Seminar 5
week 5 (31 October – 6 November 2017)

1. Questions from **Lab-Assignment3 from Laboratory 4.**
2. Discussion of **Lab-Assignment4 from Laboratory 5.**
3.Discuss a classical Java functional programming example of treating a text file as a stream of strings:

3.1. First Approach

3.1.1. Basic idea

```
Stream<String> lines = Files.lines(somePath);
//reading a file as a stream of strings

public static void main(String[] args) throws Exception {
        Files.lines(Paths.get("input-file"))
                .map(someFunction)
                .filter(someTest)
                .someOtherStreamOperation(...); }
```

3.1.2. Printing out all palindromes contained in a text file

```
public static void main(String[] args) throws Exception {
        String inputFile = "in.txt";
        Files.lines(Paths.get(inputFile))
                .filter(StringUtils::isPalindrome)
                 .forEach(System.out::println);
}

public class StringUtils {
public static String reverseString(String s) { return(new StringBuilder(s).reverse().toString()); }
public static boolean isPalindrome(String s) { return(s.equalsIgnoreCase(reverseString(s))); }
}
```

3.2. Second Approach

3.2.1. Basic Idea

```
public static void useStream(Stream<String> lines, ...) {
        lines.filter(...).map(...)...;
 }

public static void useFile(String filename, ...) {
        try(Stream<String> lines = Files.lines(Paths.get(filename))) {
                SomeClass.useStream(lines, ...);
        } catch(IOException ioe) { System.err.println("Error reading file: " + ioe); }
}
```

3.2.2. Printing out all palindromes contained in a text file

```
public class FileUtils {
    public static void printAllPalindromes(Stream<String> words){
        words.filter(StringUtils::isPalindrome)
                .forEach(System.out::println);
    }

   public static void printAllPalindromes(String filename) {
```

```
        try(Stream<String> words = Files.lines(Paths.get(filename))) {
                printAllPalindromes(words);
        } catch(IOException ioe) { System.err.println("Error reading file: " + ioe); }
    }
}

public static void testAllPalindromes(String filename) {
        List<String> testWords = Arrays.asList("bog", "bob", "dam", "dad");
        System.out.printf("All palindromes in list %s:%n", testWords);
        FileUtils.printAllPalindromes(testWords.stream());
        System.out.printf("All palindromes in file %s:%n", filename);
        FileUtils.printAllPalindromes(filename);
}
```

3.3. Third Approach

3.3.1. Basic Idea:

```
public static void useStream(Stream<String> lines) {
        lines.filter(...).map(...)...;
}

public static void useFile(String filename) {
        StreamProcessor.processFile(filename, SomeClass::useStream); }

@FunctionalInterface
public interface StreamProcessor {
        void processStream(Stream<String> strings);

        public static void processFile(String filename, StreamProcessor processor) {
                try(Stream<String> lines = Files.lines(Paths.get(filename))){
                        processor.processStream(lines);
                }
                catch(IOException ioe) { System.err.println("Error reading file: " + ioe); }
        }
}
```

3.3.2. Printing out all palindromes contained in a text file

```
public static void printAllPalindromes(Stream<String> words){
        words.filter(StringUtils::isPalindrome)
        .forEach(System.out::println); }

public static void printAllPalindromes(String filename) {
        StreamProcessor.processFile(filename, FileUtils::printAllPalindromes); }

public static void testAllPalindromes(String filename) {
        List<String> testWords = Arrays.asList("bog", "bob", "dam", "dad");
        System.out.printf("All palindromes in list %s:%n", testWords);
        FileUtils.printAllPalindromes(testWords.stream());
        System.out.printf("All palindromes in file %s:%n", filename);
        FileUtils.printAllPalindromes(filename); }
```