UNIVERSIDADE ABERTA
www.uab.pt

UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO

Mestrado em Engenharia Informática e Tecnologia Web

Arquitetura e Padrões de Software

(22304)

Ano letivo 2025/2026

Aplicação do Padrão de Criação Factory Method no ProPlan

Síntese

ProPlan é um módulo *Activity Provider* que apresenta ao aluno uma série de desafios de gestão de projeto simulados, baseados em cenários realistas.

André Sousa

1300012@estudante.uab.pt

Aplicação do Padrão de Criação Factory Method no ProPlan – Activity Provider

1. Introdução

A evolução da arquitetura de software para sistemas modulares e de baixa dependência reforça a relevância dos padrões de desenho como mecanismos formais de organização, comunicação e extensibilidade (Gamma et al., 2000). No contexto da arquitetura Inven!RA, um *Activity Provider* deve disponibilizar serviços REST que implementam uma lógica consistente, facilmente testável e extensível. Esta necessidade é particularmente evidente no endpoint *analytics_url*, cuja função é fornecer dados analíticos relativos ao desempenho ou comportamento de uma instância de atividade.

Para cumprir os objetivos da Unidade Curricular e demonstrar capacidade de aplicar padrões de criação, foi selecionado o padrão Factory Method, devido à sua adequação para encapsular pontos de variação relacionados com a construção de objetos e isolamento das suas dependências concretas.

2. Identificação do problema

Na versão inicial do projeto, o *endpoint*:

POST /analytics-proplan

construía diretamente a estrutura JSON de resposta, acedendo ao ficheiro *analytics_url.json* e gerando valores analíticos no próprio controlador *Flask*. Este desenho tem três limitações:

1. **Acoplamento elevado**
   O controlador conhece a origem dos dados e a forma como estes são construídos.
2. **Dificuldade de evolução**
   Se no futuro for necessário trocar a origem dos dados (JSON → Base de Dados), o endpoint teria de ser modificado.
3. **Ausência de um ponto de variação definido**
   A criação do repositório de dados não está abstraída nem isolada.

Segundo a literatura, estes sintomas correspondem a violações clássicas dos princípios de abertura/fecho (OCP) e inversão de dependência (DIP), frequentemente mitigados através do padrão **Factory Method** (Gamma et al., 2000).

3. Aplicação do padrão *Factory Method*

O objetivo principal da aplicação do *Factory Method* foi isolar a decisão sobre qual repositório de analytics deve ser instanciado, deixando o controlador a depender apenas de uma interface abstrata (*AnalyticsRepository*).

O desenho final encapsula a criação de instâncias numa classe dedicada (RepositoryFactory), cujo método:

create_analytics_repository()

é o verdadeiro *Factory Method*.
Nesta fase, a fábrica devolve sempre:

JsonAnalyticsRepository()

mas o desenho prepara o sistema para evoluir, podendo no futuro devolver:

SqlAnalyticsRepository()

MockAnalyticsRepository()

ApiBasedAnalyticsRepository()

sem alterar o endpoint *Flask*, implementando assim o princípio de extensibilidade previsto na literatura e nas orientações da UC.

4. Arquitetura proposta

A aplicação do padrão implicou a criação de três elementos estruturantes:

4.1. Interface – *AnalyticsRepository*

Define o contrato para qualquer mecanismo de obtenção de analytics.

```
class AnalyticsRepository(ABC):
    @abstractmethod
    def get_analytics(self, activity_id: str) -> list[dict]:
```

4.2. Implementação concreta – JsonAnalyticsRepository

Lê o esquema *analytics_url.json* e devolve dados analíticos simulados.

```
class JsonAnalyticsRepository(AnalyticsRepository):
    def get_analytics(self, activity_id: str) -> list[dict]:
```

4.3. Fábrica – *RepositoryFactory*

Responsável por decidir qual repositório instanciar.

```
class RepositoryFactory:
    @staticmethod
    def create_analytics_repository() -> AnalyticsRepository:
        return JsonAnalyticsRepository()
```

4.4. Controlador Flask (refatorado)

```
@app.post("/analytics-proplan")
def analytics_proplan():
    data = request.get_json(silent=True) or {}
    activity_id = data.get("activityID")
    repo = RepositoryFactory.create_analytics_repository()
    analytics = repo.get_analytics(activity_id)
    return jsonify(analytics)
```

O endpoint deixa de conhecer implementações concretas e passa a depender apenas da abstração *AnalyticsRepository*.

5. Diagrama de Componentes (segundo a norma UML 2)

Figura X – Diagrama de Componentes do padrão Factory Method aplicado ao serviço /analytics-proplan.

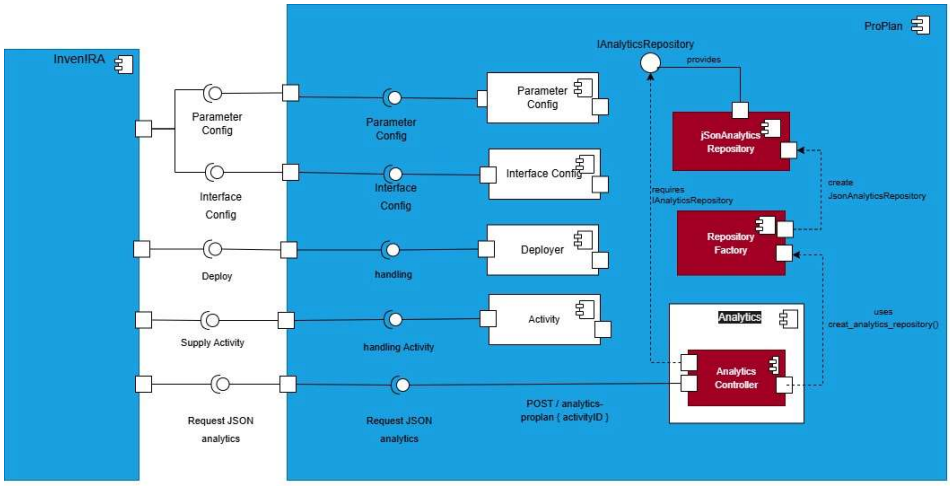

Figura 1 - Diagrama Componentes

Características verificadas, de acordo com a norma:

- Componentes representados com símbolo UML2.
- Interface *IAnalyticsRepository* modelada como elemento UML separado.
- Interface providenciada por *JsonAnalyticsRepository* (*lollipop*).
- Interface requerida pelo porto *repoPort* do controlador *Flask* (*socket*).
- Componentes **6. Diagrama de Sequência (só o padrão de criação)**

Figura Y – Sequência do processo do Factory Method para /analytics-proplan.

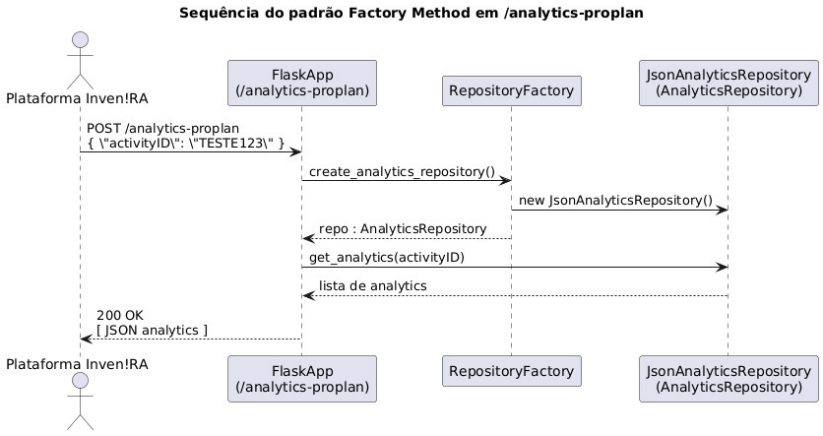

Figura 2 - Diagrama Sequência

O diagrama representa apenas:

- o pedido da Plataforma Inven!RA,
- a invocação da fábrica,
- a criação de *JsonAnalyticsRepository*,
- a chamada *get_analytics*(*activityID*),
- a resposta 200 OK.

O diagrama não inclui nenhum outro endpoint, cumprindo rigorosamente a orientação do docente.

7. Validação segundo as indicações da Unidade Curricular

7.1. Diagrama de Componentes

- Inclui todos os subcomponentes relevantes para o padrão: *FlaskApp, RepositoryFactory, AnalyticsRepository* e *JsonAnalyticsRepository*.
- Inclui portos explícitos e interfaces providas e requeridas, conforme a norma UML.

7.2. Diagrama de Sequência

- Descreve apenas a dinâmica do *Factory Method* aplicado ao endpoint */analytics-proplan*.
- Não inclui o funcionamento geral do *Activity Provider*, cumprindo a exigência de foco estrito.

8. Conclusão

A introdução do padrão **Factory Method** no *Activity Provider ProPlan* permitiu alcançar um desenho mais robusto, extensível e conforme às boas práticas de engenharia de software. O controlador passa a depender apenas de abstrações, reduzindo acoplamento e facilitando a evolução futura do sistema. A representação estrutural e comportamental (através dos diagramas de componentes e de sequência) demonstra formalmente a aplicação do padrão, alinhando-se com as orientações da UC e com referências clássicas da engenharia de software.

Referências (ABNT)

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. *Padrões de projeto: soluções reutilizáveis de software orientado a objetos*. Porto Alegre: Bookman, 2000.

MORGADO, Leonel; CASSOLA, Fernando. *Activity Providers na Inven!RA*. Lisboa: Universidade Aberta, 2022.

FAKHROUTDINOV, Kirill. *UML Component Diagrams: Reference*. uml-diagrams.org, 2025.

SOUSA, André. *ProPlan – Activity Provider*. GitHub, 2025. Disponível em: https://github.com/AndreMacielSousa/proplan-activity-provider. Acesso em: 26 nov. 2025.