OOP via Python: Session 03 Coda

Stephen Leach, 21 Oct 2021

Objects have "Phases"

- Phases are a pattern of availability for a bunch of methods
  - A method is totally available if it can be called without an exception
  - It is unavailable if it always raises an exception
  - It is partly available when it sometimes throws an exception & sometimes doesn't
- Objects have the same phase if they have the same pattern of availability for those methods

Phases: objects sharing a pattern of availability

Availability	**start**	**try_add**	**can_add_page**	**add_page**	**...**
Empty RangeOfPages	❌	✅	✅	✅	...
Non-Empty RangeOfPages	✅	✅	✅	🤔	...

Phases are a bit like types

- Phases define what you can do with an object - and how
- But types are assigned statically
- Whereas objects can change their phase dynamically

Phases make programming trickier

- So we try to make methods total everywhere
  - Enriching the return values to include "exception values" e.g. None, -1, (), …
  - But the catch is that enriched values kick the can down the road - from the callee to the caller

- OR we can try throwing specific, identified exceptions
  - Which is like letting off a firework to send a can down the road
  - It is easy to forget to put safeguards in place for the fireworks
  - And sometimes it goes off accidentally and causes injury

Alternative Return Values

- How do we safely access the non-None value?
- Check for None?

```
r = x.searchForStuff( stuff )
if r is None:
    print(r.matched_text())
else:
    print('No matches')
```

PEP 634, 635, 636 … Structural Pattern Matching (42 years on)

```
match command.split():
   case ["quit"]: ... # Code omitted for brevity
   case ["go", direction]: ...
   case ["drop", *objects]: ...
   ... # Other cases
   case _:
       print(f"Sorry, I couldn't understand {command!r}")
```

Rules for EAFP

- The EAFP style encourages you to catch guard-exceptions rather than defend against them
- **But** you must be confident there are **no side-effects** between **entering the method and raising the guard exception** (tricky because rarely documented)
- **But** you must be confident that there are **no side-effects** between **dynamically entering the try-block and dynamically throwing the exception including in every method dynamically called between the two**
- **And** you must be confident that **everyone who modifies the code is aware of these constraints and is able to maintain them**.
- **Plus** you must be confident that the exception you catch was originated from the expected throw and not simply a subsidiary method.

How to write a phase guard - *partial*

```
def start(self) -> int:
    # Guard
    if self.is_empty():
        raise IndexError("empty RangeOfPages has no start")
    # Body
    return self._start
```

} *no side-effects!*

How to write a phase guard - *partial*

```
def add_page(self, page: int):
    if self.is_empty():
        self._start = page
        self._stop = page + 1
    elif self._stop == page:
        self._stop += 1
    else:
        # At top level as the default case.
        raise Exception(f"Trying to add non-consecutive page: {page}")
```

} *no side-effects!*

How to write a phase guard - *total*

```
def start(self) -> int:
    # Guard
    if self.is_empty():
        return None      # Our exceptional value
    # Body
    return self._start
```

} *no side-effects!*