



М о д у л ь 01 - Piscine Java

ООП/коллекции

Резюме: Сегодня вы узнаете, как правильно смоделировать работу различных инкассаций и создать полноценное приложение для перевода денег

С о д е р ж а н и е

I	Предисловие	2
II	Инструкции	3
III	Введение в упражнения	5
IV	Упражнение Модели 00 :	6
V	Упражнение Генератор идентификаторов 01 :	8
VI	Упражнение Список пользователей 02 :	9
VII	Упражнение Список сделок 03 :	10
VIII	Упражнение Логика бизнеса 04 :	11
IX	Упражнение Меню 05 :	12

Г л а в а I

П р е д и с л

о в и е

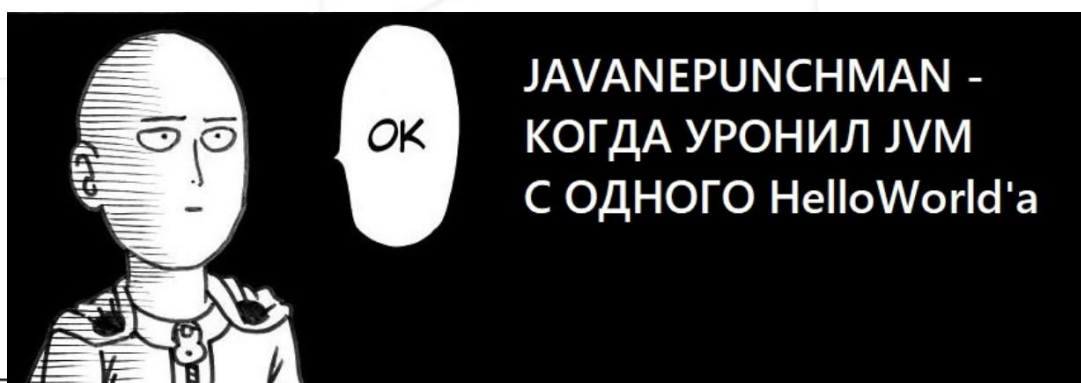
Моделирование домена является наиболее сложной задачей при разработке программного обеспечения. Правильное решение этой задачи обеспечивает гибкость внедряемой системы.

Языки программирования, поддерживающие концепцию объектно-ориентированного программирования (ООП), позволяют эффективно разделять бизнес-процессы на логические компоненты, называемые классами.

Каждый класс должен соответствовать принципам SOLID:

- Принцип единой ответственности: класс содержит единственную логически связанную функцию (кофеварка не может очищать и отслеживать изменения в стеке вызовов; ее назначение - готовить кофе).
- Принцип открытости-закрытости: каждый класс может предлагать возможность 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), которые описывают логически связанное поведение. Не существует интерфейса общего назначения.
- Принцип инверсии зависимостей: система не должна зависеть от конкретных сущностей; все зависимости основаны на абстракциях (интерфейсах).

Сегодня вам следует сосредоточиться на первом принципе SOLID.



Г л а в а II

И н с т р у к

Ц и и

- Используйте эту страницу как единственную ссылку. Не слушайте никаких слухов и домыслов о том, как приготовить раствор.
- Теперь для вас существует только одна версия Java - 1.8. Убедитесь, что компилятор и интерпретатор этой версии установлены на вашей машине.
- Вы можете использовать IDE для написания и отладки исходного кода.
- Код чаще читают, чем пишут. Внимательно прочитайте [документ](#), в котором приведены правила форматирования кода. При выполнении каждого задания убедитесь, что вы следуете общепринятым [стандартам Oracle](#)
- Комментарии не допускаются в исходном коде вашего решения. Они затрудняют чтение кода.
- Обратите внимание на разрешения ваших файлов и каталогов.
- Чтобы ваше решение было оценено, оно должно находиться в вашем GIT-репозитории.
- Ваши решения будут оценивать ваши товарищи по аквариуму.
- Вы не должны оставлять в своем каталоге никаких других файлов, кроме тех, которые явно указаны в инструкциях к упражнению. Рекомендуется изменить свой .gitignore во избежание несчастных случаев.
- Когда вам нужно получить точный вывод в ваших программах, запрещено выводить предварительно рассчитанный вывод вместо правильного выполнения упражнения.
- У вас есть вопрос? Спросите своего соседа справа. В противном случае попробуйте поговорить с соседом слева.
- Ваше справочное пособие: товарищи / Интернет / Google. И еще кое-что. На любой ваш вопрос есть ответ на Stackoverflow. Научитесь правильно задавать вопросы.
- Внимательно прочитайте примеры. В них могут потребоваться вещи, которые не указаны в предмете.
- Используйте "System.out" для вывода

- И да пребудет с вами Сила!
- Никогда не оставляйте на завтра то, что вы можете сделать сегодня ;)

Г л а в а 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. Мы должны быть уверены, 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 \ \$500 Майку. Система сохраняет транзакцию для обоих пользователей:

Джон -> Майк, -500, OUTCOME, ID транзакции

Майк -> Джон, +500, INCOME, ID транзакции


Для восстановления связи внутри таких пар следует использовать идентификаторы каждой транзакции.

В такой сложной системе запись о переводе, очевидно, может быть потеряна - она может быть не записана для одного из пользователей (чтобы эмулировать и отладить такую ситуацию, разработчику необходимо иметь возможность удалить данные о переводе у одного из пользователей по отдельности). Поскольку такие ситуации вполне реальны, необходима функциональность для отображения всех "непризнанных переводов" (транзакций, записанных только для одного пользователя) и решения подобных проблем.

Ниже приведен комплекс упражнений, которые вы можете выполнять по очереди, чтобы решить эту задачу.

Г л а в а IV

Упражнение 00: Модели

	Упражнени е 00
	Модели
Входящий каталог : <i>ex00/</i>	
Файлы для сдачи : User.java, Transaction.java, Program.java	
Разрешенные функции : Могут быть задействованы классы пользователей, а также: Типы (+ все методы этих типов) : Integer, String, UUID, перечисления	

Ваша первая задача - разработать базовые модели домена, а именно классы User и Transaction. Вполне вероятно, что в системе разные пользователи будут иметь одинаковые имена. Эта проблема должна быть решена путем добавления специального поля для уникального ID пользователя. Этот идентификатор может быть любым целым числом. Конкретная логика создания ID описана в следующем упражнении.

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

- Идентификатор
- Имя
- Баланс

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


- Идентификатор
- Получатель (тип пользователя)
- Отправитель (тип пользователя)
- Категория перевода (дебет, кредит)
- Сумма перевода

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 Program (создание, инициализация, вывод содержимого объекта на консоль). Все данные для полей класса должны быть жестко закодированы в Program.

Г л а в а V

Упражнение 01: Генератор идентификаторов

	Упражнение 01
	Генератор идентификаторов
	Входящий каталог : <i>ex01/</i>
	Файлы для сдачи : <i>UserIdsGenerator.java, User.java, Program.java</i>
	Разрешенные функции : Можно использовать все разрешения из предыдущего упражнения

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, когда каждый новый ID является значением ранее сгенерированного ID +1.

Таким образом, класс `UserIdsGenerator` содержит последний сгенерированный ID в качестве своего состояния. Поведение `UserIdsGenerator` определяется методом `int generateId()`, который при каждом вызове возвращает только что сгенерированный ID.

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


Примечания:

- Убедитесь, что существует только один объект `UserIdsGenerator` (см. паттерн Singleton). Это необходимо, поскольку существование нескольких объектов этого класса не может гарантировать, что все идентификаторы пользователей уникальны.
- Идентификатор пользователя должен быть только для чтения, так как он инициализируется только один раз (при создании объекта) и не может быть изменен позже во время выполнения программы.
- Временная логика для инициализации идентификатора должна быть добавлена в конструктор класса `User`:

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


Г л а в а VI

Упражнение 02 : Список пользователей

	Упражнение 02
	Список пользователей
	Входящий каталог : <i>ex02/</i>
	Файлы для сдачи : <i>UsersList.java, UsersArrayList.java, User.java, Program.java</i> и т.д.
	Разрешенные функции : Можно использовать все разрешения из предыдущего упражнения + бросок.

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

На данный момент у вашего приложения нет постоянного хранилища (такого как файловая система или база данных). Однако мы хотим избежать зависимости вашей логики от способа реализации хранения пользователей. Чтобы обеспечить большую гибкость, давайте определим интерфейс `UsersList`, который описывает следующее поведение:

- Добавить пользователя
- Извлечение пользователя по идентификатору
- Извлечение пользователя по индексу
- Получение количества пользователей

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

Мы также реализуем класс `UsersArrayList`, который реализует интерфейс `UsersList`. Этот класс должен использовать массив для хранения пользовательских данных. Размер массива по умолчанию равен 10. Если массив заполнен, его размер увеличивается наполовину. Метод добавления пользователя помещает объект типа `User` в первую пустую (свободную) ячейку массива.

В случае попытки получить пользователя с несуществующим ID, должно быть выброшено непроверенное исключение `UserNotFoundException`.

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


Примечание:

Java-класс `Nested ArrayList<T>` имеет такую же структуру. Самостоятельно

моделируя поведение этого класса, вы научитесь использовать механизмы этого класса стандартной библиотеки.

Г л а в а VII

Упражнение 03: Список транзакций

	Упражнение 03
	Список сделок
Входящий каталог : <i>ex03/</i>	
Файлы для сдачи : TransactionsList.java, TransactionsLinkedList.java, User.java, Program.java и т.д.	
Разрешенные функции : Можно использовать все разрешения из предыдущего упражнения	

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

В этом задании мы предлагаем вам создать интерфейс TransactionsListinterface, описывающий следующее поведение:

- Добавить транзакцию
- Удалить транзакцию по идентификатору (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.

Если предпринимается попытка удалить транзакцию с несуществующим ID, должно быть выброшено исключение времени выполнения TransactionNotFoundException.


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

Примечание:

- Нам нужно добавить поле транзакций типа 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 - двунаправленный связанный список.

Г л а в а VIII

Упражнение 04: Бизнес-логика

	Упражнение 04
	Логика бизнеса
	Входящий каталог : <i>ex04/</i>
	Файлы для сдачи : <i>TransactionService.java, Program.java</i> и т.д.
	Разрешенные функции : Можно использовать все разрешения из предыдущего упражнения

Уровень бизнес-логики приложения расположен в классах обслуживания. Такие классы содержат основные алгоритмы системы, автоматизированные процессы и т.д. Эти классы обычно проектируются на основе паттерна Facade, который может инкапсулировать поведение нескольких классов. В данном случае класс *TransactionService* должен содержать поле типа *UsersList* для взаимодействия с пользователями и обеспечивать следующую функциональность:

- Добавление пользователя
- Получение баланса пользователя
- Выполнение транзакции перевода (указаны идентификаторы пользователей и сумма перевода). В этом случае создаются две транзакции типа DEBIT/CREDIT, которые добавляются получателю и отправителю. Идентификаторы обеих транзакций должны быть одинаковыми
- Получение переводов определенного пользователя (возвращается ARRAY переводов). Повторное перемещение транзакции по ID для конкретного пользователя (указывается ID транзакции и ID пользователя)
- Проверка достоверности транзакций (возвращает ARRAY непарных транзакций).


В случае попытки осуществить перевод суммы, превышающей остаточный баланс пользователя, должно быть выброшено исключение *IllegalTransactionException runtime exception*.

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

Г л а в а IX

У п р а ж н е н

и е 05 : М е н ю

	Упражнение 05
	Меню
	Входящий каталог : <i>ex05/</i>
	Файлы для сдачи : Menu.java, Program.java и т.д.
	Разрешенные функции : Можно использовать все разрешения из предыдущего упражнения, а также try/catch

- В результате вы должны создать функционирующее приложение с консолью
- меню. Функциональность меню должна быть реализована в соответствующем классе с полем ссылки на TransactionsService.
- Каждый пункт меню должен сопровождаться номером команды, введенной пользователем для вызова действия.
- Приложение должно поддерживать два режима запуска - производственный (стандартный режим) и dev (в котором информация о переводе для конкретного пользователя может быть удалена по идентификатору пользователя, и может быть запущена функция, проверяющая достоверность всех переводов).
- Если возникает исключение, появляется сообщение, содержащее информацию об ошибке, и пользователю предоставляется возможность ввести правильные данные.
- Сценарий работы приложения выглядит следующим образом (программа должна тщательно следовать этому примеру вывода):

```
$ java Program --profile =dev
```

```
1. Добавить пользователя  
2. Просмотр балансов пользователей  
3. Выполнить перевод  
4. Просмотр всех транзакций для определенного пользователя  
5. DEV - удалить перевод по идентификатору  
6. DEV - проверка действительности перевода  
7. Завершить выполнение  
-> 1  
Введите имя пользователя и баланс  
-> Jonh 777
```

Пользователь с id = 1 добавлен ----- 1. Добавить пользователя 2. Просмотр балансов пользователей 3. Выполнить перевод 4. Просмотр всех транзакций для определенного пользователя 5. DEV - удалить перевод по идентификатору 6. DEV - проверка действительности перевода 7. Завершить выполнение -> 1 Введите имя пользователя и баланс -> Майк 100 Пользователь с id = 2 добавлен -----
1. Добавить пользователя 2. Просмотр балансов пользователей 3. Выполнить перевод 4. Просмотр всех транзакций для определенного пользователя 5. DEV - удалить перевод по идентификатору 6. DEV - проверка действительности перевода 7. Завершить выполнение -> 3 Введите идентификатор отправителя, идентификатор получателя и сумму перевода -> 1 2 100 Перевод завершен -----
1. Добавить пользователя 2. Просмотр балансов пользователей 3. Выполнить перевод 4. Просмотр всех транзакций для определенного пользователя 5. DEV - удалить перевод по идентификатору 6. DEV - проверка действительности перевода 7. Завершить выполнение -> 3 Введите идентификатор отправителя, идентификатор получателя и сумму перевода -> 1 2 150 Перевод завершен -----
1. Добавить пользователя 2. Просмотр балансов пользователей 3. Выполнить перевод 4. Просмотр всех транзакций для определенного пользователя 5. DEV - удалить перевод по идентификатору 6. DEV - проверка действительности перевода 7. Завершить выполнение -> 3 Введите идентификатор отправителя, идентификатор получателя и сумму перевода -> 1 2 50 Перевод завершен -----
1. Добавить пользователя 2. Просмотр балансов пользователей 3. Выполнить перевод 4. Просмотр всех транзакций для определенного пользователя 5. DEV - удалить перевод по идентификатору 6. DEV - проверка действительности перевода 7. Завершить выполнение -> 2 Введите идентификатор пользователя -> 2 Майк - 400 -----
1. Добавить пользователя 2. Просмотр балансов пользователей 3. Выполнить перевод 4. Просмотр всех транзакций для определенного пользователя 5. DEV - удалить перевод по идентификатору 6. DEV - проверка действительности перевода 7. Завершить выполнение -> 4 Введите идентификатор пользователя -> 1 Майку(id = 2) -100 с id = cc128842-2e5c-4cca-a44c-7829f53fc31f


```
Майку(id = 2) -150 c id = 1fc852e7-914f-4bfd-913d-0313aab1ed99  
TO Mike(id = 2) -50 c id = ce183f49-5be9-4513-bd05-8bd82214eaba
```

- ```

1. Добавить пользователя
2. Просмотр балансов пользователей
3. Выполнить перевод
4. Просмотр всех транзакций для определенного пользователя
5. DEV - удалить перевод по идентификатору
6. DEV - проверка действительности перевода
7. Завершить выполнение
-> 5
```

```
Введите идентификатор пользователя и идентификатор перевода
-> 1 1fc852e7-914f-4bfd-913d-0313aab1ed99
Передача Майку (id = 2) 150 удалено
```

- ```
-----  
1. Добавить пользователя  
2. Просмотр балансов пользователей  
3. Выполнить перевод  
4. Просмотр всех транзакций для определенного пользователя  
5. DEV - удалить перевод по идентификатору  
6. DEV - проверка действительности перевода  
7. Завершить выполнение  
-> 6
```

```
Проверьте результаты:  
Mike(id = 2) имеет непризнанный перевод id = 1fc852e7-914f-4bfd-913d-0313aab1ed99 от John(id = 1) за 150
```