Python Liskov Substitution Principle

Summary: in this tutorial, you'll learn about the Liskov Substitution Principle and how to implement it in Python.

Introduction to the Liskov substitution principle

The Liskov substitution principle (LSV) is one of the five principles in the SOLID principles. The L in SOLID stands for the Liskov substitution principle.

- **S** – Single Responsibility Principle (https://www.pythontutorial.net/python-oop/python-single-responsibility-principle/)
- **O** – Open-closed Principle (https://www.pythontutorial.net/python-oop/python-open-closed-principle/)
- **L** – Liskov Substitution Principle
- **I** – Interface Segregation Principle (https://www.pythontutorial.net/python-oop/python-interface-segregation-principle/)
- **D** – Dependency Inversion Principle (https://www.pythontutorial.net/python-oop/python-dependency-inversion-principle/)

The Liskov substitution principle states that a child class must be substitutable for its parent class. Liskov substitution principle aims to ensure that the child class can assume the place of its parent class

without causing any errors.

Consider the following example:

```
from abc import ABC, abstractmethod


class Notification(ABC):
    @abstractmethod
    def notify(self, message, email):
        pass


class Email(Notification):
    def notify(self, message, email):
        print(f'Send {message} to {email}')


class SMS(Notification):
    def notify(self, message, phone):
        print(f'Send {message} to {phone}')


if __name__ == '__main__':
    notification = SMS()
    notification.notify('Hello', 'john@test.com')
```

In this example, we have three classes: Notification, Email, and SMS. The Email and SMS classes inherit from the Notification class.

The `Notification` abstract class has the `notify()` method that sends a message to an email.

The `notify()` method of the Email class sends a message to an email, which is fine.

However, the SMS class uses a phone number, not an email, for sending a message. Therefore, we need to change the signature of the `notify()` method of the SMS class to accept a phone number instead of an email.

The following `NotificationManager` class uses the `Notification` object to send a message to a `Contact`:

```
class Contact:
    def __init__(self, name, email, phone):
        self.name = name
        self.email = email
        self.phone = phone


class NotificationManager:
    def __init__(self, notification, contact):
        self.contact = contact
        self.notification = notification

    def send(self, message):
        if isinstance(self.notification, Email):
            self.notification.notify(message, contact.email)
        elif isinstance(self.notification, SMS):
            self.notification.notify(message, contact.phone)
        else:
            raise Exception('The notification is not supported')


if __name__ == '__main__':
    contact = Contact('John Doe', 'john@test.com', '(408)-888-9999')
    notification_manager = NotificationManager(SMS(), contact)
    notification_manager.send('Hello John')
```

The `send()` method of the `NoticationManager` class accepts a notification object. It checks whether the notification is an instance of the `Email` or `SMS` and passes the email and phone of contact to the `notify()` method respectively.

Conform with the Liskov substitution principle

First, redefine the `notify()` method of the `Notification` class so that it doesn't include the `email` parameter:

```
class Notification(ABC):
    @abstractmethod
    def notify(self, message):
        pass
```

Second, add the `email` parameter to the `__init__` method of the `Email` class:

```
class Email(Notification):
    def __init__(self, email):
        self.email = email

    def notify(self, message):
        print(f'Send "{message}" to {self.email}')
```

Third, add the `phone` parameter to the `__init__` method of the `SMS` class:

```
class SMS(Notification):
    def __init__(self, phone):
        self.phone = phone

    def notify(self, message):
        print(f'Send "{message}" to {self.phone}')
```

Fourth, change the `NotificationManager` class:

```
class NotificationManager:
    def __init__(self, notification):
        self.notification = notification

    def send(self, message):
        self.notification.notify(message)
```

Put it all together:

```
from abc import ABC, abstractmethod


class Notification(ABC):
    @abstractmethod
    def notify(self, message):
        pass


class Email(Notification):
    def __init__(self, email):
        self.email = email

    def notify(self, message):
        print(f'Send "{message}" to {self.email}')


class SMS(Notification):
    def __init__(self, phone):
        self.phone = phone

    def notify(self, message):
        print(f'Send "{message}" to {self.phone}')
```

```
class Contact:
    def __init__(self, name, email, phone):
        self.name = name
        self.email = email
        self.phone = phone


class NotificationManager:
    def __init__(self, notification):
        self.notification = notification

    def send(self, message):
        self.notification.notify(message)


if __name__ == '__main__':
    contact = Contact('John Doe', 'john@test.com', '(408)-888-9999')

    sms_notification = SMS(contact.phone)
    email_notification = Email(contact.email)

    notification_manager = NotificationManager(sms_notification)
    notification_manager.send('Hello John')

    notification_manager.notification = email_notification
    notification_manager.send('Hi John')
```

Summary

- The Liskov substitution principle states that a child class must be substitutable for its parent class.