# File Backup

Max Weise

Last Update: December 21, 2021

**System Requirements and Documentation**

# Contents

# 1 System Requirements

This chapter contains a comprehensive list of all requirements for the system. The order of the requirements is arbitraryly selected and does not indicate the importance of a specific requirement in comparison to any other.

## 1.1 Functional Requirements

All rqequirements which deal with the functionality of the programm will be listed in this section.

- All files should be copied from a given source directory to a given destination directory.
- (If specified) All files that math any of the given fileextentions should be copied from a given source directory to a given destination directory.
- The structure of the copied directory has to be preserved.
  - Exception: If a subdirectory of the copied source directory does not contain any (matching) files it may be ignored and not be included in the destination directory.
- There should not be any loss of data, except the files, which do not match the extentions specified by the user.

# 2 System Architecture

This chapter contains an overview over the architecture of this project. Classes and Method sigantures use the same notation as in the code, which follows Python PEP8.

## 2.1 Description of Concept

### 2.1.1 Classes

**Comment:** Please note, that valiable and attribute names may differ from the actual implementation found in the code and may be abbreviated or use a different word, while still keeping the meaning of the original. This should not be an issue, as class names and method signatures are the same as in the code.
The keycomponents of this scirpt are the so called *Backup Classes*:

- `Backup(ABC)`
- `File_Backup(Backup)`
- `File_Type_Backup(Backup)`

Each of those classes is contained in it's own file. By using inheritence, it is possible to use the *strategy pattern*, which lets us use different implementations of a specific method based on the object that calls the method. In this case, the abstract class *Backup* defines an abstract method `backup(self) -> None` which is implemented in the subclasses according to their specific task. In the `main.py` file, it is possible to call this method all instances of classes that inherit from *Backup*.

### 2.1.2 Programflow

The behaviour of the program is defined in the *main.py* file. Is responsible for setting up loggers and the argumentparser in their respective functions. It will provide a tkinter root object, which can be used to attach GUI objects to, though this should

not be done, as the root window will be withdrawn. It is best to define further GUI windows in its own class and add objects to the main function as needed.
After setup, the programm allways follows predefined steps:

1. Ask the user source and destination directories
2. Create an instance of the Backup class. Which one it will be is dependant on the procedure the user wants to run
3. Ask the user which filetypes they want to backup (this is optional and only done if the procedure requires it)
4. Call the backup method. After this, the program terminates

## 2.2 Component Overview

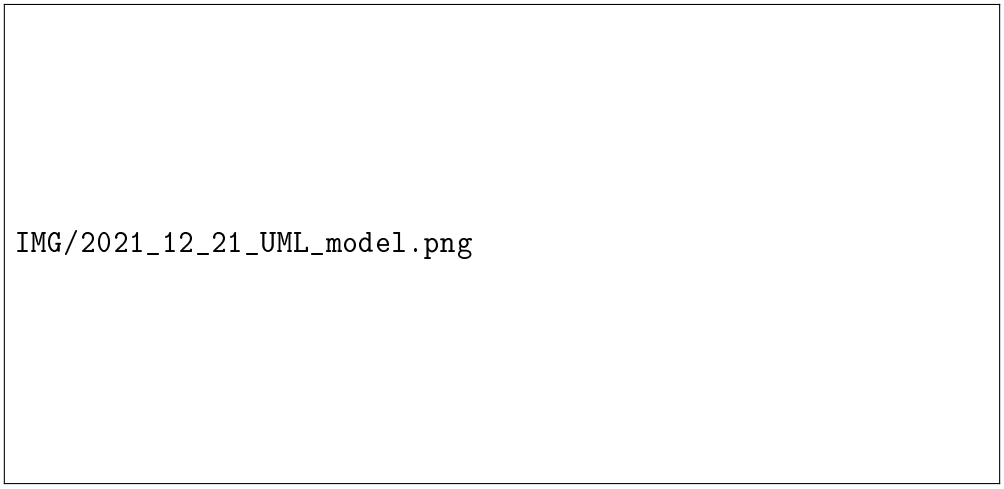


Figure 2.1: An UML overview over the classes used in the main script. On the left are all strategies used for backup. The GUI class on the bottom right is not completely shown due to readability.

# 3 Technical Code Overview

# 4 Styleguide

# 5 Usermanual