Classes Abstratas

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

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]

Dark Souls

Desenvolvedora(s)	FromSoftware
Publicadora(s)	Namco Bandai Games JP FromSoftware
Diretor(es)	Hidetaka Miyazaki
Produtor(es)	Hidetaka Miyazaki, Naotoshi Zin, Yuya

Dark Souls

- Criamos uma superclasse chamada Inimigo e outra três subclasses, herdando dela
  - ZumbiLerdo
  - CavaleiroNegro
  - CavaleiroPrata

- Seria possível criarmos instâncias de Inimigo?

```java
public class Main {

    public static void main(String[] args) {

        Inimigo inimigo = new Inimigo("Inimigo", 30, "Arma Comum");

        inimigo.atacando();

    }
}
```

- Sim, é possível! O código compila e executa sem erros
- Mas, faz sentido termos instâncias de Inimigo?
- Veja que é diferente de termos ***referências*** do tipo Inimigo

```java
public static void main(String[] args) {

    //Instância de Inimigo
    Inimigo inimigo = new Inimigo("Inimigo", 30, "Arma Comum");

    //Instância de ZumbiLerdo sendo referenciado como Inimigo
    Inimigo zumbi = new ZumbiLerdo("Zumbi Lerdo", 50, "Espada Curta");

}
```

- No código acima, temos um exemplo de uma instância de Inimigo e de uma instância de ZumbiLerdo sendo armazenado como uma ***referência*** para Inimigo

- Quando pensamos em **instâncias**, pensamos em **objetos concretos**. Que fazem sentido realizar comportamento
- Quando falamos nos *inimigos*, nós imaginamos um Zumbi Lerdo, um Cavaleiro Negro e assim por diante
- Mas apenas um “Inimigo” parece algo ***abstrato*** para termos uma instância desse tipo
- Mas por que, então, criamos a classe *Inimigo*?

```java
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!");
    }
}
```

```java
public class CavaleiroNegro extends Inimigo {

    //Construtor
    public CavaleiroNegro(String nome, double vida, String tipoArma) {
        super(nome, vida, tipoArma);
    }

    @Override
    public void atacando() {
        System.out.println("Cavaleiro Negro Atacando!");
    }
    public void ataqueRapido() {
        System.out.println("Atacando rapidamente!");
    }
}
```

- Em primeiro lugar, para evitar repetir código
- Criamos classes como ZumbiLerdo e CavaleiroNegro, todas elas ***herdando*** de Inimigo e reusando sua estrutura (atributos e métodos).
- Com isso, economizamos bastante código

- E podemos também criar novos tipos de inimigos, como *CavaleiroPrata*, *EsqueletoFogo* etc, herdando da classe ***Inimigo***. Isso favorece a ***evolução*** do nosso software
- E com herança, podemos utilizar o poder do polimorfismo. Criamos métodos genéricos que sabem apenas lidar com a superclasse e os método nela presente
- Reveja a classe Jogador

```java
public class Jogador{

    private String nome;
    private double vida;


    public Jogador(String nome, double vida) {
        this.nome = nome;
        this.vida = vida;
    }

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

- Observe que o método ***atacar(Inimigo inimigo)*** recebe instâncias ***referenciadas*** ou ***do tipo*** Inimigo. Ele não precisa conhecer nenhuma classe que ***herda*** de Inimigo. Isso também favorece a ***evolução*** do software

Classes Abstratas

- Faz sentido ter instâncias do tipo Inimigo? **Não**
- Mas, faz ***todo*** sentido termos ***referências*** do tipo Inimigo. Afinal, é assim que o método ***atacar(Inimigo inimigo)*** funciona. Ele recebe ***referências*** para Inimigo

Classes Abstratas

- Criamos a classe Inimigo apenas para ser ***referências*** (variáveis) e não ***instâncias*** (objetos na memória)
- Para isso, podemos dizer que ela é uma classe ***abstrata***
- No Java, temos a palavra chave ***abstract*** para esse fim
- Observe a nova classe ***abstrata*** Inimigo

```java
public abstract 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!");
    }
```

Classes Abstratas

- Quando fazemos uma classe ***abstract***, estamos passando a seguinte informação
  - Não desejamos instanciar essa classe
  - Ela deve ser uma superclasse e suas subclasses serão instanciadas
  - Ele deve ser usada como referência para permitir o polimorfismo

Classes Abstratas

- O compilador Java garante que ela não será instanciada. Mas pode ser referenciada normalmente
- Apenas suas subclasses poderão ser instanciadas

Classes Abstratas

```
public static void main(String[] args) {

    //Instância de Inimigo. NÃO COMPILA
    Inimigo inimigo = new Inimigo("Inimigo", 30, "Arma Comum");

    //Instância de ZumbiLerdo sendo referenciado como Inimigo
    //Compila!
    Inimigo zumbi = new ZumbiLerdo("Zumbi Lerdo", 50, "Espada Curta");

}
```

Classes Abstratas

- Seria correto dizer que toda superclasse deve ser abstrata? **Não**
- Depende do escopo do seu projeto e de suas abstrações. Não é obrigatório fazer toda superclasse abstrata

Classes Abstratas

- Poderíamos fazer *ZumbiLerdo* ser uma superclasse também. E criarmos novos tipos de zumbis a partir dela. Então teríamos *ZumbiLerdo* herdando de *Inimigo* e *ZumbiLerdoFogo* (por exemplo) herdando de *ZumbiLerdo*. Mas, faz sentido no jogo ter uma instância de *ZumbiLerdo*
- Assim, não há razão para fazê-la ser abstrata

Classes Abstratas

- Repare que podemos ter várias camadas (gerações) de Herança
- Mas não podemos ter uma mesma classe herdando de ***mais de uma*** classe em uma única declaração.

```
//Compila
//Podemos fazer a seguinte analogia:
//ZumbiLerdoFogo é filho de Zumbilerdo e neto de Inimigo
public class ZumbiLerdoFogo extends ZumbiLerdo {

    public ZumbiLerdoFogo(String nome, double vida, String tipoArma) {
        super(nome, vida, tipoArma);
    }

}
```

Classes Abstratas

- O Código abaixo não compila, pois estamos tentando, na mesma declaração fazer uma classe herdar de outras duas!

```
//Não Compila
public class ZumbiLerdoFogo extends ZumbiLerdo, Inimigo {

    public ZumbiLerdoFogo(String nome, double vida, String tipoArma) {
        super(nome, vida, tipoArma);
    }

}
```

Classes Abstratas

- Podemos pensar que, então, seu uso está restrito a polimorfismo e evitar repetição de código
- Mas, podemos fazer isso com uma classe normal, tomando cuidado de manter nosso código consistente
- Vamos pensar no **método atacando()** na superclasse *Inimigo*

```
public void atacando() {
    System.out.println("Atacando o jogador!");
}
```

Método Abstrato

- Imagine que não houvesse a implementação do método na subclasses
- Nesse caso, cada subclasse utilizaria o comportamento da superclasse, ou seja, imprimir “Atacando o jogador”
- E se quiséssemos forçar que cada subclasse sobrescreva o **método atacando()**, a fim de garantir comportamento específico?

Método Abstrato

- Quando temos uma classe abstrata, podemos ter também um método abstrato. Isto é, não possui implementação na superclasse, e toda subclasse é obrigada a implementar

Método Abstrato

```
//Não compila
//Metodo abstrato não pode ter implementação
//em sua definição
public abstract void atacando() {
    System.out.println("Atacando o jogador!");
}
```

```
//Agora compila :)
public abstract void atacando();
```

Método Abstrato

- Vamos na classe ZumbiLerdo e remover o método atacando()
- Perceba que o código não irá compilar

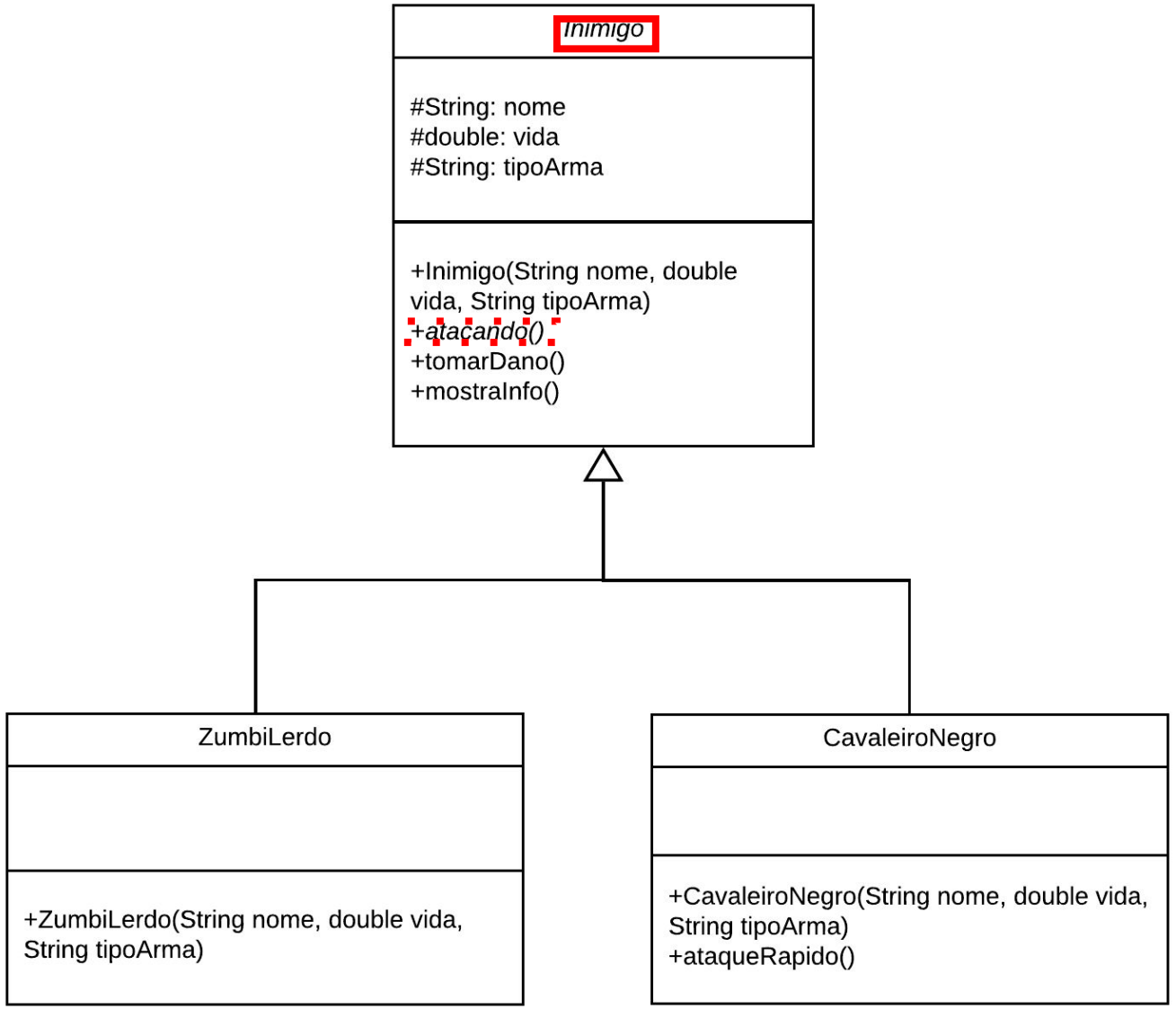
```
//Não Compila
//Precisamos, obrigatoriamente, implementar atacando
//O Eclipse nos ajuda nessa tarefa
public class ZumbiLerdo extends Inimigo {

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


}
```

Método Abstrato

```
public class ZumbiLerdo extends Inimigo {

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

    @Override
    public void atacando() {
        //Agora de a sua implementação!
    }


}
```

UML

- Classes e métodos abstratos aparecem com a fonte itálica



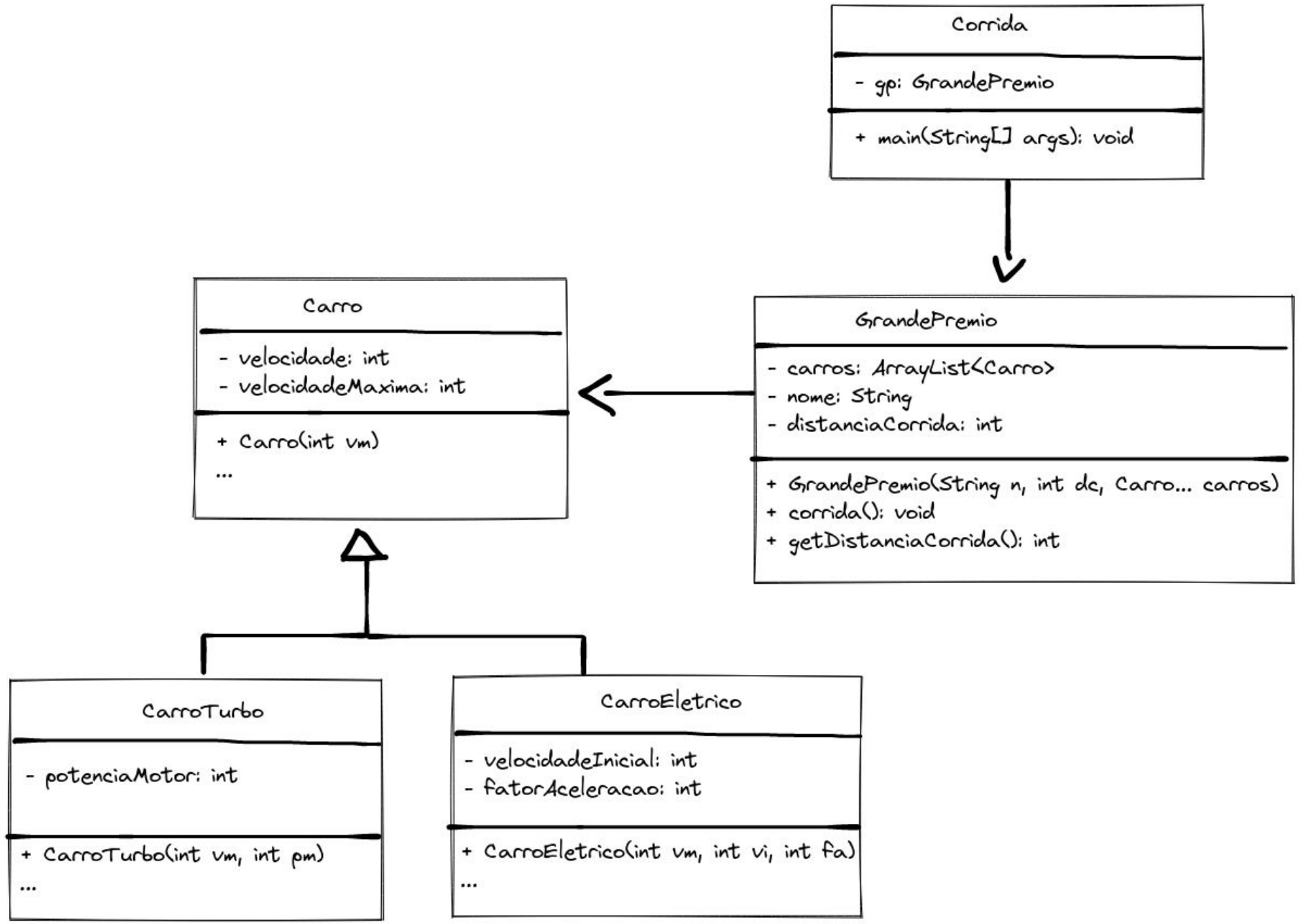
Outro Exemplo:
Corrida de Carros

```java
public class Carro {

    private int potencia;
    private int velocidade;
    private int velocidadeMaxima;

    public Carro(int potencia, int velocidadeMaxima) {
        this.potencia = potencia;
        this.velocidadeMaxima = velocidadeMaxima;
    }

    public int getVelocidade() {
        return velocidade;
    }

    public void acelerar() {

    public void frear() {
        velocidade /= 2;
    }
}
```

```java
import static org.junit.Assert.assertEquals;


public class TestCarro {

    private Carro c;

    @Before
    public void inicializaCarro(){
        c = new Carro(8, 100);
    }

    @Test
    public void criaCarroParado() {
        assertEquals(0,c.getVelocidade());
    }

    @Test
    public void acelerar(){
        c.acelerar();
        assertEquals("Acelerou 1 vez",8,c.getVelocidade());
        c.acelerar();
        c.acelerar();
        assertEquals("Acelerou 3 vez", 24,c.getVelocidade());
    }

    @Test
    public void frear(){
        c.acelerar();
        c.frear();
        assertEquals(4,c.getVelocidade());
    }
```

```java
public class Principal {

    public static void main(String[] args) {
        Carro fusca = new Carro(20, 80);
        Carro uno = new Carro(30,120);

        fusca.acelerar();
        uno.acelerar();
        uno.acelerar();
        fusca.acelerar();
        fusca.acelerar();
        uno.acelerar();
        fusca.acelerar();
        System.out.println("Velocidade do Fusca: "+fusca.getVelocidade());
        System.out.println("Velocidade do Uno: "+uno.getVelocidade());

        fusca.frear();
        uno.frear();
        uno.frear();

        System.out.println("Velocidade do Fusca: "+fusca.getVelocidade());
        System.out.println("Velocidade do Uno: "+uno.getVelocidade());
    }
}
```

```
<terminated> Principal (2) [Java Application] /u
Velocidade do Fusca: 80
Velocidade do Uno: 90
Velocidade do Fusca: 40
Velocidade do Uno: 22
```


Corrida
- gp: GrandePremio
+ main(String[] args): void
Carro
- velocidade: int
- velocidadeMaxima: int
+ Carro(int vm)
...
GrandePremio
- carros: ArrayList<Carro>
- nome: String
- distanciaCorrida: int
+ GrandePremio(String n, int dc, Carro... carros)
+ corrida(): void
+ getDistanciaCorrida(): int
CarroTurbo
- potenciaMotor: int
+ CarroTurbo(int vm, int pm)
...
CarroEletrico
- velocidadeInicial: int
- fatorAceleracao: int
+ CarroEletrico(int vm, int vi, int fa)
...

```java
public abstract class Carro {

    protected int velocidade;
    private int velocidadeMaxima;

    public Carro(int velocidadeMaxima) {
        this.velocidadeMaxima = velocidadeMaxima;
    }

    public int getVelocidade() {
        return velocidade;
    }

    public int getVelocidadeMaxima() {
        return this.velocidadeMaxima;
    }

    public abstract void acelerar();

    public abstract void frear();
}
```

```java
public class CarroTurbo extends Carro {

    private int potencia;

    public CarroTurbo(int velocidadeMaxima, int potencia) {
        super(velocidadeMaxima);
        this.potencia = potencia;

    }

    @Override
    public void acelerar() {
        velocidade += potencia;
        if(velocidade > this.getVelocidadeMaxima())
            velocidade = this.getVelocidadeMaxima();
    }

    @Override
    public void frear() {
        velocidade /= 2;
    }

}
```

```java
public class TestCarroTurbo {

    private Carro c;

    @Before
    public void inicializaCarro(){
        c = new CarroTurbo(100,8);
    }

    @Test
    public void criaCarroParado() {
        assertEquals(0,c.getVelocidade());
    }

    @Test
    public void acelerar(){
        c.acelerar();
        assertEquals("Acelerou 1 vez",8,c.getVelocidade());
        c.acelerar();
        c.acelerar();
        assertEquals("Acelerou 3 vez", 24,c.getVelocidade());
    }

    @Test
    public void frear(){
        c.acelerar();
        c.frear();
        assertEquals(4,c.getVelocidade());
    }
```

```java
public class Principal {

    public static void main(String[] args) {
        Carro fusca = new CarroTurbo(80,20);
        Carro uno = new CarroTurbo(120,30);

        fusca.acelerar();
        uno.acelerar();
        uno.acelerar();
        fusca.acelerar();
        fusca.acelerar();
        uno.acelerar();
        fusca.acelerar();
        System.out.println("Velocidade do Fusca: "+fusca.getVelocidade());
        System.out.println("Velocidade do Uno: "+uno.getVelocidade());

        fusca.frear();
        uno.frear();
        uno.frear();

        System.out.println("Velocidade do Fusca: "+fusca.getVelocidade());
        System.out.println("Velocidade do Uno: "+uno.getVelocidade());
    }
}
```

<terminated> Principal (2) [Java Application] /usr/eclipse/pl

```
Velocidade do Fusca: 80
Velocidade do Uno: 90
Velocidade do Fusca: 40
Velocidade do Uno: 22
```

```java
public class CarroEletrico extends Carro {

    private int velocidadeInicial;
    private int fatorAceleracao;

    public CarroEletrico(int velocidadeMaxima, int velocidadeInicial, int fatorAceleracao) {
        super(velocidadeMaxima);
        this.velocidadeInicial = velocidadeInicial;
        this.fatorAceleracao = fatorAceleracao;
    }

    public int getVelocidadeInicial() {
        return velocidadeInicial;
    }

    public int getFatorAceleracao() {
        return fatorAceleracao;
    }

    @Override
    public void acelerar() {
        velocidade *= fatorAceleracao;

        if (velocidade < velocidadeInicial) {
            velocidade = velocidadeInicial;
        }
        else if(velocidade > this.getVelocidadeMaxima()){
            velocidade = this.getVelocidadeMaxima();
        }
    }

    @Override
    public void frear() {
        velocidade /= fatorAceleracao;
    }
}
```

```java
import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class TestCarroEletrico {

    private Carro c;

    @Before
    public void inicializaCarro(){
        c = new CarroEletrico(150,20,2);
    }

    @Test
    public void criaCarroParado() {
        assertEquals(0,c.getVelocidade());
    }

    @Test
    public void acelerar(){
        c.acelerar();
        assertEquals("Acelerou 1 vez",20,c.getVelocidade());
        c.acelerar();
        c.acelerar();
        assertEquals("Acelerou 3 vez", 80,c.getVelocidade());
    }

    @Test
    public void frear(){
        c.acelerar();
        c.frear();
        assertEquals(10,c.getVelocidade());
    }
```

```java
public class Principal {

    public static void main(String[] args) {
        Carro fusca = new CarroTurbo(80,20);
        Carro ferrari = new CarroEletrico(300,100,2);

        fusca.acelerar();
        ferrari.acelerar();
        ferrari.acelerar();
        fusca.acelerar();
        fusca.acelerar();
        ferrari.acelerar();
        fusca.acelerar();
        System.out.println("Velocidade do Fusca: "+fusca.getVelocidade());
        System.out.println("Velocidade da Ferrari: "+ferrari.getVelocidade());

        fusca.frear();
        ferrari.frear();
        ferrari.frear();

        System.out.println("Velocidade do Fusca: "+fusca.getVelocidade());
        System.out.println("Velocidade da Ferrari: "+ferrari.getVelocidade());
    }
}
```

```
Velocidade do Fusca: 80
Velocidade da Ferrari: 300
Velocidade do Fusca: 40
Velocidade da Ferrari: 75
```

```java
import java.util.ArrayList;

public class GrandePremio {
    private String nome;
    private int distanciaCorrida;
    private ArrayList<Carro> carros;

    public GrandePremio(String nome, int distanciaCorrida, ArrayList<Carro> carros) {
        this.nome = nome;
        this.distanciaCorrida = distanciaCorrida;
        this.carros = carros;
    }

    public String getNome() {
        return nome;
    }

    public int getDistanciaCorrida() {
        return distanciaCorrida;
    }

    public int[] correr() {
        int acao = 0;
        boolean vencedor = false;
        int[] distanciasPercorridas = new int[carros.size()];

        while (!vencedor) {
            for (Carro carro: carros) {
                if(acao%4 == 0) {
                    carro.frear();
                }else {
                    carro.acelerar();
                }
                distanciasPercorridas[carros.indexOf(carro)] += carro.getVelocidade();
            }
            acao++;

            for(int i = 0; i < this.carros.size(); i++) {
                if(distanciasPercorridas[i] >= distanciaCorrida) {
                    vencedor = true;
                }
            }
        }
        return distanciasPercorridas;
    }
}
```

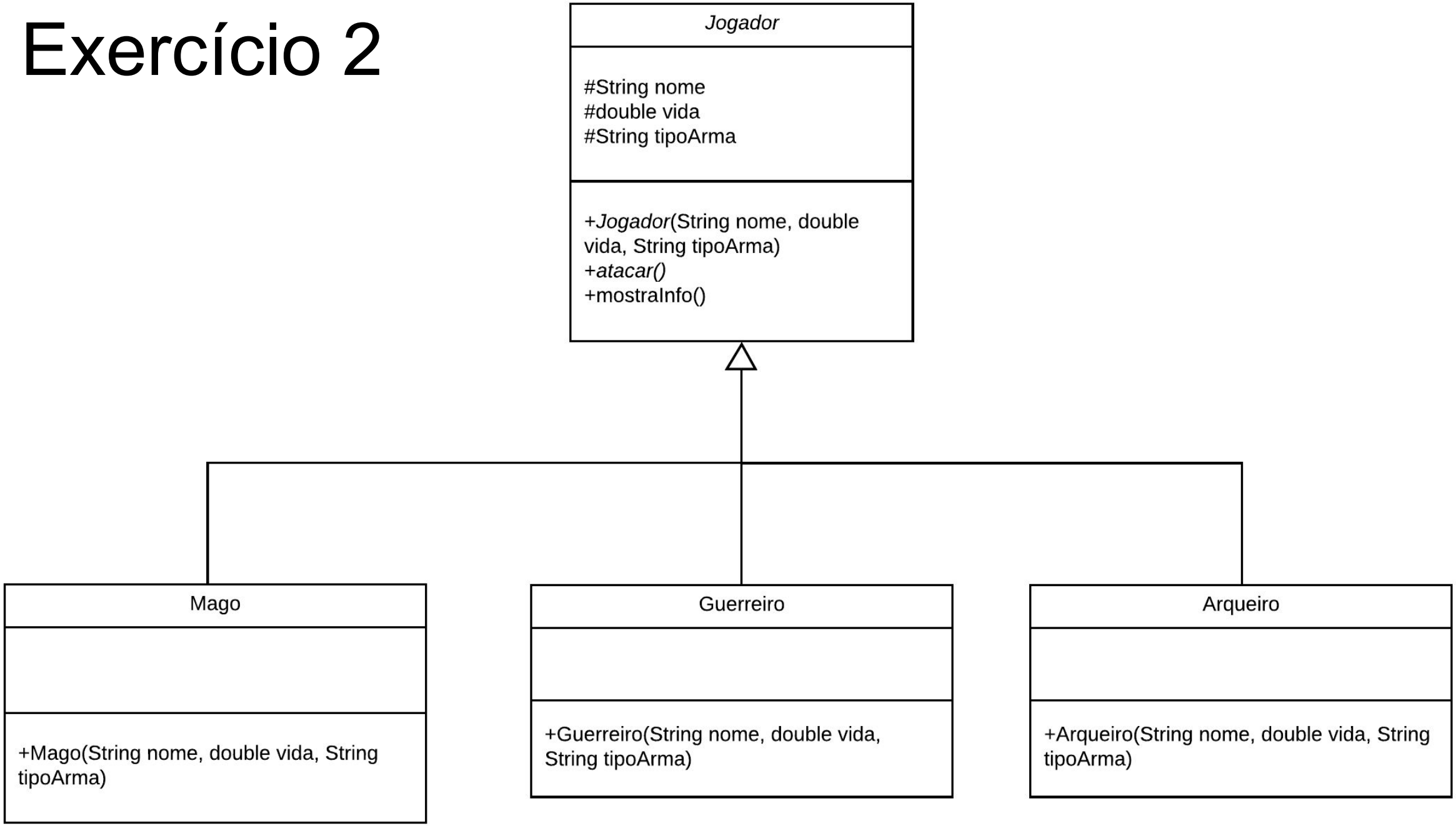
```java
import static org.junit.Assert.*;

import java.util.ArrayList;

import org.junit.Before;
import org.junit.Test;

public class TestGrandePremio {

    private GrandePremio grandePremio;

    @Before
    public void inicializaCorrida(){
        ArrayList<Carro> carros = new ArrayList<Carro>();

        carros.add(new CarroTurbo(180, 20));
        carros.add(new CarroTurbo(220, 30));
        carros.add(new CarroTurbo(250, 40));
        carros.add(new CarroEletrico(180, 20, 2));
        carros.add(new CarroEletrico(220, 30, 3));
        carros.add(new CarroEletrico(250, 40, 3));

        grandePremio = new GrandePremio("Grande Prêmio do Brasil", 1000000, carros);
    }

    @Test
    public void correr() {
        int[] distancias = grandePremio.correr();

        assertEquals(distancias[0], 427697);
        assertEquals(distancias[1], 643933);
        assertEquals(distancias[2], 860181);
        assertEquals(distancias[3], 757050);
        assertEquals(distancias[4], 879984);
        assertEquals(distancias[5], 1000224);

    }
}
```

```java
import java.util.ArrayList;

public class Corrida {

    public static void main(String[] args) {
        ArrayList<Carro> carros = new ArrayList<Carro>();

        carros.add(new CarroTurbo(180, 20));
        carros.add(new CarroTurbo(220, 30));
        carros.add(new CarroTurbo(250, 40));
        carros.add(new CarroEletrico(180, 20, 2));
        carros.add(new CarroEletrico(220, 30, 3));
        carros.add(new CarroEletrico(250, 40, 3));


        GrandePremio grandePremio = new GrandePremio("Grande Prêmio do Brasil", 1000000, carros);

        int[] distancias = grandePremio.correr();

        System.out.println(grandePremio.getNome() + "(Resultado):\n");

        int i;
        for (Carro carro: carros) {
            i = carros.indexOf(carro);
            System.out.print("Carro " + i + ": " + distancias[i] + " metros" + "\n");
        }
    }
}
```

```
<terminated> Principal (2) [Java Application] /usr/eclipse/p
Grande Prêmio do Brasil(Resultado):

Carro 0: 427697 metros
Carro 1: 643933 metros
Carro 2: 860181 metros
Carro 3: 757050 metros
Carro 4: 879984 metros
Carro 5: 1000224 metros
```

Voltando ao Dark Souls …

Exercício 2


Jogador
#String nome
#double vida
#String tipoArma
+Jogador(String nome, double vida, String tipoArma)
+atacar()
+mostraInfo()
Mago
+Mago(String nome, double vida, String tipoArma)
Guerreiro
+Guerreiro(String nome, double vida, String tipoArma)
Arqueiro
+Arqueiro(String nome, double vida, String tipoArma)

Exercício 2

- Evolua o que foi feito no Exercício 1 de forma que Inimigo seja uma classe abstrata, com o método atacar abstrato
- Implemente as classes: **Mago** (que utiliza um *cajado)*, **Guerreiro** (um *machado)* e **Arqueiro** (um *arco)*.
- Cada classe que herda **Jogador**, deve implementar o seu método *atacar()*, que retorna suas respectivas mensagens
- A classe **Jogador** e o método *atacar()* são abstratos
- Crie os testes de unidade para testar todos os métodos das classes

Classes Abstratas

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