2018 전공기초프로젝트2 (2040) 화금반

[2차 설계 문서]

: (6조) 음료 자판기 시뮬레이션 프로그램

조장 201511206 왕윤성

201511197 방승희

201611186 김나경

제출일 : 2018년 11월 27일

[목 차]

1. 개발 환경

1.1. 언어

C++ 사용 객체지향형

1.2. 개발 툴

Visual Studio 2017

1.3. UI 환경

CUI(Command Line User Interface)

2. 객체

2.1. 음료Drink

- 멤버 변수

  + bool _isRegist : 등록한 슬롯인지 판단

  + char _name[21] : 음료의 이름 저장(20자까지)

  + int _price : 음료의 가격 저장

  + int _stock : 음료의 재고 저장

- 멤버 함수

  + Drink() : 생성자. 멤버 변수들을 초기화 함.

  (_isRegist, _name, _price, _stock) =

  (false, null, 0, 0)

  + Drink(char* name, int price, int stock) : 생성자 오버로딩.

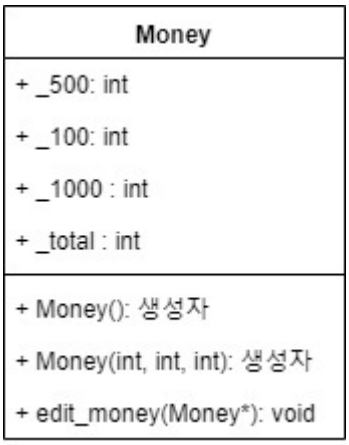
  (_isRegist, _name, _price, _stock) =

  (true, name, price, stock)

  + void set_stock(int stock) : 음료의 재고를 파라미터 stock으로 변경함.

  추가 혹은 제거할 양만큼 입력이 아니라 변경할 양을 입력 받는 것.

2.2. 돈Money



- 멤버변수

  + int _100 : 100원의 개수

  + int _500 : 500원의 개수

  + int _1000 : 1000원의 개수

  + int _total : 총액

- 멤버함수

  + Money() : 생성자. 멤버 변수들을 초기화 함.

  (_100, _500, _1000, _total) = (0, 0, 0, 0)

  + Money(int input_100, int input_500, int input_1000) :

  생성자 오버로딩.

  (_100, _500, _1000, _total) =

  (input_100, input_500, input_1000, total 자체 계산)

  + void edit_money(*Money income) : 금고의 잔고를 파라미터 income을 **더해** 갱신함.

2.3. 기계Machine

Machine
+ _drink : *Drink + _mmoney: *Money + available: bool + input: *Money + change: *Money + mode: int - instance : *Machine
- Machine() : 생성자 - Machine(&Machine) : 생성자 - ~Machine() : void + getInstance() : *Machine + check_blank_slot() : int + calculate(int) : int + optimal_change(int) : bool + show_money() : void + show_drink() : void + get_money(Money): void + return_change(): void

- 싱글톤 패턴
  - 한번 인스턴스를 생성하면 프로그램 전체에서 딱 한 번만 생성되고, 재생성되지 않아야 함.
  - 재 호출시 처음에 만들어 놓은 인스턴스를 참조 하게 해야 함.
  - 멤버 함수 static Machine* getInstance() 참고

```
static Machine* getInstance() {
        if (instance == NULL)
            instance = new Machine();
        return instance;
}
```

- 멤버변수

  - ***Machine instance** : 메인함수에서 처음으로 생성한 Machine을 가리키는 객체포인터.

  + *Drink _drink : 자판기의 음료 저장.

  + *Money _mmoney : 자판기가 보유한 돈 저장.

  + bool available : 재고 및 잔고 이용 가능 여부 저장.(true : 사용가능, false : 재고 없음 또는 잔돈 없음)

  + *Money input : 투입한 금액 저장

  + *Money change : 잔돈 저장. 최적 반환계산을 통해 나온 결과를 저장.

  + int mode : 관리자, 사용자 모드 구분 값 저장. (1 : 관리자, 2 : 사용자, 3 : 종료)

- 멤버함수

  - **Machine()** : 생성자. 멤버 변수들을 초기화함.

  - **Machine(&Machine)** : 생성자 오버로딩.

  - **~Machine()** : 소멸자. 데이터 해제.

  + *Machine getInstance() : 싱글톤패턴 구현. 멤버변수 instance가 이미 있다면 그 instance를 리턴하고, 없다면(처음 생성할 경우) 새로 생성해서 리턴함.

  + int check_blank_slot() : _drink에서 인덱스 순으로 했을 때 처음으로 찾은 빈 슬롯의 인덱스 번호 리턴. 모든 슬롯이 빈 슬롯이 아닌 경우 -1을 리턴한다.

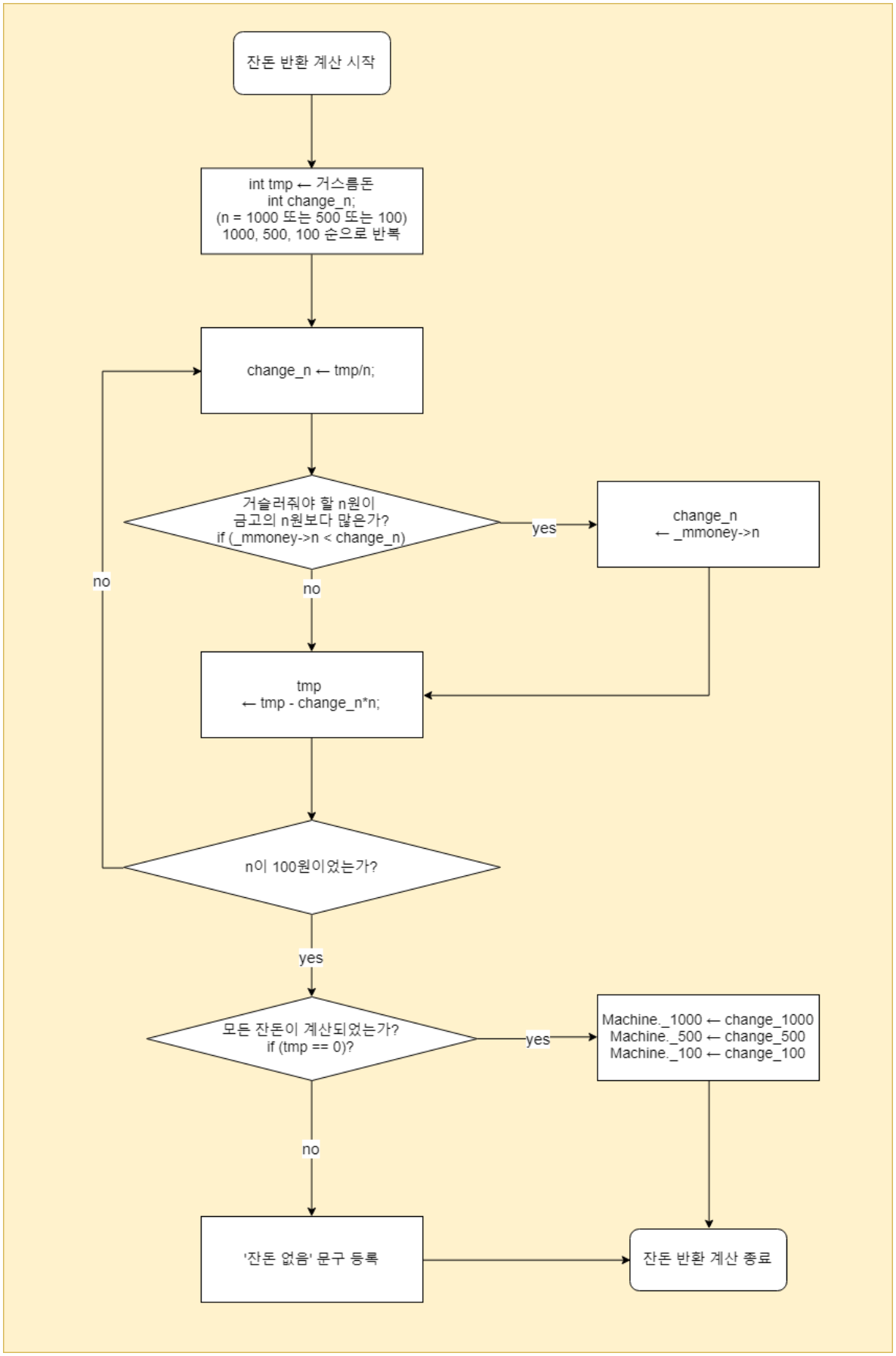
  + int calculate(int slot_num) : 슬롯 번호를 받아서 해당 슬롯의 음료를 투입된 금액으로부터 뺌(차액 계산). 돌려줄 금액의 토탈을 구해 리턴하는 함수.

  + bool optimal_change(int tmp) : 파라미터로 tmp(돌려줘야 하는 잔돈의 토탈 금액)를 받아 Machine의 금고(_mmoney)를 확인해 잔돈을 반환할 수 있는지 여부 확인. 동시에 최적으로 반환할 수 있는 방법 계산. (최적 반환이란 기계에 주어진 돈 내에서 큰 단위의 돈을 최대한 많이 쓰는 방법으로 반환하는 것.) 돌려줄 돈을 계산해 멤버 변수 change에 저장.

  1000원, 500원, 100원 순서로 계산. 100원인 경우 if문에서 return false;(false를 return했다는 뜻은 잔돈을 줄 수 없다는 뜻)

  ```
  //1000원으로 돌려줄 수 있는 최대 개수 계산
  ```

```
change_1000 = tmp / MONEY_1000;

 //금고 안의 1000원 개수와 비교해 금고 1000원이 적으면
if (_mmoney->_1000 < change_1000) {
        //금고의 1000원 개수 만큼만 돌려준다 (능력만큼)
        change_1000 = _mmoney->_1000;
}
//돌려줄 수 있는 금액을 tmp에서 뺀다
tmp -= change_1000 * MONEY_1000;
```

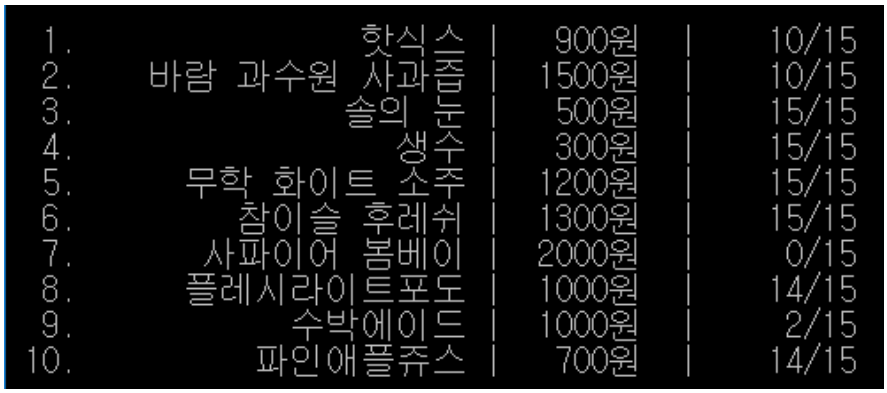

잔돈 반환 계산 시작
int tmp ← 거스름돈
int change_n;
(n = 1000 또는 500 또는 100)
1000, 500, 100 순으로 반복
change_n ← tmp/n;
거슬러줘야 할 n원이
금고의 n원보다 많은가?
if (_mmoney->n < change_n)
yes
change_n
← _mmoney->n
no
no
tmp
← tmp - change_n*n;
n이 100원이었는가?
yes
모든 잔돈이 계산되었는가?
if (tmp == 0)?
yes
Machine._1000 ← change_1000
Machine._500 ← change_500
Machine._100 ← change_100
no
'잔돈 없음' 문구 등록
잔돈 반환 계산 종료

:잔돈 반환 순서도

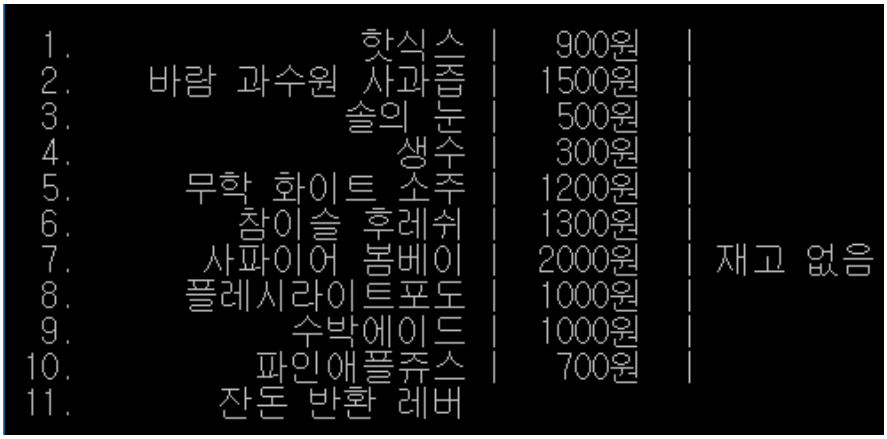
- 입력으로 잔돈의 총액을 tmp 변수에 받음
- Machine->change를 초기화함
- change_100, change_500, change_1000을 각각 선언
- 제일 큰 단위인 1000원으로 tmp를 나누어서 1000원으로 줄 수 있는 가장 큰 개수를 구하고 change_1000에 저장
- change_1000 > _mmoney->_1000인지 확인. 즉 금고가 지닌 1000원 개수가 거슬러줘야 하는 1000원보다 적은지 확인함.
- 만약 그렇다면 change_1000을 Machine->_1000으로 갱신. 금고가 줄 수 있는 최대한의 1000원 개수로 갱신하는 것.
- 거슬러줄 수 있는 1000원의 개수만큼의 금액을 tmp에서 차감.
- 이 과정을 다음단위 500, 100원으로 반복
- 마지막으로 tmp가 0이 되었는지 확인
- tmp == 0이라면 모든 거스름돈이 성공적으로 거슬러질 수 있다는 의미이므로, machine->change를 new Money(change_100, change_500, change_1000)로 갱신 후 return true
- tmp =/= 0이라면 거스름돈을 거슬러줄 수 없다는 의미이므로 return false

+ void show_money() : 금고에 있는 단위별 돈의 개수와 전체 금액을 보여줌.

+ void show_drink() : 슬롯에 저장된 음료를 보여줌. 멤버 변수 mode의 값에 따라 1(관리자)이면 재고까지 포함해서 보여줌. 2(사용자)이면 재고를 뺀 음료 슬롯번호, 음료이름, 가격 까지만 보여주고, 재고가 없을 경우만 없음을 표시함. 또한 11번 잔돈 반환레버까지 추가해서 보여줌.



관리자모드 show_drink() 예시



사용자모드 show_drink() 예시

+ void get_money(Money* m) : 투입된 돈(m)을 금고(_mmoney)에 저장.

Money 클래스 멤버함수 edit_money(Money*)를 이용해서 input(사용자가 투입한 금액) 갱신.

기계 금고에도 투입한 금액을 합산해 갱신 함.

```
//사용자로부터 투입 금액 갱신
	void get_money(Money* m) {
		//사용자로부터 투입
		input->edit_money(m);
		//기계 금고에 추가
		_mmoney->edit_money(input);
	}
```

+ void return_change() : 잔돈 반환. 사용자가 슬롯 번호 입력시 11을 입력해서 잔돈을 반환하거나 기계 내 금고 상황으로 잔돈을 돌려줄 수 없거나 슬롯 번호 입력시 아무것도 누르지 않고 30초 경과 시 지금까지 투입한 돈을 돌려줄 때 이 함수를 호출함.

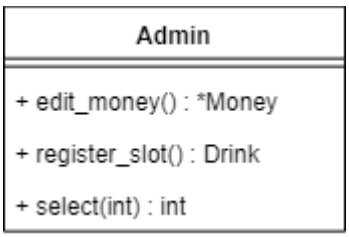
change의 100원 개수, 500원 개수, 1000원 개수를 각각 음수로 바꾼 Money객체(tmp) 생성. 금고(_mmoney)에 대해 edit_money(tmp). (금고(_mmoney)로부터 돌려줄 돈(change)을 뺀 것)

```
//임시로 뺄 돈의 정보를 (-)로 저장한 Money 객체.
		Money* tmp = new Money(-change->_100, -change->_500, -change->_1000);
```

```
//금고의 정보를 편집함. tmp만큼 변경.
_mmoney->edit_money(tmp);
```

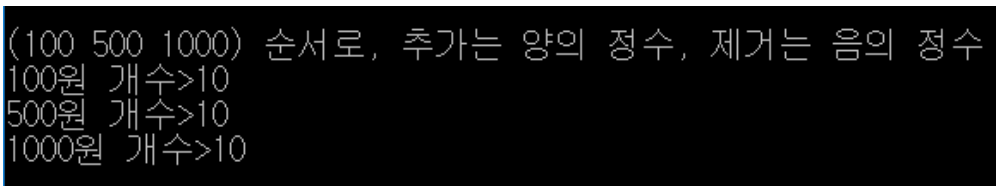
돌려줄 잔액의 상황을 출력함. 투입 금액(input)과 돌려줄 금액(change)은 다시 0으로 초기화.

2.4. 관리자Admin



- 멤버 함수

+ *Money edit_money() : 관리자가 기계의 금고를 변경하고자 할 때 변경하는 양을 전달하기 위해 사용하는 함수.
Money클래스의 **void edit_money(Money*)**와 이름이 같음.
100원, 500원, 1000원 순서로 입력을 받음.



추가하고 싶다면 양의 정수를, 제거하고 싶다면 음의 정수를 입력함.
추가/제거해서 저장할 숫자를 입력하는 것이 아니라 추가/제거하고 싶은 만큼을 입력하는 것.

```
while (input_test == false) {
        cout << "100원 개수>";
        cin >> tmp100;
        //exception_test함수는 첫번째 매개변수가 올바른 값이면 true, 아니면 false를 리턴해서 input_test에 넣는다.
        input_test = exception_test(tmp100, MONEY_100_MAX - (M->_mmoney->_100), -(M->_mmoney->_100));
}

//다음 금액권을 검사하는 반복문으로 들어가기 위해 다시 false로 바꿔준다.
input_test = false;
```

금액 별로 입력 검사를 진행함. bool input_test로 올바른 입력을 했는지 판단. 모듈화 해둔 exception_test(int,int,int)로 검사하며 최대 값은 현재

금고에서 수용 가능한 양, 최소 값은 금고가 가지고 있는 양의 음수 값으로 설정한다. 해당 예외처리 함수를 통과했을 때에만 다음 금액 권을 입력할 수 있다. 변경하고자 하는 Money* 인스턴스를 리턴함.

+ Drink register_slot() : 관리자가 기계의 음료 슬롯을 등록한다.
등록하려는 슬롯번호는 이 멤버함수로 받지 않고 main 함수에서 처리한다.
이름과 가격, 재고를 입력함. 입력한 결과를 출력하고 해당 정보는
*Drink로 리턴됨.

이름 입력 - 20자까지만 받고 나머지는 무시한다. 이때, 20Byte에서 걸리는 한글은 잘려서 출력될 수 있다.

```
cout << "name> ";

//getline함수는 두번째 매개변수로 받는 사이즈만큼의 입력만 첫번째 매개변수의 주소로 받고, 나머지는 버린다.
cin.getline(name, DRINK_NAME_MAX_LENGTH + 1);
```

가격 입력 - exception_test로 최대 2000원 최소 100원 설정, 100원 단위로만 받을 수 있도록 함.

```
while (input_test == false) {
	cout << "price> ";
	cin >> price;

	//exception_test함수는 첫번째 매개변수가 올바른 값이면 true, 아니면 false를 리턴해서 input_test에 넣는다.
	input_test = exception_test(price, 2000, 100);

	//입력한 가격이 100원단위가 아니면
	if (price % 100 != 0) {
		cout << "가격은 100원 단위로 입력해주세요." << endl;
		input_test = false;
	}
}
```

재고 입력 - exception_test로 최대는 INT_MAX, 최소는 0개로 설정해 받을 수 있도록 함. 재고가 15개보다 많이 입력된 경우 15로 자동 설정.

```
while (input_test == false) {
	cout << "stock> ";
	cin >> stock;
	//exception_test함수는 첫번째 매개변수가 올바른 값이면
true, 아니면 false를 리턴해서 input_test에 넣는다.
	input_test = exception_test(stock, INT_MAX, 0);
}
//만약 관리자가 재고를 15보다 많게 입력했을 경우
if (stock > DRINK_MAX_STOCK) {
	cout << "재고의 최대값(15개)보다 많이 입력하여 재고는 15로
편집됩니다." << endl;
	stock = 15;
}
```

+ int select(int choice) : 관리자로부터 변경할 음료 슬롯 번호나 재고 수를 입력받을 때 호출하는 함수. 선택한 숫자를 리턴함.

choice의 값이 0이면 슬롯 번호를 선택 받음 - 음료관리에서 슬롯등록, 재고편집, 슬롯삭제 시 사용, 1~11까지의 숫자 외는 다 다시 받도록 함.

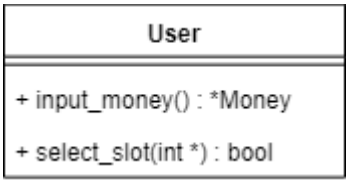
```
if (choice == 0) {
	cout << "******1~10까지의 정수, 되돌아 가려면 11을
입력하세요******" << endl;
	cout << "슬롯 번호>";
	cin >> select_num;
	//exception_test함수는 첫번째 매개변수가 올바른 값이면
true, 아니면 false를 리턴해서 input_test에 넣는다.
	//여기서 슬롯번호 선택은 -> 최대 값은 11, 최소 값은 1 (11은
돌아가기를 위한 번호)
	input_test = exception_test(select_num,
DRINK_MAX_SLOT+1, 1);
}
```

choice의 값이 1이면 재고 수를 선택 받음 - 음료관리에서 슬롯등록과 재고편집 시 재고 입력에서 사용, 0~15이상의 숫자 외는 다 다시 받도록 함. 15이상의 숫자는 15로 세팅된다.

```
else if (choice == 1) {
	cout << "변경할 재고 수>";
	cin >> select_num;
```

```
        //exception_test함수는 첫번째 매개변수가 올바른 값이면
true, 아니면 false를 리턴해서 input_test에 넣는다.
        //여기서 최대 값은 int형 변수가 담을 수 있는 최대 값 - 아무리
큰 값을 넣어도 15로 셋팅됨. 음수는 넣을 수 없음.
        input_test = exception_test(select_num, INT_MAX, 0);
        cout << endl;
}
```

2.5. 사용자User



- 멤버 함수

  + *Money input_money() : 사용자가 돈을 투입할 때 호출되는 함수. 투입받은 돈은 Money 형식으로 리턴함. Money형식으로 리턴시 각 돈의 종류마다 몇개인지의 정보를 함께 넘길 수 있음.

  금액권 별 입력 예외처리 - 만약 금고에 들어갈 수 있는 개수를 초과했다면 투입받은 개수를 **음수화**한 다음 저장 (나중에 값을 확인 했을 때 해당 값이 음수이면 금고 양을 초과했다는 것을 표시하기 위함.) 초과됐다면 바로 그 분기에서 return해버림. 초과하지 않았다면 다음 금액권을 검사하는 반복문을 똑같이 수행함.

```
//예외처리용 반복문
while (input_test == false) {
        cout << "100원 개수>";
        cin >> tmp100;
        if ((tmp100 + M->_mmoney->_100) >
MONEY_100_MAX) {
                cout << endl << "!기계에 들어갈 수 있는 100원 동전
양을 초과하였습니다!" << endl << endl;
                //tmp100을 음수로 설정함 : 사용자의 투입이 음수인
경우는 금고가 넘쳐서 돌려줘야함을 뜻한다.
                tmp100 = -tmp100;
                //더이상 500원이나 1000원 입력을 받지 않고 바로
return
                return tmp = new Money(tmp100, 0, 0);
```

```
        }
        //exception_test함수는 첫번째 매개변수가 올바른 값이면
true, 아니면 false를 리턴해서 input_test에 넣는다.
        input_test = exception_test(tmp100, MONEY_100_MAX -
(M->_mmoney->_100), 0);
}

//다음 반복문으로 들어가기 위해 다시 false로 바꿔준다.
input_test = false;
```

+ bool select_slot(int *slot) : 사용자로부터 슬롯을 입력 받음.(주문 받기)
- 슬롯 번호가 올바르게 입력됐는지 검사하는 함수(01,1 두가지 경우를 둘다 1로 인식하는 코드도 필요), 30초가 초과됐는지 검사하는 코드가 동시에 동작해야함.

슬롯 번호가 올바르게 입력됐는지 검사하는 함수 - 파란색으로 표시
시간이 초과하는지 검사하는 코드 - 빨간색으로 표시
while문이 도는 시간이 1초 이하라 가정하고 while문을 반복할 때마다 시간 차가 30초가 넘었는지 체크함.

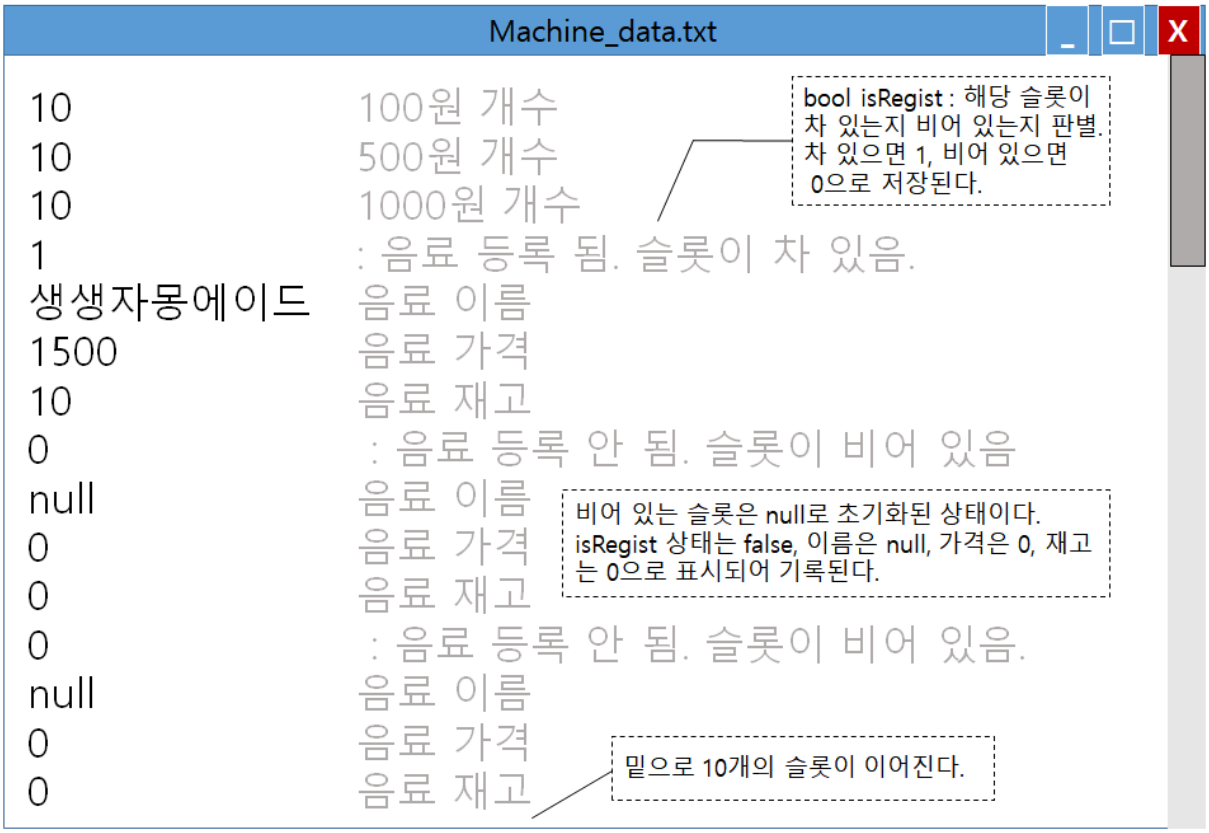
```
cout << "슬롯 번호>";
while (1) {
        end_time = (unsigned)time(0);          //이 때 엔드 타임
시작!
        //키보드 입력이 들어오면 아래의 if문 안으로 들어온다.
        if (_kbhit()) {
                cin.getline(tmp_slot, 3);

                if (cin.fail()) {
                        cin.clear();
                        //사용자가 INT_MAX(2147483647)개 이상의
알파벳을 치지 않는 이상 버퍼 문제를 해결해준다.
                        //cin.ignore함수의 첫번째 매개변수는 버릴
문자의 수, 두번째 매개변수는
                        cin.ignore(INT_MAX, '₩n');
                        //스타트 타임을 다시 지정해준다.
                        start_time = (unsigned)time(0);
                        cout << "다시 입력해주세요" << endl << endl;
                        cout << "슬롯 번호>";
```

```
                        //반복문의 남은 코드를 무시하고 반복문 가장
위로 점프한다.
                        continue;
                }
                //입력받은 값의 첫번째 문자가 0일때
                if (tmp_slot[0] == '0') {
                        //01~09입력 경우
                        if (tmp_slot[1] > '0' && tmp_slot[1] <= '9') {
                                *slot = (int)tmp_slot[1] - '0';
                                return true;
                        }
                }
                //입력받은 값의 첫번째 문자가 1~9일때
                else if (tmp_slot[0] > '0' && tmp_slot[0] <= '9' &&
tmp_slot[1] == NULL) {
                        *slot = (int)tmp_slot[0] - '0';
                        return true;
                }
                //10입력 한 경우
                if (tmp_slot[0] == '1'&&tmp_slot[1] == '0') {
                        *slot = 10;
                        return true;
                }
                //11입력 한 경우
                if (tmp_slot[0] == '1'&&tmp_slot[1] == '1') {
                        *slot = 11;
                        return true;
                }
                //스타트 타임을 다시 지정해준다.
                start_time = (unsigned)time(0);
                cout << "다시 입력해주세요" << endl;
                cout << "슬롯 번호>";
        }
        //지정해놓은 시간이 지나면 false를 리턴한다.
        if (end_time - start_time > TIME_LIMIT) {
                *slot = 0;
                //false를 리턴하면 사용자가 넣은 돈이 그대로 반환되게
한다.
                return false;
        }
}
```

3. 모듈화

3.1. 데이터 파일 읽고 쓰기

- machine_data.txt 에 저장



- **bool file_read_function()**

: 처음 프로그램 실행 시, 메인함수의 처음에서 데이터를 읽어와 프로그램 객체에 저장하는 함수. 정상적으로 읽어져 왔다면 return true; 아니라면 return fasle;

```
//처음으로 프로젝트를 돌려보면, 파일이 없으므로 바로 return true
if (!datafile) {
        cout << "데이터를 정상적으로 불러왔습니다." << endl << endl;
        return true;
}
//파일이 정상적으로 열리지 않았다면 return false
if (!datafile.is_open()) {
        cout << "데이터 불러오기에 실패했습니다.. 정상적인 작동을 위해 다시 실행해주세요." << endl;
        return false;
}
```

파일로부터 한줄씩 읽어올 데이터를 임시저장할 char 배열 buffer를 선언하고, 금고 정보(100원, 500원, 1000원의 수), 슬롯 별 음료 정보(슬롯이 등록되었는가의 여부, 음료의 이름, 가격, 재고) 순으로 Machine 객체의 데이터를 저장한다. 저장 후에는 마지막까지 읽은 게 맞는지 검사한다.

```
//파일로부터 한줄씩 읽어올 데이터를 임시저장할 char배열
//파일에서 제일 긴 항목이 최대 20자인 음료이름이므로 그를 기준으로 버퍼 크기 잡기.
//+1은 공백
char buffer[DRINK_NAME_MAX_LENGTH + 1];


//Machine 인스턴스 가져옴. main함수에서 맨 처음 생성한 그 인스턴스임.
Machine* M = Machine::getInstance();

//아래는 쭈욱 미리 설계해 둔 파일 형식에 맞게 읽어오는 코드
//한줄씩 읽어오는 코드의 반복
```

```
//금고 정보 저장
	datafile.getline(buffer, DRINK_NAME_MAX_LENGTH + 1);
	int _100 = atoi(buffer);		//금고 100원 갯수
	datafile.getline(buffer, DRINK_NAME_MAX_LENGTH + 1);
	int _500 = atoi(buffer);		//금고 500원 갯수
	datafile.getline(buffer, DRINK_NAME_MAX_LENGTH + 1);
	int _1000 = atoi(buffer);		//금고 1000원 갯수
	M->_mmoney = new Money(_100, _500, _1000);


//음료 정보 저장
	for (int i = 0; i < DRINK_MAX_SLOT; i++) {
		datafile.getline(buffer, DRINK_NAME_MAX_LENGTH + 1);
		M->_drink[i]._isRegist = atoi(buffer);	//등록된 음료인가
		datafile.getline(buffer, DRINK_NAME_MAX_LENGTH + 1);
		if (M->_drink[i]._isRegist)		//등록된 음료라면 이름
			strcpy_s(M->_drink[i]._name, buffer);
		datafile.getline(buffer, DRINK_NAME_MAX_LENGTH + 1);
		if (M->_drink[i]._isRegist)		//등록된 음료라면 가격
```

```
                M->_drink[i]._price = atoi(buffer);
            datafile.getline(buffer, DRINK_NAME_MAX_LENGTH + 1);
            if (M->_drink[i]._isRegist)          //등록된 음료라면 재고
                M->_drink[i]._stock = atoi(buffer);
        }
        datafile.getline(buffer, DRINK_NAME_MAX_LENGTH + 1);
        //마지막 개행 의미없이 읽음 -> 마지막까지 읽었는지 확인하기 위함
```

마지막까지 읽었으면 정상적으로 불러온 것이므로 안내 문구를 띄우고 true를 출력하고, 마지막까지 읽은 게 아니라면 정상적으로 저장된 게 아니므로, 안내 문구를 띄우고 false를 출력한다.
true값을 리턴받았을 경우에만 제대로 된 Vending_Machine 기능이 작동된다.

- **void file_write_function()**

  : 메인함수의 끝에서 (즉 사용자 혹은 관리자의 기능을 모두 수행하고, 모드를 선택하는 초기화면으로 돌아와 3.종료 를 입력했을 때) 프로그램 객체의 데이터를 텍스트 파일 machine_data.txt에 저장하는 함수
  : 한 줄씩 .txt 파일에 쓴다. 금고 정보를 모두 저장한 뒤에는 첫 번째 슬롯부터 마지막 슬롯까지의 음료 정보를 구성한다. 모든 데이터 저장이 완료되면 파일을 닫고, 저장이 완료되었다는 문구를 띄운다.

```
//쓰기용 파일 변수
ofstream datafile(filename);

//파일이 정상적으로 열렸는가
if (datafile.is_open()) {
        Machine* M = Machine::getInstance();

        //금고 정보 한줄씩 쓰기
        datafile << M->_mmoney->_100 << endl;
        datafile << M->_mmoney->_500 << endl;
        datafile << M->_mmoney->_1000 << endl;

        //음료 정보 한줄씩 쓰기
        for (int i = 0; i < DRINK_MAX_SLOT; i++) {
                Drink temp = M->_drink[i];
                datafile << (int)(temp._isRegist) << endl;
                datafile << temp._name << endl;
```

```
                datafile << temp._price << endl;
                datafile << temp._stock << endl;
        }
        //파일 닫기
        datafile.close();
        cout << "데이터 저장이 완료되었습니다...." << endl << endl;
        return true;
}
else {
        //파일 닫기
        datafile.close();
        cout << "파일 열기 실패! 파일 비정상종료 확인바람" << endl << endl;
        return false;
}
```

- 올바른 종료가 아닌 경우(모드 선택 화면의 3.종료를 입력하지 않는 모든 경우)에는 변경된 데이터가 정상적으로 저장되지 않는다.

3.2. 예외처리

- 기계의 금고에서
  - 0개 <=100원짜리 개수 =<100개
  - 0개 <=500원짜리 개수 =<50개
  - 0장 <=1000원짜리 장수 =<50장
- 음료 가격에서
  - 최소 100원, 최대 2000원으로 설정 가능
  - 100원 단위로 설정 가능
- 음료 이름에서
  - 글자 수는 영어 기준 20자
- 음료 슬롯의 수는 10개로 고정
- 음료 슬롯당 재고는 최대 15개
- 메뉴 선택 시, 메뉴 이외의 번호나 문자를 입력했을 경우

- **bool exception_test(int num, int max, int min)**
  : 최대와 최소를 검사해 int형 변수를 예외처리한다.

: 매개변수로 '**num** : 검사할 변수', '**max** : 최대값', '**min** : 최소값'을 차례로 입력받는다. 만약 변수가 정수형이 아니거나, 지정한 범위 [min, max] 미만이거나 초과이면 false값을 리턴한다. 검사할 변수가 옳은 형식이라면 true 값을 리턴한다.
:함수 내에서 int test1=1, test2=1, test3=1, test4=1 을 선언하고, 올바른 형식의 입력이 아니면 test1을 0으로, 올바른 형식의 입력이 아니므로 버퍼를 비워주면 test2을 0으로, 올바른 형식이지만 버퍼에 무엇인가가 남아있으면 test3을 0으로 바꾼다. 다음 test1~4를 모두 곱한 값이 0이 아니고, 매개변수로 받은 min~max사이의 값이라면 올바른 입력이므로 true를 리턴한다.
:이 함수는 min~max사이의 값이 아닌 값을 넣었을 때는 범위가 잘못되었다고 알려주고, 잘못된 형식의 값을 넣었을 때는 숫자를 입력해달라고 사용자에게 알려준다.(단 INT_MAX를 넘는 수는 문자로 판단한다.)

- **메뉴 선택 시 메뉴 이외의 번호나 문자를 입력했을 경우 (해당 화면에서 처리)**
  : 사용자가 올바른 입력을 할 때까지 반복문을 이용해 exception_test를 반복한다. 올바른 입력을 받으면 exception_test는 true를 리턴하고, 반복문을 탈출한다. 이 때 사용자가 입력한 값을 switch문에 넘겨준다. 혹시라도 올바른 입력이 아니라면 switch문의 default로 들어가지만 이미 올바른 입력만 switch로 들어오기 때문에 사실상 들어올 일 없는 분기이다.

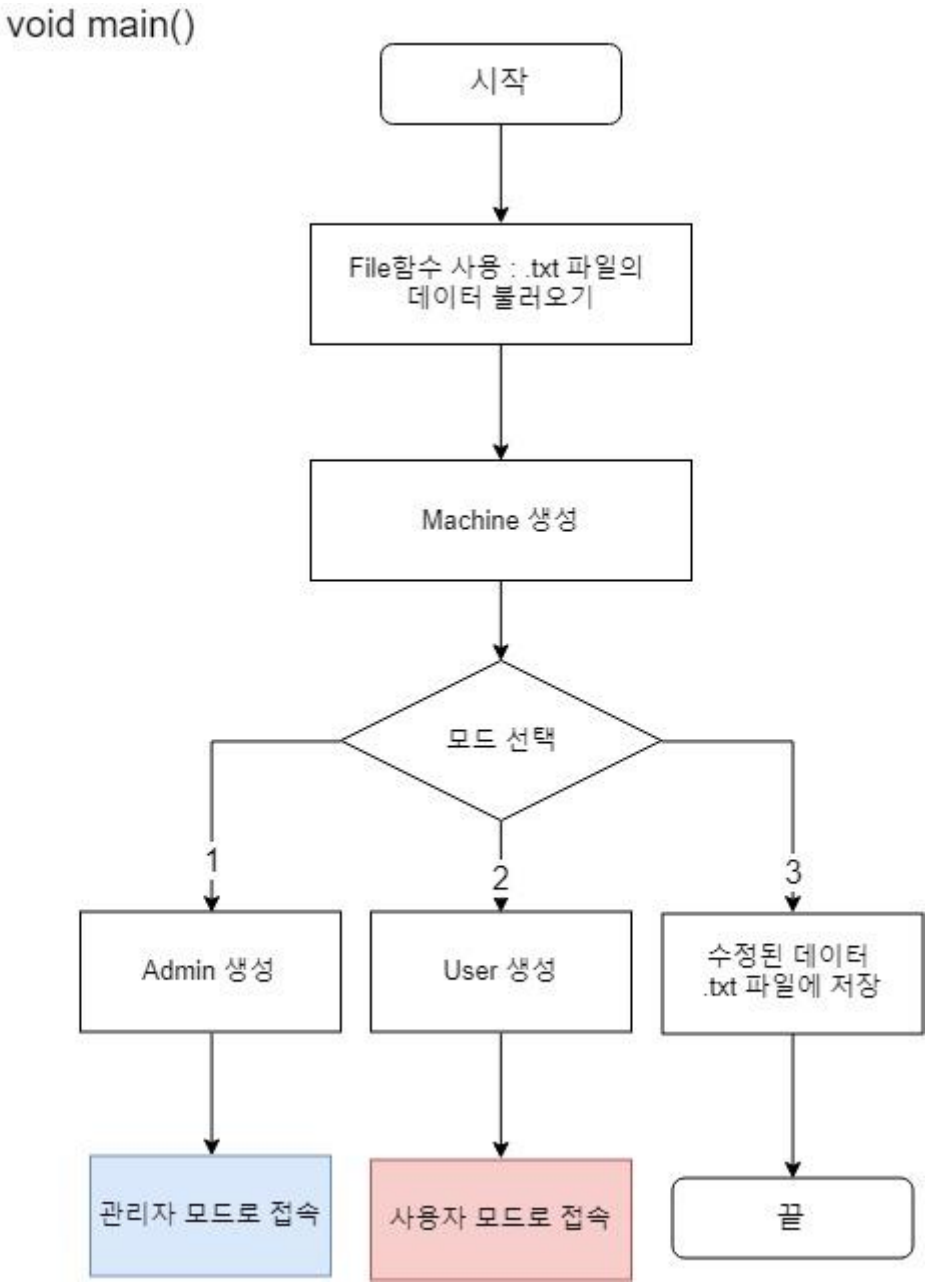
- **음료 이름이 20자를 초과했을 경우 (해당 화면에서 처리)**
  : 음료의 이름을 받는 함수로 getline을 이용하여 처리한다. getline()은 매개변수로 char*형 변수1, int형 변수2를 이용하는데, 변수1의 글자 수는 변수2를 넘지 못한다. 그러므로 변수2에 (int)를 사용해 20자가 넘으면 자동으로 무시한다. 입력을 받은 후 버퍼를 지워주고 ignore를 사용하여 int의 최댓값(2147483647)을 넘지 않는 모든 입력에 대해 입력이 가능하다.

4. 메인

4.1. 전체

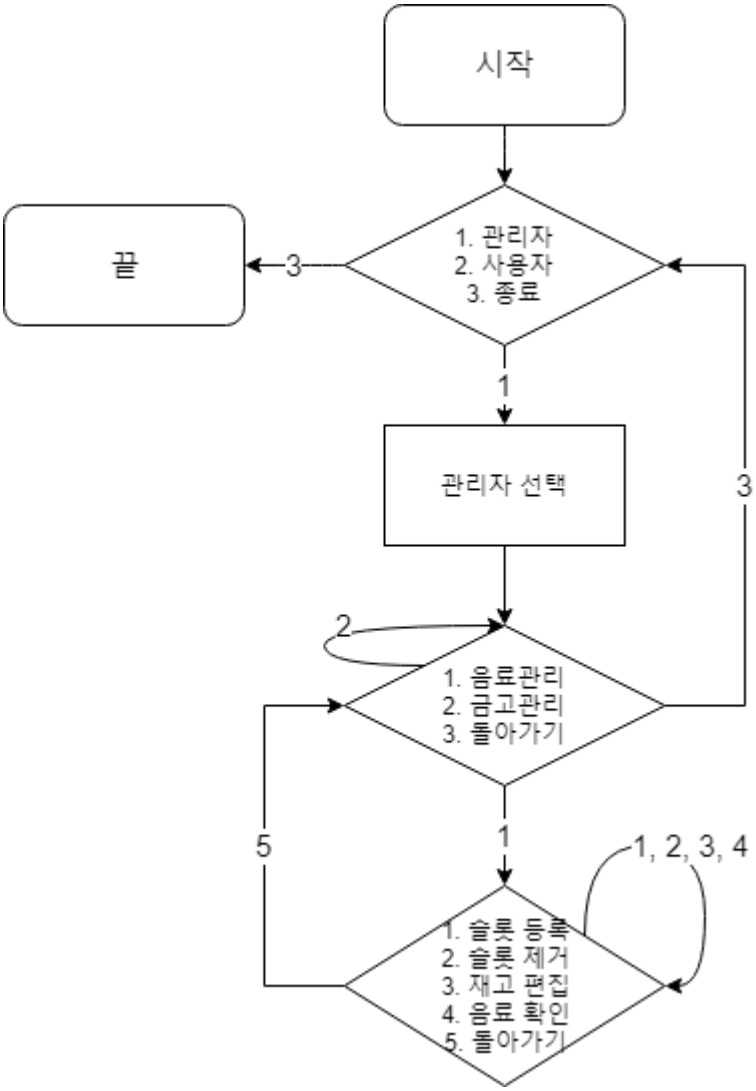
■ 순서도

: 관리자 모드와 사용자 모드 부분을 제외한 간략한 전체 순서도



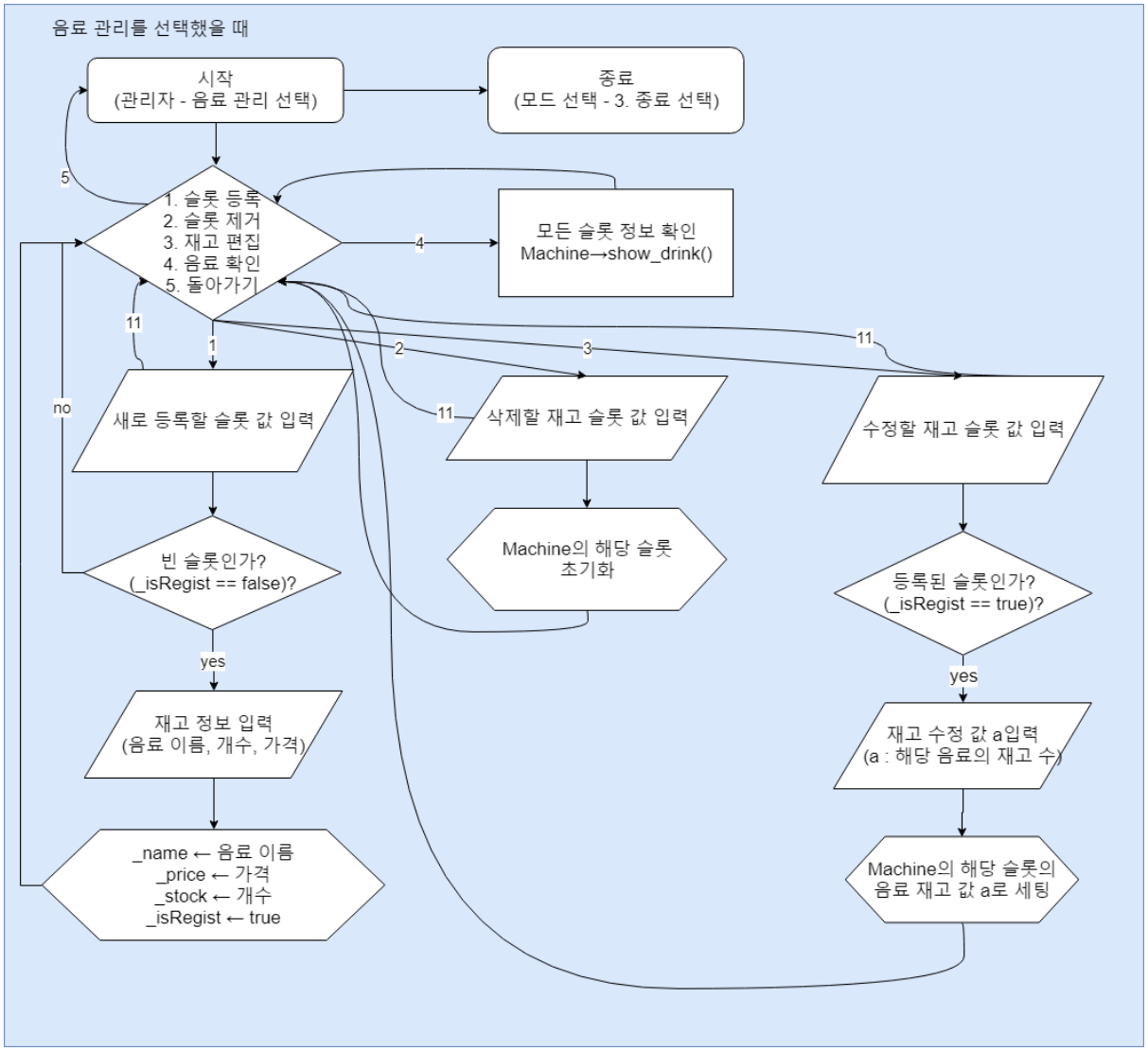
- 프로그램을 시작했을 때, 가장 최초로 데이터를 읽어와 각 객체에 저장한다.
- 프로그램이 **모드 선택**의 3으로 종료될 때 수정된 객체의 데이터를 .txt파일에 저장한다.

4.2. 관리자



- 세가지 메뉴를 입력 받는다.
- 1. 음료관리, 2. 금고관리, 3. 돌아가기가 있고 그 외의 입력은 다시 받는다.
    - 1,2에 대한 순서도는 이어서 자세히 설명한다.
- 음료 관리에서는 다시 네가지 메뉴를 입력 받는다.
- 1. 슬롯 등록, 2. 슬롯 제거, 3. 재고 편집, 4. 음료 확인, 5. 돌아가기가 있고 그 외의 입력은 다시 받는다.

■ 음료관리

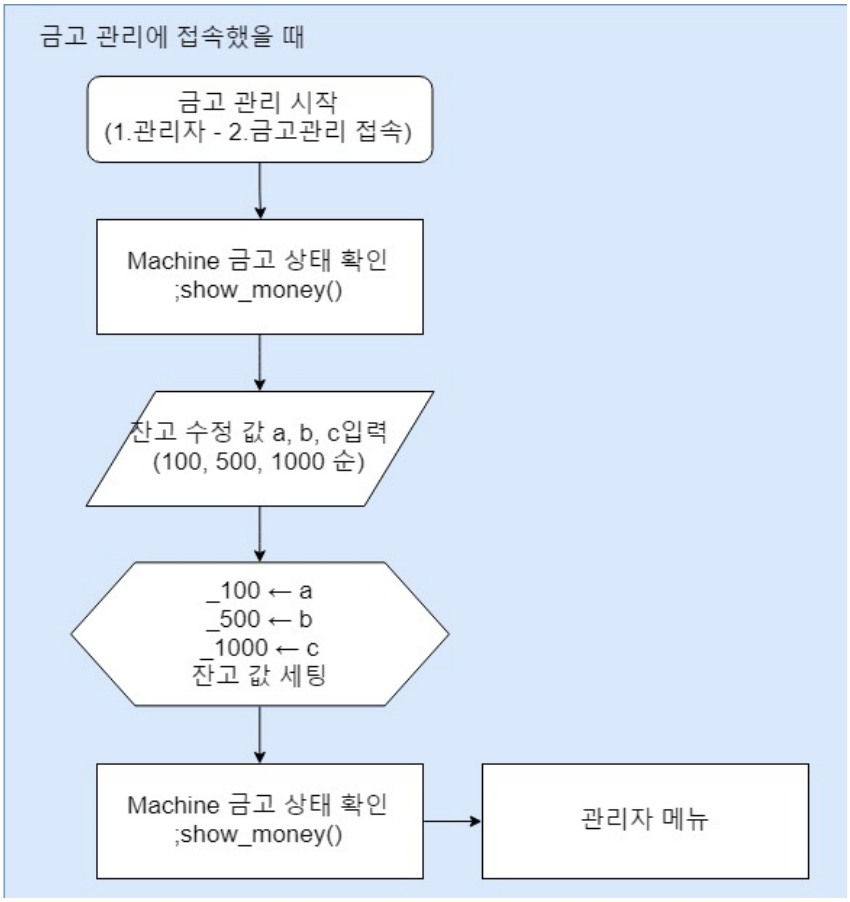


- 1. 슬롯 등록, 2. 슬롯 제거, 3. 재고 편집, 4. 음료 확인, 5. 돌아가기가 있고 그 외의 입력은 다시 받는다.
  - 1. 슬롯 등록 시 Machine->check_blank_slot()으로 인덱스순으로 맨 처음으로 발견되는 빈 슬롯을 찾는다.
    없다면(리턴이 -1이라면) 미등록 슬롯이 없음을 알리고 관리자 모드 종료.
    미등록 슬롯이 있다면 사용자로부터 슬롯 번호를 입력받고 해당 슬롯번호가 빈 슬롯인지 검사한 후 빈 슬롯이면 Admin->regist_slot() - Drink(name, price, stock)입력, 빈 슬롯이 아니면 다시 입력받기. 만약 11이면 음료 관리 메뉴 선택화면으로 돌아감.
    Machine의 해당 drink[index] 갱신.
  - 2. 슬롯 제거 시 Admin->select() 제거할 슬롯 번호 입력

Machine의 해당 drink[index] 초기화, 11이면 음료 관리 메뉴 선택화면으로 돌아감.

- ○ 3. 재고 편집 시 Admin->select() 편집할 슬롯 번호 입력, 11이면 음료 관리 메뉴 선택화면으로 돌아감. 11이 아니면, 등록된 슬롯인지 검사 후 등록됐다면 Admin->select()로 변경할 재고 수 입력
  Machine의 해당 drink[index] 갱신
  등록 안됐다면 메뉴로 돌아가기
- ○ 4. 음료 확인 시
  Machine->show_drink()

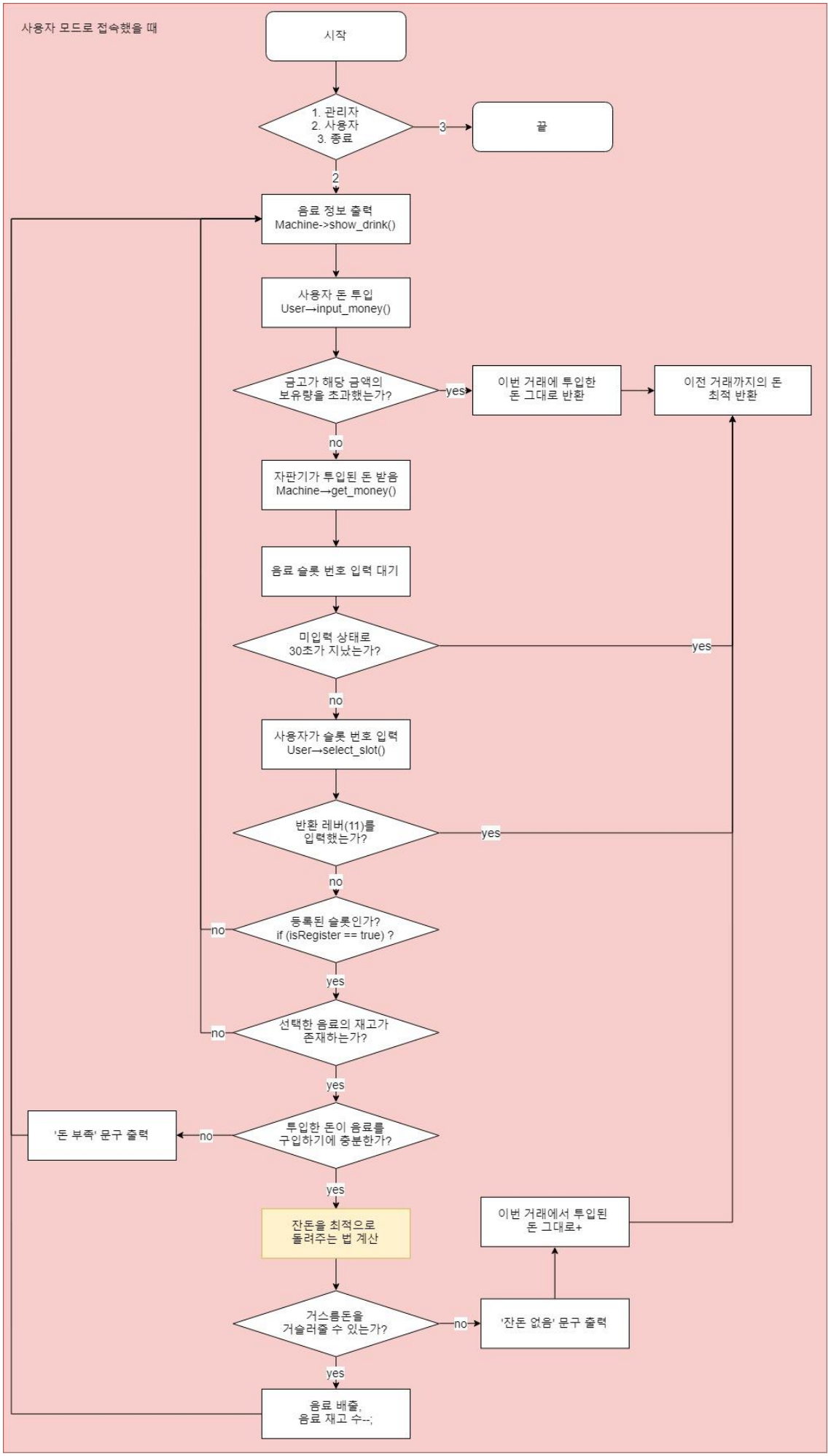
■ 금고관리



- ● 금고관리에서는 별도의 메뉴를 입력받지 않고, 금고 확인과 금고 편집이 동시에 이루어진다.
  - ○ 금고 확인 시 Machine->show_money()
  - ○ 금고 편집 시 Admin->edit_money()

4.3. 사용자

■ 사용자 기능


사용자 모드로 접속했을 때
시작
1. 관리자
2. 사용자
3. 종료
3
끝
2
음료 정보 출력
Machine->show_drink()
사용자 돈 투입
User→input_money()
금고가 해당 금액의
보유량을 초과했는가?
yes
이번 거래에 투입한
돈 그대로 반환
이전 거래까지의 돈
최적 반환
no
자판기가 투입된 돈 받음
Machine→get_money()
음료 슬롯 번호 입력 대기
미입력 상태로
30초가 지났는가?
yes
no
사용자가 슬롯 번호 입력
User→select_slot()
반환 레버(11)를
입력했는가?
yes
no
등록된 슬롯인가?
if (isRegister == true) ?
no
yes
선택한 음료의 재고가
존재하는가?
no
yes
투입한 돈이 음료를
구입하기에 충분한가?
'돈 부족' 문구 출력
no
yes
잔돈을 최적으로
돌려주는 법 계산
이번 거래에서 투입된
돈 그대로+
거스름돈을
거슬러줄 수 있는가?
no
'잔돈 없음' 문구 출력
yes
음료 배출,
음료 재고 수--;

- Machine->show_drink()로 음료 보여주기
- User->input_money()
- 사용자가 입력한 돈이 기계가 수용할 수 있는 개수를 초과하면 즉시 방금 투입한 돈을 그대로 반환하고, 또 한번 지금까지의 거래 잔돈을 최적으로 반환한다. (2번 반환)
- 기계가 투입된 금액의 개수를 수용할 수 있다면 Machine->get_money()
- User->select_slot()
- 슬롯이 등록된 슬롯인지, 슬롯에 재고가 있는지, 투입 금액이 가격 이상인지 체크
- 이 모든 체크를 넘기지 못하면 투입받은 돈은 그대로 두고 거래 처음으로 돌아감
- Machine->calculate()
- Machine->optimal_change()가 return false를 하면 기계 내 잔돈 없음이므로 받은 돈 그대로와 지금까지의 거래 잔돈의 최적반환을 **더해** 반환하고 (Machine->return_change()), 모드 선택화면으로 돌아감
- Machine->optimal_change()가 return true를 하면 optimal_change로 최적의 방법으로 돈을 돌려주고(Machine->return_change()), 음료 재고를 하나 줄임.