🧠 OOP in AI Systems

Generative AI Bootcamp – Week 1, Day 2, Session 1

November 18, 2025

Learning Objectives

- Revisit object-oriented programming fundamentals
- Understand composition, inheritance, and mixins
- Apply OOP principles to AI system design
- Recognize trade-offs in extensibility and complexity

Why OOP Matters in AI Engineering

- Organizes complex AI systems (e.g., model pipelines, agents)
- Encourages **modularity** and **code reuse**
- Simplifies testing and dependency injection
- Enables plug-and-play architectures for model backends

OOP Principles Refresher

1. **Encapsulation** – group data and behavior
2. **Inheritance** – extend existing implementations
3. **Polymorphism** – use same interface for multiple types
4. **Abstraction** – define contracts via base classes

Example: Model Abstraction

```
from abc import ABC, abstractmethod

class Model(ABC):
    @abstractmethod
    def predict(self, text: str) -> str:
        ...

class OpenAIModel(Model):
    def predict(self, text: str) -> str:
        ...

class GeminiModel(Model):
    def predict(self, text: str) -> str:
        ...
```

Composition Over Inheritance

Instead of subclassing everything:

```
class Pipeline:
    def __init__(self, model, preprocessor):
        self.model = model
        self.preprocessor = preprocessor

    def run(self, text):
        clean = self.preprocessor.clean(text)
        return self.model.predict(clean)
```

- Flexible replacement of components
- Better separation of concerns

Mixins

Lightweight reusable behaviors added via multiple inheritance.

```
class LogMixin:
    def log(self, msg): print(f"[LOG] {msg}")

class ModelWithLogging(OpenAIModel, LogMixin):
    def predict(self, text: str) -> str:
        self.log("Predict called")
        return super().predict(text)
```

Use sparingly — document behavior and avoid deep hierarchies.

Design Patterns in Practice

- **Factory:** instantiate model types dynamically
- **Strategy:** interchangeable algorithms (e.g., tokenizers)
- **Adapter:** unify different model APIs
- **Observer:** track model lifecycle events

Takeaways

- OOP enables clean architecture for scalable AI systems
- Favor **composition**, **interfaces**, and **strategies**
- Avoid overengineering with unnecessary inheritance layers