11 - Systems

2025-10-03 16:09 pm

- Como manter padrão clean a um nível mais alto, mais abstrato. - Separar construção do uso
  - Separação do startup e da operação de um software.
    - Princípio da separação de responsabilidades
  - Startup não deve depender da implementação
    - Uso de "main" para essa separação
    - Uso de Abstract Factory

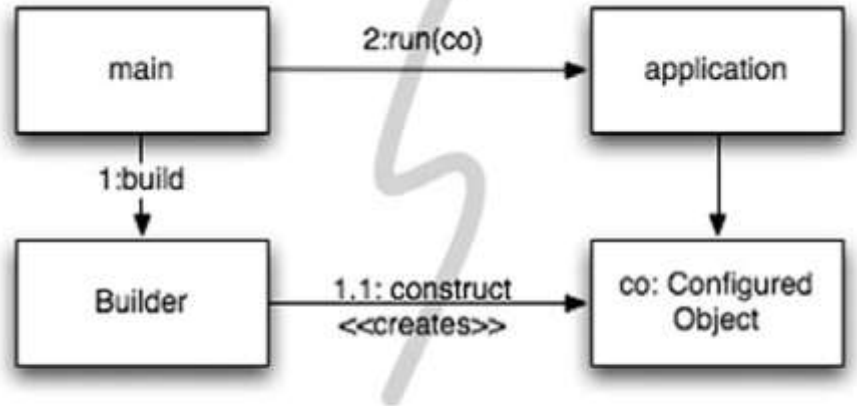

Figure 11-1

Separating construction in `main()`

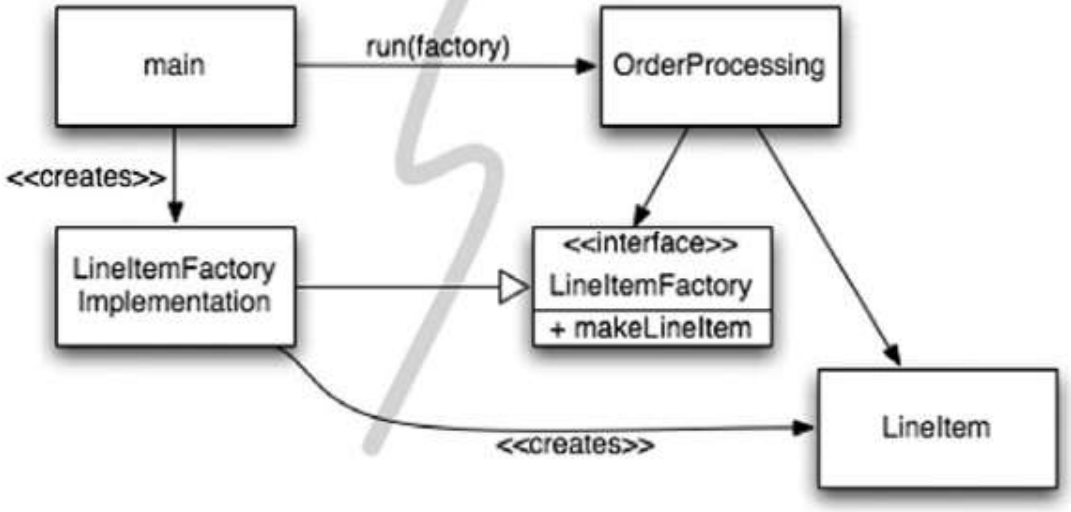

Figure 11-2

Separation construction with factory

  - Dependency Injection (DI) separa construção do uso.
    - Inversion of Control (IOC) aplicado à gestão de dependências
    - Classe recebe na instanciação os objetos de que precisa pra funcionar.
      - De preferência recebe os objetos conforme a necessidade.

Escalabilidade

- Não tem como fazer sistema grande e ótimo de código desde o começo

- Bom evitar acoplamento a objetos grandes.
- Cross-cutting Concern
    - Alguns aspectos são gerais em um software, como a forma de persistência
    - Deve ser característica adicionada ao código de forma não-invasiva.
    - *Aspect Oriented programming (AOP)*:
        - Algumas formas de fazer isso em Java:
            1. Java Proxies
            2. Frameworks AOP puros
            3. AspectJ Aspects
        - Em python dá pra fazer isso com decorators - separar cross-cutting concern da lógica de negócio
        - Esta forma de organizar o código é altamente recomendada
            - Evita **BDUF** (*Big Design Up Front*)
            - Facilita alterações e crescimento.
            - Facilita teste
- "*Uma boa arquitetura tem domínios modularizados de responsabilidades*"

```
# `The aspect (logging) is separate from the business logic`
def log_execution(func):
    def wrapper(*args, **kwargs):
        print(f"Calling {func.__name__}...")
        result = func(*args, **kwargs)
        print(f"{func.__name__} finished.")
        return result
    return wrapper

# Business logic is clean
@log_execution
def process_data(data):
    print(f"Processing: {data}")

process_data("Sample Data")
```

Otimização de Tomada de Decisões

- Modularização deste forma permite tomada de decisões decentralizadas
- É sempre bom adiar tomada de decisões até o último momento possível
    - Para ter a melhor informação
- Padrões só fazem sentido se eles adicionarem valor
- **DSL** *(Domain Specific Language)*
    - pequenas linguagem de scripting para facilitar comunicação
    - Um nível de abstração acima que pode tornar sistema mais explícito

"*Use the simplest thing that can possibly work*"