Revisão da Prova I – Técnicas de programação I - 2025

1) Node.js e NPM:

- O Node.js permite executar código JavaScript fora do navegador, abrindo portas para a criação de aplicações do lado do servidor, como APIs e aplicações web.
- O NPM (Node Package Manager) é crucial para gerenciar as dependências (bibliotecas) do projeto, facilitando a instalação, atualização e compartilhamento de código.

2) TypeScript como Superset do JavaScript:

- TypeScript é uma linguagem que adiciona tipagem estática ao JavaScript, proporcionando maior robustez e detecção de erros em tempo de desenvolvimento.
- A tipagem estática ajuda a evitar erros comuns, como atribuição de valores a variáveis de tipos incompatíveis.
- TypeScript oferece recursos avançados de POO, como classes, interfaces, herança, encapsulamento, polimorfismo, entre outros.

3) Estrutura de Projetos, Compilação e Execução:

- Um projeto Node.js com TypeScript geralmente segue uma estrutura de pastas bem definida, com diretórios como src (código fonte), public (arquivos estáticos), node_modules (dependências).
- O arquivo package.json é fundamental para gerenciar as configurações, dependências e scripts do projeto.
- O compilador TypeScript (tsc) converte o código TypeScript em JavaScript, permitindo que seja executado pelo Node.js.
- Ferramentas como ts-node facilitam a execução direta do código TypeScript sem a necessidade de compilação prévia.

4) Classes, Objetos e Herança:

- Classes servem como modelos para a criação de objetos, definindo suas propriedades (atributos) e métodos (comportamentos).

Exemplo de uma classe:

```
class Pessoa { // nome da classe, mesmo nome arquivo
     nome:string; // atributo
     idade:number; // atributo
     constructor(nome:string, idade:number){ // método construtor e
                                             // seus parâmetros
         this.nome = nome; // atributo recebe parâmetro
         this.idade = idade; // atributo recebe parâmetro
     }
     imprimir(){ // método imprimir
      console.log(`${this.nome} possui ${this.idade} anos`);
     }
}
const pessoa = new Pessoa("Ana", 21); // instanciando um objeto
```

- O construtor é um método especial usado para inicializar os objetos quando eles são criados.
    - A herança permite criar classes (subclasses) que herdam características e comportamentos de classes existentes (superclasses), promovendo a reutilização de código e a organização hierárquica.

Exemplo Herança:

```
class Pessoa {
     nome:string = "";
     idade:number = 0;
     constructor(nome:string, idade:number){
         this.nome = nome;
         this.idade = idade;
     }
}
class Cliente extends Pessoa { // classe Cliente herda classe Pessoa
     saldo: number;
     constructor(nome:string, idade:number, saldo:number){
     // super é o construtor da classe herdada/estendida
     // por este, motivo temos de passar os parâmetros
     // do construtor da classe base
     super(nome,idade);
     this.saldo = saldo;
     }
     print():void {
         console.log(`${this.nome} - ${this.idade} - ${this.saldo}`);
     }
}
const c = new Cliente("Ana",18,980); // instanciando objeto Cliente
c.print();
```

5) Conceitos Importantes da POO:

- **Polimorfismo:** Capacidade de objetos de diferentes classes responderem ao mesmo método de maneiras diferentes, proporcionando flexibilidade e extensibilidade.
- **Sobrescrita:** Redefinição de um método herdado de uma superclasse em uma subclasse, adaptando o comportamento para a classe específica.

Exemplo Sobrescrita:

```
class A {
     nome: string;
     constructor(nome:string){
     this.nome = nome.toUpperCase();
     }
     print():void {
         console.log("Classe A:", this.nome);
     }
}
class B extends A {
     // sobrescreve a propriedade nome da classe A
     nome: string;
     constructor(nome:string){
         super(nome);
         this.nome = nome.toLowerCase();
     }
     // sobrescreve o método print da classe A
     print():void {
         console.log("Classe B:", this.nome);
     }
     imprimir():void{
```

```
            // chama o método print da superclasse
            super.print();
        }
    }
    const a = new A("Tipo A");
    a.print();
    const b = new B("Tipo B");
    b.print();
    b.imprimir();
```

- **Sobrecarga:** Criação de múltiplos métodos com o mesmo nome, mas com diferentes parâmetros, proporcionando diferentes formas de usar um método.

  Exemplo de sobrecarga:

```
class Teste {
  somar(a: number, b: number): number;
  somar(a: string, b: string, c: string): string;
  somar(a: string, b: string): string;
  somar(a: any, b: any, c?: any): any {
       if (c !== undefined) {
           return a + b + c;
       } else {
           return a + b;
       }
  }
}
const t = new Teste();
// usa a assinatura somar(a: number, b: number): number
console.log(t.somar(2, 3));
// usa a assinatura somar(a: string, b: string, c: string): string;
console.log(t.somar("x", "y", "z"));
// usa a assinatura somar(a: string, b: string): string;
console.log(t.somar("x", "y"));
```

6) Criando a pasta do projeto:

- Escolha uma pasta no seu computador para armazenar os códigos.
- Crie uma subpasta chamada “src”, que irá conter o código fonte.

7) Comandos para iniciar o TypeScript no projeto:

a. npm init -y (criar package.json)
b. npm i -D ts-node typescript (instalar pacotes ts-node e typescript)
c. tsc --init (criar tsconfig.json)

8) Codificando:

a. Crie o arquivo index.ts na pasta src;

b. Altere o arquivo package.json conforme abaixo:

```
{
"name": "revisao",
"version": "1.0.0",
"description": "",
"main": "index.js",
"scripts": {
"index": "ts-node ./src/index"
},
```

Revisão da Prova I – Técnicas de programação I - 2025

Exercícios

1 – Marque a alternativa que descreve o motivo do código TS a seguir apresentar erro na atribuição da 2ª linha.

```
let entrada:string = "12";
entrada = parseInt(entrada);
```

a) Por atribuição explícita a variável `entrada` é declarada como `string`, impedindo de receber um valor `number`.
b) Por tipagem estática a variável `entrada` é declarada como `string`, impedindo de receber um valor `number`.
c) Por anotação explícita de tipo (type annotation), a variável `entrada` é declarada como `string`, impedindo de receber um valor `number`.
d) Por inferência de tipo (type inference), o TS declara a variável `entrada` com o tipo `string`, impedindo de receber um valor `number`.
e) Por tipagem dinâmica a variável `entrada` é declarada como `string`, impedindo de receber um valor `number`.

2 - Analise o trecho de código e marque a alternativa que possui uma instrução que apresenta o erro no código.

```
const nros = [11,22,33];
nros[2] = 0.5;
nros[3] = 1.5;
console.log(nros[2]);
console.log(nros[3]);
nros = [0.5,0.8,0.1];
```

a) nros[2] = 0.5;
b) nros[3] = 1.5;
c) console.log(nros[2]);
d) console.log(nros[3]);
e) nros = [0.5,0.8,0.1];

3 – Os tipos genéricos são uma alternativa para definir a tipagem de variáveis. Marque a alternativa que possui uma atribuição correta na variável `inputs`.

```
let inputs:Array<string> = [];
```

a) inputs[0] = 1;
b) inputs = ["a","b"];
c) inputs = [1,2,3];
d) inputs = ["a",2];
e) inputs = [true];

4 – A união de tipos é uma forma de combinar dois ou mais tipos em um único tipo, permitindo que uma variável possa aceitar diferentes tipos de valores. Marque a alternativa que possui uma atribuição **incorreta** na variável `inputs`.

```
let inputs:Array<string|number> = [];
```

a) inputs[0] = 1;
b) inputs = ["a","b"];
c) inputs = [1,2,3];
d) inputs = ["a",2];
e) inputs = [true];

5 – A anotação de tipos pode usar a união de tipos na definição de arrays usando a notação de colchetes e a notação de tipos genéricos. Marque a alternativa que possui uma atribuição **incorreta** na variável.

a) let a: string[] | number[] = ["a","b"];

b) let b: string[] | number[] = ["a",2];
c) let c: string[] | number[] = [1,2];
d) let d: (string|number)[] = ["a",2];
e) let e:Array<number|string> = ["a",2];

6 - No TS, é possível definir parâmetros condicionais em um método utilizando tipos condicionais. O tipo condicional é definido colocando o sinal ? após a declaração do atributo. Desta forma, o atributo terá valor "undefined" se ele não receber algum valor como parâmetro. Marque a alternativa que possui uma chamada **incorreta** do método construtor de uma classe qualquer:

```
class X {
    a: number;
    b?: number;
    c?: number;
    d?: number;

    constructor(a: number, b?: number,
        c?: number, d?: number) {
        this.a = a;
        if (b != undefined) {
            this.b = b;
        }
        if (c != undefined) {
            this.c = c;
        }
        if (d != undefined) {
            this.d = d;
        }
    }
}
```

a) const x1 = new X(1,2,3,4);
b) const x2 = new X(1,2,3);
c) const x3 = new X(1,2);
d) const x4 = new X(1);
e) const x5 = X();

7 – A POO (Programação Orientada a Objetos) permite a definição de tipos de dados através de classes. A classe é uma estrutura que define um tipo de dado composto por operações (métodos) e propriedades (variáveis definidas no escopo da classe). Marque a alternativa que possui a instrução que constrói um objeto e chama o método print desse objeto.

```
class A {
    x:number;
    y:number;

    constructor(x:number,y:number){
        this.x = x;
        this.y = y;
    }

    print(){
        console.log(this.x, this.y);
    }
}
```

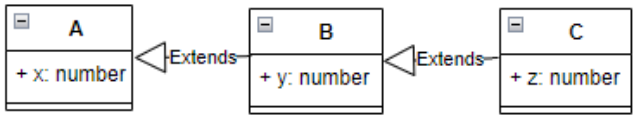
a) const a = A(1,2); c.print();
b) const b = A.print(3,4);
c) const c = new A.print(5,6);
d) const d = new A(7,8); d.print();
e) const e = A(7,8).print();

Revisão da Prova I – Técnicas de programação I - 2025

8 – Qual é o papel da palavra-reservada ***this*** no código do exercício anterior?

a) A palavra-reservada this é usada para fazer referência à instância atual da classe.
b) A palavra-reservada this é usada para fazer referência à classe atual.
c) A palavra-reservada this é usada para fazer referência ao construtor da classe.
d) A palavra-reservada this é usada para fazer referência à operação dentro da classe.
e) A palavra-reservada this é usada para fazer referência às propriedades da classe.

9 – A herança é um recurso da POO que permite que uma classe herde propriedades e métodos de outra classe. Isso permite criar uma hierarquia de classes, onde as subclasses herdam o comportamento e as características da superclasse. Marque a alternativa que possui um código **incorreto** considerando o diagrama UML de classes a seguir.



a)
```
class C extends B {
    z:number;
    constructor(x:number, y:number, z:number){
        super(x,y);
        this.z = z;
    }
}
```
b)
```
class A {
    x:number;
    constructor(x:number){
        this.x = x;
    }
}
```
c)
```
class B extends A{
    y:number;
    constructor(x:number, y:number){
        this.y = y;
    }
}
```
d) const b = new B(1,2);
e) const c = new C(1,2,3);

10 – Qual é o papel da palavra-reservada super na herança?

a) Ela é usada criar a ligação entre a superclasse (classe pai) e a subclasse (classe filha).
b) Ela é usada para referenciar a instância da superclasse (classe pai).
c) Ela é usada para construir uma instância da superclasse (classe pai).
d) Ela é usada para criar a operação de herança entre as classes.
e) Ela é usada para chamar métodos e construtores da classe pai (superclasse) a partir da classe filha (subclasse).

Revisão da Prova I – Técnicas de programação I - 2025

11 – Analise o trecho de código a seguir e marque a alternativa correta.

```
class X {
    nome:string = "Ana";

    print(){
        console.log(this.nome);
    }
}

class Y extends X {
    nome:string = "Maria";

    print(){
        console.log(this.nome);
    }
}
```

a) O método print foi sobrescrito na classe Y.
b) A propriedade nome foi sobrescrita na classe X.
c) A propriedade nome foi sobrecarregada na classe Y.
d) O método print foi sobrecarregado na classe Y.
e) A classe X foi sobrescrita na classe Y.

12 – Analise o trecho de código a seguir e marque a alternativa que imprime no console o texto Boa tarde.

```
class X {
    msg(){
        console.log("Bom dia");
    }
}

class Y extends X {}

class Z extends X {
    msg(){
        console.log("Boa tarde");
    }
}
```

a) const a:X = new X(); a.msg();
b) const b:Y = new Y(); b.msg();
c) const c:X = new Y(); c.msg();
d) const d = new Y(); d.msg();
e) const e:X = new Z(); e.msg();

Códigos dos Exercícios em: https://github.com/hdblouro/RevisaoTPIP1