

Модуль 01 - Бассейн Java

OOP/Collections

Резюме: Сегодня вы научитесь правильно моделировать работу различных коллекторов и создавать полноценное приложение для денежных переводов.

Содержание

I Предисловие

II Инструкции

III Введение в упражнения

IV Упражнение 00: Модели

V Упражнение 01: Генератор ID

VI Упражнение 02: Список пользователей

VII Упражнение 03. Список транзакций

VIII Упражнение 04. Бизнес-логика

IX Упражнение 05: Меню

# Глава I

## Предисловие

Моделирование предметной области — самая сложная задача в разработке программного обеспечения. Правильное решение этой задачи обеспечивает гибкость внедряемой системы.

Языки программирования, поддерживающие концепцию объектно-ориентированного программирования (ООП), позволяют эффективно разделить бизнес-процессы на логические компоненты, называемые классами. Каждый класс должен соответствовать принципам SOLID:

- Принцип единой ответственности: класс содержит одну логически связанную функциональность (кофемашина не может очищать и отслеживать изменения в стеке вызовов, ее назначение — приготовление кофе).
- Принцип открытости-закрытости: каждый класс может предложить вариант расширения своей функциональности. Однако такое расширение не должно предусматривать модификацию исходного кода класса.
- Принцип замещения Лискова: производные классы только ДОПОЛНЯЮТ функциональность исходного класса, не изменяя его.
- Принцип разделения интерфейсов: существует множество точек (интерфейсов), описывающих логически связанное поведение. Универсального интерфейса нет.
- принцип инверсии зависимостей: система не должна зависеть от конкретных сущностей; все зависимости основаны на абстракциях (интерфейсах).

Сегодня вы должны сосредоточиться на первом принципе SOLID.

## Глава II

### Инструкции

- Используйте эту страницу как единственную ссылку. Не слушайте никаких слухов и домыслов о том, как приготовить свой раствор.
- Теперь у вас есть только одна версия Java, 1.8. Убедитесь, что на вашем компьютере установлены компилятор и интерпретатор этой версии.
- Вы можете использовать IDE для написания и отладки исходного кода.
- Код читается чаще, чем пишется. Внимательно прочитайте документ, в котором приведены правила форматирования кода. При выполнении каждой задачи убедитесь, что вы следуете общепринятым стандартам Oracle.
- Комментарии в исходном коде вашего решения запрещены. Они затрудняют чтение кода.
- Обратите внимание на разрешения ваших файлов и каталогов.
- Для оценки ваше решение должно находиться в вашем репозитории GIT.
- Ваши решения будут оценены вашими товарищами.
- Вы не должны оставлять в своем каталоге никаких других файлов, кроме тех, которые явно указаны в инструкциях к упражнению. Рекомендуется изменить ваш .gitignore, чтобы избежать несчастных случаев.
- Когда вам нужно получить точный вывод в ваших программах, запрещается отображать предварительно рассчитанный вывод вместо правильного выполнения упражнения.
- Есть вопрос? Спросите у соседа справа. В противном случае попробуйте с соседом слева.
- Ваш справочник: гугля/интернет/гугл. И еще кое-что. На Stackoverflow есть ответ на любой вопрос, который у вас может возникнуть. Научитесь правильно задавать вопросы.
- Внимательно прочитайте примеры. Они могут требовать вещи, которые иначе не указаны в теме.
- Используйте "System.out" для вывода

## Глава III

### Введение в упражнения

Внутренняя система денежных переводов является неотъемлемой частью многих корпоративных приложений. Ваша сегодняшняя задача — автоматизировать бизнес-процесс, связанный с переводами определенных сумм между участниками нашей системы.

Каждый пользователь системы может перевести определенную сумму другому пользователю. Нам нужно убедиться, что даже если мы потеряем историю входящих и исходящих переводов для конкретного пользователя, мы все равно сможем восстановить эту информацию.

Внутри системы все денежные операции хранятся в виде пар дебет/кредит. Например, Джон перевел Майку 500 долларов. Система сохраняет транзакцию для обоих пользователей:

Джон -> Майк, -500, РЕЗУЛЬТАТ, идентификатор транзакции

Майк -> Джон, +500, ДОХОД, идентификатор транзакции


Для восстановления соединения внутри таких пар необходимо использовать идентификаторы каждой транзакции.

Очевидно, что в такой сложной системе запись о переводе может быть утеряна — она может быть не записана для одного из пользователей (для эмуляции и отладки такой ситуации разработчику необходимо иметь возможность удалять данные о переводе у одного из пользователей индивидуально). Поскольку такие ситуации реальны, требуется функционал для отображения всех «неподтвержденных переводов» (транзакций, зарегистрированных только для одного пользователя) и решения таких проблем.

Ниже приведен набор упражнений, которые вы можете выполнять одно за другим для решения задачи.

## Глава IV

### Упражнение 00: Модели

	Exercise 00
Models	
Turn-in directory : <i>ex00/</i>	
Files to turn in : <i>User.java, Transaction.java, Program.java</i>	
Allowed functions : User classes can be employed, along with: Types (+ all methods of these types) : Integer, String, UUID, enumerations	

Ваша первая задача — разработать базовые модели предметной области, а именно классы пользователей и транзакций. Вполне вероятно, что у разных пользователей в системе будет одно и то же имя. Эту проблему нужно решить, добавив специальное поле для уникального идентификатора пользователя. Этот идентификатор может быть любым целым числом. Конкретная логика создания идентификатора описана в следующем упражнении.

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

- Identifier
- Name
- Balance

Класс транзакции описывает денежный перевод между двумя пользователями. Здесь также должен быть определен уникальный идентификатор. Так как количество таких транзакций может быть очень большим, определим идентификатор как строку `UUID`. Таким образом, для класса `Transaction` характерен следующий набор состояний (полей):


- Identifier
- Recipient (User type)
- Sender (User type)
- Transfer category (debits, credits)
- Transfer amount

Необходимо проверить начальный баланс пользователя (он не может быть отрицательным), а также баланс для исходящих (только отрицательные суммы) и входящих (только положительные суммы) транзакций (использование методов `get/set`).

Пример использования таких классов должен содержаться в файле программы (создание, инициализация, вывод содержимого объекта на консоль). Все данные для полей класса должны быть жестко запрограммированы в `Program`.

## Глава V

### Упражнение 01. Генератор идентификаторов

	Exercise 01
ID Generator	
Turn-in directory : <i>ex01/</i>	
Files to turn in : <i>UserIdsGenerator.java, User.java, Program.java</i>	
Allowed functions : All permissions from the previous exercise can be used	

Убедитесь, что каждый идентификатор пользователя уникален. Для этого создайте класс `UserIdsGenerator`. Поведение объекта этого класса определяет функциональность для генерации идентификаторов пользователей. Современные системы управления базами данных поддерживают принцип автоинкремента, при котором каждый новый идентификатор является значением ранее сгенерированного идентификатора +1.

Итак, класс `UserIdsGenerator` содержит последний сгенерированный идентификатор в качестве своего состояния. Поведение `UserIdsGenerator` определяется методом `int generateId()`, который возвращает вновь сгенерированный идентификатор при каждом вызове.

Пример использования таких классов должен содержаться в файле программы (создание, инициализация, вывод содержимого объекта на консоль).


Заметки:

- Убедитесь, что существует только один объект `UserIdsGenerator` (см. шаблон `Singleton`). Это необходимо, поскольку наличие нескольких объектов этого класса не может гарантировать, что все идентификаторы пользователей уникальны.
- Идентификатор пользователя должен быть доступен только для чтения, так как он инициализируется только один раз (при создании объекта) и не может быть изменен в дальнейшем при выполнении программы.
- В конструктор класса `User` следует добавить временную логику инициализации идентификатора:

```
public User(...) {  
    this.id = UserIdsGenerator.getInstance().generateId();  
}
```

## Глава VI

### Упражнение 02. Список пользователей

	Exercise 02
List of Users	
Turn-in directory : <i>ex02/</i>	
Files to turn in : UsersList.java, UsersArrayList.java, User.java, Program.java, etc.	
Allowed functions : All permissions from the previous exercise + throw can be used.	

Теперь нам нужно реализовать функционал для хранения пользователей во время работы программы.

На данный момент ваше приложение не имеет постоянного хранилища (например, файловой системы или базы данных). Однако мы хотим избежать зависимости вашей логики от способа реализации пользовательского хранилища. Чтобы обеспечить большую гибкость, давайте определим интерфейс UsersList, который описывает следующее поведение:

- Add a user
- Retrieve a user by ID
- Retrieve a user by index
- Retrieve the number of users

Этот интерфейс позволит разработать бизнес-логику вашего приложения так, чтобы конкретная реализация хранилища не влияла на другие компоненты системы.

Мы также реализуем класс UsersArrayList, реализующий интерфейс UsersList.

Этот класс должен использовать массив для хранения пользовательских данных. Размер массива по умолчанию равен 10. Если массив заполнен, его размер увеличивается вдвое. Метод добавления пользователя помещает объект типа User в первую пустую (свободную) ячейку массива.

В случае попытки получить пользователя с несуществующим идентификатором должно быть выдано непроверенное исключение UserNotFoundException.

Пример использования таких классов должен содержаться в файле Программы (создание, инициализация,

печать содержимого объекта на консоли).


Примечание:



Вложенный класс `Java ArrayList<T>` имеет ту же структуру. Смоделировав поведение этого класса самостоятельно, вы научитесь использовать механизмы этого стандартного библиотечного класса.

## Глава VII

### Упражнение 03. Список транзакций

	Exercise 03
List of transactions	
Turn-in directory : <i>ex03/</i>	
Files to turn in : TransactionsList.java, TransactionsLinkedList.java, User.java, Program.java, etc.	
Allowed functions : All permissions from the previous exercise can be used	

В отличие от пользователей, список транзакций требует особого подхода к реализации. Поскольку количество операций создания транзакций может быть очень большим, нам нужен метод хранения, чтобы избежать дорогостоящего увеличения размера массива.

В этой задаче мы предлагаем вам создать интерфейс TransactionsList, описывающий следующее поведение:

- Добавить транзакцию
- Удалить транзакцию по ID (в данном случае используется строковый идентификатор UUID)
- Преобразование в массив (например, Transaction[] toArray())

Список транзакций должен быть реализован в виде связанного списка (LinkedList).

в классе TransactionsLinkedList. Поэтому каждая транзакция должна содержать поле со ссылкой на следующий объект транзакции.

При попытке удалить транзакцию с несуществующим идентификатором должно быть сгенерировано исключение времени выполнения TransactionNotFoundException.


Пример использования таких классов должен содержаться в файле программы (создание, инициализация, вывод содержимого объекта на консоль).

Примечание:

- Нам нужно добавить поле транзакций `muna TransactionsList` в класс `User`, чтобы каждый пользователь мог хранить список своих транзакций.
- Транзакция должна быть добавлена с помощью ОДНОЙ операции ( $O(1)$ )
- Вложенный Java-класс `LinkedList<T>` имеет ту же структуру — двусторонний связанный список.

## Глава VIII

### Упражнение 04. Бизнес-логика

	Exercise 04
Business Logic	
Turn-in directory : <i>ex04/</i>	
Files to turn in : TransactionsService.java, Program.java, etc.	
Allowed functions : All permissions from the previous exercise can be used	

Уровень бизнес-логики приложения находится в классах обслуживания. Такие классы содержат базовые алгоритмы системы, автоматизированные процессы и т. д. Эти классы обычно разрабатываются на основе паттерна Фасад, который может инкапсулировать поведение нескольких классов. В этом случае класс TransactionsService должен содержать поле типа UsersList для взаимодействия с пользователем и обеспечивать следующий функционал:


- Добавление пользователя
- Получение баланса пользователя
- Выполнение транзакции перевода (указаны идентификаторы пользователей и сумма перевода). В этом случае создаются две транзакции типа ДЕБЕТ/КРЕДИТ и добавляются к получателю и отправителю. ID обеих транзакций должны совпадать
- Получение переводов определенного пользователя (возвращается МАССИВ переводов). Удаление транзакции по ID для конкретного пользователя (указывается ID транзакции и ID пользователя)
- Проверить достоверность транзакций (возвращает МАССИВ непарных транзакций).

В случае попытки перевода суммы, превышающей остаточный баланс пользователя, должно быть сгенерировано исключение времени выполнения IllegalArgumentException.

Пример использования таких классов должен содержаться в файле программы (создание, инициализация, вывод содержимого объекта на консоль).

## Глава IX.

### Упражнение 05: Меню

	Exercise 05
Menu	
Turn-in directory : <i>ex05/</i>	
Files to turn in : <i>Menu.java, Program.java, etc.</i>	
Allowed functions : All permissions from the previous exercise can be used, as well as <i>try/catch</i>	

- В результате вы должны создать работающее приложение с консолью
- меню. Функциональность меню должна быть реализована в соответствующем классе со ссылкой на поле для `TransactionsService`.
- Каждый пункт меню должен сопровождаться номером команды, введенной пользователем для вызова действия.
- Приложение должно поддерживать два режима запуска — рабочий (стандартный режим) и `dev` (где информация о передаче для конкретного пользователя может быть удалена по идентификатору пользователя, а также может быть запущена функция, проверяющая действительность всех передач).
- Если возникает исключение, должно появиться сообщение, содержащее информацию об ошибке, и пользователю должна быть предоставлена возможность ввести действительные данные.
- Сценарий работы приложения следующий (программа должна тщательно следовать этому выходному примеру):

```
$ java Program --profile=dev
1. Add a user
2. View user balances
3. Perform a transfer
4. View all transactions for a specific user
5. DEV - remove a transfer by ID
6. DEV - check transfer validity
7. Finish execution
-> 1
Enter a user name and a balance
-> Jonh 777
User with id = 1 is added
-----
1. Add a user
2. View user balances
3. Perform a transfer
4. View all transactions for a specific user
5. DEV - remove a transfer by ID
6. DEV - check transfer validity
7. Finish execution
-> 1
Enter a user name and a balance
-> Mike 100
User with id = 2 is added
-----
1. Add a user
2. View user balances
3. Perform a transfer
4. View all transactions for a specific user
5. DEV - remove a transfer by ID
6. DEV - check transfer validity
7. Finish execution
-> 3
Enter a sender ID, a recipient ID, and a transfer amount
-> 1 2 100
The transfer is completed
-----
1. Add a user
2. View user balances
3. Perform a transfer
4. View all transactions for a specific user
5. DEV - remove a transfer by ID
```

```

6. DEV - check transfer validity
7. Finish execution
-> 3
Enter a sender ID, a recipient ID, and a transfer amount
-> 1 2 150
The transfer is completed
-----
1. Add a user
2. View user balances
3. Perform a transfer
4. View all transactions for a specific user
5. DEV - remove a transfer by ID
6. DEV - check transfer validity
7. Finish execution
-> 3
Enter a sender ID, a recipient ID, and a transfer amount
-> 1 2 50
The transfer is completed
-----
1. Add a user
2. View user balances
3. Perform a transfer
4. View all transactions for a specific user
5. DEV - remove a transfer by ID
6. DEV - check transfer validity
7. Finish execution
-> 2
Enter a user ID -> 2
Mike - 400
-----
1. Add a user
2. View user balances
3. Perform a transfer
4. View all transactions for a specific user
5. DEV - remove a transfer by ID
6. DEV - check transfer validity
7. Finish execution
-> 4
Enter a user ID
-> 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. Add a user
2. View user balances
3. Perform a transfer
4. View all transactions for a specific user
5. DEV - remove a transfer by ID 6. DEV - check transfer validity
7. Finish execution
-> 5
Enter a user ID and a transfer ID
-> 1 1fc852e7-914f-4bfd-913d-0313aab1ed99
Transfer To Mike(id = 2) 150 removed -----
1. Add a user
2. View user balances
3. Perform a transfer
4. View all transactions for a specific user
5. DEV - remove a transfer by ID 6. DEV - check transfer validity
7. Finish execution
-> 6
Check results:
Mike(id = 2) has an unacknowledged transfer id = 1fc852e7-914f-4bfd-913d-0313aab1ed99 from John(id = 1) for 150

```