Algoritmos Evolutivos Canônicos

Incluindo Algoritmos Evolutivos Canônico no ProOF

Resumo

- ProOF
  - Visão Geral
  - Diferentes linguagens de programação
  - Interface Gráfica Automática
  - Reuso de Código
- Incluindo um Algoritmo Genético

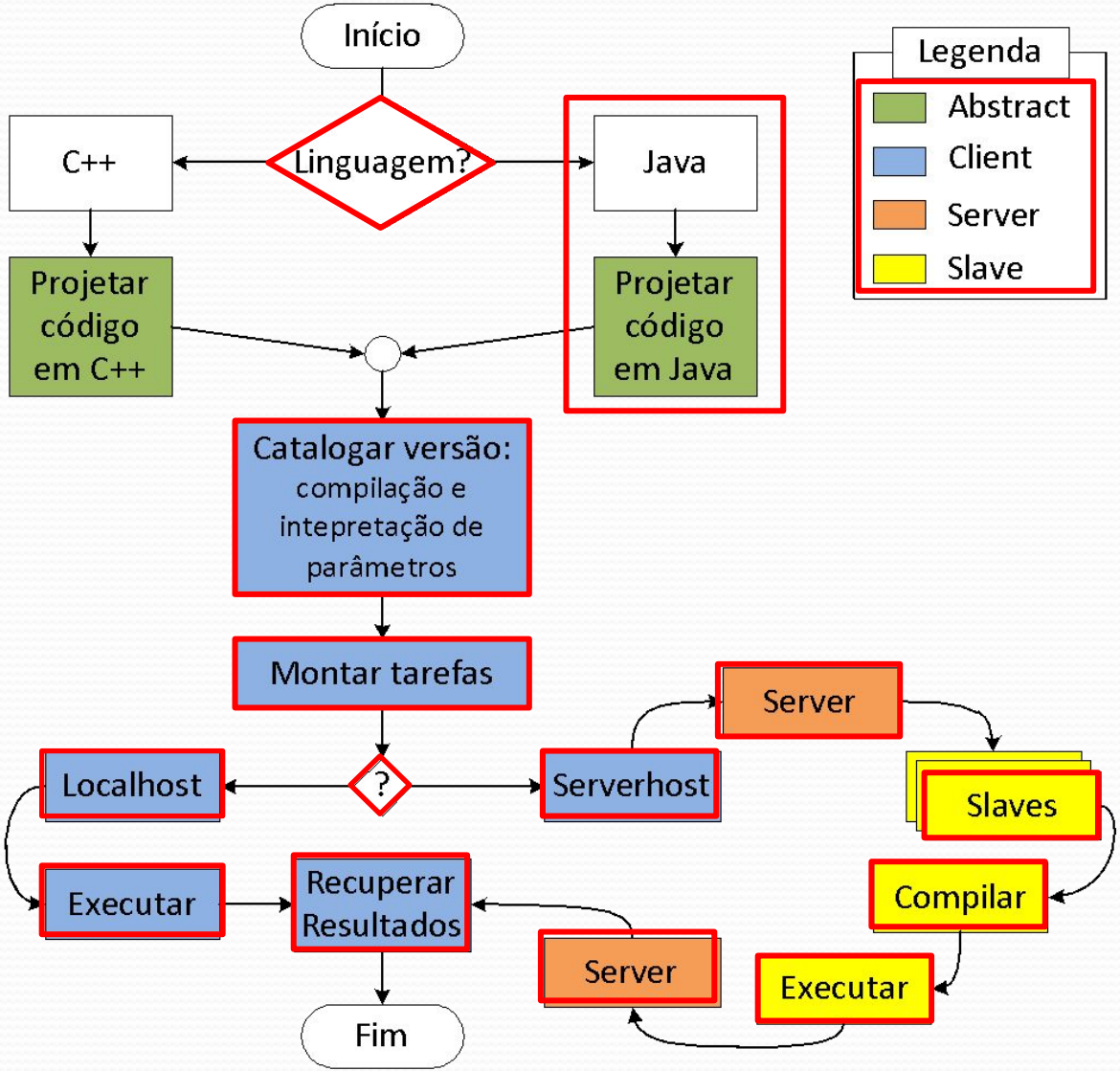
Instalação, Configuração e Teste

- Baixar e descompactar o ProOF
- Instalar Java JDK 8, NetBeans 8 e Cplex 12.5
- Configurar **`javac`**, **`jar`** e **`cplex`** no path do sistema
- Teste:
  - Catalogar versão do código
    - **C++** ou **Java**
  - Montar batches
    - Mostrar grafo de componentes
  - Execução
    - **Local host**: um computador e várias threads
    - **Server host**: vários computadores e várias threads (cluster LCR)
  - Visualizar tabela de resultados

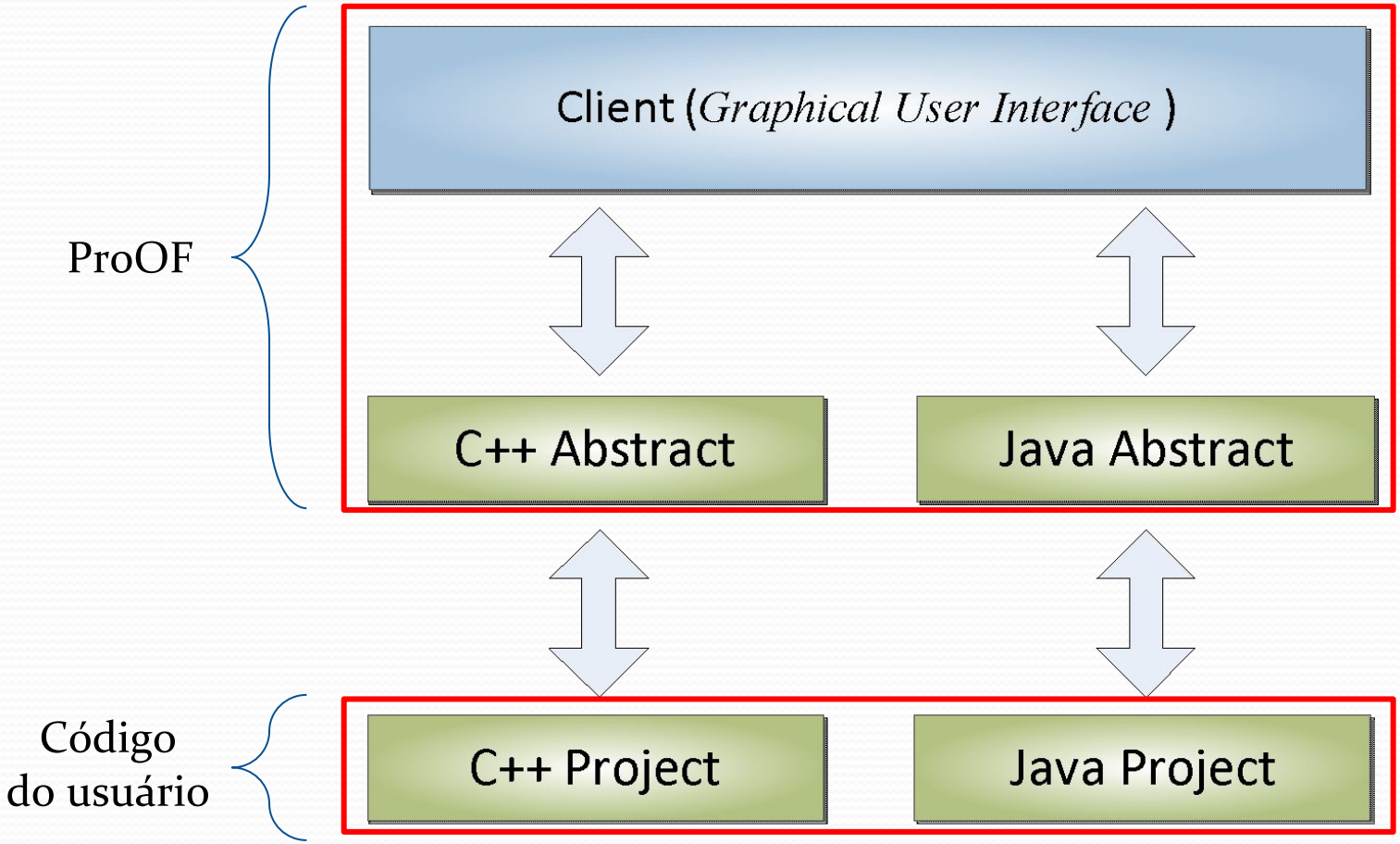
Conceitos básicos

- Objetivos do ProOF
  - Facilidade de programação
    - Interação com o ambiente centralizada
    - Poucos passos para programar
  - Robustez
    - Linguagens de programação, Métodos, Problemas, GUI automática, reuso de código
  - Agilidade
    - Montagem de testes e organização dos resultados
- Contribuições
  - Arquitetura abstrata
  - Guia o reuso de código
  - Diferentes linguagens de programação
  - *Graphical User Interface* (GUI) automática

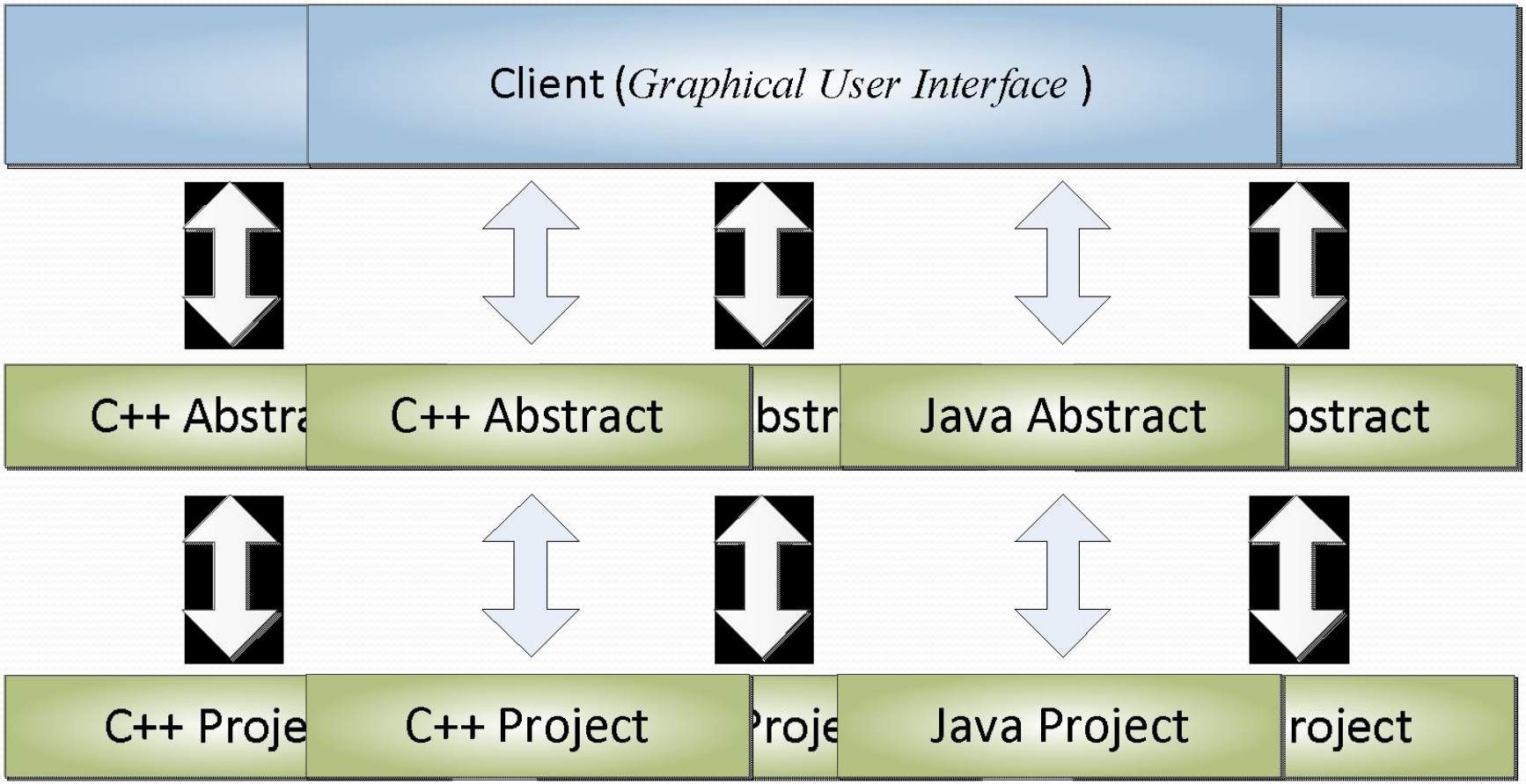
Visão Geral


Início
Linguagem?
C++
Java
Projetar código em C++
Projetar código em Java
Legenda
Abstract
Client
Server
Slave
Catalogar versão: compilação e intepretação de parâmetros
Montar tarefas
?
Localhost
Serverhost
Server
Slaves
Executar
Recuperar Resultados
Compilar
Server
Executar
Fim

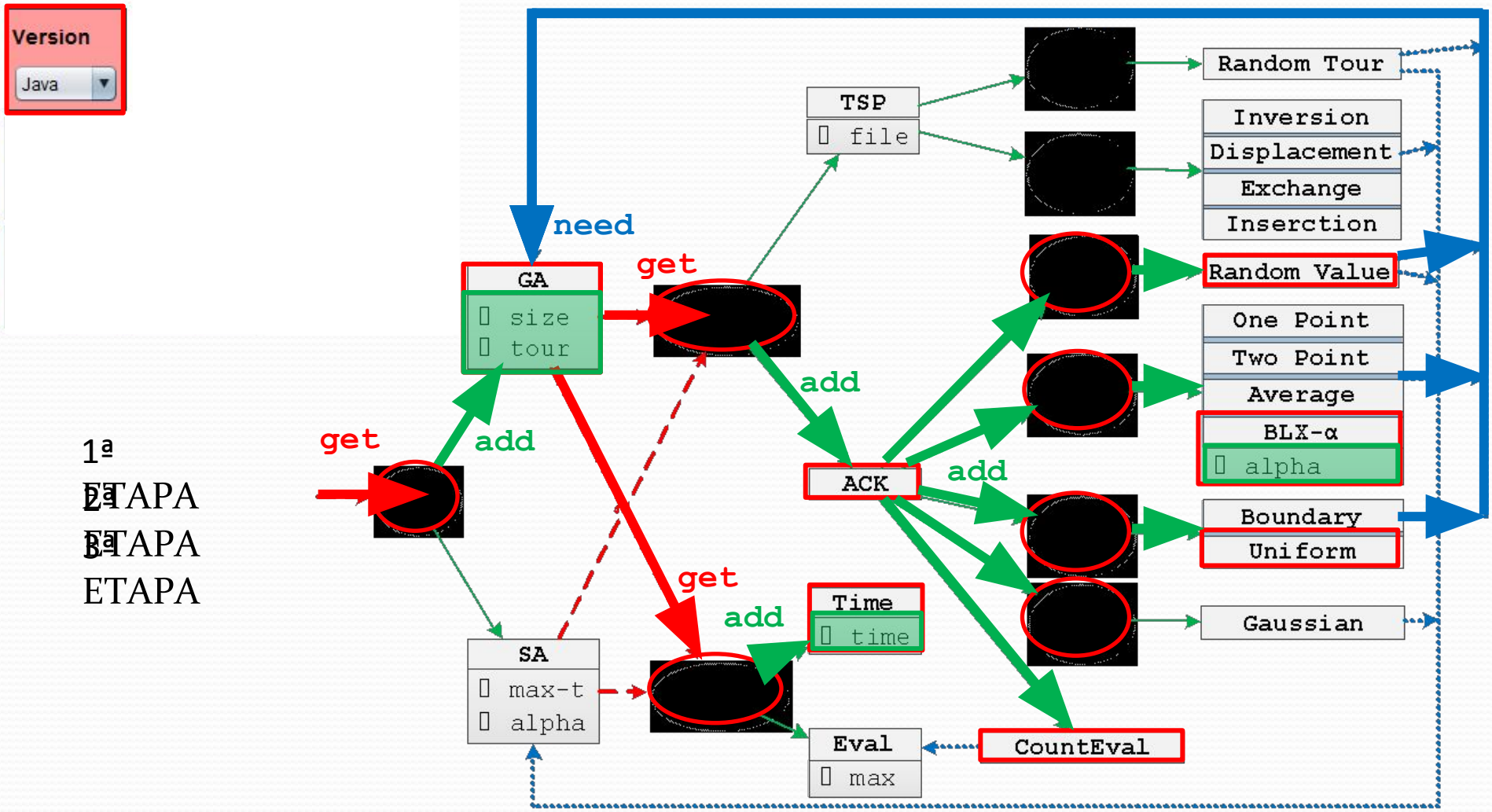
Diferentes linguagens de programação



Diferentes linguagens de programação



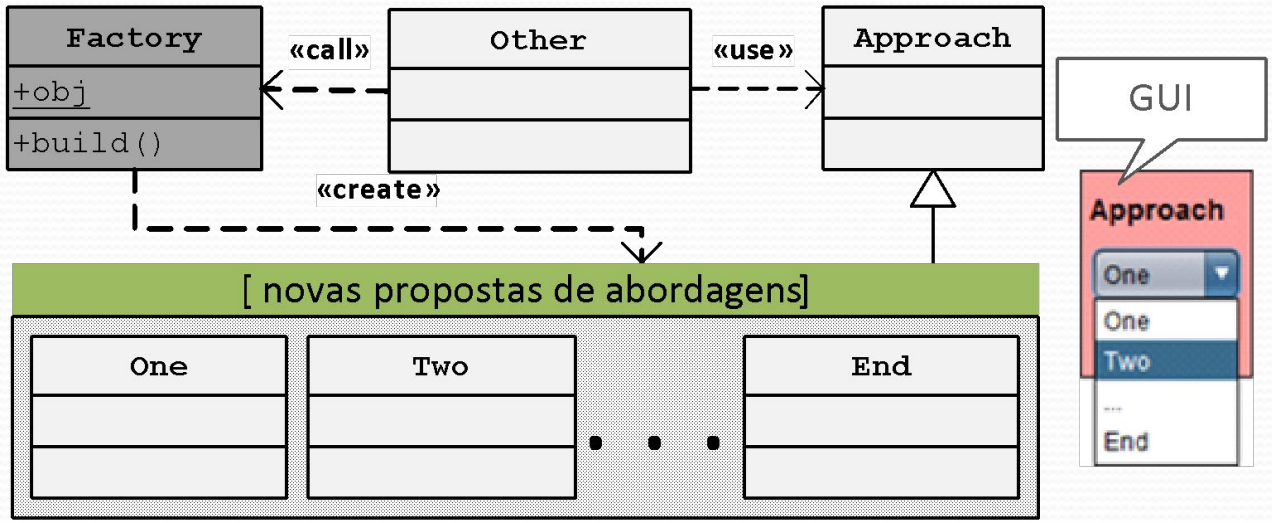
Interface Gráfica Automática



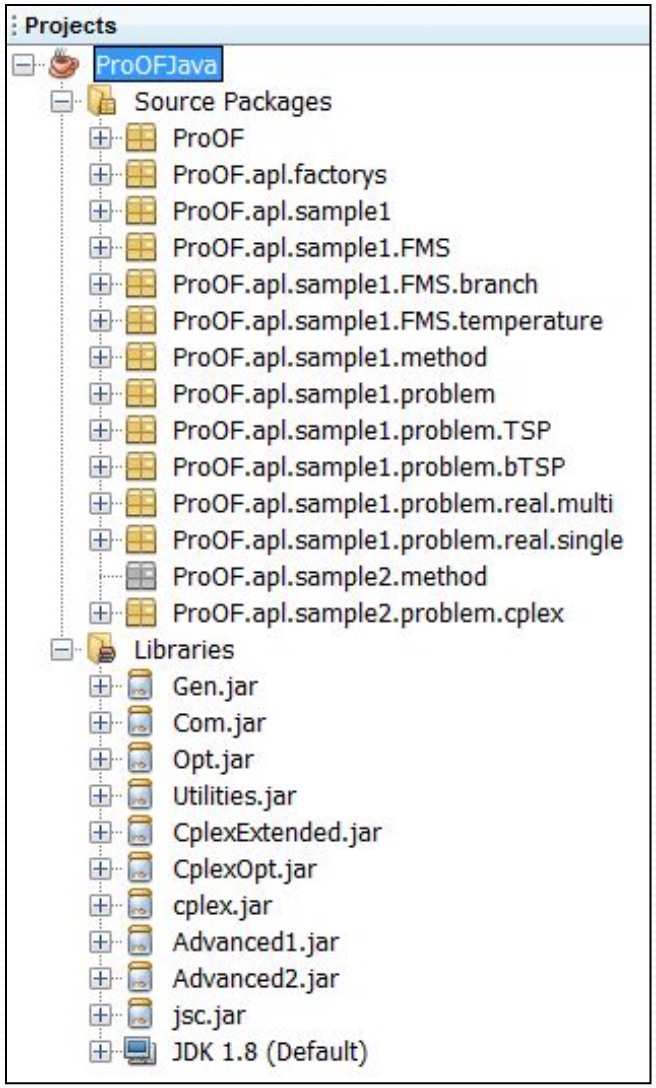
1ª ETAPA
2ª ETAPA
3ª ETAPA

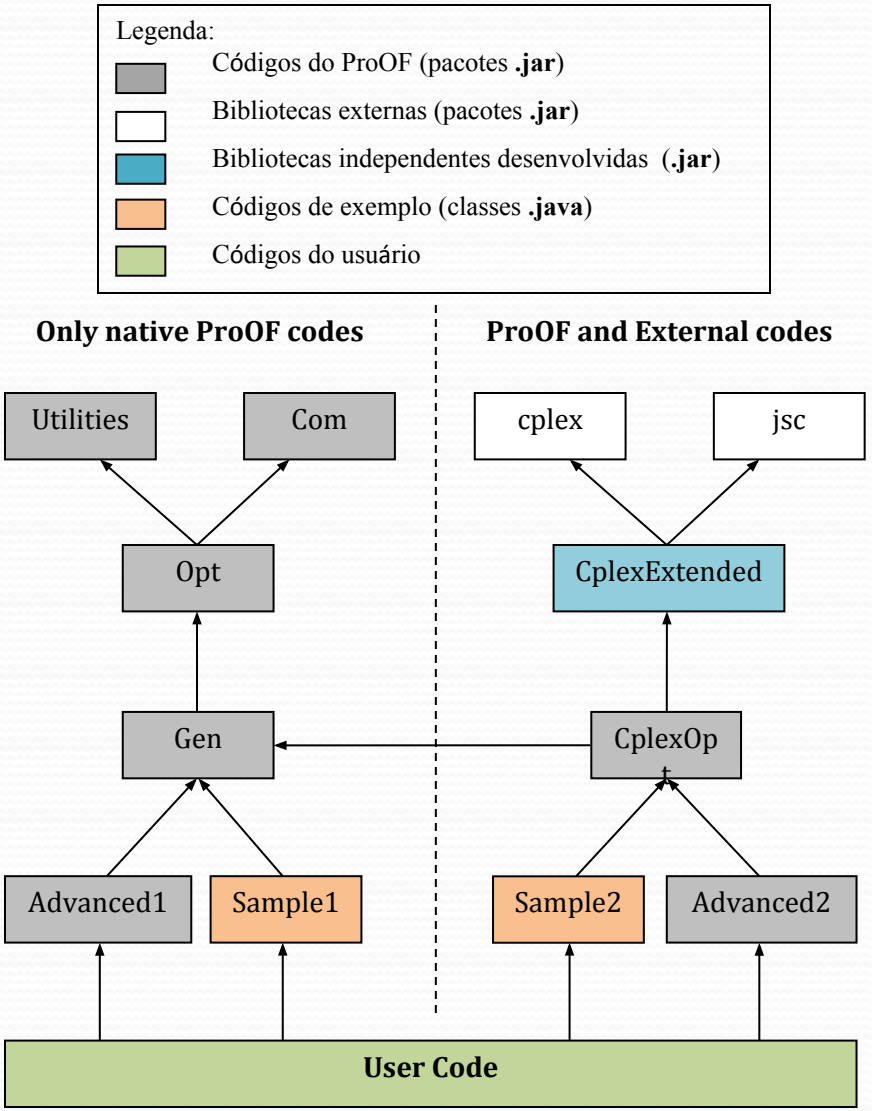
Reuso de Código

- Estrutura FMS (*Factory Method* e *Singleton*)



Desenvolvimento de código





Template para métodos

- Métodos: Heurísticos, exatos e metaheurísticos

```
01 class NovoMetodo extends Run {
02    # Declara as classes com a qual o método se relaciona ( add, get,
need)
03    #    ex: Problema, critério de parada, operadores ...
04    # Declara os parâmetro do método ( param)
05    #    ex: valores int, float, string, file, ...
06    name(){
07      return "nome do método"
08    }
09    services(){
10      # vincula as relações entre classes via  add, get, need
11    }
12    parameters(){
13      # vincula os parâmetros do método ( param)
14    }
15    execute(){
16      # implementa código de execução do método
17    }
18    results(){
19      # coleta os resultados e salva no ambiente
20    }
21 }
```

Incluindo um Algoritmo Genético

- Criar classe **`GeneticAlgorithm`**
- Modificar classe **`fRun`**

```
Genetic Algorithm
    inicia a população
    avalia a população
    repita
        seleciona pai1 e pai2
        filho ← crossover(pai1, pai2)
        mutação(filho)
        avaliação(filho)
        inserir filho na população
    até( critério de parada ser atingido )
Fim
```

Incluindo um Algoritmo Genético

1º Passo: criar uma nova classe especializada a partir da classe MetaHeuristic.

2º Passo: definir um nome para o método especializando a função membro name.

3º Passo: especializar a função membro services herdada de MetaHeuristic e nela realizar a vinculação do problema e dos operadores para o método.

4º Passo: definir os parâmetros do método especializando a função membro parameters herdada de MetaHeuristic.

5º Passo: implementar o código do método dentro da função membro execute herdada de MetaHeuristic.

6º Passo: modificar a classe do tipo Factory (fRun) para adicionar o novo método ao conjunto de métodos do ambiente.