Prof. Dr. Jasmin Blanchette
Dr. Martin Desharnais-Schäfer
Dr. Michael Kirsten
Elisabeth Lempa

Ludwig-Maximilians-Universität München
Institut für Informatik
Discussion on 21.01.2026
Homework due on 28.01.2026 at 16:00

Possible solution for Exercise Sheet 13 in Scientific and Technical English for Computer Scientists

The exercise sheets consist of in-class exercises and homework. The in-class exercises take place during the second half of the lecture time slots. The homework, which is optional and ungraded, can be submitted via the "Homework" section in Moodle. The homework is subject to peer review.

Unless indicated otherwise, generative artificial intelligence assistants such as ChatGPT may be used, as long as you acknowledge how you use them as specified by the Institute's policy on plagiarism.[1] However, you may not use such tools to generate peer reviews for you. In addition, we strongly recommend that you do not use them to generate entire solutions, since that would defeat the purpose of the exercises.

In-class exercise 13-1 ***Grasping `grep`*** The `grep` tool is a command-line program that is available on Linux and similar systems. Make yourself familiar with the manual page, abbreviated as *man page*, for `grep`. You can access the man page by typing `man grep` in your terminal if you are working on a Linux-like system or by visiting `https://man7.org/linux/man-pages/man1/grep.1.html`.

Write a brief tutorial for `grep` of 200 to 400 words where you describe its purpose and explain its basic usage. In particular, explain how to search multiple files by using file-name patterns (such as `grep hello *.txt`) and the `-r` option.

POSSIBLE SOLUTION:

What Is `grep`?

`grep`, which abbreviates "global regular expression print," is a command-line tool for searching for text patterns. The following brief tutorial will illustrate its basic usage.

Searching in One File

The most straightforward application of `grep` is searching for a given text in a single file.

```
$ grep "hello" greetings.txt
hello is a casual greeting
```

[1] `https://www.medien.ifi.lmu.de/lehre/Plagiate-IfI.pdf`

Searching in Multiple Files

One way to search through multiple files is to use the `-r` option. When this option is enabled, `grep` searches a directory recursively, i.e., including all its subdirectories.

```
$ grep -r "TODO" .
notes.txt:after we fix all TODOs we will sell the app for 1 million
app.py:#TODO add input parameter
server.py:#TODO improve logging
```

Another way to search multiple files is to use a file-name pattern. The following example searches all Java source files in the current directory:

```
$ grep "throws" *.java
Main.java:void readUserName() throws IOException {
```

Note that the expansion of `*.java` is handled by your shell, before grep itself is called. This means that the `-r` option will not work together with file-name patterns. Compare:

```
$ grep -r "TODO" .
notes.txt:after we fix all TODOs we will sell the app for 1 million
app.py:#TODO add input parameter
server.py:#TODO improve logging
bio/bird_counter.py:#TODO Add special case for the empty list

$ grep "TODO" *.py
app.py:#TODO add input parameter
server.py:#TODO improve logging
```

Because `grep` receives a concrete list of all files in the current directory ending in `.py`, and not a directory, subdirectories will not be searched, even with the `-r` option. To search recursively for file names matching a specific pattern, use the `--include` option:

```
$ grep -r --include="*.py" "TODO" .
app.py:#TODO add input parameter
server.py:#TODO improve logging
bio/bird_counter.py:#TODO Add special case for the empty list
```

Searching for a Regular Expression

If you do not only want to search for a specific string, but for a pattern, `grep` natively supports some regular expressions, such as character classes:

```
$ grep -r [0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9] .
./book/chapter_1.tex:%% Markus, 2011-12-13: Added handling of umlauts
./book/chapter_4.tex:%% Markus, 2012-02-04: Added handling for long ↲
    source lines
```

To enable support for advanced regular expression patterns, use the `-E` option.

Controlling the Shape of the Output

More options such as `-l` (show only file names) and `-c` (count matches per file) can be used to control what `grep` outputs. This is can be especially useful if you are using `grep` in conjunction with other command-line tools.

```
$ grep -rl "TODO" . | wc -l
412
```

In-class exercise 13-2 *Javadoc for a Number Queue Class* The `WaitingList` class is a Java implementation of a number queue system that can be used to manage people who are waiting for their turn. Write appropriate Javadoc comments for all public methods of the class and for the class itself. Make sure to include relevant `@` tags.

```java
public class WaitingList {
    private List<Integer> items;
    private Random random;

    public WaitingList() {
        this.items = new ArrayList<>();
        this.random = new Random();
    }

    public int drawNumber() {
        int number;
        do {
            number = random.nextInt(1000) + 1;
        } while (items.contains(number));

        items.add(number);
        return number;
    }

    public Integer callNextNumber() {
        if (items.isEmpty()) {
            return null;
        }
        return items.remove(0);
    }

    public void addNumber(int number) {
        if (items.contains(number)) {
            throw new IllegalArgumentException("Number "
                    + number + " is already in the waiting list");
        }
```

```java
            items.add(number);
        }
}
    }
```

POSSIBLE SOLUTION:

```java
/**
 * A waiting list that manages unique random numbers
 * in FIFO order.
 */
public class WaitingList {
    private List<Integer> items;
    private Random random;

    public WaitingList() {
        this.items = new ArrayList<>();
        this.random = new Random();
    }

    /**
     * Generates a random number not already in the
     * waiting list, adds it to the list, and returns it.
     *
     * @return an integer between 1 and 1000 that
     *   was newly added to the waiting list (and was not
     *   present in the list previously).
     */
    public int drawNumber() {
        int number;
        do {
            number = random.nextInt(1000) + 1;
        } while (items.contains(number));

        items.add(number);
        return number;
    }

    /**
     * Removes and returns the next number from the
     * front of the waiting list.
     *
     * @return the next number in line, or null if
     *   the waiting list is empty
     */
    public Integer callNextNumber() {
        if (items.isEmpty()) {
            return null;
        }
        return items.remove(0);
    }

    /**
     * Adds a specific number to the back of the
     * waiting list.
     *
     * @param number the number to add to the
     *   waiting list
     * @throws IllegalArgumentException if the number is
     *   already in the waiting list
     */
    public void addNumber(int number) {
        if (items.contains(number)) {
            throw new IllegalArgumentException("Number "
              + number + " is already in the waiting list");
        }
        items.add(number);
    }
}
```