DD2480 Report for assignment 4

Elin Fransholm (fransho),
Linda Nycander (lindanyc),
Mert Demirsü (demirsu),
Philip Ågren Jahnsson (philipaj),
Vilhelm Prytz (vprytz)

March 2025

1 Project

Name: Click

URL: click

URL forked repo: click-Task4

Click is a package for creating command-line interfaces. It offers an easy way to handle argument and options parsing and automatic help message generation. It is written in *Python*.

2 Onboarding experience

We continued with the same project as in assignment 4. Therefore, we already had some knowledge about the program and how to test/use it, and we had already some experience working with their code. For this assignment, we had to look into the code even more, in order to understand the issues, and the complexity of possible solutions.

3 Click: Architecture and Purpose Overview

Click is a Python library for creating command-line interfaces (CLIs). It is designed to be minimal and composable, offering a straightforward API without excessive input. This design philosophy makes Click both easy to understand and extend, facilitating integration into various projects.

Core Modules:
The *Core* module defines the primary abstractions:

- **Command** represents a single CLI command, encapsulating its name, callback, and a list of parameters.
- **Group** extends **Command** to allow subcommands, enabling hierarchical command structures.
- **Context** manages runtime state and configuration, passing information such as parsed parameters and global settings between commands.
- **Parameter** serves as the base for both **Option** and **Argument**, handling input conversion and validation.

Supporting Modules:

- **Decorators** provide functions (e.g., `@command`, `@group`, `@option`, and `@argument`) that convert regular Python functions into CLI commands by attaching metadata like help text and parameter types.
- **Exceptions** define custom error classes (e.g., **ClickException**, **UsageError**, **BadParameter**) for handling errors and displaying user-friendly messages.
- **Formatting** contains the **HelpFormatter** class and utilities to generate and format usage messages and help pages.
- **Globals** and **Utils** manage global state (such as the current context) and provide helper functions (for instance, to echo messages or handle file operations).
- **TermUI** handles terminal input/output operations, including prompts, progress bars, and ANSI color support.
- **Types** defines various parameter types (like **INT**, **FLOAT**, **BOOL**, and **UUID**) for input conversion and validation.
- **Parser** performs low-level parsing of command-line options and arguments.
- **CLI Runner** offers a testing harness (with classes such as **CliRunner** and **Result**) to invoke Click commands in an isolated environment.
- **Shell Completion** supports auto-completion for different shells.

Design Principles:

- *Command Definition & Parsing*: The core modules manage command definitions and parameter parsing, efficiently dispatching to callbacks.
- *Error Handling*: Custom exceptions catch issues early and provide consistent, helpful feedback.

- *Help Generation*: The formatting module automatically produces usage and help text, reducing the need for manual documentation.
- *Compatibility*: Careful management of I/O streams and ANSI color codes ensures that Click works reliably across platforms.

UML Diagram Overview:
The architecture of Click is depicted in three sub-diagrams:

- **Diagram 1** (Figure 1): Shows the *Core* modules, along with the *Decorators* and *Exceptions* that interact closely with them.
- **Diagram 2** (Figure 2): Illustrates *TermUI*, *Types*, *Parser*, and *Formatting*, emphasizing the user interaction, input conversion, and help generation.
- **Diagram 3** (Figure 3): Focuses on *Globals*, *Utils*, *CLI Runner*, and *Shell Completion*, which support the core runtime and testing functionalities.

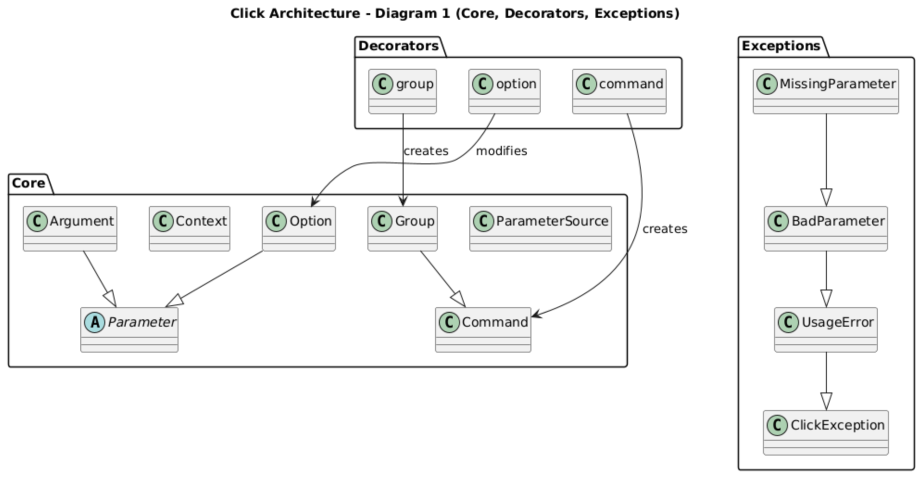

Figure 1: Diagram 1: Core, Decorators, and Exceptions.

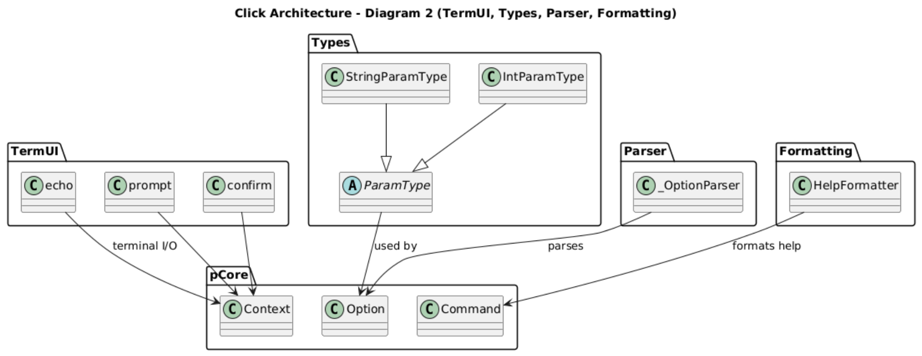

Figure 2: Diagram 2: TermUI, Types, Parser, and Formatting.

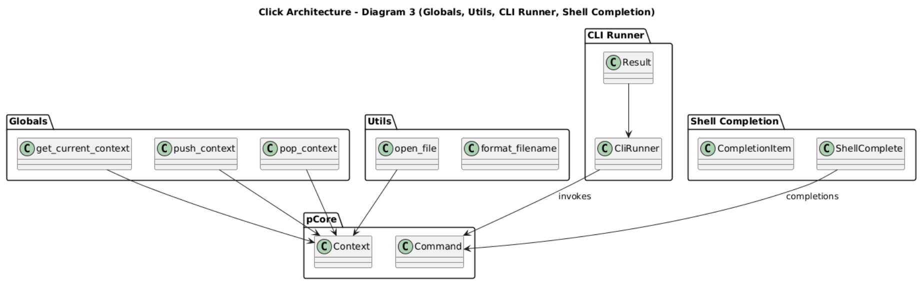

Figure 3: Diagram 3: Globals, Utils, CLI Runner, and Shell Completion.

4 Effort spent

Hours spent on each task	Elin	Linda	Philip	Vilhelm	Mert
plenary discussions/meetings	2	2	2	2	2
discussions within parts of the group	3	3	3	3	2
reading documentation	0.5	0.5	1	1.5	1.5
configuration and setup	0.5	0.5	0.5	0.5	0.5
analyzing code/output	1	1	2	2	3
writing documentation	5	5	3.5	2.5	7
writing code	7	7	8	8	2.5
running code	1.5	1.5	1	1	1.5
total time spent	20.5	20.5	21	20.5	20

Table 1: Hours spent on each part of the task

5 Overview of issue(s) and work done.

5.1 Issue: 2781

Title: [Feature Request] Allow to to emit colored click.secho() text to a pipeline.

URL: pallets#2781

5.1.1 A summary of the issue:

To summarize, the issue is about a feature request to allow forcing to allow color output from Click, even when the output is not a terminal, and for example, a pipe. This is useful when you are piping one Python program to another, where both are running Click, and you want to force the program to emit ANSI color codes.

5.1.2 The scope of the issue:

The scope of the issue affects output functions from Click. This can be quite small, but the tricky part is to figure out where to handle the logic. Since Click already strips ANSI color codes, it is best to start where the code currently strips these color codes so that logic can be implemented to override this using global policies.

5.2 Issue: 2419

Title: [Feature] ClickException message without color would be helpful

URL: pallets#2419

5.2.1 A summary of the issue:

To summarize, this issue is about a feature request, regarding an exception message. The feature request is that the plain text (without color and ansi-code) of the exception message should be available.

5.2.2 The scope of the issue:

The scope of this issue can be pretty small, but it depends on the design decisions. With an easy solution, it can be focused around one function and very few lines of code added. However, some research is required to know what to do and where.

5.3 Issue: 2764

Title: [Feature Request] Provide suggestions for misspelled subcommands

URL: pallets#2764

5.3.1 A summary of the issue:

This issue states that when click is used, if a subcommand is wrongly given, then click does not provide a suggestion if a command exists that is similar in spelling to the command the user put in. The change made align how spelling mistakes are handled with commands so that it works the same way as when misspelling options.

5.3.2 The scope of the issue:

The scope of the issue is fairly small as its only feature is to add some lines to a message string when an error is raised. This can be done with fairly few lines of code. However, which lines of code in which function can be a bit tricky, and figuring out how to get access to all available commands in a string format does also require some research.

5.4 Issue: 2809

Title: @click.Option with 'hide_input' = True (used for password) and custom 'type' parameter doesn't show custom error message.

URL: pallets#2809

5.4.1 A summary of the issue:

When using @click.Option with hide_input=True and a custom type parameter for password validation, a click.BadParameter exception does not display the expected custom error message. Instead, it shows a standard message.

6 Requirements for the new feature or requirements affected by functionality being refactored

6.1 Issue 2781

6.1.1 Requirements:

- **R1: Global Policy Configuration** Global API that sets default color behavior across entire application.
- **R2: Override Color Policy Per-Invocation** Allow to override the global setting.
- **R3: Add enum for ColorPolicy** Add a enum for ColorPolicy that has the states in the issue, `ALWAYS_KEEP`, `ALWAYS_STRIP` and `ALWAYS_AUTO`.
- **R4: Function to retriebve ColorPolicy** Create a new functoin that retrieves the current global ColorPolicy.

- **R5: Consistency with Subprocess** When an application using Click spawns subprocess that also uses Click, the color policy should be able to be shared or passed.
- **R6: All current tests should pass:** When running the existing tests after the issue has been fixed, all current tests should pass.
- **R7: All new classes and functions should be tested:** All new classes and functions implemented should be tested thoroughly and have a high branch coverage.

6.1.2 Test Cases:

- **T1**: Default Behvaior, where we are in AUTO mode. Test that when stdout is TTY, terminal has color, and check that when stdout is to pipe or file, it does not have any ANSI color codes.
- **T2**: Use `ALWAYS_KEEP` policy, check that output has ANSI colors, both in terminal and in pipe.
- **T3**: Use `ALWAYS_STRIP` policy, check that output does not have any ANSI colors, even when in a TTY.

6.1.3 Development plans:

1. Write test cases.
2. Add enum `ColorPolicy`.
3. Update `should_strip_ansi` in `_compat.py` which should handle to resepct the global override policies.

6.2 Issue 2419

6.2.1 Requirements:

- **R1: Message stores plain text** The .message property should store the text version of the exception message with ansi-encoding.
- **R2: ClickException returns plaint text** ClickException should return a plain text message when converted to a string (str(exception)). This means that the message should not include the ansi-code with the styling information.

6.2.2 Test Cases:

- *test_exception_str_plain_text_color* tests that the str()-function returns the plain text of the exception message, when the ansi-code includes a color.

- *test_exception_str_plain_text_style* tests that the str()-function returns the plain text of the exception message, when the ansi-code does not include color, but includes other styling (like bold font).
- *test_exception_message_keeps_encoding* tests that exception.message still includes the correct ansi-code.

6.2.3 Development plans:

- Write test cases.
- Change function __str__ in class ClickException so that it returns the plain text of the exception message.

6.3 Issue 2764

6.3.1 Requirements:

- **R1:** Current logic used for misspelling handling for options should be extended to subcommands.
- **R2:** Handle scenario where multiple commands are suggested, in the same way the current implementation for options is implemented.
- **R3:** Consistent UX, text should be formatted in the same way as options. Meaning, use the same words so the UX is consistent.
- **R4:** There should be tests that cover the newly written code.

6.3.2 Test Cases:

- **T1**: `test_command_suggestions_matches_found`: Test scenario where there is a command that is similar enough to the misspelled command so that it suggests the correct command. For example, if there is a command `hello` but the user inputs `hellow` it should suggest `hello` as command.
- **T2**: `test_command_suggestions_no_matches`: Test scenario where there are no similar matches, should then only return an error message as before implemented changes, thus returning no suggestions. For example, if the use inputs `xyz` and there is no command close to this command, it should just exit without suggesting another command.
- **T3**: `test_command_suggestions_multiple_matches`: Test scenario where there are multiple, equally close, suggestions, so the program suggests all of them. For example, if there is a command `hello` and a command `hallo` and the user inputs `hellow`, it should suggest `(Possible commands: hallo, hello)`.

6.3.3 Development plans:

1. Identify requirements

2. Implement test cases specified above, ensure they fail.
3. Identify which functions are responsible for error handling of commands.
4. Implement suggestion in `Group.resolve_command` in `src/click.py`, in the same way it is implemented for options using `difflib`.

6.4 Issue 2809

6.4.1 Requirements:

- **R1:** Choosing a custom type for @click.option should not result in the default exception message when a BadParameter exception occurs.
- **R2:** Custom error exception message should be shown even when hide_input is set to true.
- **R3:** Consistent error handling, the updated error handling should not interfere with previous functionalities of click.
- **R4:** Prompt should allow retry without restarting the command. (Not tested yet.)

6.4.2 Test cases:

- **T1**: *test_custom_error_message*: Implement a custom type and set @click.option with hide_input=true and the custom type. Also set the parameter to a value that results in a BadParameter exception and check that the custom exception message is returned.
- **T2**: *test_custom_error_message_password*: Implement a custom type and set @click.option with hide_input=true and the custom type. Also set the parameter to a value that does not raise a BadParameter exception and check that the custom exception message is not returned.

6.4.3 Development plans:

1. Identify requirements.
2. Write relevant test cases that fail.
3. Update termui.py to show custom error message when the UsageError is a BadParameter exception and *hide_input=true.*
4. Write test case for R4. (Not finished.)

7 Code changes

7.1 Patch

7.1.1 Issue 2781

Here is the patch: link

7.1.2 Issue 2419

Here is the patch: link

7.1.3 Issue 2764

Here is the patch: link

7.1.4 Issue 2809

Here is the patch: link

8 Test results

The test logs can be found in the Appendix 12.1 and 12.2.

9 UML class diagram and its description

9.1 Issue 2781 UML Diagram

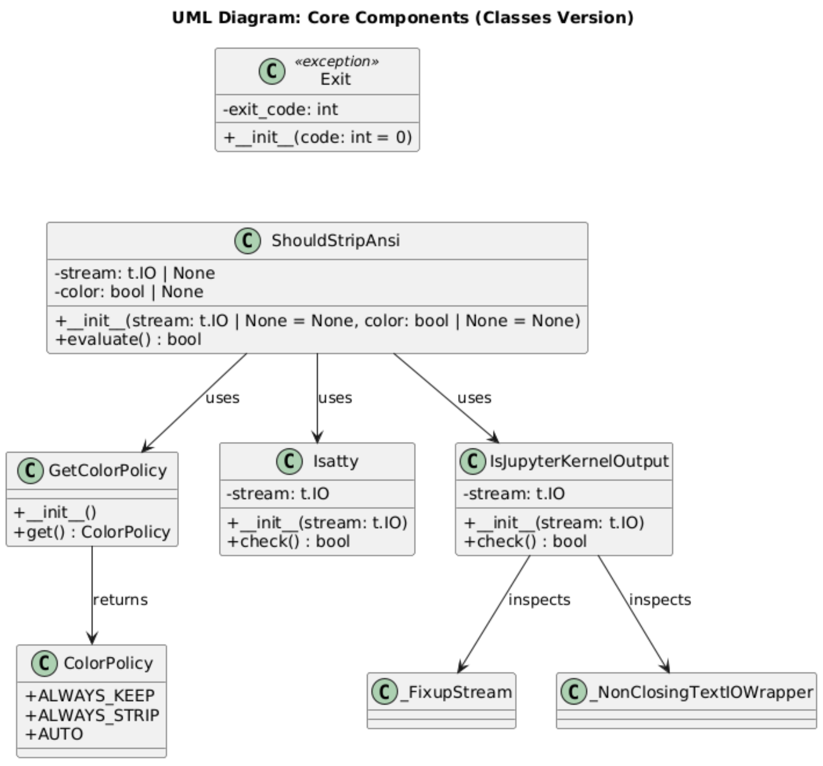

Figure 4: The UML class-diagram of issue 2781.

9.2 Issue 2419 UML Diagram

Figure 5 shows the UML class-diagram of the closest related classes. ClickException, that changes have been made to, inherits from Exception and FileError and UsageError inherits from ClickException. There some more classes that inherits from these but only level up and down is included in the diagram. Both Editor and Command raises ClickExceptions.

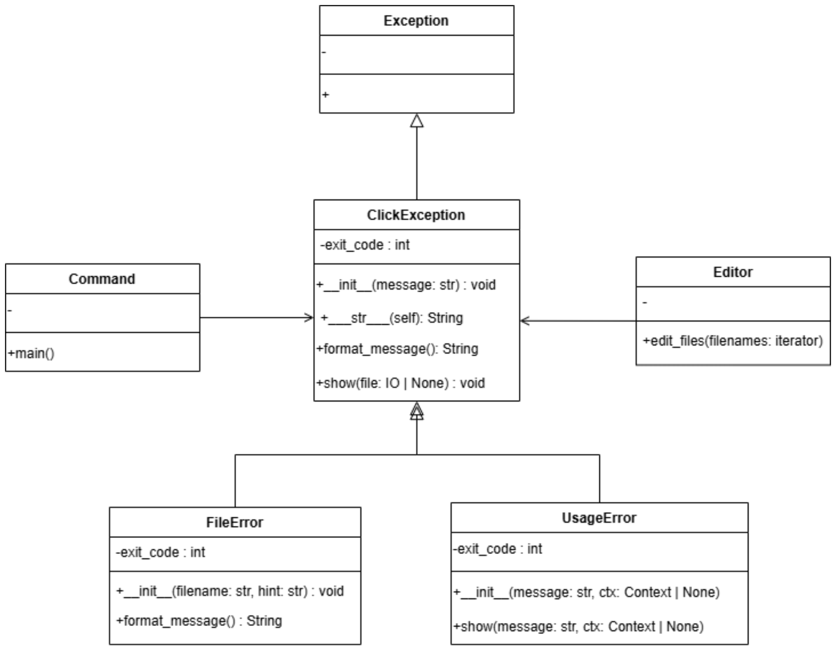
Exception
-
+
ClickException
-exit_code : int
+__init__(message: str) : void
+___str___(self): String
+format_message(): String
+show(file: IO | None) : void
Command
-
+main()
Editor
-
+edit_files(filenames: iterator)
FileError
-exit_code : int
+__init__(filename: str, hint: str) : void
+format_message() : String
UsageError
-exit_code : int
+__init__(message: str, ctx: Context | None)
+show(message: str, ctx: Context | None)

Figure 5: The UML class-diagram of issue 2419.

9.3 Issue 2764

Figure 6 shows the UML class-diagram of the closest related classes to the issue in question. The changes made were primarily in the `resolve_command` method in the `Group` class. This method is only called from places in the codebase outside classes and also within the class itself, in the `invoke` method.

The only relevant relationships that exist to other objects are:

1. A generalization/inheritance from `Group` to `Command`, since `Group` is an instance of `Command`.
2. A generalization/inheritance from `CommandCollection` to `Group`, since `CommandCollection` is an instance of `Group`.
3. A composition relationship from `Group` to `Command`, since `Group` has an attribute `commands` which is type of multiple `Command` in a list.
4. A composition relationship from `CommandCollection` to `Grop`, since `CommandCollection` has an attribute `sources` which is type of multiple `Group` in a list.

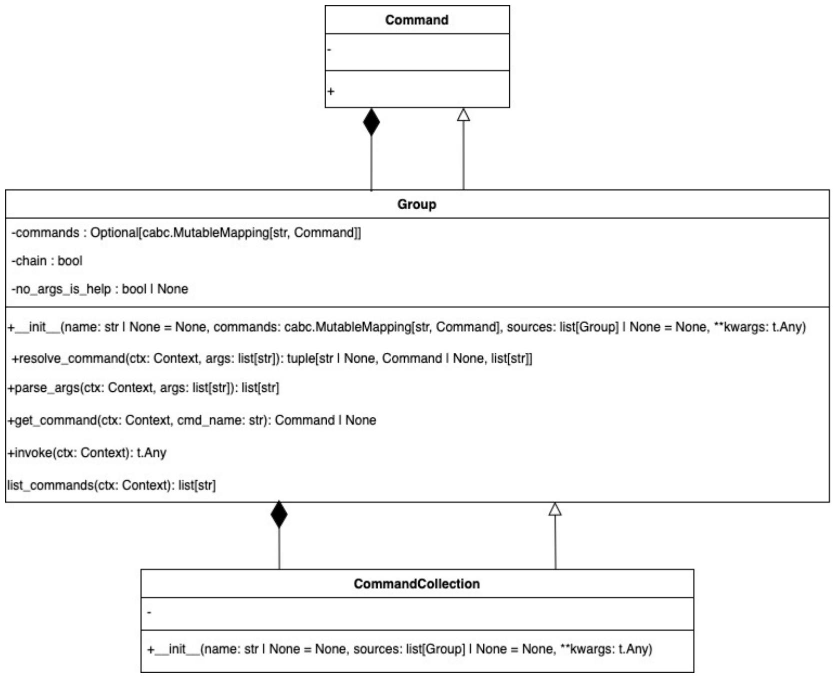
Command
-
+
Group
-commands : Optional[cabc.MutableMapping[str, Command]]
-chain : bool
-no_args_is_help : bool | None
+__init__(name: str | None = None, commands: cabc.MutableMapping[str, Command], sources: list[Group] | None = None, **kwargs: t.Any)
+resolve_command(ctx: Context, args: list[str]): tuple[str | None, Command | None, list[str]]
+parse_args(ctx: Context, args: list[str]): list[str]
+get_command(ctx: Context, cmd_name: str): Command | None
+invoke(ctx: Context): t.Any
list_commands(ctx: Context): list[str]
CommandCollection
-
+__init__(name: str | None = None, sources: list[Group] | None = None, **kwargs: t.Any)

Figure 6: The UML class-diagram of issue 2764.

9.4 Issue 2809

Figure 7 shows the UML-diagram with the classes most closely related to the function updated. Since the function edited is a standalone function that is not inside a class it is drawn as a function. The only class that calls on this function is Option. Option's subclasses inherits this functionality and are therefore also related. The *hide_input* parameter and the *type* parameter are the only parameters that affect the newly added functionality. The *type* parameter can be of datatype *ParamType* which is another class.

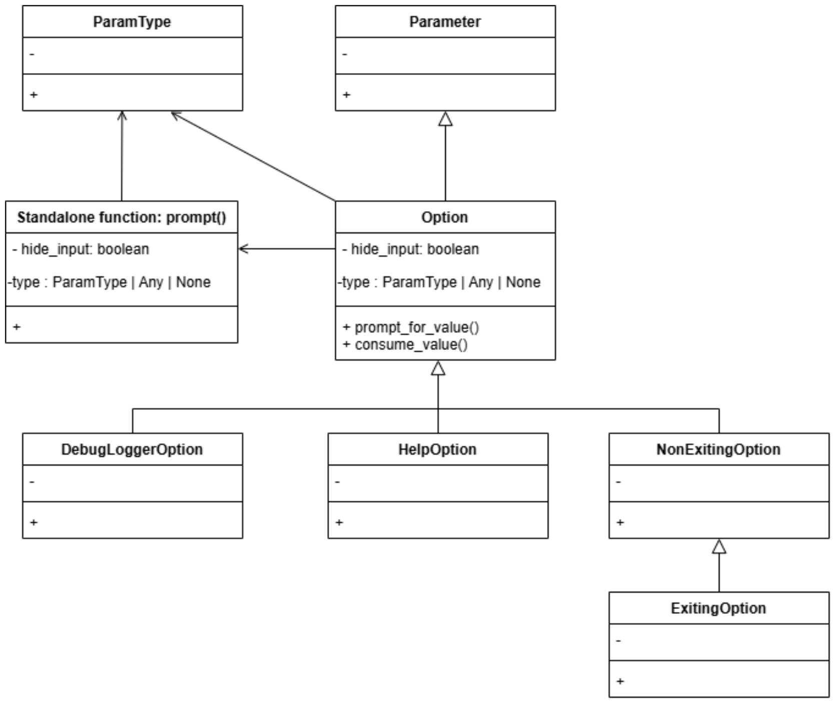

Figure 7: The UML class-diagram of issue 2809.

9.5 Key changes/classes affected

9.5.1 Issue 2781

An enumerator class `ColorPolicy` is created in globals.py, with an instance of this class created as a global object being created in order to control the color policy. The function `should_strip_ansi()` is then changed so that it checks the global color policy object when choosing wether or not to strip color information.

9.5.2 Issue 2419

The class affected is ClickException, and more specifically the function __str__. There are a two subclasses of ClickException which are affected as well and ClickException is a sub-class of the built-in class Exception. In two classes, an instance of the class is created, and therefore those are also affected.

9.5.3 Issue 2764

The class affected is `Group` in core.py, and specifically the method `resolve_command()`.

9.5.4 Issue 2809

The function edited for issue 2809 is a standalone function and does not reside inside a class. Therefore, only this function is affected and also other classes that call on this function. Option is the only class that directly uses the function, *prompt()*. The subclasses of Option inherit the call on the function *prompt()* and are therefore also affected, they are DebugLoggerOption, HelpOption and NonExcitingOption. Futhermore there are a few subclasses from these classes.

10 Overall experience

Our main take-away is that it is complicated to integrate a code change (like a new feature) with the existing code base. A lot of time and effort is put into just understanding what should be done and where in the code to work in. This is especially difficult when working with a new code base, that you have never seen before.

Beyond these technical challenges, our work was critically evaluated in the light of current software engineering practices using the SEMAT kernel framework. Specifically, we focused on three core alphas:

- **Requirements:** We ensured that every modification was rigorously validated against the functional requirements of the project. By designing comprehensive test cases and verifying that all expected behaviors were preserved or enhanced, we addressed the *Requirements* alpha and confirmed that our changes accurately fulfill the intended functionality.
- **Opportunity:** Our process involved identifying and seizing opportunities for improvement. For instance, resolving issues such as the handling of ANSI color codes and providing custom error messages not only remedied existing limitations but also significantly enhanced the overall usability of the tool. This approach demonstrates how we leveraged the *Opportunity* alpha to deliver added value to the project.
- **Software System:** Recognizing the importance of the system's overall architecture, we integrated our changes into Click's modular design in a way that maintained system integrity and future extensibility. As the new features coexist harmoniously with the core modules, it reflects our commitment to the *Software System* alpha.

11 Documentation

According to the Essence standard, we are currently back at "collaborating" when assessing the team. We welcomed a new member for assignment 3, and therefore we had to somewhat start over with our team and workflow. Now that we have integrated him into the work, we are back at where we were at assignment 2. We are working together and are transparent with what we feel and think, but we still have issues with always being effective and constantly meet our commitments, which are criteria for the next step in team assessment.

12 Appendix

12.1 Test logs before adding code:

```
==========================================================================================================
     test session starts
    ======================================================================================================

platform darwin -- Python 3.12.7, pytest-7.4.4, pluggy-1.0.0
rootdir: /Users/lindanycander/Documents/dev/kth-year-4/se/click-Task4
configfile: pyproject.toml
testpaths: tests
plugins: anyio-4.2.0
collected 651 items

tests/test_arguments.py ........................................ [ 6%]
tests/test_basic.py
    .......................................................... [ 15%]
tests/test_chain.py ..............x [ 17%]
tests/test_command_decorators.py ......... [ 19%]
tests/test_commands.py .......................... [ 23%]
tests/test_compat.py . [ 23%]
tests/test_context.py .......................... [ 27%]
tests/test_custom_classes.py ...... [ 28%]
tests/test_defaults.py ..... [ 29%]
tests/test_formatting.py ................ [ 32%]
tests/test_imports.py . [ 32%]
tests/test_info_dict.py ..................... [ 35%]
tests/test_normalization.py ... [ 36%]
tests/test_options.py
    ..........................................................................................................
     [ 55%]
tests/test_parser.py ....... [ 56%]
tests/test_shell_completion.py ........................................
    [ 63%]
tests/test_termui.py ..........................................
    sssssssssssssssssssss.................. [ 76%]
tests/test_testing.py ........................ [ 79%]
tests/test_types.py ................................s..... [ 86%]
tests/test_utils.py
    .........................................................................................
     [100%]

==========================================================================================================
     628 passed, 22 skipped, 1 xfailed in 5.34s
    ======================================================================================================
```

12.2 Test logs after adding code:

```
=====================================================================================================
     test session starts
    =================================================================================================

platform darwin -- Python 3.12.7, pytest-7.4.4, pluggy-1.0.0
rootdir: /Users/lindanycander/Documents/dev/kth-year-4/se/click-Task4
configfile: pyproject.toml
testpaths: tests
plugins: anyio-4.2.0
collected 663 items

tests/test_arguments.py ......................................... [ 6%]
tests/test_basic.py
    .......................................................... [ 15%]
tests/test_chain.py ..............x [ 17%]
tests/test_command_decorators.py ......... [ 19%]
tests/test_commands.py ............................... [ 23%]
tests/test_compat.py ..... [ 24%]
tests/test_context.py .......................... [ 28%]
tests/test_custom_classes.py ...... [ 29%]
tests/test_defaults.py ..... [ 30%]
tests/test_formatting.py ................. [ 32%]
tests/test_imports.py . [ 32%]
tests/test_info_dict.py ...................... [ 36%]
tests/test_normalization.py ... [ 36%]
tests/test_options.py
    .................................................................................................
     [ 55%]
tests/test_parser.py ....... [ 56%]
tests/test_shell_completion.py ........................................
    [ 63%]
tests/test_termui.py ............................................
    sssssssssssssssssssss..................... [ 76%]
tests/test_testing.py ............................ [ 80%]
tests/test_types.py .................................s..... [ 86%]
tests/test_utils.py
    ..........................................................................................
     [100%]

=====================================================================================================
     640 passed, 22 skipped, 1 xfailed in 1.07s
    =================================================================================================
```