank is already in householdsl relations list
            if (house.relations.get(i)[0] == bank.id) {
                house.relations.get(i)[2] += amount;
                house_exist = false;
            }
        }

        boolean bank_exist = true;
        for (int i = 0; i < bank.relations.size(); i++) {  //check if household is already in banks' relations list
            if (house.relations.get(i)[0] == house.id)
                house.relations.get(i)[1] += amount;
                bank_exist = false;
        }

        if (house_exist){ //Adding new id
            float[] info = {bank.id, 0F, amount};
            house.relations.add(info);
        }

        if (bank_exist) { //Adding new id
            float[] info2 = {house.id, amount, 0F};
            bank.relations.add(info2);
        }

        house.total_money -= amount;
        bank.total_money += amount;

        bank.reserve += amount;
        bank.deposit += amount;
    }
```

```
void giveLoan(float amount, Household house, Bank bank) { //Loan from bank to households  1 usage
    if (amount <= 0) return;
    if (bank.reserve < amount) return;

    //similar structure (deposit class)
    boolean house_exist = true;
    for (int i = 0; i < house.relations.size(); i++) {
        if (house.relations.get(i)[0] == bank.id) {
            house.relations.get(i)[1] += amount;
            house_exist = false;
        }
    }

    boolean bank_exist = true;
    for (int i = 0; i < bank.relations.size(); i++) {
        if (house.relations.get(i)[0] == house.id)
            house.relations.get(i)[2] += amount;
            bank_exist = false;
    }

    if (house_exist){
        float[] info = {bank.id, amount, 0F};
        house.relations.add(info);
    }

    if (bank_exist) {
        float[] info2 = {house.id, 0F, amount};
        bank.relations.add(info2);
    }

    house.total_money += amount;
    bank.total_money += amount;
    bank.reserve -= amount;
```

```
JSONObject borrowFromBank(float amount, Bank lender, Bank borrower) { //a bank borrowing from another bank  1 usage
    JSONObject j = new JSONObject();

    //also similar to repayLoan function
    if (amount <= 0){
        j.put("Message", "Negative Amount");
        j.put("Success", false);
        return j;
    }
    if (lender.total_money < amount) {
        j.put("Message", "Not enough money in lending bank.");
        j.put("Success", false);
        return j;
    }

    lender.loans -= amount;
    borrower.reserve += amount;

    lender.total_money -= amount;
    borrower.total_money += amount;

    boolean lend_exist = true;
    for (int i = 0; i < lender.relations.size(); i++) {
        if (lender.relations.get(i)[0] == borrower.id){
            lender.relations.get(i)[2] += amount;
            lend_exist = false;
        }
    }

    boolean borrow_exist = true;
    for (int i = 0; i < borrower.relations.size(); i++) {
        if (borrower.relations.get(i)[0] == lender.id){
            borrower.relations.get(i)[1] += amount;
```

```
        lender.loans += amount;
        borrower.reserve -= amount;

        lender.total_money += amount;
        borrower.total_money -= amount;

        boolean lend_exist = true;
        for (int i = 0; i < lender.relations.size(); i++) {
            if (lender.relations.get(i)[0] == borrower.id){
                lender.relations.get(i)[2] += amount;
                lend_exist = false;
            }
        }

        boolean borrow_exist = true;
        for (int i = 0; i < borrower.relations.size(); i++) {
            if (borrower.relations.get(i)[0] == lender.id){
                borrower.relations.get(i)[1] += amount;
                borrow_exist = false;
            }
        }

        if (lend_exist){
            float[] info = {borrower.id, 0F, amount};
            lender.relations.add(info);
        }

        if (borrow_exist){
            float[] info = {lender.id, amount, 0F};
            borrower.relations.add(info);
        }
```

```
JSONObject handleBankRun(Bank bank, float runRatio) { //runRatio는 예금 중 몇퍼 인출되는지, this method is a protocol for bankrun.
    JSONObject j = new JSONObject();
    if (runRatio <= 0f){
        j.put("Message", "run ratio is negative or zero");
    }
    if (runRatio > 1f) runRatio = 1f;
    float withdraw = bank.deposit * runRatio;

    if (bank.reserve >= withdraw) { //as reserve is larger than the amount people want to withdraw, the bank isn't bankrupt
        bank.reserve -= withdraw;
        bank.deposit -= withdraw;
        j.put("Message", "BankRun handled with reserve");
        return j;
    } else {
        float shortage = withdraw - bank.reserve;

        bank.reserve = 0f;
        bank.deposit -= withdraw;
        bank.equity  -= shortage;

        if (bank.equity < 0f) { //in this case, bankrun is out of banks' capability, so bank is bankrupt
            j.put("Message", bank.name + "bank couldn't handle the bankrun, so it is bankrupt,");
            bank.bankrupt();   //은행 파산임
        } else //Can be solved with equity (거의 최후 수단)
            j.put("Message", "Reserve was lacking, but bankrun is covered with equity.");
    }

    bank.total_money = bank.reserve + bank.loans + bank.deposit + bank.equity;
    return j;
}
```

II. 핵심 코드 설명 (부가)

JSON

-> 데이터 쉽게 통신하기 위한 규약

```
import org.json.simple.JSONObject;
```

-> Array가 불편해 bank, fed 등에서 사용

Ex)
{"Message": "Money repaid successfully."
"Success": true}

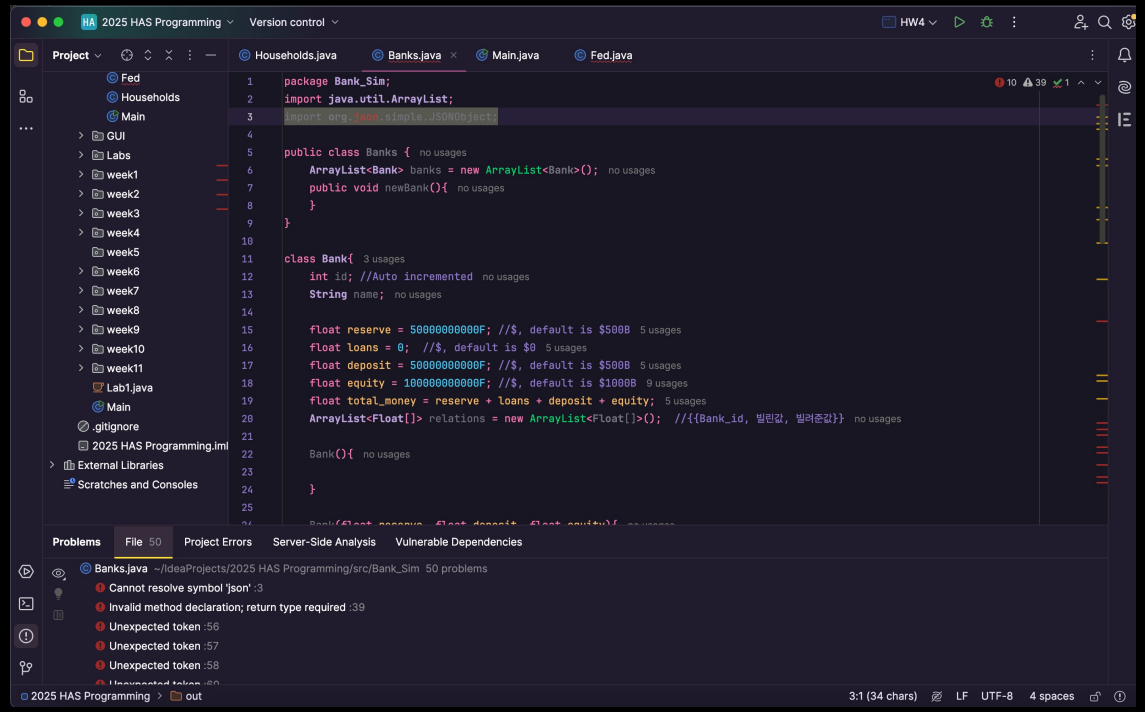
III. 코드시연

IV. 여러 시행착오

- Duplicate class: 'Main' :14
- 'Main.this' cannot be referenced from a static context :386
- Implicitly declared classes are not supported at language level '23' :394
- Cannot resolve symbol 'stateArea' :395
- Cannot resolve symbol 'banks' :399
- Cannot resolve symbol 'households' :410

너무 커진 코드로 인해 버그잡기 어려움

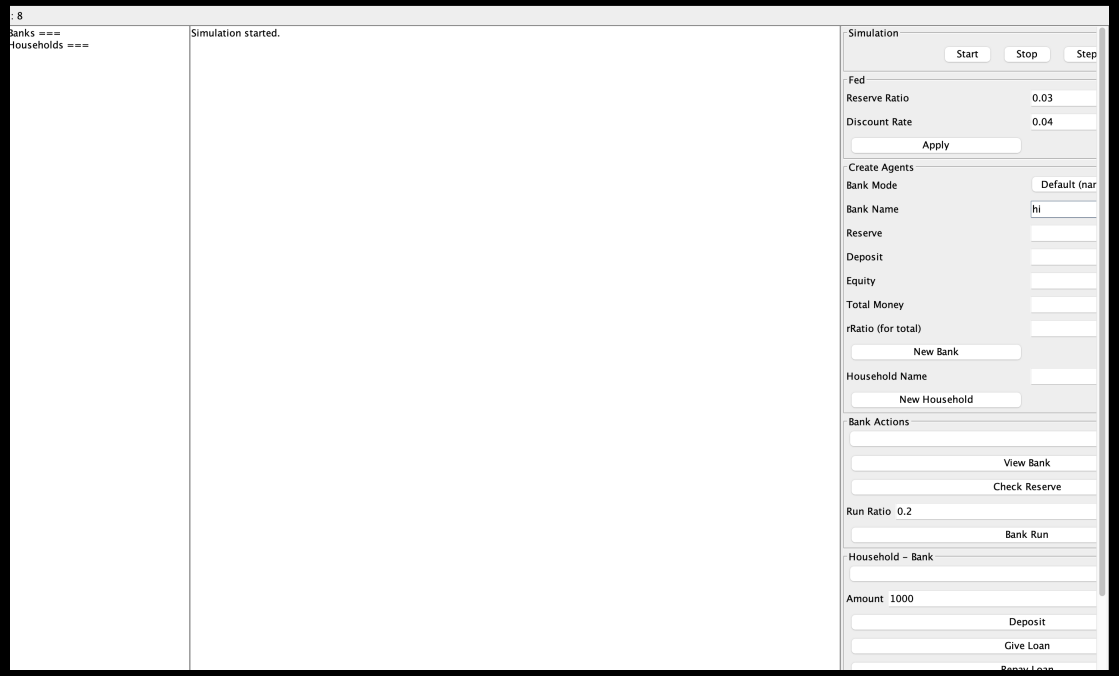
-> 그나마 extension 깔아서
디버깅은 extension으로 돌리면서 찾았다.



Java는 pip 같은 것이 없다
(특히 Maven, Gradle도 안 쓴 raw Java..)

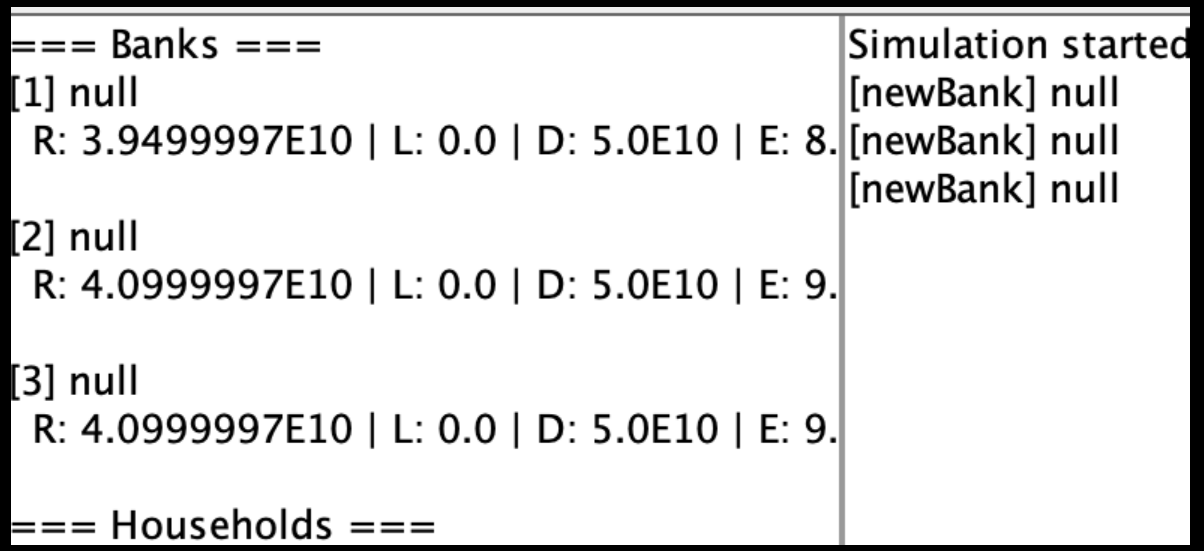
➔ 프로젝트 안 바꾸고 .jar 파일 받아서 lib에 넣었다
(그냥 모듈 거의 안 썼다)

IV. 여러 시행착오



GUI 상에서 패널, 버튼 크기 맞추기
매우 어렵다

➔ 계속 실행하면서 최적의 결과 얻었다.



곧잘 만들었더니 모든 Agent가 Null이랜다…
(모델과 GUI 간 Mismatch)

➔ GUI보단 다른 클래스에서 넘어오는 과정이
문제였어서 클래스 구현 방식을 좀 바꿨다

IV. 느낀점

I. 디버깅 하는 건 어렵지만 쾌감은 엄청나다

II. 코드가 길어지면 매우 매우 불안하다.
물론 끝내면 매우 뿌듯하다.