**Потоки ввода-вывода**

**Упраженние-3 – java.util.Map + IO.**

Создайте новое приложение – «avg-person-writer» и реализуйте следующую задачу:

Реализуйте задачу из темы «Map. Ассоциативный массив», добавив поддержку чтения и записи файлов.

Ваше предыдущее упражнение начиналось вот так:

*Пусть в вашей программе задан List, который хранит объекты типа Person:*

***List<Person> personList = new ArrayList<>();***

Теперь условие следующее:

Пусть у вас задан текстовый файл со следующим содержимым:

**Ivan|01-01-1983**

**Ivan|01-01-1993**

**Ivan|01-01-2003**

**Petr|01-01-1963**

**Petr|01-01-1995**

**Sidor|01-01-2000**

Необходимо:

1. Прочитать этот файл
2. Преобразовать каждую строку файла в объект типа Person
3. На основании полученных объектов сформировать **List<Person> personList = new ArrayList<>();**

Далее необходимо выполнить алгоритмику предыдущего задания, т. е. повторить формирование карты одинаковых имен и подсчет среднего возраста.

И завершить задание следующим образом:

Программа должна добавить в файл «**age\_report.txt**» следующую информацию:

1. Заголовок с новым отчетом. Заголовок должен содержать дату печати в файл
2. Уникальные имена
3. Количество людей с этим именем
4. И средний возраст этих людей

То есть каждая запись отчета может выглядеть следующим образом:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Age report for: <текущая дата и время отчета>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Person-1:

Name = Ivan

Count = 3

Avg Age = XXX

-------------------------------------------------------------------------------------------

Person-2:

Name = Petr

Count = 2

Avg Age = XXX

-------------------------------------------------------------------------------------------

Person-3:

Name = Sidor

Count = 1

Avg Age = XXX

-------------------------------------------------------------------------------------------

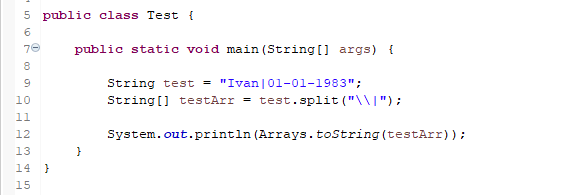
У вас возникнут вопросы.

**Вопрос-1:** У меня есть строка типа «**Ivan|01-01-1983**». Как мне отделить имя от даты?

**Ответ:** Воспользуйтесь методом **split()** класса **String**. Этот метод принимает на вход символ разделителя по которому надо разделить строку.

На выходе возвращает массив строк, где каждая ячейка – это подстрока исходной строки до разделителя.

Например, строка «**Ivan|01-01-1983**» при помощи кода:

****

Вернет вам следующий массив: **[Ivan], [01-01-1983]**.

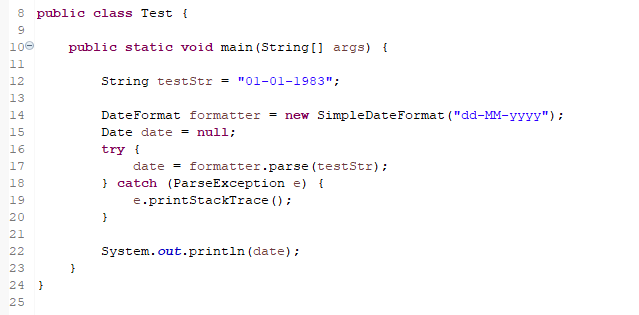
Дальше – дело техники ☺

**Вопрос:** Нет! Дальше – совсем не дело! У меня есть строка типа «**01-01-1983**». Как мне на основании этой строки получить реальный объект типа **java.util.Date**?

**Ответ:** Воспользуйтесь классами **java.text.DateFormat** и **java.text. SimpleDateFormat**.

Эти классы принимают на вход строку и формат даты, и строят на основании принятой строки объект **java.util.Date** согласно заданному формату. Ну или бросают Exception, если вы задали неверный формат.

Например, чтобы построить из строки «01-01-1983» объект **java.util.Date**, воспользуйтесь следующим кодом:

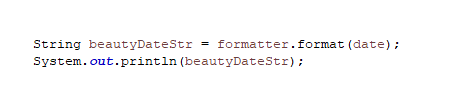


Результат на экране будет следующим: **Sat Jan 01 00:00:00 MSK 1983**

То есть, мы видим, что это действительно – 1-ое января 1983 года.

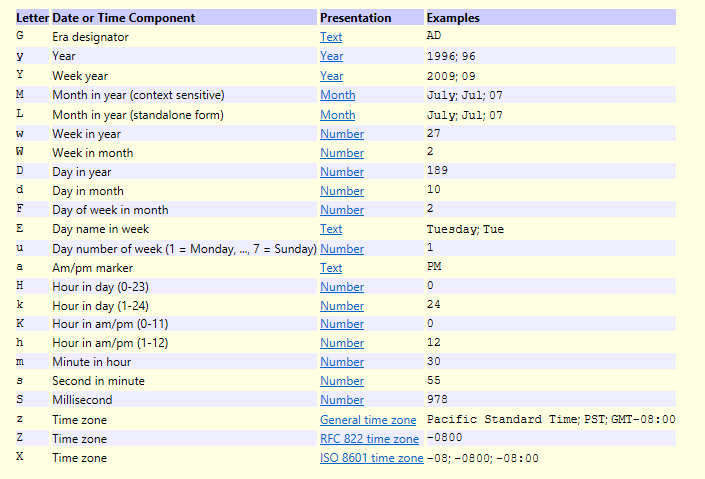
Также классы **java.text.DateFormat** и **java.text. SimpleDateFormat** поддерживают и обратный процесс – преобразование объекта типа java.util.Date в строку заданного формата.

Например, если вы не хотите видеть на экране этот некрасивый вывод: **Sat Jan 01 00:00:00 MSK 1983**, используйте следующий код:



На экран будет выведено: **01-01-1983**.

Форматы, которые вы можете использовать в классах:



Поиграйтесь с форматом. Получите дату\время в разном виде.