REPORT

전자공학도의 윤리 강령 (IEEE Code of Ethics)

(출처: http://www.ieee.org)

나는 전자공학도로서, 전자공학이 전 세계 인류의 삶에 끼치는 심대한 영향을 인식하여 우리의 직업, 동료와 사회에 대한 나의 의무를 짐에 있어 최고의 윤리적, 전문적 행위를 수행할 것을 다짐하면서, 다음에 동의한다.

1. **공중의 안전, 건강 복리에 대한 책임**: 공중의 안전, 건강, 복리에 부합하는 결정을 할 책임을 질 것이며, 공중 또는 환경을 위협할 수 있는 요인을 신속히 공개한다.

2. **지위 남용 배제**: 실존하거나 예기되는 이해 상충을 가능한 한 피하며, 실제로 이해가 상충할 때에는 이를 이해 관련 당사자에게 알린다. (이해 상충: conflicts of interest, 공적인 지위를 사적 이익에 남용할 가능성)

3. **정직성**: 청구 또는 견적을 함에 있어 입수 가능한 자료에 근거하여 정직하고 현실적으로 한다.

4. **뇌물 수수 금지**: 어떠한 형태의 뇌물도 거절한다.

5. **기술의 영향력 이해**: 기술과 기술의 적절한 응용 및 잠재적 영향에 대한 이해를 높인다.

6. **자기계발 및 책무성**: 기술적 능력을 유지, 증진하며, 훈련 또는 경험을 통하여 자격이 있는 경우이거나 관련 한계를 전부 밝힌 뒤에만 타인을 위한 기술 업무를 수행한다.

7. **엔지니어로서의 자세**: 기술상의 업무에 대한 솔직한 비평을 구하고, 수용하고, 제공하며, 오류를 인정하고 수정하며, 타인의 기여를 적절히 인정한다.

8. **차별 안하기**: 인종, 종교, 성별, 장애, 연령, 출신국 등의 요인에 관계없이 모든 사람을 공평하게 대한다.

9. **도덕성**: 허위 또는 악의적인 행위로 타인, 타인의 재산, 명예, 또는 취업에 해를 끼치지 않는다.

10. **동료애**: 동료와 협력자가 전문분야에서 발전하도록 도우며, 이 윤리 헌장을 준수하도록 지원한다.

위 IEEE 윤리헌장 정신에 입각하여 report를 작성하였음을 서약합니다.

학 부: 전자공학부
마감일: 2021. 06. 03.
과목명: 전자공학프로그래밍
교수명: 이정원 교수님
학 번: 201620837
성 명: 안민규

1. 함수 기능 및 코드 분석

1) Visitor

```
package assignment4;

public class Visitor {
	private String name; //VIP, Common 나누기 위한 name 변수
	private int money;    //고객이 원하는 혹은 입금하려는 금액 변수
	private boolean type; // true 면 입금, false면 대출

	public Visitor(String name, int money, boolean type)
	{
		this.name = name;
		this.money = money;
		this.type = type;
	}

	public Visitor(int money, boolean type)
	{
		this.name = "익명";
		this.money = money;
		this.type = type;
	}

	public int GetMoney()
	{
		return money;
	} //Visitor의 money 받아옴

	public boolean CheckType()
	{
		return this.type;
	} //Visitor의 type 받아옴 -> 대출인지 입금인지

	public String GetName()
	{
		return this.name;
	} //Visitor의 이름 받아옴 -> VIP인지 Common인지

}
```

Visitor 클래스로 고객의 정보를 담고 있다. 어떤 고객인지에 따라 담당되는 순서가 다르기 때문에 이를 저장하기 위해 name 변수를 선언하고, 고객이 원하는 업무가 입금인지 대출인지에 따라 결정되는 type을 선언하고, 또한 고객이 입금하려는 금액 혹은 원하는 대출 금액을 저장하기 위한 money 변수를 선언한다. 다른 클래스에서 Visitor 객체를 생성할 때, name, money, type을 입력하여 가장 위의 메소드를 이용하여 생성한다.

2) Waiting

```
package assignment4;

public class Waiting implements Runnable{

	private Visitor[] visitors;
	private int count; // 총 인원
	public int remainingVisitors = 0; // 남아있는 인원
	private int time = 0;
	private String name;
	int i=0;
	int j=0;

	@Override
	public void run() {
		// TODO Auto-generated method stub
		// common과 vip일 경우 다르게 진행
		if(this.name=="VIP") { //들어온 인원의 이름이 VIP로 설정되어있다면
			AddVisitor(Managing.vipVisitors[time]); //vipVisitors[]에 저장된 값을
```

```
visitors[]에 저장
                }
                else if(this.name=="Common") { //들어온 사람의 이름이 Common으로
설정되어있다면
                        AddVisitor(Managing.commonVisitors[time]); //commonVisitors[]에
저장된 값을 visitors[]에 저장
                }
                time+=1; //Waiting의 시간 값 1 증가
        }

        public void AddVisitor(Visitor visitor) {    //대기열에 매개변수로 들어온 visitor를
저장하는 메소드
                this.visitors[this.count]=visitor;      //매개변수 visitor를 visitors[count]에 저장.
count는 0부터 1씩 증가
                if(visitor!=null) {  //visitors에 저장된 값이 null이 아닐때까지 -> 즉, null을
만나면 그 순간부터 비어있는 것이므로
            this.count+=1;                                  //count 1씩 증가
                        this.remainingVisitors+=1;            //remainingVisitors 1씩 증가 ->
대기열에 있는 사람
                }
                else {                                          //visitor가 null을 만났을 때부터
                        System.out.println("No new "+name+" came in."); //들어온 고객이
없으므로 문구 출력
                }
        }

        public Visitor DealVisitor() {                //손님을 점원에게 배정하기 위한 메소드
                if(CheckWaiting()==true) {                //대기열에 사람이 있으면 true를
반환하므로 true라면
                        if(this.name=="VIP") {                //만약 이름이 VIP라면
                                Managing.vip.visitors=this.visitors;    //Addvisitor에서
매개변수를 저장해놓은 visitors를 managing.vip.visitors에 저장
                                j+=1;   //리턴을 내기위한 새로 선언한 integer 변수
                                this.remainingVisitors-=1;      //대기열의 사람을 한 명 점원에게
배정하였으므로 남은 인원 수 1 감소
                                return Managing.vip.visitors[j-1];  //DealVisitor 메소드의
리턴값으로 Managing.vip.visitors[j-1]을 리턴
                        }

                        else if(this.name=="Common") {      //이름이 common이라면
                                Managing.common.visitors=this.visitors; //Addvisitor에서
매개변수를 저장해놓은 visitors를 managing.common.visitors에 저장
                                i+=1;                                   //리턴을 내기위한 새로 선언한
integer 변수
                                this.remainingVisitors-=1;      //대기열의 사람을 한 명 점원에게
배정하였으므로 남은 인원 수 1 감소
                                return Managing.common.visitors[i-1];  //DealVisitor 메소드의
리턴값으로 Managing.common.visitors[i-1]을 리턴
                        }

                }
                return null; //대기중인 손님이 없는 경우
        }

        public boolean CheckWaiting() { //기다리고 있는 사람이 있는지 체크하는 메소드
                if(remainingVisitors == 0)  //남아있는 Visitors가 없다면
                        return false;              //기다리지 않으므로 false
                else
                        return true;               //기다리고 있으므로 true
        }

        Waiting(String name){
                this.visitors = new Visitor[10];
                this.remainingVisitors = 0;
                this.count = 0;
                this.name = name;
        }                                  //Waiting 객체 생성 위한 메소드

        public void PrintInfo()         //현재 남아있는 인원이 몇인지를 나타내는 메소드
        {
```

```
                System.out.println(name + " Waiting에 현재 남아있는 인원은 " +
remainingVisitors + "명 입니다.");
        }

}
```

run()은 쓰레드와 함께 실행되고 run 속에서는 이름에 따라 배열에 저장하는 작업이 이루어진다. 또한 배열에 저장할 때 들어가는 매개변수가 Waiting에서 미리 선언된 배열에 저장된 값이므로 이를 가져오기 위해 time을 이용해 시간에 맞추어 하나씩 저장한다.

visitor를 매개변수로 받아서 Addvisitor 메소드는 작동한다. Waiting으로 객체를 생성할 때 생성하는 배열의 첫번째부터 매개변수를 차례차례 저장한다. 이후 배열이 null이 만나지 않을 때까지 count와 remainingVisitor를 1씩 증가시킨다. 이후 null을 만나면 더 이상 인원이 들어오지 않는 것이므로 들어온 사람이 없다는 문구를 출력한다.

DealVisitor 메소드는 visitor를 반환하는 메소드이다. 이 반환된 visitor를 Managing 클래스의 common, vip 객체에 넣는다. 이를 Teller에서 담당하는 visitor에 저장하면 손님이 배정되는 알고리즘을 짰다. 앞서 addvisitor를 통해 매개변수가 저장된 배열을 Managing 클래스에서 생성된 객체에 저장한다. 이후 새로 선언한 integer 변수를 1씩 증가시키고 점원에게 배정했으므로 remainingVisitors를 1씩 감소시킨다. 이후 Managing 객체에 저장된 Visitors를 반환한다.

CheckWaiting() 메소드를 통해 현재 업무를 기다리는 사람이 있는지를 체크한다. remainingVisitors 변수를 가지고 구분한다. 0이면 기다리는 사람이 아무도 없는 것이므로 false, 0이 아니라면 기다리는 사람이 남아있으므로 true로 바꾼다.

PritnInfo() 메소드는 Managing 메소드에서 호출하여 출력한다. 이 메소드를 통해 현재 남아있는 인원을 보여준다.

3) Teller

```
package assignment4;

public class Teller implements Runnable{
        private String name;        // 이름
        private boolean working;   // 현재 상태
        private boolean workType; // true 면 입금, false면 대출
        private int progressTime; // 담당업무에 걸리는 시간
        private Visitor visitor;   // 담당중인 손님
        protected boolean Case;    // 대출 성공 실패를 나타내기 위해 선언한 Case

        public Teller(String name)
        {
                this.name = name;          //Teller의 이름
                this.working = false;     //Teller의 현재 상태
                this.workType = false;    //Teller의 업무
                this.progressTime = 0;    //Teller가 맡은 담당업무에 걸리는 시간

        }

        public void run() {
                // TODO Auto-generated method stub
                if (this.working == false) //일하는 상태가 아니라면
                {
                        DealCustomer();          //손님 배정
                }

                else                             //현재 일하는 상태라면
                {
                        this.workType=this.visitor.CheckType(); //방문자의 업무를 점원의 업무에
저장
                        if(this.progressTime>0) {
                                try{
                                        Thread.sleep(100);
                                } catch(InterruptedException e) {
                                        e.printStackTrace();
```

```
                                }                                       //try-catch 구문을 사용해
Thread.sleep()을 줌. 1000ms 동안 멈춰있음.
                                finally {
                                        this.progressTime-=1;           //try-catch 구문이
진행되면 업무시간을 1감소. 즉, 남은 업무 시간을 감소시키는 작업
                                }
                        }

                        if(this.progressTime==0) {      //주어진 업무시간이 끝나면
                                if(this.workType == false) {    //workType이 false라면 대출
업무 담당
                                        if(Case) {                      //Case에는 대출 가능
여부가 담겨있음
                                                Loan(this.visitor.GetMoney()); //Loan 메소드
통해 문구 출력
                                                this.working=false;     //작업이 끝났으므로
false로 변경
                                        }
                                }

                                else {
                                        Deposit(this.visitor.GetMoney()); //Deposit 메소드 동작
                                        this.working=false;             //작업이 끝났으므로
false
                                        this.workType=false;            //false가
기본값이므로 false로 변경
                                }
                        }
                }
        }

        public synchronized void DealCustomer()
        {
                // 해당 함수 수정 가능
                // 대출금 확인의 경우에는 대출을 시작할 때 금액 확인
                if (Managing.vip.CheckWaiting() == true){ //vip가 기다리고 있을 때
                        this.visitor=Managing.vip.DealVisitor(); //DealVisitor 메소드의 반환값을
visitor(Teller가 담당하는 고객)에 저장
                        this.working=true; //일하는 중으로 변경
                }

                else if((Managing.common.CheckWaiting() ==
true)&&(Managing.vip.CheckWaiting() == false)){ //vip가 안기다리면서 common이 기다리고 있을
때
                        this.visitor=Managing.common.DealVisitor(); //담당 중인 손님에
Managing.common.DealVisitor() 대입
                        this.working=true; //일하는 중으로 변경
                }

                else //아무도 기다리고 있지 않을 때
                {
                        this.working = false; //일하지 않으므로 false
                        return;
                }

                if(this.visitor.CheckType() == false) { //대출을 원할 때
                        if(Managing.CheckLoan(this.visitor.GetMoney()) == true) { //대출을
성공했을 경우
                                Managing.money-=this.visitor.GetMoney(); //대출의 경우 요청된
순간에 돈을 빼므로 Managing.money에서 담당받는 손님의 money를 뺌
                                this.progressTime=5;                    //5초가 걸려야하므로
업무에 걸리는 시간을 5로 지정
                                Case=true; //새로 선언한 Case 변수에 true를 저장
                        }
                        else { //대출을 원했으나 대출이 실패한 경우
                                System.out.println(this.name+" 점원은 돈이 부족해
"+this.visitor.GetName()+"의 " + this.visitor.GetMoney() + "원을 대출 해주지 못했습니다."); //대출
실패시 대출 불가 문구 출력
                                this.working=false;             //작업이 끝났으므로 false로 변경
                                Case=false; //새로 선언한 Case 변수에 false를 저장
```

```
                        }
                }
                else { //입금을 원하는 경우
                        this.progressTime=3; //입금 업무에 걸리는 시간은 3초이므로 3을 저장
                }
        }

        private synchronized void Deposit(int money) { // 입금의 경우에는 입금이 처리될 경우
진행
                System.out.println(name + " 점원은 " + visitor.GetName() +"고객님의 " + money
+ "원을 입금 처리 했습니다."); //입금 처리 했다는 문구 출력
                Managing.money += money; //입금되었으므로 은행의 머니에 입금된 만큼 추가
        }

        private synchronized void Loan(int money){ //대출이 요청될 경우 진행
                System.out.println(name + " 점원은 " + visitor.GetName() +"고객님의 " + money
+ "원을 대출 처리 했습니다."); //대출 처리 했다는 문구 출력

        }

}
```

Teller 메소드이다. Waiting 클래스를 통해 저장되어있는 손님의 정보를 바탕으로 은행 업무를 진행한다. 크게 대출과 입금으로 나뉘며 대출은 성공과 실패로 나눈다.
쓰레드가 실행될 때 run() 메소드가 함께 실행된다. 현재 은행원이 일하는 상태가 아니라면 DealCustomer 메소드를 호출해 손님을 배정한다.

DealCustomer 메소드의 경우 대기열의 정보를 가지고 손님을 배정해야 한다. 따라서 현재 대기열에 사람이 존재하는지를 확인하는 CheckWaiting() 메소드의 결과값을 조건으로 설정하였다. 만약 VIP 대기열에 사람이 존재한다면 Managing.vip.DealVisitor()의 반환값을 점원의 담당 손님에 저장한다. 만약 VIP 대기열에 사람이 없고 common 대기열에 사람이 있을 때 Managing.common.DealVisitor()의 반환값을 담당 손님에 저장한다. 이 때 각각 손님이 배정되면 점원은 일하는 상태이므로 working을 true로 바꾼다. 또한 아무도 기다리고 있지 않다면 일하지 않으므로 working을 그대로 false값을 준다. 여기까지가 손님을 배정하는 일이다. 이제 손님의 업무에 대한 정보를 미리 점원이 알 수 있게끔 구현하였다. 만약 손님이 대출을 원할 때는 먼저 대출이 가능한지에 따라 나누었다. 대출이 가능하다면 대출은 요청 시에 돈이 줄어들어야 하므로 바로 Managing.money에서 손님의 요청 금액만큼 –를 해주고, 대출업무에는 5초가 소요되므로 processTime을 5로 설정하였다. 이후 따로 선언한 변수인 Case에 true를 저장한다. 만약 대출이 불가능하다면 바로 대출 불가 문구를 1초 뒤에 출력되도록하고 Case에 false를 저장한다. Case를 사용하는 이유는 앞서 5초로 설정되었으나 기다리는 동안 금액이 모자라지게 되면 run() 속에서 대출 실패로 들어가게되어서 상태값을 저장하기 위해서 사용하였다. 입금의 경우 3초가 소요되므로 processTime을 3으로 설정하였다.

이미 일하는 상태라면 방문자의 업무를 점원의 업무에 저장한 후 Dealcustomer 메소드에서 지정한 progressTime(업무에 걸리는 시간)만큼 try-catch 문을 진행한다. try 안에서는 1초 동안 쓰레드가 멈추게 설정하고 try-catch가 끝날 때마다 업무시간을 1씩 감소하게 설정하였다. 이후 업무시간이 0초가 되면 업무에 필요한 시간을 모두 마쳤다고 생각하고 업무와 미리 설정한 Case에 따라 나누어 메소드를 호출한다. 대출이 성공할 때는 Loan 메소드를 호출한다. 그리고 입금일 때는 Deposit 메소드를 호출한다. 이 때 각각 모든 작업이 끝나면 직원이 일을 끝마친 것이므로 working을 다시 기본값인 false로 바꿔준다. 입금의 경우에는 workType이 false가 기본이므로 false로 초기화시킨다.

4) Managing

```
package assignment4;

public class Managing implements Runnable{
        public static int totalTime = 15; //15초 동안 진행 위해 15 선언
        public static int money = 15000;  //시작 잔액 15000원으로 선언
        public static int time = 0;       //시작 초 0초
        public static Waiting common;
```

```
        public static Waiting vip;          //Waiting 객체 2개 선언

        public static Visitor commonVisitors[];
        public static Visitor vipVisitors[];    //Vistitor 객체를 가지는 배열 선언


        @Override
        public void run() {
                common.PrintInfo();
                vip.PrintInfo();
                System.out.println("현재 은행 재고는 "+money+"입니다."); //현재 잔액 출력
                System.out.println("-----------시간 : "+time+"초-----------"); //현재 시간 출력
                time+=1; //시간 1씩 증가
        }

        public static boolean CheckLoan(int money) //대출 요청 시 잔액 확인
        {
                if(Managing.money - money < 0) { //현재 money에서 입력된 money를 뺐을 때
0보다 작으면
                        return false; //잔액 부족으로 대출 불가
                }
                else { //0보다 크므로
                        return true; //대출 가능
                }
        }

        public void init(Waiting common, Waiting vip) //Waiting으로 선언된 객체를 매개변수로
받음
        {
                Managing.common = common; //매개변수로 들어온 common을
Managing.common에 저장
                Managing.vip = vip;         //매개변수로 들어온 vip을 Managing.vip에 저장

                commonVisitors = new Visitor[totalTime]; //Visitor[15]로 commonVisitor 생성
                vipVisitors = new Visitor[totalTime];      //Visitor[15]로 vipVisitor 생성

                commonVisitors[0] = new Visitor("A", 3000, false);
                commonVisitors[1] = new Visitor("B", 3000, false);
                commonVisitors[2] = new Visitor("C", 3000, false);
                commonVisitors[3] = new Visitor("D", 3000, false); //객체 4개 생성

                vipVisitors[0] = new Visitor("VA", 3000, false);
                vipVisitors[1] = new Visitor("VB", 6000, false);
                vipVisitors[2] = new Visitor("VC", 6000, false);
                vipVisitors[3] = new Visitor("VD", 3000, true); //객체 4개 생성
        }


}
```

Manaing 클래스로 은행 업무에서 마지막으로 돌아가는 클래스이다. Waiting->Teller->Managing 클래스로 들어오고 Managing 클래스는 전체적인 은행 시스템과 변수들을 관리한다. Main에서 init 메소드를 통해 Visitor 객체들, 즉, 손님 객체들을 선언하고 이들이 Waiting 클래스로 인해 작동한다.

손님이 대출을 원할 때 대출이 가능한지를 확인하는 CheckLoan 메소드를 통해 Teller 클래스에서 CheckLoan 메소드를 실행하여 대출 가능 여부를 판단한다.

run() 메소드는 쓰레드로 쓰레드가 생성이 되고 난 이후 정해진 주기에 따라 작동한다. 현재 은행의 잔액과 현재 시간을 출력해주고, 시간을 1씩 증가시킨다. 주어진 목표에 맞춰 동작하게 구현하였다.

5) Main

```
package assignment4;


import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;
```

```
public class Main implements Runnable{
    public static void main(String[] args) throws InterruptedException {

        Teller tellerA = new Teller("유동연");
        Teller tellerB = new Teller("장현웅");
        Teller tellerC = new Teller("박예슬");

        Waiting common = new Waiting("Common");
        Waiting vip    = new Waiting("VIP");

        Managing managing = new Managing();

        managing.init(common, vip);

        ScheduledExecutorService service = Executors.newSingleThreadScheduledExecutor();

        service.scheduleAtFixedRate(tellerA, 500, 1000, TimeUnit.MILLISECONDS);
        service.scheduleAtFixedRate(tellerB, 500, 1000, TimeUnit.MILLISECONDS);
        service.scheduleAtFixedRate(tellerC, 500, 1000, TimeUnit.MILLISECONDS);

        service.scheduleAtFixedRate(common, 0, 1000, TimeUnit.MILLISECONDS);
        service.scheduleAtFixedRate(vip, 0, 1000, TimeUnit.MILLISECONDS);

        service.scheduleAtFixedRate(managing, 900, 1000, TimeUnit.MILLISECONDS);

        Thread.sleep(15000);
        service.shutdown();

    }

        @Override
        public void run() {
                // TODO Auto-generated method stub

        }

}
```

2. 결과화면 분석

<table>
<tr>
<td>

```
Common Waiting에 현재 남아있는 인원은 0명 입니다.
VIP Waiting에 현재 남아있는 인원은 0명 입니다.
현재 은행 재고는 9000입니다.
-----------시간 : 0초-----------
Common Waiting에 현재 남아있는 인원은 1명 입니다.
VIP Waiting에 현재 남아있는 인원은 0명 입니다.
현재 은행 재고는 3000입니다.
-----------시간 : 1초-----------
Common Waiting에 현재 남아있는 인원은 2명 입니다.
VIP Waiting에 현재 남아있는 인원은 1명 입니다.
현재 은행 재고는 3000입니다.
-----------시간 : 2초-----------
Common Waiting에 현재 남아있는 인원은 3명 입니다.
VIP Waiting에 현재 남아있는 인원은 2명 입니다.
현재 은행 재고는 3000입니다.
-----------시간 : 3초-----------
No new Common came in.
No new VIP came in.
Common Waiting에 현재 남아있는 인원은 3명 입니다.
VIP Waiting에 현재 남아있는 인원은 2명 입니다.
현재 은행 재고는 3000입니다.
-----------시간 : 4초-----------
No new Common came in.
No new VIP came in.
유동연 점원은 VA고객님의 3000원을 대출 처리 했습니다.
장현웅 점원은 A고객님의 3000원을 대출 처리 했습니다.
Common Waiting에 현재 남아있는 인원은 3명 입니다.
VIP Waiting에 현재 남아있는 인원은 2명 입니다.
현재 은행 재고는 3000입니다.
-----------시간 : 5초-----------
```

</td>
<td>

1. 15000원에서 시작했으나 0초에 VA와 A가 들어와 3천원씩 대출을 요청하였으므로 15000-6000=9000원이 출력된다.
2. 1초에 VB와 B가 들어왔으나 VB를 먼저 담당하므로 B는 대기열로 들어간다. 현재는 VA, A, VB의 업무를 담당하고 있다.
3. 2초에 VC와 C가 들어왔으나 점원 3명이 모두 일하는 중이므로 대기열에 들어간다.
4. 3초에는 VD와 D가 들어와 대기열에 들어간다.
5. 4초에는 새로 들어온 인원이 없으므로 새로 들어온 사람이 없다는 문구와 함께 그대로 유지
6. 5초에는 0초에 대출을 요청한 VA와 A는 5초가 지나 대출업무가 종료됨을 나타낸다. 이 자리에 VC와 VD가 들어간다. VC는 6천원 대출을 원하는데 남은 잔액이 3천원이므로 대출이 불가능하다. VD는 입금을 하려하므로 현재는 잔액에 변화가 없다.

</td>
</tr>
<tr>
<td>

```
No new Common came in.
No new VIP came in.
유동연 점원은 돈이 부족해 VC의 6000원을 대출 해주지 못했습니다.
박예슬 점원은 VB고객님의 6000원을 대출 처리 했습니다.
Common Waiting에 현재 남아있는 인원은 3명 입니다.
VIP Waiting에 현재 남아있는 인원은 0명 입니다.
현재 은행 재고는 3000입니다.
-----------시간 : 6초-----------
No new Common came in.
No new VIP came in.
박예슬 점원은 돈이 부족해 C의 3000원을 대출 해주지 못했습니다.
Common Waiting에 현재 남아있는 인원은 1명 입니다.
VIP Waiting에 현재 남아있는 인원은 0명 입니다.
현재 은행 재고는 0입니다.
-----------시간 : 7초-----------
No new Common came in.
No new VIP came in.
박예슬 점원은 돈이 부족해 D의 3000원을 대출 해주지 못했습니다.
Common Waiting에 현재 남아있는 인원은 0명 입니다.
VIP Waiting에 현재 남아있는 인원은 0명 입니다.
현재 은행 재고는 0입니다.
-----------시간 : 8초-----------
No new Common came in.
No new VIP came in.
장현웅 점원은 VD고객님의 3000원을 입금 처리 했습니다.
Common Waiting에 현재 남아있는 인원은 0명 입니다.
VIP Waiting에 현재 남아있는 인원은 0명 입니다.
현재 은행 재고는 3000입니다.
-----------시간 : 9초-----------

No new Common came in.
No new VIP came in.
Common Waiting에 현재 남아있는 인원은 0명 입니다.
VIP Waiting에 현재 남아있는 인원은 0명 입니다.
현재 은행 재고는 3000입니다.
-----------시간 : 10초-----------
No new Common came in.
No new VIP came in.
Common Waiting에 현재 남아있는 인원은 0명 입니다.
VIP Waiting에 현재 남아있는 인원은 0명 입니다.
현재 은행 재고는 3000입니다.
-----------시간 : 11초-----------
No new Common came in.
No new VIP came in.
유동연 점원은 B고객님의 3000원을 대출 처리 했습니다.
Common Waiting에 현재 남아있는 인원은 0명 입니다.
VIP Waiting에 현재 남아있는 인원은 0명 입니다.
현재 은행 재고는 3000입니다.
-----------시간 : 12초-----------
```

</td>
<td>

7. 6초에는 1초에 대출을 요청한 VB가 5초가 지나 대출업무가 종료된다. 5초에 들어온 VC의 요청이 거절됐으므로 대출 불가 문구가 출력이 된다. 또한 VB, VC 자리에 B, C가 들어간다. B는 3천원 대출을 원하므로 잔액에서 3천원을 차감한다. 먼저 온 B부터 대출이 되므로 C는 잔액이 없어 대출이 불가능하다.
8. 7초에는 C의 대출이 불가능하다는 문구가 출력된다. 이 후 남아있는 D가 점원에게 간다. 아직 입금과 대출이 끝나지 않았고, 잔액이 0이므로 D의 대출 요청 또한 거절한다.
9. 8초에는 D의 요청 거절 문구가 출력이 된다.
10. 5초에 입금을 요청한 VD의 입금이 끝나고 입금 완료 문구와 함께 잔액이 증가한다.
11. VA~VD, A~D 모든 사람이 점원과 일을 마치거나 일을 보는 중이므로 대기열은 둘 다 0으로 출력된다.
12. B의 대출이 이제야 완료가 되고 7초에서 5초가 지난 12초에 B에게 대출해줬다는 문구가 출력된다.

</td>
</tr>
</table>

3. 프로그램 개선 방법

가장 먼저 들었던 생각은 굳이 Waiting에서 배열을 만들어 대기열을 만들어야 하나라는 생각이 들었다. 실제로 Waiting 클래스 내의 Addvisitor나 DealVisitor 들을 사용하지 않아도 Teller에서 바로 배열에 대입하여 Managing과 연결해 볼 수 있었다. 또한 이번 과제처럼 은행 시스템 관련한 자바 코드가 많은데 그들은 좀 더 많고 자세한 업무를 다룰 수 있게끔 설정을 하였고, 직접 손님을 생성하지 않아도 Random을 통해 생성하고 이를 통해 점원에서 랜덤 배정을 통해 클래스끼리의 불필요한 연결을 완화했다는 생각이 들었다. 그만큼 이번 과제에서 주어진 방법은 불편하기도하고 불필요한 부분이 많다고 느꼈다.