Object-Oriented Patterns

SECT.11.2026

- When do we use object-oriented design?
- What are the key principles of a good object-oriented design?
- What common OOP patterns are useful to us as Python developers?

Key Ideas

Encapsulation

Inheritance

Composition

Encapsulation

OOP provides a mechanism to keep all of the behaviors (implementation) tucked away neatly within types.

When we use objects, we only interact with a limited public interface.

Loading slide...

Loading slide...

Loading slide...

Loading slide...

SOLID Principles

- Single Responsibility
- Open-Closed
- Liskov Substitution
- Interface Segregation
- Dependency Inversion

Single Responsibility Principle

A class should have a clear & distinct responsibility.

"Should only be one reason for the class to change."

Open-Closed Principle

Open for extension, closed for modification.

Functionality should be added by extending existing behavior, not modifying it.

Liskov Substitution Principle

Objects of a superclass should be replaceable with objects of its subclasses. Works when `subclass` "is a" `superclass`.

Often leads to code that violates Open/Closed, e.g. an `if` within `format_entity_for_display`

Interface Segregation

A client should not be forced to implement an interface it doesn’t use.

Favor composition of small interfaces over large ones where many properties/methods are not used.

Dependency Inversion

High-level modules should not depend on low-level modules.

Depend on interface, not implementation.

Loading slide...

Loading slide...

Loading slide...

Loading slide...

Loading slide...

Loading slide...

Loading slide...

Loading slide...

Loading slide...

Loading slide...

Loading slide...

Loading slide...

Loading slide...

Loading slide...

Loading slide...

Loading slide...

Observer In Action

Django signals

```
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.core.mail import send_mail
from .models import Order

# registers an observer of the `Order.post_save` signal
@receiver(post_save, sender=Order)
def send_order_confirmation(sender, instance, created, **kwargs):
    if created:
        send_mail(
            subject=f"Order #{instance.id} Confirmed",
            message=f"Thanks for your order!",
            from_email="noreply@example.com",
            recipient_list=[instance.user.email],
        )
```