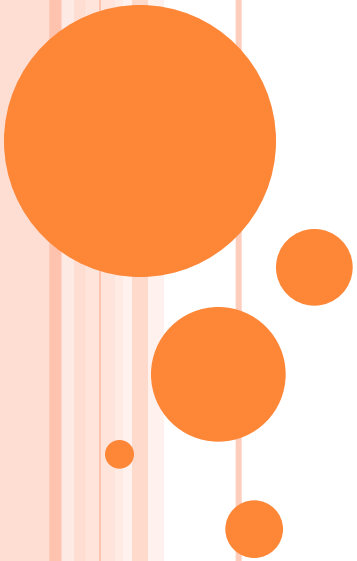


# Aspect-Oriented Development of an ATM Simulation

**Melihcan Türk**

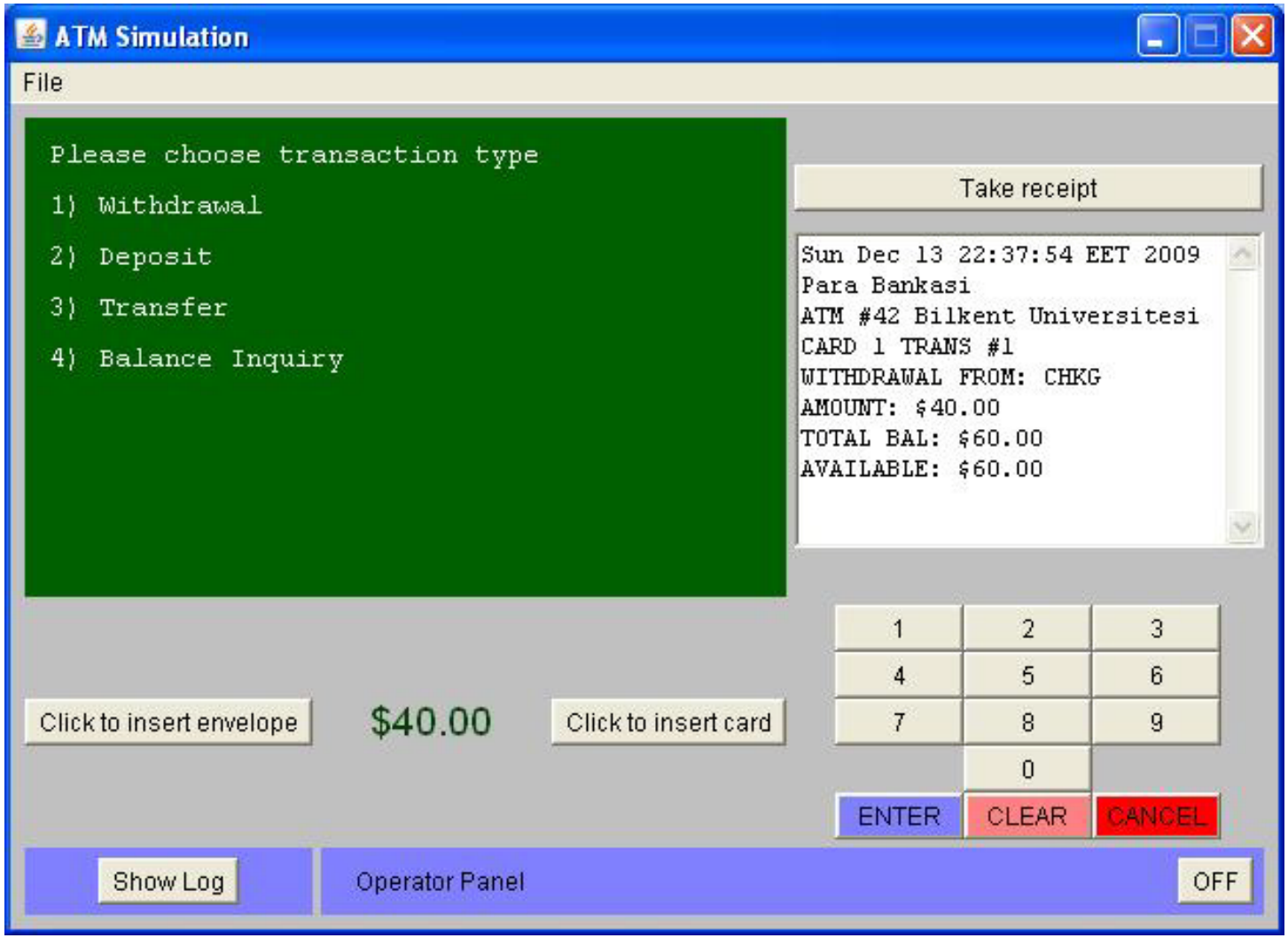
**Ahmet Çağrı Şimşek**

# Outline

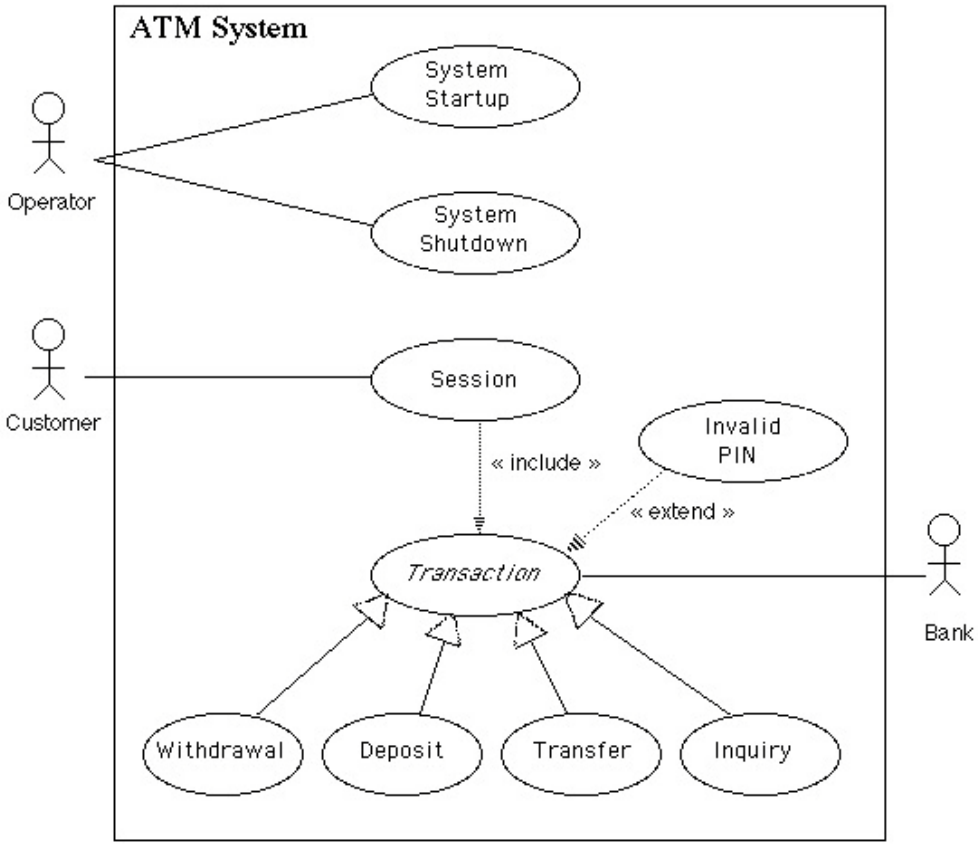
- Introduction
- Background
- OO Design of System Using Design Patterns
- Identifying Aspects
- Constraint Checking Aspect
- Security Handling Aspect
- Logging Aspect
- Aspect Oriented Programming
- Discussion
- Conclusion

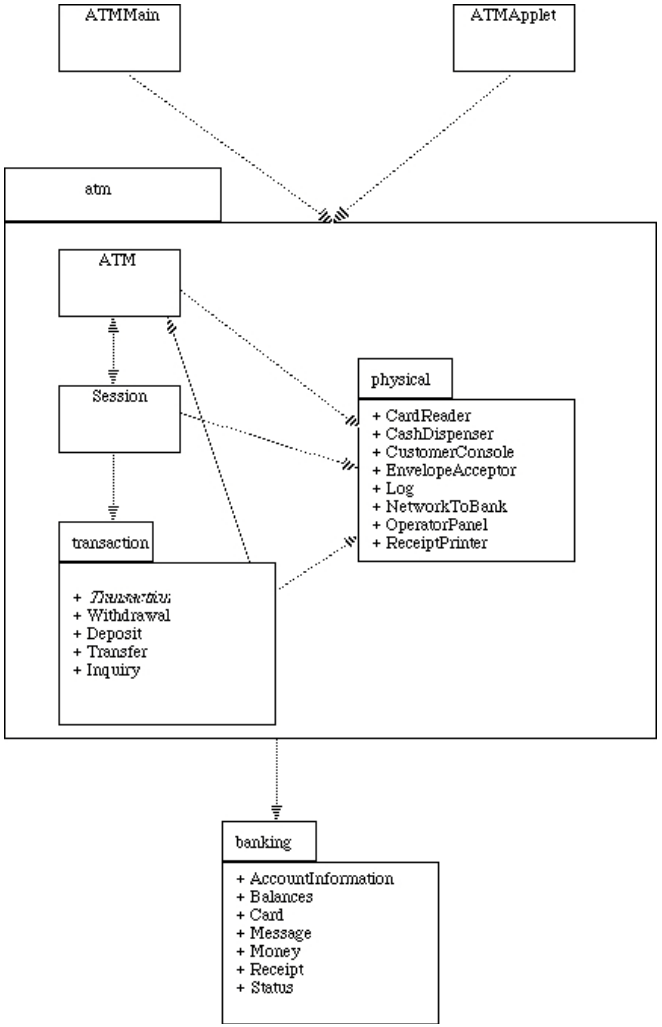
# Introduction

# Background

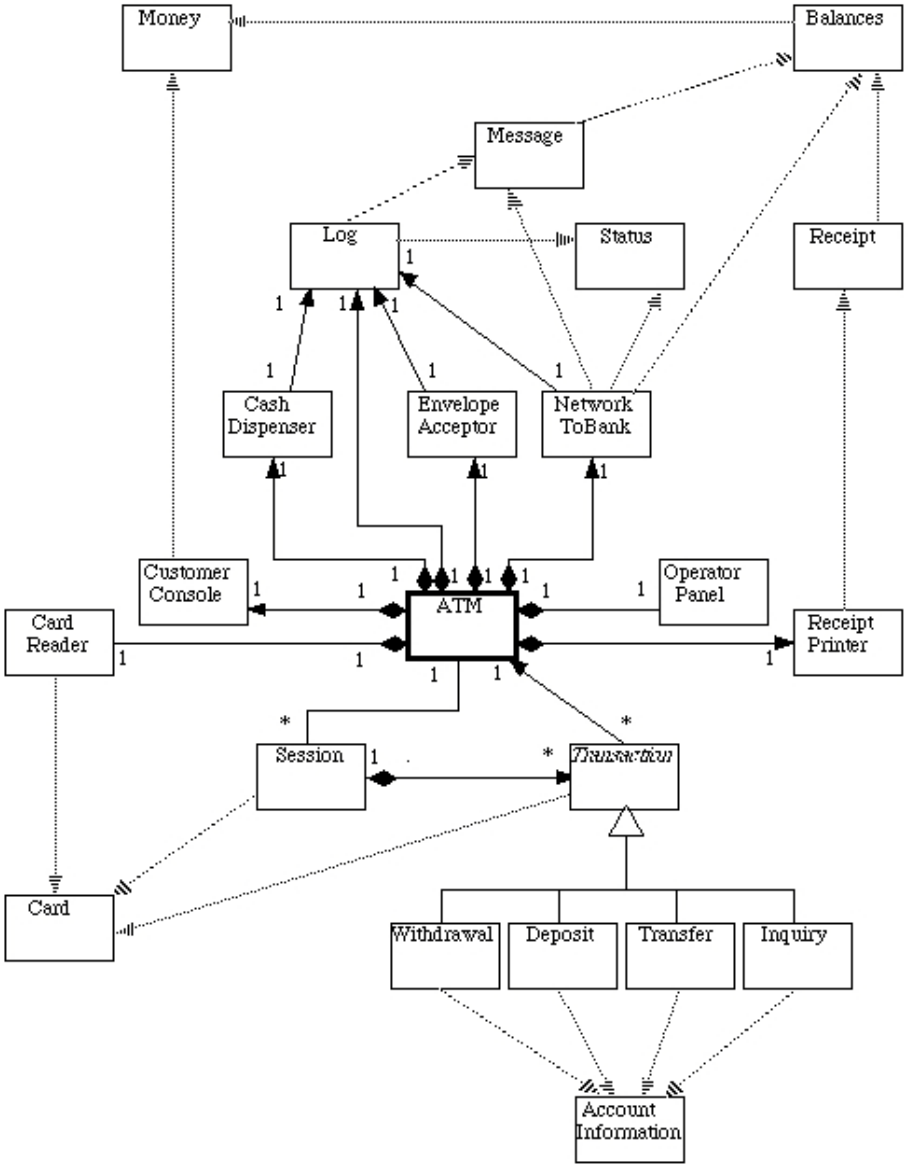

ATM Simulation
File
Please choose transaction type
1) Withdrawal
2) Deposit
3) Transfer
4) Balance Inquiry
Take receipt
Sun Dec 13 22:37:54 EET 2009
Para Bankasi
ATM #42 Bilkent Universitesi
CARD 1 TRANS #1
WITHDRAWAL FROM: CHKG
AMOUNT: $40.00
TOTAL BAL: $60.00
AVAILABLE: $60.00
1
2
3
4
5
6
7
8
9
0
ENTER
CLEAR
CANCEL
Click to insert envelope
$40.00
Click to insert card
Show Log
Operator Panel
OFF

# OO DESIGN OF SYSTEM

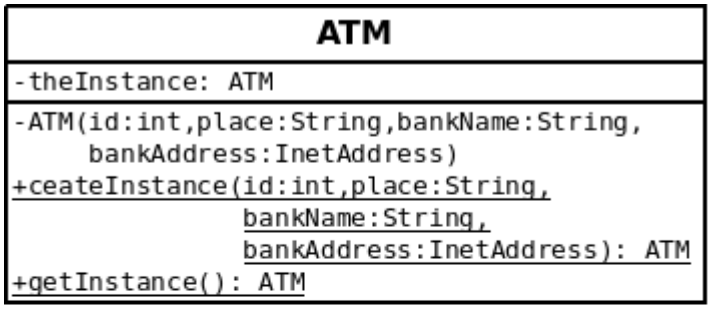


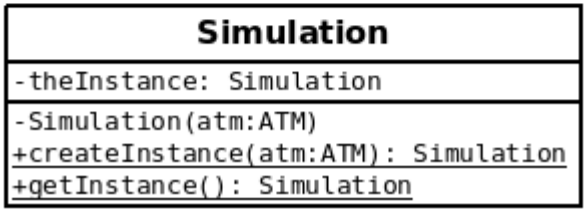



# Oo Design of System



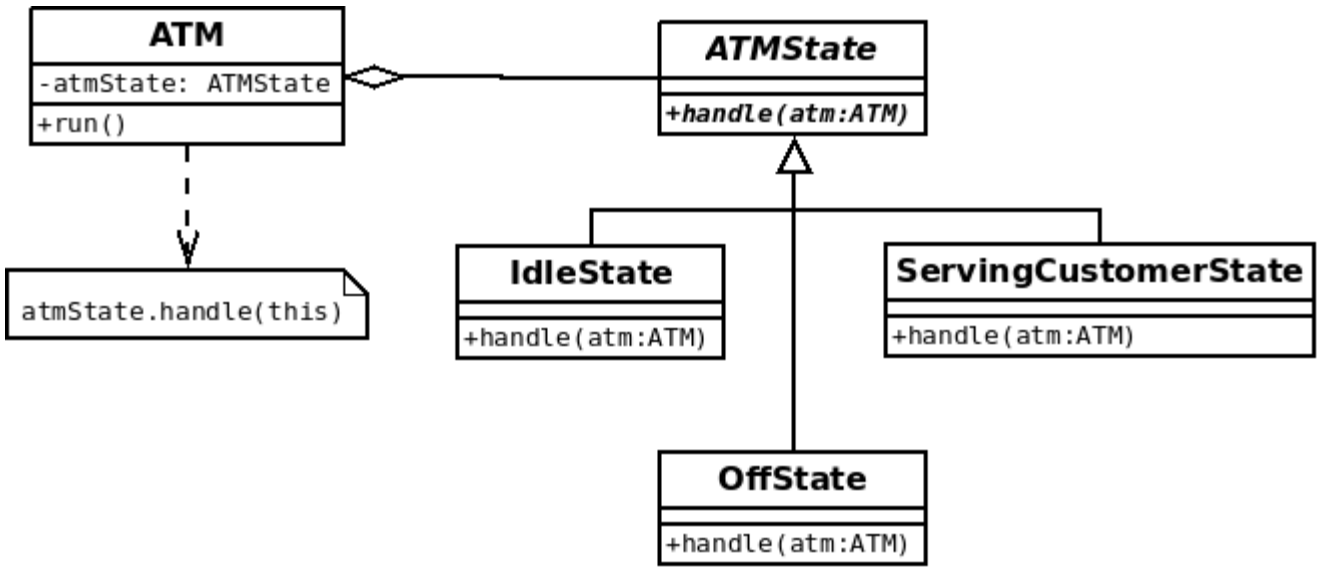
# Design Patterns



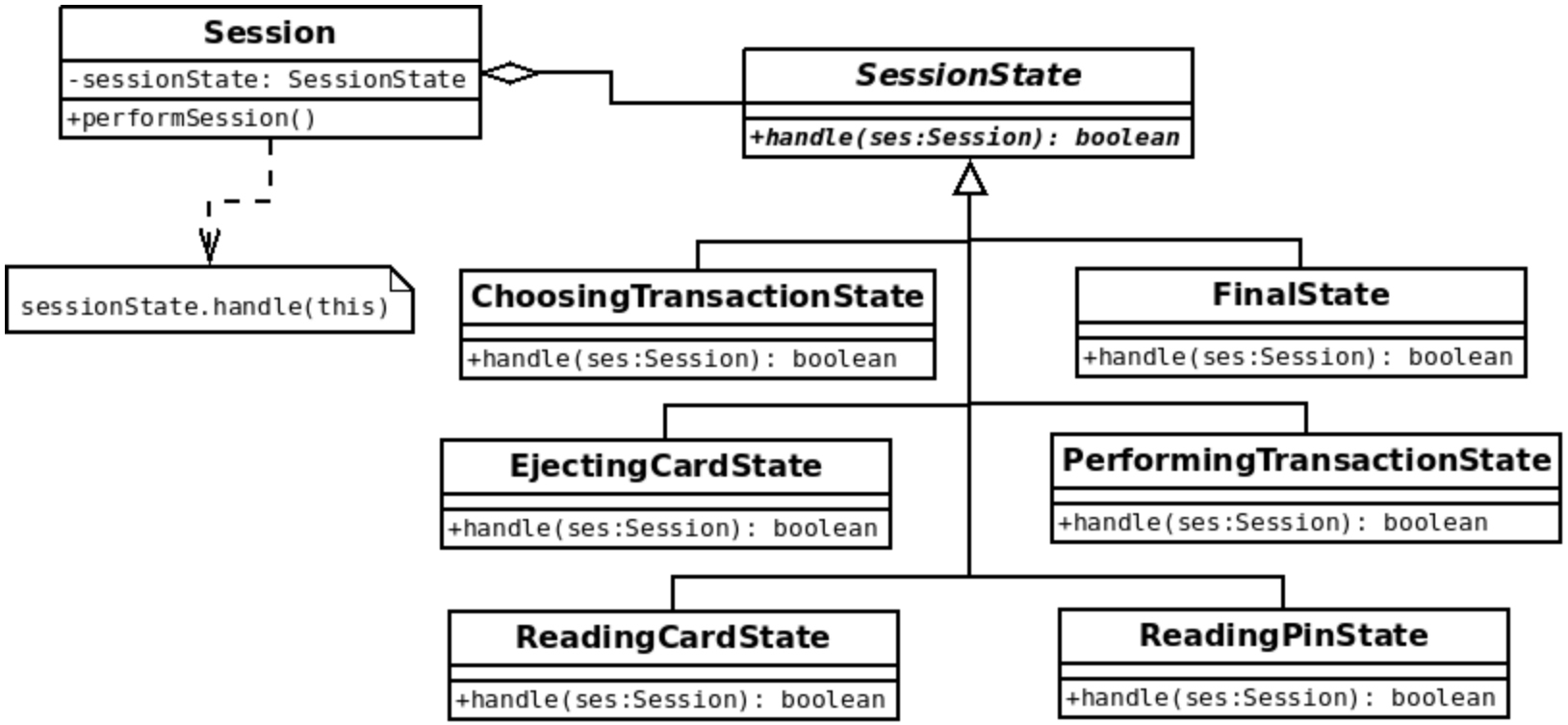

Singleton Pattern

# Design Patterns



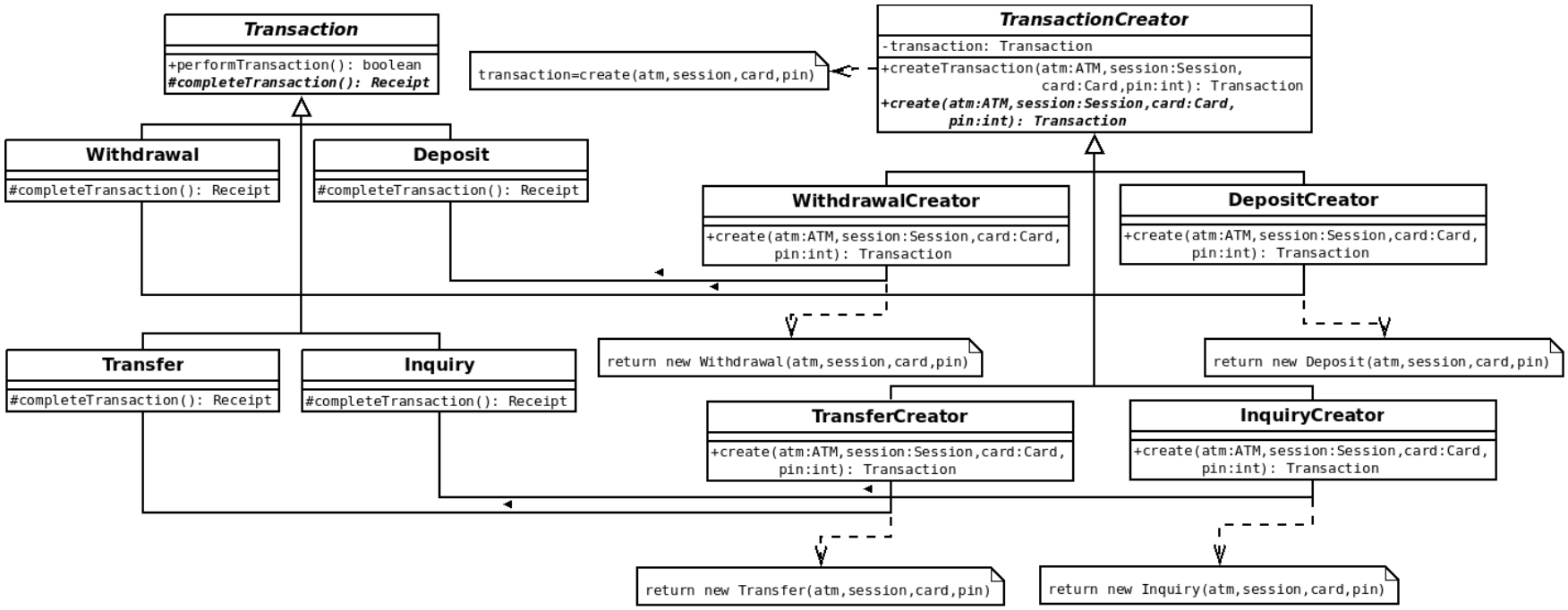

State Pattern

# Design Patterns




State Pattern

# Design Patterns



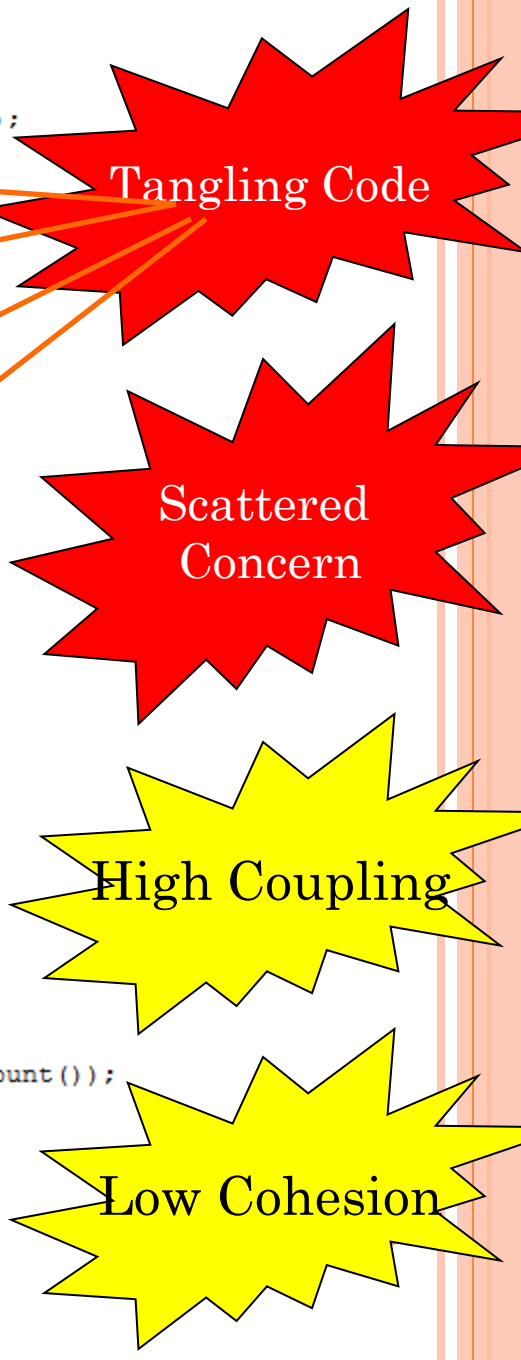

Factory Method Pattern

# Identifying Aspects

- Constraint Checking Aspect
- Security Handling Aspect
- Logging Aspect
- Transaction Handling Aspect

# Constraint Checking Aspect

```
private Status withdrawal(Message message, Balances balances)
{
     Status result = confirmAccount(message.getCard().getNumber(), message.getFromAccount());
     if(result.isSuccess())
     {

          .....

     }return new Success();
}

private Status completeDeposit(Message message, Balances balances)
{
     Status result = confirmAccount(message.getCard().getNumber(), message.getToAccount());
     if(result.isSuccess())
     {

         ......

     }
     return result;
}

private Status inquiry(Message message, Balances balances)
{
     Status result = confirmAccount(message.getCard().getNumber(), message.getFromAccount());
     if(result.isSuccess())
     {

       ....

     }
     return result;
}
private Status transfer(Message message, Balances balances)
{
     Status result = confirmAccount(message.getCard().getNumber(), message.getFromAccount());
     if(result.isSuccess())
     {

         .....

     }
     return result;
}
```



# Constraint Checking Aspect

```
pointcut confirmBalance(SimulatedBank sb, Message message):
    (call(private Status transfer(Message, Balances))
    || call(private Status withdrawal(Message, Balances)))
    && args(message, ..)
    && target(sb)
    ;

Status around(SimulatedBank sb, Message message) : confirmBalance(sb, message)
{
    Status result = sb.confirmBalance(message.getAmount(),
            sb.getAccountNumber(message.getCard().getNumber(),
                    message.getFromAccount() ));
    if(result.isSuccess())
    {
        Status tmp = proceed(sb,message);
        if(tmp != null)
        {
            result = tmp;
        }
    }
    return result;
}
```

# Security Handling Aspect

```
pointcut securityCheck(SimulatedBank sb, Message message) :
    call(private Status SimulatedBank.*(..)) && args(message, ..) && target(sb);

Status around(SimulatedBank sb, Message message) : securityCheck(sb, message)
{
    Status result = sb.checkPin(message.getPIN(), message.getCard().getNumber());
    if(result.isSuccess())
    {
        Status tmp = proceed(sb,message);
        if(tmp != null)
        {
            result = tmp;
        }
    }
    return result;
}
```

# Logging Aspect

```
pointcut message(Message message, Balances balances) :
    call(* NetworkToBank.sendMessage(..))
    && args(message, balances)
    ;

before(Message message, Balances balances) : message(message, balances)
{
    log.logSend(message);
}

after(Message message, Balances balances) returning(Status result) :
    message(message, balances)
{
    log.logResponse(result);
}
```

# Discussion

- In our other aspect, we removed the tangled code and obtained a base code.
- To implement the functionality of the tangled code, we added aspect code to the base code
- Object Oriented Design
- Improved using Design Patterns
- Some concerns were crosscutting
  - Logging
  - Constraint Checking
- We specified aspects to capture scattered and crosscutting concerns
- We improved modularity using AOP.

# DEMO

# Thank You