Source Code:

```
from abc import abstractmethod, ABC


class FileHandlingInterface(ABC):

    @abstractmethod
    def open(self) -> None:
        pass

    @abstractmethod
    def close(self) -> None:
        pass


class ReadOnly(FileHandlingInterface):

    def __init__(self,  file_name: str) -> None:
        self.file_name = file_name
        # this is only here because it's technically better to define the
instance methods in the initializer
        self.file = None

    def open(self) -> None:
        print("Since you're using reading mode, you'll get an error if the
file doesn't exist")
        self.file = open(self.file_name, "w")
        print(f"file {self.file_name} opened successfully")

    def close(self) -> None:  # handling the exception where the file is not
opened before usage

        try:
            self.file.close()
        except AttributeError:  # handling the exception where the file is
not opened before usage
            print("You have not opened a file yet. try using open() method.")

    def read_file(self) -> None:

        try:
            return self.file.read()
        except AttributeError:  # handling the exception where the file is
not opened before usage
            print("You have not opened a file yet. try using open() method.")

    def readline_file(self) -> None:

        try:
            return self.file.readline()
        except AttributeError:  # handling the exception where the file is
not opened before usage
```

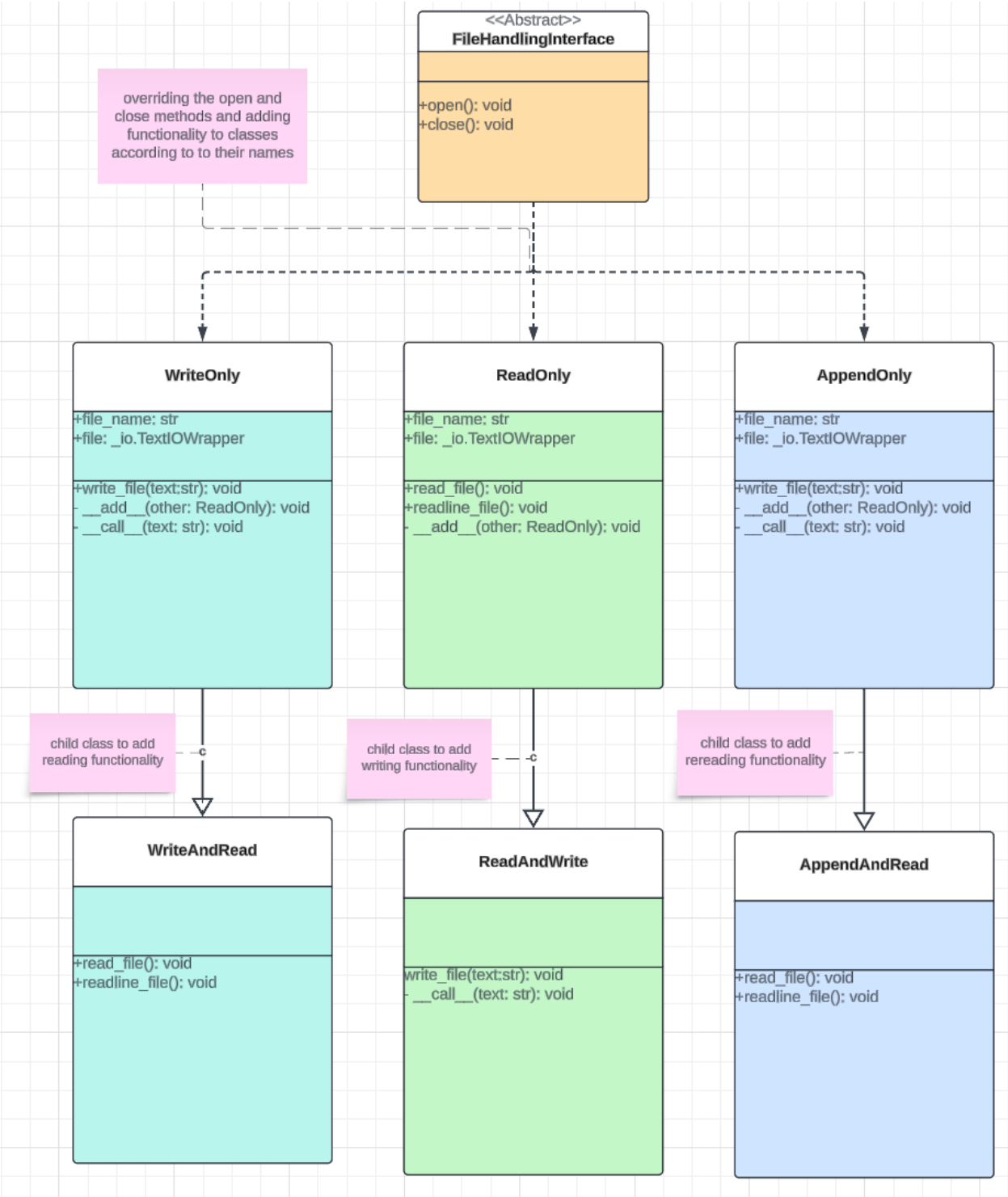
```
            print("You have not opened a file yet. try using open() method.")

    def __add__(self, other) -> None:

        try:
            self.self_text = self.file.read()
            self.other_text = other.file.read()

        except AttributeError:  # handling the exception where the file is
not opened before usage
            print("You have not opened a file yet. try using open() method.")

        with open("result.txt", "w") as result:
            result.write(self.self_text + self.other_text)


class ReadAndWrite(ReadOnly, FileHandlingInterface):

    def __init__(self) -> None:  # this is only here because despite the
project description
        # it's technically better to define the instance variables in the
initializer
        super().__init__()

    def write_file(self, text: str) -> None:

        try:
            return self.file.write(text)
        except AttributeError:  # handling the exception where the file is
not opened before usage
            print("You have not opened a file yet. try using open() method.")

    def __call__(self, text:str) -> None:
        self.write_file(text)


class WriteOnly(FileHandlingInterface):

    def __init__(self,  file_name: str) -> None:
        self.file_name = file_name
        # this is only here because it's technically better to define the
instance methods in the initializer
        self.file = None

    def open(self) -> None:
        print("Since you're' using writing mode, a file will be created if it
doesn't already exist")
        self.file = open(self.file_name, "w")
        print(f"file {self.file_name} opened successfully")

    def close(self) -> None:  # handling the exception where the file is not
opened before usage

        try:
            self.file.close()
        except AttributeError:  # handling the exception where the file is
not opened before usage
```

```
            print("You have not opened a file yet. try using open() method")

    def write_file(self, text: str) -> None:

        try:
            return self.file.write(text)
        except AttributeError:  # handling the exception where the file is
not opened before usage
            print("You have not opened a file yet. try using open() method.")

    def __add__(self, other) -> None:

        try:
            self.self_text = self.file.read()
            self.other_text = other.file.read()

        except AttributeError:  # handling the exception where the file is
not opened before usage
            print("You have not opened a file yet. try using open() method.")

        with open("result.txt", "w") as result:
            result.write(self.self_text + self.other_text)

    def __call__(self, text: str) -> None:
        self.write_file(text)


class WriteAndRead(WriteOnly, FileHandlingInterface):

    def __init__(self) -> None:  # this is only here because despite the
project description
        # it's technically better to define the instance methods in the
initializer
        super().__init__()

    def read_file(self) -> None:

        try:
            return self.file.read()
        except AttributeError:  # handling the exception where the file is
not opened before usage
            print("You have not opened a file yet. try using open() method.")

    def readline_file(self) -> None:

        try:
            return self.file.readline()
        except AttributeError:  # handling the exception where the file is
not opened before usage
            print("You have not opened a file yet. try using open() method.")


class AppendOnly(FileHandlingInterface):

    def __init__(self,  file_name: str) -> None:
        self.file_name = file_name
        # this is only here because it's technically better to define the
```

```
instance methods in the initializer
        self.file = None

    def open(self) -> None:
        print("Since you're' using appending mode, a file will be created if
it doesn't already exist")
        self.file = open(self.file_name, "w")
        print(f"file {self.file_name} opened successfully")

    def close(self) -> None:  # handling the exception where the file is not
opened before usage

        try:
            self.file.close()
        except AttributeError:  # handling the exception where the file is
not opened before usage
            print("You have not opened a file yet. try using open() method.")

    def write_file(self, text: str) -> None:

        try:
            return self.file.write(text)
        except AttributeError:  # handling the exception where the file is
not opened before usage
            print("You have not opened a file yet. try using open() method.")

    def __add__(self, other) -> None:

        try:
            self.self_text = self.file.read()
            self.other_text = other.file.read()

        except AttributeError:  # handling the exception where the file is
not opened before usage
            print("You have not opened a file yet. try using open() method.")

        with open("result.txt", "w") as result:
            result.write(self.self_text + self.other_text)

    def __call__(self, text: str) -> None:
        self.write_file(text)


class AppendAndRead(AppendOnly, FileHandlingInterface):

    def __init__(self) -> None:  # this is only here because despite the
project description
        # it's technically better to define the instance methods in the
initializer
        super().__init__()

    def read_file(self) -> None:
        try:
            return self.file.read()
        except AttributeError:  # handling the exception where the file is
not opened before usage
            print("You have not opened a file yet. try using open() method.")
```

```
    def readline_file(self) -> None:

        try:
            return self.file.readline()
        except AttributeError:  # handling the exception where the file is
not opened before usage
            print("You have not opened a file yet. try using open() method.")
```

UML Class Diagram + Notes:



You can also find the diagram here