

Decorator

Análise e Desenvolvimento de Sistemas - 6º Período

Relembrando o Bridge...

- No Bridge, concluímos que herança em excesso pode ser um problema. As subclasses podem crescer de maneira exponencial;
- Resolvemos o problema com composição

Composição

- Com a composição, um objeto tem uma referência com outro e pode delegar alguma funcionalidade
- Por causa disso podemos fazer com que um objeto utilize funcionalidades de várias classes. referencia outros objetos e delegue as tarefas

Herança VS Composição

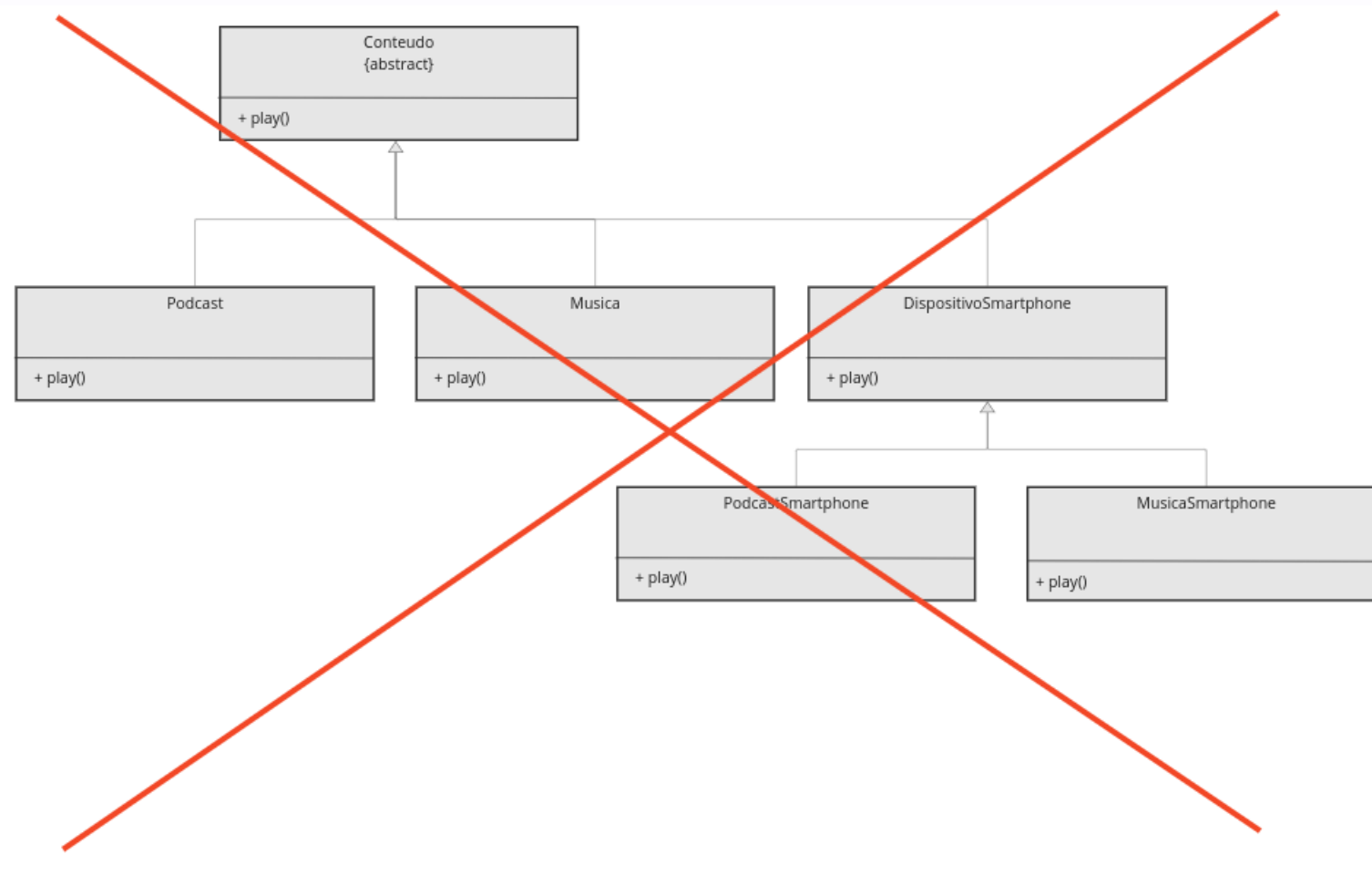
- Na herança temos um relacionamento do tipo "é um";
- Se algo muda na classe pai, é refletido nas classes filhas (acoplado);
- Na composição temos um relacionamento do tipo "tem um";
- Na composição os objetos podem assumir mais de um comportamento

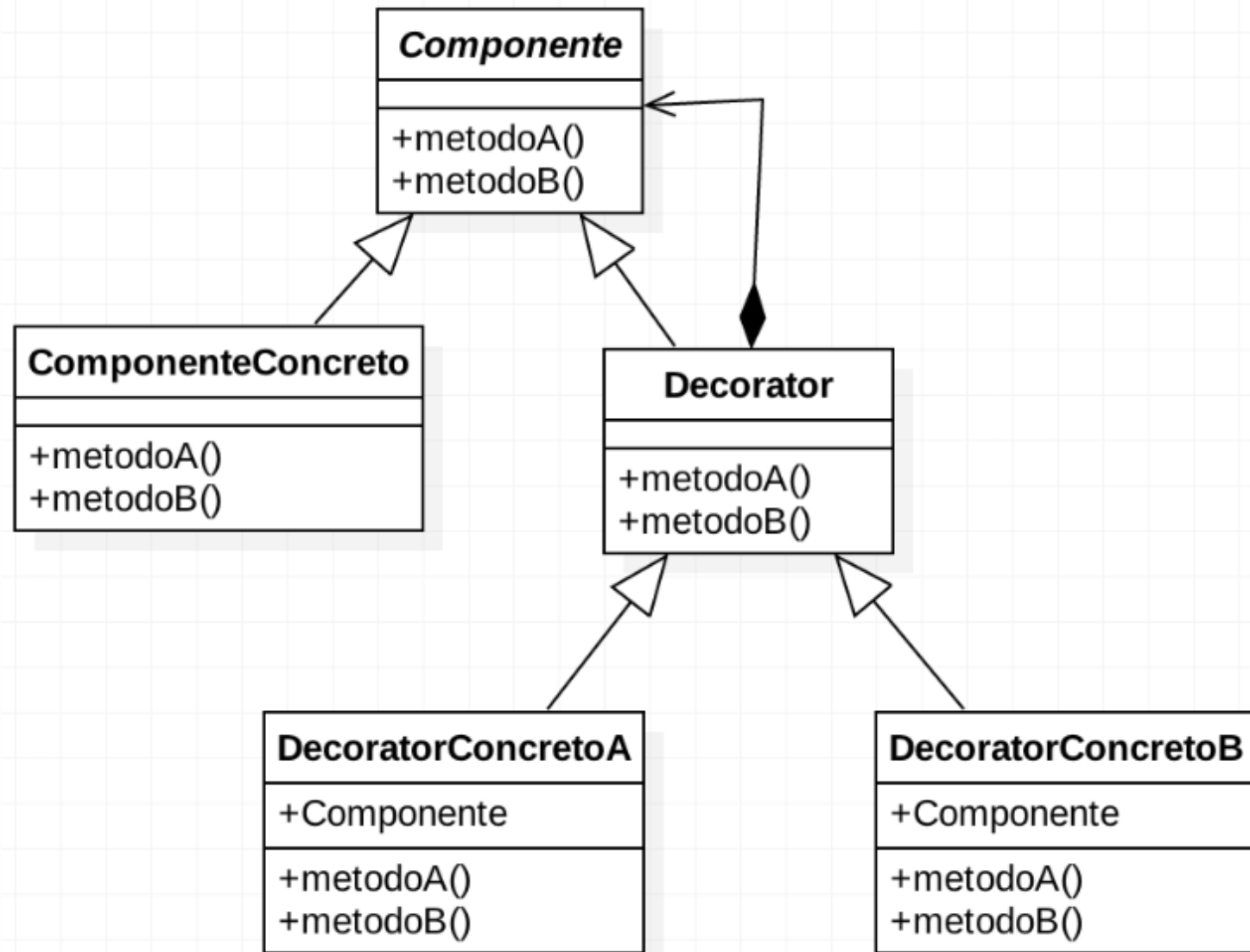
Decorator

- No Decorator vamos criar um objeto "wrapper (envoltório)", que será relacionado a outro objeto alvo e poderá delegar tarefas a ele;
- Esse objeto "wrapper" poderá alterar o resultado fazendo algo antes ou depois de passar o pedido para o alvo.

Motivações

- Usamos o decorator quando precisamos que nossos objetos realizem comportamentos adicionais em tempo de execução, sem quebrar o código que usa esses objetos.
- Podemos estender funcionalidades dos objetos de maneira flexível e sem criar um conjunto muito complexo de subclasses





Bridge VS Decorator

- O Bridge se preocupa em separar as "abstrações", permitindo o crescimento delas individualmente e fazendo uma "ponte" através de uma composição entre as abstrações. Essa ponte permite a comunicação entre os objetos dessas abstrações.

Bridge VS Decorator

- O Decorator se preocupa em adicionar novas funcionalidades aos objetos por meio de composição ou agregação.
- Ambos seguem o mesmo princípio de não crescer utilizando herança.

Composite VS Decorator

- Ambos possuem o mesmo diagrama de classes, mas o propósito é diferente;
- No Composite queremos criar hierarquias, permitindo que objetos individuais e compostos sejam tratados de maneira uniforme;
- No Decorator queremos adicionar funcionalidades aos objetos de maneira flexível, sem herança, usando um "wrapper" para estender as capacidades.