Занятие 3

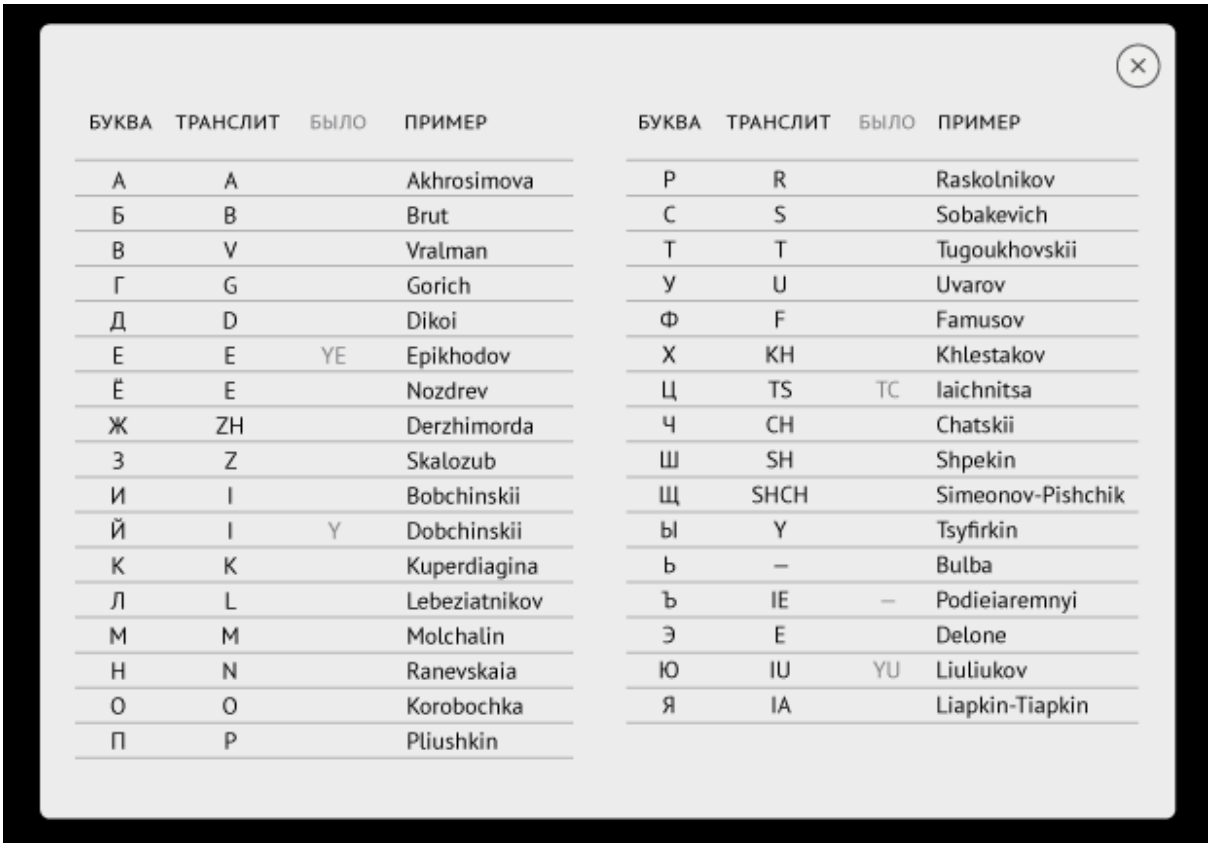
Задания

Самостоятельное изучение

- Изучить методы интерфейсов java.util.List, java.util.Set, java.util.Map
- Изучить альтернативные интерфейсы коллекций: SortedMap, Queue, Stack…
- Изучить, чем ArrayList отличается от LinkedList, в каких ситуациях какая реализация более предпочтительна https://habr.com/ru/post/162017/
- Изучить, как работает HashMap. Разобрать понятие бакета, связь функций equals() и hashCode()
- Повторить тему исключений. Исключения java (java exception): обработка try, catch, throws, finally с примерами
- Изучить основы Java IO: понятия потоков, приемы работы с файлами Потоки ввода-вывода - Java
- Изучить способы чтения файла с помощью библиотеки Java NIO

1. Transliterator

Правила транслитерации приведены в таблице ниже (колонки БУКВА и ТРАНСЛИТ):



БУКВА	ТРАНСЛИТ	БЫЛО	ПРИМЕР
А	A		Akhrosimova
Б	B		Brut
В	V		Vralman
Г	G		Gorich
Д	D		Dikoi
Е	E	YE	Epikhodov
Ё	E		Nozdrev
Ж	ZH		Derzhimorda
З	Z		Skalozub
И	I		Bobchinskii
Й	I	Y	Dobchinskii
К	K		Kuperdiagina
Л	L		Lebeziatnikov
М	M		Molchalin
Н	N		Ranevskaia
О	O		Korobochka
П	P		Pliushkin
Р	R		Raskolnikov
С	S		Sobakevich
Т	T		Tugoukhovskii
У	U		Uvarov
Ф	F		Famusov
Х	KH		Khlestakov
Ц	TS	TC	Iaichnitsa
Ч	CH		Chatskii
Ш	SH		Shpekin
Щ	SHCH		Simeonov-Pishchik
Ы	Y		Tsyfirkin
Ь	–		Bulba
Ъ	IE	–	Podieiaremnyi
Э	E		Delone
Ю	IU	YU	Liuliukov
Я	IA		Liapkin-Tiapkin

Реализовать интерфейс Transliterator

```
public interface Transliterator {
 String transliterate(String source);
}
```

Метод transliterate должен выполнять транслитерацию входной строки в выходную, то есть заменять каждый символ кириллицы на соответствующую группу символов латиницы. Каждый символ кириллицы, имеющийся во входной строке входит в нее в верхнем регистре.

```
public class TransliteratorTest {
 public static void main(String[] args) {
   Transliterator transliterator = new TransliteratorImpl();
   String res = transliterator
                   .transliterate("HELLO! ПРИВЕТ! Go, boy!");
   System.out.println(res);
 }
}

> HELLO! PRIVET! Go, boy!
```

2. DatedMap

DatedMap - это структура данных, очень похожая на Map, но содержащая дополнительную информацию: время добавления каждого ключа. При этом время хранится только для тех ключей, которые присутствуют в Map.

Реализовать DatedMap путем реализации следующего интерфейса

```
public interface DatedMap {
 void put(String key, String value);
 String get(String key);
 boolean containsKey(String key);
 void remove(String key);
 Set<String> keySet();
 Date getKeyLastInsertionDate(String key);
}
```

Методы:

put. Помещает в map пару ключ и значение. Как видно из описания метода, ключ и значение - это строки (семантика Map#put)
get. Возвращает значение, связанное с переданным в метод ключом. Если для переданного ключа значение отсутствует - возвращается null (семантика Map#get)
containsKey. Метод, проверяющий, есть ли в map значение для данного ключа (семантика Map#containsKey).
remove. Получая на вход ключ, удаляет из map ключ и значение, с ним связанное (семантика Map#remove)
ketSet. Возвращает множество ключей, присутствующее в map (семантика Map#keySet)
getKeyLastInsertionDate. Получая на вход ключ, проверяет, что для данного ключа существует значение в map. Если существует - возвращает дату, когда оно было добавлено. Если нет - возвращает null

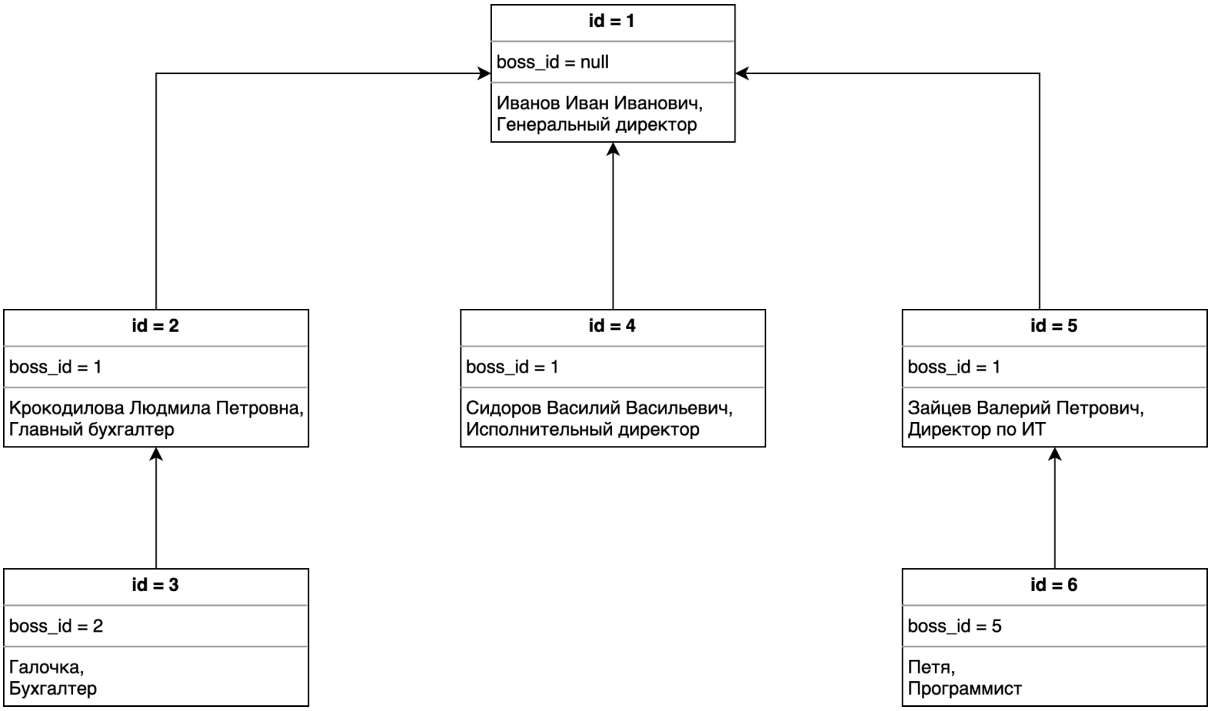
В реализации данного класса можно использовать уже готовые структуры данных в Java, такие как HashMap

3. Org Structure

Структура организации записана в виде строк в CSV файле. CSV - файл - это простой текстовый файл, содержащий строки. Каждая строка представляет собой одну запись (объект). Поля объекта разделены специальным символом **;**. Первая строка файла содержит поля имена полей, все дальнейшие сроки содержат непосредственно данные
Пример:

```
id;boss_id;name;position
1;;Иван Иванович;Генеральный директор
2;1;Крокодилова Людмила Петровна;Главный бухгалтер
3;2;Галочка;Бухгалтер
4;1;Сидоров Василий Васильевич;Исполнительный директор
5;1;Зайцев Валерий Петрович;Директор по ИТ
6;5;Петя;Программист
```

В файле поле **id** обозначает уникальный идентификатор сотрудника, **boss_id** идентификатор начальника, **name** - имя сотрудника, **position** - должность. Таким образом, данные в файле описывают следующую иерархию сотрудников



Необходимо написать программу, получает на вход CSV файл формата, описанного выше и формирует структуру объектов класса

```java
public class Employee {
 private Long id;
 private Long bossId;
 private String name;
 private String position;
 private Employee boss;
 private List<Employee> subordinate = new ArrayList<>();

 public Long getId() {
   return id;
 }
 public void setId(Long id) {
   this.id = id;
 }

 public Long getBossId() {
  return bossId;
 }
 public void setBossId(Long bossId) {
  this.bossId = bossId;
 }

 public String getName() {
   return name;
 }
 public void setName(String name) {
   this.name = name;
 }

 public String getPosition() {
   return position;
 }
 public void setPosition(String position) {
   this.position = position;
 }

 public Employee getBoss() {
   return boss;
 }
 public void setBoss(Employee boss) {
   this.boss = boss;
 }
 public List<Employee> getSubordinate() {
   return subordinate;
 }
}
```

Решение оформить в виде реализации следующего интерфейса

```
public interface OrgStructureParser {

 public Employee parseStructure(File csvFile) throws IOException;

}
```

Метод **parseStructure** должен считывать данные из файла и возвращать ссылку на Босса (Генерального директора) - сотрудника, атрибут boss_id которого не задан. Считать, что такой сотрудник в файле **ровно один**.
P.S. subordinates - список **прямых** подчиненных

4. PasswordValidator

https://www.examclouds.com/ru/java/java-core-russian/lesson17-tasks

1. Создать статический метод, который принимает на вход три параметра: login, password и confirmPassword.
2. Login должен содержать только латинские буквы, цифры и знак подчеркивания. Если login не соответствует - выбросить WrongLoginException с текстом “Логин содержит недопустимые символы”
3. Длина login должна быть меньше 20 символов. Если login не соответствует этим требованиям, необходимо выбросить WrongLoginException с текстом “Логин слишком длинный”
4. Password должен содержать только латинские буквы, цифры и знак подчеркивания. Если password не соответствует этим требованиям, необходимо выбросить WrongPasswordException с текстом “Пароль содержит недопустимые символы”
5. Длина password должна быть меньше 20 символов. Если password не соответствует этим требованиям, необходимо выбросить WrongPasswordException с текстом “Пароль слишком длинный”
6. Также password и confirmPassword должны быть равны. Если password не соответствует этим требованиям, необходимо выбросить WrongPasswordException с текстом “Пароль и подтверждение не совпадают”
7. WrongPasswordException и WrongLoginException - пользовательские классы исключения с двумя конструкторами – один по умолчанию, второй принимает сообщение исключения и передает его в конструктор класса Exception.
8. Обработка исключений проводится внутри метода. Обработка исключений - вывод сообщения об ошибке консоль

9. Метод возвращает true, если значения верны или false в другом случае.

5. FileSort

Даны следующие классы:
Класс **Generator**, которые генерирует файл с заданными количеством чисел типа long

```
public class Generator {

 public File generate(String name, int count) throws IOException {
   Random random = new Random();
   File file = new File(name);
   try (PrintWriter pw = new PrintWriter(file)) {
     for (int i = 0; i < count; i++) {
       pw.println(random.nextLong());
     }
     pw.flush();
   }
   return file;
 }
}
```

Класс **Validator**, который проверяет, что файл отсортирован по возрастанию

```
public class Validator {
 private File file;

 public Validator(File file) {
   this.file = file;
 }

 public boolean isSorted() {
   try (Scanner scanner = new Scanner(new FileInputStream(file))) {
     long prev = Long.MIN_VALUE;
     while (scanner.hasNextLong()) {
       long current = scanner.nextLong();
       if (current < prev) {
         return false;
       } else {
         prev = current;
       }
     }
     return true;
   } catch (IOException ex) {
     ex.printStackTrace();
     return false;
   }
 }
}
```

Класс **Sorter**, который получает на вход файл с числами, и возвращает отсортированный по возрастанию файл

```
public class Sorter {
 public File sortFile(File dataFile) throws IOException {
  //
 }
}
```

И класс **Test**, который запускает генерацию файла, затем сортировку и проверку, что файл отсортирован

```
public class Test {
 public static void main(String[] args) throws IOException {
   File dataFile = new Generator().generate("data.txt", 375_000_000);
   System.out.println(new Validator(dataFile).isSorted()); // false
   File sortedFile = new Sorter().sortFile(dataFile);
   System.out.println(new Validator(sortedFile).isSorted()); // true
 }
}
```

Задача - реализовать метод Sorter.sortFile используя алгоритм внешней сортировки слиянием.
Для демонстрации решения проверяющим можно использовать файл небольшого размера (до 100 элементов), однако, решение долго быть реализовано таким образом, чтобы поддерживать сортировку файлов произвольно большого размера.

Для самопроверки можно сгенерировать файл из 375000000 записей, тогда объем файла, который надо будет отсортировать, будет равен 7-8 Гб