Федеральное государственное автономное
образовательное учреждение
высшего профессионального образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и Информационных Технологий

Кафедра Информатики

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

Шаблоны распределения обязанностей

Преподаватель _____ А. К. Погребников
подпись, дата

Студент гр. КИ10-11 031010132 _____ К.О. Васильев
подпись, дата

Красноярск 2013

1 Цель

Научиться применять шаблоны распределения обязанностей в проектировании объектно-ориентированных программных систем.

2 Задание

Спроектировать структуру объектно-ориентированной программной системы для одного из вариантов использования, созданного в предыдущей лабораторной работе, таким образом, чтобы к ней были применимы шаблоны распределения обязанностей:

- Information Expert (информационный эксперт)
- Creator (создатель)

3 Определение классов для реализации шаблонов

В лабораторной будут реализованы варианты использования «авторизация», «пополнение счета», «получение денег», «перевод» и «оплата услуг».

Общим для некоторых вариантов является то, что они — все «операции». Соответственно, логично будет создать класс, который будет пораждать операции (пополнение счета, получение денег, перевод, оплата услуг).

Также для выполнения операций необходимо проверять, достаточно ли средств на некотором счёте для выполнения операции. Логично создать класс, который будет «информационным экспертом» - и будет проверять, возможно ли снять деньги с какого-либа счёта.

4 Диаграмма классов

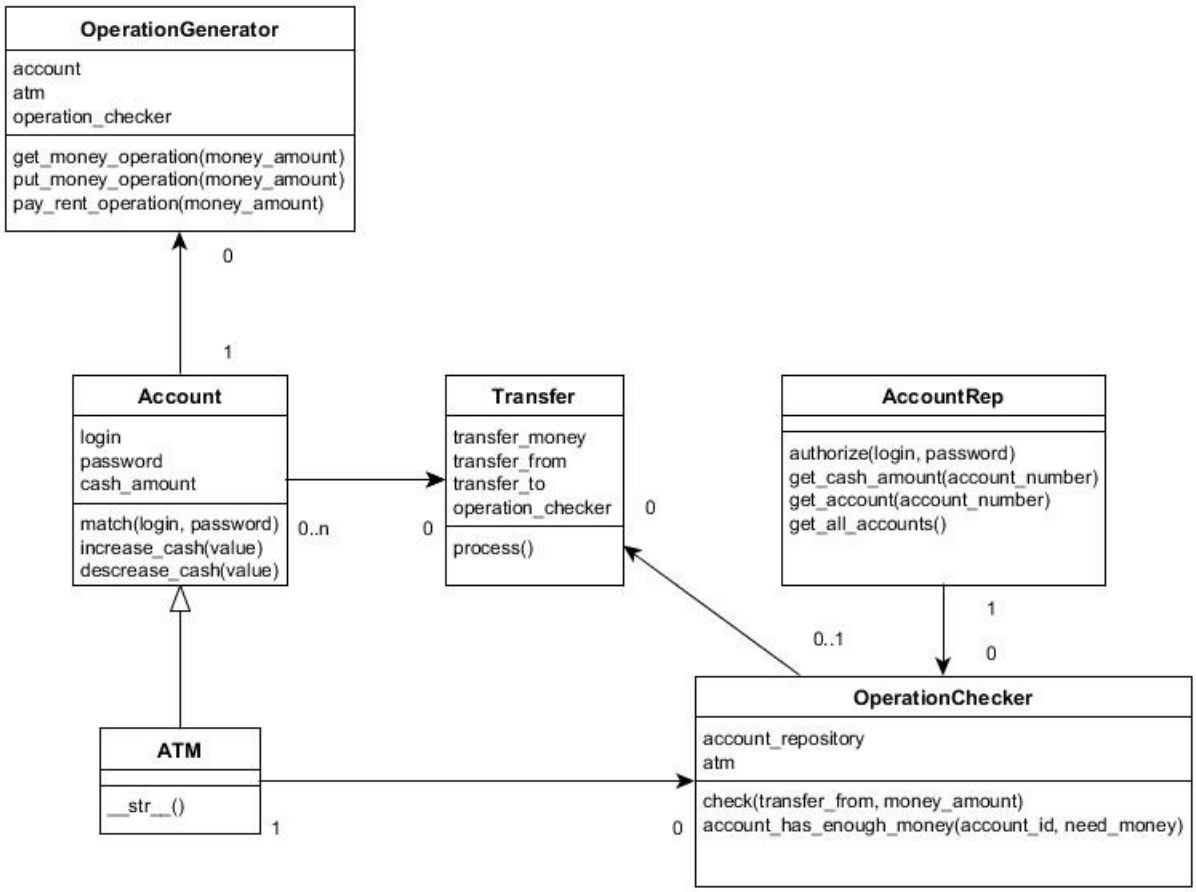

Рисунок 1 — Диаграмма классов программы

5 Исходный код классов, реализующих шаблоны проектирования

5.1 OperationChecker (Information Expert)

```
class OperationChecker:
    def __init__(self, account_rep, atm):
        self._account_repository = account_rep
        self._atm = atm

    def check(self, transfer_from, money_amount):
        for acc in transfer_from:
```

```
            if isinstance(acc, basestring):
                account_id = acc
            else:
                account_id = acc.account_number
                  if not self.account_has_enough_money(account_id,
money_amount):
                                    return '{0} has not enough
money'.format(account_id)

    def account_has_enough_money(self, account_id, need_money):
        if account_id == 'ATM':
            available_cash = self._atm.cash_amount
        else:
                                              available_cash =
self._account_repository.get_cash_amount(account_id)

        return available_cash >= need_money
```

5.2 OperationGenerator (Creator)

```
class OperationGenerator:
    def __init__(self, account, atm):
        self._account = account
        self._atm = atm
          self._operation_checker = OperationChecker(AccountRep(),
self._atm)

    def get_money_operation(self, money_amount):
                                  return Transfer(money_amount,
[self._account.account_number,          self._atm],          None,
self._operation_checker)

    def pay_rent_operation(self, money_amount):
                                  return Transfer(money_amount,
```

```
self._account.account_number,                    'RENT_ACCOUNT_NUMBER',
self._operation_checker)

    def put_money_operation(self, money_amount):
                            return    Transfer(money_amount,    None,
[self._account.account_number, self._atm], self._operation_checker)
```

6 Скриншоты работы программы

```
Enter login: buglar
Enter password: let me in
Wrong login/password combination!

Enter login: rich
Enter password: $$$

Enter command: info
Account #1010 has $500
Account #0023 has $10
Account #RENT_ACCOUNT_NUMBER has $0
Account #rich has $999999
ATM has $1000

Enter command: get money 2000
ATM has not enough money

Enter command: get money 500
Operation executed successfully

Enter command: info
Account #1010 has $500
Account #0023 has $10
Account #RENT_ACCOUNT_NUMBER has $0
Account #rich has $999499.0
ATM has $500.0

Enter command: pay rent 9999.0
Operation executed successfully

Enter command: info
Account #1010 has $500
Account #0023 has $10
Account #RENT_ACCOUNT_NUMBER has $9999.0
Account #rich has $989500.0
ATM has $500.0

Enter command: insert money 20
Operation executed successfully

Enter command: info
Account #1010 has $500
Account #0023 has $10
Account #RENT_ACCOUNT_NUMBER has $9999.0
Account #rich has $989520.0
ATM has $520.0
```

Рисунок 2 — Скриншот работы программы