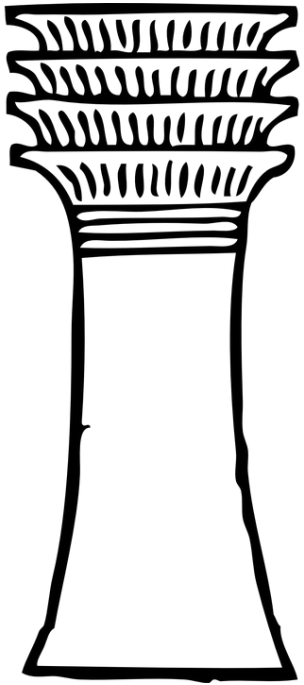
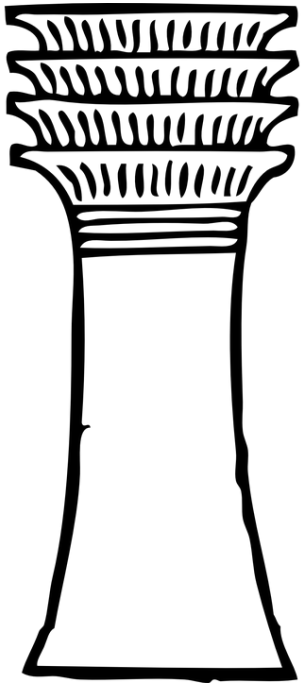
POLIMORFISMO

4 PILARES

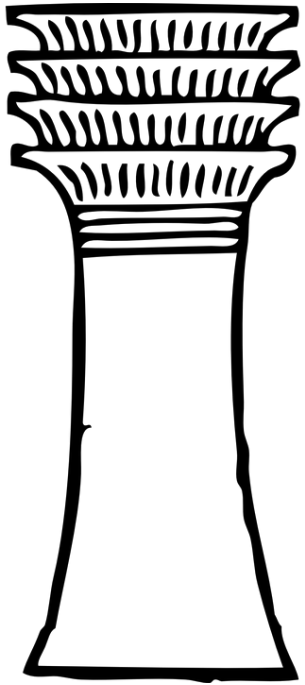
Abstração



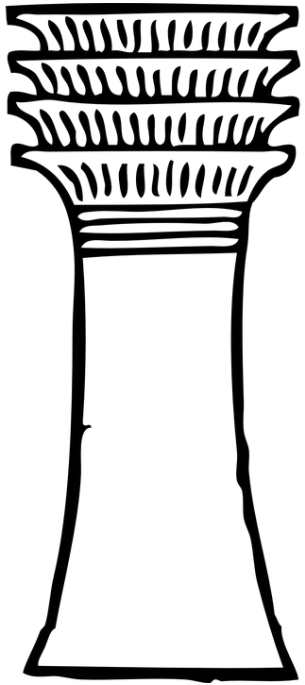
Encapsulamento



Herança



Polimorfismo



PROGRAMAÇÃO ORIENTADA A OBJETOS

UM EXERCÍCIO!

https://goo.gl/HwMBrN

OBJETIVO

Exercitar herança e identificar pontos de melhoria.

MISSÃO

Crap Games quer desenvolver um novo jogo, e solicitou a você que desenvolva a estrutura de classes dos personagens do game chamado "Walking in crap".

Utilize todas as boas práticas de programação conhecidas.

Os personagens do game serão: guerreiro, mago, padre

TAREFA 0 - PERSONAGEM

Características: nome, força, habilidade mental, poder de ataque, esquiva, resistência, vida atual e vida máxima.

Ações:

atacar

Entrada (parâmetro): Um personagem

Retorno: Sem retorno

Comportamento: Desconhecido

contra-atacar

Entrada (parâmetro): Um personagem

Retorno: Sem retorno

Comportamento: Desconhecido

aprimorar habilidade principal

Entrada (parâmetro): Sem entrada

Retorno: Sem retorno

Comportamento: Desconhecido

regenerar vida

Entrada (parâmetro): Sem entrada

Retorno: Sem retorno

Comportamento: Desconhecido

Atenção!

getters e setters podem e devem ser criados conforme necessidade para qualquer classe do projeto.

TAREFA 0 - PERSONAGEM

```ts
export class Personagem {
  constructor(
    protected _nome: string,
    protected _forca: number,
    protected _habilidadeMental: number,
    protected _podeDeAtaque: number,
    protected _esquiva: number,
    protected _resistencia: number,
    protected _vidaAtual: number,
    protected _vidaMaxima: number
  ) {}

  public atacar(personagem: Personagem): void {
    console.log("Um comportamento desconhecido");
  }

  public contraAtacar(personagem: Personagem): void {
    console.log("Comportamento desconhecido");
  }

  public aprimorarHabilidadePrincipal(): void {
    console.log("Comportamento desconhecido");
  }

  public regenerarVida(): void {
    console.log("Comportamento desconhecido");
  }
}

```

TAREFA 1 - GUERREIRO

Características:

nome – O nome de um guerreiro deve possuir o sufixo " Warrior". Ex.: Quando cadastro um guerreiro com o nome "Ragnar", será registrado como "Ragnar Warrior"

força – um valor entre 1 em 1000

habilidade mental – a habilidade mental de um guerreiro é sempre 0

Poder de ataque – o poder de ataque de um guerreiro deve ser calculado multiplicando por 10 o valor da força

Esquiva – habilidade com valor entre 0 e 50, e representa a chance do personagem desviar de um ataque

resistência – o valor da resistência deve ser um valor entre 0 e 90 e indica o percentual do dano sofrido que será absorvido

vida máxima – representa o total de vida do personagem (1 a 40000)

vida atual – quantidade atual de vida do personagem

TAREFA 1 - GUERREIRO

Uma solução

```ts
export class Util {
  public static randomizar(minimo: number, maximo: number) {
    const valorSorteado =
    minimo + Math.random() * (maximo - minimo);
    const valorInteiro = Math.round(valorSorteado);
    return valorInteiro;
  }
}
```

```ts
import { Personagem } from "./Personagem";
import { Util } from "./Util";

export class Guerreiro extends Personagem {
  constructor(nome: string) {
    super(
      nome + " Warrior",
      Util.randomizar(1, 1000),
      0,
      0,
      Util.randomizar(0, 50),
      Util.randomizar(0, 90),
      0,
      Util.randomizar(1, 40_000)
    );
    this._poderDeAtaque = this._forca * 10;
    this._vidaAtual = this._vidaMaxima;
  }
}
```

TAREFA 1 - GUERREIRO

Ações:

Atacar – o movimento de atacar de um personagem deve receber por parâmetro um personagem a ser atacado. O ataque tem chance de falhar de acordo com a "esquiva" do atacado. A quantidade de vida retirada do atacado em caso de acerto, será o valor do poder de ataque reduzido em um percentual igual a resistência do atacado. Toda tentativa de ataque gera um contra-ataque automático do oponente.

Contra-atacar – Com a mesma mecânica do ataque, desenvolva o contra-ataque.

Aprimorar Habilidade Principal – Incrementa em 10% a força do personagem

Regenerar vida – Recupera 5% da vida do personagem

TAREFA 1 - GUERREIRO

```typescript
public atacar(oponente: Personagem): void {
  console.log(`${this._nome} atacou ${oponente.nome}`);
  this.ataque(oponente);
  oponente.contraAtacar(this);
}

public contraAtacar(oponente: Personagem): void {
  console.log(`${this._nome} contra-atacou ${oponente.nome}`);

  this.ataque(oponente);
}

public aprimorarHabilidadePrincipal(): void {
  this._forca *= this._forca * 1.1;
  this.atualizarPoderDeAtaque();
}
private atualizarPoderDeAtaque(): void {
  this._poderDeAtaque = this._forca * 10;
}
```

```typescript
public regenerarVida(): void {
   this._vidaAtual += this._vidaAtual * 1.05;
   if (this._vidaAtual > this._vidaMaxima) {
     this.vidaAtual = this._vidaMaxima;
   }
 }

 private ataque(oponente: Personagem): void {
   const acertou: boolean = Util.randomizar(0, 100) > oponente.esquiva;
   if (acertou) {
     const danoCausado: number =
       (1 - oponente.resistencia / 100) * this._poderDeAtaque;
     oponente.vidaAtual = oponente.vidaAtual - danoCausado;

     const oponenteMorreu: boolean = oponente.vidaAtual <= 0;
     if (oponenteMorreu) {
       throw new Error(`${oponente.nome} foi derrotado.`);
     }
   } else {
     console.log(`${oponente.nome} esquivou o ataque de ${this._nome}`);
   }
 }
```

TAREFA 2 - PRIEST

Características:

nome – O nome de um padre deve possuir o sufixo " Priest". Ex.: Quando cadastrar um padre com o nome "Fabio de Melo", será registrado como "Fabio de Melo Priest"

força – a força de um padre é sempre 0

habilidade mental – a habilidade mental de um padre é sempre 0

Poder de ataque – o poder de ataque de padre é sempre 0

Esquiva – a esquiva de um padre é sempre 0

resistência – a resistência de um padre é sempre 0

vida – representa o total de vida do personagem (1 a 8000)

TAREFA 2 - PRIEST

Uma solução

```ts
import { Personagem } from "./Personagem";
import { Util } from "./Util";

export class Priest extends Personagem {
  constructor(nome: string) {
    super(nome + " Priest", 0, 0, 0, 0, 0, 0, Util.randomizar(1, 8_000));
    this._vidaAtual = this._vidaMaxima;
  }
}
```

TAREFA 2 - PRIEST

Ações:

Atacar – O ataque de um padre tem 40% de chance de converter o oponente a seu favor, com isso a batalha encerra imediatamente e o padre é dado como vitorioso

Contra-atacar – Com a mesma mecânica do ataque, desenvolva o contra-ataque.

Aprimorar Habilidade Principal – O padre não treina habilidade. Na tentativa de treinar a habilidade de um padre um "Error" deve ser lançado, com a mensagem "Este personagem não pode executar esta ação".

Regenerar vida – Recupera 10% da vida do personagem

TAREFA 4 - TESTAGEM

Missão:

Criar um Main.ts que crie N personagens e façam eles batalharem uns com os outros, randomicamente.

TAREFA 4 - TESTAGEM

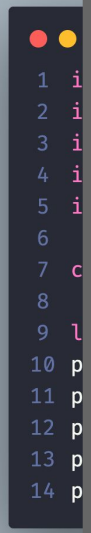
Quem está atacando? Guerreiro? Mago? Priest? Personagem? Não sabemos. Aqui temos um comportamento polimórfico!

```
import prompt from "prompt-sync";
import { Guerreiro } from "./Guerreiro";
import { Priest } from "./Padre";
import { Personagem } from "./Personagem";
import { Util } from "./Util";

const teclado = prompt();

let personagens: Personagem[] = [];
personagens.push(new Priest("Fábio de Melo"));
personagens.push(new Guerreiro("Ragnar"));
personagens.push(new Priest("Quemedo"));
personagens.push(new Guerreiro("Genghis Khan"));
personagens.push(new Guerreiro("Alexandre, o Grande"));
```

O método 'resumo' foi criado em Personagem para apresentar nome e energia do personagem.

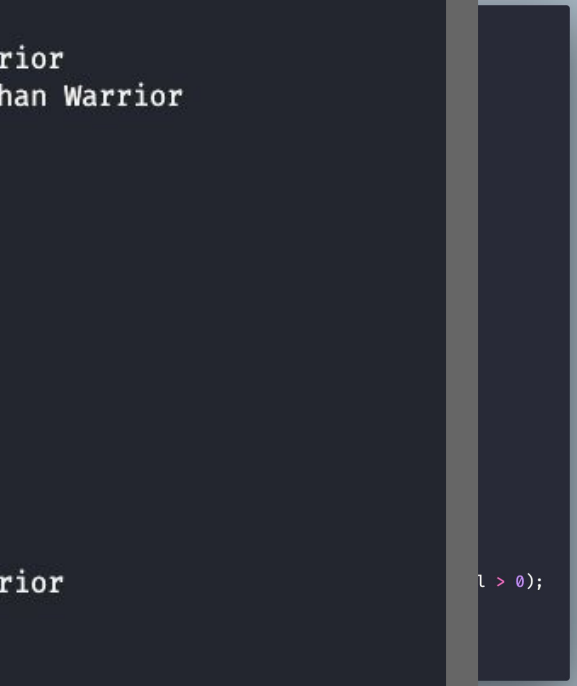
```
while (true) {
  console.log(`═══ Personagens vivos (${personagens.length}) ═══`);
  personagens.forEach((personagem) => console.log(personagem.resumo()));
  if (personagens.length === 1) {
    break;
  }
  console.log("═══════════════════════════\n");

  teclado("Tecle ENTER para rodar o próximo round\n");
  try {
    const atacantePosicao = Util.randomizar(0, personagens.length - 1);
    const atacadoPosicao = Util.randomizar(0, personagens.length - 1);
    if (atacantePosicao !== atacadoPosicao) {
      const atacante = personagens[atacantePosicao];
      const atacado = personagens[atacadoPosicao];
      atacante.atacar(atacado);
      console.log(atacante.resumo());
      console.log(atacado.resumo());
      console.log("\n");
      console.log(".".repeat(20));
    }
  } catch (e) {
    personagens = personagens.filter((personagem) => personagem.vidaAtual > 0);
    console.log((e as any).message);
  }
}
```

TA



PROBLEMAS SAÍDA TERMINAL COMENTÁRIOS CONSOLE DE DEPURAÇÃO

```
Alexandre, o Grande Warrior: 266.8/13987
============================

Tecle ENTER para rodar o próximo round
Alexandre, o Grande Warrior atacou Genghis Khan Warrior
Genghis Khan Warrior contra-atacou Alexandre, o Grande Warrior
Alexandre, o Grande Warrior esquivou o ataque de Genghis Khan Warrior
Alexandre, o Grande Warrior: 266.8/13987
Genghis Khan Warrior: 12680.1/31446


.................
===== Personagens vivos (2) =====
Genghis Khan Warrior: 12680.1/31446
Alexandre, o Grande Warrior: 266.8/13987
============================

Tecle ENTER para rodar o próximo round
Alexandre, o Grande Warrior atacou Genghis Khan Warrior
Genghis Khan Warrior contra-atacou Alexandre, o Grande Warrior
Alexandre, o Grande Warrior foi derrotado.
===== Personagens vivos (1) =====
Genghis Khan Warrior: 10595.0/31446
```



O mé
para

TAREFA 3 - MAGO

Tema de casa. :)

POLIMORFISMO

CLASSES ABSTRATAS

POLIMORFISMO

A definição vem de poli (muitas) morfismo (formas), e trata-se de termos comportamentos diferentes para um único método

Vamos analisar o projeto do game para identificar comportamentos polimórficos

Ex.:

```
const x: Personagem = new Guerreiro('Gladimir');
const y: Personagem = new Priest('Edécio');
x.atacar(y);
y.atacar(x);
```

INSTANCEOF

A palavra-chave instanceof pode ser utilizada para identificar se uma classe é ou não de uma especialização.

```
const x: Personagem = new Guerreiro('Gladimir');
if(x instanceof Guerreiro){
    // Executar ações específicas de guerreiro
    x.umMetodoDoGuerreiro()
}
```

E AS CLASSES ABSTRATAS?

CLASSES ABSTRATAS

Muitas vezes é útil se ter classes em que não se pode instanciar

```
const x: Personagem = new Personagem();
```

Instanciar um Personagem não faz sentido, já que precisamos das especializações para executar as tarefas

Uma classe abstrata é uma classe incompleta, que não pode ser instanciada e pode possuir métodos abstratos

CLASSES ABSTRATAS

Para declararmos uma classe abstrata basta adicionar um abstract antes da palavra-chave class na declaração da mesma.

```
public abstract class Personagem{....}
```

Outro problema, ou no mínimo uma característica estranha da nossa aplicação, é que tivemos que desenvolver métodos na superclasse que estão ali apenas para que possamos sobrescrevê-los nas subclasses.

CLASSES ABSTRATAS

Se esses métodos não existissem, não poderíamos invocar estes métodos através da superclasse Personagem.

```
const p: Personagem = new Guerreiro('Dedé');
p.atacar(oponente);
```

Como o objeto foi tipodo com Personagem, apenas verá o que está em Personagem.

```
1  export class Personagem {
2    constructor(
3      protected _nome: string,
4      protected _forca: number,
5      protected _habilidadeMental: number,
6      protected _podeDeAtaque: number,
7      protected _esquiva: number,
8      protected _resistencia: number,
9      protected _vidaAtual: number,
10     protected _vidaMaxima: number
11   ) {}
12
13   public atacar(personagem: Personagem): void {
14     console.log("Um comportamento desconhecido");
15   }
16
17   public contraAtacar(personagem: Personagem): void {
18     console.log("Comportamento desconhecido");
19   }
20
21   public aprimorarHabilidadePrincipal(): void {
22     console.log("Comportamento desconhecido");
23   }
24
25   public regenerarVida(): void {
26     console.log("Comportamento desconhecido");
27   }
28 }
29
```

MÉTODOS ABSTRATOS E INTERFACES

MÉTODOS ABSTRATOS

Outra característica das classes abstratas é que elas podem possuir métodos abstratos, que são métodos que possuem apenas sua assinatura declarada, assim não precisamos fazer gambiarras para sobrescrevê-los.

Poderíamos aplicar essa funcionalidade nos nossos métodos de atacar, contra-atacar, aprimorar habilidade e regenerar vida!

Um método abstrato obrigatoriamente deve ser implementado para subclasse concreta da superclasse abstrata

Métodos abstratos - exemplo

```
public abstract atacar(Personagem personagem):
void;
```

Em vez de:

```
public atacar(Personagem personagem): void{
    console.log("Um comportamento desconhecido");
}
```

```ts
export class Personagem {
  constructor(
    protected _nome: string,
    protected _forca: number,
    protected _habilidadeMental: number,
    protected _podeDeAtaque: number,
    protected _esquiva: number,
    protected _resistencia: number,
    protected _vidaAtual: number,
    protected _vidaMaxima: number
  ) {}

  public atacar(personagem: Personagem): void {
    console.log("Um comportamento desconhecido");
  }

  public contraAtacar(personagem: Personagem): void {
    console.log("Comportamento desconhecido");
  }

  public aprimorarHabilidadePrincipal(): void {
    console.log("Comportamento desconhecido");
  }

  public regenerarVida(): void {
    console.log("Comportamento desconhecido");
  }
}
```

```ts
export abstract class Personagem {
  constructor(
    protected _nome: string,
    protected _forca: number,
    protected _habilidadeMental: number,
    protected _poderDeAtaque: number,
    protected _esquiva: number,
    protected _resistencia: number,
    protected _vidaAtual: number,
    protected _vidaMaxima: number
  ) {}

  public abstract atacar(personagem: Personagem): void;

  public abstract contraAtacar(personagem: Personagem): void;

  public abstract aprimorarHabilidadePrincipal(): void;

  public abstract regenerarVida(): void;
```

AO TENTAR INSTANCIAR

```
  }
}

const person: Personagem = new Personagem();
```

Não é possível criar uma instância de uma classe abstrata. ts(2511)

Exibir o Problema (⌥F8) Nenhuma correção rápida disponível

POLIMORFISMO

INTERFACES

INTERFACE

Utilizando interfaces é o mecanismo utilizado pelo typescript para proporcionar um comportamento similar ao da herança múltipla

Assim como na herança, podemos declarar uma variável do tipo da interface, e atribuir a ela uma classe concreta.

Interface - Características

Os métodos devem ser públicos e abstratos

Atributos apenas se forem constantes

Um classe pode implementar várias interfaces

Quando se herda uma classe se usa `extends`, quando se implementa uma interface, se usa `implements`

E SE UMA CLASSE QUE IMPLEMENTA UMA INTERFACE NÃO IMPLEMENTAR TODOS OS MÉTODOS ABSTRATOS DESTA?

Interface - Sintáxe

```
public interface Calculadora{
    int somar(v1: number, int: number);
    int subtratir(v1: number, v2: number);
}
```

Note!

Automaticamente esses métodos já são `public abstract`

O `class` é substituído por `interface`

Basicamente, uma classe abstrata que não pode possuir métodos concretos.

INTERFACE - APLICAÇÕES

Especificar o que fazer mas sem dizer COMO fazer.

Ex.:

Ações dos nossos personagens

Operações de Banco de dados

Objetos em Games

Tipos de calculadora

MATERIAL DE APOIO