OEP

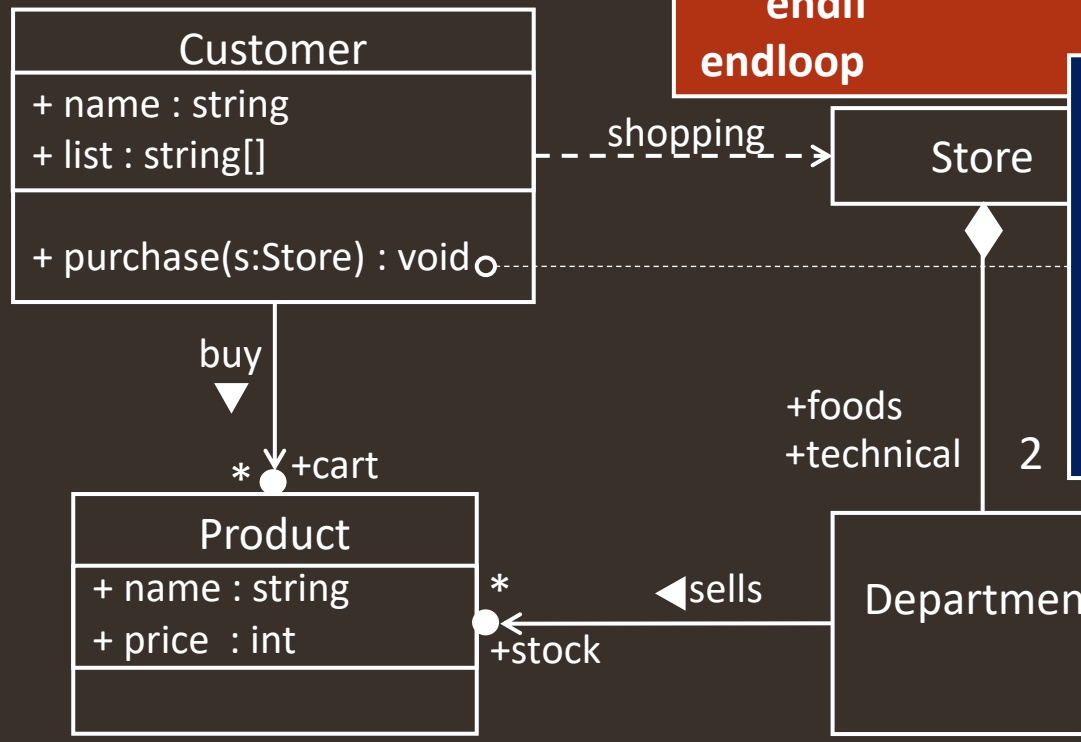
10. táblás gyakorlat

Feladatok

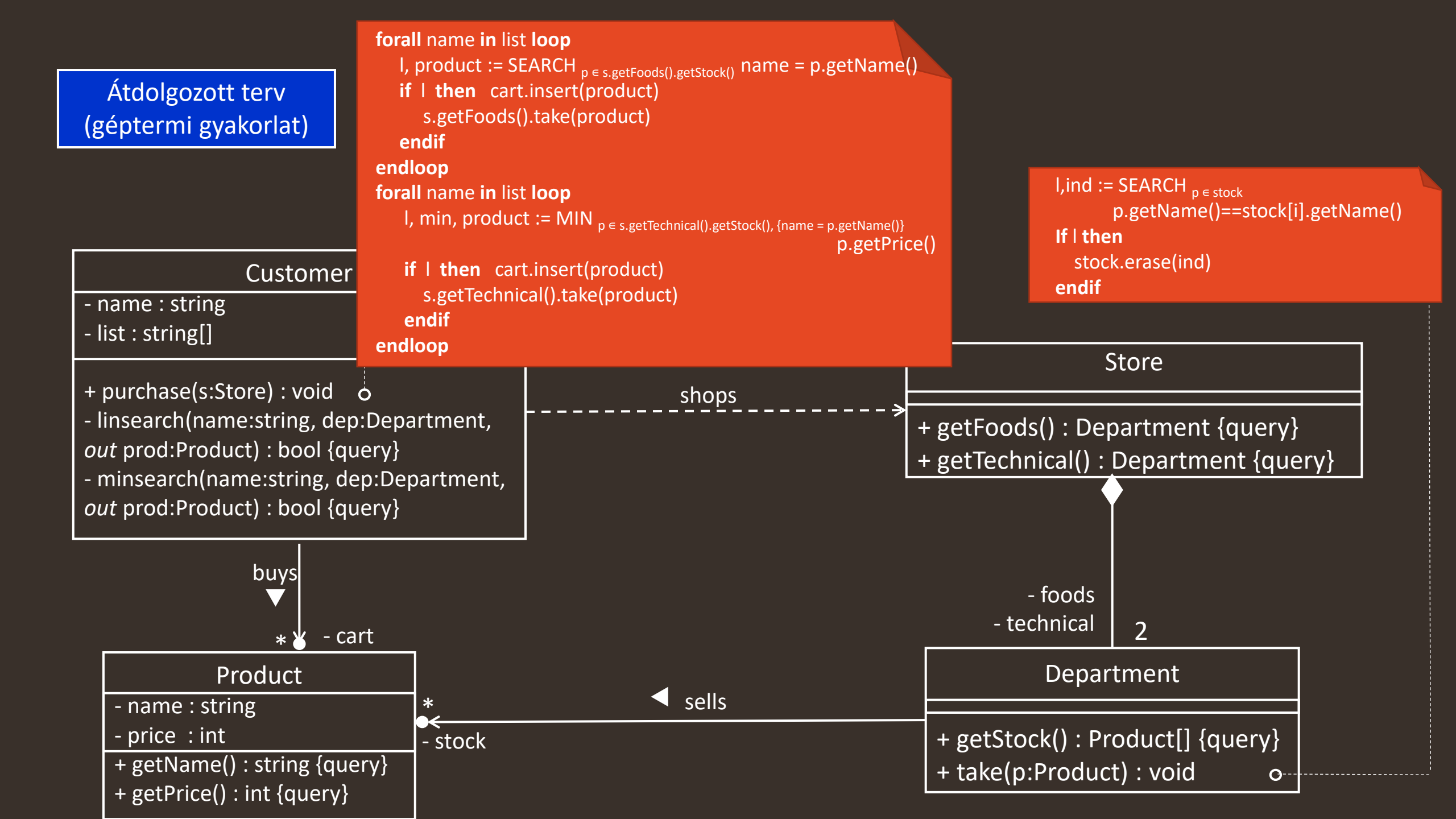
- Vásárlás
- Bankautomata
- Futár

2. Egy kisvárosi üzlet élelmiszer részlegből é ely azon termékek neveit tartalmazza, amit megve az élelmiszer részlegen nézik végig a teljes b sarukba), majd a műszaki részlegen ezt megismétlik s van a részlegen, akkor a legolcsóbbat választják

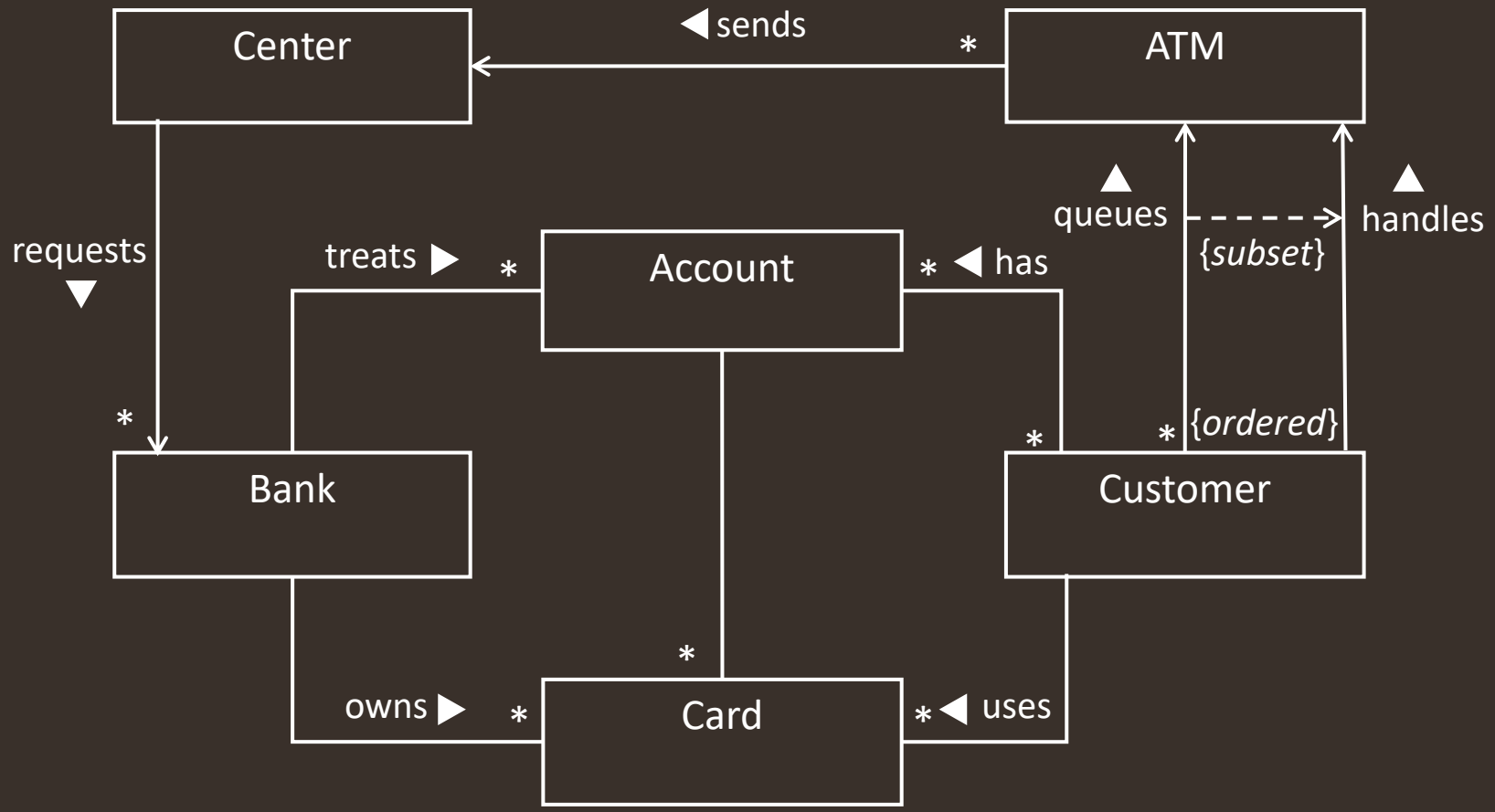
```
forall name in list loop
    l, product := SEARCH p ∈ s.foods.stock  name = p.name
    if l then  cart.insert(product)
               s.foods.remove(product)
    endif
endloop
forall name in list loop
    l, min, product := MIN p ∈ s.technical.stock {name = p.name}  p.price
    if l then  cart.insert(product)
               s.technical.remove(product)
    endif
endloop
```

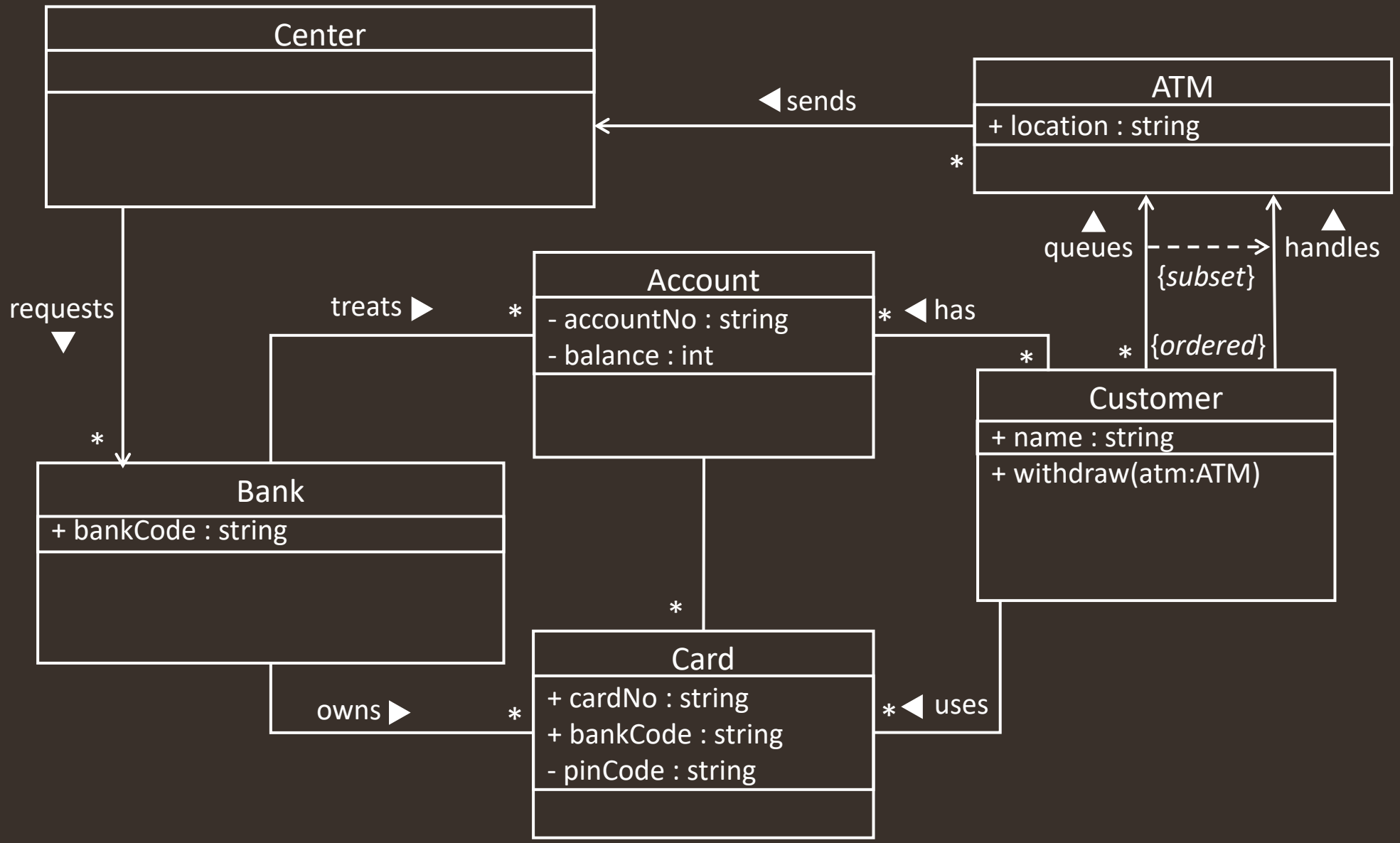


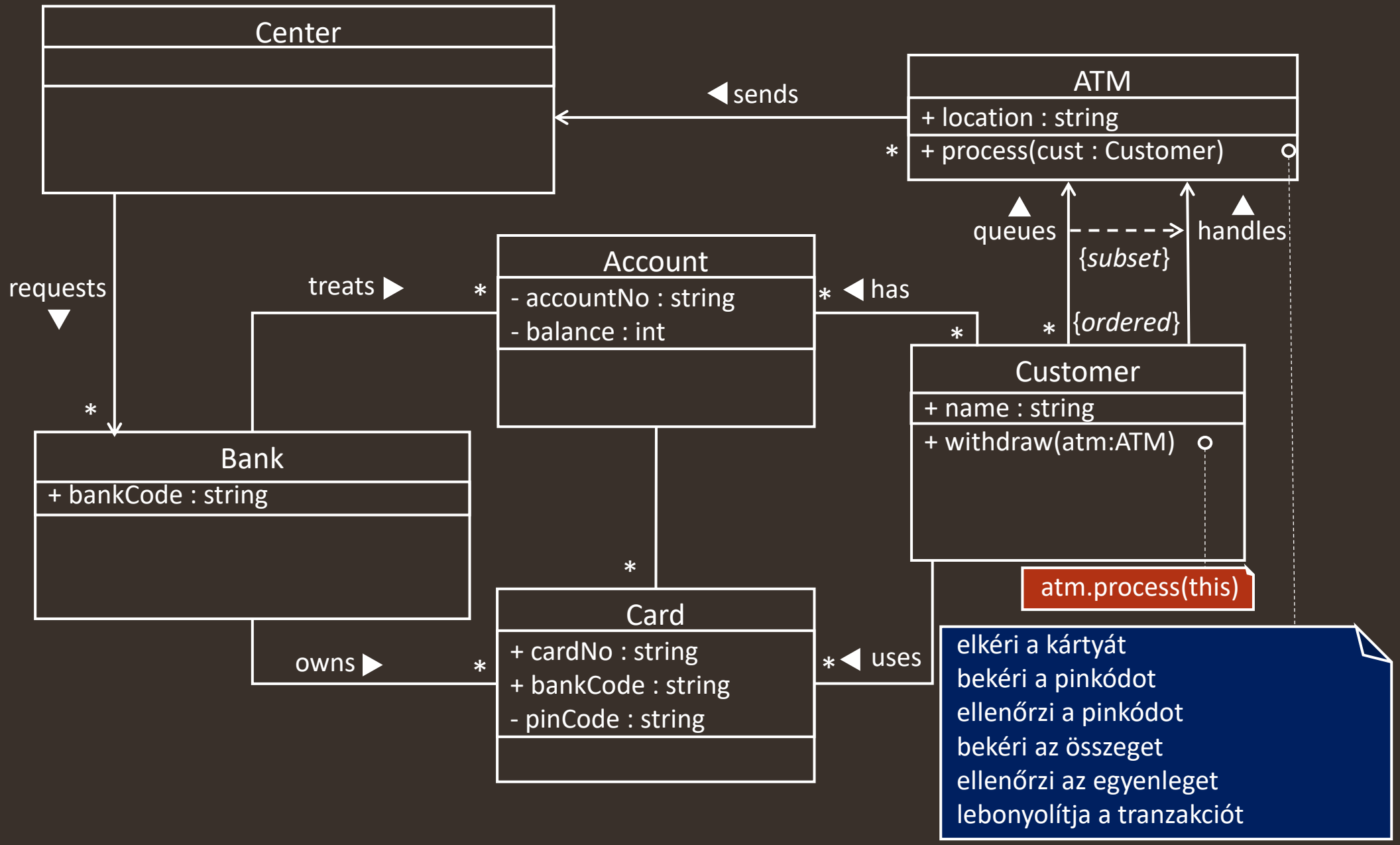
Először az élelmiszer részlegen nézi végig a listáját, és megpróbál annak minden tételével azonos nevű terméket találni. Ha sikerül, beteszi a terméket a kosarába.
Utána a műszaki részlegen nézi végig újra a listáját, és annak minden tételére megkeresi a tétel nevével azonos nevű legolcsóbb terméket, és beteszi a kosarába.

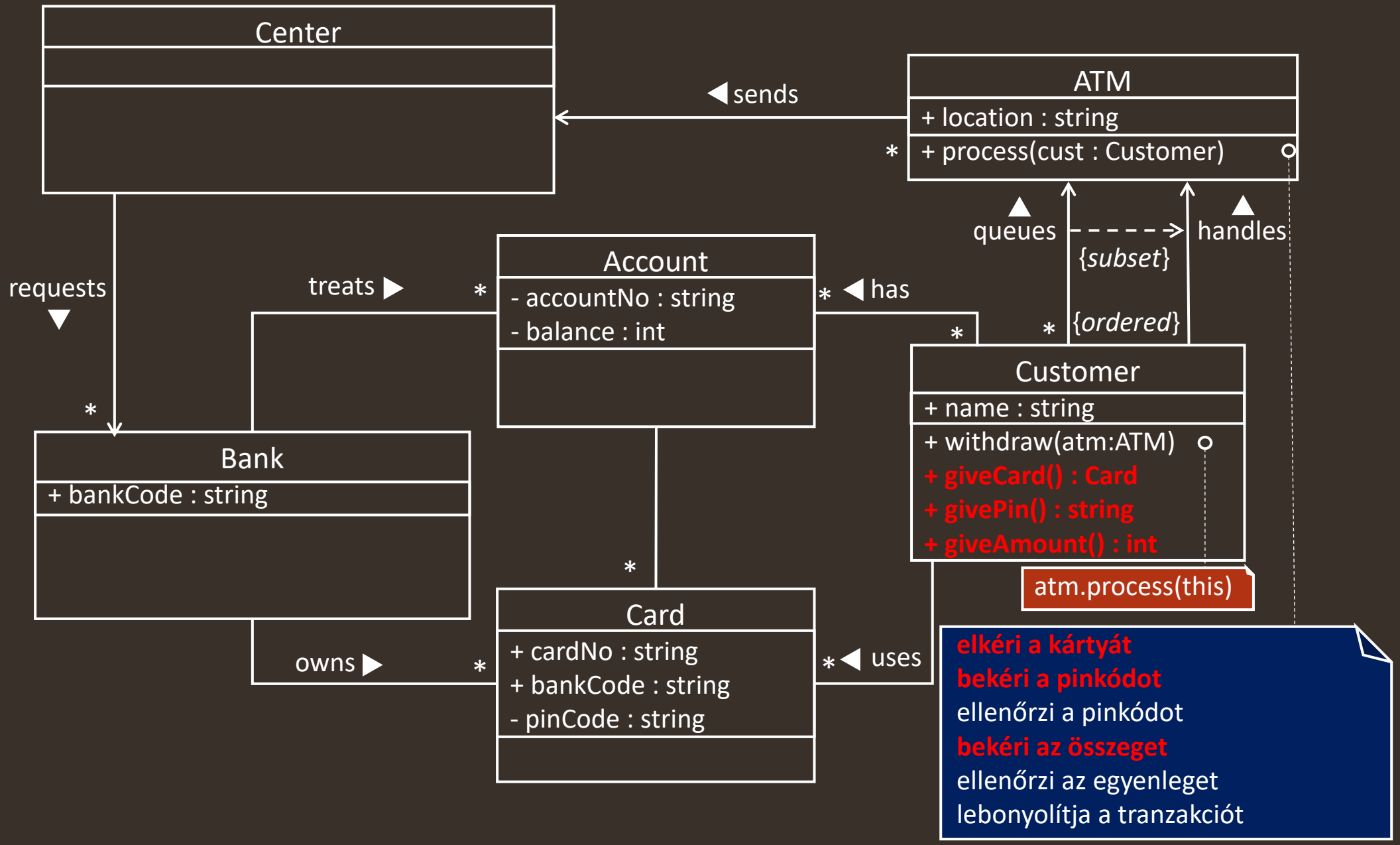

Átdolgozott terv
(géptermi gyakorlat)
forall name in list loop
l, product := SEARCH p ∈ s.getFoods().getStock() name = p.getName()
if l then cart.insert(product)
s.getFoods().take(product)
endif
endloop
forall name in list loop
l, min, product := MIN p ∈ s.getTechnical().getStock(), {name = p.getName()}
p.getPrice()
if l then cart.insert(product)
s.getTechnical().take(product)
endif
endloop
l,ind := SEARCH p ∈ stock
p.getName()==stock[i].getName()
If l then
stock.erase(ind)
endif
Customer
- name : string
- list : string[]
+ purchase(s:Store) : void
- linsearch(name:string, dep:Department, out prod:Product) : bool {query}
- minsearch(name:string, dep:Department, out prod:Product) : bool {query}
shops
Store
+ getFoods() : Department {query}
+ getTechnical() : Department {query}
buys
* - cart
- foods
- technical
2
Product
- name : string
- price : int
+ getName() : string {query}
+ getPrice() : int {query}
*
- stock
sells
Department
+ getStock() : Product[] {query}
+ take(p:Product) : void

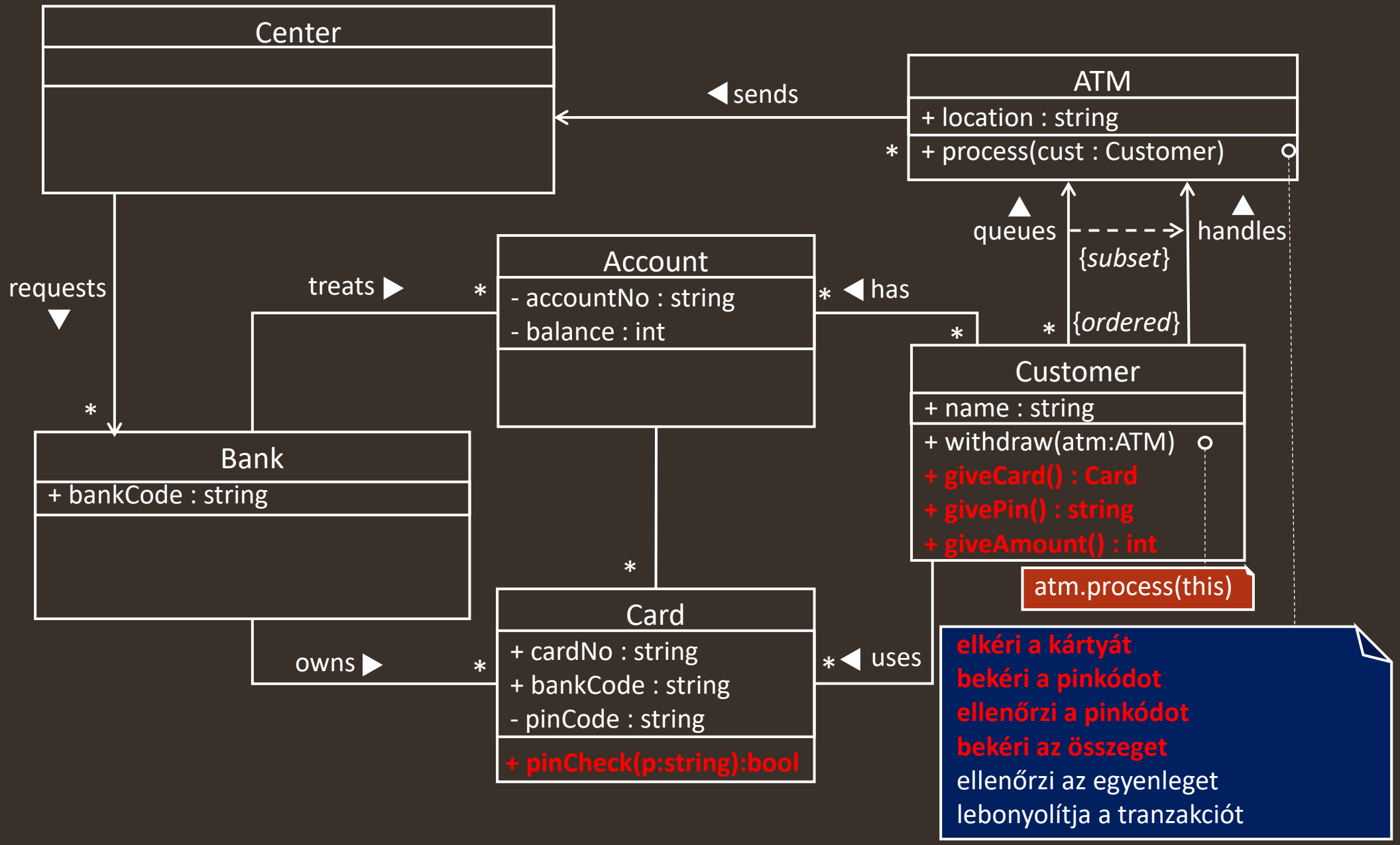
3. Egy ATM automatánál sorban állnak az ügyfelek, hogy pénzt vehessenek fel. Az ügyfelek rendelkeznek bankkártyákkal, amikhez tartozik egy PIN kód, valamint egy bankszámla. Az ügyfelek sorban vehetnek fel egy adott összeget az automatából a bankkártyájuk, valamint a kódjuk megadásával. Ha a kód megegyezik a kártya kódjával, akkor az automata kiadja az összeget, feltéve, hogy az összeget levonva az egyenlegből az továbbra is pozitív marad. Ennek megállapításához az automata egy központon keresztül a kártya adatainak megadásával lekérheti az ügyfél egyenlegét, illetve elküldhet egy jelentést a lebonyolított tranzakcióról, amely alapján a bank leveszi az összeget az ügyfél számlájáról.

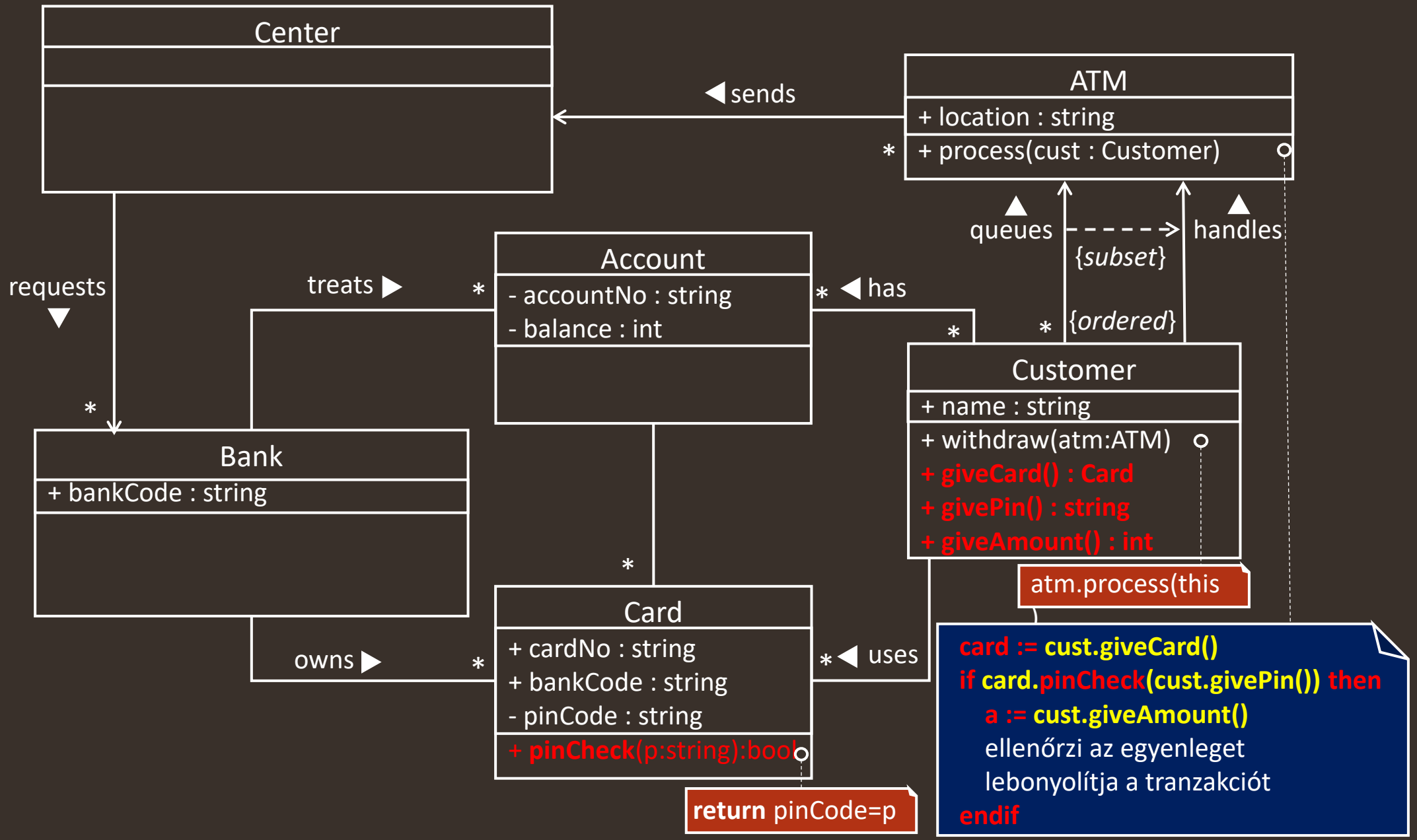


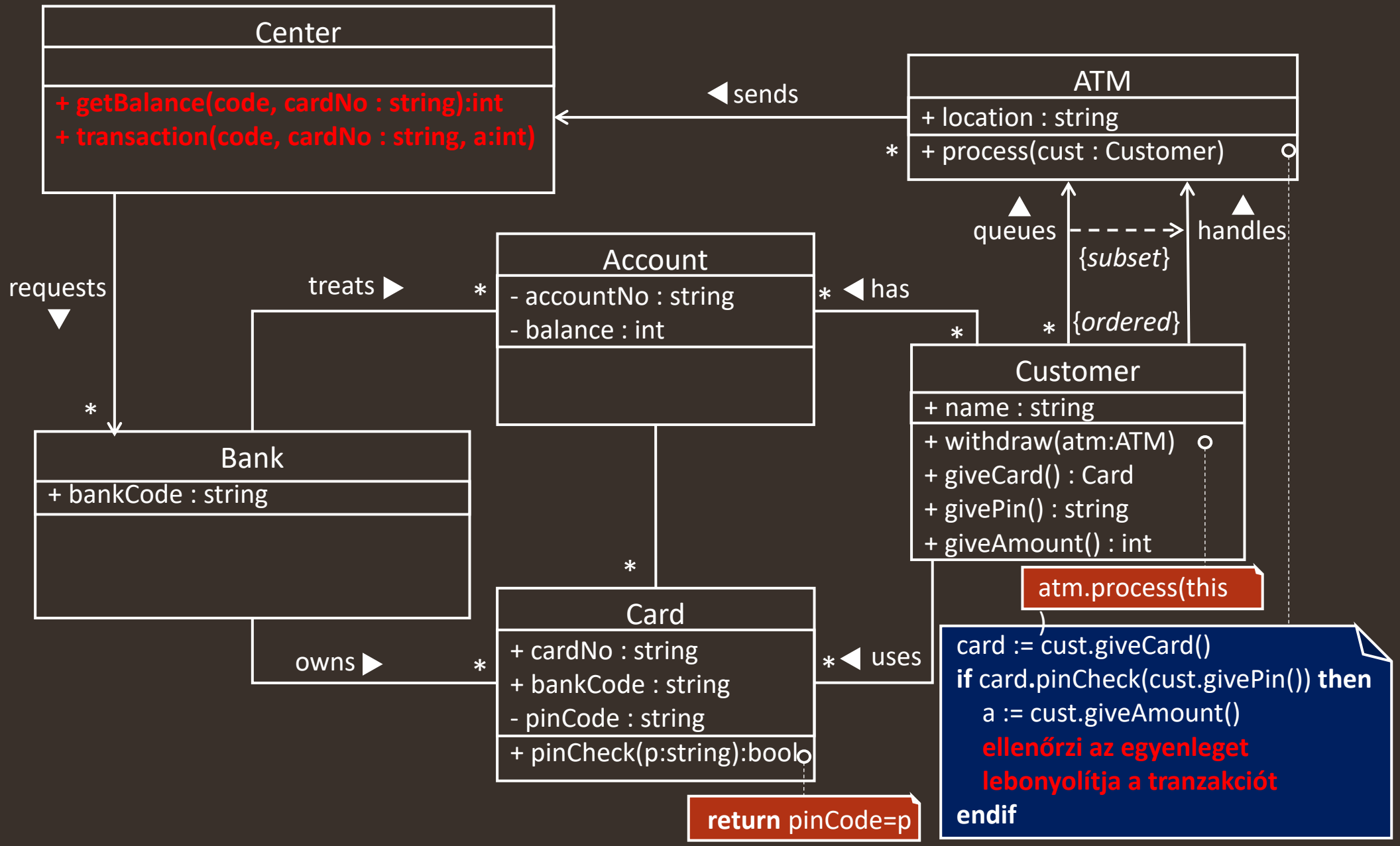

Center
ATM
+ location : string
sends
*
queues
{subset}
handles
{ordered}
requests
treats
*
Account
- accountNo : string
- balance : int
*
has
*
Customer
+ name : string
+ withdraw(atm:ATM)
*
Bank
+ bankCode : string
*
Card
+ cardNo : string
+ bankCode : string
- pinCode : string
owns
*
*
uses

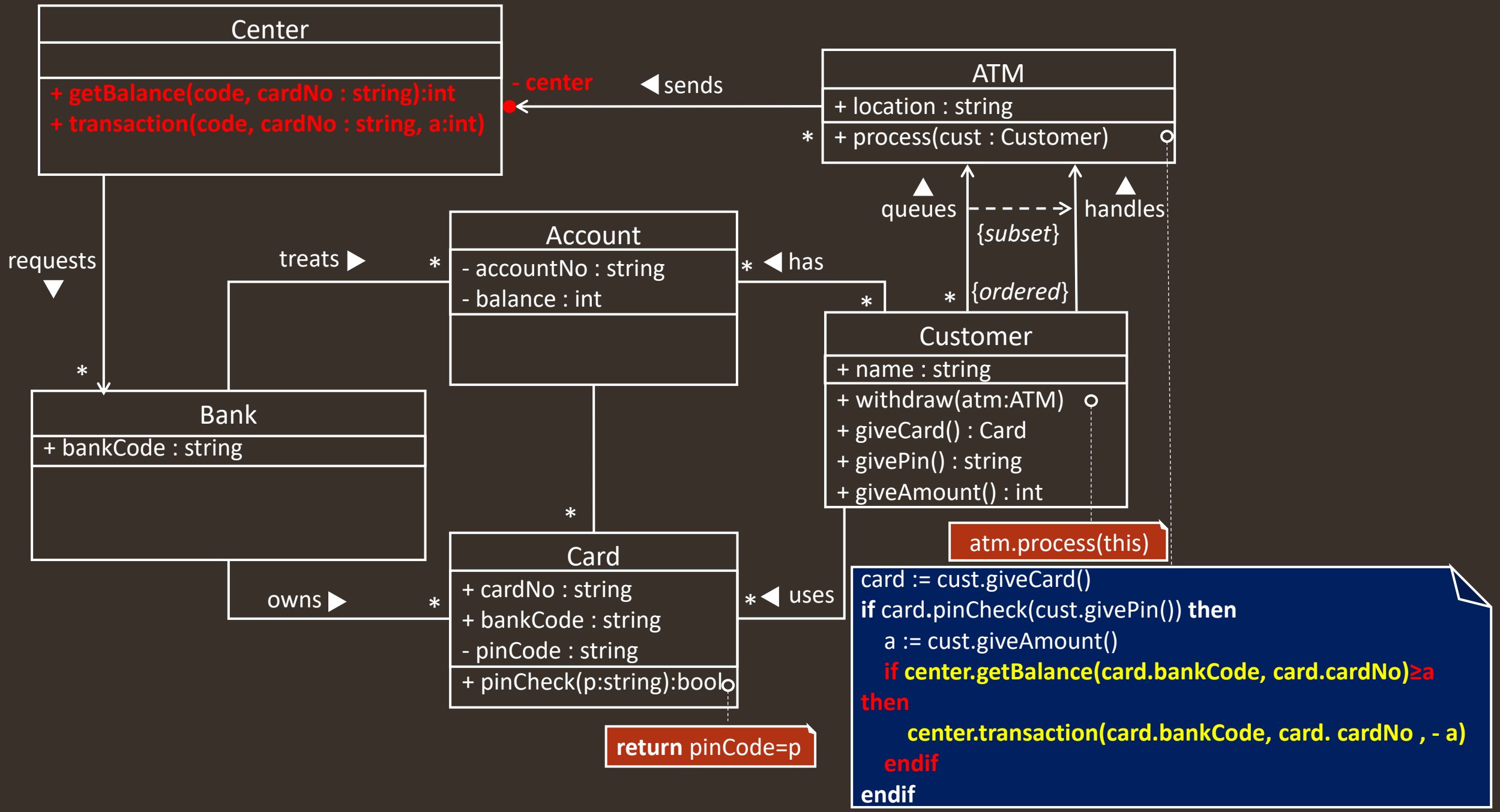

Center
ATM
+ location : string
+ process(cust : Customer)
sends
queues
{subset}
handles
{ordered}
requests
treats
Account
- accountNo : string
- balance : int
has
Customer
+ name : string
+ withdraw(atm:ATM)
Bank
+ bankCode : string
atm.process(this)
owns
Card
+ cardNo : string
+ bankCode : string
- pinCode : string
uses
elkéri a kártyát
bekéri a pinkódot
ellenőrzi a pinkódot
bekéri az összeget
ellenőrzi az egyenleget
lebonyolítja a tranzakciót

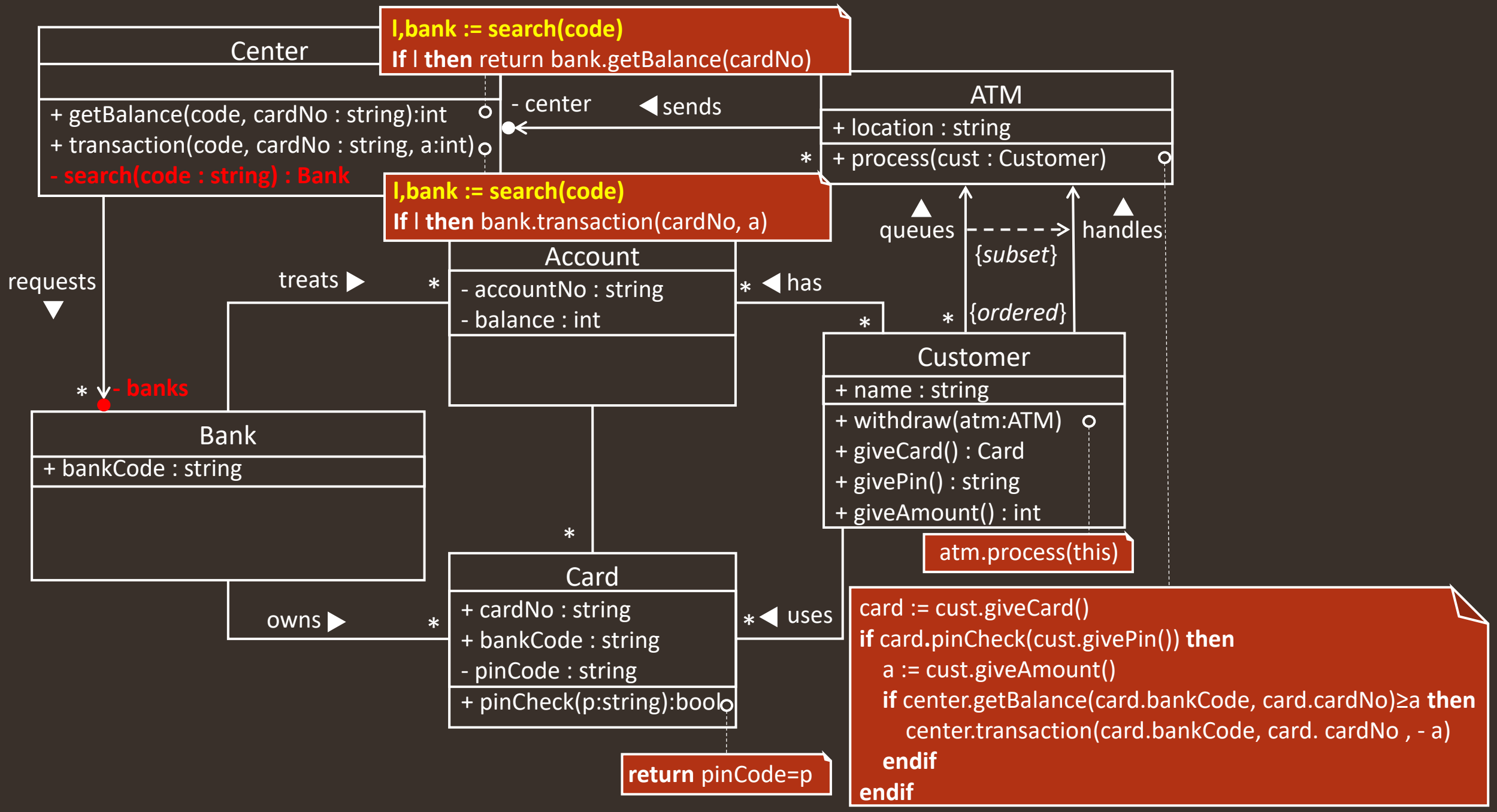

Center
ATM
+ location : string
+ process(cust : Customer)
sends
queues
{subset}
handles
{ordered}
requests
treats
Account
- accountNo : string
- balance : int
has
Customer
+ name : string
+ withdraw(atm:ATM)
+ giveCard() : Card
+ givePin() : string
+ giveAmount() : int
atm.process(this)
Bank
+ bankCode : string
owns
Card
+ cardNo : string
+ bankCode : string
- pinCode : string
uses
elkéri a kártyát
bekéri a pinkódot
ellenőrzi a pinkódot
bekéri az összeget
ellenőrzi az egyenleget
lebonyolítja a tranzakciót

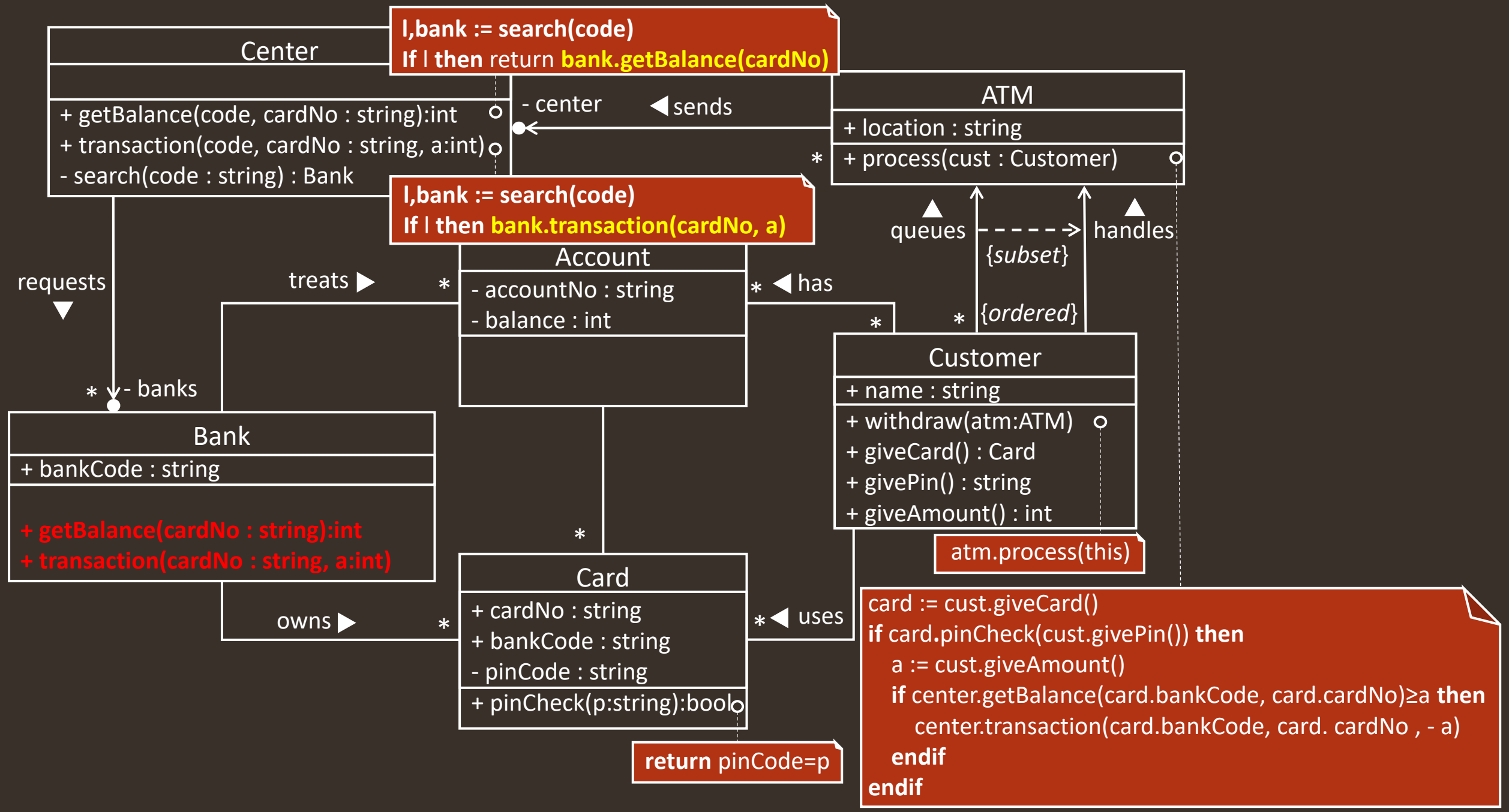

Center
ATM
+ location : string
+ process(cust : Customer)
sends
queues
{subset}
handles
{ordered}
requests
treats
Account
- accountNo : string
- balance : int
has
Customer
+ name : string
+ withdraw(atm:ATM)
+ giveCard() : Card
+ givePin() : string
+ giveAmount() : int
atm.process(this)
Bank
+ bankCode : string
owns
Card
+ cardNo : string
+ bankCode : string
- pinCode : string
+ pinCheck(p:string):bool
uses
elkéri a kártyát
bekéri a pinkódot
ellenőrzi a pinkódot
bekéri az összeget
ellenőrzi az egyenleget
lebonyolítja a tranzakciót

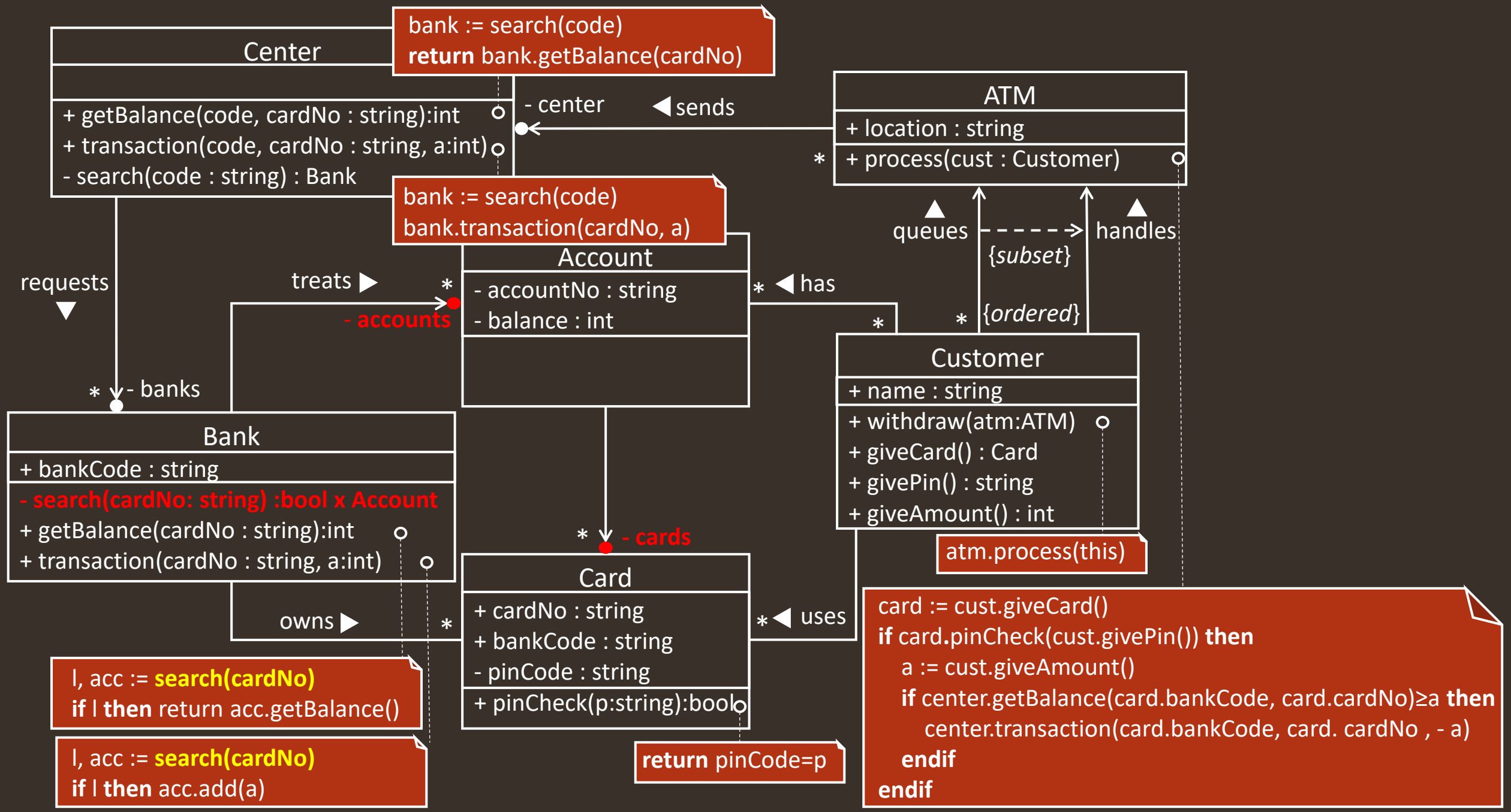

Center
sends
ATM
+ location : string
+ process(cust : Customer)
queues
{subset}
handles
{ordered}
requests
treats
Account
- accountNo : string
- balance : int
has
Customer
+ name : string
+ withdraw(atm:ATM)
+ giveCard() : Card
+ givePin() : string
+ giveAmount() : int
Bank
+ bankCode : string
owns
Card
+ cardNo : string
+ bankCode : string
- pinCode : string
+ pinCheck(p:string):bool
uses
atm.process(this
return pinCode=p
card := cust.giveCard()
if card.pinCheck(cust.givePin()) then
a := cust.giveAmount()
ellenőrzi az egyenleget
lebonyolítja a tranzakciót
endif

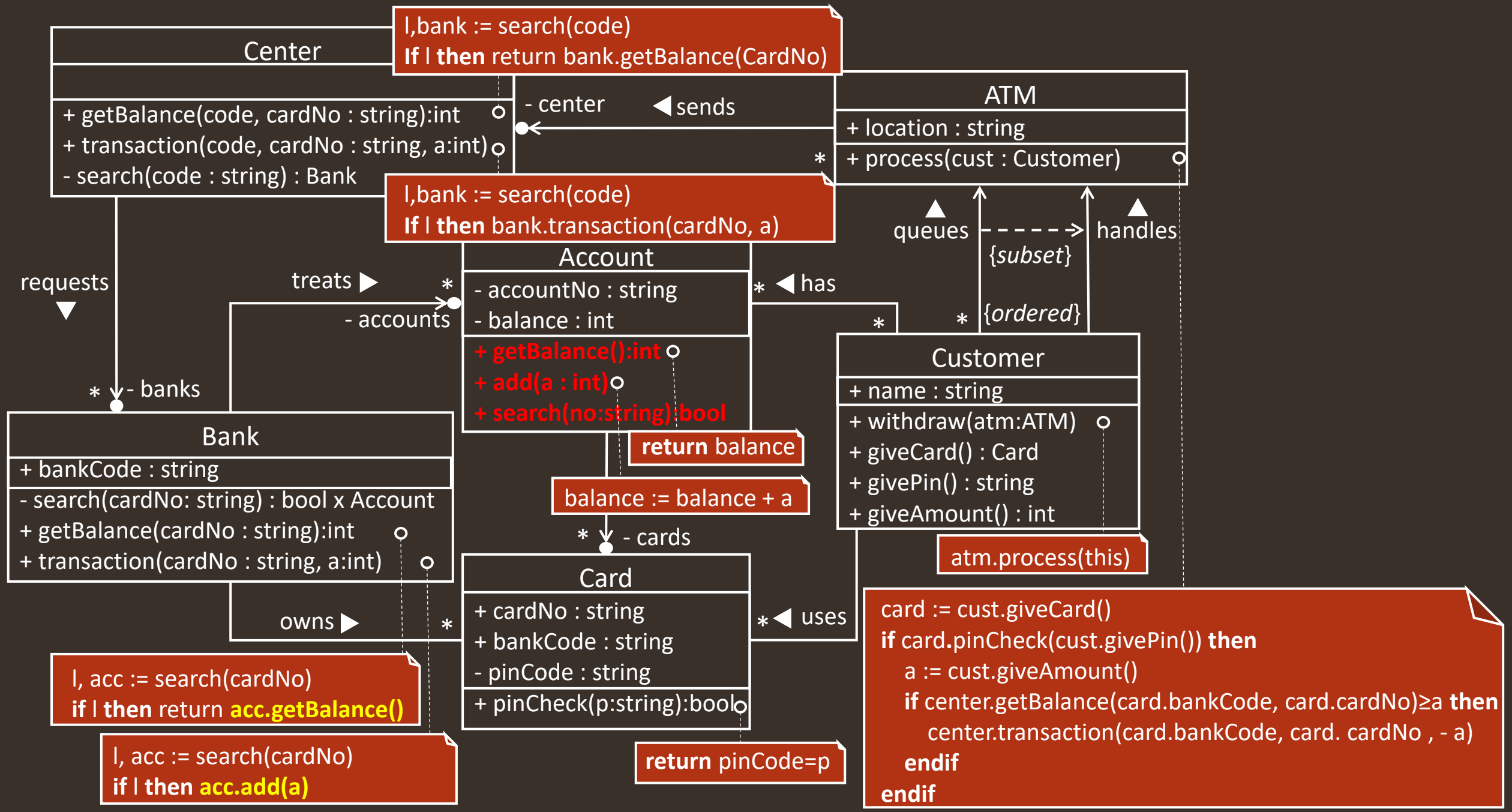

Center
+ getBalance(code, cardNo : string):int
+ transaction(code, cardNo : string, a:int)
sends
ATM
+ location : string
+ process(cust : Customer)
queues
{subset}
handles
{ordered}
requests
treats
Account
- accountNo : string
- balance : int
has
Customer
+ name : string
+ withdraw(atm:ATM)
+ giveCard() : Card
+ givePin() : string
+ giveAmount() : int
Bank
+ bankCode : string
atm.process(this
owns
Card
+ cardNo : string
+ bankCode : string
- pinCode : string
+ pinCheck(p:string):bool
uses
return pinCode=p
card := cust.giveCard()
if card.pinCheck(cust.givePin()) then
a := cust.giveAmount()
ellenőrzi az egyenleget
lebonyolítja a tranzakciót
endif

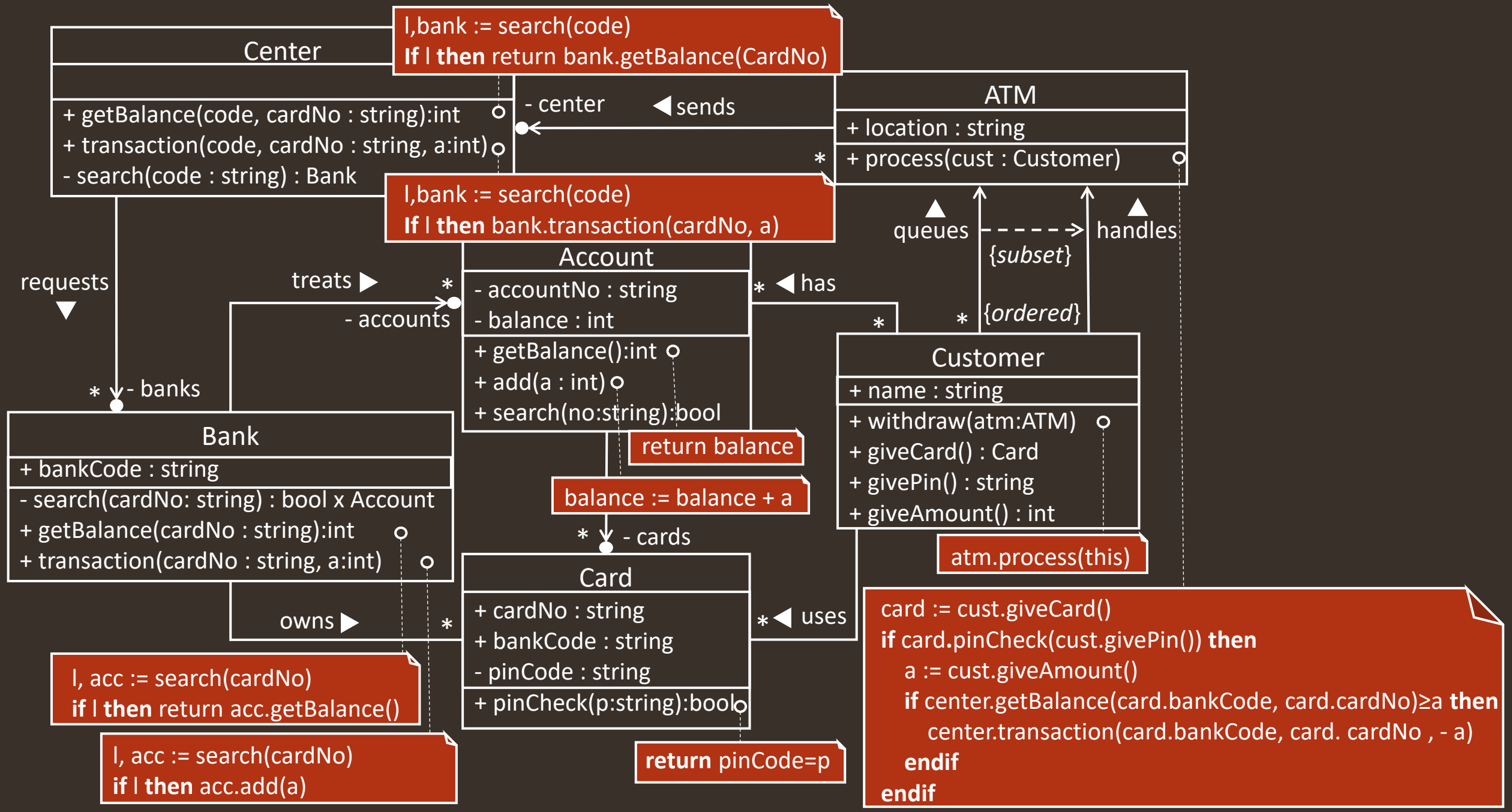

Center
+ getBalance(code, cardNo : string):int
+ transaction(code, cardNo : string, a:int)
- center
sends
ATM
+ location : string
+ process(cust : Customer)
*
queues
{subset}
handles
{ordered}
requests
treats
Account
- accountNo : string
- balance : int
has
Customer
+ name : string
+ withdraw(atm:ATM)
+ giveCard() : Card
+ givePin() : string
+ giveAmount() : int
Bank
+ bankCode : string
owns
Card
+ cardNo : string
+ bankCode : string
- pinCode : string
+ pinCheck(p:string):bool
uses
atm.process(this)
return pinCode=p
card := cust.giveCard()
if card.pinCheck(cust.givePin()) then
a := cust.giveAmount()
if center.getBalance(card.bankCode, card.cardNo)≥a
then
center.transaction(card.bankCode, card. cardNo , - a)
endif
endif


Center
+ getBalance(code, cardNo : string):int
+ transaction(code, cardNo : string, a:int)
- search(code : string) : Bank
l,bank := search(code)
If l then return bank.getBalance(cardNo)
- center
sends
ATM
+ location : string
+ process(cust : Customer)
l,bank := search(code)
If l then bank.transaction(cardNo, a)
queues
{subset}
handles
{ordered}
requests
treats
Account
- accountNo : string
- balance : int
has
- banks
Bank
+ bankCode : string
Customer
+ name : string
+ withdraw(atm:ATM)
+ giveCard() : Card
+ givePin() : string
+ giveAmount() : int
atm.process(this)
Card
+ cardNo : string
+ bankCode : string
- pinCode : string
+ pinCheck(p:string):bool
owns
uses
return pinCode=p
card := cust.giveCard()
if card.pinCheck(cust.givePin()) then
a := cust.giveAmount()
if center.getBalance(card.bankCode, card.cardNo)≥a then
center.transaction(card.bankCode, card. cardNo , - a)
endif
endif

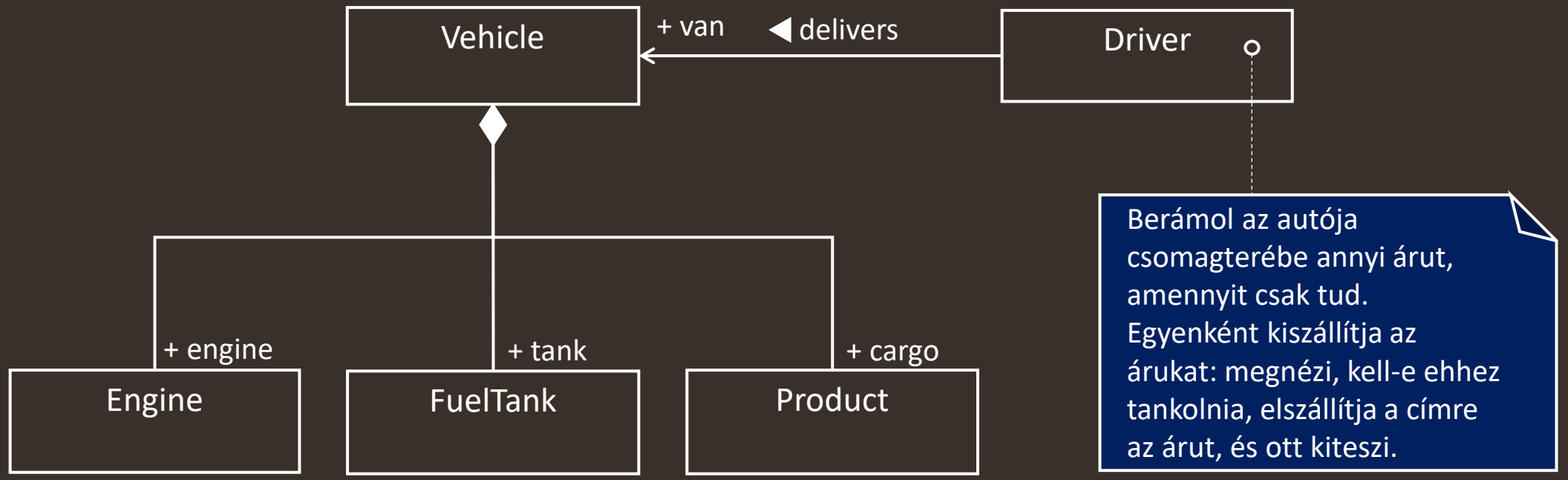

Center
+ getBalance(code, cardNo : string):int
+ transaction(code, cardNo : string, a:int)
- search(code : string) : Bank
I,bank := search(code)
If I then return bank.getBalance(cardNo)
I,bank := search(code)
If I then bank.transaction(cardNo, a)
- center
sends
ATM
+ location : string
+ process(cust : Customer)
queues
{subset}
handles
{ordered}
requests
treats
Account
- accountNo : string
- balance : int
has
- banks
Bank
+ bankCode : string
+ getBalance(cardNo : string):int
+ transaction(cardNo : string, a:int)
Customer
+ name : string
+ withdraw(atm:ATM)
+ giveCard() : Card
+ givePin() : string
+ giveAmount() : int
atm.process(this)
owns
Card
+ cardNo : string
+ bankCode : string
- pinCode : string
+ pinCheck(p:string):bool
uses
return pinCode=p
card := cust.giveCard()
if card.pinCheck(cust.givePin()) then
a := cust.giveAmount()
if center.getBalance(card.bankCode, card.cardNo)≥a then
center.transaction(card.bankCode, card. cardNo , - a)
endif
endif

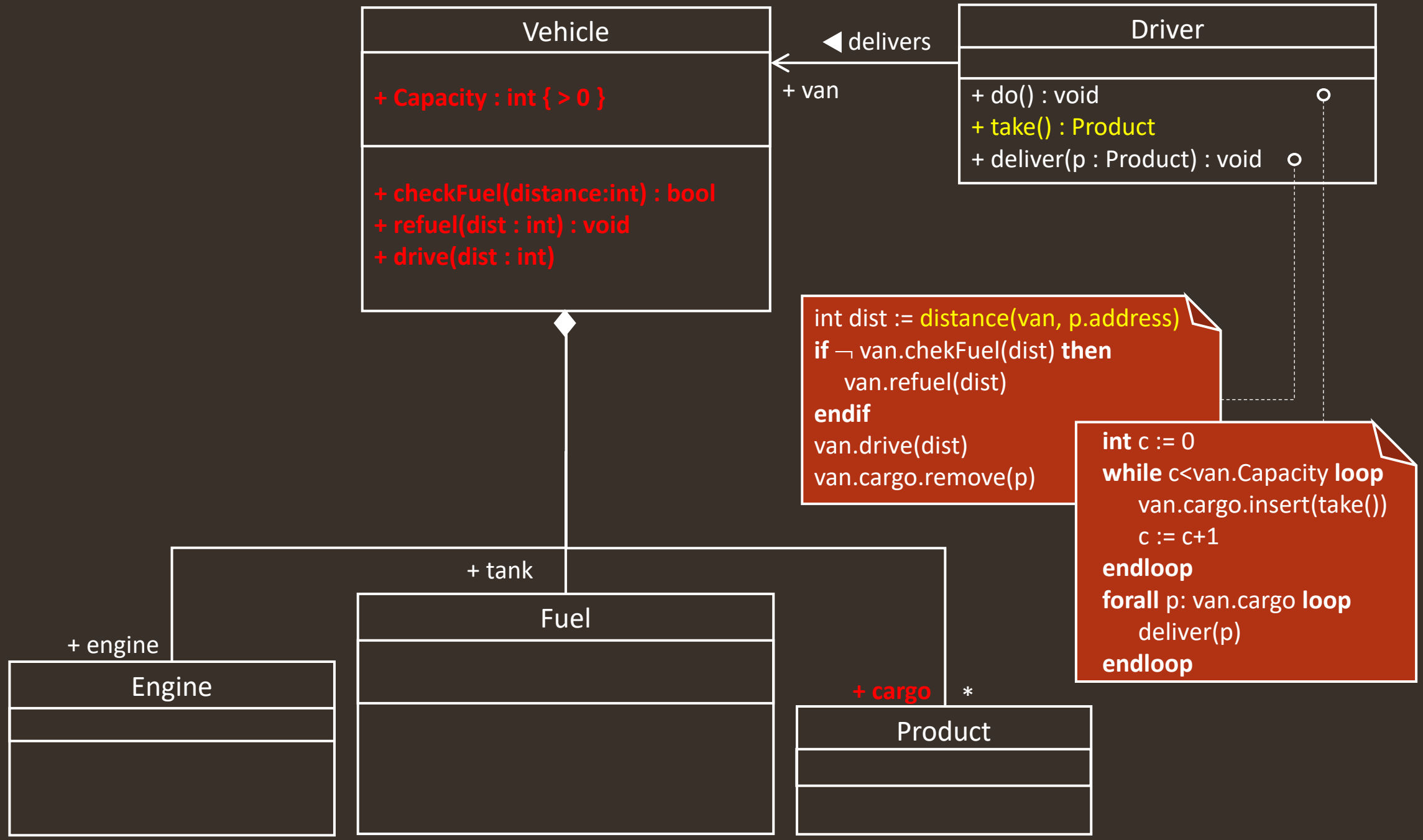

Center
+ getBalance(code, cardNo : string):int
+ transaction(code, cardNo : string, a:int)
- search(code : string) : Bank
bank := search(code)
return bank.getBalance(cardNo)
bank := search(code)
bank.transaction(cardNo, a)
- center
sends
ATM
+ location : string
+ process(cust : Customer)
queues
{subset}
handles
{ordered}
requests
treats
- accounts
Account
- accountNo : string
- balance : int
has
- banks
Bank
+ bankCode : string
- search(cardNo: string) :bool x Account
+ getBalance(cardNo : string):int
+ transaction(cardNo : string, a:int)
- cards
Card
+ cardNo : string
+ bankCode : string
- pinCode : string
+ pinCheck(p:string):bool
owns
uses
Customer
+ name : string
+ withdraw(atm:ATM)
+ giveCard() : Card
+ givePin() : string
+ giveAmount() : int
atm.process(this)
card := cust.giveCard()
if card.pinCheck(cust.givePin()) then
a := cust.giveAmount()
if center.getBalance(card.bankCode, card.cardNo)≥a then
center.transaction(card.bankCode, card. cardNo , - a)
endif
endif
return pinCode=p
I, acc := search(cardNo)
if I then return acc.getBalance()
I, acc := search(cardNo)
if I then acc.add(a)

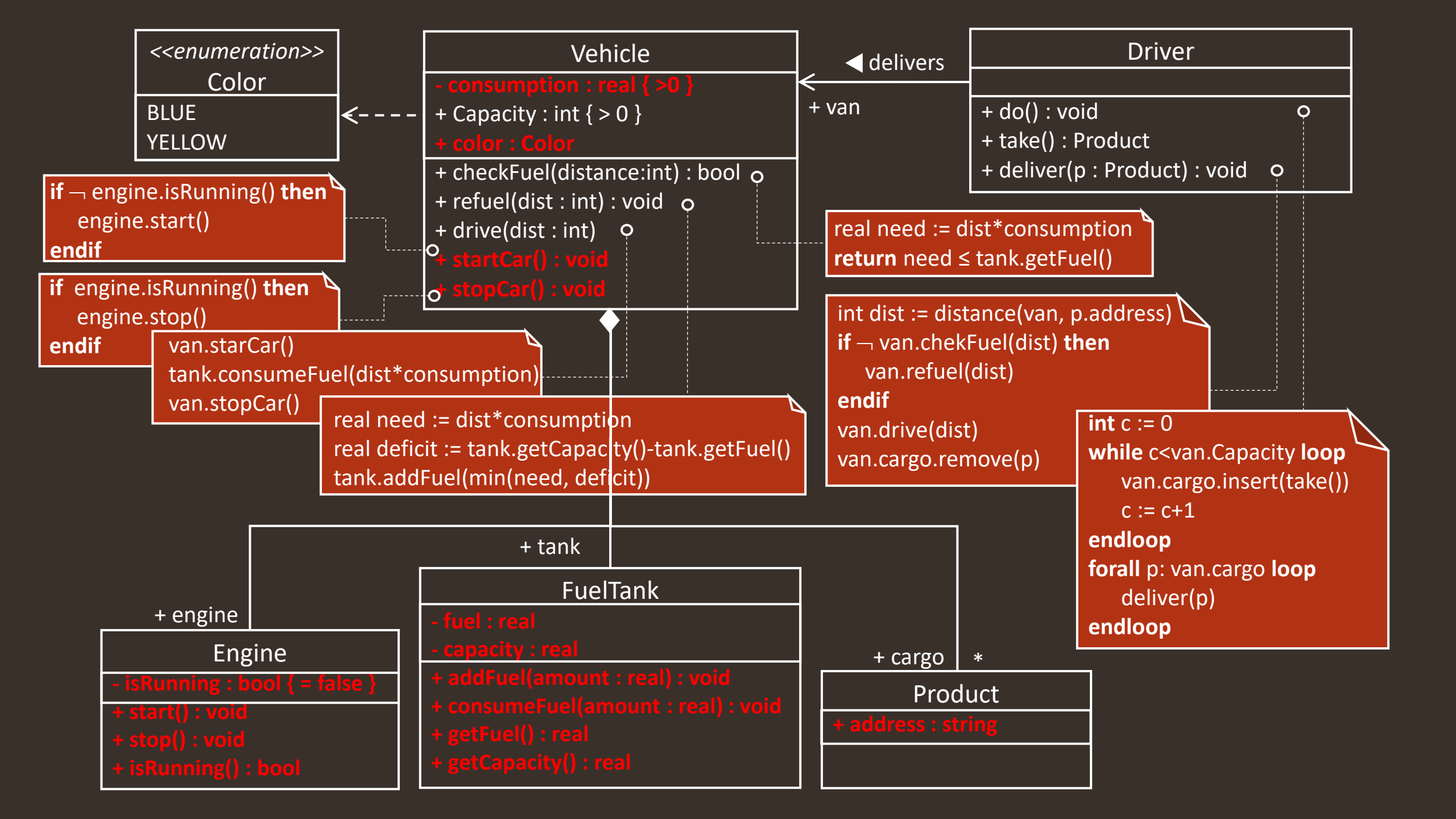

Center
+ getBalance(code, cardNo : string):int
+ transaction(code, cardNo : string, a:int)
- search(code : string) : Bank
l,bank := search(code)
If l then return bank.getBalance(CardNo)
l,bank := search(code)
If l then bank.transaction(cardNo, a)
- center
sends
ATM
+ location : string
+ process(cust : Customer)
queues
{subset}
{ordered}
handles
requests
treats
- accounts
- banks
Account
- accountNo : string
- balance : int
+ getBalance():int
+ add(a : int)
+ search(no:string):bool
has
return balance
balance := balance + a
Customer
+ name : string
+ withdraw(atm:ATM)
+ giveCard() : Card
+ givePin() : string
+ giveAmount() : int
atm.process(this)
Bank
+ bankCode : string
- search(cardNo: string) : bool x Account
+ getBalance(cardNo : string):int
+ transaction(cardNo : string, a:int)
owns
- cards
Card
+ cardNo : string
+ bankCode : string
- pinCode : string
+ pinCheck(p:string):bool
uses
return pinCode=p
l, acc := search(cardNo)
if l then return acc.getBalance()
l, acc := search(cardNo)
if l then acc.add(a)
card := cust.giveCard()
if card.pinCheck(cust.givePin()) then
a := cust.giveAmount()
if center.getBalance(card.bankCode, card.cardNo)≥a then
center.transaction(card.bankCode, card. cardNo , - a)
endif
endif

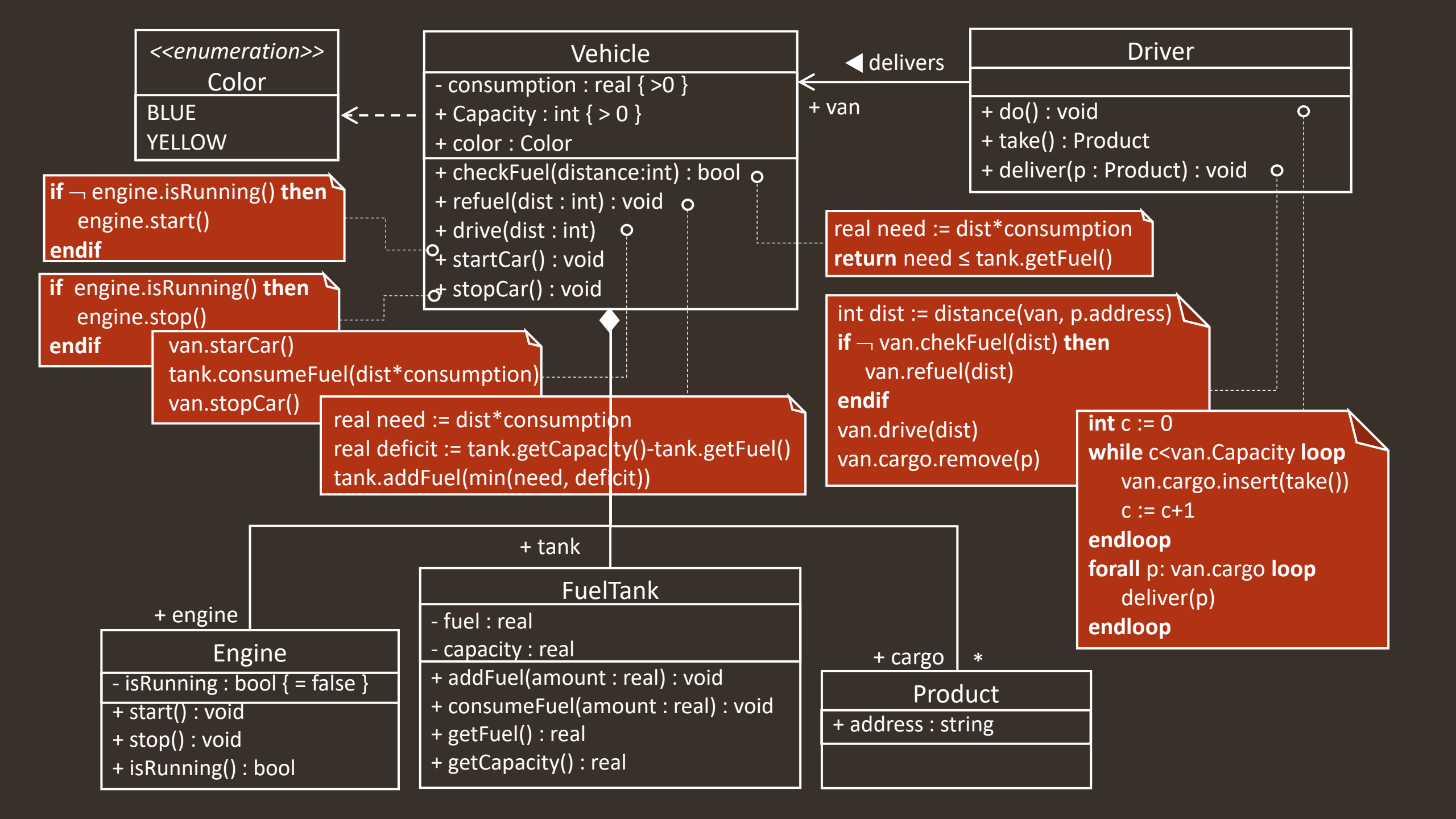

Center
+ getBalance(code, cardNo : string):int
+ transaction(code, cardNo : string, a:int)
- search(code : string) : Bank
l,bank := search(code)
If l then return bank.getBalance(CardNo)
l,bank := search(code)
If l then bank.transaction(cardNo, a)
- center
sends
ATM
+ location : string
+ process(cust : Customer)
queues
{subset}
{ordered}
handles
requests
treats
- accounts
- banks
Account
- accountNo : string
- balance : int
+ getBalance():int
+ add(a : int)
+ search(no:string):bool
has
return balance
balance := balance + a
Customer
+ name : string
+ withdraw(atm:ATM)
+ giveCard() : Card
+ givePin() : string
+ giveAmount() : int
atm.process(this)
Bank
+ bankCode : string
- search(cardNo: string) : bool x Account
+ getBalance(cardNo : string):int
+ transaction(cardNo : string, a:int)
owns
- cards
Card
+ cardNo : string
+ bankCode : string
- pinCode : string
+ pinCheck(p:string):bool
uses
l, acc := search(cardNo)
if l then return acc.getBalance()
l, acc := search(cardNo)
if l then acc.add(a)
return pinCode=p
card := cust.giveCard()
if card.pinCheck(cust.givePin()) then
a := cust.giveAmount()
if center.getBalance(card.bankCode, card.cardNo)≥a then
center.transaction(card.bankCode, card. cardNo , - a)
endif
endif

3. Egy csomag kiszállító futár a termékek kiszállítása során bepakol a járművébe annyi terméket, amely belefér (a jármű kapacitását figyelembe véve). A termékekre ráírták a kiszállítás címét, amelyből kiszámolható, hogy a cím milyen távolságra van a futár aktuális tartózkodási helyétől (km-ben). A szállítás címei minden esetben benzinkutak (az ott lévő PickPack pontok), így minden állomáson lehet tankolni is. A futárnak a járműbe bepakolt összes terméket ki kell szállítani, aminek menete a következő: először ellenőrzi, hogy a benzinszint a következő kiszállításhoz elegendő-e. Amennyiben nem, előbb tankol; egyébként elindítja a motort, elmegy a címzetthez (aminek eredményeképp csökken a benzinszint a fogyasztás szerint), leállítja a motort, majd kipakolja a terméket. Egy futár járműve kétféle színű lehet: kék és sárga. A járműnek van motorja, rakodótere és egy benzintartálya. A motort el lehet indítani és le lehet állítani. A tartályba a maximális benzinszint figyelembe vételével tankolhatunk. A járműnek ismert a benzinfogyasztása, ami liter/km mértékegységben van adva.




Vehicle
+ Capacity : int { > 0 }
+ checkFuel(distance:int) : bool
+ refuel(dist : int) : void
+ drive(dist : int)
delivers
+ van
Driver
+ do() : void
+ take() : Product
+ deliver(p : Product) : void
int dist := distance(van, p.address)
if ¬ van.chekFuel(dist) then
van.refuel(dist)
endif
van.drive(dist)
van.cargo.remove(p)
int c := 0
while c<van.Capacity loop
van.cargo.insert(take())
c := c+1
endloop
forall p: van.cargo loop
deliver(p)
endloop
+ tank
Fuel
+ engine
Engine
+ cargo
*
Product


<<enumeration>>
Color
BLUE
YELLOW
Vehicle
- consumption : real { >0 }
+ Capacity : int { > 0 }
+ color : Color
+ checkFuel(distance:int) : bool
+ refuel(dist : int) : void
+ drive(dist : int)
+ startCar() : void
+ stopCar() : void
delivers
+ van
Driver
+ do() : void
+ take() : Product
+ deliver(p : Product) : void
if ¬ engine.isRunning() then
engine.start()
endif
if engine.isRunning() then
engine.stop()
endif
van.starCar()
tank.consumeFuel(dist*consumption)
van.stopCar()
real need := dist*consumption
return need ≤ tank.getFuel()
real need := dist*consumption
real deficit := tank.getCapacity()-tank.getFuel()
tank.addFuel(min(need, deficit))
int dist := distance(van, p.address)
if ¬ van.chekFuel(dist) then
van.refuel(dist)
endif
van.drive(dist)
van.cargo.remove(p)
int c := 0
while c<van.Capacity loop
van.cargo.insert(take())
c := c+1
endloop
forall p: van.cargo loop
deliver(p)
endloop
+ tank
+ engine
+ cargo
*
Engine
- isRunning : bool { = false }
+ start() : void
+ stop() : void
+ isRunning() : bool
FuelTank
- fuel : real
- capacity : real
+ addFuel(amount : real) : void
+ consumeFuel(amount : real) : void
+ getFuel() : real
+ getCapacity() : real
Product
+ address : string


<<enumeration>>
Color
BLUE
YELLOW
Vehicle
- consumption : real { >0 }
+ Capacity : int { > 0 }
+ color : Color
+ checkFuel(distance:int) : bool
+ refuel(dist : int) : void
+ drive(dist : int)
+ startCar() : void
+ stopCar() : void
delivers
+ van
Driver
+ do() : void
+ take() : Product
+ deliver(p : Product) : void
if ¬ engine.isRunning() then
engine.start()
endif
if engine.isRunning() then
engine.stop()
endif
van.starCar()
tank.consumeFuel(dist*consumption)
van.stopCar()
real need := dist*consumption
return need ≤ tank.getFuel()
real need := dist*consumption
real deficit := tank.getCapacity()-tank.getFuel()
tank.addFuel(min(need, deficit))
int dist := distance(van, p.address)
if ¬ van.chekFuel(dist) then
van.refuel(dist)
endif
van.drive(dist)
van.cargo.remove(p)
int c := 0
while c<van.Capacity loop
van.cargo.insert(take())
c := c+1
endloop
forall p: van.cargo loop
deliver(p)
endloop
+ tank
+ engine
Engine
- isRunning : bool { = false }
+ start() : void
+ stop() : void
+ isRunning() : bool
FuelTank
- fuel : real
- capacity : real
+ addFuel(amount : real) : void
+ consumeFuel(amount : real) : void
+ getFuel() : real
+ getCapacity() : real
+ cargo
*
Product
+ address : string