Java: знакомство и как пользоваться базовым API (семинары)

Задание 1. Формирование URL с параметрами

Дана строка базового URL:

https://example.com/search?

Сформируйте полный URL, добавив к нему параметры для поиска. Параметры передаются в виде строки, где ключи и значения разделены `=`, а пары ключ-значение разделены `&`. Если значение `null`, то параметр не должен попадать в URL.

Пример:

params = "query=java&sort=desc&filter=null"

Результат:

https://example.com/search?query=java&sort=desc

```
class URLBuilder {

    public static String buildURL(String baseURL, String
params) {

        // Введите свое решение ниже

    }

}


// Не удаляйте этот класс - он нужен для вывода результатов на
экран и проверки

public class Printer {

    public static void main(String[] args) {

        String baseURL = "";

        String params = "";
```

```
        if (args.length == 0) {
            // При отправке кода на Выполнение, вы можете
варьировать эти параметры
            baseURL = "https://example.com/search?";
            params = "query=java&sort=desc&filter=null";
        } else {
            baseURL = args[0];
            params = args[1];
        }

        URLBuilder ans = new URLBuilder();
        System.out.println(ans.buildURL(baseURL, params));
    }
}
```

Подсказка № 1

Разделите строку параметров на части, используя символ & в качестве разделителя. Это даст вам массив строк, каждая из которых представляет собой пару "ключ=значение".

Подсказка № 2

Для каждой строки, представляющей пару "ключ=значение", разделите её на ключ и значение с помощью символа =. Это позволит вам обработать ключ и значение отдельно.

Подсказка № 3

Проверьте, если значение после символа = равно "null". В этом случае, этот параметр не должен добавляться к итоговому URL.

Подсказка № 4

Используйте `StringBuilder` для построения итогового URL. Начните с базового URL и добавляйте параметры, разделяя их символом `&`, если это необходимо.

Подсказка № 5

При добавлении первого параметра в URL, не забудьте проверить, есть ли уже символ `?` в базовом URL. Если он присутствует, параметры должны быть добавлены после него. Если его нет, начните с `?`.

Эталонное решение:

```
class URLBuilder {

   public static String buildURL(String baseURL, String params) {

       StringBuilder url = new StringBuilder(baseURL);

       String[] pairs = params.split("&");


       for (int i = 0; i < pairs.length; i++) {

           String[] keyValue = pairs[i].split("=");

           if (!"null".equals(keyValue[1])) {

               if (i > 0) {

                   url.append("&");

               }

url.append(keyValue[0]).append("=").append(keyValue[1]);

           }

       }

       return url.toString();

   }

}


// Не удаляйте этот класс - он нужен для вывода результатов на экран
и проверки
```

```java
public class Printer {

   public static void main(String[] args) {

       String baseURL = "";

       String params = "";


       if (args.length == 0) {

           baseURL = "https://example.com/search?";

           params = "query=java&sort=desc&filter=null";

       } else {

           baseURL = args[0];

           params = args[1];

       }


       URLBuilder ans = new URLBuilder();

       System.out.println(ans.buildURL(baseURL, params));

   }

}
```

Задача 2. Создание CSV-строки из массива объектов

Дан массив объектов, где каждый объект представляет собой строку данных, и массив заголовков. Создайте строку CSV, где строки данных разделяются новой строкой, а значения в строках разделяются запятыми.

Пример:

String[] headers = {"Name", "Age", "City"};

String[][] data = {

  {"John", "30", "New York"},

  {"Alice", "25", "Los Angeles"},

{"Bob", "35", "Chicago"}

};

Результат:

Name,Age,City

John,30,New York

Alice,25,Los Angeles

Bob,35,Chicago

```
class CSVGenerator {

    public static String generateCSV(String[] headers,
String[][] data) {

        // Введите свое решение ниже

    }

}


// Не удаляйте этот класс - он нужен для вывода результатов на
экран и проверки

public class Printer {

    public static void main(String[] args) {

        String[] headers = {};

        String[][] data = {};


        if (args.length == 0) {

            // При отправке кода на Выполнение, вы можете
варьировать эти параметры

            headers = new String[]{"Name", "Age", "City"};

            data = new String[][] {
```

```
                {"John", "30", "New York"},
                {"Alice", "25", "Los Angeles"},
                {"Bob", "35", "Chicago"}
            };
        } else {
            // Преобразование строковых параметров в массивы
            // Пример обработки данных можно дополнить в
зависимости от формата args
        }

        CSVGenerator ans = new CSVGenerator();
        System.out.println(ans.generateCSV(headers, data));
    }
}
```

Подсказка № 1

Используйте метод `String.join()`, чтобы объединить элементы массива заголовков, разделенные запятыми. Это создаст первую строку CSV-файла, которая будет содержать заголовки.

Подсказка № 2

Пройдитесь по массиву данных и для каждой строки используйте метод `String.join()` для объединения элементов строки, разделенных запятыми. Не забудьте добавить перевод строки после каждой строки данных.

Подсказка № 3

Добавьте заголовки в начало строки, затем добавьте строки данных. Убедитесь, что в конце строки нет лишнего перевода строки. Вы можете использовать `StringBuilder` для эффективного формирования строки CSV.

Подсказка № 4

После формирования всей строки CSV, последняя новая строка может быть лишней. Используйте метод `trim()` для удаления последнего символа новой строки из итоговой строки CSV.

Эталонное решение:

```
class CSVGenerator {

   public static String generateCSV(String[] headers, String[][]
data) {

       StringBuilder csv = new StringBuilder();



       // Добавление заголовков

       csv.append(String.join(",", headers)).append("\n");



       // Добавление данных

       for (String[] row : data) {

           csv.append(String.join(",", row)).append("\n");

       }



       return csv.toString().trim(); // Удалить последнюю новую
строку

   }

}



// Не удаляйте этот класс - он нужен для вывода результатов на экран
и проверки

public class Printer {

   public static void main(String[] args) {

       String[] headers = {};

       String[][] data = {};
```

```
        if (args.length == 0) {
            headers = new String[]{"Name", "Age", "City"};
            data = new String[][] {
                    {"John", "30", "New York"},
                    {"Alice", "25", "Los Angeles"},
                    {"Bob", "35", "Chicago"}
            };
        } else {
            // Преобразование строковых параметров в массивы
            // Пример обработки данных можно дополнить в зависимости от формата args
        }

        CSVGenerator ans = new CSVGenerator();
        System.out.println(ans.generateCSV(headers, data));
    }
}
```

Задача 3. Удаление пустых строк из текста

Дана строка с несколькими строками текста, разделенными переводами строки. Напишите метод, который удаляет все пустые строки из текста.

Пример:

line1

line2

line3

Результат:

line1

line2

line3

```
class TextCleaner {

    public static String removeEmptyLines(String text) {

        // Введите свое решение ниже

    }

}


// Не удаляйте этот класс - он нужен для вывода результатов на экран и проверки

public class Printer {

    public static void main(String[] args) {

        String text = "";


        if (args.length == 0) {

            // При отправке кода на Выполнение, вы можете варьировать эти параметры

            text = "line1\n\nline2\n\nline3";

        } else {

            text = args[0];

        }
```

```
        TextCleaner ans = new TextCleaner();

        System.out.println(ans.removeEmptyLines(text));

    }

}
```

Подсказка № 1

Используйте метод `split("\n")`, чтобы разбить исходный текст на массив строк по символу перевода строки. Это позволит вам обработать каждую строку отдельно.

Подсказка № 2

Пройдитесь по массиву строк и используйте метод `trim()` для проверки, является ли строка пустой (или состоит только из пробельных символов). Если строка не пуста, добавьте ее в результирующую строку.

Подсказка № 3

Используйте `StringBuilder` для формирования итоговой строки. При добавлении каждой непустой строки проверьте, не является ли это первой строкой (чтобы избежать добавления лишнего перевода строки в начале).

Подсказка № 4

После того как все строки будут обработаны и собраны, верните результат в виде строки с помощью метода `toString()` у `StringBuilder`.

Эталонное решение:

```
class TextCleaner {

   public static String removeEmptyLines(String text) {

       String[] lines = text.split("\n");

       StringBuilder cleanedText = new StringBuilder();


       for (String line : lines) {

           if (!line.trim().isEmpty()) {

               if (cleanedText.length() > 0) {
```

```
                    cleanedText.append("\n");
                }
                cleanedText.append(line);
            }
        }

        return cleanedText.toString();
    }
}

// Не удаляйте этот класс - он нужен для вывода результатов на экран и проверки
public class Printer {
    public static void main(String[] args) {
        String text = "";

        if (args.length == 0) {
            text = "line1\n\nline2\n\nline3";
        } else {
            text = args[0];
        }

        TextCleaner ans = new TextCleaner();
        System.out.println(ans.removeEmptyLines(text));
    }
}
```

Задача 4. Логирование операций с массивом во время поиска минимального и максимального элементов

Реализуйте метод поиска минимального и максимального элементов массива. После нахождения каждого элемента (минимального и максимального), сделайте запись в лог-файл `log.txt` в формате `год-месяц-день час:минуты {минимальный элемент}, {максимальный элемент}`.

```
import java.io.File;

import java.io.FileWriter;

import java.io.FileReader;

import java.io.BufferedReader;

import java.io.IOException;

import java.text.SimpleDateFormat;

import java.util.Date;


class ArrayOperations {

    private static File log;

    private static FileWriter fileWriter;


    public static void findMinMax(int[] arr) {

        // Реализуйте метод для поиска минимального и
максимального элемента

    }


    private static void logStep(int min, int max) {

        // Реализуйте метод для записи состояния в лог-файл

    }
```

```
}


// Не удаляйте этот класс - он нужен для вывода результатов на экран и проверки
public class Printer {
    public static void main(String[] args) {
        int[] arr = {};

        // При отправке кода на Выполнение, вы можете варьировать эти параметры
        if (args.length == 0) {
            arr = new int[]{9, 4, 8, 3, 1};
        } else {
            arr = Arrays.stream(args[0].split(", "))
                        .mapToInt(Integer::parseInt)
                        .toArray();
        }

        ArrayOperations ans = new ArrayOperations();
        ans.findMinMax(arr);

        try (BufferedReader br = new BufferedReader(new FileReader("log.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
```

```
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Подсказка № 1

Перед началом поиска минимального и максимального элементов убедитесь, что файл `log.txt` создан и открыт для записи. Используйте `File.createNewFile()` для создания файла, если он не существует, и `FileWriter` для записи в файл.

Подсказка № 2

Пройдитесь по массиву, используя цикл, чтобы найти минимальный и максимальный элементы. Начните с того, чтобы инициализировать переменные `min` и `max` первым элементом массива. В цикле сравнивайте текущий элемент с `min` и `max`, обновляя их при необходимости.

Подсказка № 3

Метод `logStep()` должен записывать текущие минимальные и максимальные значения в лог-файл. Для форматирования даты используйте `SimpleDateFormat`. Добавьте запись в файл с текущей датой и временем в формате `yyyy-MM-dd HH:mm`.

Подсказка № 4

После завершения работы с файлом обязательно закройте `FileWriter` в блоке `finally` или используя try-with-resources. Это гарантирует, что ресурсы будут освобождены корректно.

Эталонное решение:

```
import java.io.*;

import java.text.SimpleDateFormat;

import java.util.Arrays;
```

```
import java.util.Date;


class ArrayOperations {
   private static File log;
   private static FileWriter fileWriter;


   public static void findMinMax(int[] arr) {
       try {
           log = new File("log.txt");
           log.createNewFile();
           fileWriter = new FileWriter(log);


           int min = arr[0];
           int max = arr[0];


           for (int i = 1; i < arr.length; i++) {
               if (arr[i] < min) {
                   min = arr[i];
               }
               if (arr[i] > max) {
                   max = arr[i];
               }
               logStep(min, max);
           }
       } catch (IOException e) {
           e.printStackTrace();
       } finally {
```

```
            try {
                fileWriter.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    private static void logStep(int min, int max) {
        try {
            SimpleDateFormat dateFormat = new
SimpleDateFormat("yyyy-MM-dd HH:mm");
            String timestamp = dateFormat.format(new Date());
            fileWriter.write(timestamp + " " + min + ", " + max +
"\n");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

// Не удаляйте этот класс - он нужен для вывода результатов на
экран и проверки
public class Printer {
    public static void main(String[] args) {
        int[] arr = {};

        if (args.length == 0) {
```

```
            arr = new int[]{9, 4, 8, 3, 1};
        } else {
            arr = Arrays.stream(args[0].split(", "))
                    .mapToInt(Integer::parseInt)
                    .toArray();
        }

        ArrayOperations ans = new ArrayOperations();
        ans.findMinMax(arr);

        try (BufferedReader br = new BufferedReader(new
FileReader("log.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```