In this example:

- `BaseNotifier` defines the interface for sending notifications.
- `EmailNotifier` and `SMSNotifier` are concrete implementations of `BaseNotifier`.
- `Decorator` is an abstract decorator class that wraps a `BaseNotifier` object.
- `PriorityDecorator` and `LoggingDecorator` are concrete decorator classes that add priority and logging functionalities, respectively, to notifications.

```
class BaseNotifier:
    def send_notification(self, message):
        raise NotImplementedError("send_notification method must be implemented")


class EmailNotifier(BaseNotifier):
    def send_notification(self, message):
        print("Sending email notification:", message)


class SMSNotifier(BaseNotifier):
    def send_notification(self, message):
        print("Sending SMS notification:", message)


class Decorator(BaseNotifier):
    def __init__(self, notifier):
        self._notifier = notifier

    def send_notification(self, message):
        self._notifier.send_notification(message)


class PriorityDecorator(Decorator):
    def send_notification(self, message):
        print("Adding priority to the notification.")
        super().send_notification(message)


class LoggingDecorator(Decorator):
    def send_notification(self, message):
        print("Logging the notification.")
        super().send_notification(message)


# Example usage
if __name__ == "__main__":
    # Create a base notifier
    email_notifier = EmailNotifier()

    # Create decorators and stack them
    priority_email_notifier = PriorityDecorator(email_notifier)
    logging_priority_email_notifier = LoggingDecorator(priority_email_notifier)

    # Send notification
    logging_priority_email_notifier.send_notification("This is an important message!")
```

```
Logging the notification.
Adding priority to the notification.
Sending email notification: This is an important message!
```

Start coding or generate with AI.