Chain of responsibility pattern is used to achieve loose coupling in software design where a request from client is passed to a chain of objects to process them. Then the object in the chain will decide themselves who will be processing the request and whether the request is required to be sent to the next object in the chain or not.

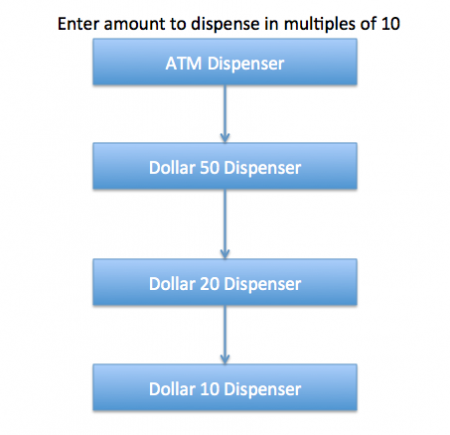
### **Chain of Responsibility Pattern Example in JDK**

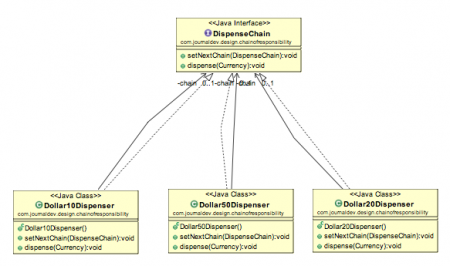
Let’s see the example of chain of responsibility pattern in JDK and then we will proceed to implement a real life example of this pattern. We know that we can have multiple catch blocks in a [try-catch block](https://www.journaldev.com/592/java-try-with-resources) code. Here every catch block is kind of a processor to process that particular exception.

So when any exception occurs in the try block, its send to the first catch block to process. If the catch block is not able to process it, it forwards the request to next object in chain i.e next catch block. If even the last catch block is not able to process it, the exception is thrown outside of the chain to the calling program.

### **Chain of Responsibility Design Pattern Example**

One of the great example of Chain of Responsibility pattern is **ATM Dispense machine**. The user enters the amount to be dispensed and the machine dispense amount in terms of defined currency bills such as 50$, 20$, 10$ etc.





Example program

/\*

\* source :

\* https://www.journaldev.com/1617/chain-of-responsibility-design-pattern-in-java

\*/

package chain\_of\_responsibility;

import java.util.Scanner;

/\*

\* we can create a class Currency that will store the amount to dispense and used by the chain implementations.

\*/

class Currency{

private int amount;

public Currency(int amount) {

this.amount = amount;

}

public int getAmount() {

return this.amount;

}

}

/\*

\* the base interface should have a method to define the next processor in the chain and the method that will process the

\* request. our ATM Dispense interface will look like below.

\*/

interface DispenseChain{

void setNextChain(DispenseChain nextChain);

void dispense(Currency cur);

}

/\*

\* we need to create different processor classes that will implement the DispenseChain interface and provide

\* implementation of dispense methods. Since we are developing our system to work with three types of currency bills

\* - 50$ 20$ 10$, we will create three concrete implementations

\*/

class Dollar50Dispenser implements DispenseChain{

private DispenseChain chain;

@Override

public void setNextChain(DispenseChain nextChain) {

this.chain = nextChain;

}

@Override

public void dispense(Currency cur) {

//check if amount is greater than 50, then dispense 50, remainder will be pushed to next chain

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

int num = cur.getAmount()/50;//dispense this num

int remainder = cur.getAmount()%50;//if remainder is greater than zero push to next chain

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

if(remainder != 0) {

this.chain.dispense(new Currency(remainder));

}

}else {

this.chain.dispense(cur);

}

}

}

class Dollar20Dispenser implements DispenseChain{

private DispenseChain chain;

@Override

public void setNextChain(DispenseChain nextChain) {

this.chain = nextChain;

}

@Override

public void dispense(Currency currency) {

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

int num = currency.getAmount()/20;

int remainder = currency.getAmount()%20;

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

if(remainder != 0) {

this.chain.dispense(new Currency(remainder));

}

}else {

this.chain.dispense(currency);

}

}

}

class Dollar10Dispenser implements DispenseChain{

private DispenseChain chain;

@Override

public void setNextChain(DispenseChain nextChain) {

this.chain = nextChain;

}

@Override

public void dispense(Currency currency) {

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

int number = currency.getAmount()/10;

int remainder = currency.getAmount()%10;

System.out.println("dispensing "+number+" 10$ note");

System.out.println(remainder);

if(remainder != 0) {

this.chain.dispense(new Currency(remainder));

}

}else {

this.chain.dispense(currency);

}

}

}

public 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 sc = new Scanner(System.in);

amount = sc.nextInt();

if(amount % 10 != 0) {

System.out.println("amount should be in multiples of 10s");

return;

}

//process the request

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

}

}

}