Everyday Design Patterns: **Facade Pattern**

Aly Sivji

@CaiusSivjus

I’m Aly Sivji. @CaiusSivjus on 

ChiPy.org

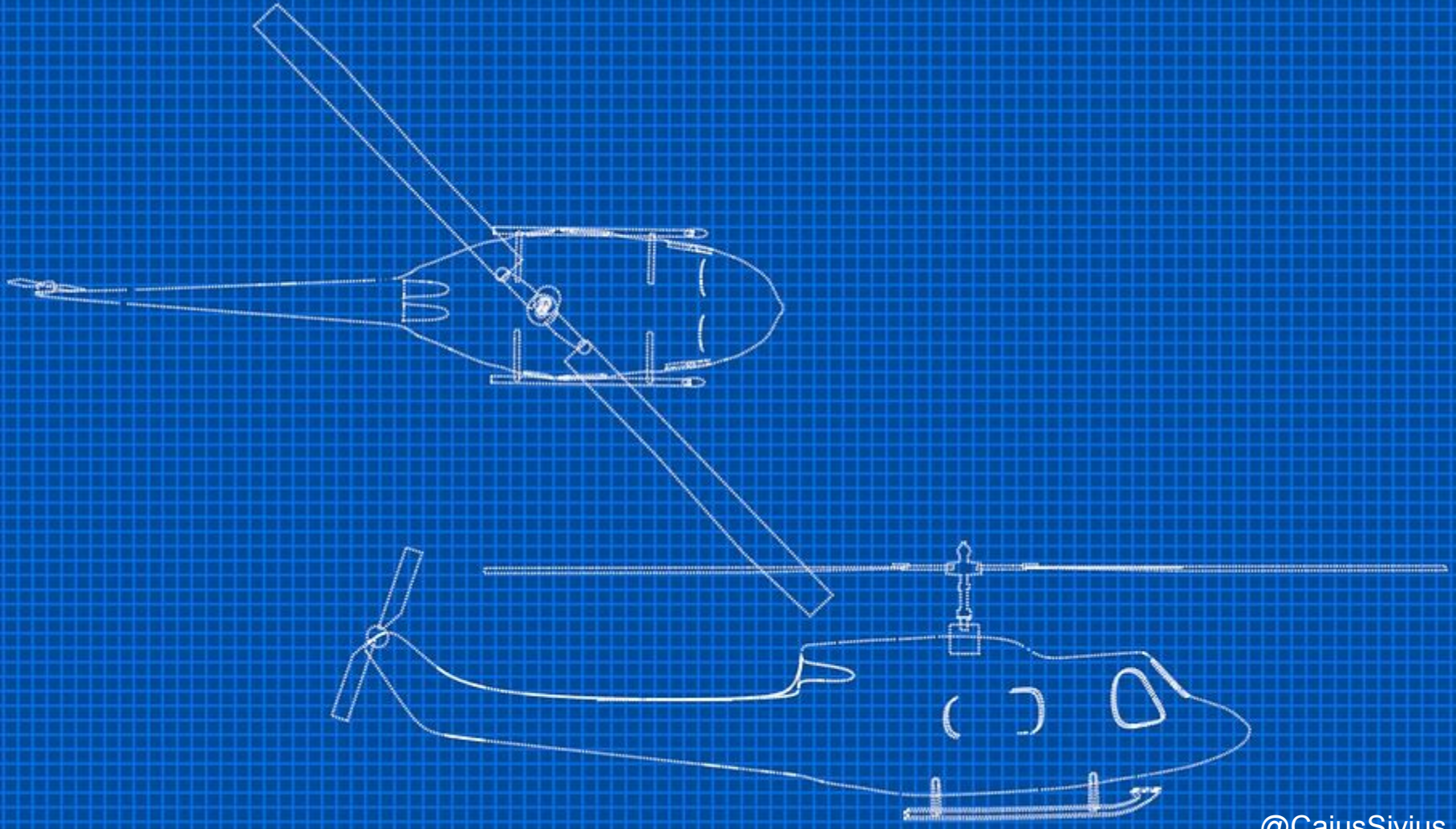
Numerator

(We’re Hiring)

NumeratorEngineering.com

Design Patterns

Design patterns are solutions to commonly occurring software design problems



@CaiusSivjus

Benefits of Design Patterns

- Tried and tested solutions to common problems in software design
- Define a common language for more effective communication
- Can be used in any type of application or domain
- Reusable in multiple projects

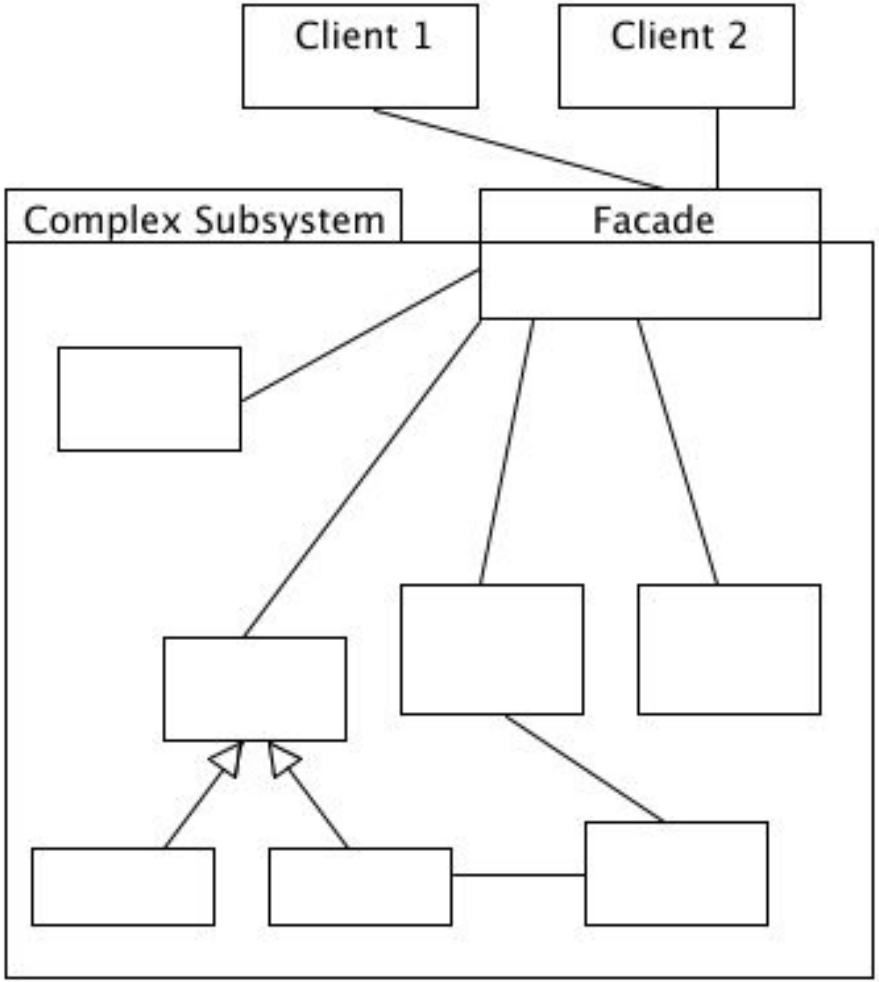
Criticism of Design Patterns

- Workarounds for limitations of a programming language
- Inefficient solutions
- If all you have is a hammer, everything looks like a nail

Types of Object-Oriented Design Patterns

- Creational - object creation
- Structural - build large, complex objects
- Behavioral - play with algorithms and relationship with objects

Facade Pattern



```
/busybeaver next
```

Next ChiPy Event:

> **ChiPy Data SIG presents Property Testing Pandas and Risk Weighting**
> **6:00 PM Wednesday, September 18th** at Braintree Payments

```
/busybeaver events
```

BusyBeaver APP 9:00 AM
(120 kB)

Upcoming Events

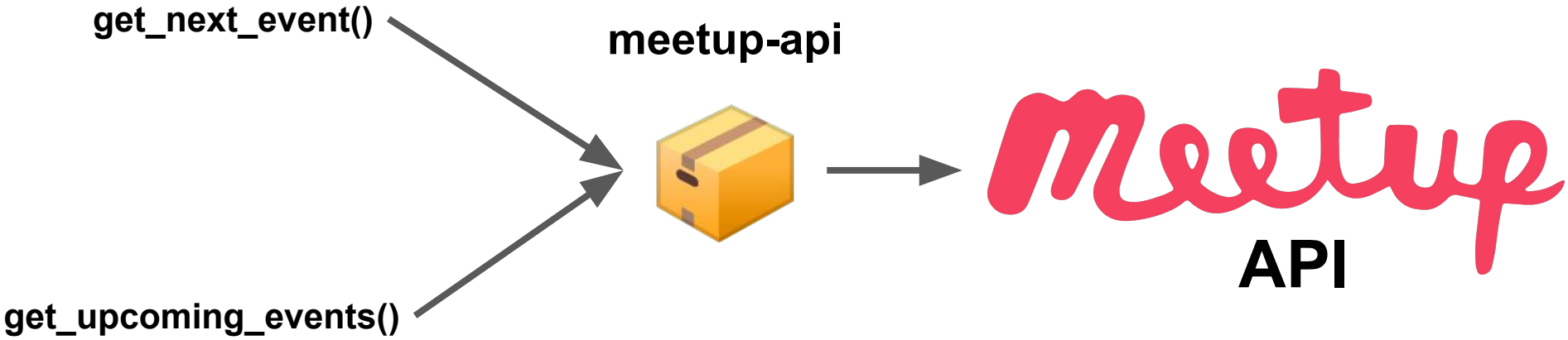
ChiPy Data SIG presents Property Testing Pandas and Risk Weighting
Wednesday, September 18th @ 6:00 PM
Location: Braintree Payments

Python Project Night
Thursday, September 19th @ 6:00 PM
Location: Braintree

fast.ai "Deep Learning for Coders" Group #1: Tooling & Image Classification
Tuesday, September 24th @ 6:00 PM
Location: Metis Data Science

Python Lunch Break SIG
Thursday, September 26th @ 12:00 PM
Location: 8th Light

Algo SIG
Thursday, October 3rd @ 6:00 PM
Location: 1 N State St


get_next_event()
get_upcoming_events()
meetup-api
meetup
API

Using Meetup's API

Meetup's API makes it possible for developers to securely build integrations with Meetup. Before applying to use Meetup's API, please review our API license terms. If you need help using the API, please use our API documentation.

Note: Meetup changed its API access on **August 15, 2019**. API keys have been removed and all version 2 integrations need to transition to version 3.

API Client Details

The following are dynamically generated methods for the `meetup.api.Client` class.

API Client Method Index

- `CreateEvent()`
- `CreateEventComment()`
- `CreateEventCommentFlag()`
- `CreateEventRating()`
- `CreateGroupAbuseReports()`
- `CreateGroupMemberApprovals()`
- `CreateGroupPhoto()`
- `CreateGroupTopics()`
- `CreateGroupVenues()`
- `CreateMemberPhoto()`
- `CreateNotificationsRead()`
- `CreatePhoto()`
- `CreatePhotoAlbum()`
- `CreatePhotoComment()`
- `CreateProfile()`
- `CreateRecommendedGroupsIgnores()`
- `CreateRsvp()`
- `GetCategories()`
- `GetCities()`
- `GetComments()`
- `GetConcierge()`
- `GetDashboard()`
- `GetEvent()`
- `GetEventComment()`
- `GetEventCommentLikes()`
- `GetEventComments()`
- `GetEventRatings()`
- `GetEvents()`
- `GetFindGroups()`
- `GetGroup()`
- `GetGroupBoards()`
- `GetGroupBoardsDiscussions()`
- `GetGroupEventsAttendance()`
- `GetGroupSimilarGroups()`

Meetup API Client Libraries

Client libraries make it easier to access the API from your programming language of choice. Most of these have been developed by the user community. If the feature or API method you want isn't supported, just fork it!

> Note: Meetup makes no guarantees about the maintenance and the support these 3rd party API client libraries offer. Please contact their maintainers if you have questions.

> Note: If you come across a library that no longer seems to be maintained and you think should be removed from this listing, please submit a pull request to do so.

Python

- python-api-client encapsulates all the logic needed to make queries to the Meetup API in Python. (discontinued)
- python-api Python API for Meetup. (maintenance)

```python
class MeetupAdapter:
    """Pull the upcoming events from Meetup and send the message to Slack."""

    def __init__(self, api_key):
        self.meetup_client = MeetupClient(api_key)

    def get_events(self, group_name: str, count: int = 1) -> List[EventDetails]:
        events = self.meetup_client.GetEvents(group_urlname=group_name)
        if not events.results:
            raise NoMeetupEventsFound

        upcoming_events = []
        for event in events.results[:count]:
            if "venue" in event:
                venue_name = event["venue"]["name"]
            else:
                venue_name = "TBD"

            start_epoch = int(event["time"] / 1000)
            upcoming_events.append(
                EventDetails(
                    id=event["id"],
                    name=event["name"],
                    url=event["event_url"],
                    venue=venue_name,
                    start_epoch=start_epoch,
                    end_epoch=start_epoch + int(event["duration"]),
                )
            )

        return upcoming_events
```

Encapsulation

Polymorphism

Object-Oriented Programming Principles

Abstraction

Inheritance

Encapsulation

- Bundle data and behavior into a logical unit aka “object”
  - Objects can communicate with each other by calling methods
- **Reduces complexity and increases readability**

Abstraction

- Want to hide the complexity of our internal implementation
  - Hidden inside high-level abstraction (i.e. object)
- Objects should communicate using **public** methods
  - Python makes internals of class available, but we are all consenting adults
- **Hides complexity and isolates the impact of changes**
  - Can change internal implementation without affecting calling code

```python
class MeetupAdapter:
    """Pull the upcoming events from Meetup and send the message to Slack."""

    def __init__(self, api_key):
        self.meetup_client = MeetupClient(api_key)

    def get_events(self, group_name: str, count: int = 1) -> List[EventDetails]:
        events = self.meetup_client.GetEvents(group_urlname=group_name)
        if not events.results:
            raise NoMeetupEventsFound

        upcoming_events = []
        for event in events.results[:count]:
            if "venue" in event:
                venue_name = event["venue"]["name"]
            else:
                venue_name = "TBD"

            start_epoch = int(event["time"] / 1000)
            upcoming_events.append(
                EventDetails(
                    id=event["id"],
                    name=event["name"],
                    url=event["event_url"],
                    venue=venue_name,
                    start_epoch=start_epoch,
                    end_epoch=start_epoch + int(event["duration"]),
                )
            )

        return upcoming_events
```

```python
class MeetupAdapter:
    """Pull the upcoming events from Meetup and send the message to Slack."""

    def __init__(self, oauth_token: str):
        default_headers = {"Authorization": f"Bearer {oauth_token}"}
        self.client = RequestsClient(headers=default_headers)

    def get_events(self, group_name: str, count: int = 1) -> List[EventDetails]:
        url = BASE_URL + f"/{group_name}/events"
        payload = {"page": count}
        resp: Response = self.client.get(url, params=payload)

        if resp.status_code != 200:
            raise UnexpectedStatusCode

        events = resp.json
        if not events:
            raise NoMeetupEventsFound

        upcoming_events = []
        for event in events:
            if "venue" in event:
                venue_name = event["venue"]["name"]
            else:
                venue_name = "TBD"

            start_epoch = int(event["time"] / 1000)
            upcoming_events.append(
                EventDetails(
                    id=event["id"],
                    name=event["name"],
                    url=event["link"],
                    venue=venue_name,
                    start_epoch=start_epoch,
                    end_epoch=start_epoch + int(event["duration"]),
                )
            )

        return upcoming_events
```

Facade Pattern -- Recap

- Facade pattern provides a simple interface to a complex subsystem
- Reduces coupling between modules
  - Depend on an interface versus an implementation
- Improves testability as unit test facade and replace it with a stub for integration testing

Resources

- The Coded Self: The Difference Between An Adapter And A Wrapper
-

Resources -- Books

- Gang of Four. Design Patterns.
- Head First Design Patterns
- Martin, Robert. (2017). Clean Architecture. 1st ed. Upper Saddle River, NJ: Prentice Hall
- Martin, Robert. (2008). Clean Code: A Handbook of Agile Software Craftsmanship. 1st ed. Upper Saddle River, NJ: Prentice Hall
- A Philosophy of Software Design

Resources -- Videos

- Christopher Okhravi: Adapter Pattern, Facade Pattern, Structural Patterns (comparison)
- Ariel Ortiz: Design Patterns in Python for the Untrained Eye
- Brandon Rhodes: The Clean Architecture in Python
- Luciano Ramalho: Think Like a Pythonista
- Sandi Metz: All the Little Things
-

Thank You

Github: alysivji/talks

Twitter: @CaiusSivjus

Blog: https://alysivji.github.io

Slides: http://bit.ly/facade-pattern

Acknowledgements (Easter Egg)

- ChiPy

- AS, ES, SF, CF, CL, TD, AS