

MatchUp

Bruno Andrade Ramos
Marcelo Gonda Stangler
Matheus Cirio
Tobias Condá Marion

Sobre

- Organizador de torneios
 - Mata-Mata;
 - Suíco;
 - Pontos Corridos.



— Desafio e Problemas

- 1 Manter alta coesão e baixo acoplamento em um domínio complexo.
- 2 Regras de pareamento (Suíço, Liga, Knockout) não podem depender diretamente do tipo de competidor.
- 3 Separar “mecânica do torneio” de “tipo de participante”.

Problemas iniciais identificados

1. Abordagem Tradicional - Sem Generics:
 - Baseada apenas em heranças;
 - Resulta em explosão de classes.

3. Ao adicionar um novo tipo (ex: Robôs):
 - SwissRobots;
 - LeagueRobots;
 - KnockoutRobots.

2. A cada algoritmo mais tipos de competidor:
Exemplos:
 - SwissPeople, SwissTeams
 - LeaguePeople, LeagueTeams

4. Consequências:
 - Alto acoplamento;
 - Código Duplicado;
 - Difícil manutenção;
 - Violação dos princípios DRY.

Solução adotada: Generics

Uso de Generics (<T extends Competitor>)

- Mecânica do algoritmo fica independente do tipo de participante;
- A lógica de Swiss.java (e outros) é escrita apenas uma vez;
- Funciona automaticamente para:
 - Person;
 - Team;
 - Qualquer novo tipo (ex.: Robots, Guild).

Benefícios

- Extensível;
- Sem duplicação;
- Baixo acoplamento;
- Regras de negócio isoladas.

— Segurança e Integridade: Factory Methods

Por que usar Factory Methods?

- Evitar torneios inválidos (ex.: misturar People com Teams).
- Garantem:
 - Estratégia (Swiss, League, etc.)
 - Lista de competidores
 - Tipo genérico
 - sejam todos compatíveis entre si.

Como funciona?

- Construtor do Tournament é privado.
- Métodos estáticos:
 - createForTeams(...)
 - createForPeople(...)
- Se adicionarmos Robot →
 - só criamos createForRobots(...).

Diagrama de Classes inicial

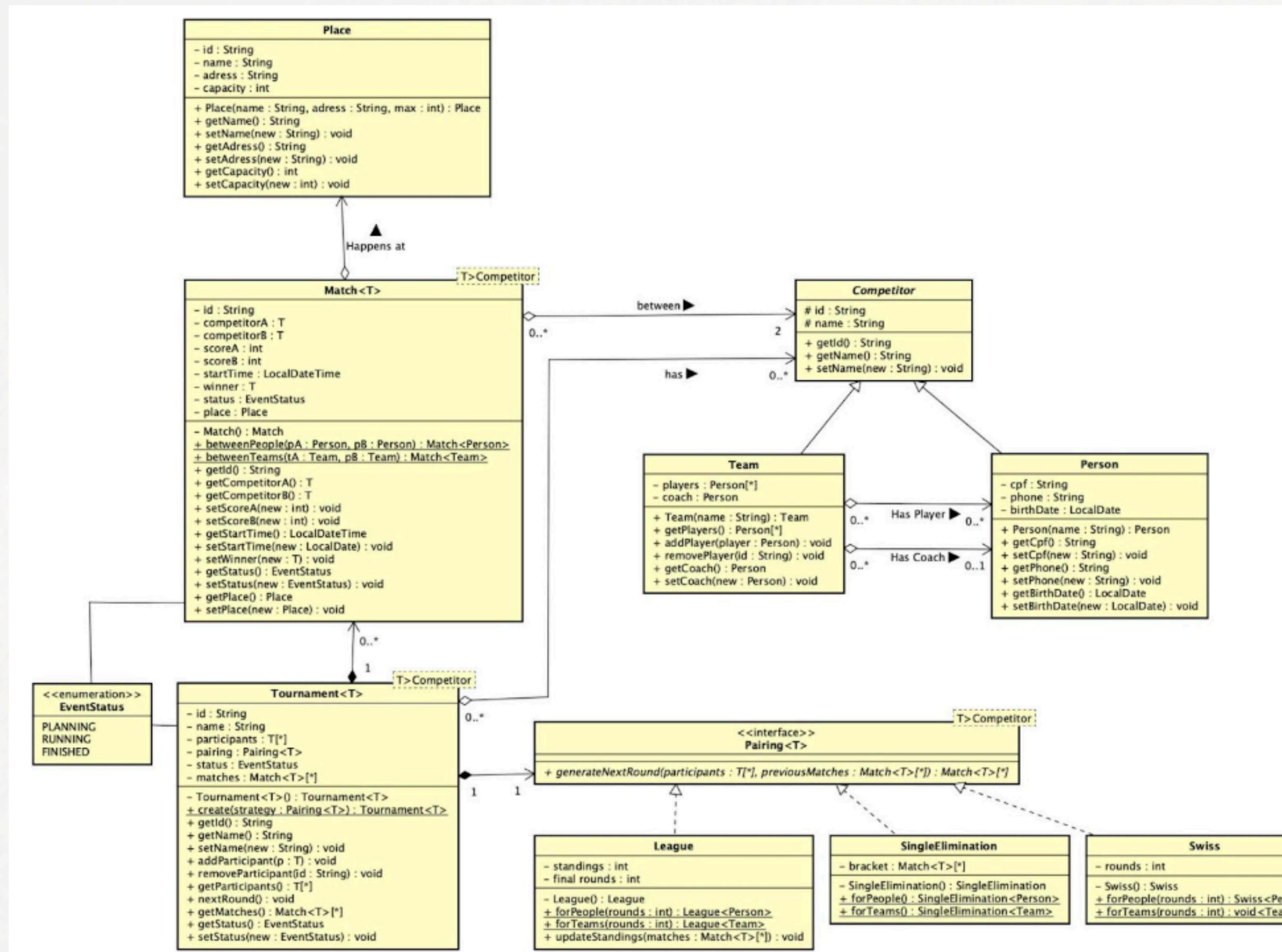


DIAGRAMA DE VIEW CONTROLLER SERVICE E MODEL

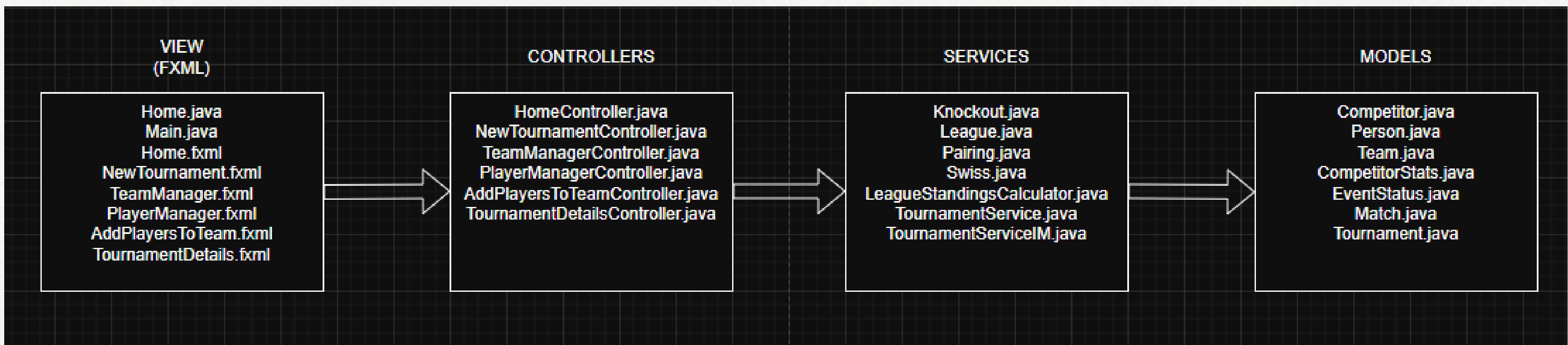
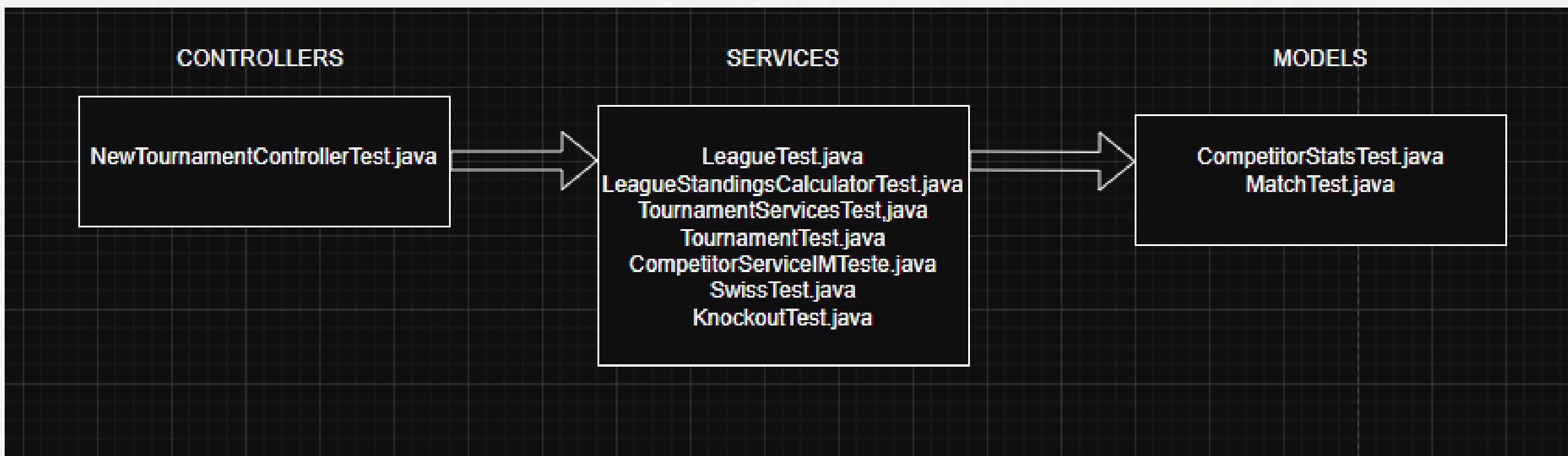


DIAGRAMA PARA OS TESTES



FIM!

OBRIGADO PELA ATENÇÃO.