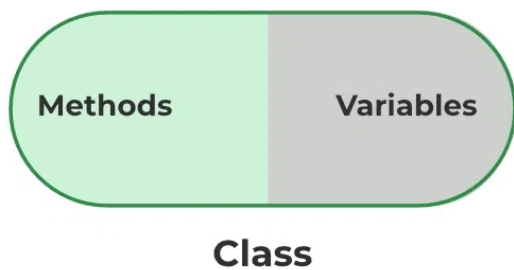


Encapsulation

Class Creation Considerations:

Encapsulation in Python



1. **Encapsulation:** the bundling of Variables and Methods in a Class to ensure that the behavior of an object can only be affected through its Public API. It lets us control how much a change to one object will impact other parts of the system by ensuring that there are no unexpected dependencies between unrelated components.
 - Attributes: Variables are assigned attributes within a `__init__` function
 - Public API: Methods are functions that manipulate the Attributes, such as `get/set`
2. **Information Hiding:** Restricting access to implementation details and violating state invariance
 - Protected Variable:
 - Members of the class that cannot be accessed outside the class but can be accessed from within the class and its subclasses
 - In Python, Using underscore (`_name`) to denote Protected
 - Private Variable:
 - Members can neither be accessed outside the class nor by any base class
 - In Python, double underscore (`__name`) to denote Private.
 - Note that Python is not a true Protected or Private enforced by the language like C++
 - Can still access protected and private variables via dot notation (`hello._name`)

```
from dataclasses import dataclass
from enum import Enum
```

```
from typing import Any
```

```
class PaymentStatus(Enum):
```

```
    CANCELLED = "cancelled"
```

```
    PENDING = "pending"
```

```
    PAID = "paid"
```

```
class PaymentStatusError(Exception):
```

```
    pass
```

```
@dataclass
```

```
class OrderNoEncapsulationNoInformationHiding:
```

```
    """Anyone can get the payment status directly via the instance variable.
```

```
    There are no boundaries whatsoever."""
```

```
    payment_status: PaymentStatus = PaymentStatus.PENDING
```

```
@dataclass
```

```
class OrderEncapsulationNoInformationHiding:
```

```
    """There's an interface now that you should use that provides encapsulation.
```

```
    Users of this class still need to know that the status is represented by an enum type."""
```

```
    _payment_status: PaymentStatus = PaymentStatus.PENDING
```

```
    def get_payment_status(self) -> PaymentStatus:
```

```
        return self._payment_status
```

```
    def set_payment_status(self, status: PaymentStatus) -> None:
```

```
        if self._payment_status == PaymentStatus.PAID:
```

```
            raise PaymentStatusError(
```

```
                "You can't change the status of an already paid order."
```

```
            )
```

```
        self._payment_status = status
```

```
@dataclass
```

```
class OrderEncapsulationAndInformationHiding:
```

```
    """The status variable is set to 'private'. The only thing you're supposed to use is the is_paid
    method, you need no knowledge of how status is represented (that information is 'hidden')."""
```

```
    _payment_status: PaymentStatus = PaymentStatus.PENDING
```

```
    def is_paid(self) -> bool:
```

```
        return self._payment_status == PaymentStatus.PAID
```

```
    def is_cancelled(self) -> bool:
```

```
        return self._payment_status == PaymentStatus.CANCELLED
```

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

```
        if self._payment_status == PaymentStatus.PAID:
```

```
            raise PaymentStatusError("You can't cancel an already paid order.")
```

```
        self._payment_status = PaymentStatus.CANCELLED
```

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

```
        if self._payment_status == PaymentStatus.PAID:
```

```
            raise PaymentStatusError("Order is already paid.")
```

```
        self._payment_status = PaymentStatus.PAID
```

```
@dataclass
```

```
class OrderInformationHidingWithoutEncapsulation:
```

```
    """The status variable is public again (so there's no boundary),
    but we don't know what the type is - that information is hidden. I know, it's a bit
    of a contrived example - you wouldn't ever do this. But at least it shows that
    it's possible."""
```

```
    payment_status: Any = None
```

```
    def is_paid(self) -> bool:
```

```
        return self.payment_status == PaymentStatus.PAID
```

```
    def is_cancelled(self) -> bool:
```

```
        return self.payment_status == PaymentStatus.CANCELLED
```

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

```
        if self.payment_status == PaymentStatus.PAID:
```

```
        raise PaymentStatusError("You can't cancel an already paid order.")
    self.payment_status = PaymentStatus.CANCELLED
```

```
def pay(self) -> None:
    if self.payment_status == PaymentStatus.PAID:
        raise PaymentStatusError("Order is already paid.")
    self.payment_status = PaymentStatus.PAID
```

```
def main() -> None:
    test = OrderInformationHidingWithoutEncapsulation()
    test.pay()
    print("Is paid: ", test.is_paid())
```

```
if __name__ == "__main__":
    main()
```

Revision #8

Created 26 February 2024 01:47:40 by victor

Updated 1 April 2024 03:40:12 by victor