Day 01 - Piscine Java

OOP/Collections

Резюме: Сегодня вы поймете, как правильно моделировать работу различных коллекций, а также сделаете полноценное приложение для работы с денежными переводами

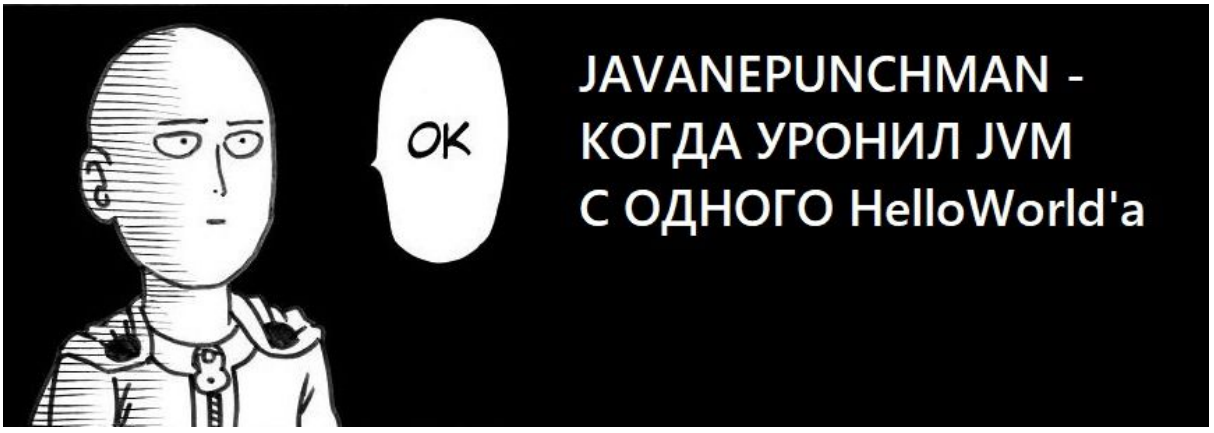
Contents

Chapter I
Preamble

Моделирование предметной области - наиболее сложная задача при разработке программного обеспечения. Правильное решение такой задачи позволит обеспечить гибкость реализованной системы.
Языки программирования, поддерживающие концепцию ООП, дают возможность качественно разбить бизнес-процесс на логические компоненты - классы.
Каждый класс должен отвечать принципам SOLID:

1. Принцип единственной ответственности - класс содержит ровно одну логически связанную функциональность (не может кофемашина отслеживать изменения в стеке вызовов, ее задача - готовить кофе).
2. Принцип открытости/закрытости - каждый класс может предоставлять возможность расширять свою функциональность. Тем не менее, данное расширение не должно предполагать изменения кода исходного класса.
3. Принцип подстановки Барбары Лисков - потомки только ДОПОЛНЯЮТ функциональность исходного класса, но не изменяют ее.
4. Принцип разделения интерфейса - существует множество точек (интерфейсов), описывающих логически связанное поведение. Отсутствует интерфейс общего назначения.
5. Принцип инверсии зависимости - система не должна быть зависима на конкретных сущностях, все зависимости строятся на абстракциях (интерфейсах).

Сегодня вам следует сосредоточиться на самом первом принципе SOLID.



Chapter II
General Rules

- Use this page as the only reference. Do not listen to any rumors and speculations about how to prepare your solution.
- Сейчас для вас существует только одна версия Java - 1.8. Убедитесь, что на вашем компьютере установлен компилятор и интерпретатор данной версии.
- Не запрещено использовать IDE для написания исходного кода и его отладки.
- Код чаще читается, чем пишется. Внимательно изучите представленный документ с правилами оформления кода. В каждом задании обязательно придерживайтесь общепринятых стандартов Oracle - https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html
- Комментарии в исходном коде вашего решения запрещены. Они мешают восприятию.
- Pay attention to the permissions of your files and directories.
- To be assessed your solution must be in your GIT repository.
- Your solutions will be evaluated by your piscine mates.
- You should not leave in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid accidents.
- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.
- Have a question? Ask your neighbor on the right. Otherwise, try with your neighbor on the left.
- Your reference manual: mates / Internet / Google. И еще, для любых ваших вопросов существует ответ на Stackoverflow. Научитесь правильно их задавать.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- And may the Force be with you!
- Не откладывайте на завтра то, что можно было сделать вчера ;)

Chapter III
Introduction to exercises

Система внутренних денежных переводов является неотъемлемой частью многих корпоративных приложений.

Сегодня вам необходимо автоматизировать бизнес-процесс, связанный с переводами некоторых сумм между участниками нашей системы.

Каждый пользователь системы может сделать перевод определенной суммы другому пользователю. Нам необходимо гарантировать, что, даже если мы потеряем историю переводов и поступлений по конкретному пользователю, мы все равно сможем восстановить эту информацию.

Внутри системы все денежные операции хранятся в виде пары приход-расход. Например, John сделал перевод пользователю с именем Mike на сумму в 500$. Система должна сохранить этот перевод для обоих пользователей:

```
John -> Mike, -500, OUTCOME, ID-перевода
Mike -> John, +500, INCOME, ID-перевода
```

Для того, чтобы можно было восстановить связь таких пар, следует использовать идентификаторы каждого перевода.

Очевидно, что в такой сложной системе есть вероятность того, что запись о переводе может потеряться и не быть зафиксирована для одного из пользователей (для имитации и отладки такой ситуации у вас, как у разработчика, должна быть возможность удалить информацию о переводе только у одного из пользователей). Ввиду того, что такие случаи возможны, необходимо предусмотреть функционал, который покажет все “неподтвержденные переводы”, т.е. такие, которые зафиксированы только у одного пользователя для последующего решения вопроса.

Ниже приведен набор упражнений, последовательное выполнение которых приведет вас к поставленной задаче.

Chapter IV
Exercise 00 - Models

Exercise 00: Models	
Turn-in directory	ex00
Files to turn-in	User.java, Transaction.java, Program.java
Возможно использование пользовательских классов, а также:	
Типы (+ все методы данных типов)	Integer, String, UUID, перечисления

Первой задачей для вас станет разработка основных моделей предметной области, а именно - классов `User`(Пользователь) и `Transaction`(Перевод).

В рамках системы вполне вероятна ситуация, когда разные пользователи будут иметь одинаковые имена. Данную проблему необходимо решить с помощью добавления особого поля для пользователя - его уникального идентификатора. Таким идентификатором будет любое целое число. Конкретная логика создания идентификатора будет описана в следующем упражнении.

Таким образом, для класса `User` характерен следующий набор состояний (полей):

- Идентификатор
- Имя
- Баланс

Класс `Transaction` описывает денежный перевод между двумя пользователями. Здесь также необходимо определить уникальный идентификатор. Поскольку количество таких транзакций потенциально очень велико, определим идентификатор как строку вида UUID. Таким образом, для класса `Transaction` характерен следующий набор состояний (полей):

- Идентификатор
- Получатель (тип `User`)
- Отправитель (тип `User`)
- Категория перевода (приход, расход)
- Сумма перевода

Необходимо обеспечить проверку начального баланса для пользователя (он не может быть отрицательным), а также для исходящей (только отрицательные суммы) и входящей (только положительные суммы) транзакций (использование get/set-методов). Пример использования данных классов должен быть в файле Program (создание, инициализация, вывод содержимого объектов в консоль). Все данные для полей классов “захардкодить” в `Program`.

Chapter V
Exercise 01 - Id Generator

Exercise 01: id Generator	
Turn-in directory	ex01
Files to turn-in	UserIdsGenerator.java, User.java, Program.java
Возможно использование всех разрешений предыдущего упражнения	

Вам необходимо обеспечить уникальность каждого идентификатора пользователя. Для этого требуется создать класс `UserIdsGenerator`. Поведение объекта данного класса определяет функционал генерации ID пользователей.

В современных системах управления базами данных реализован принцип автоинкремента, когда каждый новый ID представляет собой значение предыдущего сгенерированного идентификатора + 1.

Таким образом, класс `UserIdsGenerator` содержит последний сгенерированный идентификатор в качестве состояния. Поведение `UserIdsGenerator` определено методом `int generateId()`, возвращающим новый сгенерированный идентификатор при каждом своем вызове.

Пример использования данных классов должен быть в файле Program (создание, инициализация, вывод содержимого объектов в консоль).

Примечания:

- Необходимо обеспечить существование только одного объекта `UserIdsGenerator` (см. паттерн `Singleton`). Поскольку наличие нескольких объектов данного класса не может гарантировать уникальности идентификатора для всех пользователей.
- Идентификатор пользователя должен иметь доступ “только для чтения”. Поскольку инициализируется он один раз - в момент создания объекта, и не может быть изменен во время последующей работы программы.
- В конструктор класса `User` необходимо добавить временную логику инициализации идентификатора:

```
public User(...) {

	this.id = UserIdsGenerator.getInstance().generateId();

}
```

Chapter VI
Exercise 02 - List of users

Exercise 02: List of users	
Turn-in directory	ex02
Files to turn-in	UsersList.java, UsersArrayList.java, User.java,Program.java, etc.
Возможно использование всех разрешений предыдущего упражнения + throw	

Теперь необходимо реализовать функциональность, позволяющую хранить пользователей в момент работы приложения.

На данный момент ваше приложение не имеет персистентного хранилища (файловая система, база данных). Но мы не хотим, чтобы логика нашего приложения зависела от способа реализации хранения пользователей. Поэтому, для большей гибкости, определим интерфейс `UsersList`, описывающий следующее поведение:

- Добавить пользователя
- Получить пользователя по ID
- Получить пользователя по индексу
- Получить количество пользователей

Наличие данного интерфейса позволит разработать бизнес-логику вашего приложения таким образом, что конкретная реализация хранилища не будет влиять на остальные компоненты системы.

Также Реализуем класс `UsersArrayList`, имплементирующий интерфейс `UsersList`.

Данный класс должен использовать массив в качестве хранилища данных пользователей. Размер массива по умолчанию - 10. В случае, если массив заполнен полностью, его размер следует увеличить в полтора раза. Метод добавления пользователя включает объект типа `User` в первую пустую (свободную) ячейку массива.

В случае, если была произведена попытка получения пользователя с несуществующим ID, должно быть выброшено созданное вами непроверяемое исключение `UserNotFoundException`.

Пример использования данных классов должен быть в файле Program (создание, инициализация, вывод содержимого объектов в консоль).

Примечание:

Аналогичным образом устроен встроенный Java-класс `ArrayList<T>`. Моделируя поведение данного класса самостоятельно, вы получите правильное представление о механизмах работы такого класса стандартной библиотеки.

Chapter VII
Exercise 03 - List of transactions

Exercise 03: List of transactions	
Turn-in directory	ex03
Files to turn-in	TransactionsList.java, TransactionsLinkedList.java, User.java, Program.java, etc.
Возможно использование всех разрешений предыдущего упражнения	

В отличие от пользователей, список транзакций требует особого подхода к реализации. Поскольку количество операции создания транзакций может быть чрезвычайно большим, нам необходим такой способ хранения, который позволит избежать дорогостоящего метода расширения размера массива.

В данном задание вам предлагается реализовать интерфейс `TransactionsList`, описывающий следующее поведение:

- Добавить транзакцию
- Удалить транзакцию по ID (в данном случае, идентификатор - UUID-строка)
- Преобразовать в массив (ex. Transaction[] toArray())

Реализация списка транзакций должна основываться на концепции связного списка (LinkedList) - в классе `TransactionsLinkedList`. Таким образом, каждая транзакция в качестве поля должна содержать ссылку на объект следующей транзакции.

В случае, если была произведена попытка удаления транзакции с несуществующим ID, должно быть выброшено Runtime-исключение `TransactionNotFoundException`.

Пример использования данных классов должен быть в файле Program (создание, инициализация, вывод содержимого объектов в консоль).

Примечание:

- Для того, чтобы каждый пользователь мог хранить список своих транзакций, необходимо добавить в класс User поле `transactions` типа `TransactionsList`.
- Добавление транзакции должно происходить за ОДНУ операцию (O(1))
- Аналогичным образом устроен встроенный Java-класс `LinkedList<T>`, представляющий из себя двунаправленный связный список.

Chapter VIII
Exercise 04 - Business Logic

Exercise 04: Business Logic	
Turn-in directory	ex04
Files to turn-in	TransactionsService.java, Program.java, etc.
Возможно использование всех разрешений предыдущего упражнения	

Слой бизнес-логики приложения размещается в классах-сервисах. Такие классы содержат основные алгоритмы системы, автоматизируемые процессы и т.д. Как правило, данные классы проектируются согласно паттерну `Facade`, позволяющему инкапсулировать поведение нескольких классов внутри себя.

В данном случае класс `TransactionsService` должен содержать поле типа `UsersList` для взаимодействия с пользователями и предоставлять следующую функциональность:

- Добавление пользователя
- Получение баланса пользователя
- Осуществление перевода (указываются id пользователей и сумма перевода). В этом случае создается две транзакции с типами ПРИХОД/РАСХОД и добавляются получателю и отправителю, id обеих транзакций должны совпадать.
- Получение переводов конкретного пользователя (возвращается МАССИВ переводов). Удаление транзакции по ID у конкретного пользователя (указываются ID транзакции и ID пользователя)
- Проверить корректность транзакций (возвращается МАССИВ транзакций, не имеющих пары).

В случае, если была произведена попытка осуществления перевода, превышающего остаточный баланс пользователя, должно быть выброшено Runtime-исключение `IllegalTransactionException`.

Пример использования данных классов должен быть в файле Program (создание, инициализация, вывод содержимого объектов в консоль).

Chapter IX
Exercise 05 - Menu

Exercise 05: Menu	
Turn-in directory	ex05
Files to turn-in	Menu.java, Program.java, etc.
Возможно использование всех разрешений предыдущего упражнения, а также	
try/catch	

Завершением вашей работы станет работающее приложение с консольным меню. Функциональность меню должна быть реализована в соответствующем классе, имеющем поле-ссылку на `TransactionsService`.

Каждый пункт меню должен сопровождаться номером команды, которую вводит пользователь для вызова того или иного действия.

Приложение должно обеспечивать возможность запуска в двух режимах - production (стандартный режим) и dev (появляется возможность удалить информацию о переводе по id у конкретного пользователя, а также запустить функцию проверки корректности всех переводов).

В случае, если было выброшено какое-либо исключение, должно быть выведено информативное сообщение об ошибке с возможностью последующего ввода корректных данных.

Сценарий работы приложения (программа должна в точности повторять приведенный пример вывода):

```
$ java Program --profile=dev

1. Добавить пользователя
2. Посмотреть баланс пользователей
3. Осуществить перевод
4. Посмотреть все переводы конкретного
пользователя
5. DEV - удалить перевод по ID
6. DEV - проверить корректность переводов
```

7. Завершить выполнение
-> 1
Введите имя и баланс пользователя
-> Jonh 777
Пользователь добавлен с id = 1
---
1. Добавить пользователя
2. Посмотреть баланс пользователей
3. Осуществить перевод
4. Посмотреть все переводы конкретного пользователя
5. DEV - удалить перевод по ID
6. DEV - проверить корректность переводов
7. Завершить выполнение
-> 1
Введите имя и баланс пользователя
-> Mike 100
Пользователь добавлен с id = 2
---
1. Добавить пользователя
2. Посмотреть баланс пользователей
3. Осуществить перевод
4. Посмотреть все переводы конкретного пользователя
5. DEV - удалить перевод по ID
6. DEV - проверить корректность переводов
7. Завершить выполнение
-> 3
Введите id-отправителя, id-получается и сумму перевода
-> 1 2 100
Перевод осуществлен
---
1. Добавить пользователя
2. Посмотреть баланс пользователей
3. Осуществить перевод
4. Посмотреть все переводы конкретного пользователя

5. DEV - удалить перевод по ID
6. DEV - проверить корректность переводов
7. Завершить выполнение
-> 3
Введите id-отправителя, id-получается и сумму перевода
-> 1 2 150
Перевод осуществлен
---
1. Добавить пользователя
2. Посмотреть баланс пользователей
3. Осуществить перевод
4. Посмотреть все переводы конкретного пользователя
5. DEV - удалить перевод по ID
6. DEV - проверить корректность переводов
7. Завершить выполнение
-> 3
Введите id-отправителя, id-получается и сумму перевода
-> 1 2 50
Перевод осуществлен
---
1. Добавить пользователя
2. Посмотреть баланс пользователей
3. Осуществить перевод
4. Посмотреть все переводы конкретного пользователя
5. DEV - удалить перевод по ID
6. DEV - проверить корректность переводов
7. Завершить выполнение
-> 2
Введите id пользователя
-> 2
Mike - 400
---
1. Добавить пользователя
2. Посмотреть баланс пользователей

```
3. Осуществить перевод
4. Посмотреть все переводы конкретного
пользователя
5. DEV - удалить перевод по ID
6. DEV - проверить корректность переводов
7. Завершить выполнение
-> 4
Введите id пользователя
-> 1
To Mike(id = 2) -100 with id = cc128842-2e5c-4cca-a44c-7829f53fc31f
To Mike(id = 2) -150 with id = 1fc852e7-914f-4bfd-913d-0313aab1ed99
TO Mike(id = 2) -50 with id = ce183f49-5be9-4513-bd05-8bd82214eaba
---------------------------------------------------------
1. Добавить пользователя
2. Посмотреть баланс пользователей
3. Осуществить перевод
4. Посмотреть все переводы конкретного
пользователя
5. DEV - удалить перевод по ID
6. DEV - проверить корректность переводов
7. Завершить выполнение
-> 5
Введите id пользователя и id перевода
-> 1 1fc852e7-914f-4bfd-913d-0313aab1ed99
Перевод To Mike(id = 2) 150 удален
---------------------------------------------------------
1. Добавить пользователя
2. Посмотреть баланс пользователей
3. Осуществить перевод
4. Посмотреть все переводы конкретного
пользователя
5. DEV - удалить перевод по ID
6. DEV - проверить корректность переводов
7. Завершить выполнение
-> 6
Результаты проверки:
Mike(id = 2) имеет неподтвержденный перевод id =
1fc852e7-914f-4bfd-913d-0313aab1ed99 от John(id = 1) на сумму 150
```

CHECKLIST

https://docs.google.com/document/d/1Y30QP2Hapd2fkY5l36GxkRq3h-WDzpVyb3lcF2p-v3M/edit?usp=sharing