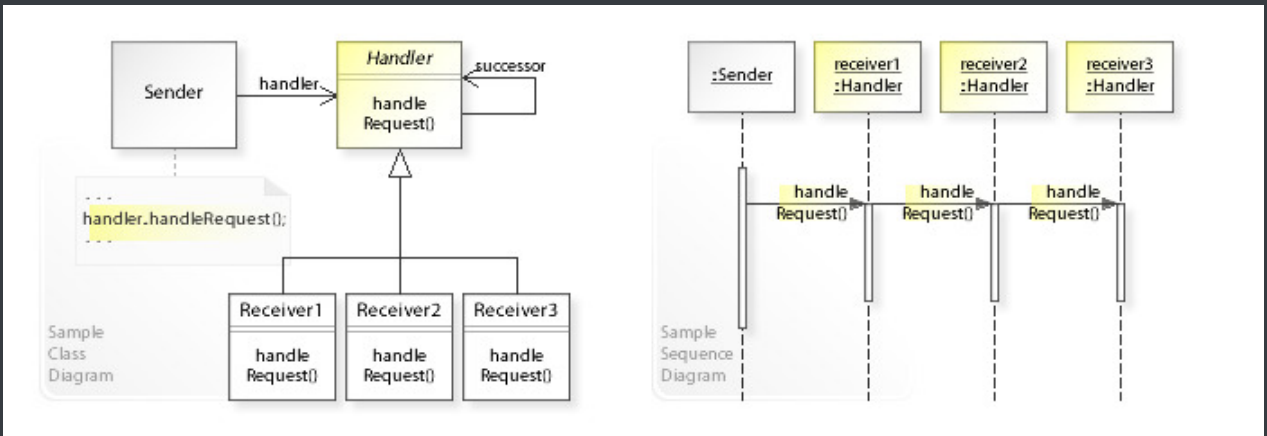
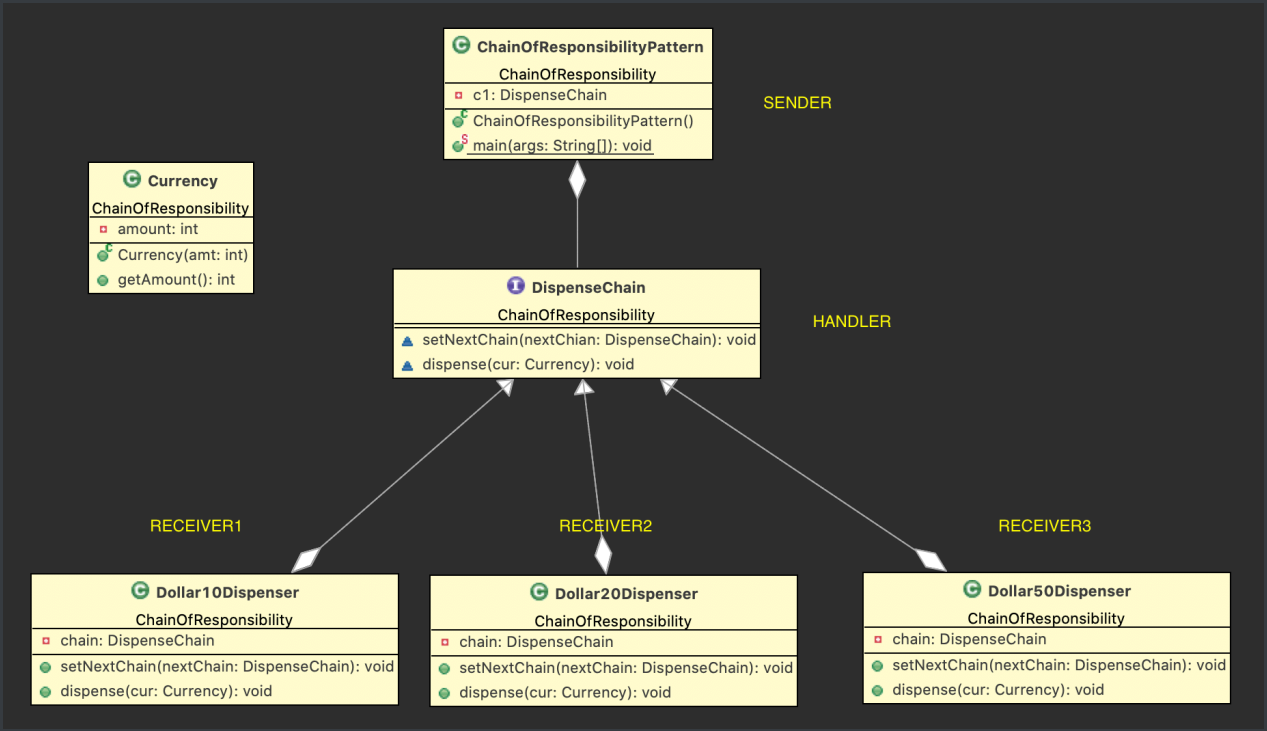
책임사슬이란?

> 어떤 요청이 그 요청을 담당하는 객체에 들어오면 각각의 요청에 대해서 특정한 객체가 담당하는 것이 일반적이지만 객체를 연결리스트와 같은 사슬 방식으로 연결한 후에 요청을 수행하지 못하는 객체라면 다음 객체에 넘기며 책임을 넘기는 형태의 패턴을 말한다. 이는 요청을 보내는 객체와 이를 처리하는 **객체간의 결합도를 느슨하게** 하기 위한 방법이며 **여러 객체에 처리 기회를 준다.**
>
> - java의 try catch 구문과 비슷한 원리이다. 예외를 잡지 못하면 나를 호출한 블럭으로 예외를 던지는 형태.



실습

Class Diagram



Currency Class

```
package ChainOfResponsibility;

public class Currency {

  private int amount;

  public Currency(int amt) {
    this.amount=amt;
  }

  public int getAmount() {
    return this.amount;
  }
}
```

Sender Class (ChainOfResponsibilityPattern)

```
package ChainOfResponsibility;

import java.util.Scanner;

public class ChainOfResponsibilityPattern {

  private DispenseChain c1;

  public ChainOfResponsibilityPattern() {
    this.c1 = new Dollar50Dispenser();
    DispenseChain c2 = new Dollar20Dispenser();
    DispenseChain c3 = new Dollar10Dispenser();

    c1.setNextChain(c2);
    c2.setNextChain(c3);
  }

  public static void main(String[] args) {
    ChainOfResponsibilityPattern atmDispenser = new
ChainOfResponsibilityPattern();
    Scanner input = null;

    while(true) {
      int amount = 0;
      System.out.println("Enter amount to dispense");
      input = new Scanner(System.in);
      amount = input.nextInt();
      if(amount % 10 != 0) {
        System.out.println("Amount should be multiple of ten! Input again.
");
        continue;
      }

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

Handler Class (DispenseChain)

```
package ChainOfResponsibility;

public interface DispenseChain {

  void setNextChain(DispenseChain nextChian);
  void dispense(Currency cur);
}
```

Reciever 1 Class (Dollar50Dispenser)

```
package ChainOfResponsibility;

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);
  }

}
```

Receiver Class 2 (Dollar20Dispenser)

```
package ChainOfResponsibility;

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);
  }
}
```

Receiver Class 3 (Dollar10Dispenser)

```
package ChainOfResponsibility;

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);
  }
}
```