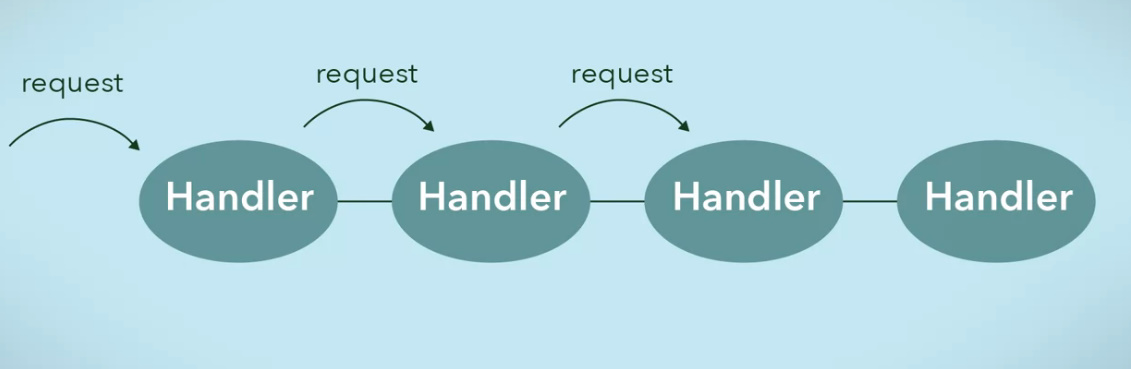
* S-ar traduce ca lant de responsabilitate
* In lumea reala, de ex, cand avem o problema de sanatate ne ducem intai la medicul de familie. Daca el nu ne poate ajuta, ne redirectioneaza la un alt medic, ca el sa ne rezolve problema. Acest medic tot ne poate trimite la altul, si pana la urma, problema va fi rezolvata de un medic, dar rezolvarea problemei a fost trimisa de un medic la altul. Nu ne pasa cine rezolva problema, dar cineva sa o faca doar.
* Asta si este cam chain of responsability.
* **Chain of responsability** – se bazeaza pe a avea o colectie de obiecte, conectate impreuna, ce sunt responsabile de a raspunde la un requests

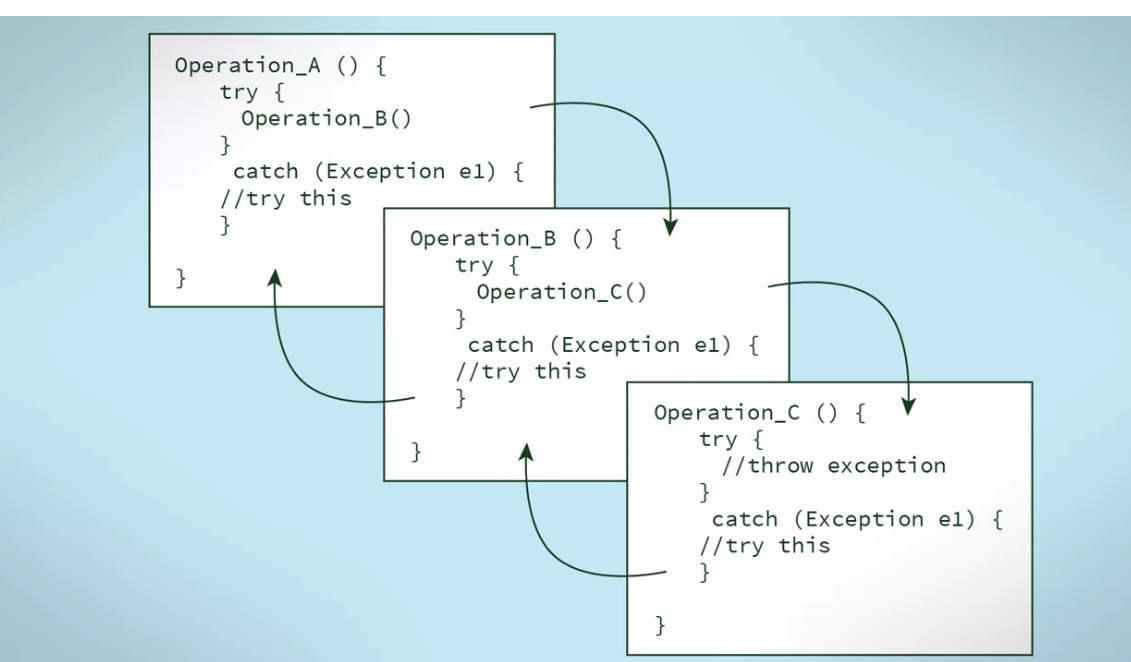
**Fiecare obiect trebuie sa aiba referinta catre urmatorul daca el nu poate sa se ocupe de request**



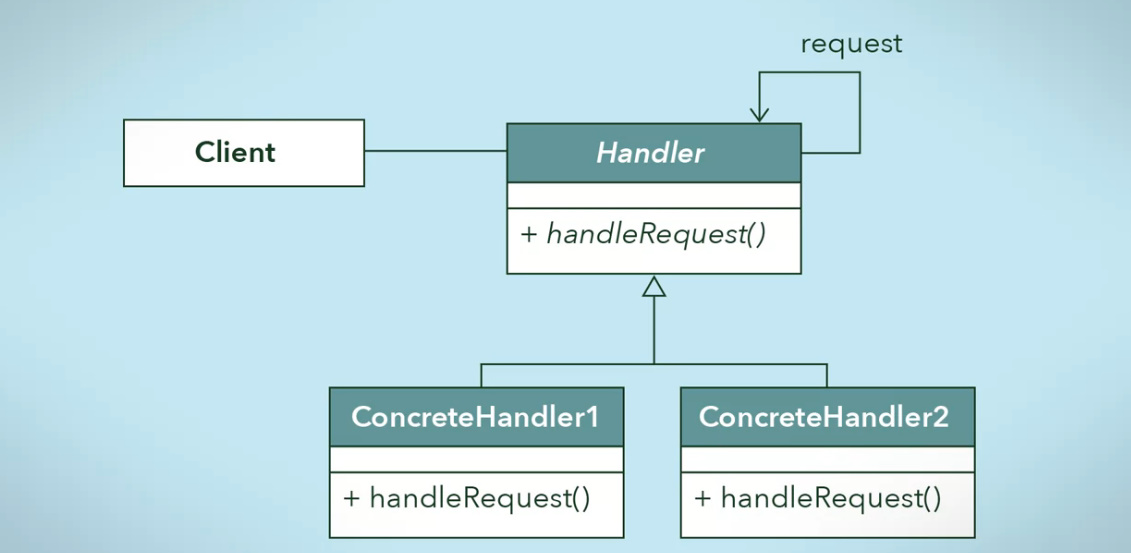
Avem aici o colectie de obiecte handles.

Trimitem un request si ele ajunge la primul. Daca el nu poate raspunde la request, il trimite la al 2, si al 2 la al 3 si tot asa.

* Fiecare obiect incearca sa raspunda la request pana reuseste vreunul.
* Cam asa si functioneaza exception handling in Java



**UML**

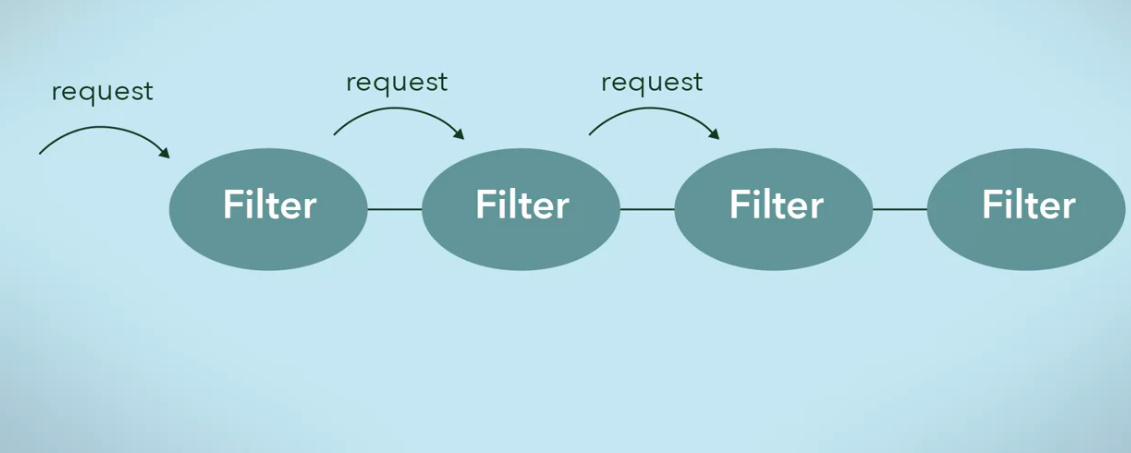
****

Handler este o clasa abstracta

Obiectele care sunt Handlers, adica implementeaza Handler, sunt conectate in lant.

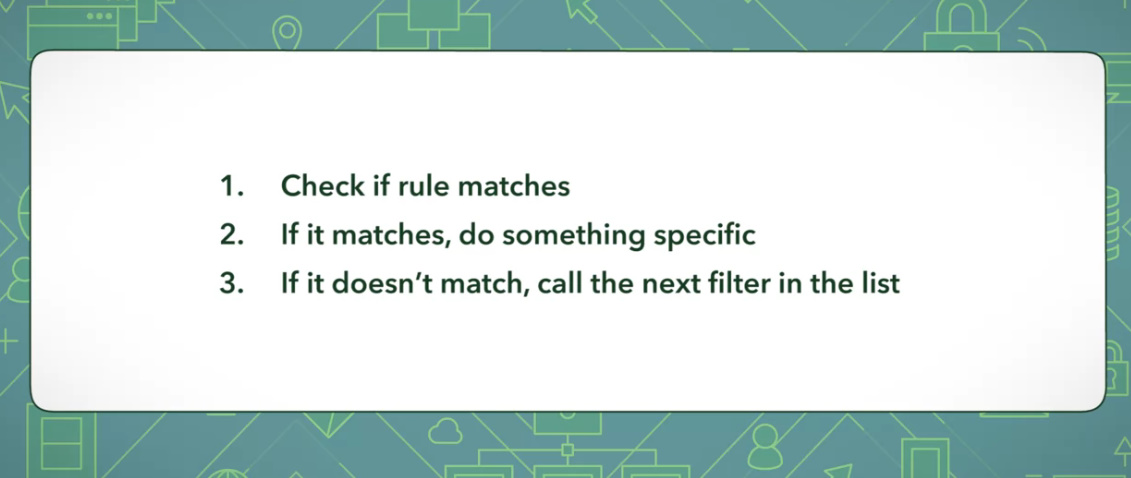
**Exemplu**

Avem un program ce primeste mesaje si le pune intr-o anumita categorie. Pentru asta avem mai multe obiecte filtre create, si mesajul trece prin fiecare pana unul din ele decide ca i se potriveste dupa cerinte



Totusi, aici poate aparea o problema. Daca gen filtrul al 2 nu trimite corect mesajul la urmatorul. Asa mesajul nu mai trece prin celelalte.

* De asta, trebuie definit un algoritm pentru comun pentru handlers

Pentru asta, putem folosi Template Method

* Fie o masina care ne da banii din cont. Ea va decide in ce bancnote sa ne dea banii, de 10,20 sau 50. Suma introdusa trebuie sa fie la urma cu un 0, gen 50,100

public class Currency {

private int amount;

public Currency(int amt){

this.amount=amt;

}

public int getAmount(){

return this.amount;

}

}

Aici e suma introdusa de client

public interface DispenseChain {

void setNextChain(DispenseChain nextChain);

void dispense(Currency cur);

}

Asta e interfata pentru a crea fiecare handler

Acum cream primul Handler pentru bancnote de 50.El are referinta catre urmatorul handler, de 20 logic

public class Dollar50Dispenser implements DispenseChain {

private DispenseChain chain;

@Override

public void setNextChain(DispenseChain nextChain) {

this.chain=nextChain;

}

@Override

public void dispense(Currency cur) {

if(cur.getAmount() >= 50){

int num = cur.getAmount()/50;

int remainder = cur.getAmount() % 50;

System.out.println("Dispensing "+num+" 50$ note");

if(remainder !=0) this.chain.dispense(new Currency(remainder));

}else{

this.chain.dispense(cur);

}

}

}

Daca el nu a reusit sa se ocupe complet de request, apeleaza pe Handler de 20$.

public class Dollar20Dispenser implements DispenseChain{

private DispenseChain chain;

@Override

public void setNextChain(DispenseChain nextChain) {

this.chain=nextChain;

}

@Override

public void dispense(Currency cur) {

if(cur.getAmount() >= 20){

int num = cur.getAmount()/20;

int remainder = cur.getAmount() % 20;

System.out.println("Dispensing "+num+" 20$ note");

if(remainder !=0) this.chain.dispense(new Currency(remainder));

}else{

this.chain.dispense(cur);

}

}

}

public class Dollar10Dispenser implements DispenseChain {

private DispenseChain chain;

@Override

public void setNextChain(DispenseChain nextChain) {

this.chain=nextChain;

}

@Override

public void dispense(Currency cur) {

if(cur.getAmount() >= 10){

int num = cur.getAmount()/10;

int remainder = cur.getAmount() % 10;

System.out.println("Dispensing "+num+" 10$ note");

if(remainder !=0) this.chain.dispense(new Currency(remainder));

}else{

this.chain.dispense(cur);

}

}

}

ublic class ATMDispenseChain {

private DispenseChain c1;

public ATMDispenseChain() {

// initialize the chain

this.c1 = new Dollar50Dispenser();

DispenseChain c2 = new Dollar20Dispenser();

DispenseChain c3 = new Dollar10Dispenser();

// set the chain of responsibility

c1.setNextChain(c2);

c2.setNextChain(c3);

}

public static void main(String[] args) {

ATMDispenseChain atmDispenser = new ATMDispenseChain();

while (true) {

int amount = 0;

System.out.println("Enter amount to dispense");

Scanner input = new Scanner(System.in);

amount = input.nextInt();

if (amount % 10 != 0) {

System.out.println("Amount should be in multiple of 10s.");

return;

}

// process the request

atmDispenser.c1.dispense(new Currency(amount));

}

}

}