Vending Machine

Problem Statement

→ load and configure
→ denominations accepted 1, 2, 5 & 10
→ user can select products to create an order
→ user can modify the order
→ insufficient amount & over amount have to be handled

low level design

Requirements

1. load and configure the vending machine
2. denominations accepted → 1, 2, 5 & 10
3. user can select items
4. user can modify the order
5. user can pay the amount
6. the system handles the amount paid

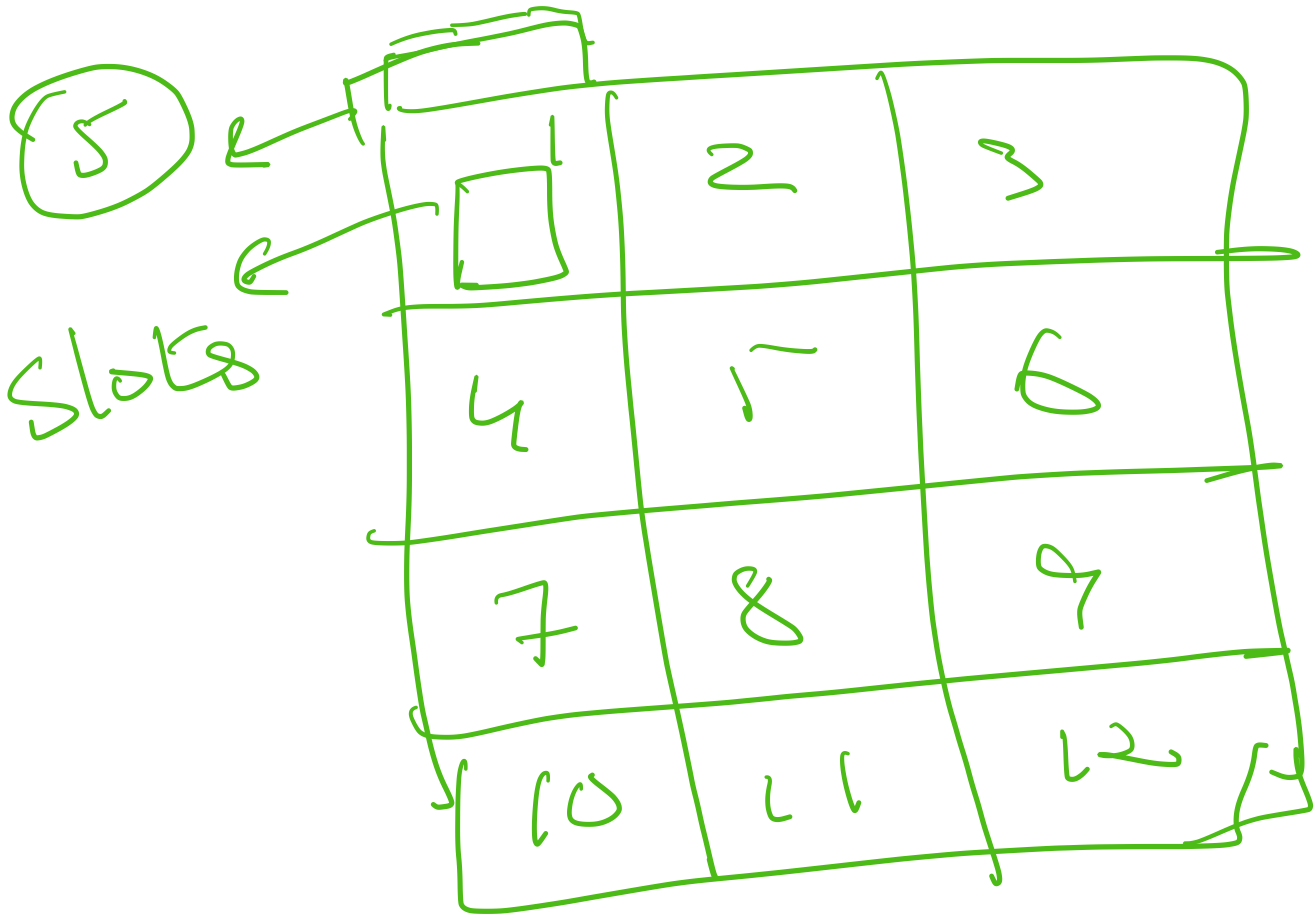
Flows:

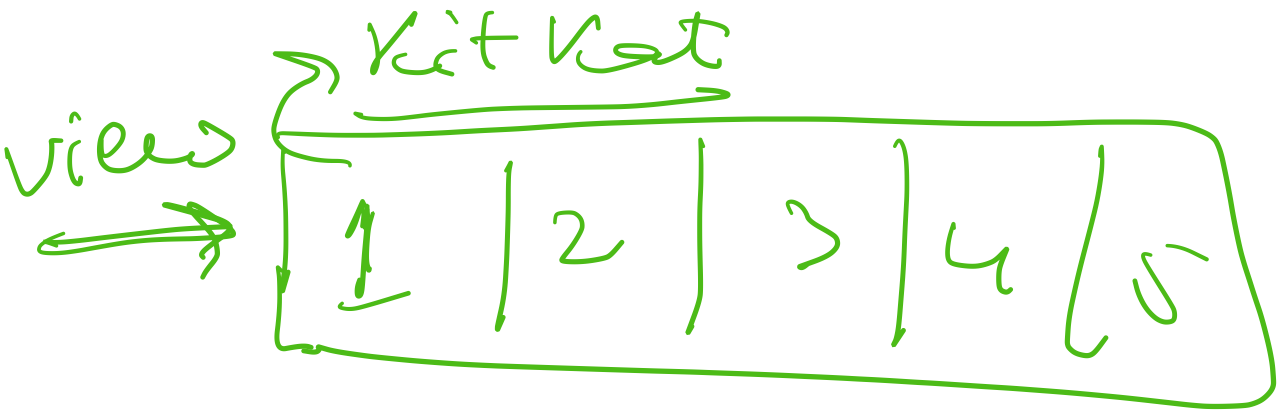
(1) load and configure the vending machine

(2) A user can place an order and pay

Load & configure

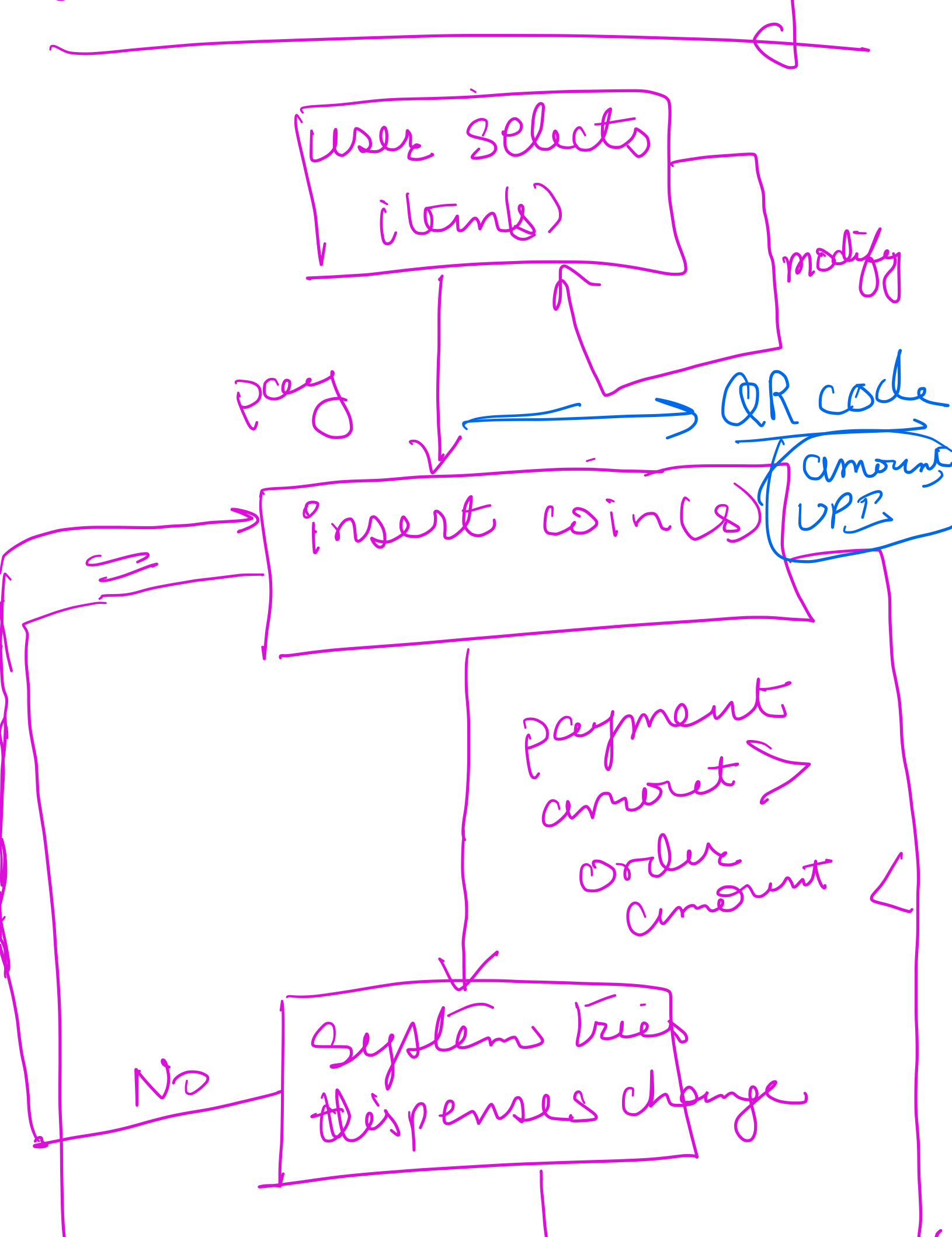
(1) An admin can configure the machine → add the information about no of slots & items per slot.





(2) Caduby dairy milk
5 qty -

User Order Journey


user selects item(s)
modify
pay
QR code
amount
UPI
insert coin(s)
payment amount
order amount
System tries dispenses change
No

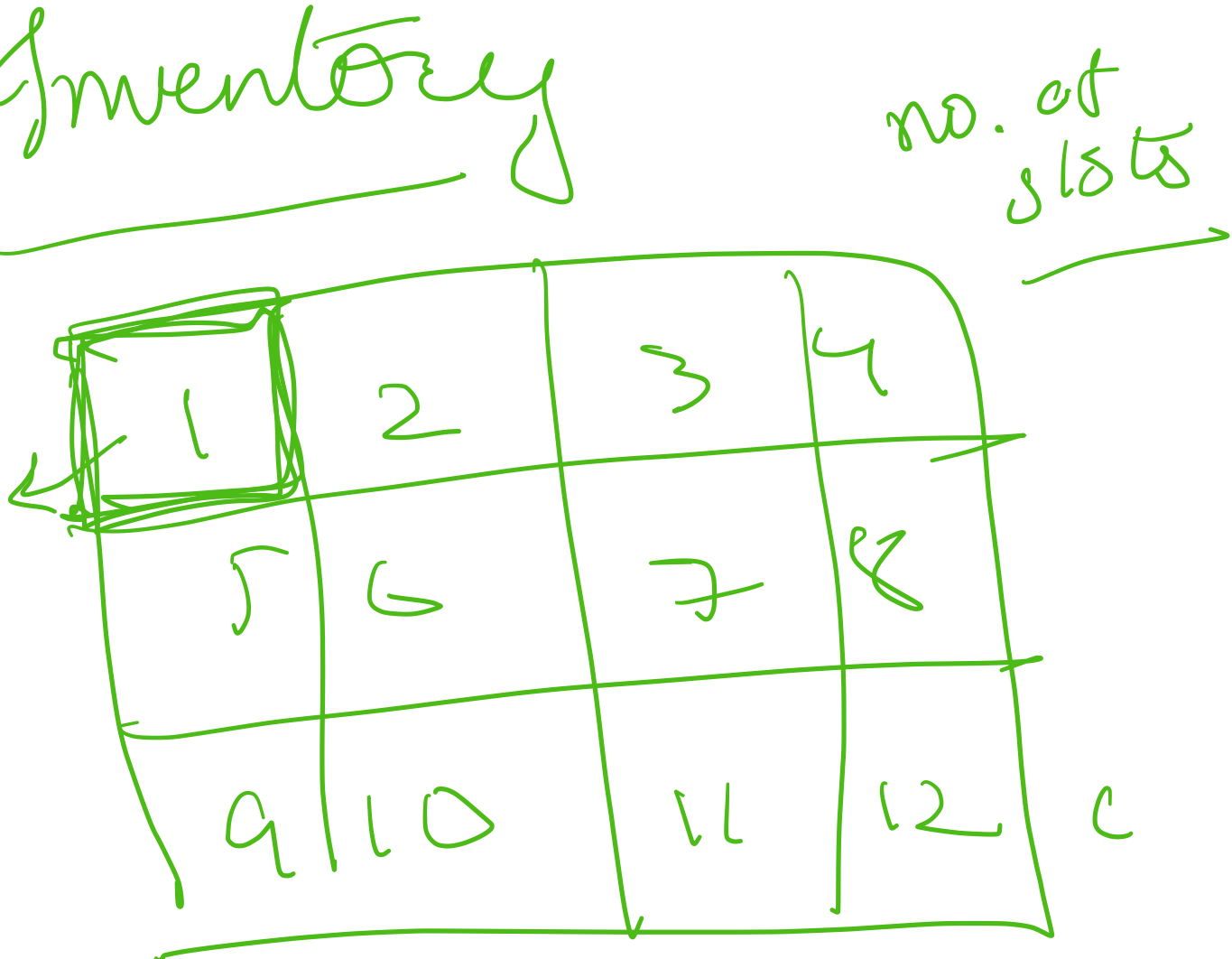
ask for adding more items

System dispenses the selected items

Yes

dispense change

Inventory



Product → capacity
↳ qty

12 slots
12 different products
Assume capacity of
each slot = 10

10 qty of same
product.

Product →

↳ price

Slot
↳ Product
↳ Qty

1: (KitKat(5), 10)
2: (Pepsi(15), 10)

Vending Machine

→ no. of slots

→ qty per slot

→ List<Slot> slots

public VM (# slots, qtyPer slot)

{

}

}

class Slot {

Product product;

int qty

}

```
class VendingMachine {
    int noOfSlots;
    int qtyPerSlot;
    Map<Product, Slot>
    List<Slot> slots;
    Map<DEnum, int> qty
    public VM(int noOfSlots,
              int qtyPerSlot) {

    }

    public void configure
        (List<Product>
            products ) {
    List<Slots> slots =
```

```
for (Product p : ...
{
    Slot s = new Slot(
        p, this.qtyPerSlot)
    slots.add(s);
}
}

public void load(
    Slot s, int qty) {
    slot.setQty(qty);
}
void dispense change
```

}

void addCoins(
Map<DEnum, int>
coinMap);

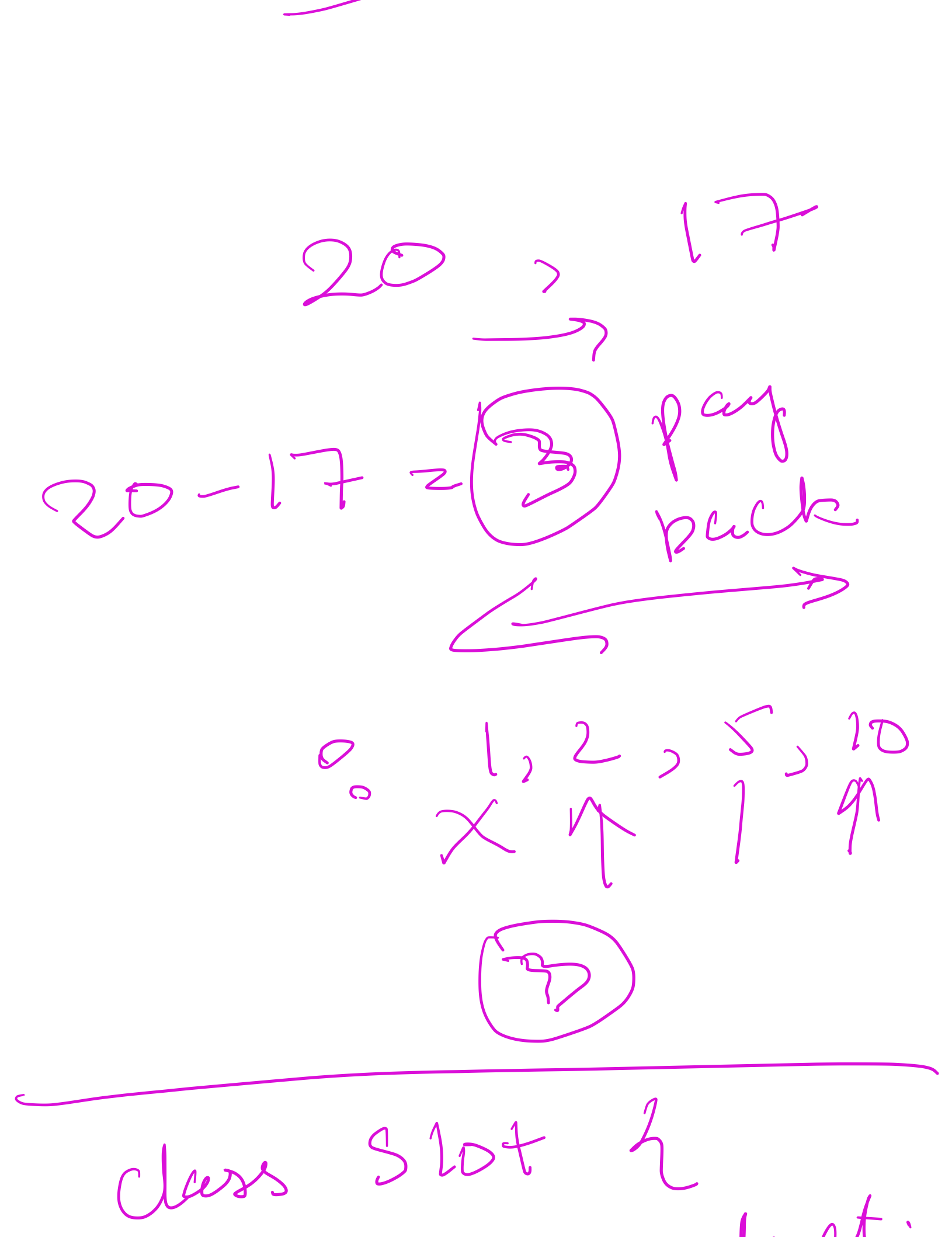
}

<10:2>

<0, 3, >

10 x 2 + 1 x 3

= 23



class slot 2

Product product)

int qty; >=

add (int qty)

dispense (int qty)

}

---

class Product {

String name;

int price;

}

class Order {

Map<Product, qty>
productqty;

public int getAmount() {

int total;

for (P p : Product

```
        int q = p.getQty() * p.getPrice();
        total += q;
    }
    return total;
}

public void resetOrder() {
    // init is back with a new ArrayList
}

public void addItem(Slots slots) {
    // add +1 qty to the product in
```

the slot

}

public boolean validate
order() {

// looks for each
product & checks
for available qty

}

coins = [1, 10, 5, 1,
2]

Order

collect coins

class Payment {
Map<Enum, qty> coins;
}

void addCoins(DEnum d) {
}

int getTotal() { }

Client

1) load & configure vending machine

```
VendingMachine vm =
    new VendingMachine(
      16, 10);
Product<products>
    = new A.L();
products.add(new
```

Product("Coke", 10);

// add more products

vm.configure(products);

(2) user can selects items, to create an order

order O = new order();

O.addItem(P1);

// add more items to the order

(3) user can add coins for payment

```
payment p = new
    Payment ();
p.addCoin (5);
// add more coin.
```

(4) system checks for payment amount & order

amount

```
if (o.getTotal() ==
    p.getTotal())

}
// check for items
   in vm
// despense items
} else if (>)

}
// check whether
```

able to dispense
change

// check for items
in inventory
// dispense items
// dispense change

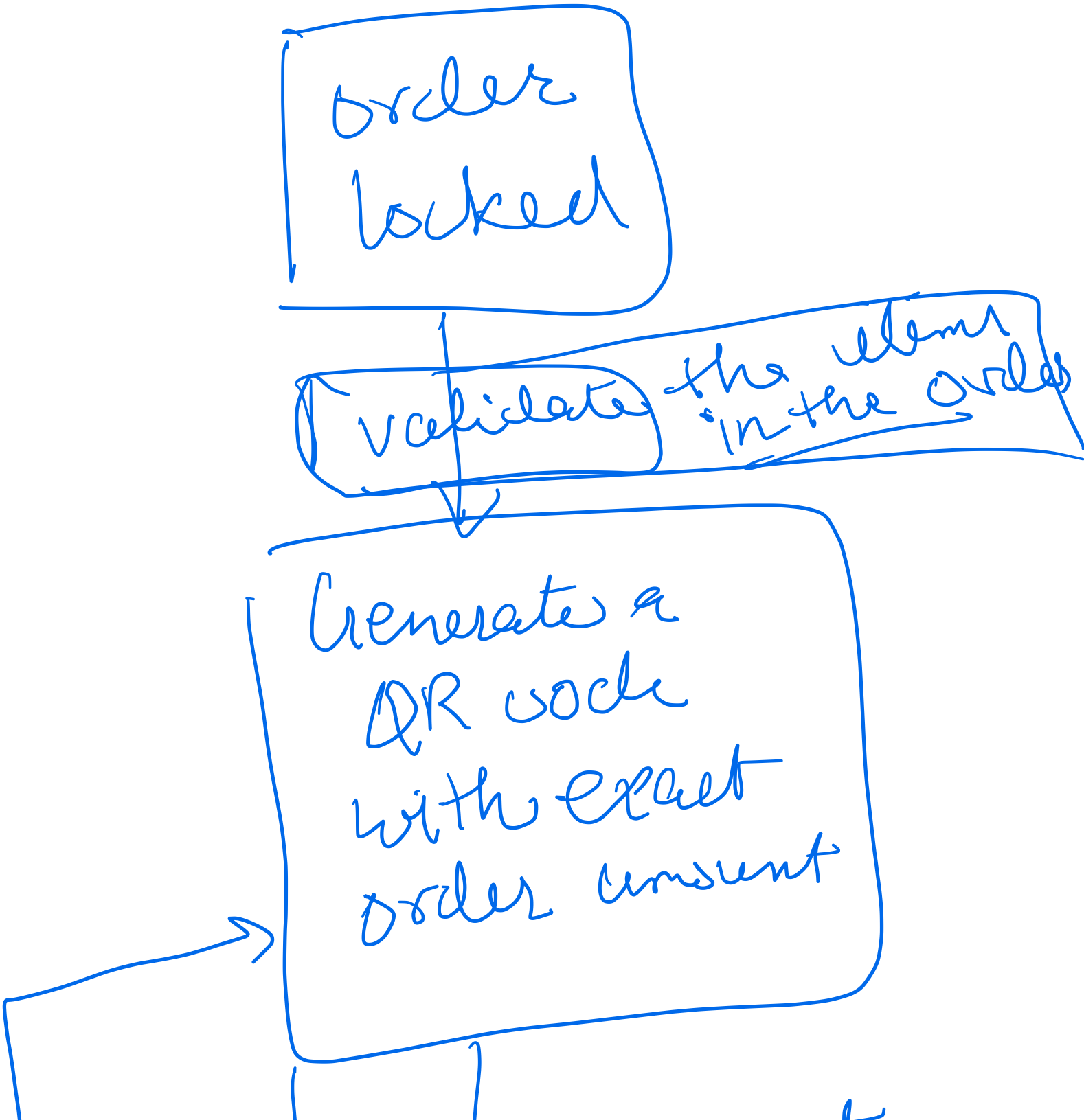
} else {

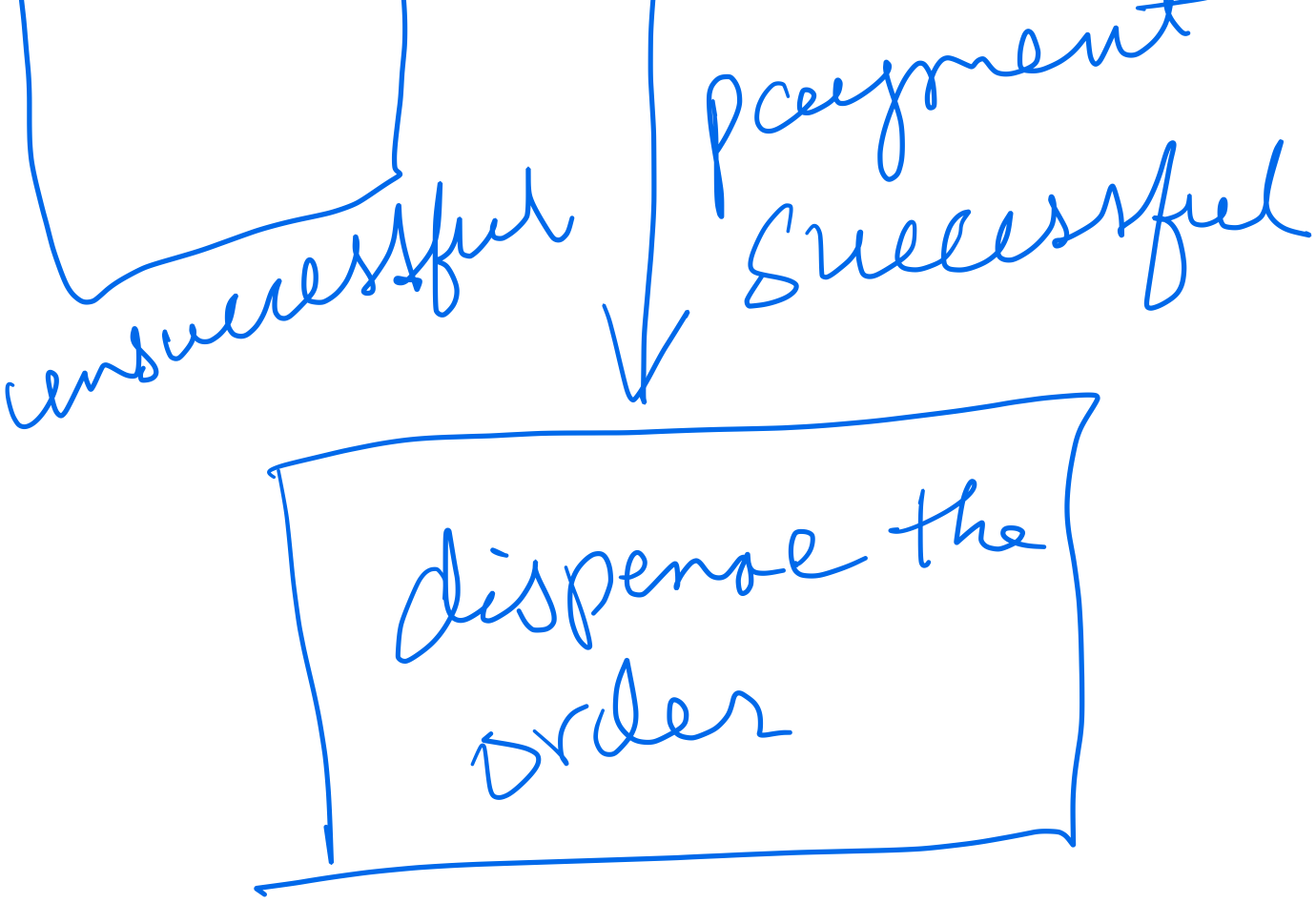
// ask the user
to add more
coins or reset
the order

}

// check whether items in inventory

// check whether we have change


order locked
validate
the items in the order
Generate a QR code with exact order content


unsuccessful
payment
Successful
dispense the
order