файл с лекциями

Задачи по курсу «Основы языка программирования Java»

Раздел I: Основы Java

1. Реализовать метод, проверяющий, является ли заданное число по абсолютной величине степенью двойки.
пример ввода: 3
пример вывода: false

2. Написать программу, выводящую на экран первые 12 членов последовательности Фибоначчи.
(Указание: первые два члена последовательности Фибоначчи равны, соответственно, 0 и 1; последующие – сумма двух предыдущих)

3. Реализовать метод, принимающий массив целых чисел и выводящий в консоль через запятую только нечетные числа из него.

4. Реализовать метод, проверяющий, является ли заданная строка палиндромом. При определении "палиндромности" строки должны учитываться только буквы и цифры (пробелы, знаки препинания, регистр символов игнорируются). Метод должен принимать только строки, состоящие из цифр, латинских букв и знаков препинания.
(Допускается использование регулярного выражения).

пример ввода: Do geese see God?
пример вывода: true

5. Написать программу, считывающую введённые пользователем n строк, и определяющую самую короткую и самую длинную строки. Вывести найденные строки и их длину на экран.

6. Дана произвольная строка, состоящая из слов, разделенных пробелами (например: "One two three four five один два три четыре one1 two2 123 567"). Написать программу, считывающую такую введённые строку и определяющую количество слов, содержащих только символы латинского алфавита.

7. Написать программу, определяющую, является ли введённая пользователем строка корректным IP адресом, записанным в десятичном виде.
(Примеры корректного IP адреса: 127.0.0.1, 255.255.255.0. Примеры некорректного IP адреса: 1234.1.2.3, abc.abc.abc.def.)

8. Создать класс, описывающий вектор в трехмерном пространстве.
Этот класс должен содержать:
    1) Конструктор с тремя параметрами, соответствующими координатам x, y, z.
    2) Методы:
        - вычисления длины вектора;
        - вычисления скалярного произведения двух векторов;
        - вычисления векторного произведения двух векторов;
        - вычисления суммы и разности двух векторов.

9. Создать класс Student и его подкласс PostGraduateStudent. Класс Student содержит:
    - поля: имя, фамилию, среднюю оценку (double);
    - метод getScholarship(), возвращающий размер стипендии.

Для студента размер стипендии составляет 80, если средняя оценка равна 5. В противном случае размер стипендии составляет 40. Переопределить метод getScholarship() для класса PostGraduateStudent. Аспирантская стипендия – 150 при средней оценке, равной 5. Иначе – 100.

10. Реализовать абстрактные классы и интерфейсы, а также наследование и полиморфизм для следующих классов:
interface Транспортное Средство ← *abstract class* Общественный Транспорт ← *class* Трамвай.

11. Создать собственное проверяемое исключение и метод, выбрасывающий его.

12. Реализовать метод, позволяющий другим методам узнать, откуда их вызвали.
Этот метод должен возвращать имя класса и метода, откуда вызван метод, вызвавший данный метод. Или null, если метод, вызвавший этот метод, является точкой входа в программу.

13. Создать собственный класс, объекты которого можно использовать в блоке try-with-resources. Продемонстрировать использование объектов этого класса в try-with-resources.

14. Написать программу, считывающую введённые пользователем с клавиатуры 5 строк и записывающую их в список ArrayList. Найти самую короткую строку в списке и вывести её на экран. Если таких строк несколько, вывести на экран каждую с новой строки.

15. Создать класс Student, содержащий следующие характеристики – имя, группа, курс, оценки по предметам.
Создать коллекцию, содержащую объекты класса Student.
Написать метод, который удаляет студентов со средним баллом < 3.
Если средний балл >= 3, студент переводится на следующий курс.

Реализовать метод printStudents(List<Student> students, int course), который получает список студентов и номер курса и выводит в консоль имена тех студентов из списка, которые обучаются на данном курсе.

16. Реализовать метод, который на вход получает коллекцию объектов, а возвращает коллекцию без дубликатов. Порядок элементов исходной коллекции должен сохраняться.

17. Дан входной файл, содержащий совокупность строк. Каждая строка файла соответствует строке квадратной матрицы. Написать программу, вводящую эту матрицу в двумерный массив. Вывести исходную и транспонированную ей матрицы.
(*Указание: для работы с файлом использовать новый API из пакета java.nio.file*)

18. Написать метод, добавляющий 500 тыс. элементов в ArrayList и LinkedList. Написать еще один метод, выбирающий из каждого из заполненных списков элемент наугад 100 тыс. раз. Замерить время, которое потрачено на это. Сравнить результаты.

19. Написать программу, считывающую из консоли последовательность целых чисел, разделенных пробелами, удаляющую из них все числа, стоящие на четных позициях, и выводящую получившуюся последовательность в обратном порядке.
Позиции чисел в последовательности нумеруются с нуля.
(**Требование: в решении использовать очередь**)

Пример ввода: 1 2 3 4 5 6 7 8 9 10
Пример вывода: 10 8 6 4 2

20. Написать метод, возвращающий реализацию функционального интерфейса UnaryOperator, который принимает целое число (тип int) и возвращает его квадрат.

21. Написать метод, позволяющий убрать из заданного массива произвольного типа лишние элементы.
Продемонстрировать работу этого метода на массивах различных типов.
(*Указание: воспользоваться функциональным интерфейсом Function<T, R>*)

22. Написать метод, принимающий массив элементов типа *T* и возвращающий *Map<T, Integer>*, где ключ — элемент из массива, а значение — количество вхождений этого элемента в массив.

23. Написать метод, принимающий *Map<T, R>* и возвращающий *Map*, где ключи и значения поменялись местами. Учесть случай, когда в исходном *Map* разным ключам соответствуют одинаковые значения.

24. Создать класс User, описывающий пользователя (необходимые поля: id, firstName, lastName, age, country). Создать список из некоторого количества пользователей.

1) Вывести всех пользователей, отсортированных по lastName
2) Вывести всех пользователей, отсортированных по age
3) Проверить, что для всех пользователей age > 7
4) Вычислить средний возраст всех пользователей
5) Вывести количество разных стран проживания (country) среди заданных пользователей

(Требование: реализовать все пункты с использованием Stream API
*Указание: можно использовать аннотации Lombok***)**

25. Написать программу, выводящую 10 (или сколько есть) наиболее часто встречающихся слов (без учета регистра) из текста, введённого в консоль. Словом считается любая непрерывная последовательность букв и цифр.
Если в тексте некоторые слова имеют одинаковую частоту, то дополнительно упорядочить их в лексикографическом порядке.

Пример ввода 1:

Мама мыла-мыла-мыла раму!

Вывод:

мыла
мама
раму

Пример ввода 2:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sodales consectetur purus at faucibus. Donec mi quam, tempor vel ipsum non, faucibus suscipit massa. Morbi lacinia velit blandit tincidunt efficitur. Vestibulum eget metus imperdiet sapien laoreet faucibus. Nunc eget vehicula mauris, ac auctor lorem. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer vel odio nec mi tempor dignissim.

Вывод:

consectetur
faucibus
ipsum
lorem
adipiscing
amet
dolor
eget
elit
mi

(**Требование: решить с использованием Stream API**
Указание: Не использовать циклы и условные операторы, решить, не разрывая цепочку методов стрима)

Раздел II: JDBC, Hibernate, Spring Framework

Общие рекомендации перед решением задач из этого раздела:

1. Использовать IDE для написания кода (желательно - Intellij IDEA Ultimate)
2. Ознакомиться с требованиями к code-style и следовать им при написании кода
3. Ознакомиться с основами работы с Apache Maven (подключение Maven к проекту, pom.xml - структура файла и работа с зависимостями)
4. Ознакомиться с основами работы с JDBC.
5. Ознакомиться с основами работы с Hibernate
6. Во всех задачах можно использовать аннотации Lombok для сокращения количества строк кода

---

26. В базе данных хранится информация о преподавателях и проводимых ими занятиях.
Для предметов необходимо хранить:
– название;
– день недели проведения;
– аудитории, в которых проводятся занятия.
Для преподавателей необходимо хранить:
– ФИО;
– преподаваемые предметы;
– количество пар в неделю по каждому предмету;
– количество студентов, занимающихся на каждой паре.
Написать программу, выводящую следующую информацию:

- информацию о преподавателях, работающих в заданный день недели в заданной аудитории;
- информацию о преподавателях, которые не ведут занятия в заданный день недели;
- дни недели, в которых проводится заданное количество занятий.

Организацию соединения с базой данных вынести в отдельный класс, метод которого возвращает соединение. Создать БД. Создать класс для выполнения запросов на извлечение информации из БД.
(Рекомендация: использовать PostgreSQL).

26. (In English) In our database we store information about lecturers and their subjects.
For subjects you should store the following:
– name;
– day of week;
– classrooms in which the lectures take place.
For lecturers you should store:
– first and last name;
– their subjects;
– number of lectures per week for each subject;
– number of students attending each lecture.
It is necessary to write the code printing out the following information:

- information about the lecturers working in a given day in a given classroom;
- information about the lecturers that do not have any lectures in a given day;
- days of week that hold a given number of lectures.

Connection to the database should be organized in a separate class, and the method of this class should return a connection. Create your own database and a class that will execute queries for obtaining information from DB.
(It is recommended to use PostgreSQL).

---

27. Клонировать/скачать заготовку проекта по ссылке. Установить на компьютер PostgreSQL и pgAdmin (опционально). Создать подключение и схему. Протестировать возможность соединения с базой через IDEA при помощи встроенной утилиты (View -> Tool Windows -> Database -> …).
Ознакомиться с заготовкой проекта и доработать приложение, которое взаимодействует с базой, оперируя пользователем (класс User) и проверить свои методы готовыми JUnit тестами. Класс UserDaoHibernateImpl в этой задаче не использовать.
User представляет из себя сущность с полями:

- Long id
- String name
- String lastName
- Byte age

Требования к классам приложения:

1. Классы dao/service должны реализовывать соответствующие интерфейсы
2. Класс dao должен иметь конструктор пустой/по умолчанию
3. Все поля должны быть private
4. service переиспользует методы dao
5. Обработка всех исключений, связанных с работой с базой данных, должна находиться в dao
6. Класс Util должен содержать логику настройки соединения с базой данных

Необходимые операции:

1. Создание таблицы для User не должно приводить к исключению, если такая таблица уже существует
2. Удаление таблицы User не должно приводить к исключению, если таблицы не существует
3. Очистка содержания таблицы
4. Добавление User в таблицу
5. Удаление User из таблицы (по id)
6. Получение всех User из таблицы

Алгоритм работы приложения:

В методе main класса Main должны происходить следующие операции:

1. Создание таблицы User(ов)
2. Добавление 4 User(ов) в таблицу с данными на свой выбор. После каждого добавления должен быть вывод в консоль (User с именем – name добавлен в базу данных)
3. Получение всех User из базы и вывод в консоль (должен быть переопределен toString в классе User)

4. Очистка таблицы User
5. Удаление таблицы

27. (In English) You should clone/download the draft of the project here. Installing PostgreSQL and pgAdmin on your PC is optional. Establish a connection and create a scheme. You can test the connection to your DB by means of IDEA's inbuilt utility (View -> Tool Windows -> Database -> …).
You should get familiar with the draft and finalize the application that is going to interact with the DB with the help of a user (class User). You should also check your methods with the already existing JUnit tests. The UserDaoHibernateImpl class should not be used in this task.
User must contain the following fields:

- Long id
- String name
- String lastName
- Byte age

Requirements for application classes:

7. The dao/service classes must implement the corresponding interfaces
8. The dao class must have a default (no-args) constructor
9. All fields must be private
10. service reuses the dao methods
11. Handling of all exceptions related to the database must lie in the dao layer
12. The Util class must contain all the settings necessary to obtain a DB connection

Required operations:

7. Creating the User's table must not lead to exception in the case the table already exists
8. Dropping the User's table must not lead to exception in the case the table does not exist
9. Clearing the table
10. Adding User to the table
11. Deleting User from the table (by it's id)
12. Getting all Users from the table

Algorithm of the application is as follows:

The main method of the Main class must contain the following operations:

6. Creating User's table
7. Adding 4 arbitrary Users to the table. Each addition should be accompanied with the console printout (User with the name – *name* was added to the database)
8. Getting all Users and the corresponding console output (the *toString* method of the User class must be overridden)
9. Clearing the User's table
10. Dropping the User's table

(необязательные задачи)

28. Должна быть готова задача № 27. Необходимо реализовать взаимодействие с базой данных с помощью Hibernate и дописать методы в классе UserDaoHibernateImpl , проверить свои методы готовыми JUnit тестами.

Требования к классам приложения:

1. UserDaoHibernateImpl должен реализовывать интерфейс UserDao
2. В класс Util должна быть добавлена конфигурация для Hibernate (рядом с JDBC), без использования xml.
3. Service использует реализацию dao через Hibernate
4. Методы создания и удаления таблицы пользователей в классе UserDaoHibernateImpl должны быть реализованы с помощью SQL (остальные методы - средствами Hibernate).

Алгоритм приложения и операции такие же, как и в задаче 27.

(**Требования:**
1. **Учесть транзакции (+ откат транзакции)**
2. **Использовать аннотации Lombok в классе User)**

28. (In English) Task 27 should be done. Now it is necessary to organize a connection with the database by means of Hibernate and create the corresponding methods in the UserDaoHibernateImpl class. You should also check your methods with the already existing JUnit tests.

Requirements for application classes:

5. UserDaoHibernateImpl must implement the UserDao interface
6. You must add the Hibernate configuration to the Util class (next to the JDBC configuration), without xml.
7. Service uses the dao implementation through Hibernate
8. Methods for creating and dropping the user's table must use plain SQL (i.e., with *createNativeQuery()* method)

Algorithm of the application and required operations are the same as in task 27.

(**Requirements:**
1. **Take transactions into account (+ transaction rollback)**
2. **Use Lombok annotations in the User class)**

29. Клонировать/скачать заготовку проекта по ссылке. Требуется написать CRUD web-приложение. Приложение должно запускаться и управляться с помощью сервера *Tomcat*.

Требования к приложению:

1. Использовать *Spring MVC + Hibernate*.
2. Должен быть класс *User* с произвольными полями (id, name, …).

3. В приложении должна быть страница, на которую выводятся все пользователи с возможностью добавлять, удалять и изменять пользователя (для работы с данными пользователя на стороне браузера использовать средства *thymeleaf*).
4. Spring должен быть сконфигурирован через JavaConfig и аннотации, **без использования xml и Spring Boot**.
5. Создать конфигурацию для работы с базой данных. Вместо *SessionFactory* должен использоваться *EntityManager*.
6. Структура приложения должна быть аналогична структуре из задач 27-28, т.е. должны присутствовать: слой DAO (*UserDao*, *UserDaoImpl*), модель *User*, сервис (*UserService*, *UserServiceImpl*). Плюс контроллер; настройки для работы с БД и с Hibernate вынести в файл *.properties* (/main/resources/ - директория “Resources Root”).

(Тестовый запуск заготовки проекта: скачать сервер Tomcat, подключить к проекту, при запуске приложения на странице *http://localhost:8080/* должны выводиться сообщения из метода *printWelcome()* класса *HelloController*. После проверки работоспособности класс *HelloController* можно удалить; написать собственный контроллер, соответствующий требованиям задачи.)
(**Требования:**

1. **Учесть транзакции**
2. **Приложение должно корректно работать при вводе данных как на латинице, так и на кириллице)**

---

30. Перенести приложение из задачи 29 на *Spring Boot*.

Список вопросов к зачету

1. Какая основная идея языка, за счет чего обеспечивается?
2. Преимущества/недостатки java.
3. Что такое JDK/JRE/JVM/JIT?
4. Сборщик мусора, как работает.
5. Heap и Stack памяти в Java, отличия.
6. Инкапсуляция/наследование/полиморфизм (с примерами).
7. Раннее и позднее связывание.
8. Примитивные типы данных (какие существуют, диапазон значений, сколько занимают памяти).
9. Классы-обертки.
10. Автоупаковка и автораспаковка.
11. Явное и неявное приведение типов. В каких случаях в Java нужно использовать явное приведение?
12. Пул интов.
13. Пул строк. Как добавить строку в пул строк?
14. Класс *String*. Нюансы работы со строками в Java.
15. *String*, *StringBuffer*, *StringBuilder* – сходства и различия .
16. Какими значениями инициализируются переменные по умолчанию?
17. Что такое массив и какие на нём есть ограничения? Максимальная длина массива.
18. Виды условных операторов.
19. Конструкция *switch-case*; какие типы данных в ней можно использовать.
20. Виды циклов, их отличия.
21. Перегрузка / переопределение методов.
22. Какие модификаторы доступа могут быть у класса?
23. Что такое ключевое слово *final*? Что может быть *final*?
24. Как реализована неизменность *String*?
25. Как реализовать свой Immutable тип данных.
26. Что такое ключевое слово *static*? Что может быть *static*?
27. Могут ли нестатические методы перегрузить статические?
28. Что такое конструктор? Конструктор по умолчанию?
29. Ключевые слова *this* и *super*, когда нужно использовать?
30. Класс *Object* и его методы.
31. Контракт между методами *equals* и *hashCode*.
32. Что такое коллизия? Из-за чего происходит?
33. Перечисления. Методы *enum*.
34. Что такое интерфейс и когда нужно использовать?
35. Можно ли создать поля / методы в интерфейсе?
36. Может ли интерфейс наследовать другой интерфейс?
37. Абстрактный класс, отличие от обычного.
38. Что такое исключения и для чего нужны?
39. Иерархия исключений.

40. Проверяемые и непроверяемые исключения. Разница между ними (в синтаксисе и при использовании).
41. Можно ли обработать непроверяемое исключение?
42. Нужно ли ловить *Error* исключения?
43. Как выбросить исключение?
44. Какая информация находится внутри исключения?
45. Подавленное исключение. Как его достать?
46. Какую информацию можно получить из *StackTraceElement*?
47. Рассказать, как работает конструкция *try-catch-finally*.
48. Рассказать, как работает конструкция *try-with-resources*.
49. Как правильно ловить исключения (иерархия *catch* блоков)?
50. Можно ли в одном *catch* обработать несколько исключений?
51. Что такое *generic* типы и для чего они нужны?
52. Что можно / нельзя параметризовать?
53. Как параметризовать статический метод?
54. Что такое *raw type*? К чему приводит использование *raw type*?
55. Если поле типизировано дженериком, как в байт коде будет представлен этот тип?
56. Что такое diamond-оператор?
57. Что такое *wildcard*?
58. Иерархия коллекций.
59. *List* и *Set*, их отличия.
60. *ArrayList* и *LinkedList*, их отличия. Когда лучше использовать *ArrayList*, а когда *LinkedList*?
61. *Queue*, *Deque* и *Stack* - в чем разница?
62. Что такое *HashSet*?
63. Что такое *Iterator*? В чём разница между *Iterable* и *Iterator*?
64. Что такое *Map*? Может ли *null* быть ключём в *HashMap*?
65. Что такое *HashMap*?
66. Что такое функциональный интерфейс? Зачем нужна аннотация *@FunctionalInterface* и обязательна ли она?
67. Что такое *default* методы в интерфейсе и для чего они были введены?
68. Может ли функциональный интерфейс содержать что-то кроме абстрактного метода?
69. Перечислить способы реализации функционального интерфейса.
70. Что такое анонимный класс и как создать экземпляр такого класса?
71. Что такое лямбда-выражение? Как его записать?
72. Что такое стримы? Для чего они нужны? Виды стримов.
73. Перечислить способы создания стримов.
74. Что возвращают промежуточные операции над стримом? Что такое терминальная операция? Можно ли вызвать 2 терминальные операции?
75. Что такое JDBC? Какие классы / интерфейсы относятся к JDBC (рассказать про них)?
76. Паттерн DAO.

(Далее - необязательная часть)

77. Что такое *JPA*?
78. Что такое *ORM*?
79. Что такое *Hibernate*? В чем разница между *JPA* и *Hibernate*?
80. Какие классы/интерфейсы относятся к *JPA/Hibernate*?
81. Основные аннотации *Hibernate*.
82. Чем *HQL* отличается от *SQL*?
83. Что такое *Query*? Как передать в объект *Query* параметры?
84. Что такое бин? Виды бинов.
85. Чем бин отличается от POJO-класса?
86. Что такое Inversion of Control и как *Spring* реализует этот принцип?
87. Как можно связать бины?
88. Что такое Dependency Injection?
89. Как получить данные из файла *.property*?
90. Какая разница между аннотациями *@Component*, *@Repository* и *@Service* в *Spring*?
91. Как выглядит структура MVC-приложения?
92. Что такое контроллер? Как вернуть страницу в контроллере? Как вернуть данные?
93. Жизненный цикл бина.
94. *Spring Boot*. Его преимущества / недостатки.