OULU UNIVERSITY OF
APPLIED SCIENCES

Mika Lahtinen

SOFTWARE UPDATER

Updating a highly customised software efficiently

SOFTWARE UPDATER

Updating a highly customised software efficiently

Mika Lahtinen
Thesis
Autumn 2017
Business Information Technology
Oulu University of Applied Sciences

YHTEENVETO

Oulun Ammattikorkeakoulu, Liiketalouden yksikkö
Bachelor of Business Information Technology

Tekijä: Mika Lahtinen
Opinnäytetyön otsikko: Software Updater
Työn valvojat: Ilkka Mikkonen, Winfried Bittner
Valmistumisjakso ja -vuosi: Syksy 2017 Sivujen määrä: 27

Lähes kaikki ohjelmistot täytyy ajoittain päivittää, jotta uudet käyttö- ja alustavaatimukset täyttyvät, ohjelmiin jääneet ohjelmointivirheet ja tietoturva-aukot korjataan sekä uudet ominaisuudet saadaan käyttöön.

Tämä opinnäytetyö on dokumentaatio päivitystyökalusta joka on itse opinnäytetyön fyysinen osio, ja ohjelmoitu Persis OY:lle. Päivitystyökalu auttaa yritystä ajamaan vanhat ja uudet päivitykset nopeammin ja helpommin, ja osittain tuottamaan uudet päivitykset yhtenäisemmin.

Opinnäytetyön teko sisälsi tutkimusta käytännön toteutukseen, prototyypin teon ja testauksen, sekä prototyypin hienosäädön sopimaan juuri tämän yrityksen ohjelmiston päivitykseen. Saatavilla ollut materiaali sisältää löytyneet verkkomateriaalit, mutta myös jo hankitun tietotaidon yrityksen omasta ohjelmistosta, sekä yleisen ohjelmointitaidon.

Työn tulos on päivitystyökalu sekä siihen liittyvät päivitysparannukset. Työkalua voidaan jatkokehittää soveltumaan muihinkin päivitystilanteisiin Persis OY:n ohjelmiston kanssa, mutta nykyisellään se soveltuu vain kyseiseen ohjelmistoon.

Tässä vaiheessa haluaisin kiittää kaikkia jotka ovat suoraan tai välillisesti avustaneet työkalun ja sen dokumentaation synnyssä, erityisesti nimeten Winfried Bittnerin, Ilkka Mikkosen, sekä Eric Sängerin (Sukunimen mukaisessa aakkosjärjestyksessä)

Asiasanat:

Päivitys, päivittää, ohjelmisto, automaatio, tietojenkäsittely

ABSTRACT

Oulu University of Applied Sciences
Bachelor of Business Information Technology

Author: Mika Lahtinen
Title of Bachelor´s thesis: Software Updater
Supervisors: Ilkka Mikkonen, Winfried Bittner
Term and year of completion: Autumn 2017 Number of pages: 27

Most software needs to be updated to meet the new requirements of usage and platforms, bugs remaining in the application, security issues and new purposes.

This thesis is documentation about an update tool made for Persis GmbH for software updating, which is the physical part of the thesis work. The software updating tool helps the company performing old and new updates in more automated and faster way, and producing the new updates in a more uniformed way.

The thesis included researching practical knowledge about updating, creating a software prototype, testing the prototype and adjusting the prototype to fit the exact needs of the company. Available material included found online material but also already acquired knowledge of the company's own software and programming knowledge in general.

Main result of the thesis is the update tool and related update improvements. The update tool can be further developed to suit all different updating occasions in Persis GmbH software but in the current form only with existing Persis GmbH software.

At this point, I would like to thank everyone who has aided directly or indirectly in creating this tool and its documentation, specially naming Winfried Bittner, Ilkka Mikkonen and Eric Sänger (Alphabetical order by last name).

Keywords:

Update, Upgrade, Software, Automating, Information Technology

ZUSAMMENFASSUNG

Hochschule Oulu, Geschäft
Bachelor über Geschäft Informatik

Autor: Mika Lahtinen
Bachelorarbeits Titel: Software Updater
Betreuers: Ilkka Mikkonen, Winfried Bittner
Fertigstellungszeitpunkt des Semester und Jahre: Herbst 2017 Nummer des Seiten: 27

Die meisten Software müssen updated werden, um den neuesten Voraussetzungen der Nutzung und Plattformen gerecht zu werden, Bugs und Sicherheitsprobleme in der Software zu beheben, sowie neue Funktionalitäten ergänzen.

Diese Bachlorarbeit befasst sich mit der Dokumentation einer neuen Updatefunktion, um die Software der Persis GmbH möglichst komfortabel auf einen aktuellen Stand zu bringen und zusätzlich Zeit beim Updaten zu sparen.

Ziel ist es mit dem neuen Tool alte sowie neue Updateprozesse zu vereinheitlichen zu automatisieren und zu vereinfachen. Das Hauptziel ist das UpdateTool mit den dazugehörigen Verbesserungen zum allgemeinen Updateprozess.

Diese Arbeit enthält Recherchen und praktisches Wissen über Updates sowie die Erstellung und das Testen eines Software-Prototyps unter allen nötigen Umständen und Abstimmung der Funktion auch unter speziellen Umständen.

Das Arbeitsmaterial, stammt sowohl aus dem Internet, als auch aus bereits erlernten und angeeigneten Kenntnissen über die firmeneigene Software.

Es ist möglich dieses Tool zukünftig weiterzuentwickeln, um weitere Updateschritte
zu automatisieren und zu vereinfachen. Die Version des hier erstellten Programmes setzt jedoch das aktuell bestehende Persis 2017 vorraus.

Zu diesem Zeitpunkt möchte ich allen Danken, die mittelbar oder unmittelbar geholfen haben dieses Tool und die Dokumentation zu erstellen. Besonders hervorheben möchte ich hier Winfried Bittner, Ilkka Mikkonen und Eric Sänger (Alphabetische Ordnung nach Nachname)

Stichworte:

Aktualisieren, Updaten, , Upgrade, Software, Automating, Information Technology

CONTENTS

1 INTRODUCTION

Every advanced machine from basic CNC machinery to highly advanced Supercomputers needs a piece of software to contain its logic and machine instructions for the machine to be able to function. This software though is in almost all cases practically impossible to be finished completely during the first production phase, without even mentioning the high possibility of software bugs in the software due to human errors.

These problems from missing features to fixing bugs can be corrected with software updates. Everyone who uses computers, smartphones or other similar devices daily, have encountered software updates. Usually these updates are really simple and easy for the end user, varying from just allowing once to be automatically updated to manually clicking update icon when starting an application or the operating system.

With highly customized company business applications, this is not always the case though. Such applications need to be updated separately and with care, because of the customisations. Now of course a question arises, how can then highly customized software be updated without serious issues with the customisations. In this case, the core engine updating is made more efficient and the customisation parts as an interface layer are then separately updated if necessary.

This thesis demonstrates one convention for updating an application, making the updating process as automatic and efficient as possible.

2 TECHNICAL JARGON EXPLAINED

As a technical programming work, this thesis contains multiple technical words that might not be familiar for all readers. Therefore in this chapter those words are explained and opened up in order to provide more fluent reading experience.

2.1 Tomcat

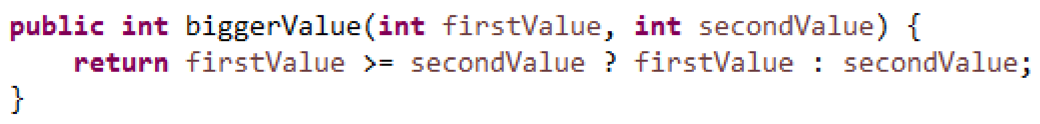
Tomcat is an application server from the Apache Software Foundation for Java applications. It can be used either as a standalone product with its own internal Web server or together with other Web servers. (Rouse 2005)

2.2 Parameter and passing an argument in programming

A parameter in programming means a variable name that will hold an argument, whereas an argument in programming is the value that is passed. Methods explained in 2.3 can take parameters to have information to work with. As seen in figure one, integers “firstValue” and “secondValue” are parameters that can hold arguments. As an example if we assigned the variables with values 2 and 4 as follows:
int firstValue = 2;
int secondValue = 4;
We could pass those to the method “biggerValue” and get an answer of ‘4’.

2.3 Method in programming

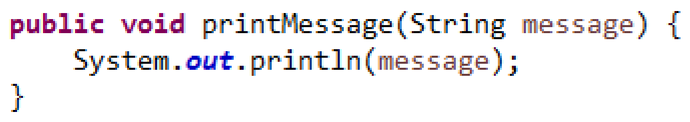
A method defines the behaviour of the objects that are created from the class. Another way to say this is that a method is an action that an object is able to perform (Zandbergen 2017). In conclusion, a method is a piece of code that can be used multiple times. It also defines the behaviour of the part. A method can take passed arguments, and it can return a value. In figure one can be seen an example of a method that takes two arguments and returns one.



```java
public int biggerValue(int firstValue, int secondValue) {
    return firstValue >= secondValue ? firstValue : secondValue;
}
```

FIGURE 1. Example method

In comparison figure two shows an example of a method that takes one argument but returns no value.



```java
public void printMessage(String message) {
    System.out.println(message);
}
```

FIGURE 2. Example method 2

2.4 Checksum

A checksum is the outcome of running an algorithm, called a cryptographic hash function, on a piece of data, usually a single file. A checksum is also sometimes called a hash sum and less often a hash value, hash code, or simply a hash (Fisher 2017). A checksum consists of predefined length of characters or characters numbers and symbols, depending on the used algorithm. For instance, the algorithm SHA-1 used in the update tool returns 160 bits long string that is converted to 40 hexadecimal numbers long string.

3 UPDATING AND UPGRADING DEFINITIONS

There are many different types of software updates, varying by the size of the update compared to the total size of the program, but also by the time scale that the update is released. This chapter will explain some most commonly used definitions.

3.1 Bug

A bug in software means nowadays an error in the application code, which results in unexpected and usually erroneous behaviour. Bug in software is a result of a human error when writing the application. The origin for the word comes from an actual bug in an old computer as seen in figure three.

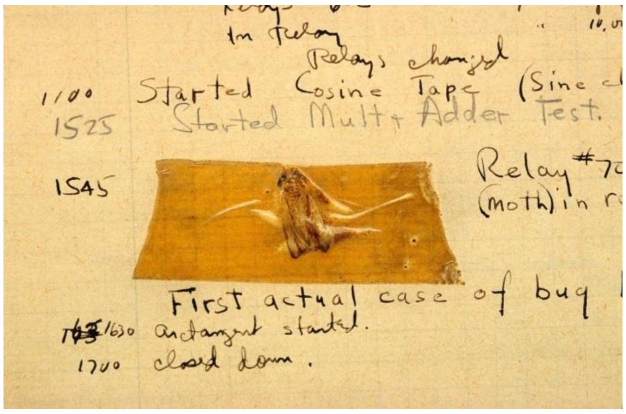


FIGURE 3. A bug in Mark II computer

In 1947, a physical malfunction in the Mark II computer was traced back to a moth stuck in one of the relays (Harvard University Collection of Historical Scientific Instruments 2017).

3.2 Software update

Definition of update for software means a minor improvement to correct bugs or add some minor features that has been missing. An update can also include driver updates that improve the operation of hardware or peripherals or add support for new models of peripherals. (Elmblad 2017) Software update is always scheduled, although the schedule might not be with even intervals.

3.3 Software upgrade

Software upgrade is a new major update release version, which can also often cost some money for the user. Software upgrades are always scheduled and usually road mapped beforehand.
If you bought your software recently, and an upgrade is released soon after that, most personal finance software companies will offer the upgrade to the latest version free (Elmblad 2017).

3.4 Bug fix

A bug fix is a change to a system or product designed to handle a programming bug/glitch (Techopedia 2017). Bug fix is usually a small fix but it has to be released, thus it is nevertheless more than simply fixing a piece of software while writing it. Bug fixes does not have to be scheduled, and are usually done as soon as possible, preferably immediately after the bug is found.

3.5 Patch

Usually addressing only a couple of serious bugs or some minor problems, which affect negatively, but is usually not completely blocking the use. The term was coined at forties, when some computing machines worked with punched cards. In the cards, an update to the card roll was made by patching specific holes and making new holes to different spots as seen in figure four.

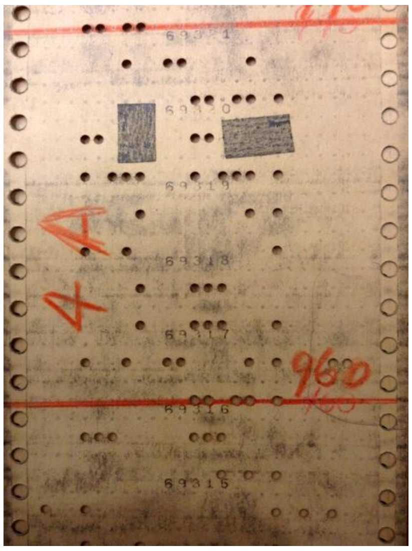


FIGURE 4. Patches from Mark I computer

Small corrections to the programmed sequence could be done by patching over portions of the paper tape and re-punching the holes in that section (Harvard Collection of Historical Scientific Instruments 2017). After the rise of the popularity of Internet, online patch downloads from developers' websites began getting easier and more popular. Starting with Apple's Mac OS 9 and Microsoft's Windows ME, PC operating systems gained the ability to get automatic software updates via the Internet (Wikipedia 2017). Apple's Mac OS 9 was available since October 1999 and Microsoft's Windows ME has been publicly available since September 2000.

3.6 Hotfix or Quick fix

Hotfixes and quick fixes are not scheduled, but applied between minor releases, usually as soon as the bug is found or improvement is needed and programmed. With customised software, a hotfix can be also made just for specific customer or usage occasion. Originally, the term hotfix was used to describe a kind of patch that could be applied without stopping or restarting a service or system. Microsoft usually uses the term hotfix to refer to a small update addressing a very specific, and often very serious, issue. (Fisher 2017)

4 UPDATING CASE WITH THE SPECIFIC CUSTOMISED SOFTWARE

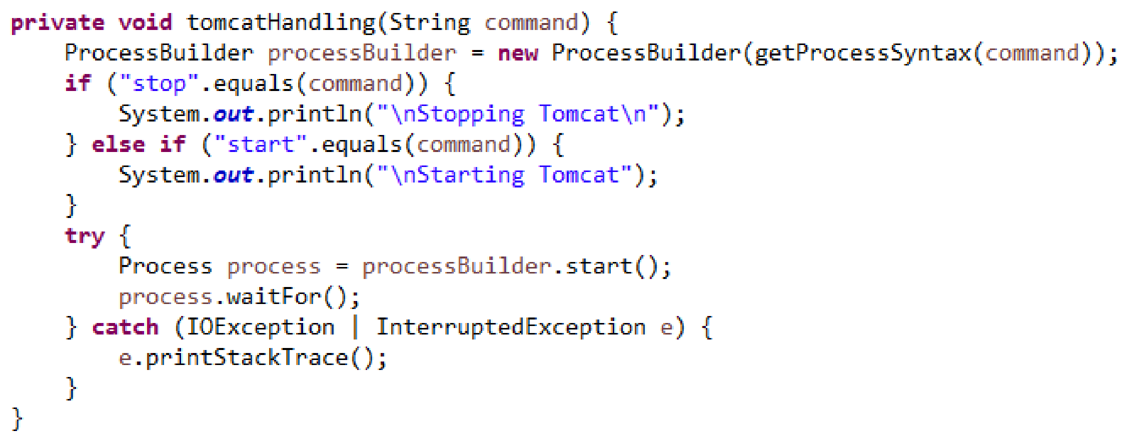
Usually updating an application is straightforward process for the end user. Commonly the application checks itself when launching if it is already the latest version or not. If the software is not the latest version, the user is prompted if the application should be updated now or later. If the user accepts, the application will download the updates and install them.

4.1 Checking the version

Later in chapter five, version controlling is explained in more detail, from checking and comparing the user’s current version to the newest version, to making sure that all parts needed for the next update are done. In this chapter, it is enough to know that it is taken care of in the process at this point.

4.2 Shutting down Tomcat automatically

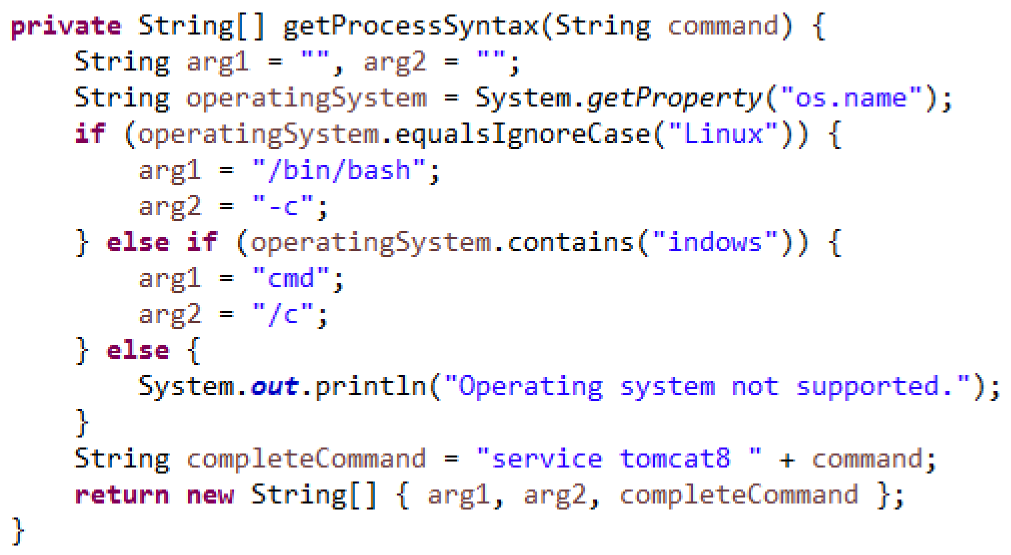
Tomcat is shut down and started again with a process command inside the application. Because the process is ran with a ProcessBuilder, the same handling method can be used for both stopping and starting, only by passing the command as an argument. The next figure shows how the ProcessBuilder is started at row nine with the command that is given to a new ProcessBuilder at row two.



```
private void tomcatHandling(String command) {
    ProcessBuilder processBuilder = new ProcessBuilder(getProcessSyntax(command));
    if ("stop".equals(command)) {
        System.out.println("\nStopping Tomcat\n");
    } else if ("start".equals(command)) {
        System.out.println("\nStarting Tomcat");
    }
    try {
        Process process = processBuilder.start();
        process.waitFor();
    } catch (IOException | InterruptedException e) {
        e.printStackTrace();
    }
}
```

FIGURE 5. Tomcat handling method

The whole process syntax is retrieved from a helper method getProcessSyntax as is shown in figure 6.



```java
private String[] getProcessSyntax(String command) {
    String arg1 = "", arg2 = "";
    String operatingSystem = System.getProperty("os.name");
    if (operatingSystem.equalsIgnoreCase("Linux")) {
        arg1 = "/bin/bash";
        arg2 = "-c";
    } else if (operatingSystem.contains("indows")) {
        arg1 = "cmd";
        arg2 = "/c";
    } else {
        System.out.println("Operating system not supported.");
    }
    String completeCommand = "service tomcat8 " + command;
    return new String[] { arg1, arg2, completeCommand };
}
```

FIGURE 6. Tomcat process syntax method

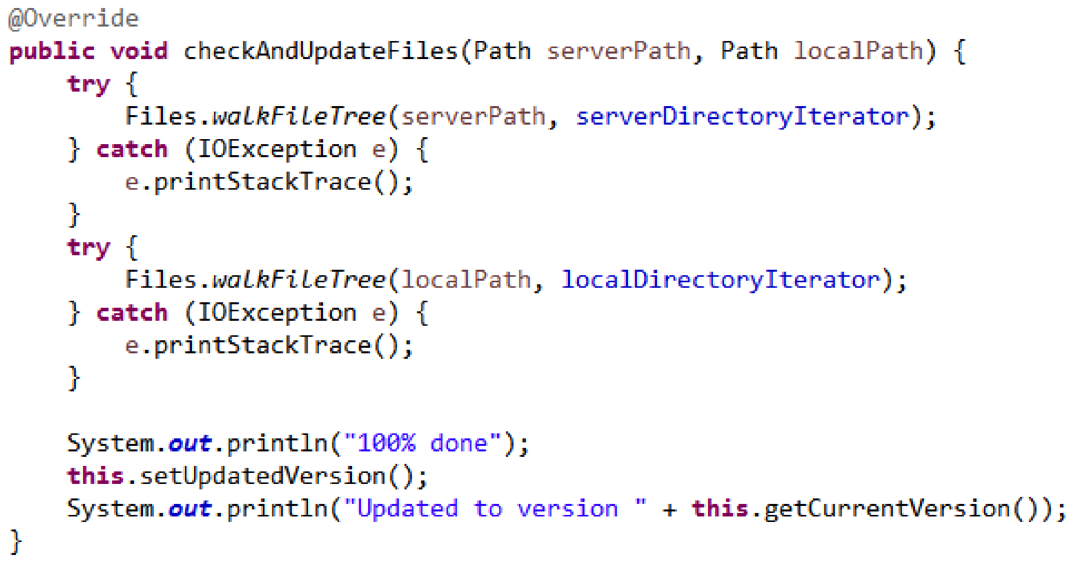
The operating system is needed to build up the whole command syntax right, as the command is different in Unix and Windows based systems. Linux operating systems retrieve the result as “Linux” or “linux”, whereas Windows operating systems can retrieve it in many forms such as “Windows XP”, “windows 2000” or “Windows 8.1”, to name a few. Therefore, row seven is only checking if the retrieved operating system name contains the word “indows”. With “equals” – check it is possible to ignore the casing. Apple’s OS X is not supported.

4.3 Updating the files

The updating of the files is done efficiently, by first getting the checksum of every file separately, both local and on the server. If the checksum of the server file is different from the local one, the new file is downloaded and replaced. However, if the checksum is same with the server file and the local file, the file is skipped. This way the updating process is much faster, as the already similar files do not have to be downloaded again and replaced, but can be just skipped instead. The stage-by-stage explanation for the process is explained in more detail in the following parts.

4.4 Browsing through all the files on server

Browsing through all the dozens of thousands of files can take time. In Java 7 was introduced a new file tree walker, which makes browsing through a complete file directory tree faster and more efficient. The update tool utilizes exactly this implementation, by walking through the whole directory tree recursively twice, as is seen in figure seven.



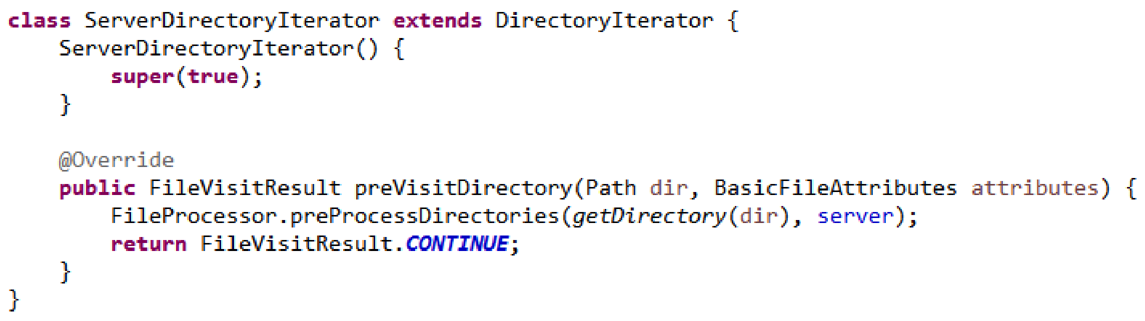
```java
@Override
public void checkAndUpdateFiles(Path serverPath, Path localPath) {
    try {
        Files.walkFileTree(serverPath, serverDirectoryIterator);
    } catch (IOException e) {
        e.printStackTrace();
    }
    try {
        Files.walkFileTree(localPath, localDirectoryIterator);
    } catch (IOException e) {
        e.printStackTrace();
    }

    System.out.println("100% done");
    this.setUpdatedVersion();
    System.out.println("Updated to version " + this.getCurrentVersion());
}
```

FIGURE 7. Walking the file tree

The previous method is the main process of checking and calling the updating methods, thus it also contains the user feedback for notifying that the file updating process is finished, to update the current version to the program, and to notify the user which version the program was updated to.

On first walk, the directories are pre-processed, creating the required new subdirectories, so the new files can be added.

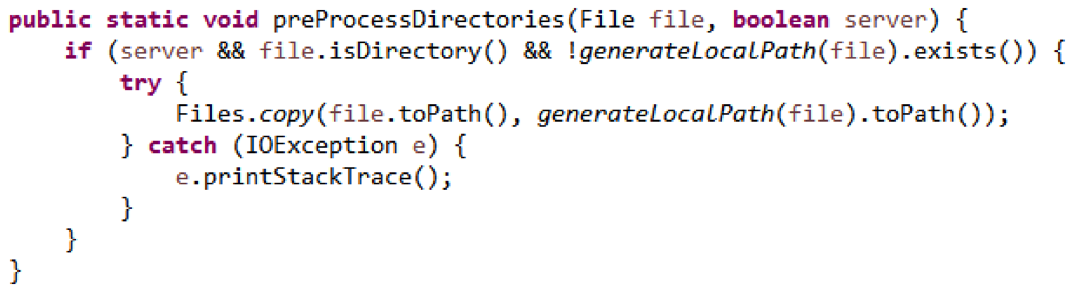


```java
class ServerDirectoryIterator extends DirectoryIterator {
    ServerDirectoryIterator() {
        super(true);
    }

    @Override
    public FileVisitResult preVisitDirectory(Path dir, BasicFileAttributes attributes) {
        FileProcessor.preProcessDirectories(getDirectory(dir), server);
        return FileVisitResult.CONTINUE;
    }
}
```

FIGURE 8. Server directory iteration

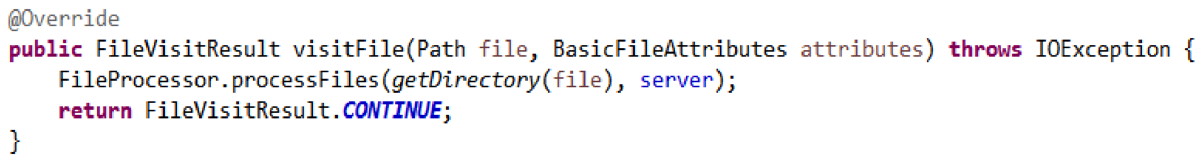
If the current walked file is a directory, but it does not exist yet, it must be created as shown in figure nine.



```java
public static void preProcessDirectories(File file, boolean server) {
    if (server && file.isDirectory() && !generateLocalPath(file).exists()) {
        try {
            Files.copy(file.toPath(), generateLocalPath(file).toPath());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

FIGURE 9. Pre-processing directories

On the second walk, the updater will check and change the necessary files, doing the actual file switching process.

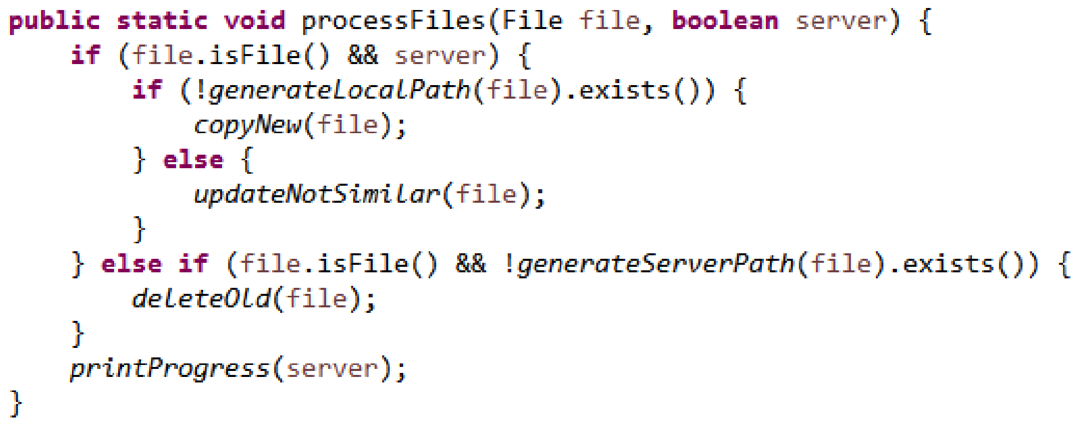


```java
@Override
public FileVisitResult visitFile(Path file, BasicFileAttributes attributes) throws IOException {
    FileProcessor.processFiles(getDirectory(file), server);
    return FileVisitResult.CONTINUE;
}
```

FIGURE 10. Iterating through the files and calling the file processing method for each

4.5 Copying new files and updating changed files

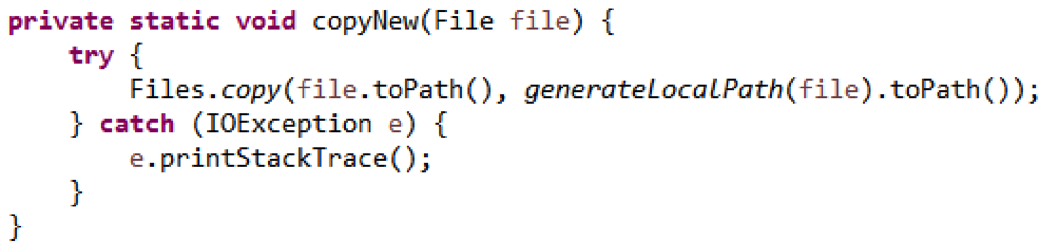
The following is the actual processing, after the directories are prepared. Processing first checks if it is going through server files or local files, as it will not delete anything from the server, but it also has to know if the file is obsolete, thus can be deleted.



```
public static void processFiles(File file, boolean server) {
    if (file.isFile() && server) {
        if (!generateLocalPath(file).exists()) {
            copyNew(file);
        } else {
            updateNotSimilar(file);
        }
    } else if (file.isFile() && !generateServerPath(file).exists()) {
        deleteOld(file);
    }
    printProgress(server);
}
```

FIGURE 11. Processing the files depending on the type

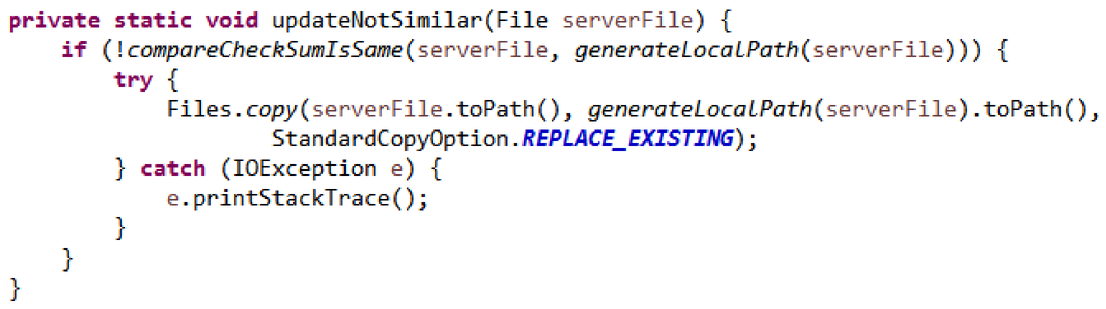
The progress printing will be explained later in chapter six. If the file exists on server, but not on local machine, it has to be copied and the following method is called. Copying the new files is done with the already existing code, and is not part of the thesis work, but figure 12 has an example of one way to copy a file with Java.



```
private static void copyNew(File file) {
    try {
        Files.copy(file.toPath(), generateLocalPath(file).toPath());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

FIGURE 12. Copying new file

If the file exists on both server and the local machine, it has to be checked and if it is changed in the updated version, it will be downloaded and switched to newer version.

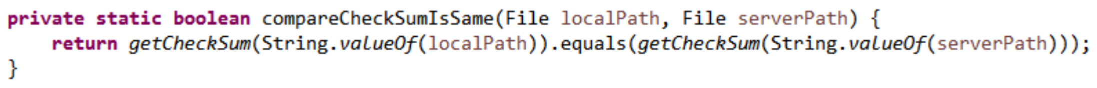


```java
private static void updateNotSimilar(File serverFile) {
    if (!compareCheckSumIsSame(serverFile, generateLocalPath(serverFile))) {
        try {
            Files.copy(serverFile.toPath(), generateLocalPath(serverFile).toPath(),
                    StandardCopyOption.REPLACE_EXISTING);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

FIGURE 13. Updating changed files

4.6 Comparing the checksum of files

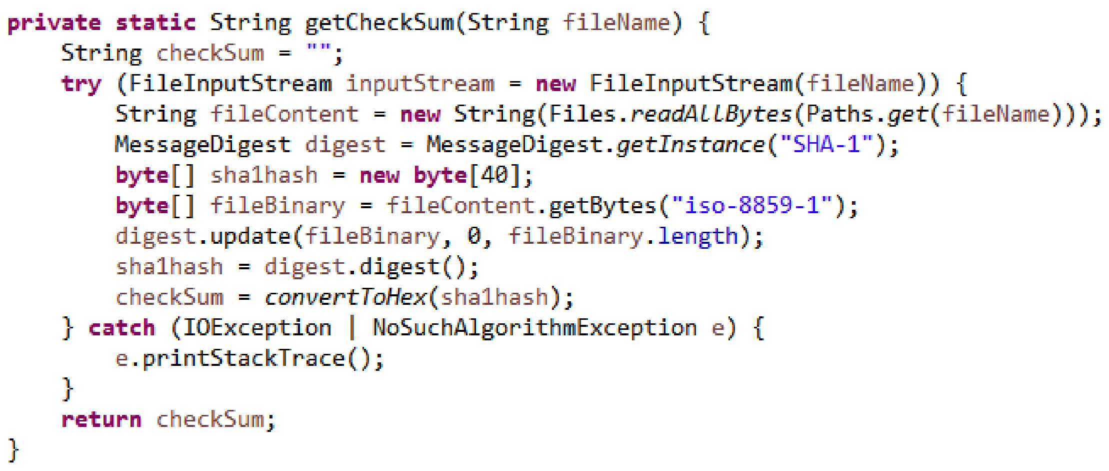
The similarity of server and local files can be checked and compared with a checksum of the file, which is shown in figure 14.



```java
private static boolean compareCheckSumIsSame(File localPath, File serverPath) {
    return getCheckSum(String.valueOf(localPath)).equals(getCheckSum(String.valueOf(serverPath)));
}
```

FIGURE 14. Comparing checksums

The getCheckSum method will get the checksum of the passed filename. First, it starts by reading the file contents byte by byte to a string, which will be converted to a byte array. The byte array is then digested by the Java standard library method to a 40 bytes long byte array. The resulting byte array is then converted to hex by the method convertToHex.



```java
private static String getCheckSum(String fileName) {
    String checkSum = "";
    try (FileInputStream inputStream = new FileInputStream(fileName)) {
        String fileContent = new String(Files.readAllBytes(Paths.get(fileName)));
        MessageDigest digest = MessageDigest.getInstance("SHA-1");
        byte[] sha1hash = new byte[40];
        byte[] fileBinary = fileContent.getBytes("iso-8859-1");
        digest.update(fileBinary, 0, fileBinary.length);
        sha1hash = digest.digest();
        checkSum = convertToHex(sha1hash);
    } catch (IOException | NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    return checkSum;
}
```

FIGURE 15. Getting the checksum of a file

The method convertToHex works by flipping the bits four times right, resulting in a value divided by 16 and rounded down. The resulting value is then converted to hex by getting the corresponding character, and appended to the end of the created string. In the end, the hex converted string is returned to the method getCheckSum, and passed forward to the original caller of getCheckSum.

```java
private static String convertToHex(byte[] data) {
    StringBuilder builder = new StringBuilder();
    for (byte element : data) {
        int halfByte = element >>> 4 & 0x0F;
        int twoHalfs = 0;
        do {
            if (0 <= halfByte && halfByte <= 9) {
                builder.append((char) ('0' + halfByte));
            } else {
                builder.append((char) ('a' + (halfByte - 10)));
            }
            halfByte = element & 0x0F;
        } while (twoHalfs++ < 1);
    }
    return String.valueOf(builder);
}
```

FIGURE 16. Converting a binary value to hexadecimal

4.7 Deleting old obsoleted files

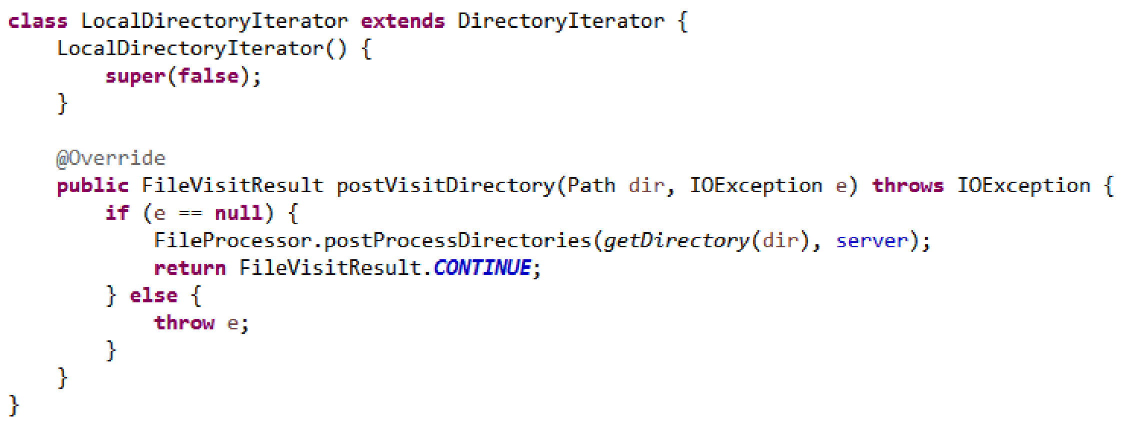
When the file processing comes to the point that it is iterating through local files, it is already doing the clean up to delete obsoleted local files that has been deleted from the updated version.

```java
private static void deleteOld(File file) {
    file.delete();
}
```

FIGURE 17. Deleting obsoleted file

When the file swapping is finished, not only copying new but also switching changed and deleting obsoleted files it is time to clean up the obsoleted directories. The following part will show how the directories are again walked through, but this time only doing some deletions when needed.

It all begins very similarly as previously, walking again the directory tree, but this time iterating the local directory.

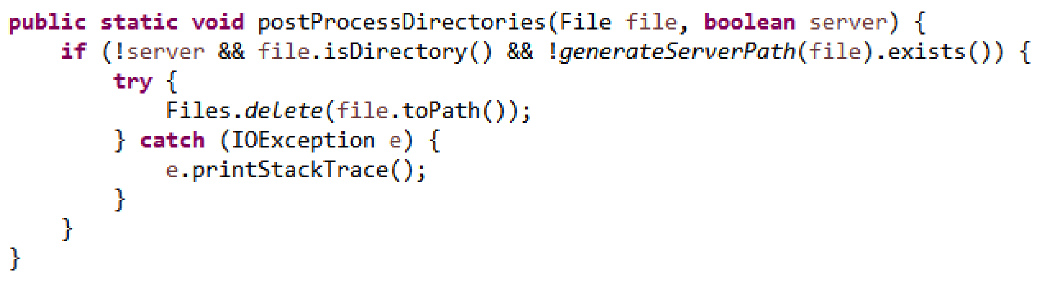


```java
class LocalDirectoryIterator extends DirectoryIterator {
    LocalDirectoryIterator() {
        super(false);
    }

    @Override
    public FileVisitResult postVisitDirectory(Path dir, IOException e) throws IOException {
        if (e == null) {
            FileProcessor.postProcessDirectories(getDirectory(dir), server);
            return FileVisitResult.CONTINUE;
        } else {
            throw e;
        }
    }
}
```

FIGURE 18. Iterating through local directory

The post processing will check if the currently walked directory also exists on the server. If it does not exist anymore, it can be deleted from the local system.



```java
public static void postProcessDirectories(File file, boolean server) {
    if (!server && file.isDirectory() && !generateServerPath(file).exists()) {
        try {
            Files.delete(file.toPath());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

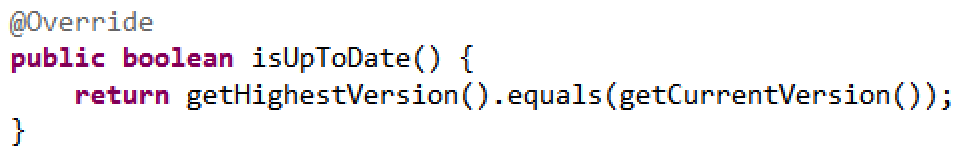
FIGURE 19. Post processing directories

5 VERSION CONTROLLING

As stated previously in the thesis, version controlling happens in the beginning of the program, but it will be explained in more detail at this point. The important thing is though, that the version is controlled in the beginning and will result in closing the updater, should the application already be up to date.

5.1 Checking if the user has the latest version

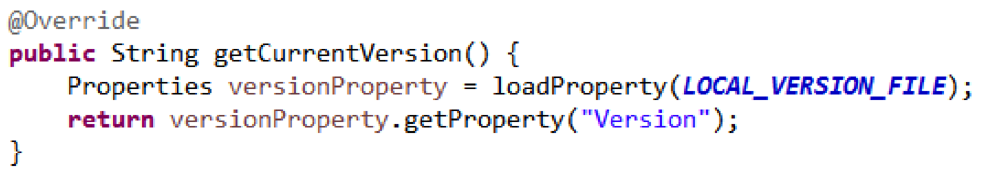
The updater has some public methods to check if the software is already up to date, getting the current version and getting the newest or highest version. The software is up to date if the newest version has the same version number as the current local version, as shown in figure 20.



```java
@Override
public boolean isUpToDate() {
    return getHighestVersion().equals(getCurrentVersion());
}
```

FIGURE 20. Check if the software is up to date

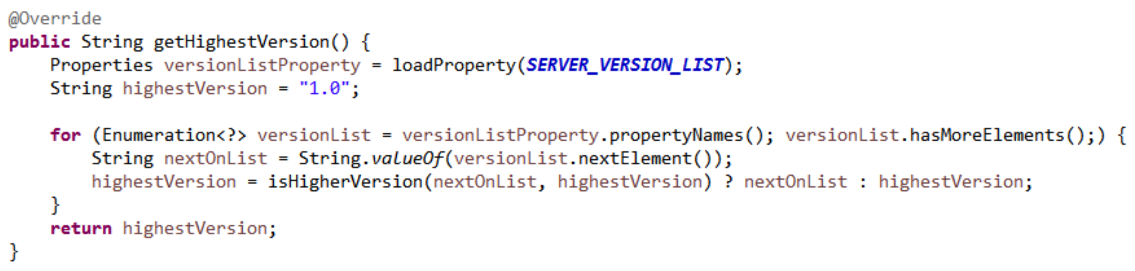
The method above is comparing two values, which are retrieved by the following methods getCurrentVersion and getHighestVersion. First of the methods gets the current version from the user's installation directory from a designated property file.



```java
@Override
public String getCurrentVersion() {
    Properties versionProperty = loadProperty(LOCAL_VERSION_FILE);
    return versionProperty.getProperty("Version");
}
```

FIGURE 21. Getting current version number

The following method is also using a comparing helper method, which can be seen in figure 22.



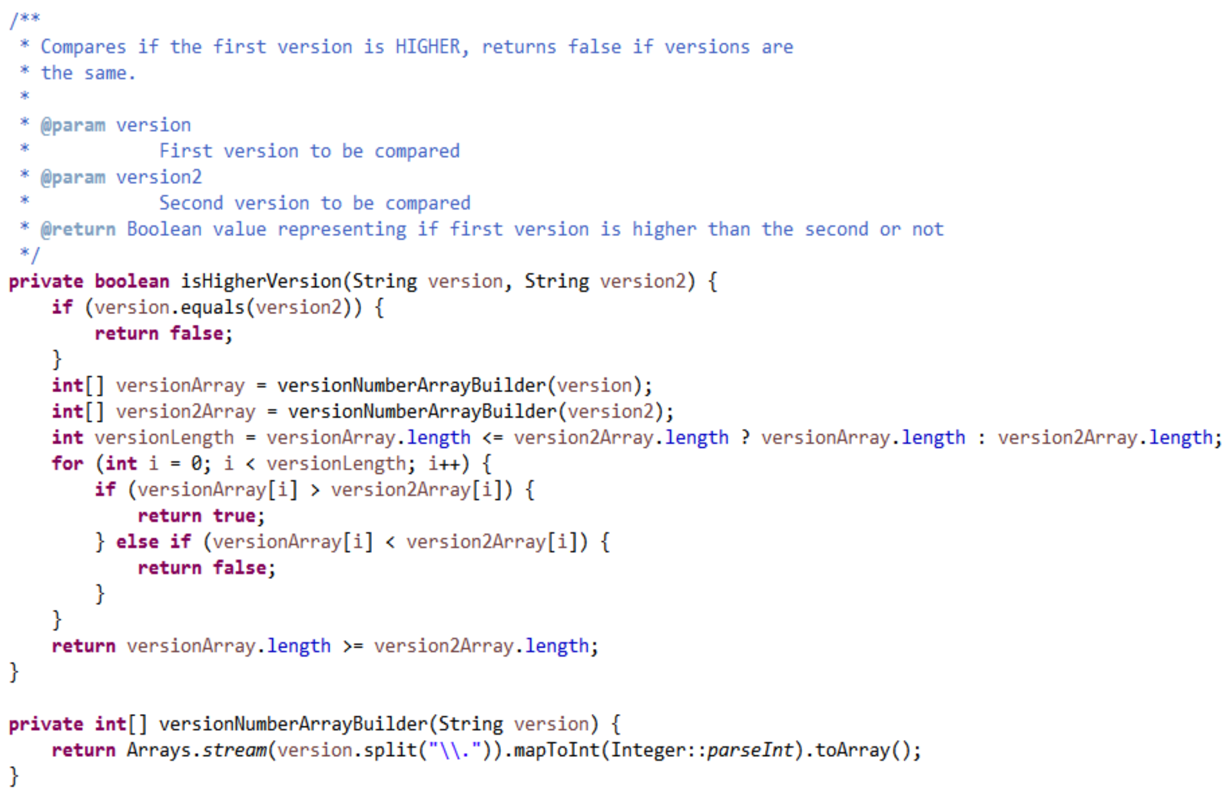
```java
@Override
public String getHighestVersion() {
    Properties versionListProperty = loadProperty(SERVER_VERSION_LIST);
    String highestVersion = "1.0";

    for (Enumeration<?> versionList = versionListProperty.propertyNames(); versionList.hasMoreElements();) {
        String nextOnList = String.valueOf(versionList.nextElement());
        highestVersion = isHigherVersion(nextOnList, highestVersion) ? nextOnList : highestVersion;
    }
    return highestVersion;
}
```

FIGURE 22. Getting highest version number

5.2 Comparing versions

The following helper method is only checking if the first given version argument is higher than the second given version argument. The method will return false if the versions are the same. With the method, it is possible to distinguish and order versions reliably.



```java
/**
 * Compares if the first version is HIGHER, returns false if versions are
 * the same.
 *
 * @param version
 *            First version to be compared
 * @param version2
 *            Second version to be compared
 * @return Boolean value representing if first version is higher than the second or not
 */
private boolean isHigherVersion(String version, String version2) {
    if (version.equals(version2)) {
        return false;
    }
    int[] versionArray = versionNumberArrayBuilder(version);
    int[] version2Array = versionNumberArrayBuilder(version2);
    int versionLength = versionArray.length <= version2Array.length ? versionArray.length : version2Array.length;
    for (int i = 0; i < versionLength; i++) {
        if (versionArray[i] > version2Array[i]) {
            return true;
        } else if (versionArray[i] < version2Array[i]) {
            return false;
        }
    }
    return versionArray.length >= version2Array.length;
}

private int[] versionNumberArrayBuilder(String version) {
    return Arrays.stream(version.split("\\.")).mapToInt(Integer::parseInt).toArray();
}
```

FIGURE 23. Checking if the given version is higher than the compared version

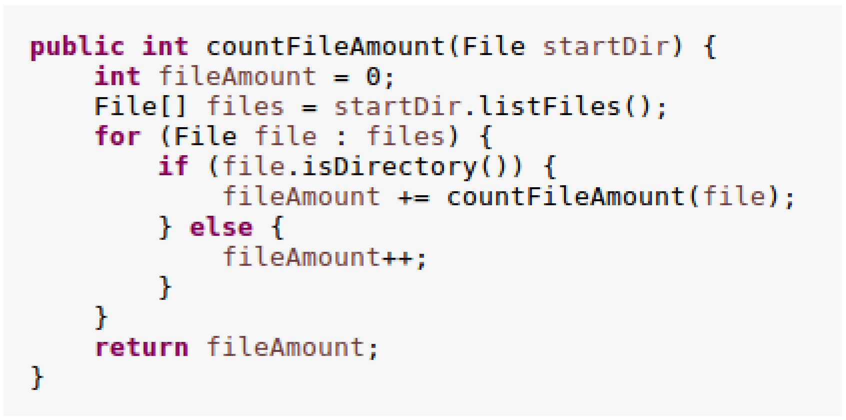
The result of this code block is a functioning system to retrieve the current version information, the highest version's information on the server, and comparing those effectively. What the method does is after splitting the version strings to integer arrays comparing the integers in different versions from highest to lowest. Whenever a higher version is found, the result is passed back, e.g. comparing versions 6.2.3 and 6.1.7, returns already an answer when comparing integers 2 and 1, without the need to compare 3 and 7 anymore.

6 USER INTERFACE NOTIFICATIONS

In software, one important thing is to let the user know what is happening behind the scenes appropriately. This is achieved by printing messages to the user, not only with relevant and useful information, but also by appropriate amounts.

6.1 Counting and printing the progress

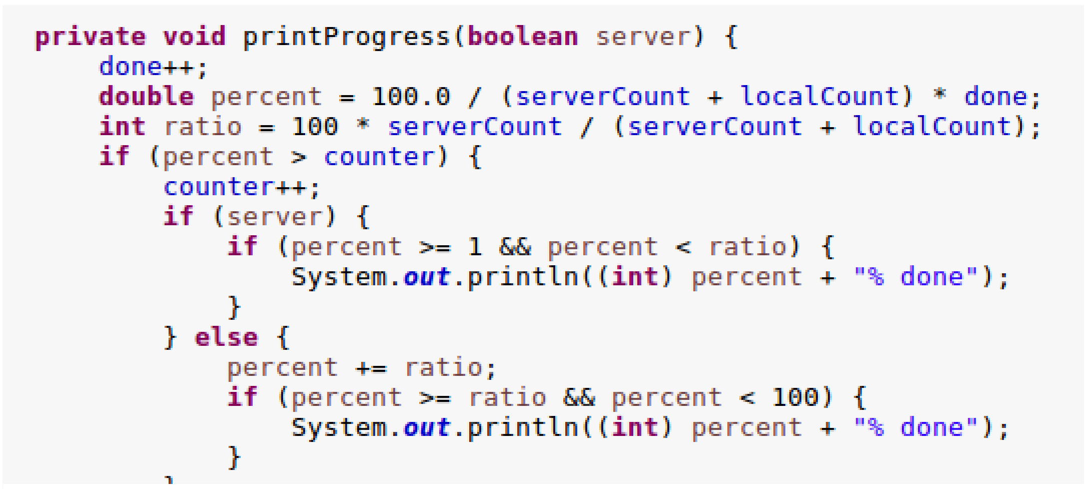
The file amount if calculated by iterating through the given directory, and on all files adding one to the fileAmount variable. If the current file is a directory, then the directory contents are counted, so all sub directories including all files are counted as well. The resulting integer is the amount of all files recursively in the file.



```java
public int countFileAmount(File startDir) {
    int fileAmount = 0;
    File[] files = startDir.listFiles();
    for (File file : files) {
        if (file.isDirectory()) {
            fileAmount += countFileAmount(file);
        } else {
            fileAmount++;
        }
    }
    return fileAmount;
}
```

FIGURE 24. Counting file amount

The progress is printed on every percent advanced, but only when the percent value has not been printed yet. With less than 100 files, the progress is only printed as many times as there is a file. With more than 100 files, the progress is printed only 100 times, not the amount of the files times.



```java
private void printProgress(boolean server) {
    done++;
    double percent = 100.0 / (serverCount + localCount) * done;
    int ratio = 100 * serverCount / (serverCount + localCount);
    if (percent > counter) {
        counter++;
        if (server) {
            if (percent >= 1 && percent < ratio) {
                System.out.println((int) percent + "% done");
            }
        } else {
            percent += ratio;
            if (percent >= ratio && percent < 100) {
                System.out.println((int) percent + "% done");
            }
```

FIGURE 25. Printing progress

7 CONCLUSION AND DISCUSSION

In conclusion updating a piece of software has much more behind the scenes than the user might notice. Despite this, the updating must be done regularly to maintain the usability, integrity and most importantly the security of the application. There are multiple ways to perform updating, varying not only from frequency of updating but also to way of changing files and handling special occasions.

Special situations can arise from customisations of the software, different user interfaces and different platforms. In this case, some specialities appeared from customer specialisations.

At this stage of the project, it is easy to see how the whole project could have been planned a bit more specifically. Especially, in order to reduce wasted time with refactoring parts that was built in wrong order or in wrong way.

Altogether, the project was successful and helps the company to reduce time and cost of driving updates. However, there remain some things to improve, such as file transferring and more unified way of proofing the version. The next part in the project will be to create a SSH Client to get the checksum, using internally SFTP protocol to transfer the updated files. That part of the project was too big to include in this thesis with such limited time scale though.

REFERENCES

Elmblad, S. 2017. Software Update Definition, Cited 9.11.2017,
https://www.thebalance.com/what-is-a-software-update-vs-software-upgrade-1294256

Fisher, T. 2017. What is a Checksum? Cited 12.11.2017,
https://www.lifewire.com/what-does-checksum-mean-2625825

Fisher, T. 2017. What is a Patch? Hot Fixes vs Patches. Cited 21.11.2017,
https://www.lifewire.com/what-is-a-patch-2625960

Harvard University Collection of Historical Scientific Instruments. 2017. A Legendary Bug. Cited 21.10.2017,
http://sites.harvard.edu/~chsi/markone/language.html

Harvard University Collection of Historical Scientific Instruments. 2017. The “Patch”. Cited 21.10.2017,
http://sites.harvard.edu/~chsi/markone/language.html

Rouse, M. 2005. Tomcat, Cited 12.11.2017,
http://www.theserverside.com/definition/Tomcat

Techopedia. 2017. What does Bug Fix mean? Cited 22.10.2017,
https://www.techopedia.com/definition/18105/bug-fix

Wikipedia. 2017. History, Patch (computing). Cited 30.09.2017,
https://en.wikipedia.org/wiki/Patch_(computing)

Zandbergen, P. 2017. Methods and Functions, Cited 12.11.2017,
http://study.com/academy/lesson/oop-object-oriented-programming-objects-classes-interfaces.html