Refactoring guet

going from spaghetti code to rotini code

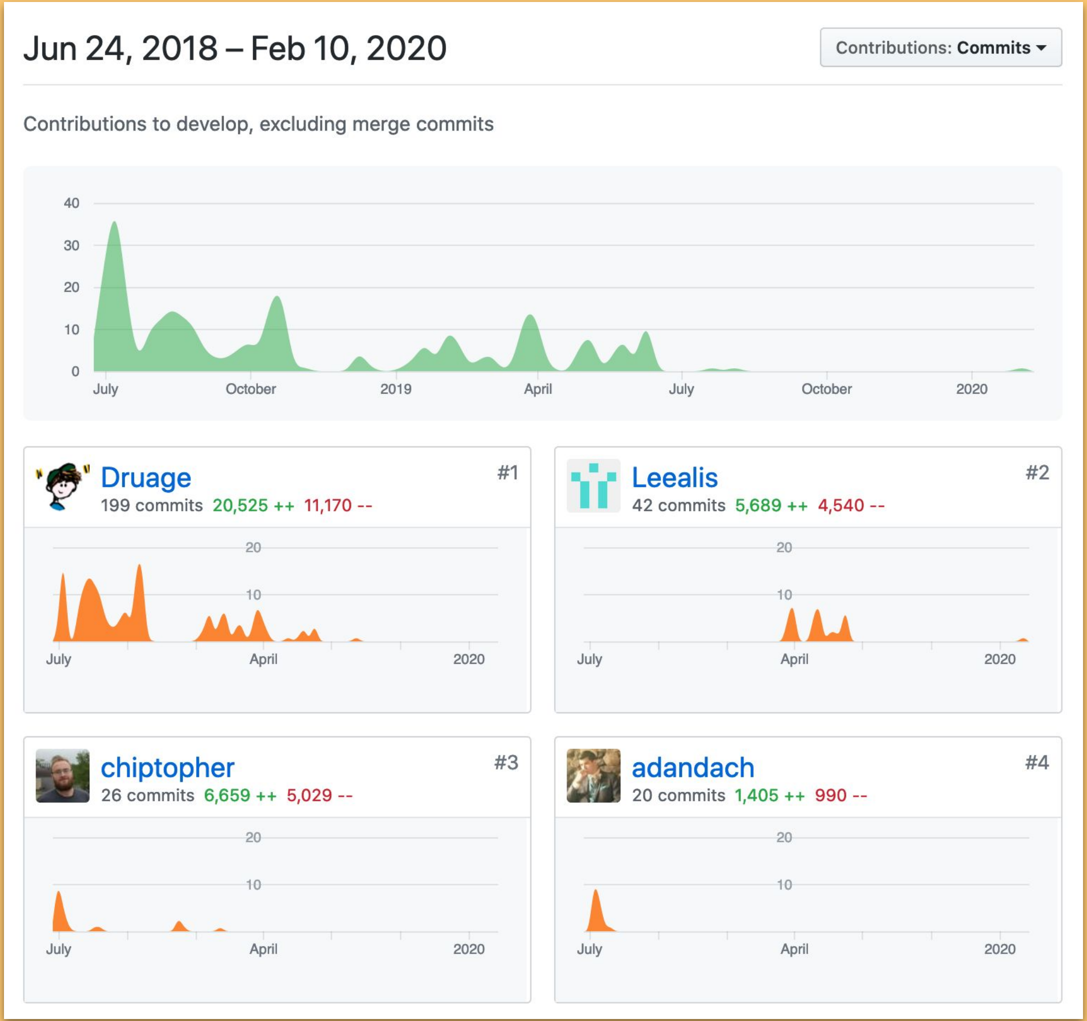
What’s a guet?

"

When pairing, you may want to track each committer's contributions. Using **guet** enables that functionality without changing the normal git workflow.

The guet README

"



Jun 24, 2018 – Feb 10, 2020

Contributions to develop, excluding merge commits

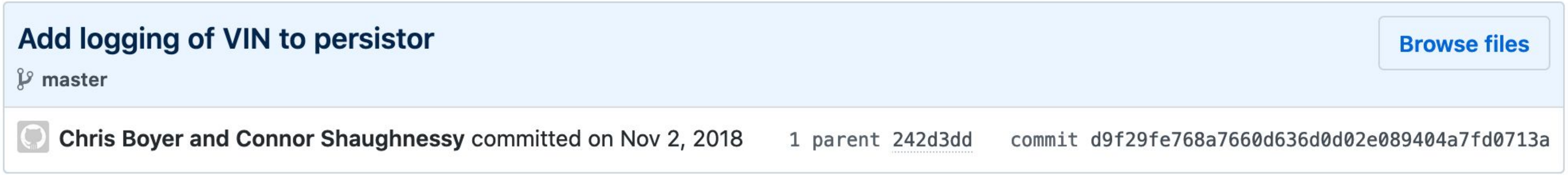
Druage #1
199 commits 20,525 ++ 11,170 --

Leealis #2
42 commits 5,689 ++ 4,540 --

chiptopher #3
26 commits 6,659 ++ 5,029 --

adandach #4
20 commits 1,405 ++ 990 --

```
From d9f29fe768a7660d636d0d02e089404a7fd0713a Mon Sep 17 00:00:00 2001
From: Chris Boyer and Connor Shaughnessy <pair+cboyer17+cshaugh1@ford.com>
Date: Fri, 2 Nov 2018 09:49:37 -0400
Subject: [PATCH] Add logging of VIN to persistor
```



Add logging of VIN to persistor

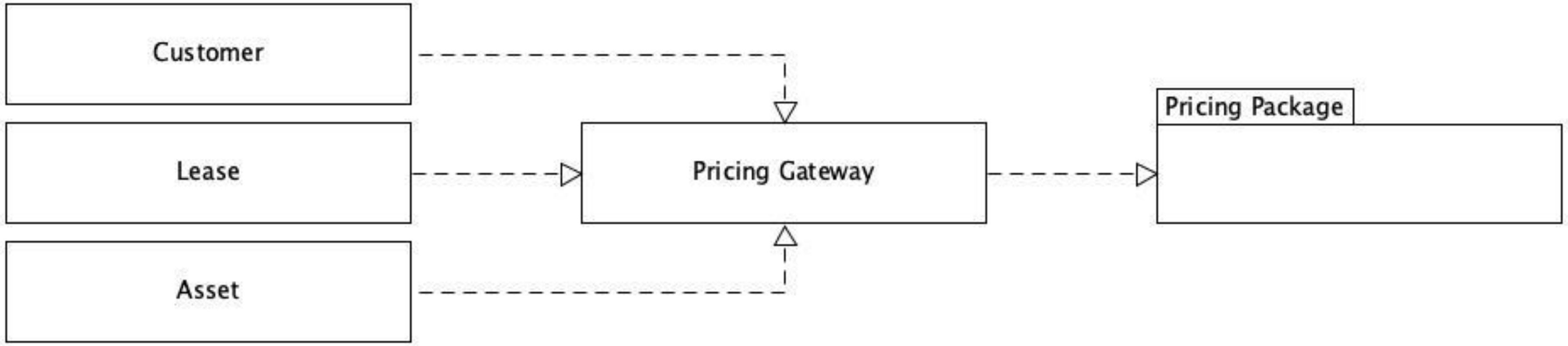
master

Chris Boyer and Connor Shaughnessy committed on Nov 2, 2018 1 parent 242d3dd commit d9f29fe768a7660d636d0d02e089404a7fd0713a

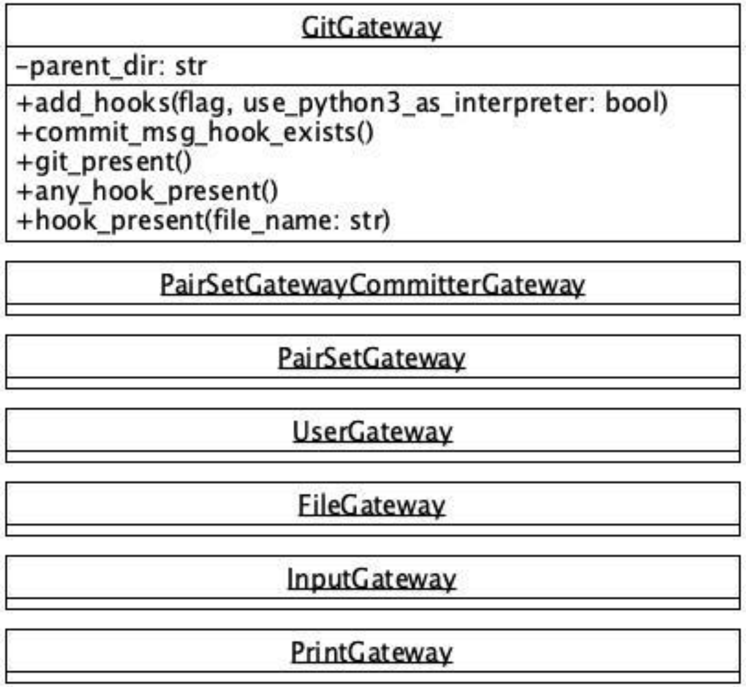
How did guet work before?

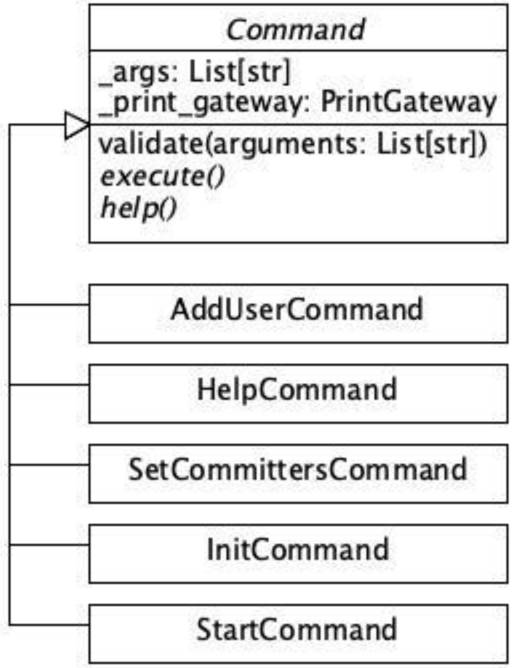
Gateway Pattern

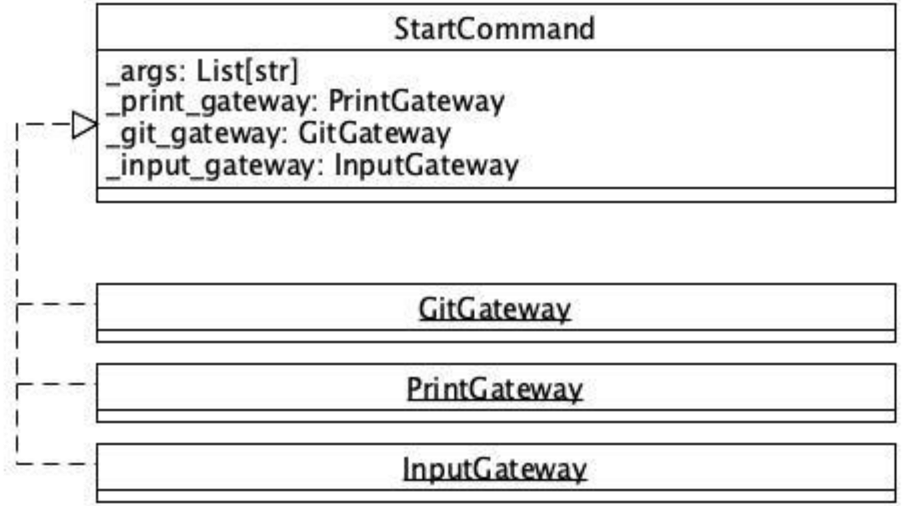
An object that encapsulates access to an external system or resource.



Martin Fowler ***Patterns of Enterprise Architecture***


GitGateway
-parent_dir: str
+add_hooks(flag, use_python3_as_interpreter: bool)
+commit_msg_hook_exists()
+git_present()
+any_hook_present()
+hook_present(file_name: str)
PairSetGatewayCommitterGateway
PairSetGateway
UserGateway
FileGateway
InputGateway
PrintGateway


Command
_args: List[str]
_print_gateway: PrintGateway
validate(arguments: List[str])
execute()
help()
AddUserCommand
HelpCommand
SetCommittersCommand
InitCommand
StartCommand

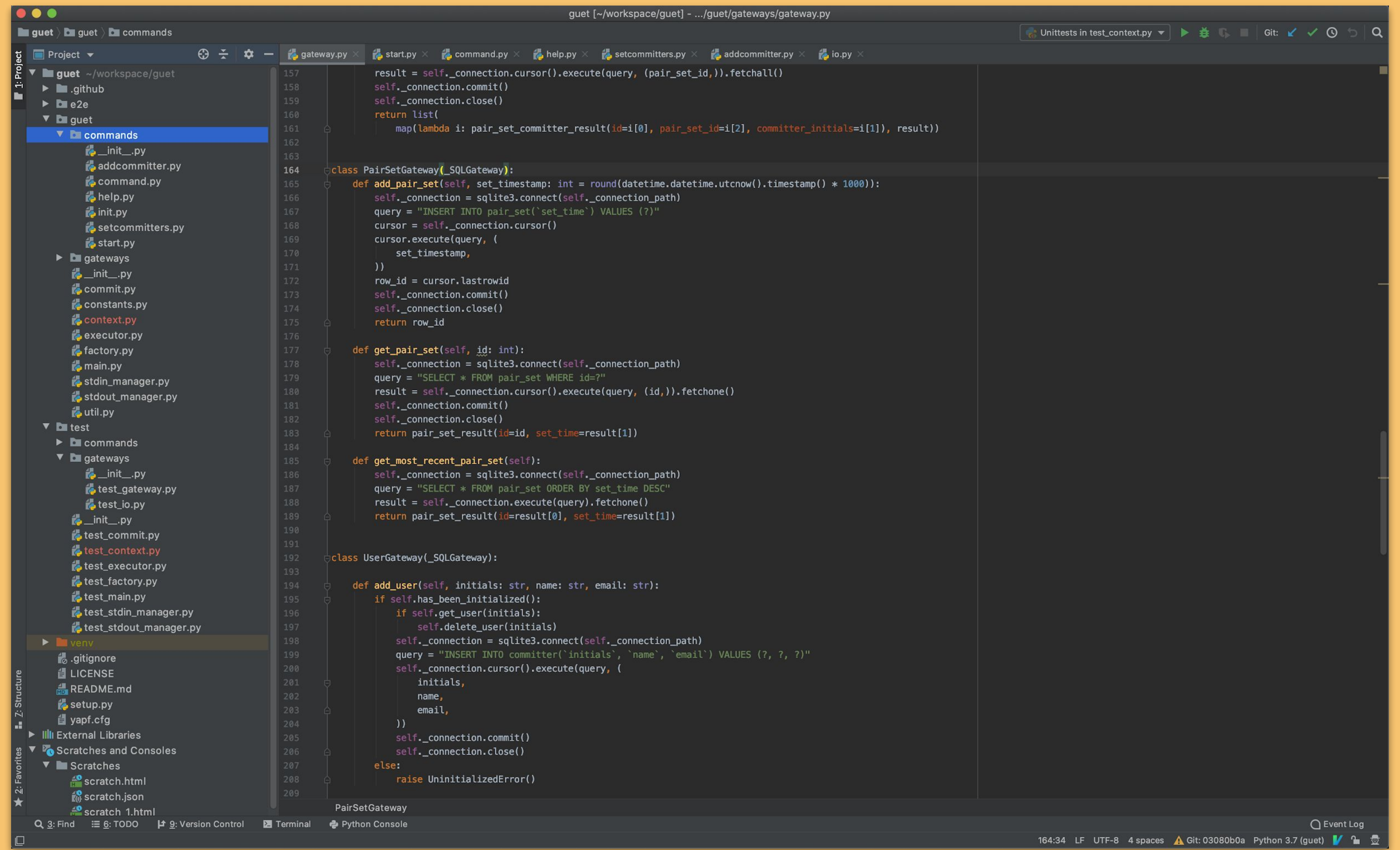

StartCommand
_args: List[str]
_print_gateway: PrintGateway
_git_gateway: GitGateway
_input_gateway: InputGateway
GitGateway
PrintGateway
InputGateway

“

I should be able to know what a file does without having to scroll.

Matt Schultz, roasting the guet codebase

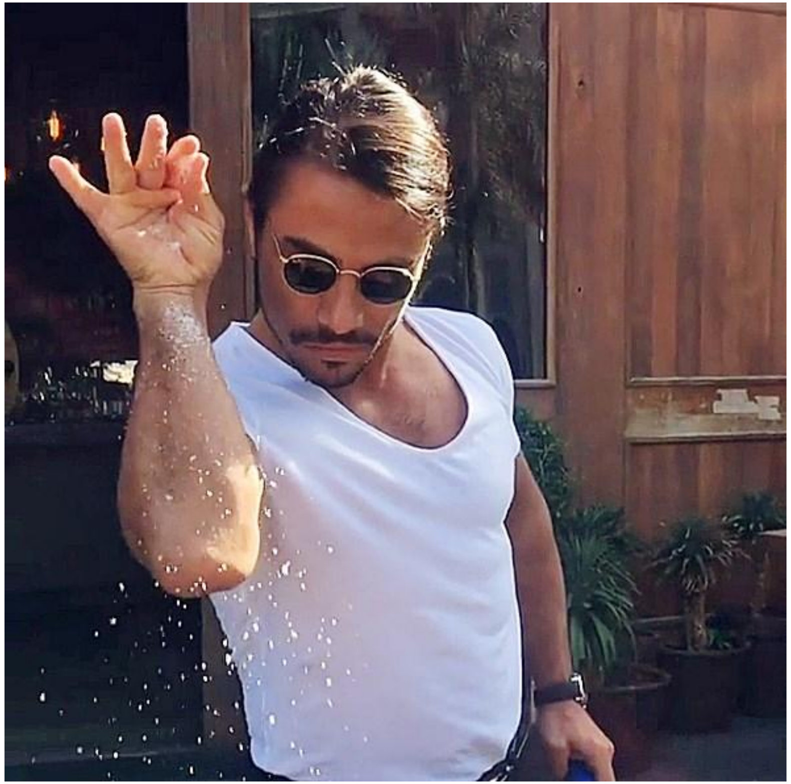
”



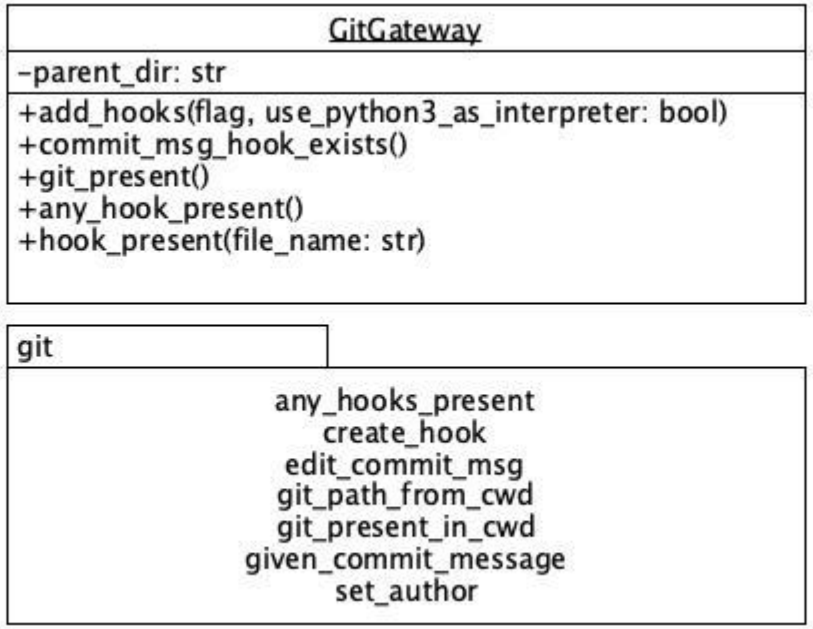
```python
        result = self._connection.cursor().execute(query, (pair_set_id,)).fetchall()
        self._connection.commit()
        self._connection.close()
        return list(
            map(lambda i: pair_set_committer_result(id=i[0], pair_set_id=i[2], committer_initials=i[1]), result))


class PairSetGateway(_SQLGateway):
    def add_pair_set(self, set_timestamp: int = round(datetime.datetime.utcnow().timestamp() * 1000)):
        self._connection = sqlite3.connect(self._connection_path)
        query = "INSERT INTO pair_set(`set_time`) VALUES (?)"
        cursor = self._connection.cursor()
        cursor.execute(query, (
            set_timestamp,
        ))
        row_id = cursor.lastrowid
        self._connection.commit()
        self._connection.close()
        return row_id

    def get_pair_set(self, id: int):
        self._connection = sqlite3.connect(self._connection_path)
        query = "SELECT * FROM pair_set WHERE id=?"
        result = self._connection.cursor().execute(query, (id,)).fetchone()
        self._connection.commit()
        self._connection.close()
        return pair_set_result(id=id, set_time=result[1])

    def get_most_recent_pair_set(self):
        self._connection = sqlite3.connect(self._connection_path)
        query = "SELECT * FROM pair_set ORDER BY set_time DESC"
        result = self._connection.execute(query).fetchone()
        return pair_set_result(id=result[0], set_time=result[1])


class UserGateway(_SQLGateway):

    def add_user(self, initials: str, name: str, email: str):
        if self.has_been_initialized():
            if self.get_user(initials):
                self.delete_user(initials)
            self._connection = sqlite3.connect(self._connection_path)
            query = "INSERT INTO committer(`initials`, `name`, `email`) VALUES (?, ?, ?)"
            self._connection.cursor().execute(query, (
                initials,
                name,
                email,
            ))
            self._connection.commit()
            self._connection.close()
        else:
            raise UninitializedError()
```

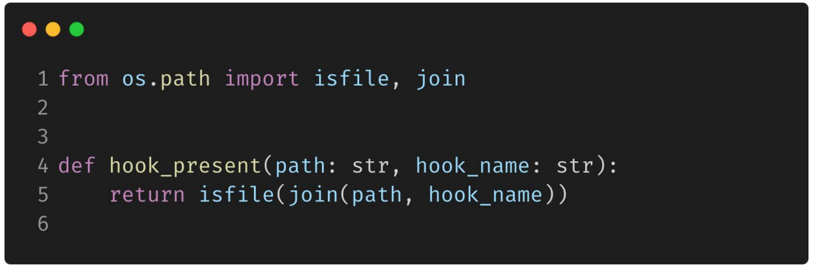
How did I fix guet?

First, make the codebase more pythonic

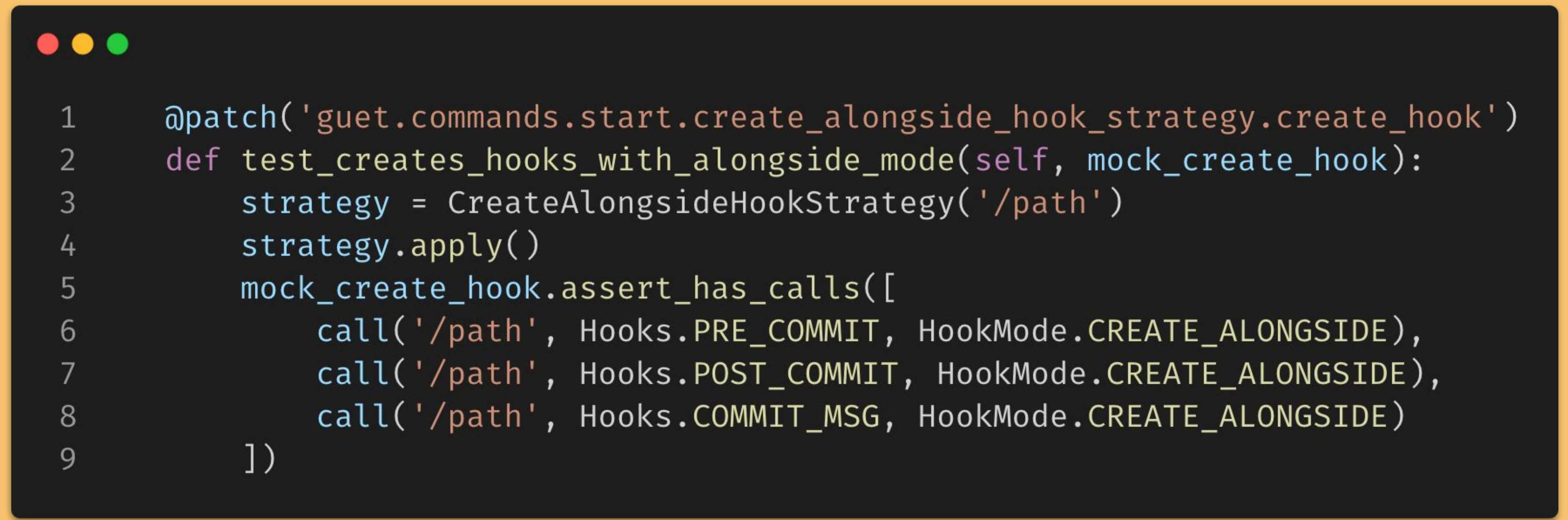


What it feels like when you see some really pythonic code





```python
from os.path import isfile, join


def hook_present(path: str, hook_name: str):
    return isfile(join(path, hook_name))

```

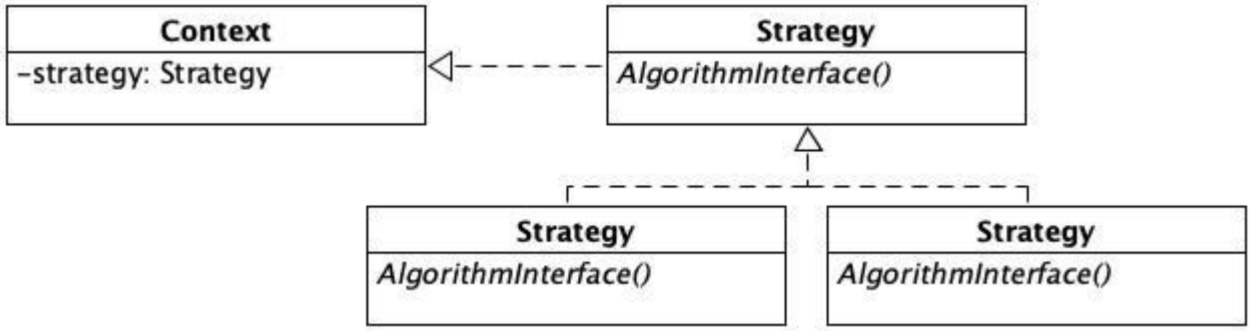


```python
@patch('guet.commands.start.create_alongside_hook_strategy.create_hook')
def test_creates_hooks_with_alongside_mode(self, mock_create_hook):
    strategy = CreateAlongsideHookStrategy('/path')
    strategy.apply()
    mock_create_hook.assert_has_calls([
        call('/path', Hooks.PRE_COMMIT, HookMode.CREATE_ALONGSIDE),
        call('/path', Hooks.POST_COMMIT, HookMode.CREATE_ALONGSIDE),
        call('/path', Hooks.COMMIT_MSG, HookMode.CREATE_ALONGSIDE)
    ])
```

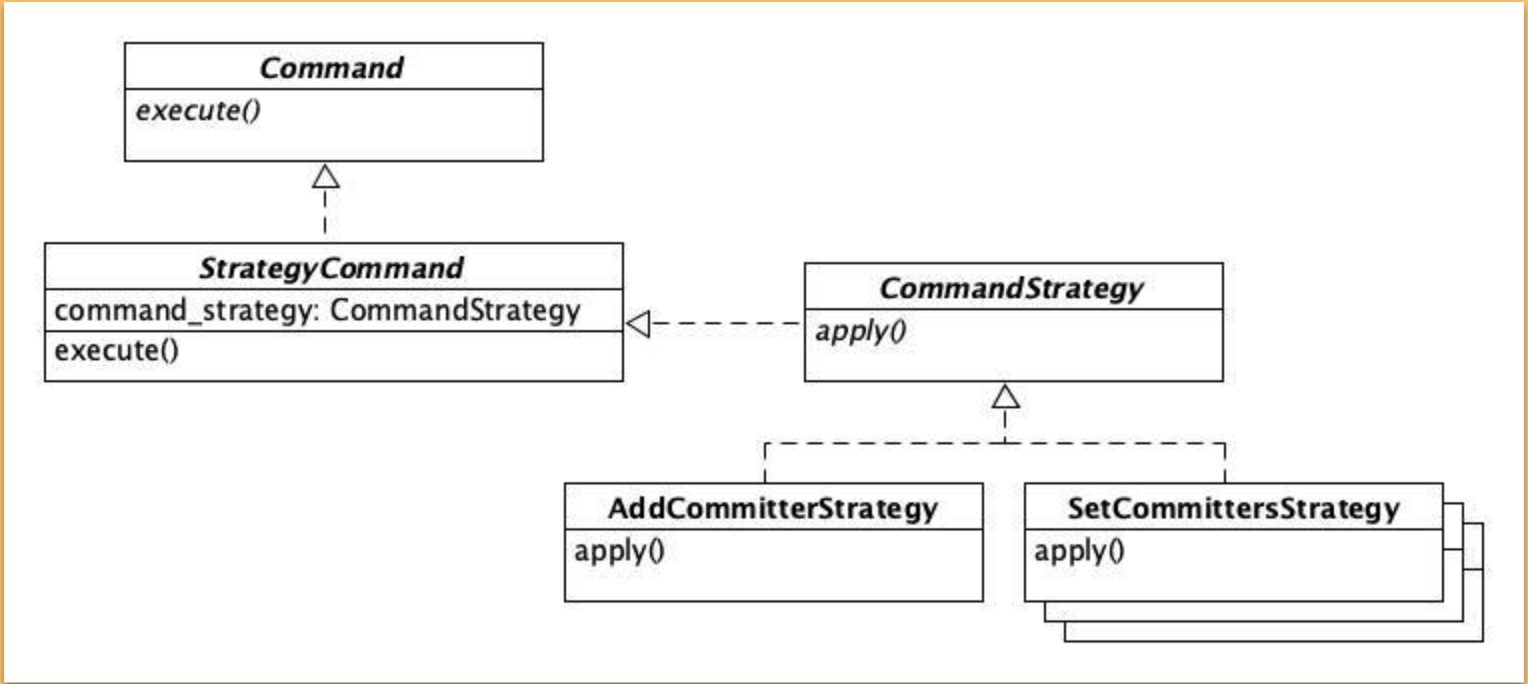
Second, slap some object oriented code onto that puppy

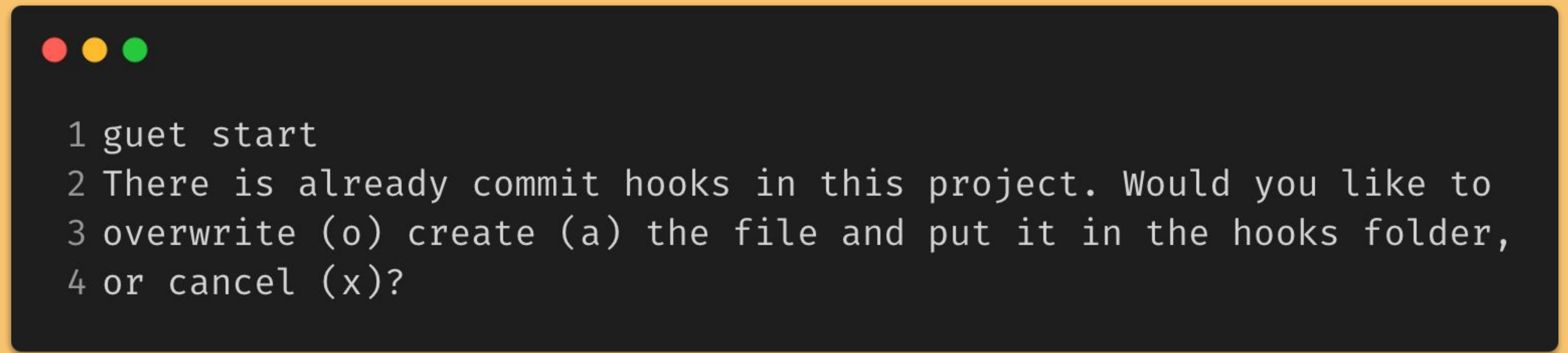
Strategy Pattern

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from client that use it.

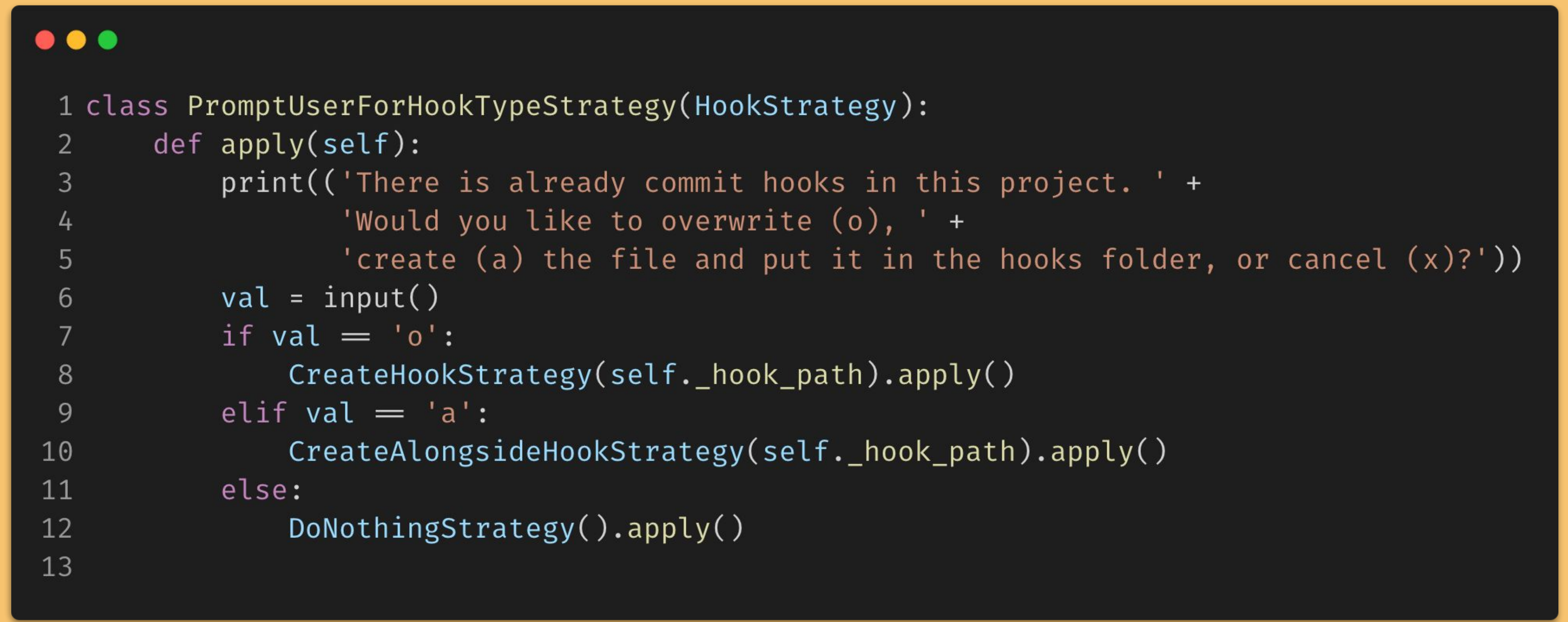


Gang of Four ***Design Patterns: Elements of Reusable Object-Oriented Software***


Command
execute()
StrategyCommand
command_strategy: CommandStrategy
execute()
CommandStrategy
apply()
AddCommitterStrategy
apply()
SetCommittersStrategy
apply()



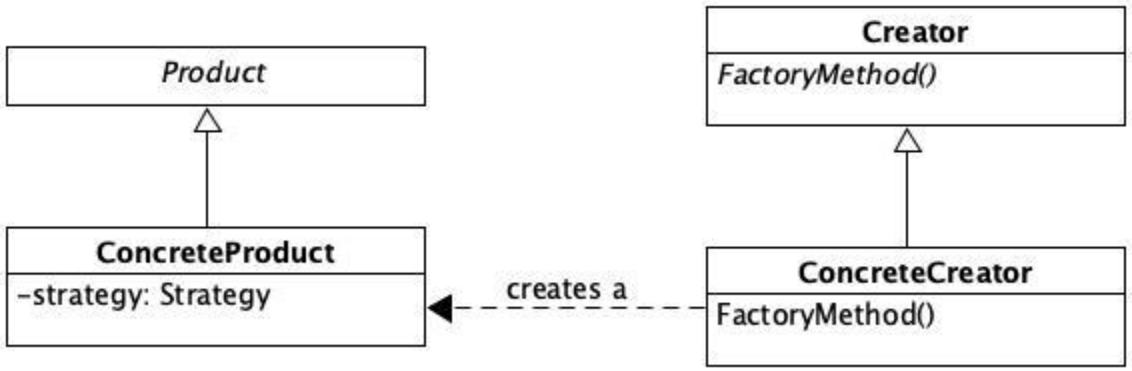
```
guet start
There is already commit hooks in this project. Would you like to
overwrite (o) create (a) the file and put it in the hooks folder,
or cancel (x)?
```



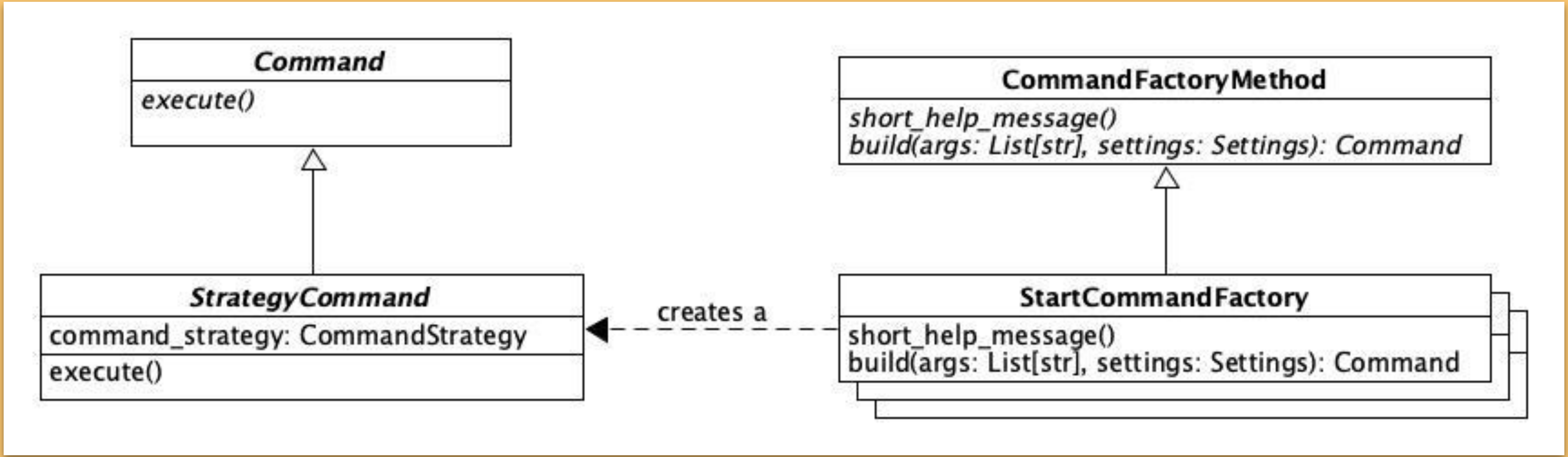
```python
class PromptUserForHookTypeStrategy(HookStrategy):
    def apply(self):
        print(('There is already commit hooks in this project. ' +
               'Would you like to overwrite (o), ' +
               'create (a) the file and put it in the hooks folder, or cancel (x)?'))
        val = input()
        if val == 'o':
            CreateHookStrategy(self._hook_path).apply()
        elif val == 'a':
            CreateAlongsideHookStrategy(self._hook_path).apply()
        else:
            DoNothingStrategy().apply()

```

Factory Method Pattern

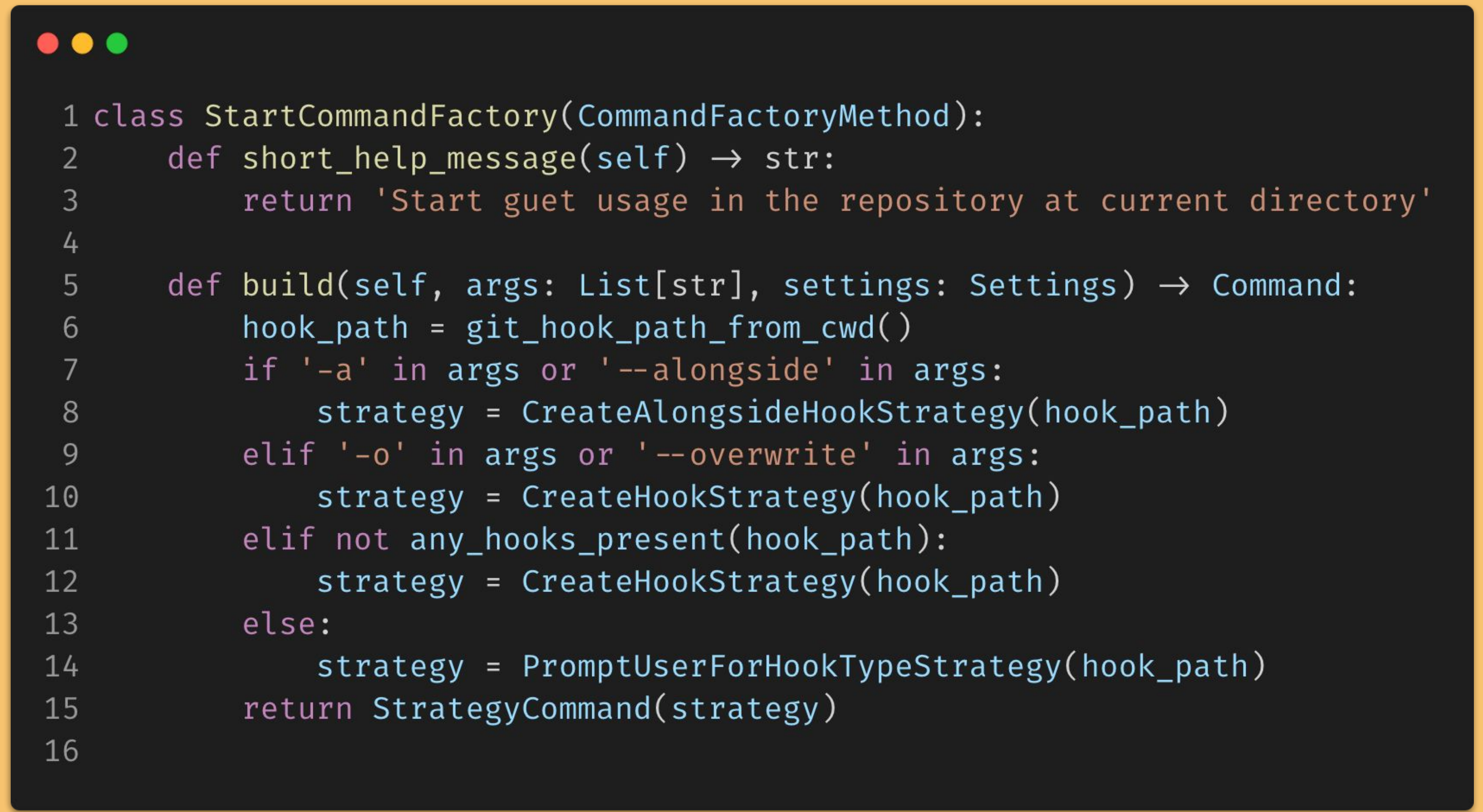
Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses.



Gang of Four ***Design Patterns: Elements of Reusable Object-Oriented Software***



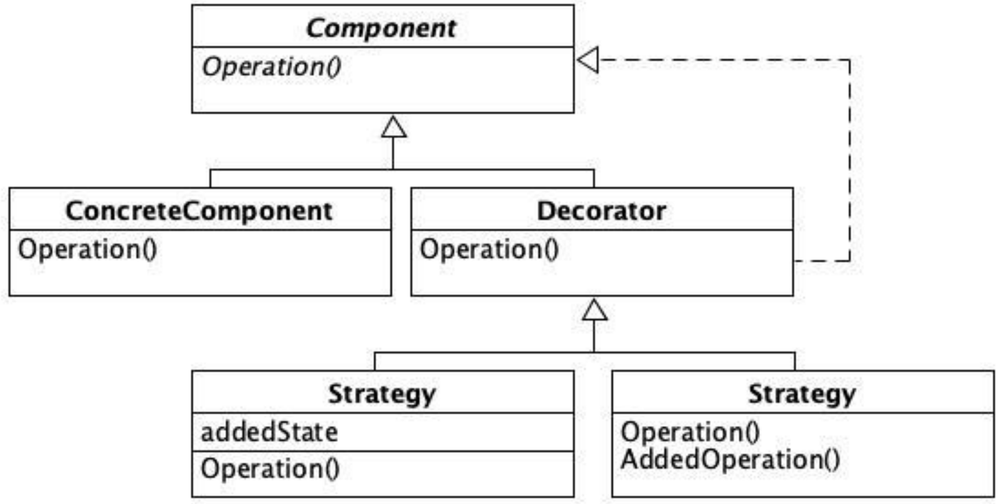
Command
execute()
StrategyCommand
command_strategy: CommandStrategy
execute()
creates a
CommandFactoryMethod
short_help_message()
build(args: List[str], settings: Settings): Command
StartCommandFactory
short_help_message()
build(args: List[str], settings: Settings): Command



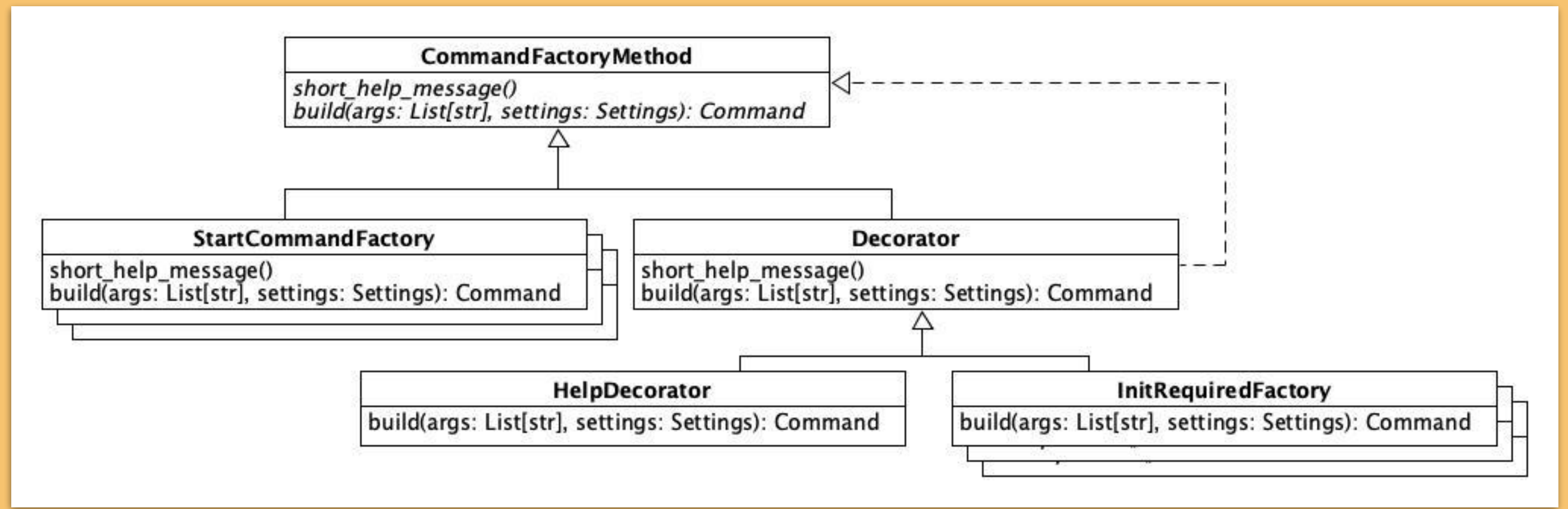
```python
class StartCommandFactory(CommandFactoryMethod):
    def short_help_message(self) -> str:
        return 'Start guet usage in the repository at current directory'

    def build(self, args: List[str], settings: Settings) -> Command:
        hook_path = git_hook_path_from_cwd()
        if '-a' in args or '--alongside' in args:
            strategy = CreateAlongsideHookStrategy(hook_path)
        elif '-o' in args or '--overwrite' in args:
            strategy = CreateHookStrategy(hook_path)
        elif not any_hooks_present(hook_path):
            strategy = CreateHookStrategy(hook_path)
        else:
            strategy = PromptUserForHookTypeStrategy(hook_path)
        return StrategyCommand(strategy)

```

Decorator Pattern

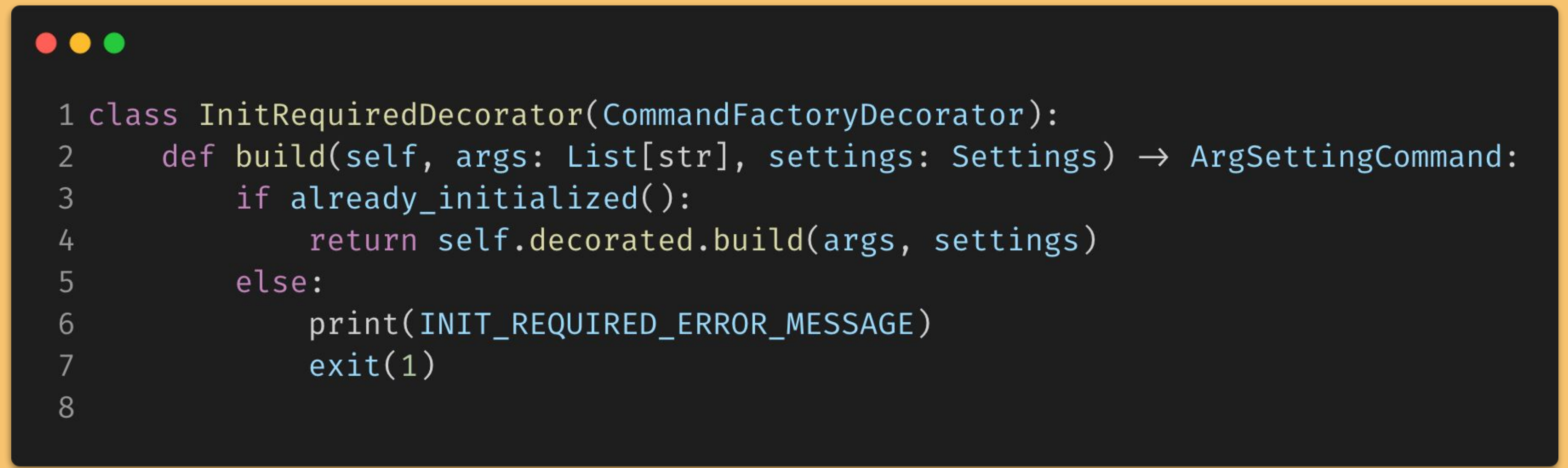
Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.



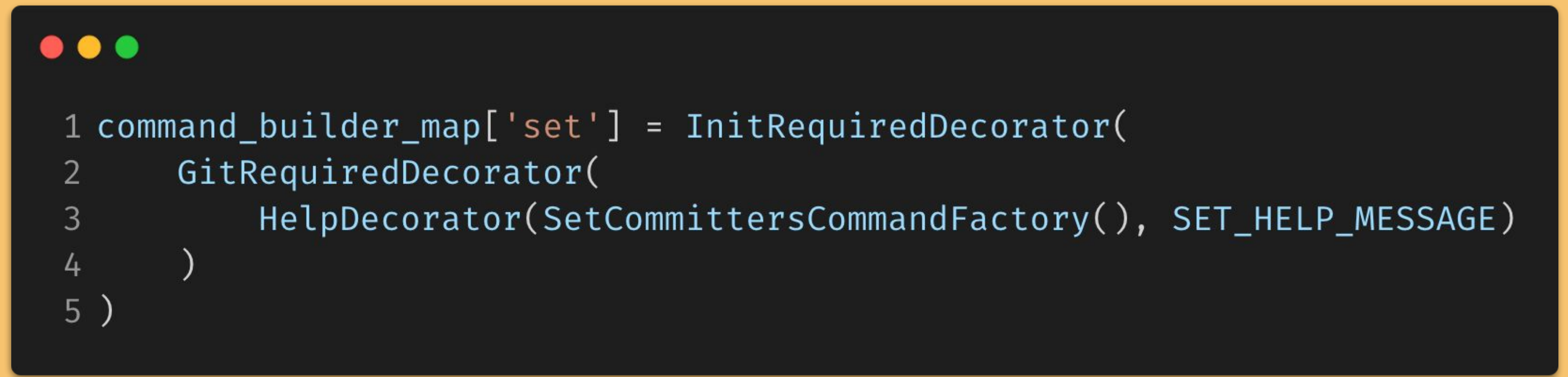
Gang of Four ***Design Patterns: Elements of Reusable Object-Oriented Software***



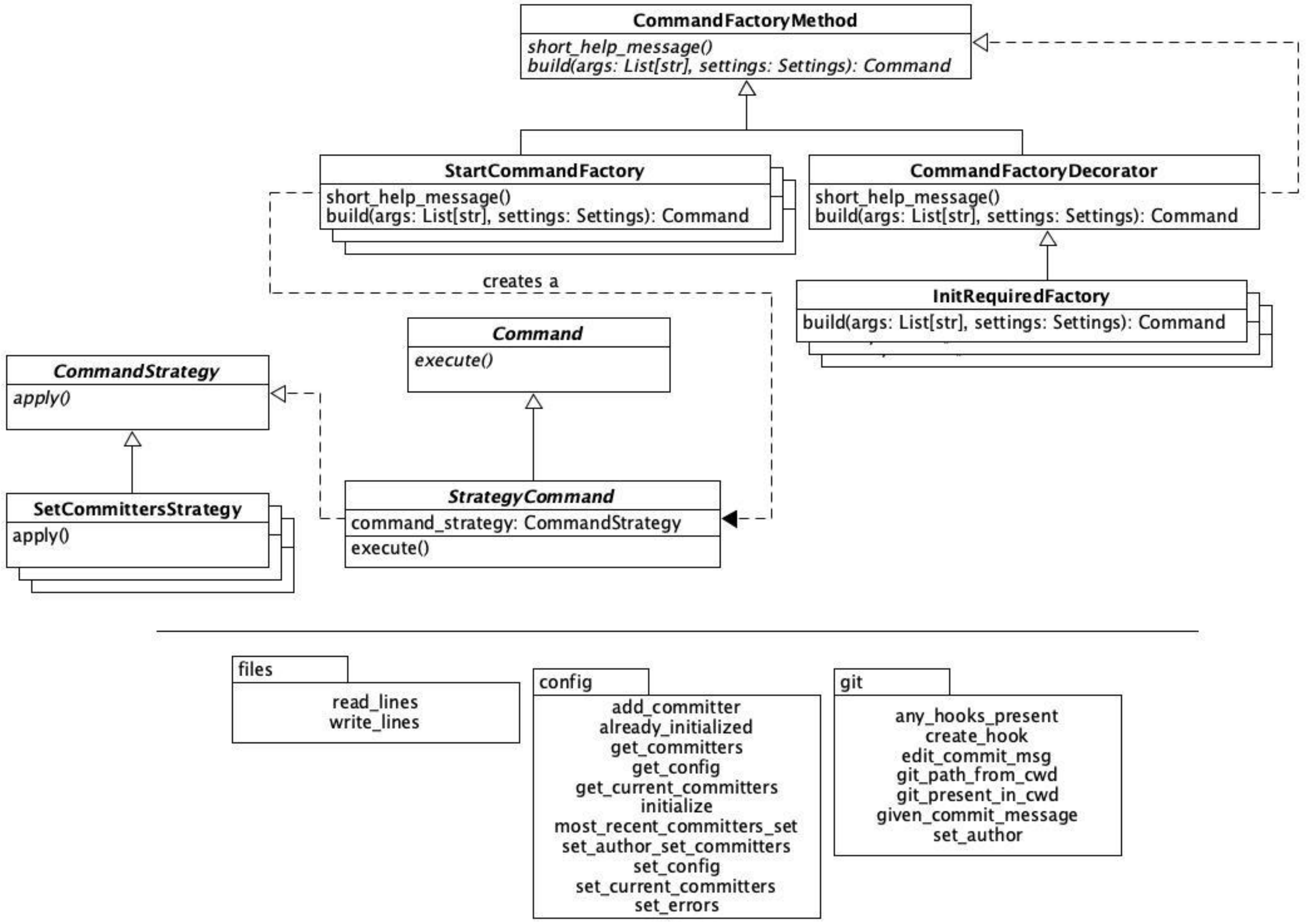
CommandFactoryMethod
short_help_message()
build(args: List[str], settings: Settings): Command
StartCommandFactory
short_help_message()
build(args: List[str], settings: Settings): Command
Decorator
short_help_message()
build(args: List[str], settings: Settings): Command
HelpDecorator
build(args: List[str], settings: Settings): Command
InitRequiredFactory
build(args: List[str], settings: Settings): Command



```python
class InitRequiredDecorator(CommandFactoryDecorator):
    def build(self, args: List[str], settings: Settings) -> ArgSettingCommand:
        if already_initialized():
            return self.decorated.build(args, settings)
        else:
            print(INIT_REQUIRED_ERROR_MESSAGE)
            exit(1)

```

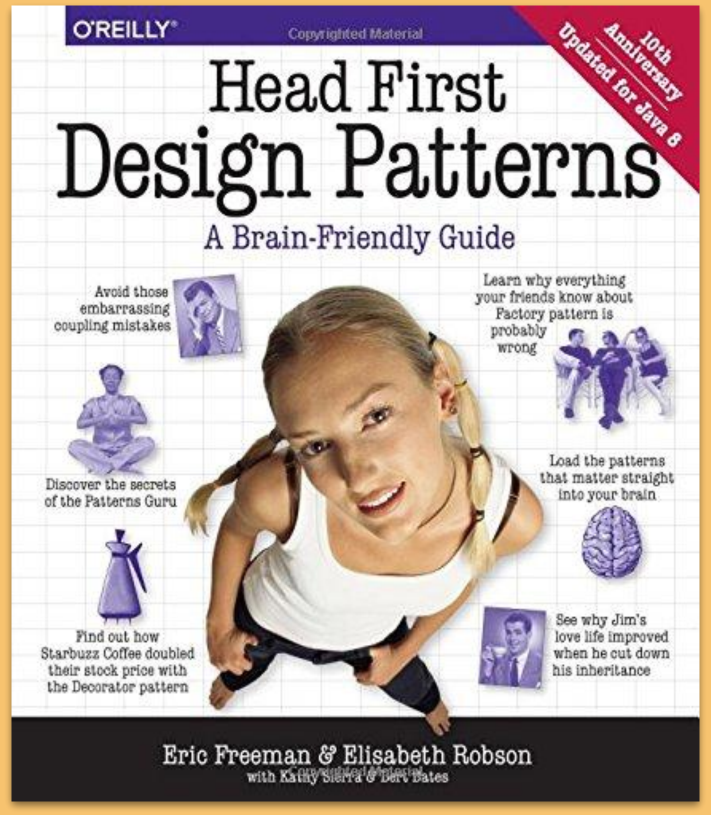


```
command_builder_map['set'] = InitRequiredDecorator(
    GitRequiredDecorator(
        HelpDecorator(SetCommittersCommandFactory(), SET_HELP_MESSAGE)
    )
)
```


CommandFactoryMethod
short_help_message()
build(args: List[str], settings: Settings): Command
StartCommandFactory
short_help_message()
build(args: List[str], settings: Settings): Command
CommandFactoryDecorator
short_help_message()
build(args: List[str], settings: Settings): Command
creates a
InitRequiredFactory
build(args: List[str], settings: Settings): Command
Command
execute()
CommandStrategy
apply()
SetCommittersStrategy
apply()
StrategyCommand
command_strategy: CommandStrategy
execute()
files
read_lines
write_lines
config
add_committer
already_initialized
get_committers
get_config
get_current_committers
initialize
most_recent_committers_set
set_author_set_committers
set_config
set_current_committers
set_errors
git
any_hooks_present
create_hook
edit_commit_msg
git_path_from_cwd
git_present_in_cwd
given_commit_message
set_author

Why should I care?

Shippensburg University
of Pennsylvania



O'REILLY

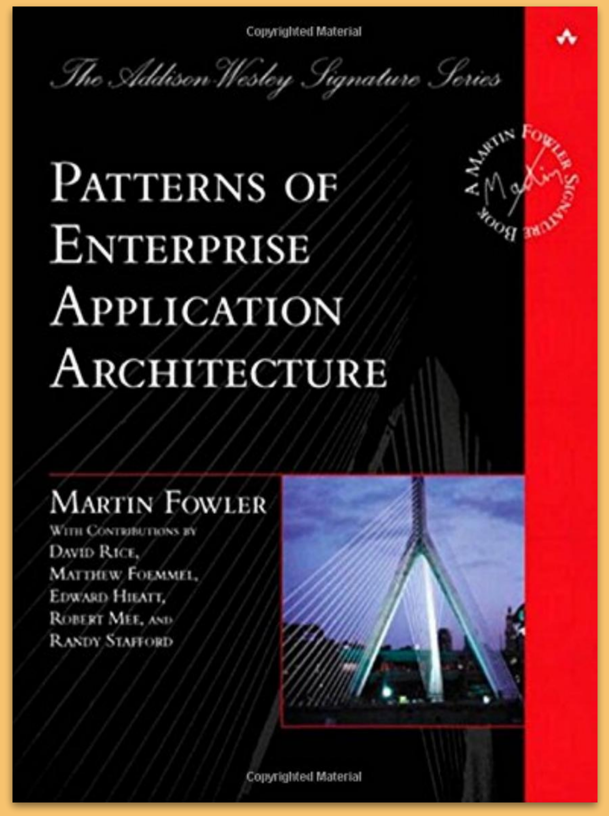
10th Anniversary Updated for Java 8

Head First Design Patterns

A Brain-Friendly Guide

Avoid those embarrassing coupling mistakes
Learn why everything your friends know about Factory pattern is probably wrong
Discover the secrets of the Patterns Guru
Load the patterns that matter straight into your brain
Find out how Starbuzz Coffee doubled their stock price with the Decorator pattern
See why Jim's love life improved when he cut down his inheritance

Eric Freeman & Elisabeth Robson
with Kathy Sierra & Bert Bates



The Addison-Wesley Signature Series

A Martin Fowler Signature Book

Patterns of Enterprise Application Architecture

Martin Fowler
With Contributions by
David Rice,
Matthew Foemmel,
Edward Hieatt,
Robert Mee, and
Randy Stafford

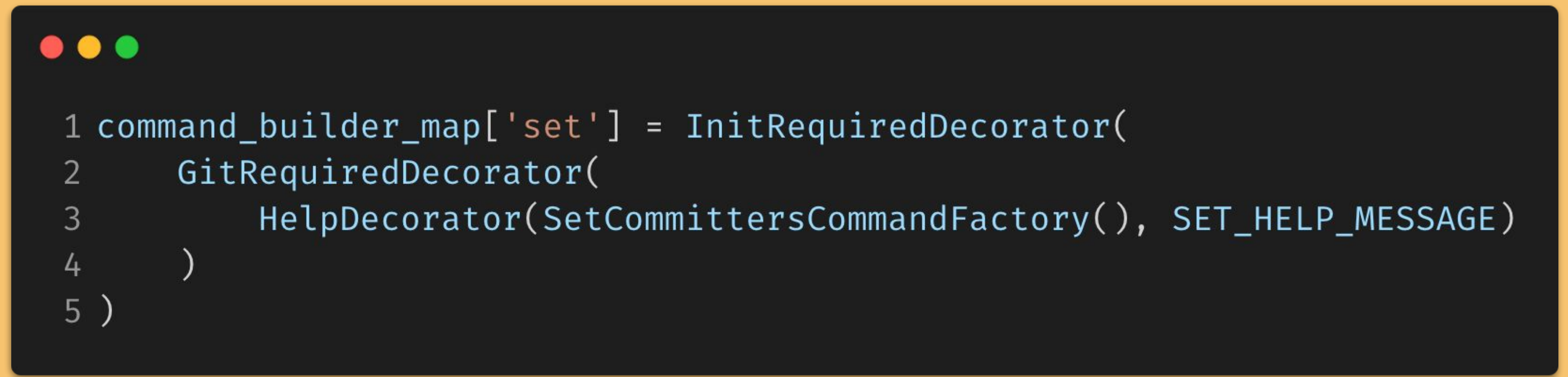
“

Wouldn't it be dreamy if there was a Design Patterns book that was more fun than going to the dentist, and more revealing than an IRS form?

The mad lads that wrote the Head First design Patterns book

”

Design Patterns are a language which software engineers can use to communicate solutions to common problems.



```python
command_builder_map['set'] = InitRequiredDecorator(
    GitRequiredDecorator(
        HelpDecorator(SetCommittersCommandFactory(), SET_HELP_MESSAGE)
    )
)
```

Thank you for attending. Goodbye.