
**Universidade Estadual de Feira de Santana**
**Departamento de Exatas**
**Curso de Engenharia de Computação**
**Disciplina: Algoritmos e Programação II**
**Professor: Rafael Tosta Santos**

# Padrões de Projeto: Iterator, Factory Method e Template method

Luis Fernando do Rosário Cintra
Vanderleicio Carvalho Leite Junior
Wagner Alexandre Ferreira Junior

# Iterator

# Iterator

- O Iterator é um padrão de projeto comportamental que permite a você percorrer elementos de uma coleção sem expor as representações dele (lista, pilha, árvore, etc.)

# Onde se aplica

- Utilize o padrão Iterator quando sua coleção tiver uma estrutura de dados complexa por debaixo dos panos.
- Utilize o padrão para reduzir a duplicação de código de travessia em sua aplicação.
- Utilize o Iterator quando você quer que seu código seja capaz de percorrer diferentes estruturas de dados ou quando os tipos dessas estruturas são desconhecidos de antemão.

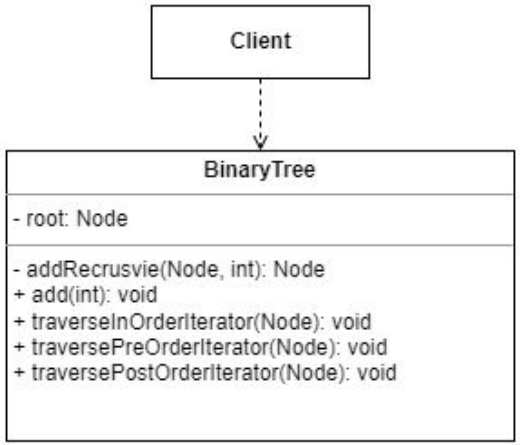
# Problema

Como você faz a travessia dos elementos de uma estrutura de dados complexas sequencialmente, tais como uma árvore?

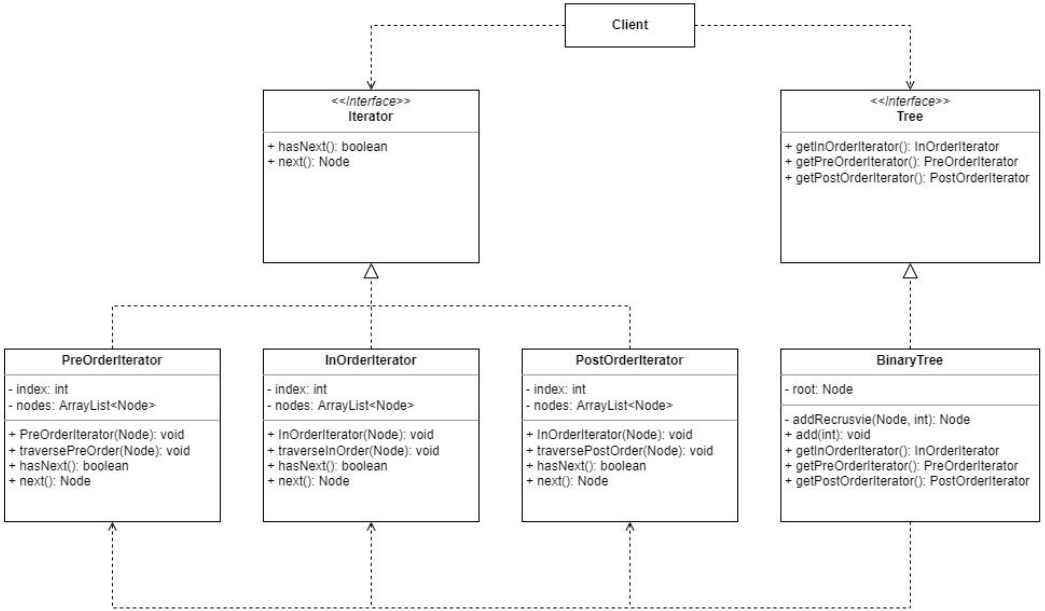
# Solução proposta pelo Iterator

- Extrair o comportamento de travessia de uma coleção para um objeto separado chamado um iterador.

# Diagrama- Sem uso do Iterator



# Diagrama - Com uso do Iterator



# Codificação - Sem uso do Iterator

# Codificação - Com uso do Iterator

# Factory Method

# Factory Method

- Padrão de criação centrado no escopo das classes;
- Pretende definir uma interface para a criação de um objeto, mas deixa a decisão de qual classe será instanciada para as subclasses, fazendo com que seja possível adiá-la;

# Onde se aplica

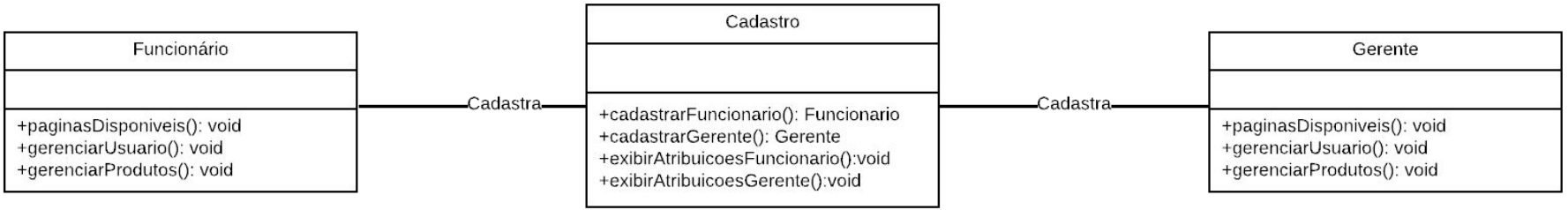
- Deve ser usado quando: uma classe não sabe a classe dos objetos que criará ou até mesmo quer que suas subclasses especifiquem os objetos criados;

- Exemplo: Num sistema de gerenciamento a classe responsável pelo cadastro de usuários não tem como saber previamente quais tipos de usuários ela irá criar. Ter que implementar cada um separadamente torna o código repetitivo e dificulta alterações e/ou extensões.

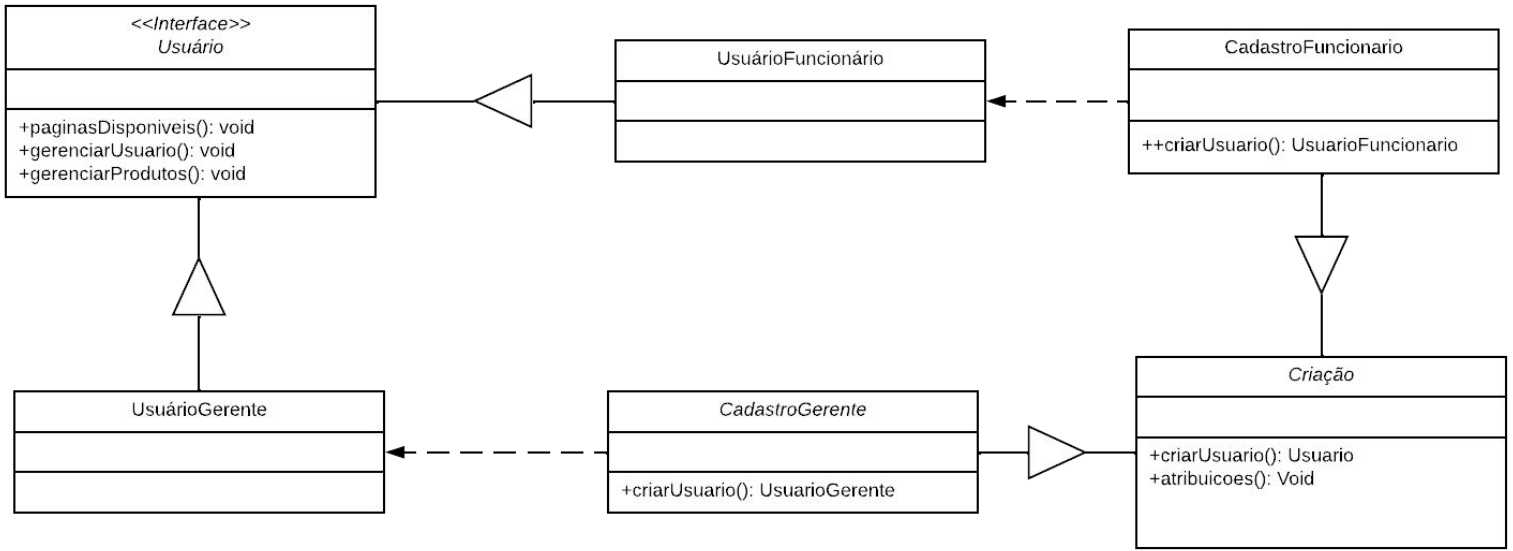
# Solução proposta pelo Factory Method

- Criar uma interface para os objetos “Usuário” e uma classe abstrata “Criadora”, que será herdada pelas classes concretas de criação, e essas serão responsáveis por criar os usuários de cada tipo.

# Diagrama sem uso do Factory Method



# Diagrama com uso do Factory Method



# Template Method

# Template Method

- Padrão comportamental baseado em herança.
- Permite que as subclasses possam sobrescrever etapas específicas do algoritmo sem modificar sua estrutura.
- Diminui a duplicidade do código.

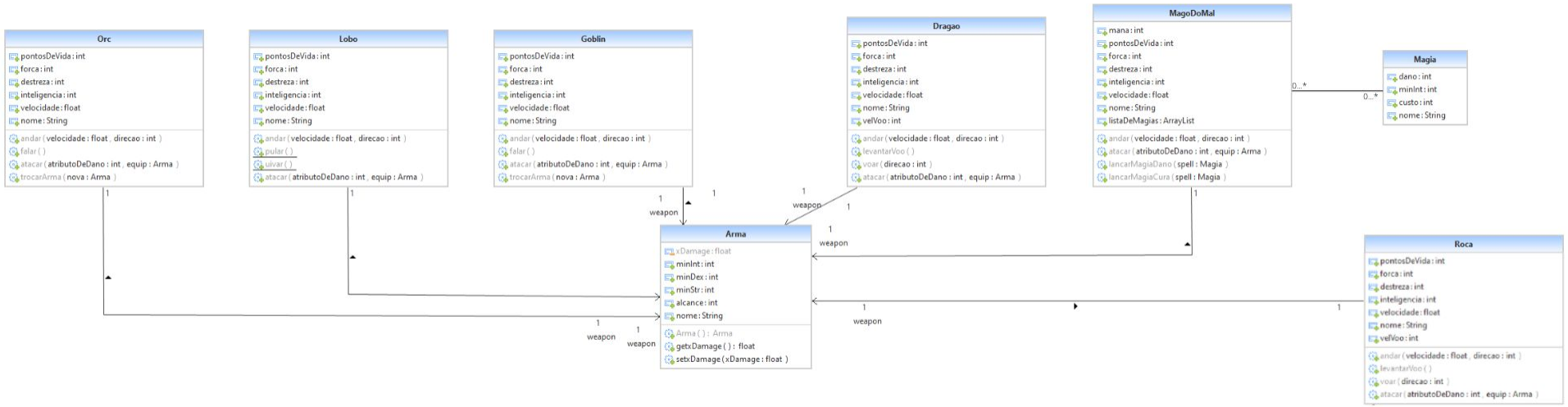
# Problema / Onde se aplica

- Tenho classes com comportamentos e/ou atributos muito parecidos/iguais, mas os métodos lidam com “problemas” diferentes.

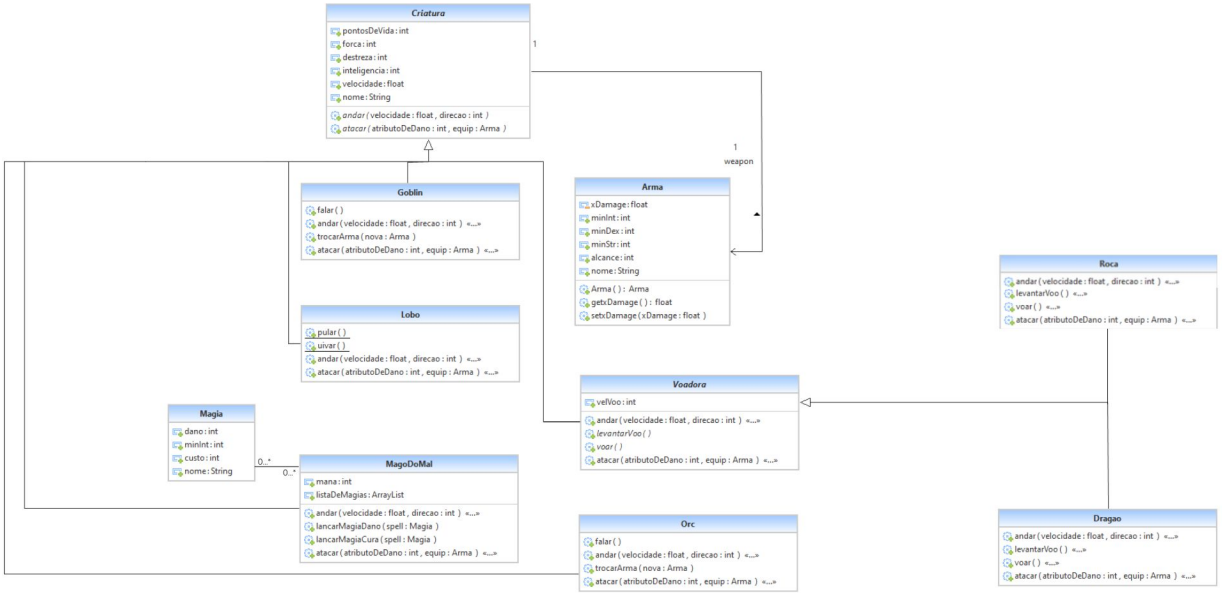
# Solução proposta pelo Template Method

- Criação de uma superclasse abstrata que possua todos os atributos (se houver) e etapas em comum das classes, as etapas que possuem uma implementação diferente, devem ser abstratas para que cada subclasse possa implementar da própria maneira.

# Template Method (Exemplo de UML SEM o padrão)



# Template Method (Exemplo de UML COM o padrão)



# Codificação - Sem uso do Template Method

# Codificação - Com uso do Template Method

# Referências

**Padrões de Projeto (refactoring.guru)**. Acesso em: 04/07/2022. Disponível em: https://refactoring.guru/pt-br/design-patterns/

I**mplementando uma árvore binária em Java | Baeldung**. Acesso em: 04/07/2022. Disponível em: https://www.baeldung.com/java-binary-tree.

**Design Patterns - Iterator Pattern (tutorialspoint.com)**. Acesso em: 04/07/2022. Disponível em: https://www.tutorialspoint.com/design_pattern/iterator_pattern.htm.

GAMMA, ERICH et al. **Padrões de Projeto–Soluções Reutilizáveis de Software Orientado a Objetos**, 2004, Ed.