

# Herança e Polimorfismo

\*Adaptado dos materiais de Phyllipe Lima (UNIFEI) por Paulo Meirelles (IME-USP).

# Dark Souls

**Dark Souls** (ダークソウル *Dāku Sōru*?) é um [jogo eletrônico](#) de [RPG de ação](#) desenvolvido pela [FromSoftware](#) e publicado pela [Namco Bandai Games](#). Lançado originalmente em setembro de 2011 para [PlayStation 3](#) e [Xbox 360](#), é um [sucessor espiritual](#) de *Demon's Souls* e a segundo título da série *Souls*. *Dark Souls* se passa no reino fictício de Lordran, onde os jogadores assumem o papel de um personagem morto-vivo amaldiçoado que inicia uma peregrinação para descobrir o destino de sua espécie. Um relançamento para [Microsoft Windows](#) foi realizado em agosto de 2012, com conteúdos adicionais não presentes em suas versões originais. Em outubro de 2012, um novo [conteúdo para download](#) foi disponibilizado para a versão de consoles, sob o subtítulo *Artorias of the Abyss*.

*Dark Souls* recebeu aclamação da crítica, com muitos citando-o como um dos [maiores jogos de todos os tempos](#). Os críticos elogiaram a profundidade de seu combate e *level design*. No entanto, a dificuldade do jogo recebeu críticas mistas, com alguns criticando-o por ser implacável demais. A versão original do jogo para Windows foi menos bem recebida, com críticas direcionadas a vários problemas técnicos. Em abril de 2013, o jogo havia vendido mais de dois milhões de cópias em todo o mundo. Duas sequências, *Dark Souls II* e *Dark Souls III*, foram lançadas em meados da década de 2010, enquanto uma versão remasterizada, *Dark Souls: Remastered*, foi lançada em 2018.

#### Índice [\[esconder\]](#)

- 1 [Jogabilidade](#)
- 2 [Enredo](#)
- 3 [Recepção da crítica](#)
  - 3.1 [Legado](#)
- 4 [Referências](#)

**Dark Souls**



<b>Desenvolvedora(s)</b>	FromSoftware
<b>Publicadora(s)</b>	Namco Bandai Games JP FromSoftware
<b>Diretor(es)</b>	Hidetaka Miyazaki
<b>Produtor(es)</b>	Hidetaka Miyazaki, Naotoshi Zin, Yuva

# Modelando os Inimigos de Dark Souls

- Suponha que o jogo Dark Souls tenha **3 tipos** de Inimigos
  - Zumbi Lerdo
  - Cavaleiro Negro
  - Cavaleiro de Prata

```
public class ZumbiLerdo {  
  
    private String nome;  
    private double vida;  
    private String tipoArma;  
  
    //Construtor  
    public ZumbiLerdo(String nome, double vida, String tipoArma) {  
        this.nome = nome;  
        this.vida = vida;  
        this.tipoArma = tipoArma;  
    }  
  
    public void atacando() {  
        System.out.println("Atacando o jogador!");  
    }  
    public void tomarDano() {  
        System.out.println("Tomando dano");  
    }  
}
```

```
public class CavaleiroNegro {  
  
    private String nome;  
    private double vida;  
    private String tipoArma;  
  
    //Construtor  
    public CavaleiroNegro(String nome, double vida, String tipoArma) {  
        this.nome = nome;  
        this.vida = vida;  
        this.tipoArma = tipoArma;  
    }  
  
    public void atacando() {  
        System.out.println("Atacando o jogador!");  
    }  
    public void ataqueRapido() {  
        System.out.println("Atacando rapidamente!");6  
    }  
    public void tomarDano() {  
        System.out.println("Tomando dano");  
    }  
}
```

# Modelando os Inimigos de Dark Souls

- Se olhar rapidamente, parecem a mesma classe!
- Mas, *CavaleiroNegro* possui o método *atacarRapido()* que o *ZumbiLerdo* não possui
- Precisamos repetir todo esse código para cada novo inimigo que queremos modelar no nosso jogo?

```
public class Inimigo {  
  
    protected String nome;  
    protected double vida;  
    protected String tipoArma;  
  
    public Inimigo(String nome, double vida, String tipoArma) {  
        this.nome = nome;  
        this.vida = vida;  
        this.tipoArma = tipoArma;  
    }  
  
    public void atacando() {  
        System.out.println("Atacando o jogador!");  
    }  
  
    public void tomarDano() {  
        System.out.println("Tomando dano");  
    }  
}
```



# Modelando o “Inimigo”

- Observe um novo modificador chamado ***protected***. Ele possui uma visibilidade mais limitada que o ***public*** e menos restrita que ***private***. Com esse modificador, somente a própria classe e as subclasses podem ter acesso a esses membros
- Normalmente esse modificador é utilizado nos membros das superclasses

# Modelando o Inimigo

- Observe que na classe *Inimigo*, não existe método *ataqueRapido()*, pois é específico do *CavaleiroNegro*. Na classe inimigo deixamos apenas o que for comum a **TODOS** os inimigos!

```
public class ZumbiLerdo extends Inimigo {  
  
    //Construtor  
    public ZumbiLerdo(String nome, double vida, String tipoArma) {  
        super(nome, vida, tipoArma);  
    }  
}
```

```
public class CavaleiroNegro extends Inimigo {  
  
    //Construtor  
    public CavaleiroNegro(String nome, double vida, String tipoArma) {  
        super(nome, vida, tipoArma);  
    }  
  
    public void ataqueRapido() {  
        System.out.println("Atacando rapidamente!");  
    }  
}
```

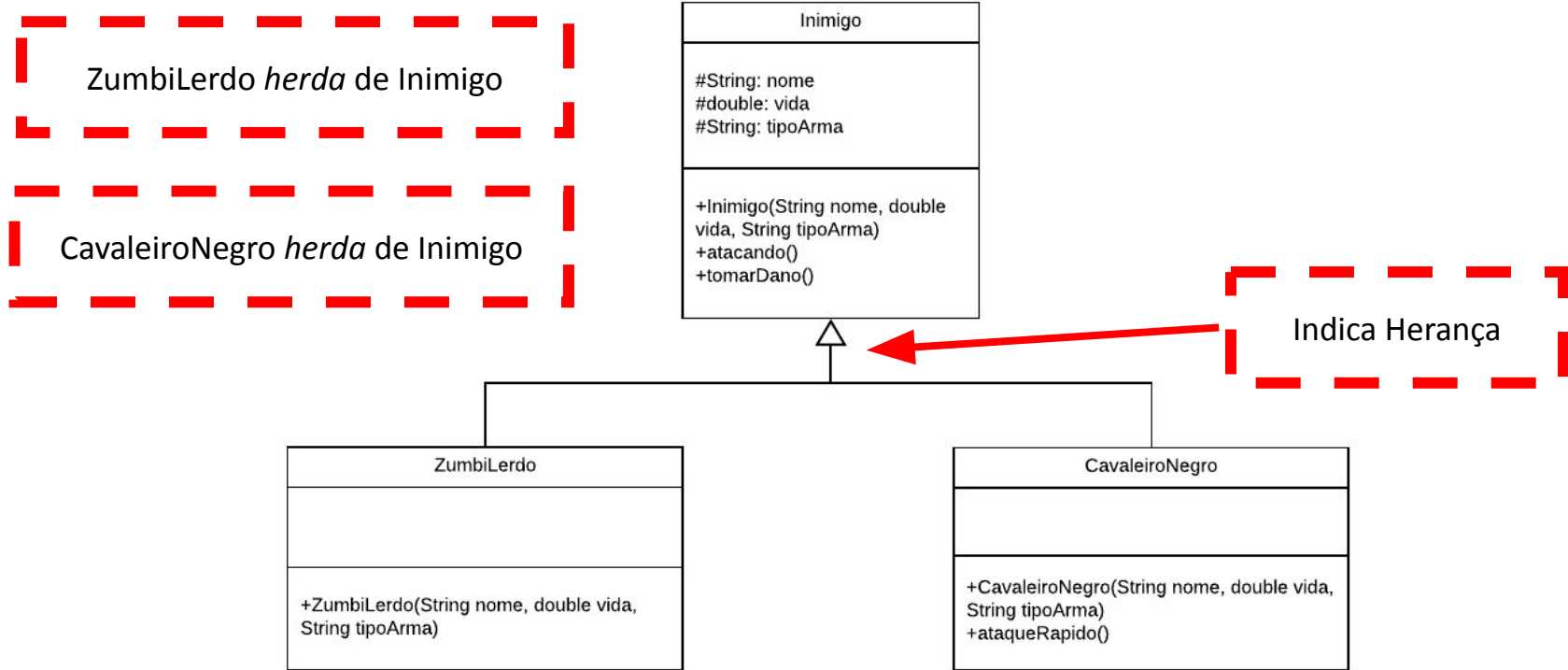
# Modelando o Inimigo com Herança

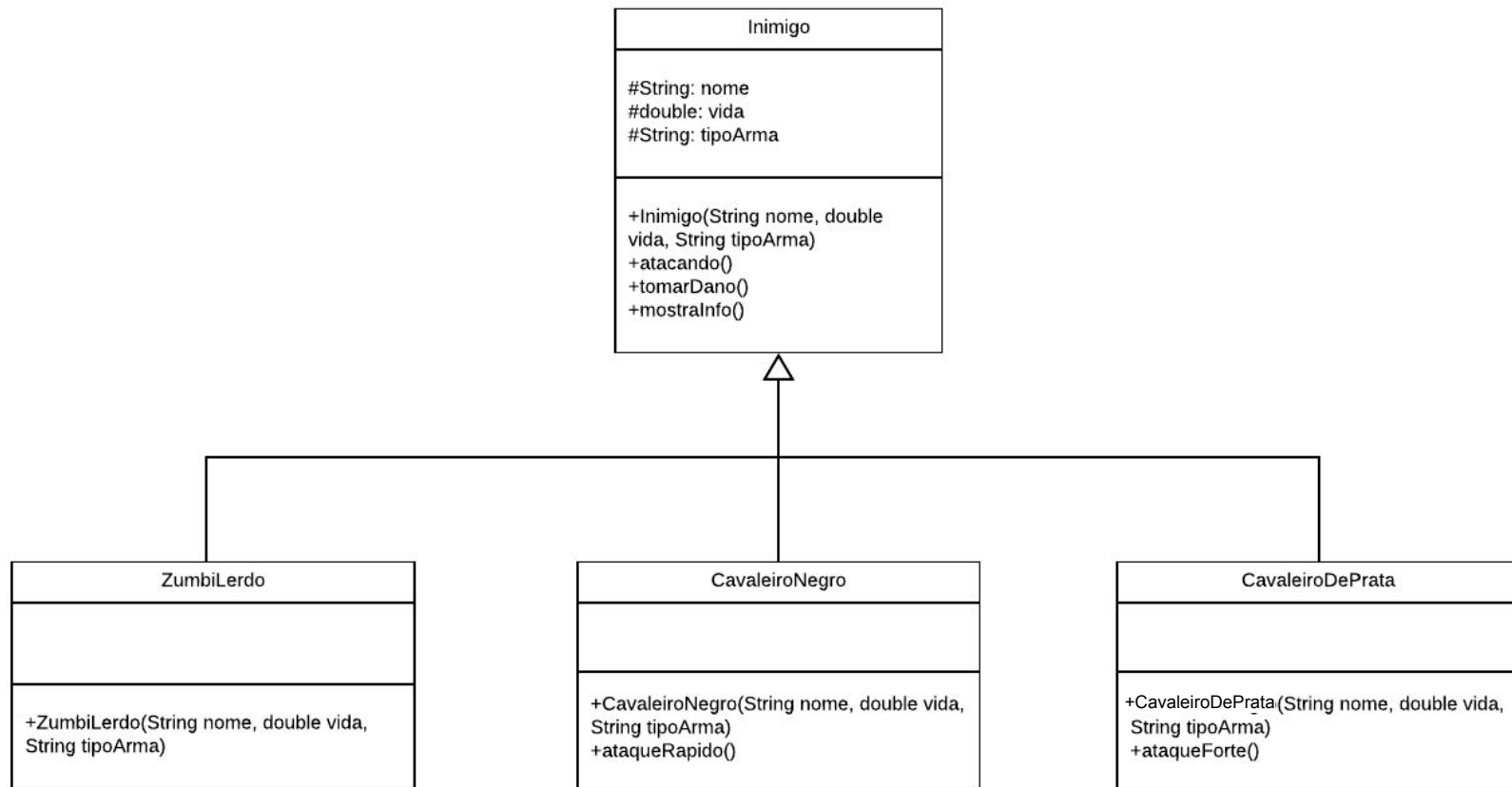
- A palavra chave ***super*** se refere a superclasse, nesse exemplo a *Inimigo*. Ou seja, estamos chamando o construtor da superclasse *Inimigo*
- Na classe *CavaleiroNegro*, colocamos o método ***ataqueRapido()***, pois é uma ***especialização*** dessa classe. Não existe razão para a colocarmos na classe *ZumbiLerdo*

```
public static void main(String[] args) {  
    ZumbiLerdo zumbiLerdo =  
        new ZumbiLerdo("Lerdao", 50, "Espada Curta");  
    CavaleiroNegro cavaleiro =  
        new CavaleiroNegro("Cavaleiro", 100, "Espada Longa");  
  
    zumbiLerdo.atacando();  
  
    cavaleiro.atacando();  
    cavaleiro.ataqueRapido();  
  
}
```

```
Atacando o jogador!  
Atacando o jogador!  
Atacando rapidamente!
```

# UML







# Sobrescrita de Métodos

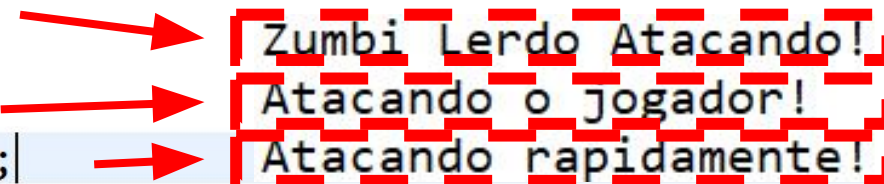
- Observe que as mensagens exibidas no método ***atacando()*** foram as mesmas para ambas as instâncias (*ZumbiLerdo* e *CavaleiroNegro*)
- Temos como especializar esse comportamento através da sobrescrita de métodos

```
public class ZumbiLerdo extends Inimigo {  
  
    //Construtor  
    public ZumbiLerdo(String nome, double vida, String tipoArma) {  
        super(nome, vida, tipoArma);  
    }  
  
    @Override  
    public void atacando() {  
        System.out.println("Zumbi Lerdo Atacando!");  
    }  
}
```

# Sobrescrita de Métodos

- Observe que colocamos uma ***anotação*** `@Override` em cima do método “atacando()” para indicar que estamos sobrescrevendo um método da superclasse. Fizemos isso na classe *ZumbiLerdo*

```
public static void main(String[] args) {  
    ZumbiLerdo zumbiLerdo =  
        new ZumbiLerdo("Lerdao", 50, "Espada Curti");  
    CavaleiroNegro cavaleiro =  
        new CavaleiroNegro("Cavaleiro", 100, "Espada Longa");  
  
    zumbiLerdo.atacando();  
    cavaleiro.atacando();  
    cavaleiro.ataqueRapido();  
}
```



Zumbi Lerdo Atacando!  
Atacando o jogador!  
Atacando rapidamente!

# Sobrescrita de Métodos

- Podemos fazer uma modificação no código, de forma que, cada classe que herde de Inimigo, faça a sobrescrita do método *atacando()*, e personalize a mensagem

# Polimorfismo

- Quando dizemos que *ZumbiLerdo* **herda** da classe *Inimigo*, dizemos um *ZumbiLerdo* **É UM** *Inimigo*
- Com essa definição, se uma classe *Jogador* é capaz de causar dano a um *Inimigo*, ele pode causar dano em qualquer classe que **herda** de *Inimigo*

# Polimorfismo

- Se toda subclasse de *Inimigo* **É UM** *Inimigo*, então podemos salvar uma referência de em uma variável do tipo *Inimigo*

```
public static void main(String[] args) {  
    Inimigo zumbiLerdo =  
        new ZumbiLerdo("Lerdao", 50, "Espada Curti");  
    Inimigo cavaleiro =  
        new CavaleiroNegro("Cavaleiro", 100, "Espada Longa");  
  
    zumbiLerdo.atacando();  
    cavaleiro.atacando();  
}
```



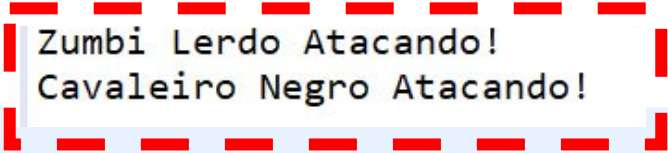
# Polimorfismo

- Perceba que agora as instâncias de *CavaleiroNegro* e *ZumbiLerdo* foram salvas em uma variável do tipo *Inimigo*

# Polimorfismo

- Essas instâncias não mudaram de tipo. O que mudou foi a forma como elas estão sendo referenciadas. Elas continuam sendo instâncias de *CavaleiroNegro* e *ZumbiLerdo*, mas suas referências podem ser armazenadas em qualquer variável que seja de uma superclasse, como a *Inimigo*, por exemplo

```
public static void main(String[] args) {  
    Inimigo zumbiLerdo =  
        new ZumbiLerdo("Lerdao", 50, "Espada Curti");  
    Inimigo cavaleiro =  
        new CavaleiroNegro("Cavaleiro", 100, "Espada Longa");  
  
    zumbiLerdo.atacando();  
    cavaleiro.atacando();  
}
```



Zumbi Lerdo Atacando!  
Cavaleiro Negro Atacando!

# Polimorfismo

- Dado que as instâncias são de fato *ZumbiLerdo* e *CavaleiroNegro*, o método chamado será o presente nessas classes e não ao da classe *Inimigo*

# Polimorfismo

- Se não tivéssemos feito uma sobrescrita, seria chamado o método original definido na classe *Inimigo*
- Porém, perceba que como as instâncias estão salvas em variáveis do tipo *Inimigo*<sup>26</sup>, não podemos invocar métodos específicos das instâncias

# Polimorfismo

- Não conseguimos invocar o método *ataqueRapido()*, definido na classe *CavaleiroNegro*, pois a classe *Inimigo* não “conhece” esse método

# Polimorfismo

```
public static void main(String[] args) {  
    Inimigo zumbiLerdo =  
        new ZumbiLerdo("Lerdao", 50, "Espada Curti");  
    Inimigo cavaleiro =  
        new CavaleiroNegro("Cavaleiro", 100, "Espada Longa");  
  
    zumbiLerdo.atacando();  
    cavaleiro.atacando();  
  
    cavaleiro.ataqueRapdio();//Não compila
```

# Polimorfismo

- Para invocar o método *ataqueRapido()*, primeiro, precisamos verificar se uma instância é de um tipo, e depois trocar o tipo de variável de referência



# Polimorfismo

- Podemos usar o operador *instanceof()*. Com esse operador podemos testar se uma instância é de um determinado tipo. Se for, podemos fazer o ***casting*** e em seguida trabalhar com métodos específicos

```
public static void main(String[] args) {  
    Inimigo zumbiLerdo =  
        new ZumbiLerdo("Lerdao", 50, "Espada Curti");  
    Inimigo cavaleiro =  
        new CavaleiroNegro("Cavaleiro", 100, "Espada Longa");  
  
    zumbiLerdo.atacando();  
    cavaleiro.atacando();  
  
    if(cavaleiro instanceof CavaleiroNegro) {  
        //Salva cavaleiro em uma variável do tipo CavaleiroNegro  
        CavaleiroNegro cav = (CavaleiroNegro)cavaleiro;  
        cav.ataqueRapido(); //Agora compila!  
    }  
}
```

# Polimorfismo

- Podemos criar classes que lidam apenas com as abstrações (ou superclasse) e assim podemos deixar nossas funcionalidades mais genéricas

# Polimorfismo

- Considere uma classe *Jogador*, que possui um método que recebe um parâmetro do tipo *Inimigo* e faz alguma operação nessa variável

```
public class Jogador {  
  
    public void atacar(Inimigo inimigo) {  
        System.out.println("Atacando o Inimigo: " + inimigo.getNome());  
    }  
  
}
```

- Qualquer instância que **herda** de *Inimigo* pode ser passada como parâmetro
- Isso traz poder e flexibilidade para escrever programas
- Fica fácil evoluir esse sistema, criando novos tipos de inimigos para o jogo Dark Souls

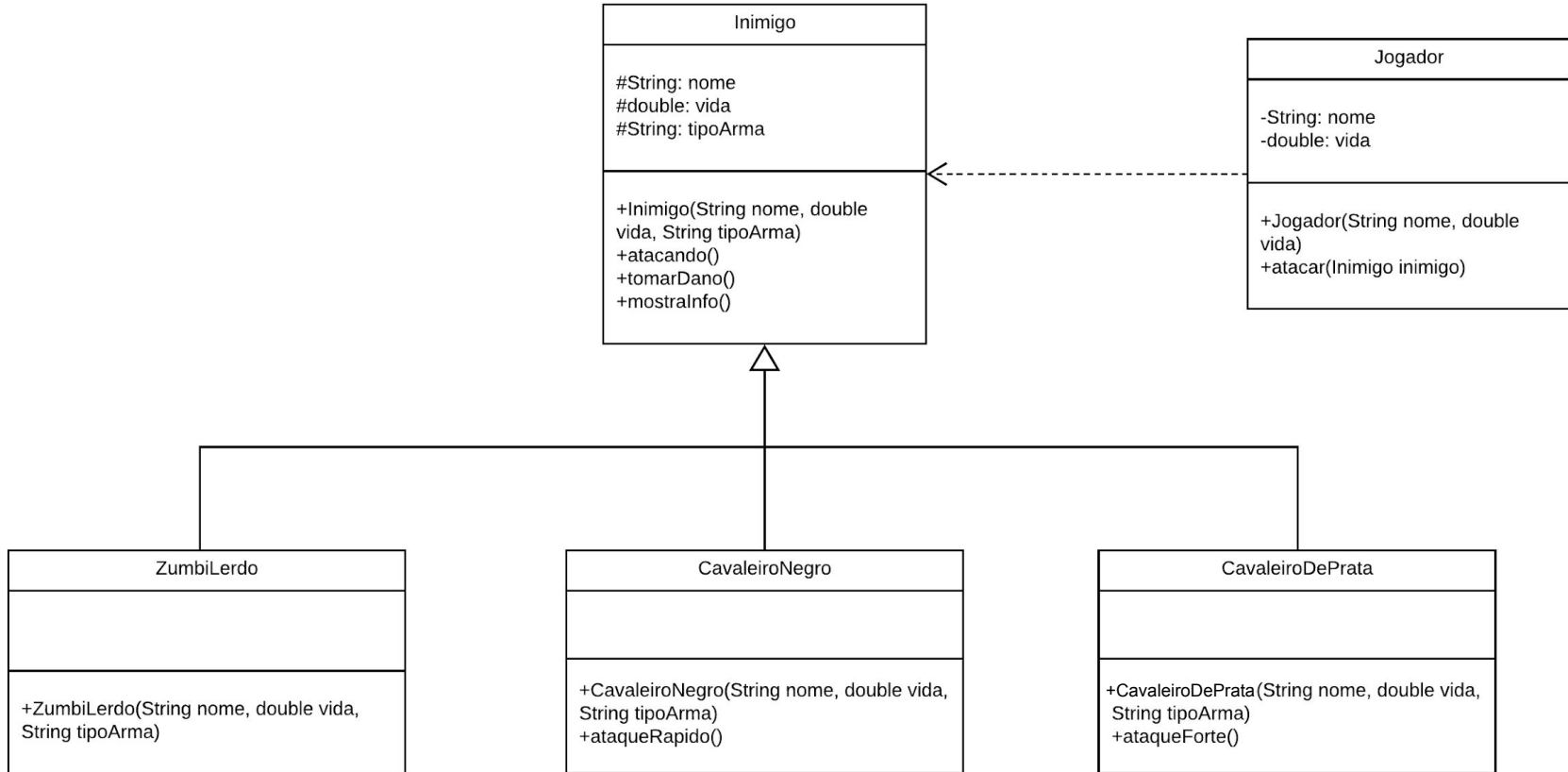
# Polimorfismo

- Por que simplesmente não criar todas as classes como *Inimigo*? Nem precisaríamos de herança e polimorfismo. Dentre as várias razões, uma delas está relacionada às camadas do software

# Polimorfismo

- Exemplo: Podem existir trechos do código que faz sentido trabalhar com instâncias do tipo *ZumbiLerdo*, mas em outras partes do código, pode fazer sentido atuar nessa mesma instância, mas como se fosse um *Inimigo*

# Exercício 1





# Exercício 1

- Crie a classe *Inimigo* e seus tipos: *ZumbiLerdo*, *CavaleiroNegro* e *CavaleiroPrata*.
- Crie uma classe *Jogador*, que recebe um *Inimigo* e causa dano via o método *atacar()*. Faça com o que quando um “jogador” ataque seja invocado o método *tomarDano()* do “*inimigo*”, retornando uma mensagem: "Jogador atacou o inimigo "+ *inimigo.getNome()*
- Faça o método *tomarDano()* diminuir a vida do “inimigo” que foi atacado pelo “jogador”

# Herança e Polimorfismo

\*Adaptado dos materiais de Phyllipe Lima (UNIFEI) por Paulo Meirelles (IME-USP), paulormm@ime.usp.br