Programação Orientada a Objetos

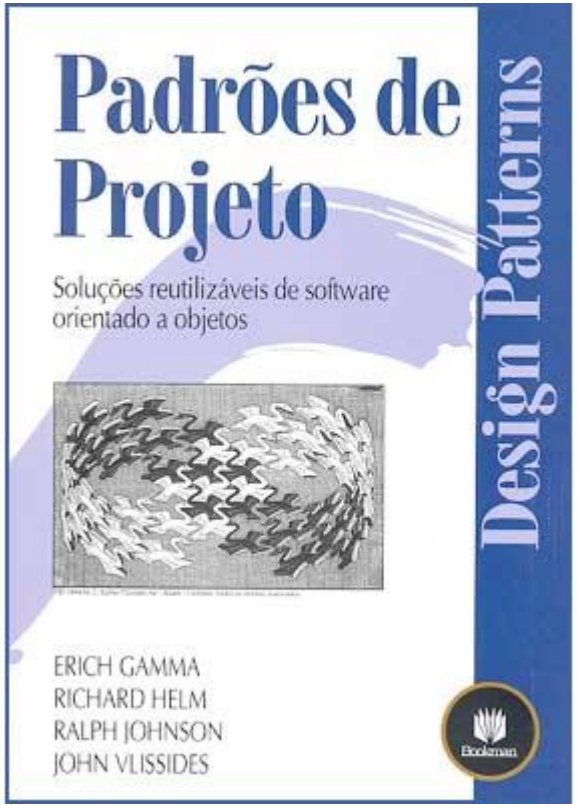
UniRitter
Laureate International Universities

Mais Padrões de Projetos GoF

UniRitter
Laureate International Universities

GoF – Padrões de Projeto / Histórico

- Em 1995 é publicado ***Design Patterns: Elements of Reusable Object-Oriented Software***, de GAMMA, HELM, JOHNSON e VLISSIDES, ou Gang of Four (GoF).



Padrões de Projeto / GoF

		Propósito		
		De Criação	Estrutural	Comportamental
Escopo	Classe	**Factory Method**	**Adapter (class)**	**Interpreter** **Template Method**
	Objeto	**Abstract Factory** **Builder** **Prototype** **Singleton**	**Adapter (object)** **Bridge** **Composite** **Decorator** **Façade** **Flyweight** **Proxy**	**Chain of Responsability** **Command** **Iterator** **Mediator** **Memento** **Observer** **State** **Strategy** **Visitor**

Factory Method – Criacional

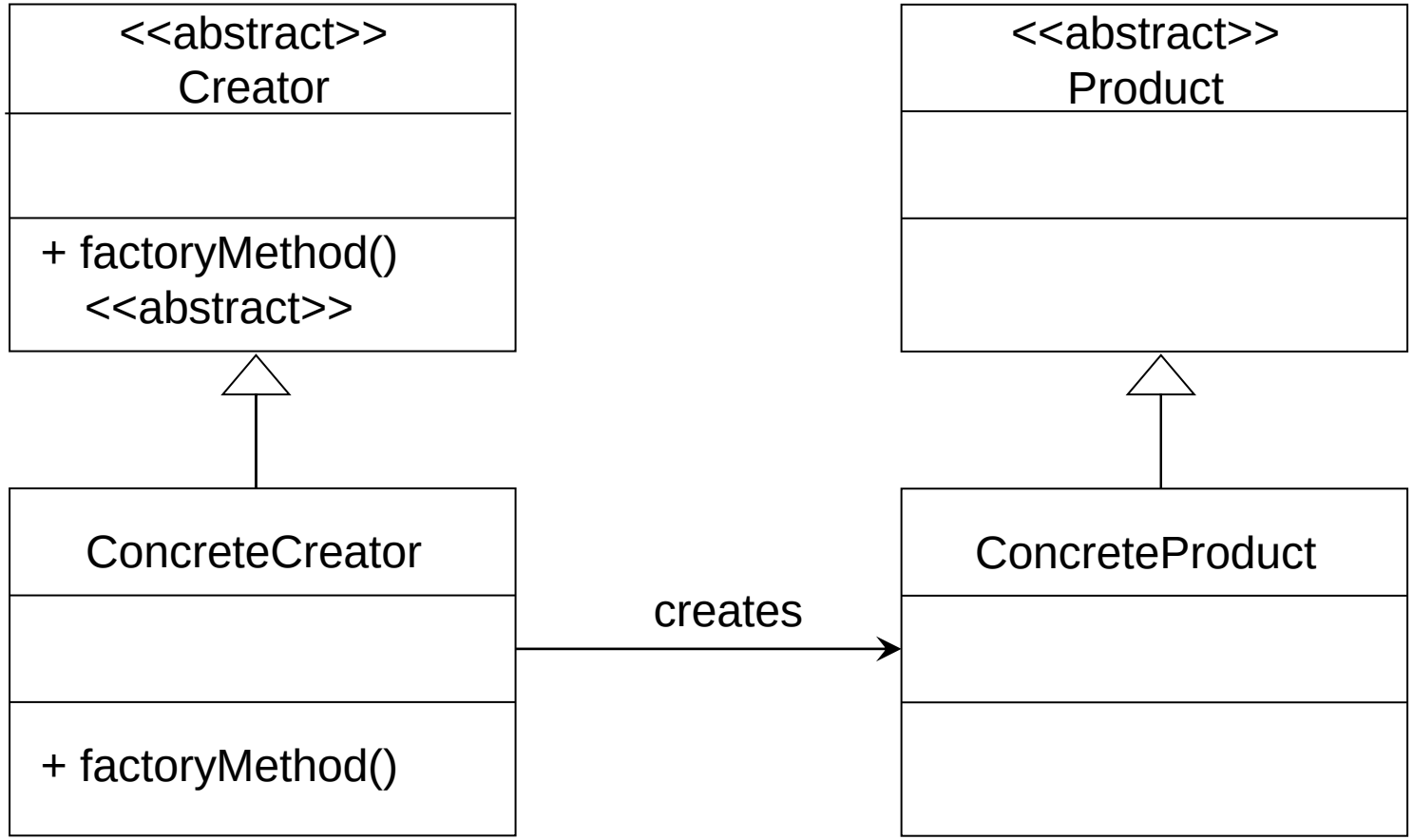
Intenção

- Definir uma interface para criação de um objeto, mas deixar as subclasses decidirem qual classe instanciar

Quando usar ?

- Uma classe não pode antecipar a classe de objetos que precisa criar
- Uma classe deseja que suas subclasses especifiquem os objetos que cria

Factory Method – Criacional



Factory Method – Criacional

```
abstract class Product{
   ...
}

class ConcreteProductA extends Product{
   ...
}

class ConcreteProductB extends Product{
   ...
}
```

```
abstract class Creator{
  public abstract Product create();
}

class ConcreteCreatorA extends Creator{
  public Product create(){
    return new ConcreteProductA() ;
 }
}

class ConcreteCreatorB extends Creator{
  public Product create(){
    return new ConcreteProductB() ;
 }
}
```

```
class GoFTest{
  public static void main( String a[] ){
    Creator c ;
    // If A is needed
      c = new ConcreteCreatorA() ;
    // else
    c = new ConcreteCreatorB() ;
    Product p = c.create() ;
  }
}
```

Abstract Factory – Criacional

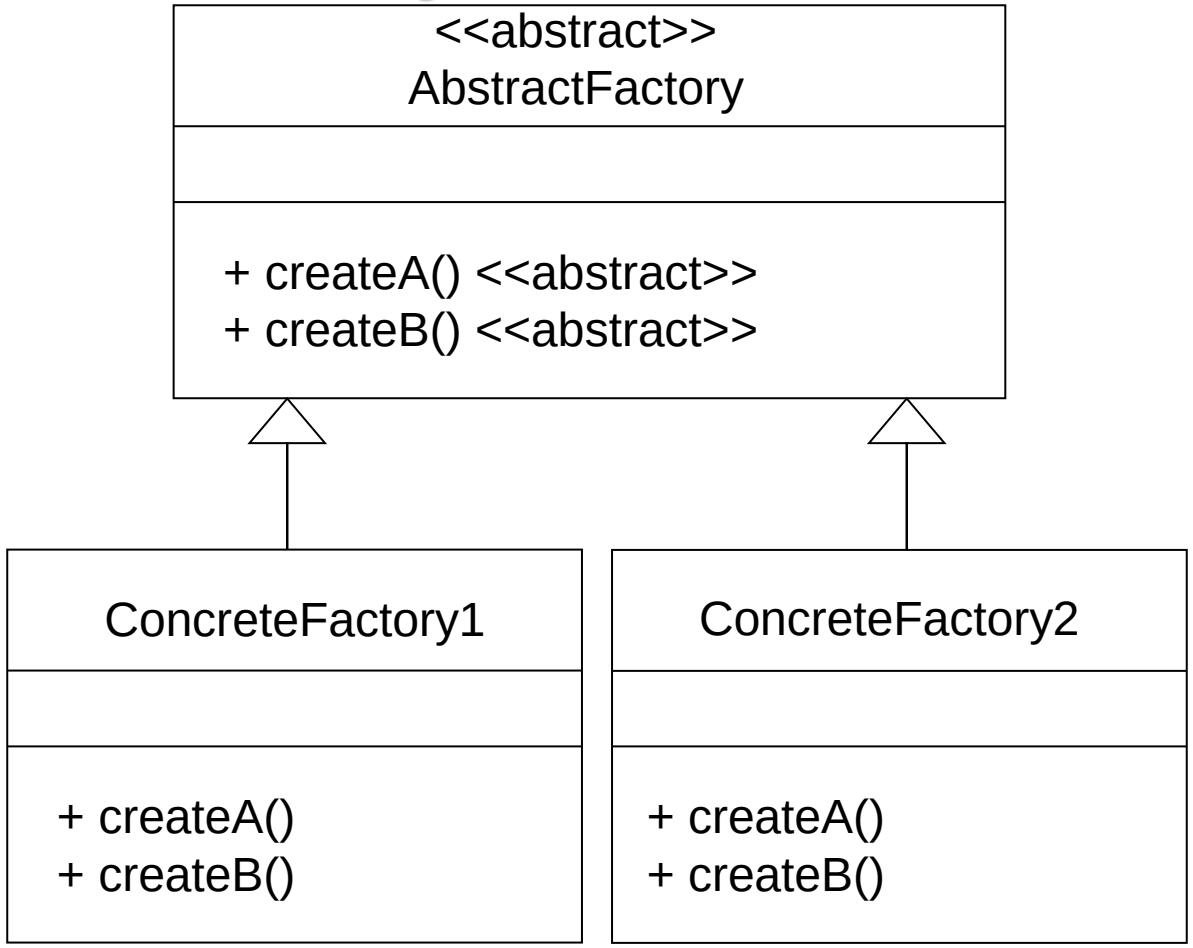
Intenção:

- Fornecer interface para criação de **famílias** de objetos relacionados ou dependentes sem especificar suas classes concretas

Quando usar ?

- Um sistema deveria ser independente de como seus produtos são criados, compostos e representados
- Um sistema deveria ser configurados com uma ou várias famílias de produtos
- Uma família de objetos é destinada a ser usada de maneira única

Abstract Factory – Criacional



Builder – Criacional

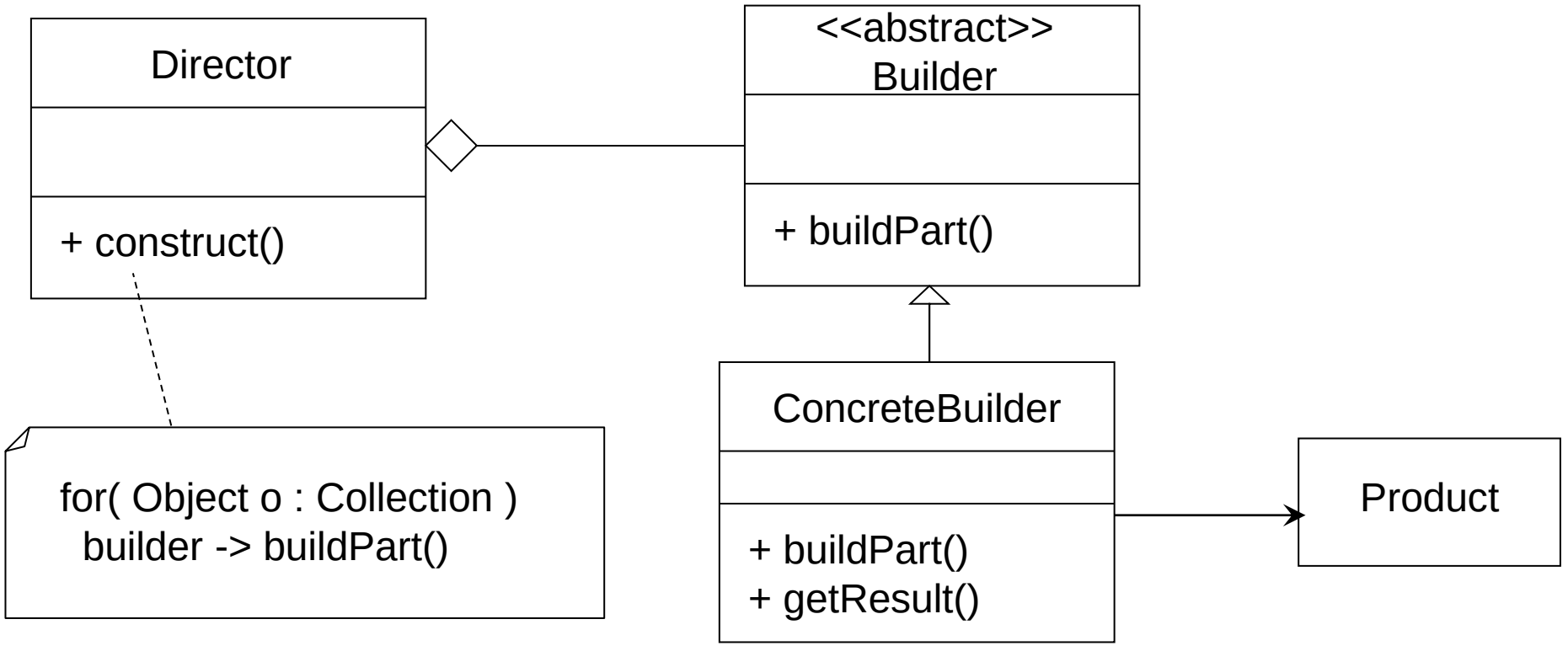
Intenção:

- Separar a construção de um objeto complexo de sua representação de modo que o mesmo processo de construção possa criar diferentes representações

Quando usar :

- O algoritmo para criar um objeto complexo deveria ser independente das partes que compõem o objeto
- O processo de construção deve permitir diferentes representações para o objeto que é construídos

Builder – Criacional



Exemplo:

```
class Director{
   ...
   private Builder b ;
   b = new ConcreteBuilder() ;
   for( Object o : Collection )
     b.buildPart( o ) ;
   Product p = b.getResult() ;
   ...
}
```

```
abstract class Builder{
  abstract buildPart( Part p );
}

class ConcreteBuilder extends Builder{
  public Part buildPart( PartA a ){...}
  public Part buildPart( PartB b ){...}
  public Product getResult(){...}
  // Product of A&B is returned
}
```

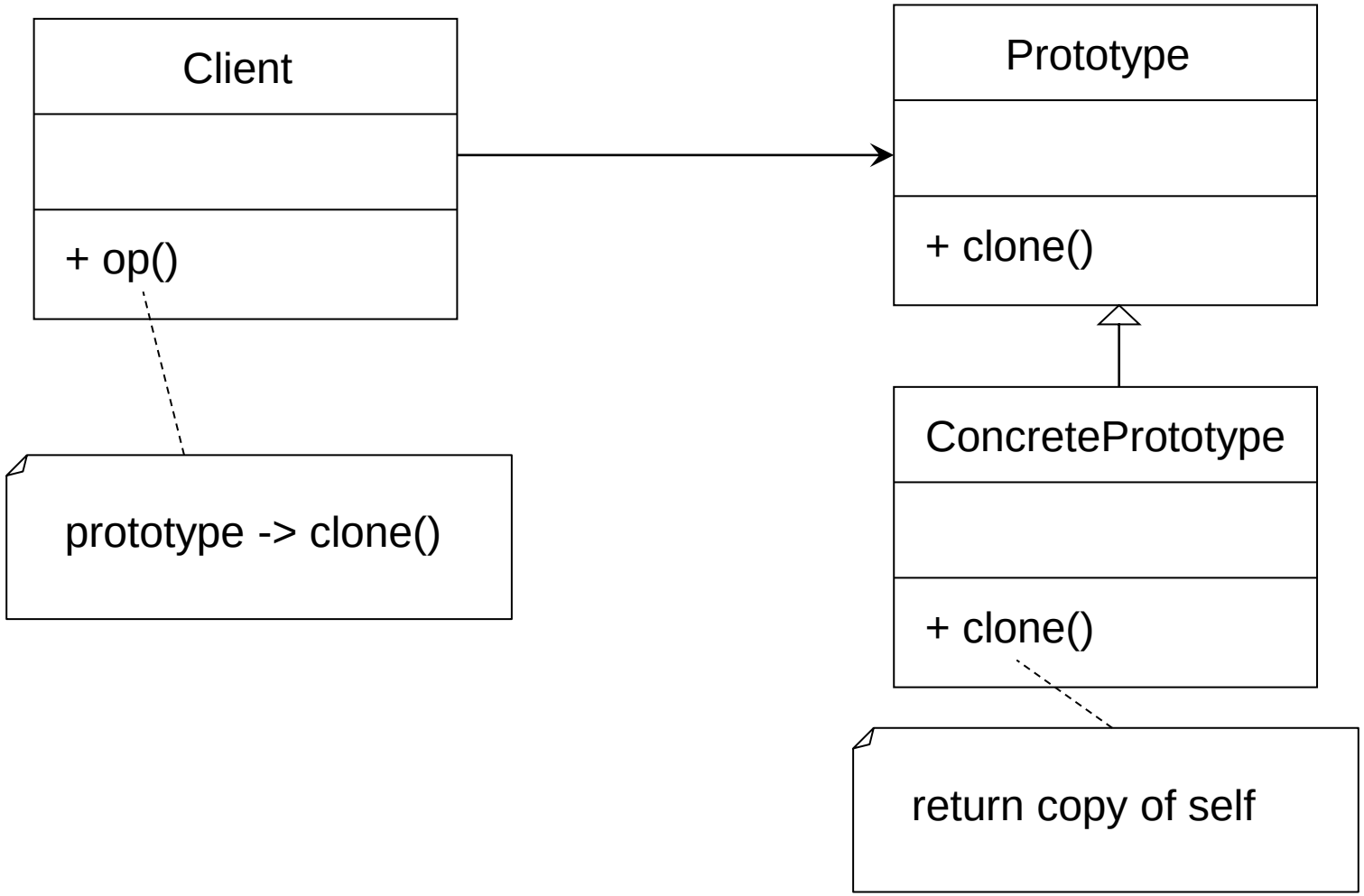
Prototype – Criacional

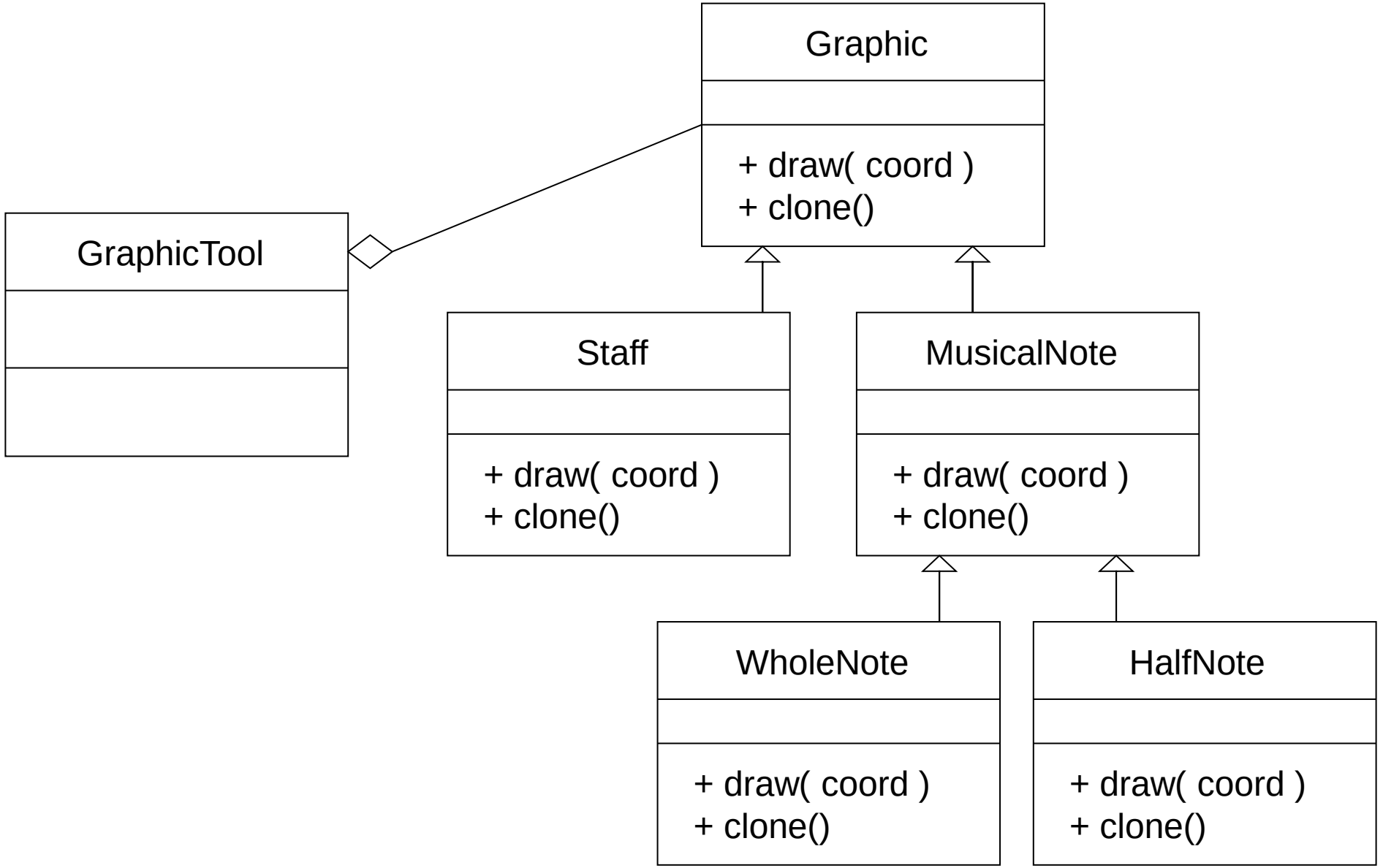
Intenção:

- Especificar os tipos de objetos a serem criados usando uma instância-protótipo. Novos objetos são criados pela cópia desse protótipo

quando:

- Classes a instanciar são especificadas em tempo de execução
- Instâncias de classes podem ter poucas combinações de estado





Singleton – Criacional

Intenção:

- Garantir que uma classe tenha somente uma instância e fornecer um ponto de acesso à instancia

quando:

- Deve haver exatamente uma instância da classe
- Deve ser facilmente acessível aos clientes em um ponto de acesso bem conhecido