

# DESENVOLVIMENTO DE SISTEMAS

## TEORIA + QUESTÕES DEBULHADAS

Autor: Leonardo Costa Passos

Resumo teórico com introdução ao desenvolvimento de sistemas e análise de dezenas de questões de concursos da banca CESGRANRIO, analisadas no detalhe de cada item e separadas por categorias que sempre se repetem.

*Não existe nenhum material desse tipo no mercado de concursos, finalmente é possível entender porque cada alternativa de código está certa ou errada e, assim, avançar rapidamente na matéria.*

*Se o conteúdo for útil na sua jornada de estudos, você pode me agradecer fazendo um PIX de um valor que considere justo para a seguinte chave:*

[leonardx@gmail.com](mailto:leonardx@gmail.com)



## Table of Contents

1.	Orientação à Objeto.....	4
1.1.	VANTAGENS:.....	4
1.2.	Classes e Objetos:.....	4
1.3.	Atributos:.....	5
1.4.	Métodos:.....	5
1.5.	Abstração:.....	5
1.6.	Interface.....	6
1.7.	Encapsulamento:.....	7
1.8.	Polimorfismo:.....	8
1.9.	Herança (Generalização/Especialização):.....	8
1.10.	Questões.....	10
2.	Conceitos e Propriedades do Java .....	20
3.	Código Java .....	36
4.	PHP.....	53
4.1.	JAVA x PHP .....	53
4.1.1.	Atribuição de Array:.....	53
	PHP:.....	53
	Java: .....	53
4.1.2.	Funções FOR, WHILE e IF:.....	53
	PHP:.....	53
	Java: .....	54
4.1.3.	Variáveis .....	54
	PHP:.....	54
	Java: .....	54
4.2.	Questões.....	55
5.	Árvores, algoritmos de ordenação e busca .....	62

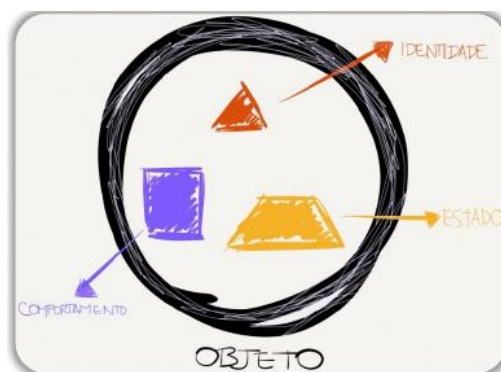
# 1. Orientação à Objeto

## 1.1. VANTAGENS:

- Produção de software natural. Os programas naturais são mais inteligíveis. Em vez de programar em termos de regiões de memória, o profissional pode programar usando a terminologia de seu problema em particular.
- Programas orientados a objetos, bem projetados e cuidadosamente escritos são confiáveis.
- Pode-se reutilizar prontamente classes orientadas a objetos bem-feitas. Assim como os módulos, os objetos podem ser reutilizados em muitos programas diferentes.
- Um código orientado a objetos bem projetado é manutenível. Para corrigir um erro, o programador simplesmente corrige o problema em um lugar. Como uma mudança na implementação é transparente, todos os outros objetos se beneficiarão automaticamente do aprimoramento.
- O software não é estático. Ele deve crescer e mudar com o passar do tempo, para permanecer útil. A programação orientada a objetos apresenta ao programador vários recursos para estender código. Esses recursos incluem herança, polimorfismo, sobreposição e uma variedade de padrões de projeto.
- O ciclo de vida do projeto de software moderno é frequentemente medido em semanas. A programação orientada a objetos ajuda nesses rápidos ciclos de desenvolvimento. Ela diminui o tempo do ciclo de desenvolvimento, fornecendo software confiável, reutilizável e facilmente extensível.

## 1.2. Classes e Objetos:

- A classe é a descrição dos atributos e serviços comuns a um grupo de objetos (reais ou abstratos), logo podemos dizer que é um modelo a partir do qual objetos são construídos.
- Objetos são instâncias de classes.
- Objetos são coisas (Carro, Foto, Bola, etc) e classes são um agrupamento de coisas.



- Identidade é o que torna o objeto único;
- Estado se refere aos seus atributos;
- Comportamento se refere aos seus métodos e procedimentos.

### 1.3. Atributos:

- **Atributos de classes:**

Similar a uma variável global, trata-se de uma variável cujo valor é comum a todos os objetos membros de uma classe. Mudar o valor de uma variável de classe em um objeto membro automaticamente muda o valor para todos os objetos membros.

- **Atributos de objetos (instância):**

Trata-se de uma variável cujo valor é específico ao objeto e, não, à classe. Em geral, possui um valor diferente para cada instância. As linguagens de programação possuem palavras para definir o escopo da variável (Ex: em Java, por padrão, é de instância; para ser de classe, deve vir precedida de *static*).

### 1.4. Métodos:

- Similares a procedimentos e funções, os métodos consistem em descrições das operações que um objeto executa quando recebe uma mensagem.

*Os métodos construtores são métodos especiais, que são chamados automaticamente quando instâncias são criadas. Seu objetivo é garantir que o objeto será instanciado de forma correta. Ele tem exatamente o mesmo nome da classe em que está inserido, não possui tipo de retorno e não é obrigatório declará-lo.*

- **Método de Classe:**

Similar a um método global, trata-se de um método que realiza operações genéricas, isto é, não relativas a uma instância particular. Linguagens de programação possuem palavras para definir o escopo do método (Ex: em Java, deve vir precedida de *static*).

- **Método de Instância:**

Similar a um método local, trata-se de um método que realiza operações específicas para um objeto e, não, à classe, isto é, são relativas a uma instância particular. Por padrão, todos os métodos de uma determinada classe são considerados métodos de instância.

- **Early binding:**

Também conhecida como Ligação Estática, ocorre quando o método a ser invocado é definido em tempo de compilação.

- **Late binding:**

Também conhecida como Ligação Dinâmica, ocorre quando o método a ser invocado é definido em tempo de execução.

### 1.5. Abstração:

- Trata-se da habilidade de concentrar nos aspectos essenciais de um contexto qualquer, ignorando características menos importantes ou acidentais. Em modelagem orientada a objetos, uma classe é uma abstração de entidades existentes no domínio do sistema de software.



- Uma classe abstrata é desenvolvida para representar entidades e conceitos abstratos.
- Ela é sempre uma superclasse (classe mãe) que não possui instâncias.
- Ela define um modelo/template para uma funcionalidade e fornece uma implementação incompleta (isto é, a parte genérica dessa funcionalidade) que é compartilhada por um grupo de classes derivadas. Cada uma das classes derivadas completa a funcionalidade da classe abstrata adicionando comportamentos específicos.
- Elas geralmente contêm ao menos um método abstrato (sem corpo) e não se pode criar uma instância dela.
- As classes abstratas são usadas para serem herdadas e funcionam como uma superclasse.
- Se trata de um contrato para que alguma subclasse concretize seus métodos.
- Métodos abstratos: aqueles para os quais não é definida uma forma de implementação específica.
- Classes concretas implementam todos os seus métodos e permitem a criação de instâncias. Em suma: classes concretas são aquelas que podem ser instanciadas diretamente e classes abstratas, não.

## 1.6. Interface

- Trata-se de um contrato sem implementação entre dois ou mais objetos. É utilizada para reduzir o acoplamento, facilitando o reuso de classes.
- Elas não possuem atributos, apenas assinaturas dos métodos – sendo que todos os métodos são, por padrão, abstract e public.

CARACTERÍSTICAS	INTERFACES	CLASSE ABSTRATA
HERANÇA MÚLTIPLA	Suporta Herança Múltipla. Pode implementar diversas interfaces.	Não suporta Herança Múltipla. Não pode estender várias classes abstratas.
IMPLEMENTAÇÃO	Não pode conter qualquer método concreto, apenas abstratos.	Pode conter métodos concretos ou abstratos.
CONSTANTES	Suporta somente constantes estáticas.	Suporta constantes estáticas e de instância.
ENCAPSULAMENTO	Métodos e membros devem sempre ser públicos por padrão.	Métodos e membros podem ter qualquer visibilidade.
MEMBROS DE DADOS	Não contêm atributos de instância, apenas constantes.	Pode conter atributos de instância.
CONSTRUTORES	Não contêm construtores.	Contém construtores.
VELOCIDADE	Em geral, são mais lentas que classes abstratas.	Em geral, são mais rápidas que interfaces.

- Uma interface é similar a uma classe abstrata. Aliás, podemos dizer que uma interface é praticamente uma classe abstrata pura, isto é, todos os seus métodos são abstratos. Uma classe concreta – ao implementar uma interface – deverá escrever o corpo de todos os métodos. Observem, portanto, que uma classe abstrata pode também implementar uma interface sem nenhum problema.
- A classe abstrata pode conter métodos concretos, então se você quer representar métodos concretos, você pode utilizar classes abstratas, mas não poderá jamais utilizar interfaces (que não pode conter métodos concretos).
- Existe outra diferença importante: você pode declarar variáveis em classes abstratas, mas se você declarar uma variável em uma interface, ela não terá o comportamento de uma variável, mas – sim – de uma constante (será implicitamente *public static final*). Não é papel da interface lidar com o estado interno de um objeto – é muito raro ver atributos em uma interface. Por fim, se você usar uma classe abstrata pura, realmente não tem muitas diferenças práticas em relação a interfaces.
- De todo modo, conceitualmente, uma classe abstrata especifica o que um objeto é; uma interface especifica o que um objeto pode fazer.

## 1.7. Encapsulamento:

- O mecanismo de encapsulamento é uma forma de restringir o acesso ao comportamento interno de um objeto. Um objeto que precise da colaboração de outro objeto para realizar alguma operação simplesmente envia uma mensagem a este último.
- Trata-se de uma forma de restringir o acesso ao comportamento interno de um objeto de modo a evitar que eles sofram acessos indevidos. A ideia é tornar o software mais flexível, fácil de modificar e de criar novas implementações.
- Segundo o mecanismo do encapsulamento, o método que o objeto requisitado utiliza para realizar a operação não é conhecido dos objetos requisitantes. Em outras palavras, o objeto remetente da mensagem não precisa conhecer a forma pela qual a operação requisitada é realizada; tudo o que importa a esse objeto remetente é obter a operação realizada, não importando como.
- No entanto, o remetente da mensagem precisa conhecer pelo menos quais operações o receptor sabe realizar ou o que ele pode oferecer. Para tal, as classes descrevem seus comportamentos por meio de uma interface! Ela descreve o que o objeto sabe fazer, sem precisar detalhar como ele fará!

- Através do encapsulamento, a única coisa que um objeto precisa saber para pedir a colaboração de outro objeto é conhecer a sua interface.
- Isso contribui para a autonomia dos objetos, pois cada objeto envia mensagens a outros objetos para realizar certas operações, sem se preocupar em como se realizaram as operações.
- A interface permite que a implementação de uma operação pode ser trocada sem que o objeto requisitante dela precise ser alterado. Isso mantém as partes de um sistema tão independentes quanto possível.

## 1.8. Polimorfismo:

- Trata-se capacidade de abstrair várias implementações diferentes em uma única interface. Em outras palavras, é o princípio pelo qual duas ou mais classes derivadas da mesma superclasse podem invocar métodos que têm a mesma assinatura, mas comportamentos distintos.

- **Assinatura de um método:**

Dois métodos possuem a mesma assinatura se, e somente se, tiverem o mesmo nome e os mesmos parâmetros (quantidade, tipo e ordem dos parâmetros).

- O polimorfismo diz respeito à capacidade de duas ou mais classes de objetos responderem à mesma mensagem, cada qual de seu próprio modo.
- O polimorfismo permite que a mesma mensagem seja enviada a diferentes objetos e que cada objeto execute a operação que é mais apropriada a sua classe.
- Há uma relação estreita com o conceito de abstração, na medida em que um objeto pode enviar a mesma mensagem para objetos semelhantes, mas que implementam a sua interface de formas diferentes.

- **Polimorfismo Estático:**

Também conhecido como polimorfismo por sobrecarga ou overloading, é representado com o nome do método igual e parâmetros diferentes. A decisão do método a ser chamado é tomada em tempo de compilação de acordo com os argumentos passados. Pode ser uma diferença na quantidade, tipo ou ordem dos parâmetros.

- **Polimorfismo Dinâmico (por sobrescrita, por inclusão, por herança, por subtipo, redefinição ou overriding):**

Está associado ao conceito de herança e é representado com o nome e parâmetros do método iguais. Nesse caso, a subclasse redefine o método da superclasse e a decisão do método a ser chamado é tomada em tempo de execução.

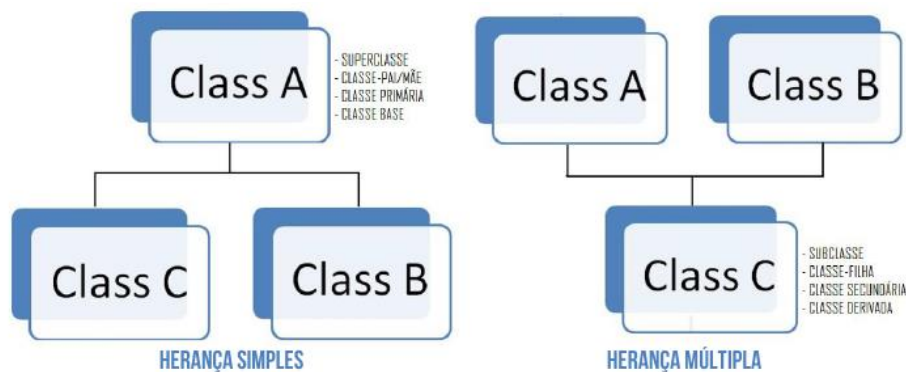
## 1.9. Herança (Generalização/Especialização):

- Trata-se do princípio que permite que classes compartilhem atributos e métodos. É utilizada com a intenção de reaproveitar código ou comportamento generalizado ou especializar operações ou atributos.
- A herança é outra forma de abstração utilizada na orientação a objetos que pode ser vista como um nível de abstração acima da encontrada entre classes e objetos. Na Herança, classes semelhantes são agrupadas em uma hierarquia. Cada nível dessa hierarquia pode ser visto como um nível de



abstração. Trata-se de uma relação entre classes e, não, entre objetos. Antes de prosseguir, vamos falar um pouco sobre nomenclatura!

- A Classe que herda é chamada Subclasse, Classe-Filha, Classe Secundária ou Classe Derivada. A Classe que é herdada é chamada Superclasse, Classe-Pai/Mãe, Classe Primária ou Classe Base. A semântica do código da herança é variável de acordo com a linguagem de programação utilizada (Ex: em Java, utiliza-se a palavra-chave `extends`). Cada classe em um nível de hierarquia herda as características e o comportamento das classes às quais está associada nos níveis acima dela.
- Além disso, essa classe pode definir características e comportamento particulares. Dessa forma, uma classe pode ser criada a partir do reúso da definição de classes preexistentes. A herança facilita o compartilhamento de comportamento comum entre classes. Podemos dizer que se trata do mecanismo que permite que classes compartilhem atributos e métodos, com o intuito de reaproveitar o comportamento generalizado ou especializar operações e atributos.



- Quando uma subclasse herda diretamente de duas ou mais superclasses, trata-se de herança múltipla; já quando uma subclasse herda diretamente de apenas uma superclasse, trata-se de herança simples.

*Pegadinha clássica de concursos de tecnologia da informação: em orientação a objetos, é permitido herança múltipla? É claro que sim! O lance é que algumas linguagens de programação específicas não implementam herança múltipla (Ex: Java e C#).*

- Um detalhe importante: uma subclasse sempre herdará métodos/atributos de suas superclasses – não importa se é herança simples ou herança múltipla.
- Herdar é diferente de acessar! Se uma classe estende a outra, ela sempre herdará seus métodos/atributos.
- Define-se herança como um mecanismo que permite ao programador basear uma nova classe na definição de uma classe previamente existente.
- Usando herança, sua nova classe herda todos os atributos e comportamentos presentes na classe previamente existente. Quando uma classe herda de outra, todos os métodos e atributos que surgem na interface da classe previamente existente aparecerão automaticamente na interface da nova classe. Já o polimorfismo permite que um único nome de método represente um código diferente, selecionado por algum mecanismo automático.
- Dessa forma, um nome pode assumir muitas formas e, como pode representar código diferente, o mesmo nome pode representar muitos comportamentos diferentes.

## 1.10. Questões

### 1.10.1. CESGRANRIO – Profissional de Nível Superior (ELETRONUCLEAR)/Analista de Sistemas/Aplicações e Segurança de TIC/2022 (e mais 1 concurso)

O encapsulamento é um dos quatro conceitos fundamentais da orientação a objetos. Seu objetivo é tratar os dados (variáveis) e as operações sobre esses dados (métodos), de forma unitária. De acordo com esse princípio, as variáveis de uma classe ficam ocultas de outras classes, de forma que só possam ser acessadas pelos métodos públicos da classe em que se encontram.

Entretanto, a maioria das linguagens de programação orientadas a objetos disponibilizam um mecanismo para que o encapsulamento possa ser atenuado sob certas condições. Por exemplo, as variáveis de instância de uma classe podem ser livremente acessadas pelos métodos de instância de todas as suas subclasses.

Esse acesso é possível, caso essas variáveis sejam declaradas como

#### A) Protegidas

CERTO: De acordo com o texto, as variáveis de instância de uma classe podem ser livremente acessadas pelos métodos de instância de todas as suas subclasses. Essa é a característica das variáveis declaradas como "protected" (protegidas), que têm visibilidade restrita ao pacote e às subclasses.

#### B) Estáticas

ERRADO: As variáveis estáticas pertencem à classe e não às instâncias da classe. Portanto, elas não são afetadas pelo mecanismo de encapsulamento em relação ao acesso às variáveis de instância de uma classe.

#### C) Globais

ERRADO: As variáveis globais não estão relacionadas ao encapsulamento dentro do contexto da orientação a objetos. Elas possuem escopo global e podem ser acessadas de qualquer lugar no programa, o que viola o princípio do encapsulamento.

#### D) Automáticas

ERRADO: As variáveis automáticas, também conhecidas como variáveis locais, são criadas e destruídas automaticamente durante a execução de um bloco de código específico, como um método. Elas também não estão relacionadas ao encapsulamento em si.

#### E) Virtuais

ERRADO: As funções virtuais são um conceito relacionado à herança e à polimorfismo, mas não estão diretamente relacionadas ao acesso às variáveis de instância de uma classe. Portanto, não estão corretamente relacionadas ao mecanismo de atenuação do encapsulamento mencionado no texto.

### 1.10.2. CESGRANRIO – Técnico Científico (BASA)/Tecnologia da Informação/2021

Ao construir uma aplicação bancária, um projetista de software modelou a classe “Conta”. Posteriormente, percebeu que cada instância da classe “Conta” poderia ter um conjunto de responsabilidades variadas e independentes, sendo que uma requisição poderia ter que ser atendida por uma ou várias dessas responsabilidades. Isso não permitiria usar de forma eficiente o mecanismo de subclasses para representar essas responsabilidades. Buscando uma solução adequada para essa limitação, o projetista encontrou um padrão de projeto que permite adicionar e retirar dinamicamente responsabilidades apenas aos objetos individuais, e não à classe inteira, estendendo a funcionalidade do objeto, o que seria a solução ideal para o seu caso.

A questão aborda um padrão de projeto específico que permite adicionar e retirar dinamicamente responsabilidades aos objetos individuais, estendendo sua funcionalidade. Isso é uma descrição clássica do padrão de projeto Decorator.

**A) superclasse abstrata, por exemplo “ComponenteConta”, que também é superclasse de uma segunda classe, e essa segunda classe, também abstrata, será superclasse das várias classes concretas que representam as responsabilidades adicionais.**

{CERTO}: Esta opção descreve corretamente o padrão de projeto Decorator. No padrão Decorator, temos uma superclasse abstrata, que seria a “ComponenteConta” no exemplo, e outras classes que herdam desta superclasse, também conhecidas como classes decoradoras, que são usadas para adicionar funcionalidades extras (responsabilidades adicionais) aos objetos.

**B) classe, por exemplo “InterfaceConta”, que converte a interface de uma classe em outra interface que o cliente espera, evitando incompatibilidades causadas por interfaces diferentes.**

{ERRADO}: Esta opção descreve o padrão de projeto Adapter, que é usado para converter a interface de uma classe em outra interface esperada pelos clientes. O Adapter permite que classes com interfaces incompatíveis trabalhem juntas.

**C) classe, por exemplo “FabricaContas”, que separa a construção de um objeto complexo da sua representação, de forma que o mesmo processo de construção possa criar diferentes representações.**

{ERRADO}: Esta opção descreve o padrão de projeto Builder, que é utilizado para construir um objeto complexo passo a passo e o mesmo processo de construção pode criar diferentes tipos e representações do objeto

**D) classe que define uma dependência um-para-muitos entre objetos, de forma que, quando o estado de um objeto da classe “Conta” é alterado, todos os outros objetos dependentes são notificados e podem implementar atualização automática de suas propriedades, em uma relação publicar-subscriver.**

{ERRADO}: Esta opção descreve o padrão de projeto Observer, que é usado quando existe uma relação de um-para-muitos entre objetos, de tal forma que, quando um objeto muda de estado, todos os seus dependentes são notificados e atualizados automaticamente.

**E) classe abstrata, por exemplo “InterfaceConta”, cuja finalidade é definir a interface que permite que suas subclasses tratem uma requisição, sendo que as subclasses concretas são estruturadas em uma cadeia onde cada classe trata a requisição ou a envia para a classe sucessora, até que uma delas atenda a requisição.**

{ERRADO}: Esta opção descreve o padrão de projeto Chain of Responsibility. Neste padrão, uma requisição percorre uma cadeia de potenciais objetos responsáveis até que um deles possa lidar com ela. Se um objeto não pode atender a requisição, ele passa a responsabilidade para o próximo objeto na cadeia.

### 1.10.3. CESGRANRIO – Técnico (UNIRIO)/Tecnologia da Informação/2019

Em orientação a objetos, uma classe abstrata é uma classe que

**A) possui apenas métodos estáticos.**

ERRADO: Uma classe abstrata não é definida pela presença exclusiva de métodos estáticos. Ela pode ter métodos estáticos, mas também pode ter métodos de instância, métodos abstratos, variáveis de instância, etc. A característica definidora de uma classe abstrata é que ela não pode ser instanciada diretamente.

**B) não pode ser instanciada.**

CERTO: Esta é a definição correta de uma classe abstrata. Em orientação a objetos, uma classe abstrata é uma classe que não pode ser instanciada diretamente. Ela serve como uma classe base a partir da qual outras classes podem ser derivadas. Essas classes derivadas são obrigadas a implementar quaisquer métodos abstratos definidos na classe abstrata.

**C) não possui métodos.**

ERRADO: Uma classe abstrata pode, sim, possuir métodos. Aliás, é comum que classes abstratas definam métodos abstratos, que são métodos sem implementação na classe abstrata, mas que devem ser implementados por qualquer classe que herde desta classe abstrata.

**D) não possui variáveis de instância.**

ERRADO: Classes abstratas podem possuir variáveis de instância. Uma variável de instância é uma variável que é associada a uma instância de uma classe. Embora uma classe abstrata não possa ser instanciada diretamente, as classes que herdam dela podem ser instanciadas, e essas instâncias podem ter variáveis de instância definidas na classe abstrata.

**E) não pode ter subclasses.**

ERRADO: Justamente o oposto é verdadeiro. Uma das principais finalidades de uma classe abstrata é servir como uma classe base a partir da qual outras classes podem ser derivadas. Essas classes derivadas são chamadas subclasses da classe abstrata. Portanto, uma classe abstrata não apenas pode ter subclasses, como esse é um dos seus principais usos.

#### 1.10.4. CESGRANRIO - Analista de Sistemas Júnior (TRANSPETRO)/Processos de Negócio/2018

Qual a propriedade, típica da orientação a objeto, que habilita uma quantidade de operações diferentes a ter o mesmo nome, diminuindo o acoplamento entre objetos?

##### A) Encapsulamento

ERRADO: O encapsulamento não se refere à habilidade de ter várias operações com o mesmo nome, mas sim à prática de esconder detalhes de implementação de uma classe, expondo apenas métodos e propriedades seguros ao uso externo. O encapsulamento ajuda a aumentar a segurança e a simplicidade do código.

##### B) Especialização

ERRADO: A especialização é um princípio relacionado à herança, onde uma classe filha (subclasse) se especializa em um comportamento mais específico do que a classe mãe (superclasse). No entanto, isso não tem relação direta com a habilidade de ter várias operações com o mesmo nome.

##### C) Herança

ERRADO: A herança é um princípio que permite a uma classe herdar características e comportamentos de uma outra classe. Embora a herança possa estar envolvida em situações onde operações de mesmo nome se comportam de maneira diferente em classes diferentes (por exemplo, um método sendo sobrescrito na subclasse), ela não é a propriedade que habilita explicitamente essa funcionalidade.

##### D) Padrões de projeto

ERRADO: Padrões de projeto são soluções reutilizáveis para problemas comuns encontrados no desenvolvimento de software. Embora alguns padrões possam envolver o uso de polimorfismo, eles não são, em si mesmos, a propriedade que habilita a existência de várias operações com o mesmo nome.

##### E) Polimorfismo

CERTO: O polimorfismo é a propriedade que permite que operações diferentes tenham o mesmo nome. No contexto da orientação a objetos

### 1.10.5. CESGRANRIO - Analista de Gestão Corporativa (EPE)/Tecnologia da Informação/2012

Na programação orientada a objeto, na linguagem C# em particular, a capacidade de construir vários métodos com um mesmo nome, porém com parâmetros diferentes na mesma classe, é chamada de

#### A) Polimorfismo universal

ERRADO: O polimorfismo universal não é um termo comumente usado na programação orientada a objetos. O polimorfismo, por si só, refere-se à capacidade de um objeto poder ser tratado como uma instância de diferentes tipos, mas não é específico para a situação de múltiplos métodos com o mesmo nome e parâmetros diferentes na mesma classe.

#### B) Polimorfismo paramétrico

ERRADO: O polimorfismo paramétrico é um conceito mais relacionado a tipos genéricos e não descreve a situação de métodos com o mesmo nome e parâmetros diferentes na mesma classe.

#### C) Polimorfismo de subtipo

ERRADO: O polimorfismo de subtipo permite que um objeto da subclasse possa ser tratado como um objeto da superclasse, o que é diferente de ter vários métodos com o mesmo nome, mas com parâmetros diferentes na mesma classe.

#### D) Sobrecarga de operadores

ERRADO: A sobrecarga de operadores refere-se à capacidade de alterar o comportamento de operadores em tipos personalizados, como a adição de dois objetos de uma classe específica. Não se refere à situação de múltiplos métodos com o mesmo nome e parâmetros diferentes na mesma classe.

#### E) Sobrecarga de métodos

CERTO: A sobrecarga de métodos é a capacidade de ter vários métodos com o mesmo nome, mas com parâmetros diferentes na mesma classe. Esta é uma forma de polimorfismo que permite que diferentes métodos realizem ações similares de formas adequadas aos seus parâmetros específicos.

1.10.6. CESGRANRIO - Profissional Petrobras de Nível Superior  
(PETROBRAS)/Analista de Sistemas/Engenharia de Software/2010/PSP-  
RH-1-2010

```
class B extends A {  
    int m1() {  
        return a + b + c + d + e;  
    }  
}  
  
public class A {  
    static int a;  
    public int b;  
    int c;  
    protected int d;  
    private int e;  
}
```

A classe B acima encontra-se no mesmo pacote que a classe A. O método m1 apresenta erro de compilação porque a seguinte variável não pode ser acessada no ponto

A) a.

ERRADO: A variável 'a' é declarada como 'static', o que significa que ela pertence à classe 'A' e não a uma instância específica dela. Ela pode ser acessada por qualquer outra classe, incluindo 'B'.

B) b.

ERRADO: A variável 'b' é declarada como 'public', o que significa que ela pode ser acessada por qualquer classe, incluindo 'B'.

C) c.

ERRADO: A variável 'c' tem o modificador de acesso padrão (também chamado de package-private em Java). Isso significa que ela pode ser acessada por qualquer classe no mesmo pacote. Como 'B' está no mesmo pacote que 'A', ela pode acessar 'c'.

D) d.

ERRADO: A variável 'd' é declarada como 'protected', o que significa que ela pode ser acessada por qualquer classe no mesmo pacote, bem como por subclasses de 'A'. Como 'B' é uma subclasse de 'A' e está no mesmo pacote, ela pode acessar 'd'.

E) e.

CERTO: A variável 'e' é declarada como 'private', o que significa que ela só pode ser acessada dentro da classe 'A'. Mesmo que 'B' seja uma subclasse de 'A', ela não pode acessar 'e' porque 'e' é private. Isso causaria um erro de compilação.

1.10.7. CESGRANRIO - Profissional Petrobras de Nível Superior  
(PETROBRAS)/Analista de Sistemas/Engenharia de Software/2010/PSP-  
RH-1-2010

```
class B extends A {
    static int m1() { return 0; }
    int m2() { return 1; }
}

public class A {
    static int m1() { return 2; }
    int m2() { return 3; }

    public static void
    main(String[] args) {
        A a = new B();

        System.out.println(a.m1()+a.m2()+B.m1());
    }
}
```

A saída da execução da classe A é

- A) 1
- B) 2
- C) 3**
- D) 4
- E) 5

Primeiramente, é importante entender como funciona a sobrecarga e a sobreposição de métodos em Java.

No caso do método `m1()`, estamos lidando com um método estático. Métodos estáticos pertencem à classe, não a instâncias da classe. Eles não são sujeitos à sobreposição (overriding), mas sim à ocultação (hiding). Portanto, `A.m1()` e `B.m1()` são métodos diferentes. A chamada ao método é determinada pelo tipo de referência, não pelo tipo do objeto referenciado.

Já o método `m2()` é um método de instância e é sujeito à sobreposição. Quando um método é sobreposto em uma subclasse, a versão na subclasse é chamada quando o método é invocado em um objeto da subclasse, mesmo se esse objeto está sendo referenciado como um objeto da superclasse.

Dito isso, vamos analisar as chamadas no método `main()`:

`a.m1()`: Embora `a` seja uma instância de `B`, a referência é do tipo `A`, então `A.m1()` é chamado, retornando 2.

`a.m2()`: Aqui a referência `a` é do tipo `A`, mas aponta para um objeto do tipo `B`. Como `m2()` é um método de instância e `B` o sobrepõe, `B.m2()` é chamado, retornando 1.

`B.m1()`: A referência é do tipo `B`, então `B.m1()` é chamado, retornando 0.

Então, somando os retornos de `a.m1()`, `a.m2()` e `B.m1()`, temos  $2 + 1 + 0 = 3$ .

Portanto, a resposta correta é:

- C) 3



1.10.8. CESGRANRIO – Profissional Petrobras de Nível Superior  
(PETROBRAS)/Analista de Sistemas/Engenharia de Software/2010/PSP-  
RH-1-2010

```
1 class Base {  
2     protected int a;  
3     private int b;  
4     public int c;  
5 }  
6  
7 class Filha extends Base {  
8     public int c;  
9  
10    public void metodo1() {  
11        a=b+c;  
12    }  
13 }
```

Linguagens orientadas a objeto possuem modificadores de acesso que são palavras-chaves que costumam limitar ou liberar o acesso a variáveis e/ou métodos, de forma a implementar o conceito de encapsulamento. Existem vários modificadores de acesso em Java que controlam este acesso, tais como os modificadores public, private e protected. Com base nestes conceitos, qual será o resultado obtido se o fragmento de código acima for incluído em uma classe e compilado em linguagem Java?

A) Um erro de compilação na linha 10, pois atributos com modificadores private só podem ser acessados na classe que os definem.

CERTO: No método metodo1(), a variável 'b' está sendo acessada, porém ela é declarada como 'private' na classe Base. Isso significa que 'b' só pode ser acessada dentro da classe Base e não em suas subclasses. Portanto, ocorrerá um erro de compilação na linha 10.

B) Um erro de compilação na linha 8, pois atributos com modificadores public não podem ser redefinidos nas classes filhas.

ERRADO: Atributos com modificadores 'public' podem ser redefinidos nas classes filhas. A resposta correta está na opção A.

C) Um erro de compilação na linha 7, pois classes que contêm atributos com modificadores de acesso private são finais, isto é, não podem ser extendidas.

ERRADO: A presença de atributos com modificadores de acesso 'private' não torna uma classe final e não impede que ela seja estendida.

D) Uma exceção na linha 11, pois apesar de reconhecer o atributo b em tempo de compilação, o fato dele ter modificador de acesso private impede que a classe filha obtenha seu valor em tempo de execução.

ERRADO: O erro ocorre em tempo de compilação, como mencionado na opção A. Não há exceção em tempo de execução, pois o código nem mesmo será compilado devido ao erro na linha 10.

E) A compilação bem sucedida do código.

ERRADO: A resposta correta está na opção A, já que ocorre um erro de compilação na linha 10 devido ao acesso indevido à variável 'b'.

### 1.10.9. CESGRANRIO – Especialista em Recursos Minerais (ANM)/Tecnologia da Informação Mineral/2006

A Orientação a Objetos (OO) é um dos grandes paradigmas que servem hoje de base para o desenho de soluções de software. Para os conceitos relacionados a OO, são feitas as seguintes afirmativas:

I – o mecanismo de herança permite que uma classe filha possa herdar atributos e métodos da classe mãe;

CORRETO: A herança é um dos principais conceitos da programação orientada a objetos. Ela permite que uma classe (chamada de subclasse ou classe filha) herde características (atributos e métodos) de outra classe (chamada de superclasse ou classe mãe).

II – a sobreposição é um tipo de polimorfismo que permite definir em uma mesma classe métodos com o mesmo nome, mas com funções e assinaturas diferentes;

ERRADO: A descrição está incorreta. O que permite definir em uma mesma classe métodos com o mesmo nome, mas com funções e assinaturas diferentes é a sobrecarga de métodos, não a sobreposição. A sobreposição (ou overriding, em inglês) é um tipo de polimorfismo onde um método de uma subclasse tem o mesmo nome, retorno e parâmetros que um método da superclasse, podendo então alterar o comportamento definido na superclasse.

III – a coesão múltipla é uma propriedade por meio da qual uma variável poderá apontar para objetos de diferentes classes em momentos distintos.

ERRADO: A descrição parece estar confusa. A coesão é uma medida de quão fortemente relacionadas e focadas as responsabilidades de um módulo ou classe estão. Não tem relação com uma variável apontar para objetos de diferentes classes. O conceito que permite que uma variável aponte para objetos de diferentes classes em momentos distintos é o polimorfismo.

Está(ão) correta(s) a(s) afirmativa(s):

A

I, apenas.

B

II, apenas.

C

III, apenas.

D

I e II, apenas.

E

I, II e III.

Portanto, apenas a afirmativa I está correta.

A resposta correta é:

A) I, apenas.

### 1.10.10. CESGRANRIO – Técnico Bancário Novo (CEF)/Tecnologia da Informação/2008

Na POO (Programação Orientada a Objetos), qual o princípio em que duas ou mais classes, derivadas de uma mesma superclasse, podem invocar métodos que têm a mesma identificação (assinatura), mas comportamentos distintos, especializados para cada classe derivada?

#### A) Herança múltipla

ERRADO: Herança múltipla refere-se à capacidade de uma classe herdar comportamentos e atributos de mais de uma superclasse. Isso não se refere diretamente ao princípio onde métodos com a mesma identificação têm comportamentos distintos nas classes derivadas.

#### B) Encapsulamento

ERRADO: O encapsulamento é um princípio que oculta os detalhes de implementação de uma classe, expondo apenas métodos e propriedades seguras para o uso. Ele visa proteger o objeto, garantindo que seus dados internos sejam acessados apenas de maneira controlada.

#### C) Polimorfismo

CERTO: O polimorfismo é o princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma identificação, mas comportamentos distintos, especializados para cada classe derivada. Esse princípio permite que objetos de diferentes tipos sejam tratados como objetos de um tipo comum, aumentando assim a flexibilidade e a reutilização do código.

#### D) Abstração

ERRADO: A abstração é o processo de simplificar um sistema complexo modelando classes apropriadas para ele. Ela permite que os programadores lidem com problemas complexos, dividindo-os em partes menores ou mais simples.

#### E) Reuso

ERRADO: O reuso é um conceito que se refere à prática de usar partes do código em diferentes programas ou partes de um programa, com o objetivo de reduzir redundâncias e melhorar a manutenibilidade do código. Embora a herança e o polimorfismo possam facilitar o reuso, essa opção não é a resposta para a pergunta em questão.

## 2. Conceitos e Propriedades do Java

### 2.1. CESGRANRIO - Escriturário (BB)/Agente de Tecnologia/2023

Um programador foi instruído pelo seu gerente a implementar, em Java, uma classe `MemoriaCalculoVenda` que implementasse a interface `MemoriaCalculo`, já criada pela organização e que representa as exigências da organização para classes que implementam memórias de cálculo.

Nesse cenário, com que fragmento de código o programador deve começar, de forma correta, a implementação da classe?

A) `class MemoriaCalculoVenda extends MemoriaCalculo`

{ERRADO}: A palavra-chave "extends" em Java é usada para estabelecer uma relação de herança entre classes, não entre uma classe e uma interface. Neste caso, `MemoriaCalculo` é uma interface, então a palavra-chave "extends" não é apropriada.

B) `class MemoriaCalculoVenda implements MemoriaCalculo`

{CERTO}: A palavra-chave "implements" em Java é usada para indicar que uma classe irá implementar uma ou mais interfaces. Isso significa que a classe compromete-se a fornecer implementações para todos os métodos declarados na(s) interface(s) que está implementando. Neste caso, a classe `MemoriaCalculoVenda` deve implementar todos os métodos definidos na interface `MemoriaCalculo`.

C) `class MemoriaCalculoVenda imports MemoriaCalculo`

{ERRADO}: A palavra-chave "imports" é usada para importar classes ou pacotes em Java, mas não para estabelecer uma relação entre uma classe e uma interface. Não é o caso aqui.

D) `class MemoriaCalculoVenda inherits MemoriaCalculo`

{ERRADO}: Embora a palavra "inherits" possa sugerir uma relação de herança, essa não é uma palavra-chave válida em Java. A herança entre classes é expressa com a palavra-chave "extends", e a implementação de interfaces é expressa com a palavra-chave "implements". Portanto, essa opção é inválida.

E) `class MemoriaCalculoVenda uses MemoriaCalculo`

{ERRADO}: "uses" não é uma palavra-chave em Java e, portanto, não pode ser usada para estabelecer uma relação entre uma classe e uma interface. A relação correta entre uma classe e uma interface em Java é expressa com a palavra-chave "implements". Portanto, essa opção é inválida.

## 2.2. CESGRANRIO - Escriturário (BB)/Agente de Tecnologia/2023

Um método Java, chamado `converte`, deve receber uma `string` (`str`) como parâmetro e retornar uma `string` igual a `str`, exceto pelas letras minúsculas, que devem ser convertidas em letras maiúsculas.

Exemplo: `String` recebida: "Abc \$12d"

`String` retornada: "ABC \$12D"

Qual método realiza essa tarefa?

**A**

```
public static String converte(String str) {
    String r=" ";
    int i=0;

    do {
        char x=str.charAt(i);
        if(Character.isLowerCase(x))
            r=r.concat(Character.toString(Character.toUpperCase(x)));
        else
            r=r.concat(Character.toString(x));
        i++;
    } while(i < str.length());
    return r;
}
```

{CERTO}: Este método inicializa uma variável de `string` 'r' com um espaço em branco. Em seguida, ele percorre a `string` de entrada 'str' utilizando um loop `do-while`. Para cada caractere, verifica se é minúsculo e, se for, converte para maiúsculo e adiciona à `string` 'r'. Se o caractere não for minúsculo, é simplesmente adicionado à `string` 'r'. No entanto, note que a `string` 'r' começará com um espaço em branco extra.

**B**

```
public static String converte(String str) {
    String r=null;
    int i=0;

    while(i < str.length()) {
        char x=str.charAt(i);
        if(Character.isLowerCase(x))
            r=r.concat(Character.toString(Character.toUpperCase(x)));
        else
            r=r.concat(Character.toString(x));
        i++;
    }
    return r;
}
```

{ERRADO}: Este método inicializa a variável de `string` 'r' como `null`. Quando tentamos usar o método 'concat' em 'r', um `NullPointerException` será lançado, pois 'r' é nulo. Portanto, este método irá falhar ao executar.

```

public static String converte(String str) {
    String r=null;
    int i=0;

    while(i < str.length()) {
        if(str[i].isLowerCase(x))
            r=r.concat(Character.toString(Character.toUpperCase(str[i])));
        else
            r=r.concat(Character.toString(str[i]));
        i++;
    }
    return r;
}

```

**{ERRADO}**: A expressão `str[i]` é inválida em Java, pois `String` não é um tipo de array. Além disso, a variável `'x'` não é definida e a variável `'r'` é inicializada como `null`, o que causará um `NullPointerException` quando tentarmos usar o método `'concat'` em `'r'`.

```

public static String converte(String str) {
    String r=" ";

    for(char x : str)
        if(Character.isLowerCase(x))
            r=r.concat(Character.toString(Character.toUpperCase(x)));
        else
            r=r.concat(Character.toString(x));
    return r;
}

```

**{ERRADO}**: A sintaxe `"for(char x : str)"` é inválida em Java. O operador `":"` é utilizado para iterar sobre elementos de uma coleção ou um array, mas `'str'` é uma string, não uma coleção ou um array.

```

public static String converte(String str) {
    String r=" ";

    for(int i=0; i < str.length(); i++)
        if(Character.isLowerCase(str.charAt(i)))
            r=r+Character.toUpperCase(str.charAt(i));
        else
            r=r+str.charAt(i);
    return r;
}

```

**{CERTO}**: Este método inicializa uma variável de string `'r'` com um espaço em branco. Em seguida, percorre a string de entrada `'str'` utilizando um loop `for`. Para cada caractere, verifica se é minúsculo e, se for, converte para maiúsculo e adiciona à string `'r'`. Se o caractere não for minúsculo, é simplesmente adicionado à string `'r'`. No entanto, note que a string `'r'` começará com um espaço em branco extra.

### 2.3. CESGRANRIO – Técnico Científico (BASA)/Tecnologia da Informação/2022

Qual definição de interface Java NÃO produz erro de compilação?

```
interface P {  
    int x;  
    int y;  
    public void op1(String x);  
}
```

{ERRADO}: Em uma interface Java, todas as variáveis são implicitamente public, static e final. No entanto, essas variáveis precisam ser inicializadas no momento da declaração. Como as variáveis 'x' e 'y' não são inicializadas, isso causará um erro de compilação.

```
abstract interface Q {  
    public int x;  
    public int y;  
    public default void op1() {  
        System.out.println(x+y);  
    }  
}
```

{ERRADO}: Novamente, todas as variáveis em uma interface Java são implicitamente public, static e final e precisam ser inicializadas no momento da declaração. As variáveis 'x' e 'y' não são inicializadas, o que causa um erro de compilação. Além disso, a palavra-chave 'abstract' é redundante, pois todas as interfaces são implicitamente abstratas.

```
final interface R {  
    int x=10;  
    int y=20;  
    public static void op1() {  
        System.out.println(x+y);  
    }  
}
```

{ERRADO}: A palavra-chave 'final' é usada para indicar que algo não pode ser alterado. No contexto das interfaces, 'final' não faz sentido porque uma interface não pode ser estendida. Portanto, 'final interface' causará um erro de compilação.

```
public interface S {  
    public static int x=10;  
    public static int y=20;  
    public static void op1() {  
        x++;  
        y++;  
        System.out.println(x+y);  
    }  
}
```

{ERRADO}: Como mencionado anteriormente, todas as variáveis em uma interface são implicitamente public, static e final. Isso significa que elas não podem ser modificadas após a inicialização. No método 'op1', estamos tentando incrementar as variáveis 'x' e 'y', o que não é permitido e causará um erro de compilação.

```
public abstract interface T {  
    static int x=20;  
    public int y=20;  
    static void op1() {  
        System.out.println(x+y);  
    }  
}
```

{CERTO}: Embora a palavra-chave 'abstract' seja redundante (todas as interfaces são implicitamente abstratas), ela não causará um erro de compilação. As variáveis 'x' e 'y' são corretamente declaradas e inicializadas, e o método 'op1' está corretamente definido como um método estático. Portanto, esta definição de interface é válida e não produzirá um erro de compilação.

## 2.4. CESGRANRIO – Técnico Científico (BASA)/Tecnologia da Informação/2018

Considere o código Java listado a seguir, onde a numeração de linhas está sendo utilizada apenas como referência:

```
i. package teste;  
ii. public class Classe {  
iii. private static int a = 5;  
iv. public static void main(String[] args) {  
v.         int a = 8;  
vi.  
vii. }  
viii. }
```

Que comando deve ser inserido na linha vi para exibir o valor 5 na console?

A) `System.out.println(a);`

{ERRADO}: Esta opção é inválida porque a sintaxe está incorreta. O método `println()` exige que seus argumentos estejam dentro de parênteses.

B) `System.out.println(Classe.a);`

{CERTO}: Esta opção é correta porque ela acessa corretamente a variável estática 'a' da classe 'Classe'. Em Java, variáveis estáticas são acessadas através do nome da classe.

C) `System.out.println(Integer.parseInt(a));`

{ERRADO}: Esta opção é inválida porque a sintaxe está incorreta. Além disso, o método `Integer.parseInt()` é usado para converter uma string em um inteiro, o que não é necessário neste caso porque 'a' já é um inteiro.

D) `System.out.println(super.a);`

{ERRADO}: A palavra-chave 'super' é usada para referenciar a classe pai de uma classe. No entanto, o método `main()` está em uma classe que não tem uma classe pai explicitamente definida. Além disso, 'super' não pode ser usado em um contexto estático.

E) `System.out.println(this.a);`

{ERRADO}: A palavra-chave 'this' é usada para referenciar a instância atual de uma classe. No entanto, o método `main()` é um método estático, o que significa que ele pertence à classe e não a uma instância específica da classe. Portanto, 'this' não pode ser usado em um contexto estático.



## 2.5. CESGRANRIO – Profissional Básico (BNDES)/Análise de Sistemas – Desenvolvimento/2013

Na linguagem Java, a palavra-chave final pode ser usada na declaração de classes, de métodos e de variáveis.

Quando essa palavra-chave é usada na declaração de uma classe, ela indica que

A) a classe não pode ser estendida.

{CERTO}: A palavra-chave 'final' quando aplicada a uma classe em Java indica que a classe não pode ser estendida. Ou seja, nenhuma outra classe pode herdar dessa classe.

B) a classe só pode ser instanciada uma única vez.

{ERRADO}: A palavra-chave 'final' não impede que uma classe seja instanciada mais de uma vez. Portanto, a afirmação é falsa.

C) a classe é considerada uma interface.

{ERRADO}: A palavra-chave 'final' não tem nada a ver com a classe ser uma interface. Interfaces e classes são conceitos separados em Java.

D) as variáveis da classe só podem sofrer atribuições de valores uma única vez.

{ERRADO}: Embora a palavra-chave 'final' faça com que uma variável só possa ser atribuída uma vez, isso não se aplica automaticamente a todas as variáveis em uma classe final. Cada variável deve ser declarada individualmente como final para ter essa propriedade.

E) os métodos da classe não podem ser sobrescritos.

{ERRADO}: Embora os métodos de uma classe final não possam ser sobrescritos devido ao fato de que a classe não pode ser estendida, a palavra-chave 'final' não precisa ser aplicada à classe para que seus métodos não possam ser sobrescritos. Um método pode ser declarado como final dentro de uma classe não-final para evitar sua sobrescrita.

## 2.6. CESGRANRIO – Profissional Petrobrás de Nível Técnico (PETROBRAS)/Informática/2012

Java é uma linguagem fortemente tipada, havendo regras de conversão específicas entre os tipos. Para facilitar o desenvolvimento de programas, entretanto, existem algumas conversões implícitas (typecast) que são feitas automaticamente pela JVM.

Um técnico de informática foi chamado para avaliar, com base nesses conceitos, um programa na linguagem Java cujas instruções estão na seguinte ordem:

- 1º - Integer meuInteger=2;
- 2º - int meuInt= new Integer(2);
- 3º - String umaString= meuInteger;
- 4º - String outraString= ""+meuInt;

Uma vez que as instruções foram colocadas em um único programa na ordem em que foram apresentadas, o técnico identificou que causará(ão) erro de compilação apenas a(s) seguinte(s) instrução(ões):

A) 1ª

{ERRADO}: A primeira instrução Integer meuInteger=2; é válida. Em Java, isso é chamado de autoboxing, onde um tipo primitivo é automaticamente encapsulado em seu equivalente de tipo de objeto.

B) 3ª

{CERTO}: A terceira instrução String umaString= meuInteger; causará um erro de compilação. Em Java, não é possível atribuir diretamente um objeto Integer a uma variável String. Uma conversão explícita seria necessária para converter o Integer em String.

C) 1ª e 3ª

{ERRADO}: A primeira instrução é válida, conforme explicado anteriormente. A terceira instrução é inválida, portanto, essa opção está parcialmente correta.

D) 2ª e 3ª

{ERRADO}: A segunda instrução int meuInt= new Integer(2); é válida. Isso é chamado de unboxing, onde um objeto do tipo Wrapper é convertido de volta para seu tipo primitivo correspondente. A terceira instrução é inválida, portanto, essa opção está parcialmente correta.

E) 2ª e 4ª

{ERRADO}: As instruções segunda e quarta são válidas. A quarta instrução String outraString= ""+meuInt; é uma forma comum de converter um inteiro em uma string em Java. Portanto, essa opção está completamente errada.

## 2.7. CESGRANRIO – Profissional Petrobrás de Nível Técnico (PETROBRAS)/Exploração de Petróleo/Informática/2012/PSP-RH-2-2011

Ao escrever o código da Classe PortaDeCofre em Java para que ela atenda a interface Porta, como um programador deve começar a declaração da classe?

A) `public class Porta:PortaDeCofre {`

{ERRADO}: A sintaxe usada nesta opção é inválida em Java. A palavra-chave correta para implementar uma interface é 'implements'.

B) `public class PortaDeCofre :: Porta {`

{ERRADO}: A sintaxe usada nesta opção é inválida em Java. O símbolo '::' não é usado para implementar uma interface ou estender uma classe.

C) `public class PortaDeCofre inherits Porta {`

{ERRADO}: Embora 'inherits' seja a palavra-chave usada para herança em algumas linguagens de programação, ela não é usada em Java. Em Java, 'extends' é usada para herança de classes, e 'implements' é usada para implementação de interfaces.

D) `public class PortaDeCofre extends Porta {`

{ERRADO}: A palavra-chave 'extends' é usada para herança de classes em Java. No entanto, a questão menciona que 'Porta' é uma interface. Portanto, a palavra-chave correta para usar neste caso é 'implements'.

E) `public class PortaDeCofre implements Porta {`

{CERTO}: Esta opção é correta. Em Java, a palavra-chave 'implements' é usada para implementar uma interface em uma classe. Portanto, a declaração correta para a classe 'PortaDeCofre' implementar a interface 'Porta' é 'public class PortaDeCofre implements Porta'.

2.8. CESGRANRIO – Profissional Petrobras de Nível Superior  
(PETROBRAS)/Analista de Sistemas/Engenharia de Software/2011/PSP-RH-1  
2011

A linguagem de programação Java, lançada em 1995, tem demonstrado ser muito estável.

A respeito dessa linguagem, considere as afirmativas a seguir.

I – Java é uma linguagem orientada a objetos de herança simples e mista que contém tipos de dados primitivos, como int e objetos.

{ERRADO}: A linguagem Java suporta herança simples, não herança múltipla ou mista. Classes em Java podem herdar de apenas uma superclasse. No entanto, uma classe em Java pode implementar várias interfaces, o que pode dar a impressão de herança múltipla, mas não é verdadeira herança múltipla.

II – Java usa semântica de cópia para tipos de dados primitivos, e semântica de referência para objetos.

{CERTO}: Quando um tipo de dado primitivo é atribuído a outro, o valor é copiado. Por outro lado, quando um objeto é atribuído a outro, apenas a referência ao objeto é copiada, não o objeto em si. Portanto, ambos se referem ao mesmo objeto na memória.

III – Java é uma linguagem multiplataforma, com enfoque no desenvolvimento de aplicações para a Web.

{CERTO}: Java é de fato uma linguagem de programação multiplataforma, o que significa que o código Java pode ser executado em qualquer sistema operacional que possua uma Máquina Virtual Java (JVM). Além disso, Java é comumente usado para desenvolver aplicações para a Web.

Está correto o que se afirma em

A

I, apenas.

B

III, apenas.

C

I e II, apenas.

D

II e III, apenas.

E

I, II e III.

Portanto, a resposta correta é:

D) II e III, apenas.

## 2.9. CESGRANRIO – Profissional Petrobrás de Nível Técnico (PETROBRAS)/Informática/2011/PSP-RH-1 2011

Entre outros métodos da linguagem Java, o método pertencente à Classe String que remove espaços em branco existentes no início ou no final de uma string é o

A) `abs()`

{ERRADO}: O método `abs()` é um método da classe `Math`, não da classe `String`. Ele retorna o valor absoluto de um número, o que não tem relação com a remoção de espaços em branco de uma string.

B) `trim()`

{CERTO}: O método `trim()` é um método da classe `String` em Java. Ele remove os espaços em branco no início e no final de uma string.

C) `exit()`

{ERRADO}: O método `exit()` é um método da classe `System`, não da classe `String`. Ele é usado para terminar a execução do programa atual.

D) `load()`

{ERRADO}: O método `load()` é um método da classe `System` e da classe `Runtime`, não da classe `String`. Ele é usado para carregar arquivos de biblioteca dinâmica.

E) `random()`

{ERRADO}: O método `random()` é um método da classe `Math`, não da classe `String`. Ele gera um número aleatório entre 0.0 (inclusivo) e 1.0 (exclusivo).

## 2.10. CESGRANRIO - Profissional Básico (BNDES)/Análise de Sistemas - Desenvolvimento/2009

Qual das afirmações a seguir faz uma apreciação correta a respeito da linguagem de programação Java?

A) O conceito de herança múltipla é implementado nativamente.

{ERRADO}: Java não suporta herança múltipla diretamente. Uma classe em Java pode herdar de apenas uma superclasse. No entanto, uma classe pode implementar várias interfaces.

B) Uma classe pode implementar somente uma interface ao mesmo tempo.

{ERRADO}: Uma classe em Java pode implementar várias interfaces ao mesmo tempo.

C) Uma classe pode implementar uma interface ou ser subclasse de outra classe qualquer, mas não ambos simultaneamente.

{ERRADO}: Uma classe em Java pode ser subclasse de outra classe e implementar uma ou mais interfaces ao mesmo tempo.

D) A construção de um método que pode levantar uma exceção, cuja instância é uma subclasse de `java.lang.RuntimeException`, não exige tratamento obrigatório por parte do programador dentro daquele método.

{CERTO}: Em Java, exceções que são subclasses de `RuntimeException` são chamadas de exceções não verificadas e não exigem tratamento obrigatório (ou seja, não é necessário usar `try/catch` ou declarar que o método "lança" a exceção com a palavra-chave `"throws"`).

E) Objetos da classe `java.lang.String` têm comportamento otimizado para permitir que seu valor seja alterado sempre que necessário, liberando imediatamente a memória usada pelo conteúdo anterior.

{ERRADO}: Em Java, objetos da classe `String` são imutáveis, o que significa que, uma vez criados, o valor que eles representam não pode ser alterado. Se você quiser alterar o valor de uma string, um novo objeto `String` será criado. O Garbage Collector do Java eventualmente libera a memória do objeto `String` original quando não há mais referências a ele.

## 2.11. CESGRANRIO – Técnico Bancário Novo (CEF)/Tecnologia da Informação/2008

NÃO é uma característica da linguagem JAVA:

A) suporte à Unicode.

{CERTO}: Java suporta Unicode para representação de strings e caracteres.

B) orientação a objetos.

{CERTO}: Java é uma linguagem orientada a objetos. Ela suporta conceitos como classes, objetos, herança, polimorfismo e encapsulamento.

C) herança múltipla.

{ERRADO}: Java não suporta herança múltipla diretamente devido a problemas como a ambiguidade que ela pode causar. No entanto, uma classe em Java pode implementar várias interfaces, o que é uma forma de alcançar a funcionalidade de herança múltipla.

D) multiplataforma.

{CERTO}: Java é uma linguagem multiplataforma. Um programa Java pode ser executado em qualquer dispositivo que tenha uma Máquina Virtual Java (JVM), independentemente do sistema operacional subjacente.

E) sintaxe similar a C/C++.

{CERTO}: A sintaxe de Java é semelhante à de C e C++. Se você conhece C ou C++, provavelmente achará a sintaxe de Java bastante familiar.

## 2.12. CESGRANRIO – Assistente Legislativo Especializado (ALTO)/Programação de Computadores/2005

Relacione a palavra-chave do Java à sua respectiva descrição.

Palavra-chave

I – case

II – super

III – throw

Descrição

(P) Usada para gerar uma exceção.

(Q) Em um método ou construtor, refere-se à classe de base dessa classe.

A relação correta é:

A) I – P, II – Q

B) I – P, III – Q

C) I – Q, II – P

D) II – P, III – Q

E) II – Q, III – P

A correspondência correta entre as palavras-chave do Java e suas respectivas descrições é:

I – case: Esta palavra-chave é usada em um bloco switch para marcar um bloco de código que será executado quando um valor específico for correspondido. Portanto, não corresponde a nenhuma das descrições dadas.

II – super: Em um método ou construtor, refere-se à classe de base dessa classe. Assim, II corresponde a (Q).

III – throw: Usada para gerar uma exceção. Assim, III corresponde a (P).

Portanto, a alternativa correta é:

E) II – Q, III – P



### 2.13. FUNRIO - Analista de Desenvolvimento (AgeRIO)/Análise de Sistemas/2010

Na linguagem de programação Java, qual o modificador que serve para declarar uma variável como constante?

A) throw

{ERRADO}: 'throw' é uma palavra-chave em Java usada para indicar explicitamente uma exceção, não tem relação com a declaração de constantes.

B) void

{ERRADO}: 'void' é um tipo de retorno para métodos em Java, que indica que o método não retorna nenhum valor. Não está relacionado à declaração de constantes.

C) static

{ERRADO}: 'static' é um modificador que indica que um membro da classe (campo ou método) pertence à classe e não a uma instância específica da classe. Não está diretamente relacionado à declaração de constantes.

D) final

{CERTO}: 'final' é o modificador usado para declarar uma variável como constante em Java. Quando uma variável é declarada como 'final', seu valor não pode ser modificado após a inicialização.

E) private

{ERRADO}: 'private' é um modificador de acesso que restringe a visibilidade de um membro da classe (campo ou método) apenas à própria classe. Não está relacionado à declaração de constantes.

## 2.14. FUNRIO - Analista de Desenvolvimento (AgeRIO)/Análise de Sistemas/2010

Qual o significado de classe abstrata, declarada na linguagem de programação Java pelo modificador “abstract”?

A) Classe que pode ser utilizada apenas no mesmo package.

{ERRADO}: A palavra-chave 'abstract' não limita a utilização de uma classe ao mesmo pacote. Ela tem a ver com a instância da classe, não com sua visibilidade.

B) Classe que pode ser utilizada apenas na classe externa.

{ERRADO}: A declaração 'abstract' não limita a utilização de uma classe apenas na classe externa.

C) Classe que pode ser instanciada e utilizada livremente.

{ERRADO}: Classes abstratas não podem ser instanciadas diretamente. Elas são usadas como base para outras classes.

D) Classe que não pode ser instanciada.

{CERTO}: Classes abstratas não podem ser instanciadas diretamente. Elas são tipicamente usadas como classes base para serem estendidas por outras classes.

E) Classe que não permite definição de subclasses.

{ERRADO}: Classes abstratas são precisamente projetadas para serem estendidas por outras classes. É por isso que elas existem: para fornecer características comuns às subclasses.



### 3. Código Java

#### 3.1. CESGRANRIO - Técnico Científico (BASA)/Tecnologia da Informação/2022

Sejam dois arrays de inteiros, com zero ou mais elementos cada, ordenados ascendentemente. Deseja-se escrever uma função que receba esses dois arrays como parâmetros e insira os seus elementos em um terceiro array, também recebido como parâmetro, de modo que os elementos inseridos no terceiro array permaneçam ordenados ascendentemente, como no exemplo abaixo.

Exemplo:

```
int v1[]={10,20,30,40,50};  
int v2[]={5,10,15,20};
```

O conteúdo do terceiro array, após a chamada da função de intercalação, será

**{5,10,10,15,20,20,30,40,50}**

Nesse contexto, considere a seguinte função main de um programa Java:

```
public class Main {  
    public static void main(String[] args) {  
        int v1[]={10,20,30,40,50};  
        int v2[]={5,10,15,20};  
        int v3[]=new int [v1.length + v2.length];  
        int p1=0,p2=0,p3=0;  
  
        intercala(v1,p1,v2,p2,v3,p3);  
    }  
}
```

Qual função deve ser inserida na classe Main para que a intercalação do array v1 com o array v2 seja feita corretamente?

```
static void intercala(int v1[],int p1,int v2[],int p2,int v3[],int p3) {  
    if(p1 == v1.length && p2 == v2.length)  
        return;  
    if(v1[p1] < v2[p2]) {  
        v3[p3++]=v1[p1++];  
        intercala(v1,p1,v2,p2,v3,p3);  
    }  
    else {  
        v3[p3++]=v2[p2++];  
        intercala(v1,p1,v2,p2,v3,p3);  
    }  
}
```

A) ERRADO: Esta função não verifica adequadamente se p1 ou p2 ultrapassaram os limites dos arrays v1 e v2, respectivamente. Isso pode causar um erro de `ArrayIndexOutOfBoundsException`.

```

static void intercala(int v1[],int p1,int v2[],int p2,int v3[],int p3) {
    if(p1 == v1.length && p2 == v2.length)
        return;
    if(p1 < v1.length)
        if(p2 < v2.length)
            if(v1[p1] < v2[p2]) {
                v3[p3++] = v1[p1++];
                intercala(v1,p1,v2,p2,v3,p3);
            }
            else {
                v3[p3++] = v2[p2++];
                intercala(v1,p1,v2,p2,v3,p3);
            }
        }
        else {
            v3[p3++] = v1[p1++];
            intercala(v1,p1,v2,p2,v3,p3);
        }
    }
    else
        if(p2 < v2.length) {
            v3[p3++] = v2[p2++];
            intercala(v1,p1,v2,p2,v3,p3);
        }
    }
}

```

B

```

java

static void intercala(int v1[],int p1,int v2[],int p2,int v3[],int p3) {
    // 1º IF
    if(p1 == v1.length && p2 == v2.length)
        return;

    // 2º IF
    if(p1 < v1.length)
        // 3º IF
        if(p2 < v2.length)
            // 4º IF
            if(v1[p1] < v2[p2]) {
                v3[p3++] = v1[p1++];
                intercala(v1,p1,v2,p2,v3,p3);
            }
            // 1º ELSE
            else {
                v3[p3++] = v2[p2++];
                intercala(v1,p1,v2,p2,v3,p3);
            }
            // 2º ELSE
            else {
                v3[p3++] = v1[p1++];
                intercala(v1,p1,v2,p2,v3,p3);
            }
        }
        // 3º ELSE
        else
            // 5º IF
            if(p2 < v2.length) {
                v3[p3++] = v2[p2++];
                intercala(v1,p1,v2,p2,v3,p3);
            }
    }
}

```

- O 1º `if` verifica se já percorremos todos os elementos de ambos os arrays `v1` e `v2`. Se sim, a função retorna, pois não há mais elementos para intercalar.
- O 2º `if` é a primeira verificação após sabermos que ainda existem elementos para intercalar. Ele verifica se ainda temos elementos no array `v1` para intercalar.
- Dentro do bloco do 2º `if`, temos o 3º `if`, que verifica se ainda temos elementos no array `v2` para intercalar. Isso ocorre se ambos os arrays ainda tiverem elementos a serem intercalados.
- Dentro do bloco do 3º `if`, temos o 4º `if`, que compara os elementos atuais de `v1` e `v2`. Se o elemento atual de `v1` for menor, ele é colocado no array `v3` e os índices correspondentes são incrementados. A função é então chamada recursivamente com os novos índices.
- O 1º `else` é a contrapartida do 4º `if`. Se o elemento atual de `v2` for menor ou igual, ele é colocado no array `v3` e os índices correspondentes são incrementados. A função é então chamada recursivamente com os novos índices.
- O 2º `else` é a contrapartida do 3º `if`. Se já percorremos todos os elementos de `v2`, mas ainda temos elementos em `v1`, o elemento atual de `v1` é colocado no array `v3` e os índices correspondentes são incrementados. A função é então chamada recursivamente com os novos índices.
- O 3º `else` é a contrapartida do 2º `if`. Se já percorremos todos os elementos de `v1`, mas ainda temos elementos em `v2`, o controle passa para o próximo `if`.
- Dentro do bloco do 3º `else`, temos o 5º `if`, que verifica se ainda temos elementos no array `v2` para intercalar. Isso ocorre se já percorremos todos os elementos de `v1`, mas ainda temos elementos em `v2`. Se verdadeiro, o elemento atual de `v2` é colocado no array `v3` e os índices correspondentes são incrementados. A função é então chamada recursivamente com os novos índices.

B) CERTO: Esta função implementa a lógica correta para a intercalação dos dois arrays.

- Ela primeiro verifica se p1 e p2 estão dentro dos limites de v1 e v2, respectivamente.
- Em seguida, compara os elementos atuais dos dois arrays e insere o menor no array v3.
- Se um dos arrays já tiver sido totalmente percorrido, ela insere os elementos restantes do outro array em v3.
- A recursão continua até que todos os elementos de ambos os arrays tenham sido inseridos em v3 em ordem ascendente.

```

static void intercala(int v1[],int p1,int v2[],int p2,int v3[],int p3) {
    if(p1 < v1.length && p2 < v2.length)
        intercala(v1,p1+1,v2,p2+1,v3,p3+1);
    else
        if(p1 < v1.length && p2 == v2.length)
            intercala(v1,p1+1,v2,p2,v3,p3+1);
        else
            if(p1 == v1.length && p2 < v2.length)
                intercala(v1,p1,v2,p2+1,v3,p3+1);
            else
                return;
    if(p1 < v1.length)
        if(p2 < v2.length)
            if(v1[p1] < v2[p2])
                v3[p3]=v1[p1];
            else
                v3[p3]=v2[p2];
        else
            v3[p3]=v1[p1];
    else
        v3[p3]=v2[p2];
}

```

C) ERRADO:

- Esta função não atualiza adequadamente os índices p1 e p2 ao inserir elementos no array v3.
- não insere corretamente elementos repetidos no array v3.
- não há incremento nas posições do vetor. Com isso as trocas acontecem sempre na mesma posição.

```

static void intercala(int v1[],int p1,int v2[],int p2,int v3[],int p3) {
    while(p1 < v1.length && p2 < v2.length)
        if(v1[p1] < v2[p2]) {
            v3[p3]=v1[p1];
            p3+=1;
            p1+=1;
        }
        else
            if(v1[p1] > v2[p2]) {
                v3[p3]=v2[p2];
                p3+=1;
                p2+=1;
            }
        else {
            v3[p3]=v1[p1];
            p1+=1;
            p3+=1;
            v3[p3]=v2[p2];
            p3+=1;
            p2+=1;
        }
}

```

D) ERRADO: Esta função não insere corretamente elementos repetidos no array v3. Além disso, não insere os elementos restantes do array mais longo em v3 uma vez que o array mais curto tenha sido totalmente percorrido.

```
static void intercala(int v1[],int p1,int v2[],int p2,int v3[],int p3) {  
    while(p1 < v1.length || p2 < v2.length)  
    {  
        if(v1[p1] < v2[p2]) {  
            v3[p3]=v1[p1];  
            p3++;  
            p1++;  
        }  
        else  
        {  
            if(v1[p1] > v2[p2]) {  
                v3[p3]=v2[p2];  
                p3++;  
                p2++;  
            }  
            else {  
                v3[p3]=v1[p1];  
                p1++;  
                p3++;  
                v3[p3]=v2[p2];  
                p3++;  
                p2++;  
            }  
        }  
    }  
}
```

E) ERRADO: Mesmo problema da alternativa A: esta função não verifica adequadamente se p1 ou p2 ultrapassaram os limites dos arrays v1 e v2, respectivamente. Isso pode causar um erro de `ArrayIndexOutOfBoundsException`.

Portanto, a opção B é a correta. Ela implementa a lógica correta para a intercalação dos dois arrays em ordem ascendente.

### 3.2. CESGRANRIO – Técnico Científico (BASA)/Tecnologia da Informação/2021

A classe Java a seguir contém dois métodos (busca e buscaBin) que implementam um algoritmo de busca binária sobre um array de inteiros.

```
public class Main {
    public static void main(String[] args) {
        int array[] = {220,158,133,100,98,96,80,60,55,22,8};
        busca(array,61);
    }
    public static int busca(int vet[], int elem) {
        return buscaBin(vet,elem,0,vet.length-1);
    }
    private static int buscaBin(int vet[], int elem, int ini, int fin) {
        if(ini > fin)
            return -1;
        int m=(ini+fin)/2;
        System.out.printf("%d ",vet[m]);
        if(vet[m]==elem)
            return m;
        else
            if(vet[m]>elem)
                return buscaBin(vet,elem,m+1,fin);
            else
                return buscaBin(vet,elem,ini,m-1);
    }
}
```

O que será exibido no console quando o método main() for executado?

- A) 96 80 60
- B) 96 133 220
- C) 96 55 60 80
- D) 96 55 80 60
- E) 96 133 158 220

O método buscaBin realiza uma busca binária no array. A busca binária é um algoritmo eficiente para encontrar um elemento em um array ordenado. Ela começa comparando o elemento no meio do array com o elemento de destino. Se o elemento do meio é igual ao elemento de destino, a busca termina. Se o elemento do meio é maior que o elemento de destino, a busca continua na metade inferior do array. Se o elemento do meio é menor que o elemento de destino, a busca continua na metade superior do array.

No caso presente, o elemento de destino é 61.

- Primeiramente, a função buscaBin é chamada com ini = 0 e fin = 10 (os índices do primeiro e do último elemento do array, respectivamente). O meio do array (índice m) é calculado como  $(ini+fin)/2 = 5$ , e o valor correspondente  $array[m] = 96$  é impresso. Como  $96 > 61$ , a busca prossegue na metade inferior do array, chamando buscaBin com ini = m+1 = 6 e fin = 10.
- Agora, ini = 6 e fin = 10. O meio do array é  $(ini+fin)/2 = 8$ , e o valor correspondente  $array[m] = 55$  é impresso. Como  $55 < 61$ , a busca prossegue na metade superior do array, chamando buscaBin com ini = 6 e fin = m-1 = 7.
- Em seguida, ini = 6 e fin = 7. O meio do array é  $(ini+fin)/2 = 6$ , e o valor correspondente  $array[m] = 80$  é impresso. Como  $80 > 61$ , a busca prossegue na metade inferior do array, chamando buscaBin com ini = m+1 = 7 e fin = 7.
- Por fim, ini = 7 e fin = 7, o meio do array é m = 7 e o valor correspondente  $array[m] = 60$  é impresso. Como  $60 < 61$ , a busca tentaria prosseguir na metade superior do array. No entanto, como ini seria maior que fin, a busca é interrompida.

Portanto, a sequência impressa no console é "96 55 80 60", correspondendo à alternativa (D).



### 3.3. CESGRANRIO – Técnico Científico (BASA)/Tecnologia da Informação/2021

A classe Queue a seguir é uma implementação parcial do tipo abstrato de dados Fila.

```
import java.util.ArrayList;

public class Queue<ELM> {

    private ArrayList<ELM> lst=new ArrayList<ELM>();

    public boolean isEmpty() {
        return lst.isEmpty();
    }

    public void enqueue(ELM s) {
    }

    public ELM dequeue() {
    }

}
```

Nesse contexto, qual implementação dos métodos enqueue() e dequeue() completa a classe Queue, de modo que todos os elementos inseridos em uma fila possam ser recuperados de acordo com a propriedade FIFO?

```
public void enqueue(ELM s) {
    lst.add(s);
}

public ELM dequeue() {
    A if(!lst.isEmpty())
        return lst.get(0);
    else
        return null;
}
```

**ERRADO:**

**FILA:** precisa inserir em uma ponta e ser removida na outra ponta. Essa alternativa insere corretamente numa ponta porém falha ao remover da outra.

dequeue() NÃO usa o comando remove para remover na posição oposta da fila em que foi inserido

```
public void enqueue(ELM s) {
    lst.add(0,s);
}

public ELM dequeue() {
    B if(!lst.isEmpty())
        return lst.remove(lst.size()-1);
    else
        return null;
}
```

**CERTO:**

- enqueue() está adicionando o elemento na EXTREMIDADE ESQUERDA DA FILA;
- dequeue() está removendo o elemento da EXTREMIDADE DIREITA DA FILA;

FILA: precisa inserir em uma ponta e ser removida na outra ponta.

Passo a passo da execução da alternativa B:

- Quando o método enqueue(s) é chamado, o elemento 's' é adicionado NA POSIÇÃO MAIS A ESQUERDA método lst.add(0,s).
- Se o método dequeue() é chamado, primeiro verifica-se se a lista não está vazia com o método lst.isEmpty().
- Se a lista não está vazia, o elemento na POSIÇÃO MAIS A DIREITA da lista é removido e retornado com o método lst.remove(lst.size()-1).
- Se a lista está vazia, então o método retorna null.

```
public void enqueue(ELM s) {  
    lst.add(0,s);  
}  
  
public ELM dequeue() {  
    (C) if(!lst.isEmpty())  
        return lst.remove(0);  
    else  
        return null;  
}
```

ERRADO:

- enqueue() está adicionando o elemento na primeira posição da lista.

Isso resulta numa estrutura em que o último elemento adicionado será o primeiro a ser removido, que também não corresponde à propriedade FIFO.

```
public void enqueue(ELM s) {  
    lst.add(s);  
}  
public ELM dequeue() {  
    (D) if(!lst.isEmpty())  
        return lst.remove(lst.size()-1);  
    else  
        return null;  
}
```

ERRADO:

- dequeue() está removendo o elemento da última posição da lista.

Isso resulta numa estrutura LIFO (Last In, First Out), o oposto de uma fila.

```
public void enqueue(ELM s) {  
    lst.add(lst.size(),s);  
}  
public ELM dequeue() {  
    (E) if(!lst.isEmpty())  
        return lst.remove(lst.size()-1);  
    else  
        return null;  
}
```

ERRADO:

dequeue() está removendo o elemento da última posição da lista.

Isso resulta numa estrutura LIFO (Last In, First Out), o oposto de uma fila.

### 3.4. CESGRANRIO – Técnico Científico (BASA)/Tecnologia da Informação/2021

A classe Java ArvNo, exibida abaixo, é usada para representar os nós de uma árvore binária.

```
package estruturas;

class ArvNo {
    int info;
    ArvNo esq=null,dir=null;
}
```

Ela é usada na implementação de uma árvore binária pela classe Arv, exibida a seguir.

```
package estruturas;

public class Arv {
    private ArvNo raiz;

    public Arv(){
    }

    public void exibe(){
        percorre(raiz);
        System.out.println("\n");
    }

    private void percorre(ArvNo r) {
        if(r==null)
            return;

        percorre(r.dir);
        percorre(r.esq);
        System.out.print(r.info+" ");
    }
}
```

Que árvore terá os valores de seus nós exibidos em ordem descendente quando for percorrida pelo método percorre(), definido na classe Arv?

A questão propõe um código em Java para busca. Ele define um árvore de busca decrescente. A questão quer saber qual das árvores representam o código, de forma que a ordem descendente seja respeitada (55 47 39 35 31 27 20 18 15 10). Dito isso, vamos analisar o que o código imprime:

```
percorre(r.dir);
percorre(r.esq);
System.out.print(r.info+" ");
```

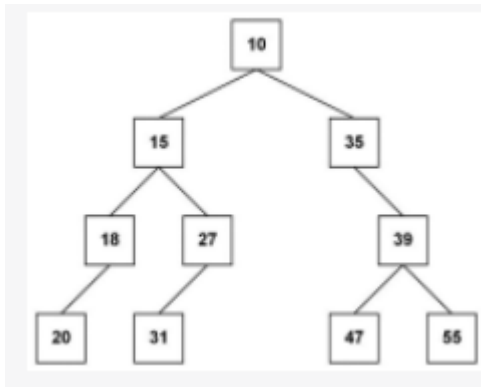
- O método percorre() na classe Arv segue um padrão semelhante ao de um percurso em pós-ordem, porém com uma diferença chave: a ordem dos filhos (DIREITO e depois ESQUERDO)

Em um percurso em pós-ordem padrão, a árvore é percorrida em esquerda, direita, raiz. Neste código, a árvore é percorrida em direita, esquerda, raiz, que é uma variação do percurso em pós-ordem.

Nesse caso, a busca irá começar pela direita e depois para a esquerda, baseada em cada nó. O nó na extrema direita deve possuir o maior valor portanto. As únicas árvores que colocam o 55 na ponta direita são o A e C. Dessa maneira, vamos analisa-las:

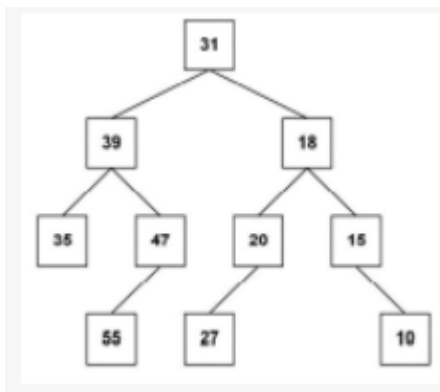
Letra A. Faz a sequência correta, pois o nó 39, tem 55 a direita e 47 a esquerda. Já o 35, não tem nó a esquerda. Por isso, volta-se para o nó raiz e vai para a ponta direita da outra parte da árvore. Assim, acha-se o 31.

Letra C. A esquerda do 39 temos números menores, o que já deixa ela incorreta.



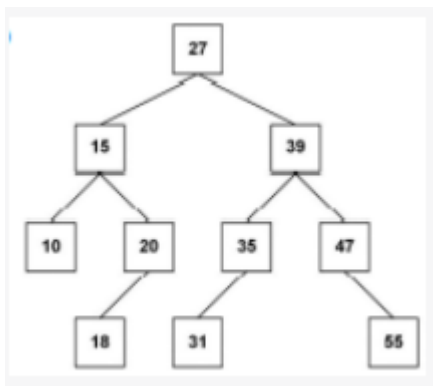
a)

CERTO: Faz a sequência correta, pois o nó 39, tem 55 a direita e 47 a esquerda. Já o 35, não tem nó a esquerda. Por isso, volta-se para o nó raiz e vai para a ponta direita da outra parte da árvore. Assim, acha-se o 31.



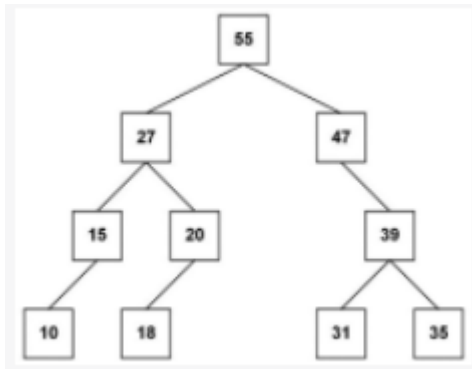
b)

ERRADO: o maior número não está à direita.



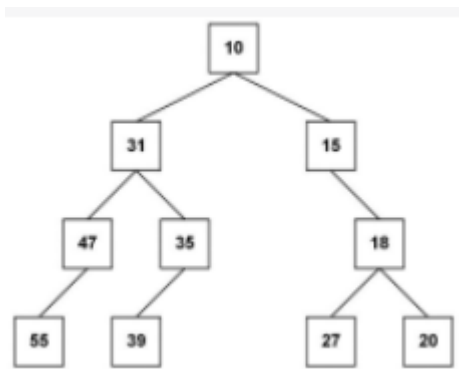
c)

ERRADO: o maior número (55) está à direita, porém à esquerda do 39 temos números menores, o que deixa ela incorreta.



d)

ERRADO: o maior número não está à direita.



e)

ERRADO: o maior número não está à direita.

### 3.5. CESGRANRIO – Técnico Científico (BASA)/Tecnologia da Informação/2021

Considere a classe Java abaixo.

```
public static void main(String[] args) {
    String s1="Brasil",s2="";

    s2=geraString(s1, s2, 0);
    System.out.println(s2);
}

public static String geraString(String s1, String s2, int cont) {
    if(cont==s1.length())
        return s2;

    String vogal="aeiou";
    char c=s1.charAt(s1.length()-cont-1);

    if(vogal.indexOf(c) < 0)
        s2+=Character.toString(c);
    else
        s2+=".";

    cont++;
    return geraString(s1, s2, cont);
}
```

7. cont = 6, que é igual ao comprimento de s1, então a função retorna s2, que é "l.s.rB".

1. cont = 0, c = 'l', como 'l' não é uma vogal, adiciona 'l' a s2. s2 se torna "l".  
3. cont = 2, c = 's', como 's' não é uma vogal, adiciona 's' a s2. s2 se torna "l.s".  
5. cont = 4, c = 'r', como 'r' não é uma vogal, adiciona 'r' a s2. s2 se torna "l.s.r".  
6. cont = 5, c = 'B', como 'B' não é uma vogal, adiciona 'B' a s2. s2 se torna "l.s.rB".

2. cont = 1, c = 'i', como 'i' é uma vogal, adiciona '.' a s2. s2 se torna "l.".   
4. cont = 3, c = 'a', como 'a' é uma vogal, adiciona '.' a s2. s2 se torna "l.s.".

O que o console exibirá quando o método main() for executado?

- O indexOf() método retorna a posição da primeira ocorrência do (s) caractere (s) especificado (s) em uma string.
- O charAt() método retorna o caractere no índice especificado em uma string.
  - O índice do primeiro caractere é 0, o segundo caractere é 1 e assim por diante.

O código percorre a string s1 de trás para frente. Para cada caractere, ele verifica se é uma vogal. Se for, adiciona um ponto à string s2. Se não for, adiciona o caractere à string s2. Este processo continua até que todos os caracteres em s1 tenham sido verificados. O resultado final é a string s2, que é exibida no console.

Aqui está uma execução passo a passo da função geraString para a string s1="Brasil":

- cont = 0, c = 'l', como 'l' não é uma vogal, adiciona 'l' a s2. s2 se torna "l".
- cont = 1, c = 'i', como 'i' é uma vogal, adiciona '.' a s2. s2 se torna "l".
- cont = 2, c = 's', como 's' não é uma vogal, adiciona 's' a s2. s2 se torna "l.s".
- cont = 3, c = 'a', como 'a' é uma vogal, adiciona '.' a s2. s2 se torna "l.s.".
- cont = 4, c = 'r', como 'r' não é uma vogal, adiciona 'r' a s2. s2 se torna "l.s.r".
- cont = 5, c = 'B', como 'B' não é uma vogal, adiciona 'B' a s2. s2 se torna "l.s.rB".
- cont = 6, que é igual ao comprimento de s1, então a função retorna s2, que é "l.s.rB".

Portanto, o resultado exibido no console será "l.s.rB".

### 3.6. CESGRANRIO - Escriturário (BB)/Agente de Tecnologia/2021

As classes Java a seguir são públicas e ocupam arquivos separados.

```
public class Tst {  
  
    int ini=0, fim=25; 2. Primeiro, os campos ini e fim são inicializados com 0 e 25, respectivamente.  
    void print() {  
        System.out.println(ini+fim); 6. O método print é chamado, imprimindo ini+fim, que é 6+18, ou seja, 24.  
    }  
    {  
        ini=fim%7; 3. O primeiro bloco de inicialização é executado. Ele redefine ini para fim%7 (25%7), que é 4,  
        fim=ini*3; e redefine fim para ini*3, que é 12.  
    }  
    Tst(int a, int b) {  
        ini+=a; 5. Agora o construtor é chamado. Ele adiciona os argumentos a e b a ini e fim,  
        fim+=b; respectivamente. Isso redefine ini para ini+4 (2+4), que é 6, e redefine fim para fim-4 (22-  
        4), que é 18.  
    }  
    {  
        ini/=2; 4. O segundo bloco de inicialização é então executado. Ele redefine ini para ini/2 (4/2), que é  
        fim+=10; 2, e redefine fim para fim+10, que é 22.  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        new Tst(4, -4).print(); 1. Um objeto da classe Tst é instanciado com a chamada "new Tst(4, -4)" no método main  
        da classe Main. Antes do construtor ser chamado, os blocos de inicialização são  
        executados na ordem em que aparecem no código.  
    }  
}
```

O que será exibido no console quando o método main for executado?

- Nesta questão, temos que lidar com blocos de inicialização não estáticos e um construtor em uma classe Java. Os blocos de inicialização não estáticos são executados sempre que um objeto da classe é criado, antes de qualquer construtor ser chamado, e na ordem em que aparecem no código.
  - A classe Tst tem dois desses blocos, além de um construtor.
- A ordem de execução segue, basicamente, o seguinte: executamos primeiro os componentes que estão fora dos métodos, em seguida executamos os métodos na ordem em que foram invocados.
  - 1) No nosso código, os componentes que estão fora do método são os seguintes:

```
{  
    ini=fim%7;  
    fim=ini*3;  
}  
  
{  
    ini/=2;  
    fim+=10;  
}
```

- Eles serão executados exatamente nessa ordem. Com isso teremos o seguinte:

```
ini=fim%7 => 25 % 7 => ini = 4, pois queremos o resto da divisão entre 25 e 7.
fim=ini*3 => fim = 4 * 3 => fim = 12
```

- Os valores acima seguem para a próxima execução, portanto:

```
ini/=2 => 4 / 2 => ini = 2
fim+=10 => fim + 10 => fim = 12 + 10 => fim = 22
```

2) Assim, concluímos a execução dos elementos que estão fora dos métodos. Agora, precisamos executar os métodos na ordem em que foram invocados.

- Na classe principal foi dado o seguinte: `new Tst(4, -4).print()`; Isso significa que o primeiro método a ser executado é o `Tst` e o segundo o `print`. O método `Tst` é o que chamamos de método construtor e o `print` método "normal".
- Ao executar o método `Tst` teremos o seguinte:

```
Tst(int a, int b) {
    ini+=a => ini + a => 2 + 4 => ini = 6;
    fim+=b => fim + b => 22 - 4 => fim = 18
}
```

3) Por fim, vamos fazer o que é pedido no método `print`:

```
void print() {
    System.out.println(ini+fim) => ini + fim => 6 + 18 => 24
}
```



### 3.7. CESGRANRIO - Analista (UNIRIO)/Tecnologia da Informação/2019

As classes Java a seguir ocupam arquivos distintos, situados no pacote default.

```
public class CX {
    int v1, v2;
    {
        // Passo 2: o bloco de inicialização não estático de CX é chamado.
        // Mas os valores serão sobrescritos pelo construtor a seguir.
        v1 += 20;
        v2 += 30;
    }

    public CX() {
        // Passo 3: o construtor de CX é chamado e redefine v1 e v2 para 0.
        v1 = v2 = 0;
    }

    public void print() {
        System.out.println(v1 + v2);
    }
}

public class CY extends CX {
    {
        // Passo 4: o bloco de inicialização não estático de CY é chamado.
        // Como v1 e v2 são 0, v1 e v2 permanecem 0.
        v1 *= 2;
        v2 *= 3;
    }

    public CY() {
        // Passo 5: o construtor de CY é chamado, que adiciona 10 a v1 e 20 a v2.
        // Agora v1 é 10 e v2 é 20.
        v1 += 10;
        v2 += 20;
    }
}

public class CZ extends CY {
    public CZ(int c) {
        // Passo 6: o construtor de CZ é chamado, que multiplica v1 e v2 por (3+c),
        // ou seja, 8. Agora v1 é 80 e v2 é 160.
        v1 *= 3 + c;
        v2 *= 3 + c;
    }

    public static void main(String[] args) {
        // Passo 1: A classe CZ é carregada e seu método main é chamado.
        // O objeto CZ é criado com o construtor CZ(5).
        CX p = new CZ(5);

        // Passo 7: O método main chama p.print(), que imprime a soma de v1 e v2, que é 80 + 160 = 240.
        p.print();
    }
}
```

O que será exibido no console quando o método main() for executado?

A ordem de execução deste código será:

1º) Classe CZ instanciada chama pelo construtor da sua classe mãe, que é CY. Como CY também é herdada, ela chamará pelo construtor da sua classe mãe, que é CX.

2º) Antes, porém, de executar o construtor de CX, devemos inicializar as variáveis do seu código de inicialização ( $v1+=20$ ;  $v2+=30$ ;). Logo após executamos o seu construtor ( $v1=v2=0$ ;). Então neste momento temos  $v1 = 0$  e  $v2 = 0$ .

3º) Terminando a classe CX, voltamos para CY, que também possui um bloco de inicialização ( $v1*=2$ ;  $v2*=3$ ;). Executando este código continuaremos com  $v1 = 0$  e  $v2 = 0$  pois 0 vezes qualquer coisa é zero. Agora sim executamos o construtor de CY ( $v1+=10$ ;  $v2+=20$ ;) e com isso teremos  $v1 = 10$  e  $v2 = 20$ .

4º) Finalmente voltamos a classe CZ, que recebeu como parâmetro  $c = 5$ .

$v1 = 10 * (3 + 5)$ ; -> 80

$v2 = 20 * (3 + 5)$ ; -> 160

5º) Imprimindo  $v1 + v2$  (  $80 + 160$  ) = 240

Em linguagens de programação como Java, onde existem estruturas de repetição, a recursão pode ser muitas vezes substituída pela repetição, com ganhos de desempenho.

Considere a seguinte função recursiva segredo, em Java:

```
public static int segredo(int a) {  
    if (a<2) {  
        return 0;  
    } else {  
        return segredo(a-2)+1;  
    }  
}
```

Que fragmento de código, em Java, contendo uma estrutura de repetição, é adequado para substituí-la?

O código original é uma função recursiva que recebe um valor inteiro 'a' como argumento. Se 'a' for menor do que 2, a função retorna 0. Caso contrário, ela chama a si mesma, passando 'a-2' como argumento, e soma 1 ao resultado. Isso efetivamente conta quantas vezes 'a' pode ser reduzido por 2 até se tornar menor que 2.

Agora vamos avaliar as alternativas:

```
public static  
    int alternativaA(int a) { int s = 0;  
    for (int i=a;i>2;i--) {  
        s++;  
    }  
    return s;  
}
```

ERRADO:

- Aqui deveríamos ter maior ou igual a 2.
- Essa opção usa um loop for para decrementar 'i' de 'a' até 2 e incrementar 's' a cada iteração. No entanto, ela não decrementa 'i' por 2 a cada iteração, então não é equivalente à função original.

```
public static int alternativaB(int a) {  
    int s = 0;  
    for (int i=a;i<2 && i>0;i--) {  
        s++;  
    }  
    return s;  
}
```

ERRADO:

- Colocou-se para a condição de entrada ser entre 0 e 2. Não é essa a proposta do código original. Não poderíamos inserir outros números como 3.
- Similar à opção A, essa opção decrementa 'i' de 'a' até 2 ou 0, o que for maior, e incrementa 's' a cada iteração. No entanto, ela ainda não decrementa 'i' por 2 a cada iteração, então também não é equivalente à função original.

```
public static int alternativaC(int a) {  
    int s = 0;  
    while (a>=2) {  
        a-=2;  
        s++;  
    }  
    return s;  
}
```

CERTO: Essa opção usa um loop while para repetir enquanto 'a' é maior ou igual a 2. A cada iteração, ela decrementa 'a' por 2 e incrementa 's'. Isso é equivalente à função original e, portanto, é a alternativa correta.

(D)

```
public static int alternativaD(int a) {
    int s = 0;
    do {
        a-=2;
        s++;
    } while (a>0 && a<=2);
    return s;
}
```

**ERRADO:**

- Inverteu-se novamente os intervalos e as instruções.
- Essa opção usa um loop do-while que decrementa 'a' por 2 e incrementa 's' a cada iteração. No entanto, o loop continua enquanto 'a' é maior que 0 e menor ou igual a 2, que não é equivalente à condição da função original.

(E)

```
public static int alternativaE(int a) {
    int s = 0;
    do {
        a-=2;
        s++;
    } while (a>0 && a<2);
    return s;
}
```

**ERRADO:**

- Inverteu-se os intervalos e as instruções.
- Similar à opção D, essa opção usa um loop do-while que decrementa 'a' por 2 e incrementa 's' a cada iteração. No entanto, o loop continua enquanto 'a' é maior que 0 e menor que 2, que também não é equivalente à condição da função original.

Portanto, a alternativa correta é a Alternativa C.

## 4. PHP

### 4.1. JAVA x PHP

PHP e Java são duas linguagens de programação que possuem diferenças significativas, incluindo a maneira como os arrays são atribuídos e manipulados e como as estruturas de controle, como FOR, WHILE e IF, são utilizadas.

#### 4.1.1. Atribuição de Array:

##### PHP:

Em PHP, a atribuição de arrays é dinâmica e associativa. Você pode criar arrays com qualquer tipo de dados e o tamanho do array é ajustado dinamicamente.

Exemplo:

```
php
$arr = array("A", "B", "C");
$arr = ["A", "B", "C"]; // A partir do PHP 5.4
$associativeArr = array("Name" => "John", "Age" => 30);
```

##### Java:

Em Java, a atribuição de arrays é estática e baseada em índices. Você precisa especificar o tipo de dados e o tamanho do array no momento da criação. Além disso, todos os elementos devem ser do mesmo tipo.

Exemplo:

```
java
int[] arr = new int[3];
arr[0] = 1; arr[1] = 2; arr[2] = 3;
String[] strArr = {"A", "B", "C"};
```

#### 4.1.2. Funções FOR, WHILE e IF:

A sintaxe básica das estruturas de controle FOR, WHILE e IF é bastante semelhante entre PHP e Java. No entanto, existem diferenças sutis devido às diferenças de linguagem.

##### PHP:

```
php
// FOR loop
for ($i = 0; $i < 10; $i++) {
    echo $i;
}
// WHILE loop
$i = 0;
while ($i < 10) {
    echo $i;
    $i++;
}
// IF statement
if ($i > 5) {
    echo "Greater";
} else {
    echo "Not Greater";
}
```

## Java:

```
java

// FOR loop
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}

// WHILE loop
int i = 0;
while (i < 10) {
    System.out.println(i);
    i++;
}

// IF statement
if (i > 5) {
    System.out.println("Greater");
} else {
    System.out.println("Not Greater");
}
```

Uma diferença importante é que, em PHP, as variáveis não precisam ser declaradas com um tipo específico, enquanto em Java, o tipo de dados deve ser especificado ao declarar uma variável.

Além disso, para imprimir na tela, PHP usa a função echo, enquanto Java usa System.out.println.

### 4.1.3. Variáveis

Uma diferença notável entre PHP e Java diz respeito ao uso do símbolo do dólar (\$) na frente das variáveis em PHP, o que não é feito em Java.

#### PHP:

Em PHP, todas as variáveis começam com um símbolo de dólar (\$). Este símbolo é usado para indicar que a palavra seguinte é uma variável. Além disso, o PHP é uma linguagem de tipagem fraca, o que significa que você não precisa declarar o tipo de dados de uma variável quando a define. Aqui estão alguns exemplos de variáveis em PHP:

```
php

$name = "John";
$age = 30;
$isActive = true;
```

#### Java:

Em contraste, em Java, as variáveis não começam com um símbolo de dólar. Além disso, o Java é uma linguagem de tipagem forte, o que significa que você precisa declarar o tipo de dados de uma variável quando a define. Aqui estão alguns exemplos de variáveis em Java:

```
java

String name = "John";
int age = 30;
boolean isActive = true;
```

Portanto, a necessidade de um símbolo de dólar (\$) para as variáveis é uma das diferenças distintas entre PHP e Java. Além disso, a exigência de declaração de tipo em Java, mas não em PHP, é outra diferença fundamental entre as duas linguagens.

## 4.2. Questões

### 4.3. CESGRANRIO - Técnico Científico (BASA)/Tecnologia da Informação/2022

Considere o seguinte trecho de código em PHP:

```
$A = 3;  
$B = "2";  
echo $A+$B;  
echo $B.$A;  
echo $A|$B;  
echo $A&$B;  
echo "\n";
```

Qual será a saída desse trecho de código?

- A) 5623
- B) 5632
- C) 52323
- D) 52332
- E) 332323

Vamos analisar o código passo a passo:

- `$A = 3;`  
A variável `$A` é atribuída ao valor 3 (inteiro).
- `$B = "2";`  
A variável `$B` é atribuída ao valor '2' (string).
- `echo $A+$B;`  
Aqui, a soma de `$A` e `$B` é realizada. Como `$B` é uma string, ela é convertida em um inteiro (2) antes da soma. Portanto, a saída será 5 (3 + 2).
- `echo $B.$A;`  
Nesta linha, a concatenação de strings é realizada. `$A` será convertido em uma string antes da concatenação, então a saída será '23' ('2' . '3').
- `echo $A|$B;`  
Aqui, o operador OR bit a bit é usado. Para realizar essa operação, ambos `$A` e `$B` são convertidos em inteiros (já que `$B` é uma string). Então, temos 3 (em binário, 011) OR 2 (em binário, 010), que é igual a 3 (em binário, 011). Portanto, a saída será 3.
- `echo $A&$B;`  
Nesta linha, o operador AND bit a bit é usado. Como antes, ambos `$A` e `$B` são convertidos em inteiros. Então, temos 3 (em binário, 011) AND 2 (em binário, 010), que é igual a 2 (em binário, 010). Portanto, a saída será 2.
- `echo "\n";` - Aqui, é impressa uma quebra de linha.
- Juntando todas as saídas, temos: 52323.

Então, a resposta correta é a alternativa C (52323).

#### 4.4. CESGRANRIO – Técnico Científico (BASA)/Tecnologia da Informação/2021

O corredor com camisa de numeração 327 foi prejudicado durante uma competição pelos 10 corredores que chegaram logo a sua frente. A direção da corrida resolveu, então, desclassificar esses 10 corredores. Um programador designado para construir um programa que listasse somente os números das camisas desses corredores desclassificados chegou ao programa apresentado abaixo, que recebe como entrada o número de cada um dos competidores pela ordem de chegada, do primeiro ao último, e acha o competidor 327.

```
<?php
$fh = fopen('php://stdin','r');
$array = array();
while ($array[] = fgets($fh));
fclose($fh);
for ($i=0;$array[$i]<>327;$i++);
// TODO: laço para imprimir os 10 anteriores
//
?>
```

No entanto, esse programa está incompleto porque apenas encontra o corredor 327. Que instruções PHP devem ser colocadas, no lugar do comentário marcado com a palavra TODO no programa apresentado, para que sejam impressas as numerações das camisas dos 10 competidores desclassificados?

```
for ($j=0;$j<10;$j++) {
  A    echo $array[$i--];
}
```

{ERRADO}: Este trecho de código imprime os 10 corredores anteriores ao corredor 327, mas o decremento da variável `$i` ocorre após a execução do comando `echo`. Isso significa que o primeiro corredor impresso será o próprio corredor 327, e não um dos corredores que chegou à sua frente.

```
for ($j=0;$j<=10;$j++) {
  B    echo $array[--$i];
}
```

{ERRADO}: Este trecho de código imprime 11 corredores, não 10. Isso acontece porque o loop começa em 0 e vai até 10, inclusive. Além disso, o decremento de `$i` ocorre antes da execução do comando `echo`, então o corredor 327 não seria impresso, mas ainda assim, um corredor adicional seria impresso além dos 10 desclassificados.

```
for ($j=0;$j<10;$j++) {
  C    echo $array[--$i];
}
```

{CERTO}: Este trecho de código é o correto. O decremento da variável `$i` ocorre antes da execução do comando `echo`, garantindo que o corredor 327 não seja impresso. Além disso, como o loop começa em 0 e vai até 9, um total de 10 corredores são impressos, correspondendo aos 10 corredores desclassificados.

```
for ($j=1;$j<10;$j++) {
  D    echo $array[--$i];
}
```

{ERRADO}: Este trecho de código imprime apenas 9 corredores, não 10. Isso acontece porque o loop começa em 1 e vai até 9, o que totaliza 9 iterações.

```
for ($j=0;$j<10;$j++) {
  E    echo $array[$i+$j];
}
```

{ERRADO}: Este trecho de código imprime os 10 corredores que chegaram após o corredor 327, não os que chegaram à sua frente. Isso ocorre porque a variável `$j` é adicionada ao índice `$i`, fazendo com que os corredores subsequentes sejam selecionados ao invés dos anteriores.



#### 4.5. CESGRANRIO – Técnico Científico (BASA)/Tecnologia da Informação/2022

Considere o seguinte fragmento de código em PHP.

```
<?php
$var = 2;
function primeira(&$var) { $var++;}
function segunda($var) { $var++;}
function terceira() { $var++;}
echo $var;
primeira($var); echo $var;
segunda($var);echo $var;
terceira($var); echo $var;
?>
```

Qual será a saída gerada pelo fragmento de código acima?

- A) 2222
- B) 2234
- C) 2333
- D) 2334
- E) 2344

Vamos analisar o código passo a passo:

- `$var = 2;`  
A variável `$var` é atribuída ao valor 2 (inteiro).
- `function primeira(&$var) { $var++;}`  
Esta função recebe uma referência de variável (passagem por referência) e incrementa seu valor em 1.
- `function segunda($var) { $var++;}`  
Esta função recebe uma variável (passagem por valor) e incrementa seu valor em 1.
- `function terceira() { $var++;}`  
Esta função tenta incrementar uma variável `$var` não definida dentro do escopo da função.
- `echo $var;`  
A saída será 2, já que é o valor atual da variável `$var`.
- `primeira($var); echo $var;`  
A função `primeira` é chamada com a variável `$var` como argumento. Como a função recebe uma referência, o valor da variável original será incrementado. Assim, `$var` agora vale 3 e a saída será 3.
- `segunda($var);echo $var;`  
A função `segunda` é chamada com a variável `$var` como argumento. Como a função recebe a variável por valor, ela não afeta o valor original de `$var`. Portanto, a saída será 3.
- `terceira($var); echo $var;`  
A função `terceira` é chamada, mas como a variável `$var` não está definida dentro do escopo da função, ela não afetará o valor original de `$var`. Assim, a saída será 3.

Juntando todas as saídas, temos: 2333.

Então, a resposta correta é a alternativa C (2333).

#### 4.6. CESGRANRIO - Técnico Científico (BASA)/Tecnologia da Informação/Suporte Técnico à Infraestrutura de TI/2014

Um programa CGI feito em PHP pode ser ativado por meio da URL

`http://prova.xx/programa.php?op=1&tx=novo`

Qual linha de código PHP deve ser usada para recuperar o parâmetro `op` e armazená-lo na variável `$myop`?

A) `$myop = $_GET[op]`

{ERRADO}: Neste caso, a chave do array 'op' deve ser uma string, e as strings em PHP devem estar entre aspas. Portanto, a correta sintaxe seria `$_GET['op']`.

B) `$myop = $_GET['op']`

{CERTO}: Este é o método correto para recuperar um parâmetro de URL em PHP. A superglobal `$_GET` é usada para coletar valores enviados por um formulário HTTP com o método GET. Os valores são acessíveis usando a sintaxe `$_GET['nome_do_campo']`.

C) `$myop = $GET['op']`

{ERRADO}: A variável `$GET` não existe em PHP. A superglobal correta para recuperar os parâmetros de URL é `$_GET`.

D) `$myop = cgi_get('op')`

{ERRADO}: Não existe uma função chamada 'cgi\_get' em PHP. Para obter parâmetros de URL, deve-se usar a superglobal `$_GET`.

E) `$myop = cgi_get['op']`

{ERRADO}: Novamente, 'cgi\_get' não é uma função ou variável em PHP. A superglobal correta para recuperar os parâmetros de URL é `$_GET`.

#### 4.7. CESGRANRIO – Técnico Científico (BASA)/Tecnologia da Informação/2018

O comando PHP que cria corretamente um vetor ou conjunto (array) associativo é:

A) `$pessoa=array("Nome":"Ana","Idade":"52", "Cidade":"Manaus");`

{ERRADO}: Este código está usando dois pontos simples (:) para tentar atribuir valores às chaves no array associativo, o que não é a sintaxe correta em PHP. Em PHP, usamos o operador '=' para atribuir valores em um array associativo.


B) `$pessoa=array("Nome"="Ana","Idade"="52", "Cidade"="Manaus");`

{ERRADO}: Este código está usando dois pontos e um igual (:=) para tentar atribuir valores às chaves no array associativo, o que não é a sintaxe correta em PHP. Em PHP, usamos o operador '=' para atribuir valores em um array associativo.

C) `$pessoa=array("Nome"="Ana","Idade"="52", "Cidade"="Manaus");`

{ERRADO}: Este código está usando um igual (=) para tentar atribuir valores às chaves no array associativo, o que não é a sintaxe correta em PHP. Em PHP, usamos o operador '=' para atribuir valores em um array associativo.

D) `$pessoa=array('Nome'=>'Ana', 'Idade'=>'52', 'Cidade'=>'Manaus');`

 `$pessoa=array("Nome"=>"Ana","Idade"=>"52", "Cidade"=>"Manaus");`

{CERTO}: Este código está correto. Em PHP, usamos o operador '>' para atribuir valores em um array associativo. Este código cria um array associativo com as chaves 'Nome', 'Idade' e 'Cidade' associadas aos valores 'Ana', '52' e 'Manaus', respectivamente.

E) `$pessoa=array("Nome"->"Ana","Idade"->"52", "Cidade"->"Manaus");`

{ERRADO}: Este código está usando o operador de objeto (->) para tentar atribuir valores às chaves no array associativo, o que não é a sintaxe correta em PHP. Em PHP, usamos o operador '=' para atribuir valores em um array associativo.

#### 4.8. FCC - Analista de Tecnologia da Informação (SANASA)/Análise e Desenvolvimento/2019

Considere os comandos PHP abaixo, que objetivam criar um array.

I. `$servico = array("Água"=>"8,70", "Esgoto"=>"4,30");`  
{CERTO} Está usando a função `array()` para criar um array associativo

II. `$servico [ 'Água' ] = "8,70"; $servico [ 'Esgoto' ] = "4,30";`  
{CERTO} está criando um array associativo atribuindo valores diretamente às chaves 'Água' e 'Esgoto'.

III. `$servico = new array { 'Água' -> "8,70"; 'Esgoto' -> "4,30" };`  
{ERRADO} Está tentando criar uma nova instância de 'array' como um objeto, o que é inválido

IV. `$servico = array { [ "Água" ] [ "8,70" ]; [ "Esgoto" ] [ "4,30" ] };`  
{ERRADO} Está usando uma sintaxe inválida para a criação de um array.

Está correto o que consta APENAS em

- A
- III e IV.
- B
- I e III.
- C
- I e IV.
- D
- II e III.
- E
- I e II.

#### 4.9. FCC - Analista Judiciário (TRF 3ª Região)/Apoio Especializado/Informática/2019

Para concatenar a string "TRF" com a variável reg que contém o valor inteiro 3, em Java e PHP, utilizam-se, respectivamente,

A) `String("TRF")+reg` e `str("TRF").$reg`

{ERRADO}: Em Java, a forma correta de concatenar é `"TRF" + reg`. Em PHP, a sintaxe correta é `"TRF" . $reg`. Não existe a função 'str' em PHP para esse propósito.

B) `"TRF"+reg` e `"TRF".$reg`

{CERTO}: Em Java, a concatenação de strings e números pode ser realizada com o operador '+', então `"TRF" + reg` é correto. Em PHP, o operador '.' é usado para concatenação, portanto `"TRF" . $reg` é a forma correta.

C) `"TRF"+reg` e `"TRF"+$reg`

{ERRADO}: Em PHP, o operador '+' não é usado para concatenação de strings e números, mas sim o operador '.'. Portanto, `"TRF" + $reg` é incorreto.

D) `"TRF",reg` e `Concat("TRF",$reg)`

{ERRADO}: Ambas as formas estão incorretas. Em Java, a forma correta de concatenar é `"TRF" + reg`. Em PHP, a sintaxe correta é `"TRF" . $reg`. Não existe a função 'Concat' em PHP para esse propósito.

E) `String.Concat("TRF",reg)` e `"TRF"+$reg`

{ERRADO}: Em Java, a concatenação de strings pode ser feita diretamente com o operador '+', então `"TRF" + reg` é a forma correta. Em PHP, o operador '+' não é usado para concatenação de strings e números, mas sim o operador '.'. Portanto, `"TRF" + $reg` é incorreto.

## 5. Árvores, algoritmos de ordenação e busca

### 5.1. CESGRANRIO - Escriturário (BB)/Agente de Tecnologia/2023

**Ponto de Exclamação Atenção:** Esta é uma questão com gabarito preliminar.

Para entender como o algoritmo de busca binária se comporta, um estudante de computação resolveu inserir um comando `System.out.printf()` em um método chamado `busca`. Esse método, escrito em Java, realiza uma busca binária em um array de números inteiros, ordenados de forma ascendente. O objetivo do `printf` é exibir, no console, o valor de cada elemento do array visitado pelo algoritmo de busca binária.

Para testar o código que criou, o estudante escreveu o método `main` a seguir.

```
public class Main {  
  
    public static void main(String[] args) {  
        int lista[]={5,18,27,33,44,49,54,67,69,72,79,86,87,92};  
  
        // o array lista possui 14 elementos  
  
        busca(78, lista);  
    }  
  
    public static int busca(int val,int lista[]) {  
  
        // código relativo ao algoritmo de busca binária  
    }  
}
```

O que será exibido no console quando o método `main` for executado?

A busca binária é um algoritmo eficiente para encontrar um item específico em uma lista ordenada de itens. Ele funciona dividindo a lista pela metade repetidamente até que o item seja encontrado ou até que todas as possibilidades sejam esgotadas.

No caso desta questão, o array ordenado é: {5,18,27,33,44,49,54,67,69,72,79,86,87,92} e o número que estamos procurando é 78.

Vamos passo a passo:

- O algoritmo começa no meio do array. A lista tem 14 elementos, então o elemento do meio é o 54 (7º elemento). O comando `System.out.printf()` exibe "54".
- O número 78 é maior que 54, então a próxima busca é feita na metade superior do array. Agora estamos considerando a sublista {67,69,72,79,86,87,92}.
- Dentro dessa sublista, o elemento do meio é o 79 (4º elemento). O comando `System.out.printf()` exibe "79".
- O número 78 é menor que 79, então a próxima busca é feita na metade inferior da sublista, que é {67,69,72}.
- Dentro dessa sublista, o elemento do meio é o 69 (2º elemento). O comando `System.out.printf()` exibe "69".
- O número 78 é maior que 69, então a próxima busca é feita na metade superior da sublista, que é {72}.
- Nesta última sublista, o único elemento é 72. O comando `System.out.printf()` exibe "72".
- Como 78 é maior que 72 e não existem mais elementos na sublista, a busca termina. O número 78 não está na lista.

Portanto, o console exibirá: "54 79 69 72".

## 5.2. CESGRANRIO – Profissional de Nível Superior (ELETRONUCLEAR)/Analista de Sistemas/Aplicações e Segurança de TIC/2022 (e mais 1 concurso)

Seja uma função que realiza uma busca binária sobre um array de números inteiros ordenados. Não se sabe, em princípio, se os números estão ordenados ascendente ou decendentemente. O cabeçalho dessa função é o seguinte:

```
int busca (int [ ] vet, int elem)
```

Isto é, a função busca recebe um array de números inteiros (vet) e um número inteiro (elem) como parâmetros, e retorna um número inteiro. Caso exista em vet um inteiro igual a elem, a função retornará o índice desse inteiro no array; caso contrário, a função retornará -1.

O algoritmo de busca binária produz um índice (ind) a cada iteração sobre o array, tendo em vista comparar o elemento que se deseja procurar (elem) com o elemento vet [ ind ]. Isto é:

```
if ( vet [ ind ] == elem )  
  
    return ind;
```

No comando acima, diz-se que houve uma visita ao elemento vet [ ind ].

Admita que a função busca foi chamada por meio do comando a seguir:

```
int resp = busca (vet, 50);
```

Sabendo-se que os elementos visitados foram 54, 17, 33 e 50, nesta ordem, qual array foi passado como parâmetro para a função busca?

A questão trata da busca binária no vetor do enunciado. Nela, adota-se o conceito de dividir para conquistar. Isso quer dizer que o vetor é dividido várias vezes de forma a procurar pelo número que é pedido. Isso sempre é feito com o vetor ordenado. No caso da nossa questão, pede-se o número 50. Se dividirmos o número ao meio e acharmos da primeira vez o 50, paramos. Caso contrário, deve-se procurar abaixo ou acima, a depender do número encontrado.

Dito isso, vamos procurar conforme o enunciado da questão nos orienta: 54, 17, 33 e 50. O primeiro é 54. Com isso, devemos buscar o vetor em que a posição do meio é o 54. Vejamos:

a) [ 95, 90, 87, 54, 52, 50, 33, 17, 11, 10 ]

b) [ 5, 10, 11, 17, 33, 50, 54, 87, 90, 95 ]

c) [ 121, 111, 93, 87, 60, 54, 50, 33, 17, 5 ]

d) [ 5, 17, 33, 50, 54, 60, 87, 93, 111, 121 ]

{CERTO}

- Divide-se 10 por 2, sendo assim temos a posição 5 do 54.
- Divide de novo por 2 para a esquerda: 2,5. Vamos para a posição 2: 17.
- Como a posição 2 é 17 que é menor que 50, deve-se andar para a direita.
- Pronto, agora temos a sequência 54, 17, 33 e 50.

e) [ 130, 121, 111, 90, 70, 60, 54, 50, 33, 17 ]

### 5.3. CESGRANRIO – Profissional de Nível Superior (ELETRONUCLEAR)/Analista de Sistemas/Aplicações e Segurança de TIC/2022 (e mais 1 concurso)

Seja um array composto por 7 números inteiros.

[ 5, 15, 77, 21, 5, 25, 2 ]

Esse array foi usado por um profissional de teste de software para testar uma função que ordena, de forma ascendente, um array de números inteiros. Essa função implementa o algoritmo de ordenação por seleção.

Para avaliar a evolução do array sendo ordenado, o profissional de teste solicitou ao programador que criou a função de ordenação que fizesse uma modificação no código, de modo que o somatório dos elementos do array com índices 2, 3 e 4 seja exibido no console imediatamente antes do incremento da variável ( i ) que controla a execução do comando de repetição mais externo.

Feitas as modificações solicitadas, o código da função passou a ter a seguinte forma geral:

```
ordena (int vetor[ ]) {  
  
    int i = 0;  
  
    int tam = length (vetor); // comentário: a função length retorna a quantidade de  
    elementos de um array  
  
    while ( i < tam ) {  
  
        while ( ) {  
  
            // comentário: isso é apenas a forma geral do algoritmo de ordenação  
            // não é o código completo  
  
        }  
  
        print ( vetor[2] + vetor[3] + vetor[4] );  
  
        i = i + 1;  
  
    }  
  
}
```

O que será exibido no console pelo comando print na 3ª iteração do comando de repetição mais externo?

- A) 32
- B) 38
- C) 41
- D) 103
- E) 113

Vamos entender a lógica do Selection Sort e então aplicar ao array fornecido para entender o que ocorrerá em cada iteração.

O algoritmo de ordenação por seleção (Selection Sort) é um método simples que opera da seguinte maneira:

- Encontra o menor valor na lista.



- Troca a posição do menor valor encontrado com o valor da primeira posição.
- Repete estes passos para o restante da lista (os próximos elementos).

Agora, aplicando este algoritmo ao array fornecido:

Array inicial: [5, 15, 77, 21, 5, 25, 2]

- 1ª iteração: O menor valor é 2. Trocamos a posição do número 2 (última posição) com o número na primeira posição (5).
- Array após a 1ª iteração: [2, 15, 77, 21, 5, 25, 5]
- Somatório dos índices 2, 3 e 4:  $77 + 21 + 5 = 103$
- 2ª iteração: Ignoramos o primeiro elemento (pois já está na posição correta) e encontramos o menor valor no restante do array. O menor valor é 5. Trocamos a posição do número 5 (quinta posição) com o número na segunda posição (15).
- Array após a 2ª iteração: [2, 5, 77, 21, 15, 25, 5]
- Somatório dos índices 2, 3 e 4:  $77 + 21 + 15 = 113$
- 3ª iteração: Ignoramos os dois primeiros elementos e encontramos o menor valor no restante do array. O menor valor é 5. Trocamos a posição do número 5 (última posição) com o número na terceira posição (77).
- Array após a 3ª iteração: [2, 5, 5, 21, 15, 25, 77]

Somatório dos índices 2, 3 e 4:  $5 + 21 + 15 = 41$

Portanto, na terceira iteração do comando de repetição mais externo, o console exibirá 41. Assim, a resposta correta é a opção C) 41.

#### 5.4. CESGRANRIO – Técnico Científico (BASA)/Tecnologia da Informação/2021

Um determinado programador é responsável por tarefas de ordenação e, ao estudar determinados produtos, resolveu ordenar, de maneira crescente, a sequência [64, 34, 25, 12, 90, 11, 22] utilizando dois algoritmos, o Bubble Sort e o Select Sort, nessa ordem.

Ele iniciou o teste com o Bubble Sort, mas, na iteração em que a chave 64 atingiu a sua posição correta pela primeira vez, copiou a sequência alcançada nesse estágio e utilizou-a para continuar o trabalho com o algoritmo Select Sort.

A partir do momento em que o programador começa a utilizar o segundo algoritmo, quantas trocas de posições de chaves serão realizadas para atingir, pela primeira vez, a situação em que a sequência está ordenada?

- A) 1
- B) 2
- C) 3
- D) 4
- E) 5

Vamos começar entendendo os dois algoritmos:

**Bubble Sort:** Este algoritmo de ordenação compara cada par de elementos adjacentes de uma lista e os troca de lugar se estiverem em ordem errada. Este processo é repetido até que não sejam necessárias mais trocas, o que indica que a lista está ordenada.

**Selection Sort:** Este algoritmo de ordenação divide a lista em duas partes: a parte ordenada e a parte desordenada. Inicialmente, toda a lista é desordenada. O algoritmo procura o menor (ou maior, dependendo da ordem de classificação) elemento da parte desordenada e o troca com o primeiro elemento não ordenado.

Agora, vamos aplicar estes algoritmos à lista fornecida:

Lista inicial: [64, 34, 25, 12, 90, 11, 22]

**Bubble Sort:**

- 1ª iteração: [34, 25, 12, 64, 11, 22, 90]
- 2ª iteração: [25, 12, 34, 11, 22, 64, 90]
- 3ª iteração: [12, 25, 11, 22, 34, 64, 90]

Após a 3ª iteração, a chave 64 atinge sua posição correta. O programador, então, copia essa sequência e passa para o próximo algoritmo.

**Selection Sort** (começando da lista obtida após a execução parcial do Bubble Sort):

Lista inicial: [12, 25, 11, 22, 34, 64, 90]

- 1ª iteração (não há necessidade de troca, pois 12 já é o menor elemento): [12, 25, 11, 22, 34, 64, 90]
- 2ª iteração: [12, 11, 25, 22, 34, 64, 90]
- 3ª iteração: [12, 11, 22, 25, 34, 64, 90]
- 4ª iteração (não há necessidade de troca, pois 25 já está no lugar correto): [12, 11, 22, 25, 34, 64, 90]
- 5ª iteração (não há necessidade de troca, pois 34 já está no lugar correto): [12, 11, 22, 25, 34, 64, 90]

Neste ponto, a lista está ordenada. Portanto, apenas duas trocas de posições de chaves foram necessárias para ordenar a lista usando o algoritmo Selection Sort. Assim, a resposta correta é a opção B) 2.

## 5.5. CESGRANRIO – Escriturário (BB)/Agente de Tecnologia/2021

Desejam-se realizar buscas nas seguintes coleções de dados, representadas na linguagem Java:

I – Um array de 1.000 números inteiros ordenados de forma decrescente;

- Quando fazemos buscas em um vetor que não esteja ordenado, estamos diante da busca sequencial.
- Porém, quando o vetor está ordenado não faz sentido utilizar o método de busca sequencial. Nesse caso, é melhor utilizar a busca binária.
- Por isso que quando o array está ordenado vamos utilizar a busca binária. No nosso item temos um vetor com 1000 números e estão ordenados, então é certeza que vamos utilizar a busca binária.
- Mesmo que o array esteja ordenado de forma decrescente, podemos usar uma variação da Busca Binária que lida com arrays ordenados de maneira decrescente. Isto é, em vez de ir para a direita (valores maiores) no caso padrão, iríamos para a esquerda (valores menores), e vice-versa. Portanto, a Busca Binária é aplicável aqui.

II – Uma lista encadeada desordenada e alocada dinamicamente, cujos 1.000 nós contêm strings (uma string por nó);

- Nesse item, como temos um vetor desordenado, a melhor forma de se fazer buscas nele é utilizando a busca sequencial.
- A Busca Sequencial é a única opção viável aqui, pois a lista está desordenada. A Busca Binária necessita de uma lista ordenada para funcionar corretamente.

III – Uma lista encadeada, alocada dinamicamente, cujos 1.000 nós contêm números decimais (um número double por nó) ordenados de forma ascendente.

- Se um vetor está ordenado, seja de forma crescente ou decrescente, o método de busca mais indicado é a busca binária.

Levando-se em consideração a exequibilidade e a eficiência, quais métodos de busca devem ser empregados, respectivamente, em cada um dos três casos acima?

- A) I – sequencial; II – sequencial; III – binária
- B) I – binária; II – sequencial; III – sequencial
- C) I – binária; II – sequencial; III – binária
- D) I – sequencial; II – sequencial; III – sequencial
- E) I – sequencial; II – binária; III – binária

Portanto, a resposta correta para esta pergunta é a opção B) I – binária; II – sequencial; III – binária.

## 5.6. CESGRANRIO – Escriturário (BB)/Agente de Tecnologia/2021

As agências bancárias negociam seguros residenciais com seus clientes e, muitas vezes, precisam arquivar cópias de forma ordenada para que consultas eventuais sejam facilitadas. O gerente de uma agência precisava ordenar um vetor de documentos referentes a esses seguros, e o seu adjunto, da área de TI, o aconselhou a usar o algoritmo de ordenação chamado Bubble Sort.

Utilizando-se o algoritmo sugerido, qual será a quantidade de trocas de posições realizadas para ordenar, de modo crescente, o vetor de números de contrato (77, 51, 11, 37, 29, 13, 21)?

- A) 14
- B) 15
- C) 16
- D) 17
- E) 18

O algoritmo Bubble Sort é um algoritmo simples de ordenação que repetidamente percorre a lista a ser ordenada, compara cada par de itens adjacentes e os troca se estiverem na ordem errada. Este passo é repetido até que não seja mais necessário, o que indica que a lista está ordenada.

Aqui está a sequência inicial de números:  
[77, 51, 11, 37, 29, 13, 21]

Agora vamos acompanhar cada passo do algoritmo Bubble Sort:

1ª iteração:

- Compara 77 e 51.  $77 > 51$ , então troca. [51, 77, 11, 37, 29, 13, 21] (1 troca)
- Compara 77 e 11.  $77 > 11$ , então troca. [51, 11, 77, 37, 29, 13, 21] (2 trocas)
- Compara 77 e 37.  $77 > 37$ , então troca. [51, 11, 37, 77, 29, 13, 21] (3 trocas)
- Compara 77 e 29.  $77 > 29$ , então troca. [51, 11, 37, 29, 77, 13, 21] (4 trocas)
- Compara 77 e 13.  $77 > 13$ , então troca. [51, 11, 37, 29, 13, 77, 21] (5 trocas)
- Compara 77 e 21.  $77 > 21$ , então troca. [51, 11, 37, 29, 13, 21, 77] (6 trocas)

2ª iteração:

- Compara 51 e 11.  $51 > 11$ , então troca. [11, 51, 37, 29, 13, 21, 77] (7 trocas)
- Compara 51 e 37.  $51 > 37$ , então troca. [11, 37, 51, 29, 13, 21, 77] (8 trocas)
- Compara 51 e 29.  $51 > 29$ , então troca. [11, 37, 29, 51, 13, 21, 77] (9 trocas)
- Compara 51 e 13.  $51 > 13$ , então troca. [11, 37, 29, 13, 51, 21, 77] (10 trocas)
- Compara 51 e 21.  $51 > 21$ , então troca. [11, 37, 29, 13, 21, 51, 77] (11 trocas)

3ª iteração:

- Compara 37 e 11.  $37 > 11$ , então troca. [11, 37, 29, 13, 21, 51, 77] (12 trocas)
- Compara 37 e 29.  $37 > 29$ , então troca. [11, 29, 37, 13, 21, 51, 77] (13 trocas)
- Compara 37 e 13.  $37 > 13$ , então troca. [11, 29, 13, 37, 21, 51, 77] (14 trocas)
- Compara 37 e 21.  $37 > 21$ , então troca. [11, 29, 13, 21, 37, 51, 77] (15 trocas)

4ª iteração:

- Compara 29 e 11.  $29 > 11$ , então troca. [11, 29, 13, 21, 37, 51, 77] (16 trocas)
- Compara 29 e 13.  $29 > 13$ , então troca. [11, 13, 29, 21, 37, 51, 77] (17 trocas)

5ª iteração:

- Compara 21 e 11.  $21 > 11$ , então troca. [11, 13, 21, 29, 37, 51, 77] (18 trocas)

Então, para ordenar a lista de números de contrato, são necessárias 18 trocas utilizando o algoritmo Bubble Sort.

Portanto, a resposta correta é a alternativa E (18 trocas).

## 5.7. CESGRANRIO – Profissional Petrobras de Nível Superior (PETROBRAS)/Analista de Sistemas/Processos de Negócio/2018

Um programador construiu uma função para ordenar vetores de inteiros por meio do algoritmo de ordenação por inserção (insertion sort). A versão iterativa desse algoritmo possui dois loops aninhados. Suponha que esse programador tenha inserido, imediatamente antes do incremento da variável de controle do loop mais externo, uma chamada de uma função para percorrer e exibir o conteúdo do vetor que está sendo ordenado. O trecho de código a seguir ilustra como essa chamada é feita.

```
for (int i = 1; i < vetor.length; i++){  
  
    /* Os demais comandos que implementam o algoritmo  
    de ordenação por inserção devem substituir essas  
    linhas com comentários  
    */  
    exibeVetor(vetor);  
}
```

Abaixo vemos o vetor que foi passado como parâmetro em uma chamada da função de ordenação.

[78-12-35-1-17-4-43-11-17-1]

O que será exibido no console quando o valor da variável *i* for igual a 3?

- A) 1 1 11 12 17 4 17 35 43 78
- B) 1 12 4 17 11 17 1 35 43 78
- C) 1 1 4 78 17 35 43 11 17 12
- D) 1 12 35 78 17 4 43 11 17 1
- E) 1 1 4 11 17 35 43 78 17 12

O algoritmo de ordenação por inserção (insertion sort) trabalha ao dividir o array em duas partes: a parte ordenada e a parte desordenada. Inicialmente, a parte ordenada contém um único elemento, então ele pega o próximo elemento da parte desordenada e o insere na posição correta na parte ordenada, e este processo é repetido até que a parte desordenada esteja vazia.

Vamos acompanhar o progresso do algoritmo a cada iteração quando *i* é igual a 3.

Vetor inicial:

[78, 12, 35, 1, 17, 4, 43, 11, 17, 1]

- 1ª iteração (*i* = 1):
  - Compara 78 e 12. 78 > 12, então troca. [12, 78, 35, 1, 17, 4, 43, 11, 17, 1]
- 2ª iteração (*i* = 2):
  - Compara 78 e 35. 78 > 35, então troca. [12, 35, 78, 1, 17, 4, 43, 11, 17, 1]
- 3ª iteração (*i* = 3):
  - Compara 78 e 1. 78 > 1, então troca. [12, 35, 1, 78, 17, 4, 43, 11, 17, 1]
  - Compara 35 e 1. 35 > 1, então troca. [12, 1, 35, 78, 17, 4, 43, 11, 17, 1]
  - Compara 12 e 1. 12 > 1, então troca. [1, 12, 35, 78, 17, 4, 43, 11, 17, 1]

Então, quando *i* é igual a 3, o vetor exibido no console será [1, 12, 35, 78, 17, 4, 43, 11, 17, 1].

Portanto, a resposta correta é a alternativa D.

**5.8. CESGRANRIO – Profissional Petrobras de Nível Superior  
(PETROBRAS)/Analista de Sistemas/Processos de Negócio/2018**

Dada a sequência numérica (15,11,16,18,23,5,10,22,21,12) para ordenar pelo algoritmo Selection Sort, qual é a sequência parcialmente ordenada depois de completada a quinta passagem do algoritmo?

- A) [15, 11, 16, 18, 12, 5, 10, 21, 22, 23]
- B) [15, 11, 5, 10, 12, 16, 18, 21, 22, 23]
- C) [15, 11, 16, 10, 12, 5, 18, 21, 22, 23]
- D) [10, 11, 5, 12, 15, 16, 18, 21, 22, 23]
- E) [12, 11, 5, 10, 15, 16, 18, 21, 22, 23]

O método selection sort separa o array em parte ordenada (normalmente à direita) e parte desordenada (normalmente à esquerda). O método consiste em pegar o maior número da parte desordenada e colocar no início parte ordenada. Pode-se começar a partir do penúltimo valor.

Sendo assim:

Inicial: [15,11,16,18,23,5,10,22,21][12]

- 1a. passagem: [15,11,16,18,23,5,10,22,21][12,23]
- 2a. passagem: [15,11,16,18,5,10,22,21][12,22,23]
- 3a. passagem: [15,11,16,18,5,10,21][12,21,22,23]
- 4a. passagem: [15,11,16,18,5,10][12,18,21,22,23]
- 5a. passagem: [15,11,16,5,10][12,16,18,21,22,23]

Portanto, na 5a. passagem o vetor estará com a sequência [15,11,5,10,12,16,18,21,22,23], Letra B.

### 5.9. CESGRANRIO - Profissional Petrobras de Nível Superior (PETROBRAS)/Analista de Sistemas/Processos de Negócio/2018

A sequência de chaves 20 – 30 – 25 – 31 – 12 – 15 – 8 – 6 – 9 – 14 – 18 é organizada em uma árvore binária de busca. Em seguida, a árvore é percorrida em pré-ordem.

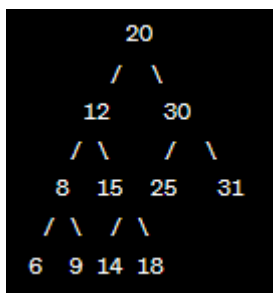
Qual é a sequência de nós visitados?

- A) 6 – 9 – 8 – 14 – 18 – 15 – 12 – 25 – 31 – 30 – 20
- B) 20 – 12 – 8 – 6 – 9 – 15 – 14 – 18 – 30 – 25 – 31
- C) 6 – 8 – 9 – 12 – 14 – 15 – 18 – 20 – 25 – 30 – 31
- D) 20 – 30 – 31 – 25 – 12 – 15 – 18 – 14 – 8 – 9 – 6
- E) 6 – 8 – 9 – 14 – 15 – 18 – 12 – 25 – 30 – 31 – 20

Para resolver essa questão, primeiro precisamos entender como a árvore binária de busca é construída. Uma árvore binária de busca tem a propriedade de que para cada nó, todos os elementos à esquerda são menores que o valor do nó, e todos os elementos à direita são maiores.

A sequência fornecida é 20 – 30 – 25 – 31 – 12 – 15 – 8 – 6 – 9 – 14 – 18.

A árvore seria construída da seguinte forma:



Em uma travessia em pré-ordem, visitamos primeiro a raiz, depois o lado esquerdo e por fim o lado direito. Portanto, a sequência de nós visitados seria:

20 – 12 – 8 – 6 – 9 – 15 – 14 – 18 – 30 – 25 – 31

Assim, a resposta correta é a opção B.

## 5.10. CESGRANRIO – Analista de Sistemas Júnior (TRANSPETRO)/Processos de Negócio/2018

Analise o algoritmo de ordenação que se segue.

```
def ordenar(dado):  
    for passnum in range(len(dado)-1,0,-1):  
        for i in range(passnum):  
            if dado[i]>dado[i+1]:  
                temp = dado[i]  
                dado[i] = dado[i+1]  
                dado[i+1] = temp  
dado = [16,18,15,37,13]  
ordenar(dado)  
print(dado)
```

Com o uso desse algoritmo, qual é a quantidade de trocas realizadas para ordenar a sequência dado?

- A) 4
- B) 5
- C) 6
- D) 7
- E) 8

O algoritmo fornecido é um exemplo de Bubble Sort, um algoritmo de ordenação que percorre repetidamente a lista, compara elementos adjacentes e os troca se estiverem na ordem errada.

A lista inicial é [16, 18, 15, 37, 13]. Agora, vamos percorrer o processo de ordenação:

- Na primeira passagem completa, comparamos 16 e 18 (sem troca), 18 e 15 (troca), 18 e 37 (sem troca), 37 e 13 (troca). Portanto, temos duas trocas. A lista agora é [16, 15, 18, 13, 37].
- Na segunda passagem completa, comparamos 16 e 15 (troca), 16 e 18 (sem troca), 16 e 13 (troca), 18 e 37 (sem troca). Temos mais duas trocas. A lista agora é [15, 16, 13, 18, 37].
- Na terceira passagem completa, comparamos 15 e 16 (sem troca), 16 e 13 (troca), 16 e 18 (sem troca). Temos uma troca. A lista agora é [15, 13, 16, 18, 37].
- Na quarta passagem completa, comparamos 15 e 13 (troca), 15 e 16 (sem troca). Temos uma troca. A lista agora é [13, 15, 16, 18, 37].

No total, realizamos  $2 + 2 + 1 + 1 = 6$  trocas para ordenar a lista.

Portanto, a resposta correta é C) 6.



## 5.11. CESGRANRIO – Escriturário (BB)/Agente Comercial/2018

Uma árvore binária cujos nós armazenam números inteiros pode ser representada na linguagem Python por uma lista com três elementos:

- o primeiro representa a informação armazenada no nó (número inteiro);
- o segundo é uma lista que representa a subárvore esquerda;
- o terceiro é uma lista que representa a subárvore direita.

As variáveis a seguir representam os nós de uma árvore binária construída segundo a estrutura acima descrita. Os nós n3, n4 e n6 são as folhas; n1, n2 e n5 são os nós intermediários; e n0 é o nó raiz.

```
n6=[4, [], []]
n5=[6, [], n6]
n2=[8, n5, []]
n3=[5, [], []]
n4=[9, [], []]
n1=[7, n3, n4]
n0=[3, n1, n2]
```

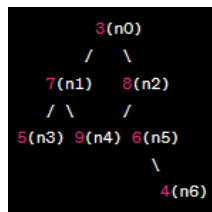
Seja o seguinte programa Python:

```
def lista(n):
    if n==[]:
        return
    lista(n[1])
    lista(n[2])
    print(n[0],end=' ')
```

O que será exibido no console quando ele for executado?

- A) 4 6 8 9 5 7 3
- B) 8 4 6 3 9 7 5
- C) 5 9 7 4 6 8 3
- D) 3 7 5 9 8 6 4
- E) 5 7 9 3 6 4 8

A função lista fornecida é um exemplo de percurso de árvore binária em pós-ordem, pois lê primeiro a posição n[1] (sub-árvore esquerda), depois a posição n[2] (sub-árvore direita) e, por último, a posição n[0] (nó).



- O nó raiz é n0, que tem valor 3. Ele tem dois filhos: n1 e n2.
- O nó n1 tem valor 7 e dois filhos: n3 e n4.
- Os nós n3 e n4 são folhas com valores 5 e 9, respectivamente.
- O nó n2 tem valor 8 e um filho: n5.
- O nó n5 tem valor 6 e um filho: n6.
- O nó n6 é uma folha com valor 4.

Dada a estrutura da árvore e a função, aqui está a sequência que a função lista imprimirá:

- Primeiro, ela irá para a subárvore esquerda de  $n_0$ , que é  $n_1$ , e recursivamente visitará sua subárvore esquerda ( $n_3$ ) e a direita ( $n_4$ ), imprimindo 5 e 9, respectivamente. Então, imprimirá o valor do nó  $n_1$ , que é 7.
- Depois, ela irá para a subárvore direita de  $n_0$ , que é  $n_2$ , e recursivamente visitará sua subárvore esquerda ( $n_5$ ) e a direita (vazia), imprimindo 4 e 6, respectivamente (da subárvore  $n_6$  de  $n_5$ ). Então, imprimirá o valor do nó  $n_2$ , que é 8.
- Finalmente, imprimirá o valor do nó raiz  $n_0$ , que é 3.
- Portanto, a sequência impressa será 5, 9, 7, 4, 6, 8, 3.

A resposta correta é C) 5 9 7 4 6 8 3.

## 5.12. CESGRANRIO – Escriturário (BB)/Agente Comercial/2018

O programa a seguir, em Python, implementa o algoritmo do método de bolha, imprimindo o resultado de cada passo.

```
def bolha(lista):
    for passo in range(len(lista)-1,0,-1):
        for i in range(passo):
            if lista[i]>lista[i+1]:
                lista[i],lista[i+1]=lista[i+1],lista[i]
        print(lista)
```

Qual será a quarta linha impressa para a chamada bolha([ 4, 3, 1, 9, 8, 7, 2, 5]) ?

- A) [3, 1, 4, 8, 7, 2, 5, 9]
- B) [1, 3, 4, 7, 2, 5, 8, 9]
- C) [1, 2, 3, 4, 5, 7, 8, 9]
- D) [1, 3, 2, 4, 5, 7, 8, 9]
- E) [1, 3, 4, 2, 5, 7, 8, 9]

- Primeira passagem (A maior passagem – 8 iterações)

- Comparamos 4 e 3, 4 é maior, então trocamos. A lista agora é [3, 4, 1, 9, 8, 7, 2, 5].
- Comparamos 4 e 1, 4 é maior, então trocamos. A lista agora é [3, 1, 4, 9, 8, 7, 2, 5].
- Comparamos 4 e 9, 4 é menor, então não trocamos.
- Comparamos 9 e 8, 9 é maior, então trocamos. A lista agora é [3, 1, 4, 8, 9, 7, 2, 5].
- Comparamos 9 e 7, 9 é maior, então trocamos. A lista agora é [3, 1, 4, 8, 7, 9, 2, 5].
- Comparamos 9 e 2, 9 é maior, então trocamos. A lista agora é [3, 1, 4, 8, 7, 2, 9, 5].
- Comparamos 9 e 5, 9 é maior, então trocamos. A lista agora é [3, 1, 4, 8, 7, 2, 5, 9].

- Segunda passagem (7 iterações)

- Comparamos 3 e 1, 3 é maior, então trocamos. A lista agora é [1, 3, 4, 8, 7, 2, 5, 9].
- Comparamos 3 e 4, 3 é menor, então não trocamos.
- Comparamos 4 e 8, 4 é menor, então não trocamos.
- Comparamos 8 e 7, 8 é maior, então trocamos. A lista agora é [1, 3, 4, 7, 8, 2, 5, 9].
- Comparamos 8 e 2, 8 é maior, então trocamos. A lista agora é [1, 3, 4, 7, 2, 8, 5, 9].
- Comparamos 8 e 5, 8 é maior, então trocamos. A lista agora é [1, 3, 4, 7, 2, 5, 8, 9].

- Terceira passagem (6 iterações)

- Comparamos 1 e 3, 1 é menor, então não trocamos.
- Comparamos 3 e 4, 3 é menor, então não trocamos.
- Comparamos 4 e 7, 4 é menor, então não trocamos.
- Comparamos 7 e 2, 7 é maior, então trocamos. A lista agora é [1, 3, 4, 2, 7, 5, 8, 9].
- Comparamos 7 e 5, 7 é maior, então trocamos. A lista agora é [1, 3, 4, 2, 5, 7, 8, 9].

- Quarta passagem (5 iterações)

- Comparamos 1 e 3, 1 é menor, então não trocamos.
- Comparamos 3 e 4, 3 é menor, então não trocamos.
- Comparamos 4 e 2, 4 é maior, então trocamos. A lista agora é [1, 3, 2, 4, 5, 7, 8, 9].
- Comparamos 4 e 5, 4 é menor, então não trocamos.

Na quarta passagem, o resultado final é [1, 3, 2, 4, 5, 7, 8, 9], que é a opção D da questão.