

Aula 00

*PEFOCE (Perito Criminal - Análise de
Sistemas) Engenharia de Software - 2021
(Pós-Edital)*

Autor:

**Diego Carvalho, Equipe
Informática e TI, Fernando
Pedrosa Lopes**

27 de Maio de 2021

Sumário

Paradigma Orientado a Objetos.....	2
1 – Conceitos Básicos	2
2 – Classes e Objetos.....	4
3 – Atributos	5
4 – Métodos.....	6
5 – Mensagens.....	7
6 – Abstração	8
7 – Abstração	9
8 – Encapsulamento.....	10
9 – Poliformismo.....	12
10 – Herança (Generalização/Especialização).....	15
Análise e Projeto	18
1 – Conceitos Básicos	18
Questões Comentadas.....	25
Lista de Questões	40
Gabarito	72



PARADIGMA ORIENTADO A OBJETOS

1 – Conceitos Básicos

Agora vamos falar das linguagens orientadas a objetos. Bem, um dos problemas da programação estruturada é que, muitas vezes, partes do código que servem apenas para tratar os dados se misturam com partes do código que tratam da lógica do algoritmo. Essa prática não é saudável, na medida em que diminui a reusabilidade e dificulta leitura, depuração e manutenção.

Galera... a modularização da programação estruturada foi um grande avanço na busca pela reusabilidade de código, **no entanto nem se compara ao que trouxe a programação orientada a objetos.** Esse novo paradigma reflete bem mais fielmente os problemas atuais. É um paradigma que se baseia na abstração de coisas ou objetos do mundo real em um sistema de forma potencialmente reusável!

*A reusabilidade de classes melhora a...? Agilidade! Permite que programas sejam escritos mais rapidamente. **Galera, vocês trabalham com isso e sabem que a demanda só aumenta, logo existe uma busca por maneiras de se desenvolver sistemas de forma mais rápida! Para quê reinventar a roda?*** Nada disso! Devemos aproveitar tudo que puder ser aproveitado.

Além disso, é importante citar a escalabilidade. Softwares construídos seguindo os preceitos da orientação a objetos são mais escaláveis, isto é, podem crescer facilmente sem aumentar demasiadamente sua complexidade ou comprometer seu desempenho. É possível construir tanto um software da Padaria do Joãozinho quanto o sistema que controla o Acelerador de Partículas da NASA.

Outra grande vantagem do paradigma é o seu caráter unificador, i.e., tratar todas as etapas do desenvolvimento de sistemas e ambientes sob uma única abordagem. Nesse sentido, podemos ter análise, projeto, modelagem, implementação, banco de dados, e ambientes orientados a objetos. Isso elimina as diferenças de paradigmas utilizadas em cada um desses contextos.

O Paradigma Orientado a Objetos visualiza um sistema de software como uma coleção de agentes interconectados chamados objetos. Cada objeto é responsável por realizar tarefas específicas e é pela interação entre objetos que uma tarefa computacional é realizada. *Galera, vocês percebem que a sociedade utiliza o conceito de objeto cotidianamente para resolver seus problemas?*





Pois é, já é algo natural! **Ele auxilia a modelagem de sistemas, reduzindo a diferença semântica entre a realidade sendo modelada e os modelos construídos.** Um sistema orientado a objetos consiste em objetos em colaboração com o objetivo de realizar as funcionalidades desse sistema. Cada objeto é responsável por tarefas específicas e a cooperação entre eles é importante para o desenvolvimento do sistema.

O Paradigma Orientado a Objetos tem papel importante no desenvolvimento de sistemas, pois **sua finalidade maior é de facilitar a vida dos programadores para que eles consigam desenvolver softwares que satisfaçam os clientes,** transformando coisas do dia a dia em objetos e permitindo que sejam empregados os seus recursos de forma eficaz.

O Paradigma Orientada a Objetos tem evoluído muito, principalmente em questões voltadas para segurança e reaproveitamento de código, requisitos estes considerados importantes no desenvolvimento de qualquer aplicação moderna. **Galera, no que diz respeito às vantagens da programação orientada a objetos, nós podemos destacar:**

- 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;

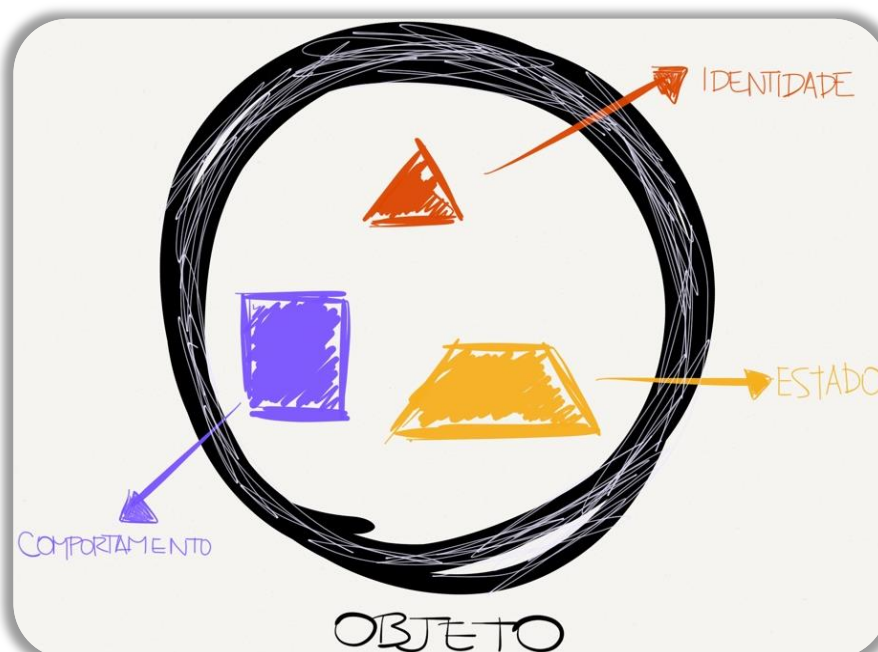
- O ciclo de vida do projeto de software moderno é frequentemente medido em semanas. A programação orientada a objetos ajuda nesses rápidos ciclos de desenvolvimento. Ela diminui o tempo do ciclo de desenvolvimento, fornecendo software confiável, reutilizável e facilmente extensível.

A Orientação a Objetos possui alguns princípios básicos ou pilares fundamentais, como mostra a imagem acima. São eles: Encapsulamento, Herança e Polimorfismo – veremos todos eles detalhadamente! No entanto, **alguns autores afirmam que esses princípios são todos simplesmente a aplicação de um único conceito: o princípio da abstração.**

2 – Classes e Objetos

Objetos são coisas (Carro, Foto, Bola, etc) e classes são um agrupamento de coisas. 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. Além disso, objetos são instâncias de classes. *O que é um carro?* Posso abstrair um carro como um objeto que tem motor, volante, porta, rodas, etc.

Ora, existem carros que têm dois motores (um elétrico e um à gasolina), outros têm somente duas portas. Como nós sabemos que ambos são carros? Porque, independentemente de pequenas diferenças entre as instâncias, nós conseguimos entender que se tratam de carros. Para fins de modelagem de um sistema, somente um subconjunto de características é relevante, portanto ignoramos o restante.



São componentes de um objeto: identidade, estado (propriedades) e comportamento (operações). A identidade é responsável por distinguir um objeto dos outros, i.e., eles são únicos, mesmo que sejam instâncias de uma mesma classe e que tenham os mesmos valores de variáveis. O estado reflete os valores correntes dos atributos do objeto em um determinado momento. *Entenderam esse conceito?*

Já o comportamento se refere a como os objetos reagem em relação a mudança de estado e troca de mensagens, i.e., é um conjunto de atividades externamente observáveis do objeto. *Galera, vamos resumir? Identidade é o que torna o objeto único; Estado se refere aos seus atributos; e Comportamento se refere aos seus métodos e procedimentos. Bacana?*

Um objeto é capaz de armazenar estados por meio de seus atributos e reagir a mensagens enviadas a ele, assim como se relacionar com outros objetos e enviar mensagens. Já a classe é como um projeto, formato ou descrição geral de um objeto! São abstrações do domínio do problema, não são diretamente suportadas em todas as linguagens, mas são necessárias em linguagens orientada a objeto.

3 – Atributos

Consiste em uma informação de estado para o qual cada objeto de uma classe tem seu próprio valor. Há dois tipos: atributos de objetos e de classes. O primeiro descreve valores mantidos em um objeto. Diferentes objetos de uma mesma classe não compartilham os atributos de objetos, i.e., cada um possui sua própria cópia do atributo. O segundo é aquele cujo valor todos os seus objetos devem compartilhar.

As mensagens enviadas a um objeto (isto é, a chamada de um método) podem mudar o valor de um ou mais atributos, alterando o estado de um objeto. Um atributo é um dado para o qual cada objeto tem seu próprio valor. **Atributos são, basicamente, a estrutura de dados que vai representar a classe.** Galera, não tem muito o que falar sobre esse tema não, é isso mesmo! Vamos ver os escopos de atributos:

- **Atributo de Classe:** similar a uma variável global, é 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.
- **Atributo de Instância:** é 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 default, é de instância; para ser de classe, deve vir precedida de static).



4 – Métodos

Similares a procedimentos e funções, consistem em descrições das operações que um objeto executa quando recebe uma mensagem. Há, portanto, uma correspondência um-para-um entre mensagens e métodos que são executados quando a mensagem é recebida através de um objeto. A mesma mensagem pode resultar em métodos diferentes quanto enviada para objetos diferentes.

Existe um método que nós devemos conhecer bem: Método Construtor! Os métodos construtores são métodos especiais, que são chamados automaticamente quando instâncias são criadas. Seu objetivo é garantir que o objeto será instanciado corretamente. Ele tem exatamente o mesmo nome da classe que está inserido, não possui tipo de retorno e não é obrigatório declará-lo. *Bacana?*

Através da criação de construtores, podemos garantir que o código que eles contêm será executado antes de qualquer outro código de outros métodos. Eles geralmente são usados para preparar um objeto, inicializando as variáveis do objeto. Pode existir mais de um método construtor em uma classe através da sobrecarga de construtores. Em algumas linguagens, são acionados por meio do operador New.

- **Método de Classe:** similar a um método global, é um método que realiza operações genéricas, i.e., 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, é um método que realiza operações específicas para um objeto e, não, à classe, i.e., são relativas a uma instância particular. Por default, todos os métodos de uma determinada classe são considerados métodos de instância.

Cabe salientar que um método nada mais é que uma definição, pois a ação em si só ocorre quando o objeto é invocado através de um método – por meio de uma mensagem! **Para quem lembra de programação estruturada, métodos são similares a procedimentos ou funções.** Eles definem o comportamento a ser exibido pelas instâncias da classe associada em tempo de execução.

Em tempo de execução, eles possuem acesso aos dados armazenados em um objeto que estão associados e são, desta forma, capazes de controlar o estado da instância. A associação entre classe e método é chamada de ligação (Binding). Um método associado a uma classe é dito ligado (Bound) à classe. Aqui nós podemos ver os tipos de ligação:

- **Early Binding (Ligação Prematura):** também conhecida como Ligação Estática, ocorre quando o método a ser invocado é definido em tempo de compilação.
- **Late Binding (Ligação Tardia):** também conhecida como Ligação Dinâmica, ocorre quando o método a ser invocado é definido em tempo de execução.



Aqueles que já conhecem um pouco sobre orientação a objetos, respondam-me: *Qual desses aí deve ser mais comum quando utilizamos Polimorfismo?* **Late Binding!**

5 – Mensagens

Um objeto sozinho geralmente é muito pouco útil! Por meio da interação entre objetos é que se torna possível obter uma grande funcionalidade ou um comportamento mais complexo. **Logo, assim como no mundo real, objetos estão sempre interagindo uns com os outros.** Bem, essa interação ocorre por meio de troca de mensagens. *Como assim, professor?*

É bom sempre pensar em nosso dia-a-dia! Imaginemos uma pessoa dirigindo um carro e pensemos em como modelar essa ação para o mundo orientado a objetos! Podemos imaginar um Objeto Pessoa enviando uma mensagem para o Objeto Carro, dizendo-lhe para acelerar, frear, trocar marcha, etc. Um objeto manda outro objeto realizar alguma operação enviando-o uma mensagem.



Pois bem, algumas vezes o objeto receptor necessita de algumas informações para realizar o que lhe foi requisitado! Por exemplo: Objeto Carro precisa saber quanto é para acelerar, que horas frear, para qual marcha trocar, etc e o Objeto Pessoa precisa informá-lo. *Como ele pode fazer isso?* **Ele envia a mensagem acompanhada de um conjunto de parâmetros ou argumentos que podem afetar as operações.**

Uma mensagem é composta por três componentes básicos: **objeto a quem a mensagem é endereçada; nome do método ou serviço que se deseja executar; e parâmetros necessários ao método (se existirem).** Logo, podemos dizer que mensagens são requisições enviadas de um objeto para outro com o intuito de receber algo em retorno por meio da execução de uma operação.

A natureza das operações realizadas para alcançar o resultado requerido é determinada pelo objeto receptor. Em suma, trata-se de um ciclo completo onde uma mensagem é enviada a um



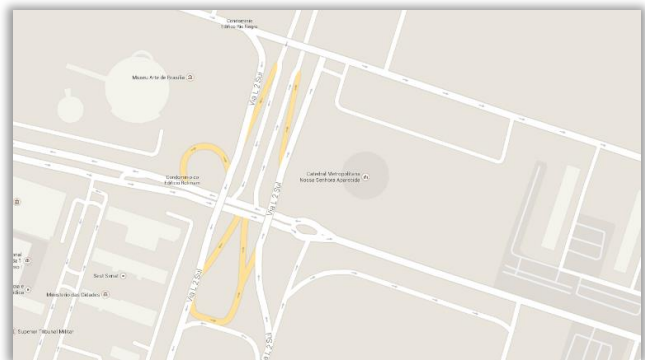
objeto, operações são executadas dentro desse objeto e uma mensagem contendo o resultado da operação é enviada ao objeto solicitante.

Em nosso paradigma, objetos vivem dando ordens para outros objetos executarem métodos (ou realizarem serviços). O bacana disso tudo é que objetos em processos distintos, máquinas distintas ou redes distintas podem comunicar-se através de mensagens. *E daí, professor?* E daí que isso aumenta a coesão e diminui o acoplamento, melhorando substancialmente a reusabilidade!

Algumas vezes, sistemas de software possuem tratadores (*handlers*) de mensagens! **Eles são responsáveis por processar mensagens de mais de um transmissor.** Em outras palavras, eles recebem mensagens de diversos objetos e encaminham para seus respectivos donos. Relaxem, nós veremos isso mais a frente com o conceito de polimorfismo! *Bacana? ;-)*

6 – Abstração

De modo simples e direto: abstração é a subtração de detalhes, i.e., quanto mais abstrato, há menos detalhes; e quanto menos abstrato, há mais detalhes. Observem as imagens abaixo: trata-se de uma visão aérea da Catedral de Brasília! Na primeira imagem, há muitos detalhes. **A segunda imagem é uma abstração em que se subtraiu diversos detalhes que não são relevantes para o domínio do problema.**



A Abstração é um processo mental pelo qual nós seres humanos nos atemos aos aspectos mais importantes (relevantes) de alguma coisa, ao mesmo tempo que ignoramos os aspectos menos importantes. **Esse processo mental nos permite gerenciar a complexidade de um objeto, ao mesmo tempo que concentramos nossa atenção nas características essenciais do mesmo.**

Note que uma abstração de algo é dependente da perspectiva (contexto) sobre a qual uma coisa é analisada: o que é importante em um contexto pode não ser importante em outro. **Temos que falar também sobre classes abstratas e concretas. Uma classe abstrata é desenvolvida para representar entidades e conceitos abstratos.** Ela é sempre uma superclasse que não possui instâncias.

Ela define um modelo (ou template) para uma funcionalidade e fornece uma implementação incompleta (i.e., 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.**

Por outro lado, as classes concretas implementam todos os seus métodos e permitem a criação de instâncias. Em suma, podemos afirmar que classes concretas são aquelas que podem ser instanciadas diretamente e classes abstratas são aquelas que não podem ser instanciadas diretamente. **Temos também métodos abstratos: aqueles para os quais não é definida uma forma de implementação específica¹.**

7 – Abstração

Galera, nós já vimos o que são classes abstratas! Resumindo: **são classes desenvolvidas para representar entidades e conceitos abstratos.** Nós vimos que elas geralmente contêm pelo menos um método abstrato (ou seja, sem corpo) e não se pode criar uma instância dela. As classes abstratas são usadas para serem herdadas e funcionam como uma superclasse.

Nós podemos dizer, então, que se trata de um contrato para que alguma subclasse concretize seus métodos. *Perfeito?* Disso tudo que eu afirmei acima, uma coisa é muito importante para entender a diferença entre uma interface e uma classe abstrata: "(...) *ela deve conter pelo menos um método abstrato (sem corpo)*". Imaginemos uma classe qualquer – temos três possibilidades:

1. se ela tem pelo menos um método abstrato, será obrigatoriamente abstrata;
2. se ela tem todos os métodos abstratos, será obrigatoriamente abstrata;
3. se ela tem todos os métodos concretos, poderá ser concreta ou abstrata.

No Caso 1, é fácil ver que a classe é abstrata se ela tem pelo menos um método abstrato; no Caso 2, também é fácil ver que se todos os métodos são abstratos, ela também deverá ser abstrata; no Caso 3, mesmo quando todos os métodos são concretos, ainda assim eu posso declará-la como abstrata – **isso é uma decisão do designer da classe.**

Agora voltemos um pouco para o Caso 2! Se todos os métodos são abstratos, a classe será abstrata! *Professor, uma interface é uma classe?* Não, uma interface é uma entidade em que todos os métodos são obrigatoriamente abstratos. *Opa, perceberam a semelhança?* Existem classes abstratas que contêm todos os métodos abstratos justamente como uma interface (Caso 2). Vejamos a diferença entre Classes Abstratas e Interfaces:

¹ É possível haver uma classe abstrata contendo somente métodos concretos. No entanto, se ela tiver um único método abstrato, que seja, deverá ser declarada como abstrata!



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 contém atributos, apenas assinatura de métodos.	Pode conter atributos.
CONSTRUTORES	Não contém construtores.	Contém construtores.
VELOCIDADE	Em geral, são mais lentas que classes abstratas.	Em geral, são mais rápidas que interfaces.

Portanto, uma Interface é similar a uma classe abstrata. Aliás, podemos dizer que uma interface é praticamente uma classe abstrata pura, i.e., 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.**

Uma aluna certa vez me perguntou a diferença entre ambas e quando se deve utilizar uma ou a outra. **Galera, lembre-se que a Classe Abstrata pode conter Métodos Concretos, então se você quer representar Métodos Concretos, você pode usar Classes Abstratas, mas não pode usar Interfaces** (que não pode conter métodos concretos). *Beleza?*

Há 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 utilizar 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.** Bacana? Lembrem-se dessa diferença e vocês não errarão em prova – apesar de que eu nunca vi isso cair em prova.

8 – Encapsulamento

Objetos possuem comportamentos, concordam? Já vimos que eles realizam operações em outros objetos, conforme recebam mensagens. 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.



Segundo o mecanismo do encapsulamento, o método que o objeto requisitado usa 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. *Como?* Por meio de uma interface! *O que é isso?* É tudo que o objeto sabe fazer, sem precisar informar como ele sabe fazer! *Vamos ver um exemplo concreto?*

Quando enviamos uma encomenda para alguém em outro país, eu pago pelo serviço de entrega internacional oferecido pelos Correios e só! Eu não quero saber se ele vai de avião, trem, navio, submarino! Como ele fará para entregar minha encomenda não me importa, o que importa é que ele entregue a encomenda. Portanto, a interface de um objeto deve definir os serviços que ele pode fornecer.

Através do encapsulamento, a única coisa que um objeto precisa saber para pedir a colaboração de outro objeto é conhecer a sua interface. Nada mais! 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.

Qual a vantagem disso? A interface permite que a implementação de uma operação pode ser trocada sem que o objeto requisitante da mesma precise ser alterado. **Isso mantém as partes de um sistema tão independentes quanto possível.** Daí a importância do mecanismo do encapsulamento no desenvolvimento de software orientado a objetos.

Pessoal, existe o conceito de especificadores ou modificadores de acesso. Conhecidos também como visão de método ou visão de atributo, definem a visibilidade de um atributo, método ou classe. Em geral, **utilizam-se especificadores de acesso para privar o acesso direto a atributos e obrigar o usuário a fazê-lo por meio de métodos públicos.**

MODIFICADOR/ESPECIFICADOR			CLASSE	PACOTE	SUBCLASSE	TODOS
UML	PÚBLICO	+	X	X	X	X
	PROTEGIDO	#	X		X	
	PACOTE	~	X	X		
	PRIVADO	-	X			

MODIFICADOR/ESPECIFICADOR			CLASSE	PACOTE	SUBCLASSE	TODOS
JAVA	PÚBLICO	+	X	X	X	X
	PROTEGIDO	#	X	X	X	
	DEFAULT	~	X	X		



9 – Poliformismo

O Polimorfismo indica a capacidade de abstrair várias implementações diferentes em uma única interface. **Imaginem que vocês tenham um videocassete antigo e, em determinado dia, ele decida parar de funcionar!** Você o joga no lixo, mas mantém seu controle remoto. Meses depois, compra um blu-ray da mesma marca e, de repente, seu controle remoto antigo funciona também no aparelho novo!

Olha que maneiro! Dois objetos, um novo e um antigo, respondem à mesma mensagem! *E no mundo da orientação a objetos?* Nesse contexto, o polimorfismo diz respeito à capacidade de duas ou mais classes de objetos responderem à mesma mensagem, cada qual de seu próprio modo. **Pensem em uma coleção de formas geométricas que contenha círculos, retângulos e outras formas específicas.**

Seguindo os princípios de polimorfismo na orientação a objetos, nós podemos facilmente calcular a área de todas essas figuras geométricas! No entanto, vocês sabem que o cálculo da área de círculo é diferente do cálculo da área de um retângulo, que é diferente do cálculo da área de um trapézio, que é diferente da área de um triângulo, etc. *Vocês concordam até aqui?*

Pois é... vamos relembrar um pouquinho de matemática? Para calcular a área do círculo, é necessário saber o raio; do retângulo, é necessário saber a base e a altura; do trapézio, é necessário saber a base maior, base menor e altura; e do triângulo, é necessário saber a base e altura. **Ora, para aplicar o polimorfismo, eu devo enviar a mesma mensagem e ele se virar para entender qual área ele deve calcular!**

E como ele faz isso, professor? Ele vê a lista de parâmetros, i.e., a mensagem enviada! Se eu envie um argumento, ele sabe que é o círculo; se eu envie três argumentos, ele sabe que é o trapézio; se eu envie dois argumentos, pode ser o retângulo ou o triângulo! *E agora professor?* **Aí temos outro tipo de polimorfismo, que é tratado em tempo de execução.**

Em outras palavras, 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.

O Polimorfismo pode ser Estático ou Dinâmico. O primeiro é 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

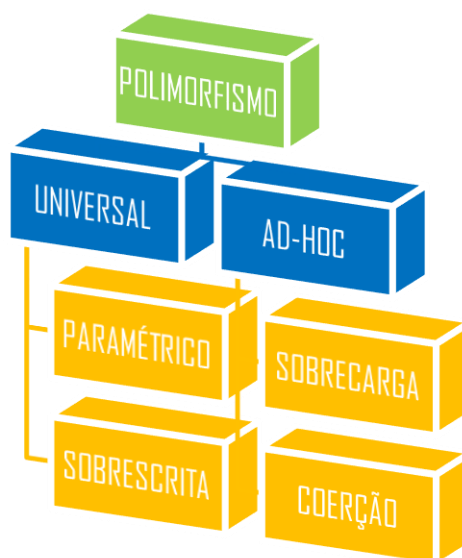


de acordo com os argumentos passados. *Professor, o que você quer dizer com parâmetros diferentes?*

Pode ser uma diferença na quantidade, tipo ou ordem dos parâmetros. O segundo é também conhecido como polimorfismo por sobrescrita, polimorfismo por inclusão, polimorfismo por herança, polimorfismo por subtipo, redefinição ou *overriding*. Ele 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. Alguns alunos sempre me perguntam o que é a assinatura de um método! Pois bem, vejamos: **dois métodos possuem a mesma assinatura se, e somente se, tiverem o mesmo nome e os mesmos parâmetros (quantidade, tipo e ordem dos parâmetros).**

Algumas linguagens (Ex: Java) ignoram o tipo de retorno para verificar se métodos possuem a mesma assinatura; outras linguagens (Ex: C++) validam também o tipo de retorno para verificar se métodos possuem a mesma assinatura. **Portanto, a questão do tipo de retorno não é pacífica, porque depende da linguagem de programação em questão.**



Galera, existe também uma outra classificação que cai em prova de veeeeez em quando! O **Polimorfismo pode ser classificado em: Universal e Ad-hoc!** Grosso modo, o Polimorfismo Universal pode trabalhar com um número infinito de tipos; e o Polimorfismo Ad-hoc pode trabalhar com um número finito de tipos. Dentro de cada uma dessas duas categorias, existem mais duas categorias!

Considerado como o polimorfismo verdadeiro, o polimorfismo paramétrico é aquele que permite que se escreva um código genérico para servir os subtipos (que só serão descobertos em tempo de execução). **Em outras palavras, um mesmo objeto pode ser utilizado como parâmetro em diferentes contextos sem necessidade de quaisquer alterações.**



Ao utilizar o polimorfismo paramétrico, um elemento (função, classe, método ou tipo de dado) pode ser escrito genericamente para que possa suportar valores idênticos sem depender de seu tipo. Como assim, professor? Vamos imaginar comigo que você tem listas de diversos tipos – você tem uma lista de carros, uma lista de pessoas, uma lista de animais e uma lista de filmes.

Ok! E você sabe que pode realizar várias operações nessas listas, tais como: acessar um elemento da lista, adicionar um elemento na lista, excluir um elemento da lista, atualizar um elemento da lista. **Perceba que essas operações servem para qualquer lista, independente de seu tipo.** Logo, por que criar métodos de acesso, adição, exclusão e atualização para todas as listas em vez de criar apenas um de cada?

Veremos que não há necessidade disso! **Uma única função é codificada e ela trabalhará uniformemente em um intervalo de tipos (funções paramétricas também são chamadas de funções genéricas).** Bem, no Java, isso começou na versão 1.5 (com Generics). Lá, existe um tipo genérico chamado <List>. Toda vez que eu preciso instanciar uma lista, basta fazer:

```
List<TipoDaLista> NomeDaLista = new ArrayList<List>();
```

Dessa forma, **caso eu queira instanciar uma lista de Strings, Integer ou um tipo criado por mim**, basta fazer conforme o código a seguir:

```
List<String> listaDeString = new ArrayList<List>();
```

```
List<Integer> listaDeInteger = new ArrayList<List>();
```

```
List<TipoMeuQualquer> listaDeTipoMeuQualquer = new ArrayList<List>();
```

E olha o mais legal! Eu não preciso implementar todos os métodos de uma lista para cada tipo específico, basta fazer (por exemplo, para adição):

```
listaDeString.add("Primeira String");
```

```
listaDeInteger.add(4.000);
```

```
listaDeTipoMeuQualquer.add(TipoMeuQualquer);
```

Diga se isso não é genial e uma mão na roda! Pois é! Bem, o segundo tipo de polimorfismo universal, nós já vimos acima – **a única novidade é que ele é chamado também de polimorfismo por inclusão, polimorfismo por herança ou polimorfismo por subtipo.** Já o



Polimorfismo ad-hoc se divide em sobrecarga e coerção – alguns autores mais rigorosos afirmam que polimorfismo ad-hoc não é polimorfismo.

Por que? Porque não ocorrem em tempo de execução! **Vamos lá... nós já vimos o primeiro tipo anteriormente, então não cabe repetir.** Nosso interesse aqui é no polimorfismo de coerção. Ele é suportado através da sobrecarga de operadores, ou seja, ocorre quando se converte um elemento de um tipo no tipo apropriado para o método (é o famoso *casting* implícito).

Ele permite que um argumento seja convertido para o tipo esperado por uma função, evitando assim um erro de tipo. Imaginem uma variável do tipo inteiro e uma variável do tipo real. É possível atribuir um valor inteiro a um tipo real (visto que ele é “maior”) de forma implícita. **Nesse momento, ocorre uma coerção (também chamada conversão) de uma variável de um tipo em outro tipo.**

Agora vejam que bacana! Já recebi uma dúvida algumas vezes: *Professor, um método pode sobrecarregar um método herdado?* Em nossa página no Facebook, nós já discutimos sobre essa dúvida! Vejamos dois cenários: primeiro, **Classe Veículo possui um método dirigir(a) e uma Classe Carro (filha de Classe Veículo) possui um método dirigir(a), logo sobrescrevendo o método da classe-pai.**

Se eu inserir um método dirigir(a,b) na classe-filha, eu posso afirmar que esse método sobrecarrega o método dirigir(a) da classe-filha, mas não da classe-pai. E ele não sobrescreve o método da classe-pai, porque as assinaturas são diferentes. **É impossível que um método realize sobrescrita e sobrecarga simultaneamente sobre um mesmo método.**

Segundo cenário: imaginem que eu não tenho uma sobrescrita de dirigir(a) na classe-filha. Eu tenho apenas dirigir(a) na classe-pai e dirigir(a,b) na classe-filha. *Posso afirmar que esse método da classe-filha sobrecarrega dirigir(a)?* Sim, **porque dirigir(a) é herdado na classe-filha, logo – de certo modo – há uma sobrecarga do método implícito herdado da classe-pai.** No Java, há a referência:

In a subclass, you can overload the methods inherited from the superclass. Such overloaded methods neither hide nor override the superclass instance methods—they are new methods, unique to the subclasse.

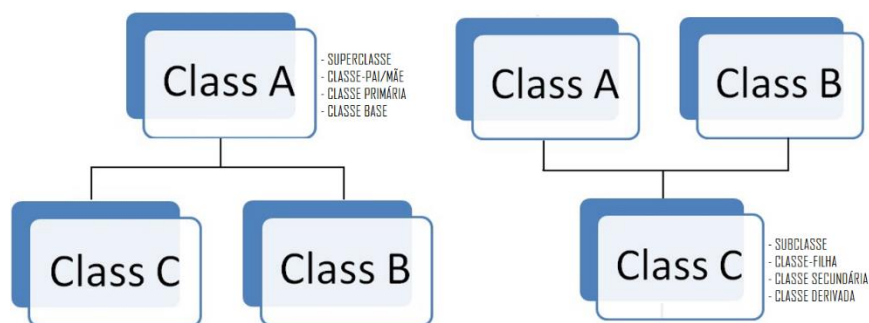
10 – Herança (Generalização/Especialização)

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. Ademais, essa classe pode definir características e comportamento particulares. Dessa forma, uma classe pode ser criada a partir do reuso 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; quando uma subclasse herda diretamente de apenas uma superclasse, trata-se de Herança Simples.**

*Em Orientação a Objetos, é permitido Herança Múltipla? Pegadinha mais clássica das provas de Tecnologia da Informação. Claro que sim, galera! **Não confundam isso jamais: algumas linguagens (Ex: Java e C#) não permitem Herança Múltipla, no entanto nós estamos falando sobre o paradigma e não sobre linguagens!** Professor, porque algumas linguagens não permitem Herança Múltipla?*

Excelente pergunta! **Em geral, é porque ela pode causar problemas de ambiguidade.** Em outras palavras, quando superclasses possuem membros homônimos e a subclasse não redefine esses membros, no momento em que um objeto da subclasse tentar referenciar diretamente o membro homônimo das superclasses, o compilador não saberá a qual membro ele está se referindo.

Um detalhe importante: uma subclasse sempre herdar métodos/atributos de suas superclasses – não importa se é Herança Simples ou Herança Múltipla. Você não pode dizer que você é um primata, mas não é um mamífero, porque todos os primatas são mamíferos. Vamos aproveitar para definir alguns conceitos importantes.



Herdar é diferente de acessar! Se uma classe estende a outra, ela sempre herdará seus métodos/atributos. Não importa, por exemplo, se eles são privados - a subclasse sempre os herdará, mesmo que não os acesse! *Como eu gosto de visualizar isso?* Imaginem que um tio-avô distante deixe um cofre entupido de dinheiro para vocês.

No entanto, esse cofre é completamente indestrutível e ele não deixou nenhuma senha. Nesse caso, vocês herdaram todo o dinheiro, mas não podem acessá-lo. No mundo orientado a objetos acontece a mesma coisa: se uma subclasse é filha de uma superclasse, ela herdará absolutamente tudo, mesmo que ela não consiga acessar (que é o caso de membros privados). *Fechado?*

Já caiu em prova discursiva: *qual a diferença entre Polimorfismo e Herança?* **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.

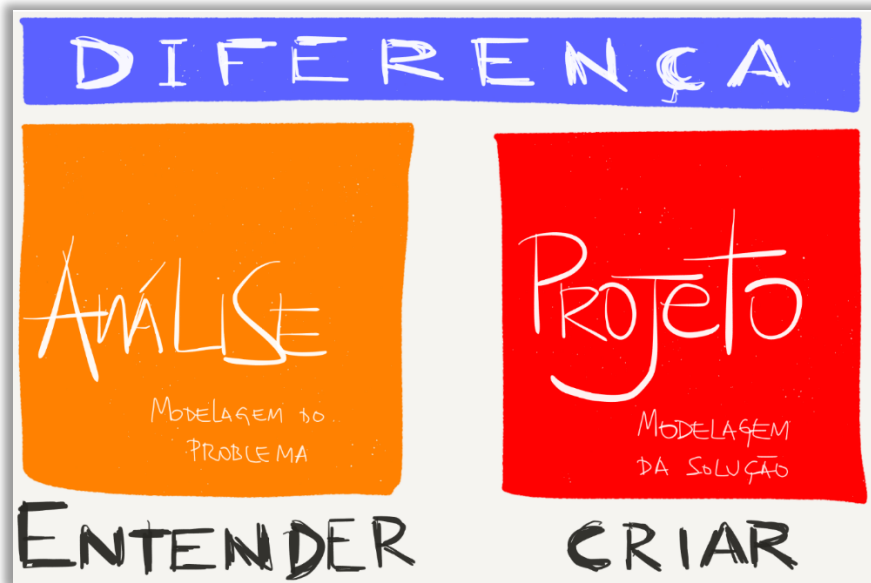
Quando uma classe herda de outra, todos os métodos e atributos que surgem na interface da classe previamente existente aparecerão automaticamente na interface da nova classe. Já o polimorfismo permite que um único nome de método represente um código diferente, selecionado por algum mecanismo automático.

Assim, um nome pode assumir muitas formas e, como pode representar código diferente, o mesmo nome pode representar muitos comportamentos diferentes. **Basicamente foi esse o padrão de resposta esperado pela banca examinadora.** *Bacana?* Pessoal, agora vamos ver algumas questões para exercitar tudo isso que nós vimos...

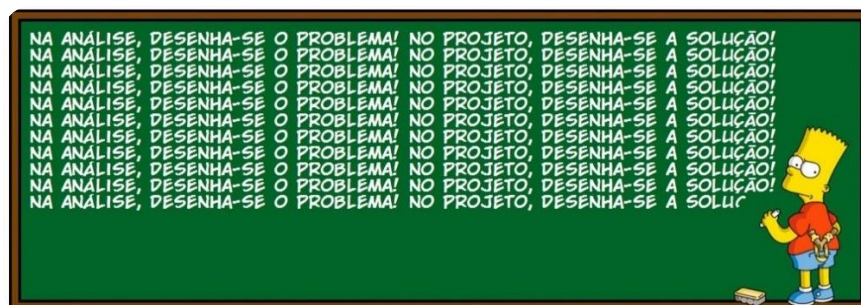


ANÁLISE E PROJETO

1 – Conceitos Básicos



Bem, vamos começar a distinguir o que é Análise e o que é Projeto. Para tal, **preciso que vocês memorizem, decorem, tatuem, componham uma música com a frase:**



Pessoal, a Análise consiste em atividades necessárias para entender o domínio do problema, i.e., o que deve ser feito. É uma atividade de investigação, **com foco no cliente**. Já o Projeto consiste em atividades necessárias para entender o domínio da solução do problema, i.e., como deve ser feito. É uma atividade técnica, **com foco no programador**.

Na Análise, a tecnologia de implementação e os requisitos não-funcionais não são modelados. Essa é uma tarefa do Projeto! Também não se pensa sobre soluções técnicas, pensa-se apenas em modelar funções, dados e relacionamentos do sistema. O modelo de análise deve ser aprovado pelo cliente – pode incluir até pequenas discussões sobre a solução, como sobre Interfaces Gráficas.



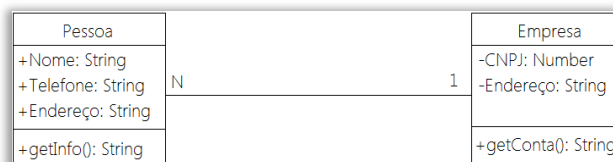
O Modelo de Casos de Uso representa o aspecto funcional de um domínio de negócio e, de forma similar, **o Modelo de Classes representa o aspecto estrutural de um domínio de negócio**. Como é? Pessoal, um exemplo de Modelo de Classes é o Diagrama de Classes (da UML). Ele representa visualmente conceitos de um determinado domínio por meio de classes.

É importante notar que o modelo de classes é utilizado durante a maior parte do desenvolvimento de um sistema orientado a objetos. Além disso, ele evolui durante as iterações do desenvolvimento do sistema. **À medida que o sistema é desenvolvido, o Modelo de Classes é incrementado com novos detalhes**. Há três estágios sucessivos de abstração: análise, especificação e implementação.

- **Modelo de Classes de Análise (ou Domínio):** construído durante a atividade de análise, representa as classes de domínio do negócio. Não leva em consideração restrições inerentes à tecnologia a ser utilizada na solução de um problema.



- **Modelo de Classes de Especificação (ou Projeto):** construído durante a atividade de projeto, estende o modelo de classes de análise e contém detalhes específicos inerentes à solução de software escolhida.



- **Modelo de Classes de Implementação:** construído durante a atividade de implementação, estende o modelo de classes de projeto e contém detalhes específicos inerentes ao desenvolvimento das classes em alguma linguagem.

```
1 private class Pessoa {
2     public String Nome;
3     public String Telefone;
4     public String Endereco;
5     private Empresa empresa[];
6
7     public String getInfo() {
8         return empresa.getConta();
9     }
10
11 public class Empresa {
12     public Number CNPJ;
13     public String Endereco;
14
15     public getConta() {
16         return 0;
17     }
18 }
```

À medida que o sistema é desenvolvido, o modelo é incrementado com novos detalhes. O Modelo de Análise enfatiza o desenho lógico, a visão externa, conceitual, abstrata, caixa-preta,



de alto nível de abstração. O Modelo de Projeto enfatiza o desenho físico, a visão interna, de implementação, concreta, caixa-branca, de baixo nível de abstração.

Sabe-se que o **Modelo de Análise é mais estável que o Modelo de Projeto**. Sob a perspectiva organizacional, o primeiro sofre bem menos com mudanças tecnológicas, externas, regulatórias, legais, etc. Além disso, é importante salientar que, na prática, muitas pessoas não fazem Análise e já partem para o Projeto. Inclusive, ela é uma disciplina opcional no RUP!

Professor, eu posso não fazê-la? Pode, sim. *Então para que ela existe?* Imaginem se ocorre uma revolução tecnológica e todas as organizações comecem a implementar sistemas com um novo paradigma. **Com um modelo de análise pronto, torna-se muito mais fácil adaptar o modelo de projeto**. Caso contrário, tem-se que refazer o modelo de projeto do início.

Em geral, as classes de análise evoluem para classes de projeto. A meta da Análise é identificar um esboço preliminar do comportamento do sistema. A meta do Projeto é transformar esse esboço preliminar em um conjunto implementável. O resultado é que há um refinamento detalhado e preciso quando alguém se move da Análise para o Design. Pessoal, a Análise apresenta quatro atividades principais:



Em 1992, Ivar Jacobson (aquele da UML) propôs uma técnica chamada Análise de Robustez, que propunha a categorização das classes de acordo com sua responsabilidade: **Classe de Fronteira; Classe de Controle; e Classe de Entidade**.



- **Classe de Fronteira**: classe utilizada para modelar a interação entre um ator e o sistema. Para cada ator, é identificada pelo menos uma classe de fronteira para permitir sua interação com o sistema. Então, uma classe de fronteira existe para que o sistema se comunique com o mundo exterior, logo elas são altamente dependentes do ambiente.

A Interação entre sistema e atores envolve transformar e converter eventos, bem como observar mudanças na apresentação do sistema (como a interface). As classes de fronteira modelam as partes do sistema que dependem do ambiente. As classes de entidade e de controle modelam as partes que são independentes de fatores externos ao sistema.

Portanto, alterar a GUI ou o protocolo de comunicação significa alterar somente as classes de fronteira, e não as classes de entidade e de controle. As classes de fronteira também facilitam a compreensão do sistema, pois definem suas fronteiras. Elas ajudam no design, fornecendo um bom ponto de partida para identificar serviços relacionados.



Algumas classes de fronteira comuns são janelas, protocolos de comunicação, interfaces de impressora, sensores e terminais. Se você estiver usando um construtor GUI, não será necessário modelar partes da interface de rotinas (botões, por exemplo) como classes de fronteira separadas. Geralmente, a janela inteira é o objeto de fronteira mais refinado.

As classes de fronteira também são úteis para capturar interfaces para APIs possivelmente não orientadas a objetos (como código mais antigo, por exemplo). Você deve modelar as classes de fronteira de acordo com o tipo de fronteira que elas representam. A comunicação com outro sistema e a comunicação com um ator humano (através de uma interface do usuário) têm objetivos diferentes.

Durante a modelagem da interface do usuário, a principal preocupação deve ser a forma como a interface será apresentada ao usuário. Durante a modelagem da comunicação do sistema, a principal preocupação deve ser o protocolo de comunicação. **Um objeto de fronteira poderá durar mais que uma instância de caso de uso, porém costumam ter a mesma duração da instância de caso de uso.**

- **Classe de Controle:** classe utilizada para controlar a lógica de execução ou negócio correspondente a cada caso de uso. Servem como uma ponte de comunicação entre objetos de fronteira e objetos de entidade. Decidem o que o sistema deve fazer quando um evento externo relevante ocorre, agindo como coordenador para a realização de casos de uso.

Como objetos de controle (instâncias de classes de controle) geralmente controlam outros objetos, o comportamento de objetos de controle é do tipo coordenador. **As classes de controle encapsulam um comportamento específico de caso de uso.** O comportamento de um objeto de controle está estreitamente relacionado à realização de um caso de uso específico.

Em muitos cenários, é possível até dizer que os objetos de controle "executam" as realizações de casos de uso. Entretanto, se as tarefas de caso de uso estiverem intrinsecamente relacionadas, alguns objetos de controle poderão participar de mais de uma realização de casos de uso. Além disso, vários objetos de controle de diferentes classes de controle podem participar de um único caso de uso.

Nem todos os casos de uso exigem um objeto de controle. Se o fluxo de eventos em um caso de uso estiver relacionado a um objeto de entidade, um objeto de fronteira poderá realizar o caso de uso em cooperação com o objeto de entidade. **As classes de controle podem ajudar a entender o sistema, pois representam a dinâmica do sistema, controlando as principais tarefas e os fluxos de controle.**

Quando o sistema executar o caso de uso, um objeto de controle normalmente será criado. Os objetos de controle geralmente desaparecem após a execução do correspondente caso de uso. **Observe que uma classe de controle não controla tudo o que é necessário em um caso de uso.** Em vez disso, ela coordena as atividades de outros objetos que implementam a funcionalidade.



- **Classe de Entidade:** classe utilizada para armazenar a informação que é manipulada ou processada pelo caso de uso, partindo do domínio do negócio. Geralmente, essas classes armazenam informações persistentes. Há várias instâncias ou objetos de uma mesma classe de entidade coexistindo dentro do sistema.

Os objetos de entidade (instâncias de classes de entidade) são usados para manter e atualizar informações sobre alguns fenômenos, como um evento, uma pessoa ou algum objeto real. Esses objetos geralmente são persistentes, precisando de atributos e relacionamentos durante muito tempo, algumas vezes durante todo o ciclo de vida do sistema.

Um objeto de entidade geralmente não é específico para uma realização de casos de uso. Às vezes, um objeto de entidade não é nem mesmo específico para o próprio sistema. Os valores de seus atributos e relacionamentos costumam ser fornecidos por um ator. Um objeto de entidade também pode ajudar a executar tarefas internas do sistema.

Seu comportamento pode ser tão complicado quanto o de outros estereótipos de objeto. No entanto, ao contrário de outros objetos, esse comportamento está intrinsecamente relacionado ao fenômeno que o objeto de entidade representa. Os objetos de entidade independem do ambiente (os atores). Os objetos de entidade representam os conceitos-chave do sistema que está sendo desenvolvido.

Exemplos típicos de classes de entidade em um sistema bancário são Conta e Cliente. Em um sistema de gerenciamento de redes, os exemplos são Nó e Link. Se o fenômeno que você deseja modelar não for usado por outras classes, será possível modelá-lo como um atributo de uma classe de entidade ou mesmo como um relacionamento entre classes de entidade.

Por outro lado, se o fenômeno for usado por qualquer outra classe do modelo de design, será preciso modelá-lo como uma classe. As classes de entidade fornecem um outro ponto de vista do sistema, pois mostram a estrutura lógica dos dados, que pode ajudá-lo a compreender o que o sistema deve oferecer aos usuários. *Ficou mais tranquilo de entender? Vamos para frente...*

A segunda atividade se refere à identificação de responsabilidades. Essa identificação e categorização implica que cada classe seja especialista em realizar uma tarefa específica que comporá o modelo de análise: comunicar-se com atores (Fronteira); manter as informações do sistema (Entidade); e coordenar a realização de um caso de uso (Controle).

As atividades seguintes são bastante intuitivas e fáceis de entender. A terceira atividade se refere à identificação de atributos, em que se busca apenas descobrir quais são os atributos, sem nenhuma preocupação sobre qual seu tipo (String, Date, Time, Integer, etc). Um bom conhecimento do domínio do problema é extremamente útil nesta fase.

Por fim, identificam-se os relacionamentos como associações, agregações, composições, dependências, generalizações, especializações, entre outros. Fim da Análise, agora nós sabemos



o que fazer, chegou a hora de saber como fazer. **Partamos, então, para o Projeto! Vamos falar bastante sobre arquitetura de software, estrutura e comportamento de classes.**

A Arquitetura de Software é a **organização ou a estrutura dos componentes significativos do sistema que interagem por meio de interfaces**. Uma arquitetura bem projetada e solidamente desenhada deve ser capaz de atender aos requisitos funcionais e não-funcionais do sistema e ser suficientemente flexível para suportar requisitos voláteis.

A arquitetura é importante, pois permite uma comunicação efetiva entre as partes interessadas, abrangendo a compreensão, negociação e consenso. Ademais, permite decisões tempestivas, i.e., **possibilita correção e validação do sistema antes da implementação**. Por fim, permite uma reutilizável em sistemas com características similares.

Uma boa arquitetura deve ter componentes projetados com baixo acoplamento e alta coesão. *Como é isso, professor?* Pessoal, esse é outro mantra que eu preciso que vocês memorizem! Acoplamento trata do nível de dependência entre módulos de um software. Já a Coesão trata do nível de responsabilidade de um módulo em relação a outros.

Professor, por que é bom ter baixo acoplamento? Porque se os **módulos pouco dependem um do outro, modificações de um não afetam os outros, além de prejudicar o reúso**. *Professor, por que é bom ter alta coesão?* Porque se os módulos têm responsabilidades claramente definidas, eles serão altamente reusáveis independentes e simples de entender.

Uma forma de organizar a arquitetura de um sistema complexo em partes menores é por meio de camadas, em que cada uma corresponderá a um conjunto de funcionalidades de um sistema— sendo que as funcionalidades de alto nível dependerão das funcionalidades de baixo nível. A separação em camadas fornece um nível de abstração através do agrupamento lógico de subsistemas relacionados.

Parte-se do princípio de que as camadas de abstração mais altas devem depender das camadas de abstração mais baixas. Isso permite que o sistema de software seja mais portátil e modificável. **É importante salientar que mudanças em uma camada mais baixa, que não afetem a sua interface, não implicarão mudanças nas camadas mais superiores.**

Assim como mudanças em uma camada mais alta, que não impliquem a criação de um novo serviço em uma camada mais baixa, não afetarão as camadas mais inferiores. **A arquitetura em camadas permite melhor separação de responsabilidades; decomposição de complexidade; encapsulamento de implementação; maior reúso e extensibilidade.**

No entanto, podem penalizar o desempenho do sistema e aumentar o esforço e complexidade de desenvolvimento do software. Sistemas Cliente-Servidor em duas camadas foram dominantes durante aproximadamente toda a década de 90 e são utilizados até hoje, mas **para minimizar o impacto de mudanças, decidiu-se separar a camada de negócio da camada de interface gráfica**, gerando três camadas:



- A **Camada de Apresentação** possui classes que contêm funcionalidades para visualização dos dados pelos usuários (Por exemplo: classes de fronteiras para atores humanos). *E qual a real importância dela?* Ela tem o objetivo de exibir informações ao usuário e traduzir ações do usuário em requisições às demais partes dos sistemas.
- A **Camada Lógica de Negócio** possui classes que implementam as regras de negócio no qual o sistema será implantado. Ela realiza computações com base nos dados armazenados ou nos dados de entrada, decidindo que parte da camada de acesso deve ser ativada com base em requisições provenientes da camada de apresentação.
- A **Camada de Acesso** possui classes que se comunicam com outros sistemas para realizar tarefas ou adquirir informações para o sistema. Tipicamente, essa camada é implementada utilizando algum mecanismo de armazenamento persistente. Pode haver uma subcamada dentro desta camada chamada Camada de Persistência.

O padrão de arquitetura em três camadas mais utilizado no mercado é designado Model-View-Controller (MVC):

- **Modelo:** responsável por modelar os dados da aplicação e regras de negócio. Tem o foco em armazenamento, manipulação e geração de dados. Objetos do Modelo são geralmente reusáveis, distribuídos, persistentes e portáteis.
- **Visão:** responsável pela apresentação dos dados aos usuários. Ele recebe entradas de dados e apresenta resultados. Essa camada não persiste nenhum dado no sistema e também não busca dados, apenas os renderiza em tela.
- **Controle:** responsável por definir o comportamento da aplicação. Processa e responde a eventos, geralmente ações do usuário, e pode invocar alterações no modelo. Realiza, também, a validação de dados do usuário.



QUESTÕES COMENTADAS

PARADIGMA ORIENTADO A OBJETOS

1. (CESGRANRIO – 2016 – UNIRIO – Técnico em Tecnologia de Informação) Em linguagens orientadas a objetos (OO), classes representam a descrição da implementação de tipos abstratos a partir dos quais instâncias podem ser criadas. Cada instância, depois de criada, guarda seu estado próprio independente das demais instâncias. Esse estado pode ser alterado de acordo com operações definidas pela classe, mas, ao serem executadas, as operações atuam individualmente sobre cada instância.

Na nomenclatura OO, instâncias e operações são conhecidas, respectivamente, como

- a) Métodos e Funções
- b) Objetos e Heranças
- c) Objetos e Métodos
- d) Tipos e Objetos
- e) Tipos e Heranças

Comentários:

Pessoal essa é fácil! Os objetos são a materialização em runtime (tempo de execução) das classes, enquanto o meio de ter acesso ao seus atributos modificando o estado são realizados através dos métodos.

Gabarito: Letra C

2. (CESGRANRIO – 2014 – EPE – Analista de Gestão Corporativa - Tecnologia de Informação) Considere que um programa orientado a objeto possui 5 classes: Máquina, Motor, MotorExplosão, MotorVapor e Gerador. MotorExplosão e MotorVapor são especializações de Motor. Motor e Gerador são especializações de Máquina. Todas as classes respondem a uma mensagem chamada "calcularPotencia", sem argumentos, que calcula e retorna um número real que indica potência do objeto, em watts, de acordo com os valores de alguns atributos, com um algoritmo diferente em cada classe. O exemplo acima caracteriza a capacidade de enviar a mesma mensagem para vários objetos e que cada objeto responda a essa mensagem de acordo com sua classe. Tal característica é conhecida como:
- a) Polimorfismo
 - b) Refatoração
 - c) Herança Múltipla
 - d) Independência de Dados
 - e) Tratamento de Exceção



Comentários:

Vamos relembrar os conceitos:

Polimorfismo:

- Permite que referências de tipos de classes mais abstratas (Pai) representem o comportamento das classes concretas (Filhas) que referenciam. Assim, é possível tratar vários tipos de maneira homogênea (através da interface do tipo mais abstrato);
- O polimorfismo é caracterizado quando duas ou mais classes distintas têm métodos de mesmo nome, de forma que uma função possa utilizar um objeto de qualquer uma das classes polimórficas, sem necessidade de tratar de forma diferenciada conforme a classe do objeto;
- Uma das formas de implementar o polimorfismo é através de uma classe abstrata, cujos métodos são declarados mas não são definidos, e através de classes que herdam os métodos desta classe abstrata;
- Outra forma é através da sobrecarga de métodos de mesmo nome mas com assinaturas diferentes.

Refatoração:

- Consiste no processo de modificação um sistema de software para melhorar a estrutura interna do código sem alterar seu comportamento externo.

Herança Múltipla:

- Consiste implementar uma classe através da herança de duas ou mais classes.

Independência de Dados:

- Conceito de Banco de Dados que diz que é permitido efetuar alterações no esquema ou no nível de um banco de dados, sem alterar um nível superior. Existem dois tipos: independência de dados lógica e independência de dados física.

Tratamento de Exceção:

- É um mecanismo capaz de realizar o tratamento da ocorrência de condições que alteram o fluxo normal da execução de programas de computadores. O fluxo então é alterado para uma nova condição que trata a condição não contida no fluxo normal.

Portanto, trata-se do Polimorfismo!

Gabarito: Letra A



3. (CESGRANRIO – 2014 – IBGE – Analista – Análise e Desenvolvimento de Sistemas) Em linguagens orientadas a objetos, existem dois conceitos fundamentais:

I – a definição de uma estrutura, a partir da qual é possível especificar todas as características da implementação, operações e armazenamento de informações para instâncias que serão criadas posteriormente.

II – instâncias específicas criadas a partir da definição das estruturas referentes ao conceito I.

Esses conceitos correspondem, respectivamente, ao que se conhece pelos nomes de:

- a) I - Tipo; II - Classe
- b) I - Tipo; II - Construtor
- c) I - Classe; II - Tipo
- d) I - Classe; II - Objeto
- e) I - Classe ; II - Metaclass

Comentários:

Sem mistério, pessoal! Classe é uma definição esquemática abstrata que se manifesta em tempo de execução através de suas instâncias chamadas Objetos.

Gabarito: Letra D

4. (FGV – 2015 – TCE/SE – Analista de Sistemas) Em POO (Programação Orientada a Objetos), dizer que a classe A estende a classe B é o mesmo que dizer que:

- a) a classe B é subclasse de A;
- b) a classe A é superclasse de B;
- c) a classe A é derivada de B;
- d) a classe B é derivada de A;
- e) as classes A e B são irmãs.

Comentários:

Essa ficou fácil! Se A estende B, A é derivada de B.

Gabarito: Letra C

5. (FGV – 2015 – TCE/SE – Analista de Sistemas) Em POO (programação orientada a objetos), dizer que a classe A é superclasse de B é o mesmo que dizer que:



- a) A é derivada de B;
- b) A estende B;
- c) B é derivada de A;
- d) B implementa A;
- e) A implementa B.

Comentários:

Essa ficou fácil! Se A é superclasse de B, B é derivada de A.

Gabarito: Letra C

6. (IBFC /Prefeitura de Divinópolis/MG – 2018) Assinale a alternativa que complete correta e respectivamente as lacunas da frase a seguir:

O _____ permite que referências de tipos de classes mais _____ representem o comportamento das classes _____ que referenciam:

- a) polimorfismo - abstratas - concretas
- b) encapsulamento - abstratas - concretas
- c) encapsulamento - concretas - abstratas
- d) polimorfismo - concretas - abstratas

Comentários:

O polimorfismo permite que referências de tipos de classes mais abstratas representem o comportamento das classes concretas que referenciam. Esse recurso permite tratar vários tipos diferentes de maneira homogênea, por meio da interface do tipo mais abstrato. Em suma, o polimorfismo busca obter um comportamento específico através de uma mesma interface da classe.

Gabarito: Letra A

7. (IBFC /TJ-PE – 2017) Em um programa orientado a objetos, verifica-se que a classe X estende a classe Y. Ou seja, pode-se afirmar, pelos preceitos da POO (Programação Orientada a Objetos), que:

- a) a classe X é superclasse de Y
- b) a classe X é uma interface de Y
- c) a classe Y é derivada de X
- d) a classe Y é subclasse de X
- e) a classe X é derivada de Y



Comentários:

Se a Classe X estende a Classe Y, então a Y é superclasse de X. Dito de outra forma, X é subclasse de Y ou X é classe derivada de Y.

Gabarito: Letra E

8. (IBFC / EMBASA – 2017) Quanto aos fundamentos básicos de programação orientada a objetos, relacione os quatro conceitos abaixo com os respectivos significados mencionados logo em seguida:

CONCEITOS:

- (1) herança.
- (2) método.
- (3) polimorfismo.
- (4) encapsulamento.

SIGNIFICADOS:

- (A) definem as habilidades dos objetos.
- (B) 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.
- (C) é o mecanismo pelo qual uma classe pode estender outra classe ou, ainda, ser estendida de outra classe.
- (D) consiste na separação de aspectos internos e externos de um objeto.

- a) 1C - 2B - 3A - 4D
- b) 1C - 2A - 3B - 4D
- c) 1D - 2A - 3B - 4C
- d) 1A - 2C - 3D - 4B.

Comentários:

(1C) Herança é o mecanismo pelo qual uma classe pode estender outra classe ou, ainda, ser estendida de outra classe; (2A) Métodos definem as habilidades dos objetos; (3B) 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; (4D) Encapsulamento consiste na separação de aspectos internos e externos de um objeto.

Gabarito: Letra B



9. (IBFC / EBSERH – 2017) Um dos conceitos em Orientação a Objetos é a classe abstrata. Assinale a alternativa que complete correta e respectivamente as lacunas da frase abaixo:

"A classe abstrata é sempre um(a) _____ que não possui _____"

- a) método - instâncias
- b) ator/atriz - atributos
- c) superclasse - instâncias
- d) superclasse - atores/atrizes
- e) método - atributos.

Comentários:

A classe abstrata é sempre uma superclasse que não possui instâncias – lembrem-se que ela não pode ser instanciada.

Gabarito: Letra C

10. (IBFC / EMDEC – 2016) Quanto a Programação Orientada a Objeto identifique a alternativa que representa o mecanismo pelo qual uma classe (sub-classe) pode estender outra classe (super-classe), aproveitando seus comportamentos (métodos) e variáveis possíveis (atributos):

- a) herança
- b) encapsulamento
- c) mensagem
- d) polimorfismo.

Comentários:

Mecanismo pelo qual uma subclasse pode estender uma superclasse (opa, isso já parece herança...) aproveitando seus comportamentos e variáveis (é, realmente é herança!).

Gabarito: Letra A

11. (IBFC / TRE-AM – 2014) Em programação orientada a objetos significa separar o programa em partes, o mais isoladas possível, tornando o software mais flexível e fácil de modificar:

- a) Herança.
- b) Encapsulamento.
- c) Polimorfismo.
- d) Atributo.



Comentários:

Separar em partes? Isoladas? Tornando o software mais flexível e fácil de modificar? Trata-se do Encapsulamento!

Gabarito: Letra B

12. (IBFC / EBSERH – 2017) Consiste no princípio pelo qual 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:

- a) abstração
- b) herança
- c) interface
- d) mensagem
- e) polimorfismo

Comentários:

Princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma identificação (assinatura), mas comportamentos distintos é o princípio do... polimorfismo.

Gabarito: Letra E

13. (CESPE - 2015 – TRE/GO - Técnico Judiciário - Tecnologia da Informação) Uma classe abstrata possui instâncias diretas, bem como suas classes descendentes.

Comentários:

Galera, classe abstrata não gera instâncias diretas! Item já está errado daí...

Gabarito: Errado

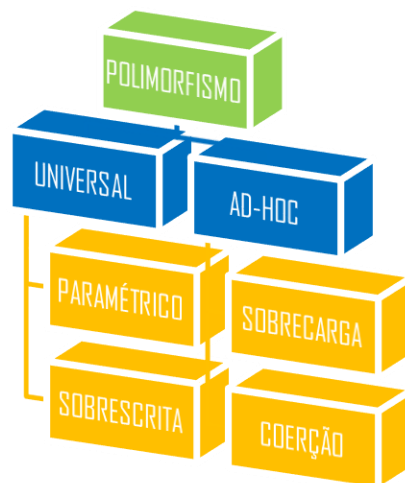
14. (CESPE - 2016 – TCE/PR – Analista de Sistemas) Em se tratando de orientação a objetos, o polimorfismo refere-se:

- a) ao reconhecimento do comportamento variado de um método, o que melhora o aproveitamento do código.
- b) à transmissão dos métodos e atributos de uma classe para suas subclasses, quando ad hoc.



- c) à variação das saídas de um método com relação às entradas recebidas, caso em que é considerado universal.
- d) ao uso que um objeto faz dos recursos de outro objeto.
- e) à utilização de métodos específicos para manipular dados com proteção por meio de encapsulamento.

Comentários:



(a) Correto, métodos de mesmo nome com comportamentos variados; (b) Errado, a questão trata de sobrescrita que é polimorfismo universal e, não, ad-hoc; (c) Errado, saída diferente, logo assinatura diferente, portanto é ad-hoc e, não, universal; (d) Errado, isso é um relacionamento de associação e, não, polimorfismo; (e) Errado, não são métodos específicos e o encapsulamento é irrelevante.

Gabarito: Letra A

15. (CESPE – 2015 – TJDF – Analista de Sistemas) Recurso de classes é a propriedade dos objetos que viabiliza a implementação de hierarquia entre objetos.

Comentários:

O que é Recurso de Classes? Não faço ideia, isso não existe no contexto de orientação a objetos! *E o que é a propriedade dos objetos que viabiliza a implementação de hierarquia entre objetos?* É a Herança! O examinador trocou herança por um nome qualquer – questão errada!

Gabarito: Errado

16. (CESPE – 2015 – TJDF – Analista de Sistemas) A combinação de dados com o código que os manipula em um único objeto é denominada encapsulamento.



Comentários:

Galera, não se desesperem! Quando eu li essa questão, eu também pensei: “*Einh?*”. Achei MUITO mal escrita e, em minha opinião, essa não é uma boa definição de encapsulamento. Encapsulamento é a restrição ao acesso direto a dados de um objeto e também uma forma de agrupar dados com os métodos que vão manipular esses dados – e foi isso que a questão quis dizer com essa redação estranha. Ou seja, a combinação de dados com o código que os manipula em um único objeto e esse código que os manipula são os métodos. *Bacana?*

Gabarito: Correto

17.(CESPE – 2016 – FUB – Analista de Sistemas) Uma das vantagens de se utilizar encapsulamento em orientação a objetos é impedir o acesso direto aos atributos de um objeto.

Comentários:

Perfeito! O Encapsulamento realmente protege os dados do objeto, impedindo acesso direto a eles.

Gabarito: Correto

18.(CESPE – 2015 – TCE/RN – Analista de Sistemas) O processo de herança permite a reutilização de código, como também o reaproveitamento de atributos e métodos. Assim, em aplicações que utilizam herança, a obtenção de polimorfismo é uma possibilidade.

Comentários:

Corretíssimo! A primeira parte está correta, visto que ela permite a reutilização de código, reaproveitamento de atributos e métodos. E, aplicações que utilizam herança, podem realizar polimorfismo quando em classes diferentes – a isso, designamos sobrescrita de métodos. Então, o polimorfismo dinâmico está intimamente ligado ao conceito de herança.

Gabarito: Correto

19.(CESPE – 2015 – MEC – Analista de Sistemas) Objetos são definidos como entidades da modelagem de sistemas que armazenam estados com a utilização de atributos dos próprios objetos, sem interação com outros objetos por meio de mensagens.

Comentários:



A questão vinha bonitinha até dizer que objetos não interagem com outros objetos por meio de mensagens. Eles interagem, sim. E por meio de mensagens, sim.

Gabarito: Errado

20. (CESPE – 2017 – SEDF – Analista de Sistemas) Um objeto define atributos, comportamentos e abstrações comuns compartilhados por um tipo de classe.

Comentários:

Na verdade, uma classe define atributos, comportamentos e abstrações comuns compartilhados por um tipo de objeto.

Gabarito: Errado

21. (CESPE – 2016 – FUB – Analista de Sistemas) O conjunto de valores das características de determinado objeto é denominado estado.

Comentários:

Perfeito! O estado de um objeto é o conjunto de valores das características (atributos) desse objeto.

Gabarito: Correto

22. (CESPE – 2016 – FUB – Analista de Sistemas) Quando uma classe é subclasse de várias superclasses, mas somente herda características de uma classe, a herança é do tipo simples.

Comentários:

Galera, a questão já está conceitualmente errada porque é impossível herdar apenas de uma superclasse. Você não pode dizer que você é um primata, mas não é um mamífero, porque todos os primatas são mamíferos. Vamos aproveitar para definir alguns conceitos importantes. Primeiro: herdar é diferente de acessar! Se uma classe estende a outra, ela sempre herdará seus métodos/atributos.

Não importa, por exemplo, se eles são privados - a subclasse sempre os herdará, mesmo que não os acesse! Como eu gosto de visualizar isso? Imaginem que um tio-avô distante deixe um cofre entupido de dinheiro para vocês. No entanto, esse cofre é completamente indestrutível e ele não deixou nenhuma senha. Nesse caso, vocês herdaram todo o dinheiro, mas não podem acessá-lo.

Segundo: há que se tomar um certo cuidado com a redação desse tipo de questão. Por que? Porque quando a questão afirma que uma classe é subclasse de várias superclasses, ela não disse



em nenhum momento que essas superclasses não estavam em uma mesma hierarquia ou disse que a classe era subclasse direta de várias superclasses. Ok? Já houve questões cobrando essa pegadinha!

Esse tipo de questão é rodeado por polêmicas. Por que? Porque alguns dizem que o lance de herdar/acessar é válido para objetos, mas não para classes. Além disso, algumas especificações de linguagens (Ex: Java) deixam explícito que subclasses não herdam atributos/métodos privados. No entanto, aqui estamos falando do Paradigma Orientado a Objetos.

Gabarito: Errado

23. (CESPE – 2017 – TRE/BA - Analista de Sistemas) Na orientação a objetos, o conceito de polimorfismo é implementado, em algumas linguagens, por meio da técnica de sobrecarga de métodos. Sobre a aplicação desse conceito, é possível afirmar que:

- a) um mesmo método pode estar em classes diferentes, com a mesma assinatura, sem que isso prejudique a sua identificação pelo compilador.
- b) a identificação da assinatura do método corresponde ao seu nome.
- c) uma classe derivada da classe-mãe tem vários métodos com a mesma assinatura, e o compilador realiza o primeiro deles.
- d) a classe derivada da classe-mãe herda os métodos da sua classe-mãe na forma como são implementados.
- e) cada método tem um nome único na classe derivada da classe-mãe a ser identificado pelo compilador.

Comentários:

Vejam a pegadinha! O enunciado fala sobre o conceito de polimorfismo e diz que ele pode ser implementado por meio da técnica de sobrecarga de métodos. No entanto, ao final ele diz “sobre esse conceito” e, não, sobre essa técnica. Logo, estamos falando do polimorfismo como um todo e, não, sobre a técnica de implementação de polimorfismo por meio da sobrecarga de dados. Ok?

(a) Correto, um mesmo método pode estar em classes diferentes, i.e., sobrescrita de métodos – com exatamente a mesma assinatura e isso não prejudica em nada o compilador; (b) Errado, nós vimos em aula que a assinatura é composta pelo nome e lista de parâmetros (tipo, ordem, quantidade); (c) Errado, não necessariamente tem vários métodos com a mesma assinatura e não existe isso de o compilador realizar o primeiro deles; (d) Errado, se ele herda na forma como são implementados, não temos polimorfismo, temos apenas herança; (e) Errado, métodos podem ter o mesmo nome – isso é polimorfismo.



Gabarito: Letra A

24. (CESPE – 2017 – TRE/BA - Analista de Sistemas) A partir de uma classe derivada de uma superclasse, podem-se invocar métodos que tenham a mesma assinatura, mas comportamentos distintos, ou seja, em que haja alteração do funcionamento interno de um método herdado de um objeto pai. Na orientação a objetos, isso é possível por meio de:

- a) polimorfismo.
- b) abstração.
- c) encapsulamento.
- d) namespaces.
- e) atributos.

Comentários:

Método com a mesma assinatura do método da classe-pai, mas funcionamento interno diferente. *Qual seria o nome disso?* Fácil, é polimorfismo! Vamos deixar a questão mais difícil: *De que tipo? É Estático ou Dinâmico? É Sobrescrita ou Sobrecarga? É Overriding ou Overloading? Ocorre em tempo de compilação ou de execução? Ou seria uma Redefinição?*

A questão trata de polimorfismo dinâmico, porque o método a ser executado será definido em tempo de execução. Está intimamente ligado ao conceito de herança e também é chamado de Sobrescrita, Redefinição ou Overriding. Easy! :)

Gabarito: Letra A

ANÁLISE E PROJETO

25. (CESPE – 2011 – MPE/TO – Analista de Sistemas) Entre os diversos diagramas utilizados em análise e projeto orientados a objetos, o diagrama de casos de uso, por procurar representar todas as possíveis situações de utilização do sistema, é considerado o diagrama responsável por mostrar a estrutura estática do sistema.

Comentários:

Aqui estamos falando de UML! Diagramas de Casos de Uso são responsáveis por representar a funcionalidade de um sistema, logo é dinâmica e, não, estática.

Gabarito: Errado

26. (CESPE – 2007 – PETROBRÁS – Analista de Sistemas) Em um modelo de análise, as classes de fronteira modelam interações entre o sistema e os atores. Cada classe de fronteira deve



estar relacionada a um ou mais atores. Pode-se também ter classes de entidade, as quais tipicamente modelam dados persistentes.

Comentários:

- **Classe de Fronteira:** classe utilizada para modelar a interação entre um ator e o sistema. Para cada ator, é identificada pelo menos uma classe de fronteira para permitir sua interação com o sistema. Então, uma classe de fronteira existe para que o sistema se comunique com o mundo exterior, logo elas são altamente dependentes do ambiente.

- **Classe de Entidade:** classe utilizada para armazenar a informação que é manipulada ou processada pelo caso de uso, partindo do domínio do negócio. **Geralmente, essas classes armazenam informações persistentes.** Há várias instâncias ou objetos de uma mesma classe de entidade coexistindo dentro do sistema.

Perfeito, é exatamente isso!

Gabarito: Correto

27. (CESPE – 2007 – PETROBRÁS – Analista de Sistemas) Em um modelo de análise, as classes de controle podem encapsular controles relacionados a casos de uso e representar lógicas de negócio que não se relacionem a uma classe de entidade específica.

Comentários:

- **Classe de Controle:** classe utilizada para controlar a lógica de execução ou negócio correspondente a cada caso de uso. Servem como uma ponte de comunicação entre objetos de fronteira e objetos de entidade. Decidem o que o sistema deve fazer quando um evento externo relevante ocorre, agindo como coordenador para a realização de casos de uso.

Perfeito, não confundam: Regras de Negócio ficam nas Classes de Entidades, mas a Lógica de Negócio fica nas Classes de Controle.

Gabarito: Correto

28. (CESPE – 2007 – PETROBRÁS – Analista de Sistemas) Em um modelo de projeto, para que um subsistema seja coeso, seus conteúdos devem ser fortemente relacionados e, para que ele seja fracamente acoplado, é necessário que se minimizem as dependências entre subsistemas.

Comentários:



Essa questão é uma pegadinha! Ele diz que "seus conteúdos" devem ser fortemente relacionados, portanto estamos falando de dentro de um subsistema (de um objeto, por exemplo). Ele não está falando da relação entre dois subsistemas (se estivesse, estaria tratando de acoplamento e, não, coesão). Logo, dentro de um subsistema, os conteúdos devem ser fortemente relacionados para que um subsistema seja coeso, ou seja, tenha uma responsabilidade única.

Gabarito: Correto

29. (CESPE – 2006 – ANATEL – Analista de Sistemas) Uma classe na análise orientada a objeto representa uma abstração que pode ser mapeada para mais de uma classe no projeto. As classes na análise podem ser fronteiras, controladoras ou entidades. Uma fronteira modela interações entre o sistema e atores, uma entidade modela apenas objetos persistentes e uma controladora só pode controlar interações entre instâncias de uma mesma classe.

Comentários:

De fato, uma classe de análise pode estar mapeada para mais de uma classe de projeto e podem ser de Fronteira, Controle ou Entidade. No entanto, uma Classe de Controle pode controlar interações entre instâncias de classes diferentes.

Gabarito: Errado

30. (CESPE – 2006 – TSE – Analista de Sistemas - A) Um modelo de análise é menos abstrato que um de projeto e as classes em um modelo de análise não podem ser conceituais. As classes na análise podem modelar objetos persistentes, mas não transientes.

Comentários:

Galera, o Modelo de Análise é mais abstrato que um Modelo de Projetos. Esse segundo já começa a pensar na solução do problema, e o primeiro pensa apenas na modelagem do problema. Ademais, as classes em um Modelo de Análise podem também ser conceituais. De modo geral, as classes de análise modelam objetos persistentes, mas não transientes – basta lembrar que, na análise, estamos apenas tratando do problema e, não, de sua solução.

Gabarito: Errado

31. (CESPE – 2006 – TSE – Analista de Sistemas - D) Um modelo de análise pode realizar casos de uso. A realização de um caso de uso descreve interações entre objetos. Na UML, essas realizações podem ser documentadas via diagramas de colaboração.

Comentários:



Perfeito, é exatamente isso! Um modelo de análise realmente pode realizar casos de uso, descrevendo interações entre objetos.

Gabarito: Correto

32. (CESPE – 2012 – IPEA – Analista de Sistemas) A análise orientada a objetos, o projeto orientado a objetos e a programação orientada a objetos compreendem atividades de engenharia de software voltadas à construção de sistemas orientados a objetos. Nesses sistemas, objetos interagem para prover serviços. No nível de programação, as interações ocorrem via interfaces das classes das quais os objetos são instâncias. Essas interfaces contêm membros públicos das classes.

Comentários:

Perfeito! Definições impecáveis...

Gabarito: Correto



LISTA DE QUESTÕES

PARADIGMA ORIENTADO A OBJETOS

1. **(CESGRANRIO – 2010 – PETROBRÁS – Analista de Sistemas)** Analise as afirmativas a seguir relativas ao paradigma da orientação a objetos.

I - O princípio do encapsulamento preconiza que um objeto deve esconder a sua complexidade interna.

II - Uma mensagem de um objeto A para um objeto B indica que A realizou uma tarefa requisitada por B.

III - A existência da mesma operação polimórfica definida em duas classes, ClasseA e ClasseB, implica necessariamente que ou ClasseA seja subclasse de ClasseB, ou que ClasseB seja subclasse de ClasseA.

É correto APENAS o que se afirma em:

- a) I.
 - b) II.
 - c) I e II.
 - d) I e III.
 - e) II e III.
2. **(CESGRANRIO – 2016 – UNIRIO – Técnico em Tecnologia de Informação)** Em linguagens orientadas a objetos (OO), classes representam a descrição da implementação de tipos abstratos a partir dos quais instâncias podem ser criadas. Cada instância, depois de criada, guarda seu estado próprio independente das demais instâncias. Esse estado pode ser alterado de acordo com operações definidas pela classe, mas, ao serem executadas, as operações atuam individualmente sobre cada instância.

Na nomenclatura OO, instâncias e operações são conhecidas, respectivamente, como

- a) Métodos e Funções
 - b) Objetos e Heranças
 - c) Objetos e Métodos
 - d) Tipos e Objetos
 - e) Tipos e Heranças
3. **(CESGRANRIO – 2014 – EPE – Analista de Gestão Corporativa - Tecnologia de Informação)** Considere que um programa orientado a objeto possui 5 classes: Máquina, Motor, MotorExplosão, MotorVapor e Gerador. MotorExplosão e MotorVapor são especializações de



Motor. Motor e Gerador são especializações de Máquina. Todas as classes respondem a uma mensagem chamada "calcularPotencia", sem argumentos, que calcula e retorna um número real que indica potência do objeto, em watts, de acordo com os valores de alguns atributos, com um algoritmo diferente em cada classe. O exemplo acima caracteriza a capacidade de enviar a mesma mensagem para vários objetos e que cada objeto responda a essa mensagem de acordo com sua classe. Tal característica é conhecida como:

- a) Polimorfismo
- b) Refatoração
- c) Herança Múltipla
- d) Independência de Dados
- e) Tratamento de Exceção

4. (CESGRANRIO – 2014 – IBGE – Analista – Análise e Desenvolvimento de Sistemas) Em linguagens orientadas a objetos, existem dois conceitos fundamentais:

I – a definição de uma estrutura, a partir da qual é possível especificar todas as características da implementação, operações e armazenamento de informações para instâncias que serão criadas posteriormente.

II – instâncias específicas criadas a partir da definição das estruturas referentes ao conceito I.

Esses conceitos correspondem, respectivamente, ao que se conhece pelos nomes de:

- a) I - Tipo; II - Classe
- b) I - Tipo; II - Construtor
- c) I - Classe; II - Tipo
- d) I - Classe; II - Objeto
- e) I - Classe ; II - Metaclass

5. (ESAF - 2010 – SUSEP - Analista de Sistemas) Em relação à programação orientada a objetos, é correto afirmar que:

- a) o objeto é definido por atributos.
- b) objetos são instâncias de um atributo.
- c) apenas atributos numéricos são válidos.
- d) atributos podem ser agrupados em pointvalues.
- e) atributos adequados dispensam referências a objetos.

6. (ESAF - 2010 – SUSEP - Analista de Sistemas – Letra B) É correto afirmar que em herança simples uma superclasse pode ter apenas uma subclasse.

7. (ESAF - 2010 – SUSEP - Analista de Sistemas) Polimorfismo é a:



- a) utilização múltipla de programas em análise orientada a objetos.
- b) habilidade de uma única operação ou nome de atributo ser definido em mais de uma classe e assumir diferentes implementações em cada uma dessas classes.
- c) habilidade de um programador em desenvolver aplicações e caracterizar objetos com múltiplos atributos.
- d) utilização de uma classe com diferentes formatos em programas com definição de objetos e atributos.
- e) habilidade de uma única variável ser utilizada em diferentes programas orientados a objetos.

8. (ESAF - 2012 – CGU - Analista de Sistemas) Assinale a opção correta.

- a) As classes podem formar heranças segmentadas em classes adjacentes.
- b) Overflow é a redefinição do fluxo de uma classe, em uma de suas subclasses.
- c) Overriding é a redefinição de um método, definido em uma classe, em uma de suas subclasses.
- d) Overriding é a redefinição de uma classe através de métodos de objetos diferentes.
- e) As classes não podem formar hierarquias de herança de superclasses e subclasses.

9. (ESAF - 2013 – DNIT - Analista de Sistemas) A herança de D a partir de C é a habilidade que uma classe D tem implicitamente definida:

- a) em atributos e análises da classe C.
- b) em cada um dos modelos e concepções da classe C.
- c) em cada um dos atributos e operações da classe C.
- d) em parte das funcionalidades e operações de classes equivalentes.
- e) nos programas das classes.

10. (FCC - 2012 - TRE-SP - Analista Judiciário - Análise de Sistemas) Nos conceitos de orientação a objetos, ..I... é uma estrutura composta por ...II... que descrevem suas propriedades e também por ...III.... que moldam seu comportamento.IV.... sãoV.... dessa estrutura e só existem em tempo de execução.

Para completar corretamente o texto as lacunas devem ser preenchidas, respectivamente, por



- a) objeto, métodos, assinaturas, Classes, cópias.
- b) polimorfismo, funções, métodos, Herança, cópias.
- c) classe, atributos, operações, Objetos, instâncias.
- d) multiplicidade, símbolos, números, Classes, herdeiros.
- e) domínio, diagramas, casos de caso, Diagramas de classe, exemplos.

11. (FCC - 2012 - TJ-RJ - Analista Judiciário - Análise de Sistemas) No contexto de programação orientada a objetos, considere as afirmativas abaixo.

- I. Objetos são instâncias de classes.
- II. Herança é uma relação entre objetos.
- III. Mensagens são formas de executar métodos.
- IV. Classes são apenas agrupamentos de métodos.
- V. Ocorre herança múltipla quando mais de um método é herdado.
- VI. Herança é uma relação entre classes.

Está correto o que se afirma APENAS em:

- a) I, III e IV.
- b) I, III e VI.
- c) III, IV e VI.
- d) II, III e V.
- e) II, IV e V.

12. (FCC - 2009 - TJ-SE - Técnico Judiciário - Programação de Sistemas) Na programação orientada a objetos, são características dos objetos:

- a) As classes, os métodos e as mensagens.
- b) A identidade, os atributos e as operações.
- c) O encapsulamento, a herança e o polimorfismo.
- d) A instanciação, a generalização e a especialização.
- e) A classificação, a composição e a decomposição.

13. (FCC - 2012 - TRF - 2ª REGIÃO - Técnico Judiciário – Informática – E) Os objetos de uma classe são idênticos no que se refere à sua interface e ao seu estado.

14. (FCC - 2008 - TCE-AL - Programador) Considere: *Casas ABC Ltda., Empresa e Nome da Empresa*. Na orientação a objetos, os itens acima representam, respectivamente,

- a) atributo, classe e objeto.
- b) classe, atributo e objeto.
- c) classe, objeto e atributo.
- d) objeto, atributo e classe.
- e) objeto, classe e atributo.



15. (FCC - 2009 - SEFAZ-SP - Agente Fiscal de Rendas - Tecnologia da Informação - Prova 3)

Uma classe é uma abstração que ajuda a lidar com a complexidade e um bom exemplo de abstração é:

- a) um aluno e as disciplinas que está cursando.
- b) um professor e os cursos nos quais ministra aulas.
- c) um funcionário e o departamento em que trabalha.
- d) uma pessoa e o número do seu CPF na Receita Federal.
- e) uma casa e a empresa que a projetou e construiu.

16. (FCC - 2011 - INFRAERO - Analista de Sistemas - Desenvolvimento e Manutenção – A)

Uma classe é o projeto do objeto. Ela informa à máquina virtual como criar um objeto de um tipo específico. Cada objeto criado a partir da classe terá os mesmos valores para as variáveis de instância da classe.

17. (FCC - 2010 - DPE-SP - Agente de Defensoria - Programador) Classes e objetos são dois conceitos-chave da programação orientada a objetos. Com relação a estes conceitos, é correto afirmar que:

- a) uma classe é uma descrição de um ou mais objetos por meio de um conjunto uniforme de atributos e serviços. Além disso, pode conter uma descrição de como criar novos objetos na classe.
- b) uma classe é capaz de armazenar estados através de seus atributos e reagir a mensagens enviadas a ela, assim como se relacionar e enviar mensagens a outras classes.
- c) uma classe é uma abstração de alguma coisa no domínio de um problema ou na sua implementação, refletindo a capacidade de um sistema para manter informações sobre ela, interagir com ela ou ambos.
- d) um objeto em uma classe é apenas uma definição, pois a ação só ocorre quando o objeto é invocado através de um método.
- e) herança é o mecanismo pelo qual um objeto pode estender outro objeto, aproveitando seus comportamentos e variáveis possíveis.

18. (FCC - 2010 - TCE-SP - Agente da Fiscalização Financeira - Informática - Suporte de Web)

A descrição de um conjunto de entidades (reais ou abstratas) de um mesmo tipo e com as mesmas características e comportamentos. Trata-se da definição de:

- a) String.
- b) Método.
- c) Conjunto.



- d) Classe.
- e) Objeto.

19. (FCC - 2010 - DPE-SP - Agente de Defensoria - Programador) A cidade de São Paulo, que possuía uma população de 10.000.000 de habitantes, teve um aumento de mais 2.000.000 de novos habitantes.

Na associação da frase acima aos conceitos da modelagem orientada a objeto, é correto afirmar que São Paulo, população e aumento, referem-se, respectivamente, a:

- a) classe, objeto, instância de classe.
- b) objeto, atributo, implementação por um método do objeto.
- c) classe, objeto, atributo.
- d) objeto, instância, operação.
- e) classe, objeto, associação pelo método de agregação.

20. (FCC - 2011 - INFRAERO - Analista de Sistemas - Desenvolvimento e Manutenção – A) Sobre a programação orientada a objetos, analise:

- I. Neste tipo de programação, objetos executam ações, mas não suportam propriedades ou atributos.
- II. Uma classe especifica o formato geral de seus objetos.
- III. As propriedades e ações disponíveis para um objeto não dependem de sua classe.
- IV. A tecnologia orientada a objetos permite que classes projetadas adequadamente sejam reutilizáveis em vários projetos.

Está correto o que consta em:

- a) II, III e IV, apenas.
- b) I e II, apenas.
- c) II e IV, apenas.
- d) I, II e III, apenas.
- e) I, II, III e IV.

21. (FCC - 2010 - TRE-RS - Analista Judiciário - Analista de Sistemas Suporte) Um objeto é, na orientação a objetos,

- a) uma rotina de programação contida em uma classe que pode ser chamada diversas vezes possibilitando assim reuso de código de programação.
- b) um conjunto de atributos primitivos tipados contido em uma classe.



- c) uma entidade que possui um estado e um conjunto definido de operações definidas para funcionar nesse estado.
- d) um elemento de uma classe que representa uma operação (a implementação de uma operação).
- e) uma porção de código que resolve um problema muito específico, parte de um problema maior.

22. (FCC - 2011 - TRT - 4ª REGIÃO (RS) - Analista Judiciário - Tecnologia da Informação) O aumento da produtividade de desenvolvimento e a capacidade de compartilhar o conhecimento adquirido, representa uma vantagem no uso de projetos orientados a objeto, porque:

- a) um objeto pode ser chamado por objetos de classe diferente da sua.
- b) os objetos podem ser potencialmente reutilizáveis.
- c) as classes podem ser concretas ou abstratas.
- d) todo método pode ser derivado naturalmente das operações de sua classe.
- e) o encapsulamento impossibilita equívocos de código.

23. (FCC - 2011 - INFRAERO - Analista de Sistemas - Desenvolvimento e Manutenção – D) Os objetos têm seu estado definido pelos métodos e seu comportamento definido nas variáveis de instância.

24. (FCC - 2009 - TRE-PI - Técnico Judiciário - Programação de Sistemas – II) A classe é constituída por atributos que representam os dados e operações que representam os métodos que podem ser executados.

25. (FCC - 2011 - TRT - 14ª Região (RO e AC) - Técnico Judiciário - Tecnologia da Informação – IV) Um objeto é capaz de armazenar estados através de seus atributos e reagir a mensagens enviadas a ele, assim como se relacionar e enviar mensagens a outros objetos.

26. (FCC - 2011 - TRT - 23ª REGIÃO (MT) - Analista Judiciário - Tecnologia da Informação) Sobre os conceitos de orientação a objetos, considere:

- I. Classe encapsula dados para descrever o conteúdo de alguma entidade do mundo real.
- II. Objetos são instâncias de uma classe que herdam os atributos e as operações da classe.
- III. Superclasse é uma especialização de um conjunto de classes relacionadas a ela.
- IV. Operações, métodos ou serviços fornecem representações dos comportamentos de uma classe.



Está completo e correto o que consta em

- a) I, II, III e IV.
- b) I, II e IV, apenas.
- c) II, III e IV, apenas.
- d) I e II, apenas.
- e) II e IV, apenas.

27. (FCC - 2012 - TRE-CE - Analista Judiciário - Análise de Sistemas) Orientação a Objetos é um paradigma de análise, projeto e programação de sistemas de software. A respeito desse paradigma, assinale a afirmativa incorreta.

- a) Um objeto pode ser considerado um conjunto de dados.
- b) Os objetos possuem identidade, estado e comportamento.
- c) Um evento pode existir se não houver um objeto a ele associado.
- d) Um objeto pode existir mesmo que não exista nenhum evento associado a ele.
- e) A orientação a objetos implementa o conceito de abstração, classe, objeto, encapsulamento, herança e polimorfismo.

28. (FCC - 2012 - TJ-PE - Técnico Judiciário - Programador de Computador – II) Objetos com os mesmos atributos e operações possuem a mesma identidade, podendo ser referenciados por outros objetos.

29. (FCC - 2010 - TRT - 22ª Região (PI) - Técnico Judiciário - Tecnologia da Informação – I) Um objeto pode ser real ou abstrato. Sendo uma instância de uma classe, possui informações e desempenha ações.

30. (FCC - 2010 - TRT - 22ª Região (PI) - Técnico Judiciário - Tecnologia da Informação – II) Uma classe especifica uma estrutura de dados e os métodos operacionais permissíveis que se aplicam a cada um de seus objetos. Pode ter sua própria estrutura de dados e métodos, bem como pode herdá-la de sua superclasse.

31. (FCC - 2009 - SEFAZ-SP - Agente Fiscal de Rendas - Tecnologia da Informação - Prova 3) Os valores das propriedades de um objeto em um determinado instante, que podem mudar ao longo do tempo, representam:

- a) a instância de uma classe.
- b) a identidade de um objeto.
- c) o estado de um objeto.



- d) o comportamento de um objeto.
- e) as operações de uma classe.

32. (FCC - 2012 - TRF - 2ª REGIÃO - Técnico Judiciário - Informática) As variáveis de uma classe só podem ser alteradas por métodos definidos nos seus objetos.

33. (FCC - 2009 - SEFAZ-SP - Agente Fiscal de Rendas - Tecnologia da Informação - Prova 3) Na orientação a objetos, ao nível de classe, são definidos os:

- a) atributos e os valores dos atributos.
- b) atributos e a invocação das operações.
- c) atributos e os métodos.
- d) métodos e os valores dos atributos.
- e) métodos e a invocação das operações.

34. (FCC - 2012 - TRE-CE - Analista Judiciário - Análise de Sistemas – C) Um construtor visa inicializar os atributos e pode ser executado automaticamente sempre que um novo objeto é criado.

35. (FCC - 2011 - Nossa Caixa Desenvolvimento - Analista de Sistemas) Na orientação a objetos, é um recurso que serve para inicializar os atributos e é executado automaticamente sempre que um novo objeto é criado:

- a) método.
- b) polimorfismo.
- c) interface.
- d) classe.
- e) construtor.

36. (FCC - 2011 - TRE-RN - Analista Judiciário - Análise de Sistemas) Método especial destinado ao preparo de novos objetos durante sua instanciação. Pode ser acionado por meio do operador new, recebendo parâmetros como métodos comuns, o que permite caracterizar os objetos já na instanciação. Trata-se de:

- a) operação polimórfica.
- b) construtor.
- c) atributo.
- d) herança polimórfica.
- e) herança múltipla.

37. (FCC - 2009 - SEFAZ-SP - Agente Fiscal de Rendas - Tecnologia da Informação - Prova 3) O método utilizado para inicializar objetos de uma classe quando estes são criados é denominado:



- a) void.
- b) interface.
- c) agregação.
- d) composição.
- e) construtor.

38.(FCC - 2009 - TJ-PI - Analista Judiciário - Tecnologia da Informação) Na programação orientada a objetos, é o princípio que oferece a capacidade de um método poder ser implementado de diferentes formas, ou mesmo de realizar coisas diferentes, ou seja, um único serviço pode oferecer variações, conforme se aplique a diferentes subclasses de uma superclasse. No contexto, o termo método é:

- a) o mecanismo pelo qual um objeto utiliza os recursos de outro.
- b) uma instância de uma classe.
- c) o elemento que define as habilidades do objeto.
- d) uma chamada a um objeto para invocar uma classe.
- e) um objeto capaz de armazenar estados através de seus atributos.

39.(FCC - 2013 - TRT - 12ª Região (SC) - Analista Judiciário - Tecnologia da Informação) Na programação orientada a objetos, as classes podem conter, dentre outros elementos, métodos e atributos. Os métodos:

- a) devem receber apenas parâmetros do mesmo tipo.
- b) não podem ser sobrecarregados em uma mesma classe.
- c) precisam possuir corpo em interfaces e classes abstratas.
- d) podem ser sobrescritos em aplicações que possuem relação de herança.
- e) definidos como private só podem ser acessados de classes do mesmo pacote.

40.(FCC - 2012 - TRE-CE - Analista Judiciário - Análise de Sistemas) Uma classe define o comportamento dos objetos através de seus métodos, e quais estados ele é capaz de manter através de seus atributos.

41.(FCC - 2009 - PGE-RJ - Técnico Superior de Análise de Sistemas e Métodos) Sobre orientação a objetos, considere:

- I. Os valores dos atributos são definidos no nível de classe.
- II. Os métodos são definidos no nível de objeto.
- III. A invocação de uma operação é definida no nível de objeto.

Está correto o que se afirma em:

- a) II e III, apenas.
- b) I, II e III.
- c) III, apenas.



- d) I e II, apenas.
- e) I e III, apenas.

42. (FCC - 2011 - Nossa Caixa Desenvolvimento - Analista de Sistemas) Na programação orientada a objetos, subprogramas (ou subrotinas) são encapsuladas nos próprios objetos e passam a designar-se:

- a) atributo.
- b) herança.
- c) instância.
- d) método.
- e) encapsulamento.

43. (FCC - 2011 - Nossa Caixa Desenvolvimento - Analista de Sistemas – III) Objetos se comunicam por passagem de mensagem, eliminando áreas de dados compartilhados.

44. (FCC - 2009 - TJ-PA - Analista Judiciário - Tecnologia da Informação) A especificação de uma comunicação entre objetos, que contém informações relacionadas ao que se espera resultar dessa atividade, é:

- a) uma restrição.
- b) uma mensagem.
- c) uma operação.
- d) um processo oculto.
- e) um diálogo.

45. (FCC - 2010 - TCM-PA - Técnico em Informática) Não possui instâncias diretas, mas apenas classes descendentes:

- a) a classe concreta.
- b) o objeto.
- c) a classe abstrata.
- d) o caso de uso de inclusão.
- e) o pacote.

46. (FCC - 2011 - TCE-PR - Analista de Controle – Informática – C) Interfaces são como as classes abstratas, mas nelas não é possível implementar nenhum método, apenas declarar suas assinaturas; uma classe ao implementar uma interface deverá escrever todos os seus métodos.

47. (FCC - 2011 - INFRAERO - Analista de Sistemas - Desenvolvimento e Manutenção – C) Uma interface é uma classe 100% abstrata, ou seja, uma classe que não pode ser instanciada.



- 48.(FCC - 2013 - TRT - 12ª Região (SC) - Analista Judiciário - Tecnologia da Informação – C)** Na programação orientada a objetos, as classes podem conter, dentre outros elementos, métodos e atributos. Os métodos precisam possuir corpo em interfaces e classes abstratas.
- 49.(FCC - 2009 - SEFAZ-SP - Agente Fiscal de Rendas - Tecnologia da Informação - Prova 3 - I)** Os métodos públicos de uma classe definem a interface da classe.
- 50.(FCC - 2009 - SEFAZ-SP - Agente Fiscal de Rendas - Tecnologia da Informação - Prova 3 - II)** Os métodos privados de uma classe não fazem parte da interface da classe.
- 51. (FCC - 2013 - AL-RN - Analista Legislativo - Analista de Sistemas)** Um dos conceitos básicos de orientação a objetos é o fato de um objeto, ao tentar acessar as propriedades de outro objeto, deve sempre fazê-lo por uso de métodos do objeto ao qual se deseja atribuir ou requisitar uma informação, mantendo ambos os objetos isolados. A essa propriedade da orientação a objetos se dá o nome de:
- a) herança.
 - b) abstração.
 - c) polimorfismo.
 - d) mensagem.
 - e) encapsulamento.
- 52. (FCC - 2010 - TRT - 9ª REGIÃO (PR) - Técnico Judiciário - Tecnologia da Informação)** Uma técnica que consiste em separar aspectos externos dos internos da implementação de um objeto, isto é, determinados detalhes ficam ocultos aos demais objetos e dizem respeito apenas ao próprio objeto.
- Trata-se de:
- a) polimorfismo.
 - b) generalização.
 - c) encapsulamento.
 - d) herança.
 - e) visibilidade.
- 53. (FCC - 2009 - TRE-PI - Técnico Judiciário - Programação de Sistemas – I)** A afirmação de que o estado de um objeto não deve ser acessado diretamente, mas sim por meio de métodos de acesso, está associada ao conceito de encapsulamento.
- 54. (FCC - 2009 - SEFAZ-SP - Agente Fiscal de Rendas - Tecnologia da Informação - Prova 3)** Sobre a visibilidade dos métodos na orientação a objetos considere:
- I. Os métodos públicos de uma classe definem a interface da classe.
 - II. Os métodos privados de uma classe não fazem parte da interface da classe.



III. O nome dos métodos é a informação reconhecida como a assinatura dos métodos.

Está correto o que consta APENAS em:

- a) I e II.
- b) I e III.
- c) II e III.
- d) II.
- e) I.

55. (FCC - 2012 - TRT - 11ª Região (AM) - Analista Judiciário - Tecnologia da Informação – I) A encapsulação garante que apenas as interfaces necessárias para interação com o objeto estejam visíveis, e atributos internos não sejam acessíveis.

56. (FCC - 2009 - MPE-SE - Analista do Ministério Público – Especialidade Análise de Sistemas) "A utilização de um sistema orientado a objetos não deve depender de sua implementação interna, mas de sua interface." Esta afirmação remete ao conceito de:

- a) herança múltipla.
- b) herança polimórfica.
- c) prototipação.
- d) encapsulamento.
- e) especialização.

57. (FCC - 2009 - TRT - 7ª Região (CE) - Analista Judiciário - Tecnologia da Informação) Considere: A classe Pedido contém um método chamado obter Produtos() que retorna uma lista de produtos pertencentes a um determinado pedido. O código que usa esta classe desconhece completamente como esta lista de produtos é montada. Tudo que interessa é a lista de produtos que o método retorna.

Na essência, o texto explica um dos fundamentos das linguagens OO que é

- a) polimorfismo.
- b) encapsulamento.
- c) dependência.
- d) herança múltipla.
- e) estereotipagem.

58. (FCC - 2011 - INFRAERO - Analista de Sistemas - Desenvolvimento e Manutenção – E) A principal regra prática do encapsulamento é marcar as variáveis de instância como públicas e fornecer métodos de captura e configuração privados.

59. (FCC - 2011 - Nossa Caixa Desenvolvimento - Analista de Sistemas) Um detalhe importante que deve ser especificado para os atributos e operações das classes é a visibilidade. Desta



forma, os símbolos: + (sinal de mais), # (sinal de número), - (sinal de menos) e ~ (til) correspondem respectivamente a:

- a) público, pacote, privado e protegido.
- b) público, protegido, privado e pacote.
- c) privado, protegido, público e pacote.
- d) privado, pacote, público e protegido.
- e) pacote, protegido, privado e público.

60.(FCC - 2007 - TRF - 4ª REGIÃO - Técnico Judiciário - Programação de Sistemas) A proteção de atributos e operações das classes, fazendo com que estas se comuniquem com o meio externo por meio de suas interfaces, define o conceito de:

- a) polimorfismo.
- b) encapsulamento.
- c) herança.
- d) agregação.
- e) especialização.

61.(FCC - 2012 - TCE-AM - Analista de Controle Externo - Tecnologia da Informação) A visibilidade protegida é representada pelo símbolo til (~) e significa que somente os objetos da classe detentora do atributo ou método poderão enxergá-lo ou utilizá-lo.

62.(FCC - 2012 - TST - Técnico Judiciário - Programação) Considere que a classe Pessoa possui 3 métodos que podem ser aplicados aos seus objetos: cadastrar, alterar e excluir. Considere que Aluno e Professor são classes derivadas da classe Pessoa e, por isso, herdam os métodos cadastrar, alterar e excluir, mas estes métodos são sobrescritos na classe Aluno e Professor com implementações bastante distintas, em função dos dados associados a cada um deles.

O exemplo ilustra o conceito de:

- a) hereditariedade.
- b) polimorfismo.
- c) encapsulamento.
- d) abstração.
- e) reusabilidade.

63.(FCC - 2009 - MPE-SE - Analista do Ministério Público – Especialidade Análise de Sistemas) *"...distintas implementações de uma operação de classe e que, no entanto, o nome e os parâmetros dessa operação sejam os mesmos".* Trata-se de:

- a) objeto persistente.
- b) enumeração.
- c) polimorfismo.



- d) subclasse.
- e) pseudo-estado.

64.(FCC - 2011 - TRT - 24ª REGIÃO (MS) - Analista Judiciário - Tecnologia da Informação)

Propriedade pela qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma assinatura mas comportamentos distintos. Trata-se de:

- a) polimorfismo.
- b) herança múltipla.
- c) operação agregada.
- d) multiplicidade.
- e) visibilidade.

65.(FCC - 2012 - TRT - 11ª Região (AM) - Analista Judiciário - Tecnologia da Informação – II)

O polimorfismo garante que objetos possam herdar métodos e atributos de uma superclasse para a geração de uma nova classe.

66. (FCC - 2010 - TRT - 22ª Região (PI) - Técnico Judiciário - Tecnologia da Informação – IV)

No polimorfismo 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.

67.(FCC - 2011 - TRT - 14ª Região (RO e AC) - Técnico Judiciário - Tecnologia da Informação – III)

Polimorfismo é o princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma identificação e mesmo comportamento.

68. (FCC - 2011 - TCE-PR - Analista de Controle – Informática – A)

Polimorfismo pode ser entendido como um conceito complementar ao de herança. Assim, no polimorfismo é possível enviar a mesma mensagem a diferentes objetos e cada objeto responder da maneira mais apropriada para sua classe.

69. (FCC - 2010 - MPE-RN - Analista de Tecnologia da Informação - Engenharia de Software)

Uma operação pode ter implementações diferentes em diversos pontos da hierarquia de classes, desde que mantenham a mesma assinatura. Na orientação a objetos, este é o conceito que embasa:

- a) a multiplicidade.
- b) o encapsulamento.
- c) o protótipo.
- d) o polimorfismo.
- e) o estereótipo.

70.(FCC - 2012 - TJ-PE - Técnico Judiciário - Programador de Computador – III)

A possibilidade de uma operação ter o mesmo nome, diferentes assinaturas e possivelmente diferentes



semânticas dentro de uma mesma classe ou de diferentes classes é chamada de polimorfismo.

71. (FCC - 2009 - TJ-PI - Analista Judiciário - Tecnologia da Informação) Na programação orientada a objetos, é o princípio que oferece a capacidade de um método poder ser implementado de diferentes formas, ou mesmo de realizar coisas diferentes, ou seja, um único serviço pode oferecer variações, conforme se aplique a diferentes subclasses de uma superclasse. O texto acima trata do Princípio de:

- a) Polimorfismo.
- b) Reutilização.
- c) Abstração.
- d) Herança.
- e) Encapsulamento.

72. (FCC - 2009 - TRT - 16ª REGIÃO (MA) - Analista Judiciário - Tecnologia da Informação) Um analista desenvolveu métodos de impressão de dados com a mesma assinatura para três classes de impressoras (jato de tinta, laser e matricial) derivadas de uma mesma superclasse impressora. Tal prática:

- a) aplica o conceito de herança múltipla.
- b) aplica o conceito de polimorfismo.
- c) constitui-se em ferimento à regra de herança.
- d) visa ao aumento da coesão entre os atributos da superclasse.
- e) não é recomendada na orientação a objetos.

73. (FCC - 2009 - PGE-RJ - Técnico Superior de Análise de Sistemas e Métodos) Um comando "abrir" ao provocar diferentes ações em objetos distintos, por exemplo: em uma caixa, porta ou janela, representa figurativamente na orientação a objetos o princípio denominado:

- a) persistência.
- b) polimorfismo.
- c) abstração.
- d) agregação.
- e) herança.

74. (FCC - 2012 - TST - Analista Judiciário - Análise de Sistemas – B) A herança e o polimorfismo são complementares, ou seja, devem ser aplicados em conjunto. A herança existe a partir de classes abstratas que contêm atributos e métodos abstratos. O polimorfismo obriga que as classes-filhas implementem os métodos e atributos desta classe-pai. O acesso aos atributos da classe-pai independe do modificador utilizado.

75. (FCC - 2012 - TCE-AM - Analista de Controle Externo - Tecnologia da Informação – C) O polimorfismo associado à herança trabalha com a redeclaração de métodos previamente



herdados por uma classe. Esses métodos, embora semelhantes, diferem de alguma forma da implementação utilizada na superclasse, sendo necessário, portanto, reimplementá-los na subclasse.

76. (FCC - 2009 - SEFAZ-SP - Agente Fiscal de Rendas - Tecnologia da Informação - Prova 3)

Compartilhamento de atributos e operações genéricas entre diversas classes descendentes de uma classe ancestral remete ao conceito de:

- a) cardinalidade.
- b) encapsulamento.
- c) herança.
- d) agregação.
- e) multiplicidade.

77. (FCC - 2010 - TRT - 22ª Região (PI) - Técnico Judiciário - Tecnologia da Informação – III)

Todas as características de uma superclasse são reusáveis por aquelas classes que são seus subtipos. Assim, uma superclasse é um supertipo de uma ou mais classes.

78. (FCC - 2012 - TRE-CE - Analista Judiciário - Análise de Sistemas – A)

Os conceitos de generalização e especialização da orientação a objetos estão diretamente associados ao conceito de herança.

79. (FCC - 2009 - TRE-PI - Técnico Judiciário - Programação de Sistemas– III)

Herança pode ser compreendida como a propriedade que uma classe tem em legar seus elementos constituintes à sua subclasse.

80. (FCC - 2011 - TRT - 14ª Região (RO e AC) - Técnico Judiciário - Tecnologia da Informação – II)

Na herança cada classe derivada (subclasse) apresenta as características (estrutura e métodos) da classe base (superclasse) e acrescenta a elas o que for definido de particularidade para ela.

81. (FCC - 2011 - TRT - 14ª Região (RO e AC) - Analista Judiciário - Tecnologia da Informação)

A classe Veiculo contém alguns atributos de interesse da classe Aeronave. Todavia, as aeronaves também demonstram interesse em captar atributos e também operações da classe Elemento Turbinado. O enunciado enfatiza o conceito OO de:

- a) polimorfismo.
- b) herança múltipla.
- c) dependência funcional.
- d) realização.
- e) encapsulamento.



- 82. (FCC - 2011 - INFRAERO - Analista de Sistemas - Desenvolvimento e Manutenção – B)** Um relacionamento de herança significa que a superclasse herdará as variáveis de instância e métodos da subclasse.
- 83. (FCC - 2012 - TJ-PE - Técnico Judiciário - Programador de Computador)** A relação de herança permite modelar as similaridades inerentes a uma classe e também as diferenças especializadas que distinguem uma classe de outra.
- 84. (FCC - 2012 - TJ-PE - Técnico Judiciário - Programador de Computador – III)** A herança possibilita que distintas operações na mesma classe tenham o mesmo nome, desde que alterada a assinatura.
- 85. (FCC - 2009 - PGE-RJ - Técnico Superior de Análise de Sistemas e Métodos)** O conceito de Herança, na orientação a objetos, está especificamente associado ao significado de:
- a) cardinalidade.
 - b) generalização.
 - c) multiplicidade.
 - d) encapsulamento.
 - e) composição.
- 86. (FCC - 2008 - TCE-AL - Programador)** Os conceitos de generalização e especialização da orientação a objetos estão diretamente relacionados ao conceito de:
- a) Agregação.
 - b) Associação.
 - c) Encapsulamento.
 - d) Polimorfismo.
 - e) Herança.
- 87. (FCC - 2011 - TCE-PR - Analista de Controle – Informática – D)** No contexto da herança, uma instância da subclasse é, também, uma instância da superclasse.
- 88. (FCC - 2010 - Sergipe Gás S.A. - Analista de Sistemas)** "É o mecanismo pelo qual uma classe pode estender outra classe, aproveitando seus comportamentos e variáveis possíveis." Na programação orientada a objetos esta afirmação refere-se aos conceitos essenciais de:
- a) herança, métodos e atributos.
 - b) subclasse, instância e associação.
 - c) subclasse, encapsulamento e abstração.
 - d) herança, abstração e associação.
 - e) encapsulamento, polimorfismo e interface.



- 89. (FCC - 2012 - TST - Analista Judiciário - Análise de Sistemas – A)** A herança permite que os membros de uma classe, chamada de classe-pai, possam ser reaproveitados na definição de outra classe, chamada de classe-filha. Esta classe-filha tem acesso aos membros públicos e protegidos da classe-pai. O polimorfismo, associado à herança, permite que métodos abstratos definidos em uma classe abstrata sejam implementados nas classes-filhas, podendo estes métodos, nas classes-filhas, apresentar comportamentos distintos.
- 90. (FCC - 2012 - TST - Analista Judiciário - Análise de Sistemas – B)** Atributos e métodos podem ser reaproveitados através da herança, quando uma subclasse herda as características de uma superclasse. Uma subclasse pode ter acesso aos membros de uma superclasse, independente do modificador atribuído. O polimorfismo é um recurso que permite a uma subclasse reimplementar os métodos herdados de uma superclasse, sendo este método abstrato ou não.
- 91. (FCC - 2012 - TCE-AM - Analista de Controle Externo - Tecnologia da Informação – A)** Herança permite o reaproveitamento de atributos e métodos, porém, isso não altera o tempo de desenvolvimento, não diminui o número de linhas de código e não facilita futuras manutenções.
- 92. (FCC - 2012 - TCE-AM - Analista de Controle Externo - Tecnologia da Informação – B)** Em uma aplicação que utiliza herança múltipla, uma superclasse deve herdar atributos e métodos de diversas subclasses. Todas as linguagens de programação orientadas a objeto permitem herança múltipla.
- 93. (FCC - 2012 - TCE-AM - Analista de Controle Externo - Tecnologia da Informação – E)** Em uma relação de herança é possível criar classes gerais, com características compartilhadas por muitas classes. Essas classes não podem possuir diferenças.
- 94. (FCC - 2012 - TRF - 2ª REGIÃO - Técnico Judiciário – Informática – A)** Na hierarquia de classes, se superclasse é uma generalização de subclasses, pode-se inferir que a subclasse é uma especialização de superclasse.
- 95. (FCC - 2012 - TRF - 2ª REGIÃO - Técnico Judiciário – Informática – B)** Numa árvore genealógica de classes, a classe mais baixa herda os atributos e métodos somente da superclasse no nível imediatamente acima.
- 96. (FCC - 2012 - TRT - 11ª Região (AM) - Técnico Judiciário - Tecnologia da Informação)** No contexto de Programação Orientada a Objetos (OOP), sobre a relação de agregação e composição, ou relação todo-parte, considere:
- I. A relação de agregação expressa o ato ou resultado de formar um objeto usando outros objetos como seus componentes.
 - II. Na relação de agregação, as partes só existem enquanto o todo existir.
 - III. Na relação de composição, as partes são independentes da existência do todo.



Está correto o que se afirma em:

- a) I, apenas.
- b) II, apenas.
- c) II e III, apenas.
- d) III, apenas.
- e) I, II e III.

97. (FCC - 2011 - TCE-PR - Analista de Controle – Informática – B) Uma agregação representa um todo que é composto de várias partes e constitui um relacionamento de contenção; se qualquer uma das partes for destruída, as demais partes também o serão.

98. (FCC - 2008 – TRT/18 - Analista de Sistemas) São dois tipos de relacionamento todo-parte:

- a) agregação e composição.
- b) generalização e composição.
- c) generalização e especialização.
- d) composição e dependência.
- e) especialização e agregação.

99. (FCC - 2011 – TRT/RS - Tecnologia da Informação) Na taxonomia utilizada para as formas de polimorfismo são, respectivamente, dois tipos categorizados como universal e dois como Ad Hoc:

- a) Paramétrico e Inclusão; Sobrecarga e Coerção.
- b) Paramétrico e Coerção; Sobrecarga e Inclusão.
- c) Paramétrico e Sobrecarga; Inclusão e Coerção.
- d) Sobrecarga e Inclusão; Paramétrico e Coerção.
- e) Sobrecarga e Coerção; Paramétrico e Inclusão.

100. (FCC - 2012 – TRE/CE - Tecnologia da Informação) Sobre conceitos em programação orientada a objetos (OOP), analise:

I. No polimorfismo ad-hoc, métodos com o mesmo nome e pertencentes à mesma classe, podem receber argumentos distintos, consequentemente alterando a assinatura do método.

II. No polimorfismo paramétrico é possível determinar o método como atributos de objetos são acessados por outros objetos, protegendo o acesso direto aos mesmos através de operações.

III. Na restrição de multiplicidade é possível determinar o número de atributos e operações que uma classe pode herdar de uma superclasse.



Está correto o que consta em:

- a) I, II e III.
- b) I, apenas.
- c) III, apenas.
- d) II e III, apenas.
- e) I e II, apenas.

101. (FGV – 2015 – TCE/SE – Analista de Sistemas) Em POO (Programação Orientada a Objetos), dizer que a classe A estende a classe B é o mesmo que dizer que:

- a) a classe B é subclasse de A;
- b) a classe A é superclasse de B;
- c) a classe A é derivada de B;
- d) a classe B é derivada de A;
- e) as classes A e B são irmãs.

102. (FGV – 2015 – TCE/SE – Analista de Sistemas) Em POO (programação orientada a objetos), dizer que a classe A é superclasse de B é o mesmo que dizer que:

- a) A é derivada de B;
- b) A estende B;
- c) B é derivada de A;
- d) B implementa A;
- e) A implementa B.

103. (IADES – 2011 – PGDF - Analista de Sistemas) Dentro do paradigma de programação orientada a objetos (POO), há um mecanismo utilizado para impedir o acesso direto ao estado de um objeto, restando apenas os métodos externos que podem alterar esses estados. Assinale a alternativa que apresenta o nome deste mecanismo.

- a) Mensagem
- b) Herança
- c) Polimorfismo
- d) Encapsulamento
- e) Subclasse

104. (IADES – 2010 – IADES - Analista de Sistemas) A análise de sistemas no mundo orientado a objeto é feita analisando-se os objetos e os eventos que interagem com esses objetos. O projeto de software é feito reusando-se classes de objetos existentes e, quando necessário, construindo-se novas classes. Análise e projeto orientados a objeto modelam o mundo em termos de objetos que têm propriedades e comportamentos e eventos que disparam operações que mudam o estado dos objetos que interagem entre si. Sobre os



conceitos ou ideias fundamentais da metodologia da análise de sistemas orientada a objeto, assinale a alternativa incorreta.

- a) Uma classe é a implementação de software de um tipo de objeto, podendo ser abstrata (quando possui objetos instanciados a partir dela) ou concreta (quando não possui objetos criados a partir dela).
- b) Um objeto é qualquer coisa, real ou abstrata, a respeito do qual armazenamos dados e os métodos que os manipulam.
- c) Um método de um tipo de objeto referência somente as estruturas de dados desse tipo de objeto. Comparativamente, é similar às funções e procedures do universo da programação.
- d) O encapsulamento é importante porque separa a maneira como um objeto se comporta da maneira como ele é implementado, uma vez que a definição sobre como implementar os conhecimentos ou ações de uma classe não são informadas.

105. (IBFC /Prefeitura de Divinópolis/MG – 2018) Assinale a alternativa que complete correta e respectivamente as lacunas da frase a seguir:

O _____ permite que referências de tipos de classes mais _____ representem o comportamento das classes _____ que referenciam:

- a) polimorfismo - abstratas - concretas
- b) encapsulamento - abstratas - concretas
- c) encapsulamento - concretas - abstratas
- d) polimorfismo - concretas - abstratas

106. (IBFC /TJ-PE – 2017) Em um programa orientado a objetos, verifica-se que a classe X estende a classe Y. Ou seja, pode-se afirmar, pelos preceitos da POO (Programação Orientada a Objetos), que:

- a) a classe X é superclasse de Y
- b) a classe X é uma interface de Y
- c) a classe Y é derivada de X
- d) a classe Y é subclasse de X
- e) a classe X é derivada de Y

107. (IBFC / EMBASA – 2017) Quanto aos fundamentos básicos de programação orientada a objetos, relacione os quatro conceitos abaixo com os respectivos significados mencionados logo em seguida:

CONCEITOS:



- (1) herança.
- (2) método.
- (3) polimorfismo.
- (4) encapsulamento.

SIGNIFICADOS:

- (A) definem as habilidades dos objetos.
- (B) 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.
- (C) é o mecanismo pelo qual uma classe pode estender outra classe ou, ainda, ser estendida de outra classe.
- (D) consiste na separação de aspectos internos e externos de um objeto.

- a) 1C - 2B - 3A - 4D
- b) 1C - 2A - 3B - 4D
- c) 1D - 2A - 3B - 4C
- d) 1A - 2C - 3D - 4B.

108. (IBFC / EBSERH – 2017) Um dos conceitos em Orientação a Objetos é a classe abstrata. Assinale a alternativa que complete correta e respectivamente as lacunas da frase abaixo:

"A classe abstrata é sempre um(a) _____ que não possui _____"

- a) método - instâncias
- b) ator/atriz - atributos
- c) superclasse - instâncias
- d) superclasse - atores/atrizes
- e) método - atributos.

109. (IBFC / EMDEC – 2016) Quanto a Programação Orientada a Objeto identifique a alternativa que representa o mecanismo pelo qual uma classe (sub-classe) pode estender outra classe (super-classe), aproveitando seus comportamentos (métodos) e variáveis possíveis (atributos):

- a) herança
- b) encapsulamento
- c) mensagem
- d) polimorfismo.

110. (IBFC / TRE-AM – 2014) Em programação orientada a objetos significa separar o programa em partes, o mais isoladas possível, tornando o software mais flexível e fácil de modificar:



- a) Herança.
- b) Encapsulamento.
- c) Polimorfismo.
- d) Atributo.

111. (IBFC / PC-RJ – 2013) Quanto à programação orientada a objeto, simplificada, a classe é o conjunto de objetos com características similares. O conjunto de atributos e métodos agregados a um só objeto, que podem ser visíveis ou invisíveis, é denominado de:

- a) Evento
- b) Subclasse.
- c) Estado.
- d) Encapsulamento.
- e) Herança.

112. (IBFC / HEMOMINAS – 2013) Complete a frase a seguir com uma das alternativas abaixo:
"_____ permite que os atributos de classes possam ser declarados como públicos, privados ou protegidos".

- a) Polimorfismo.
- b) Herança.
- c) Abstração.
- d) Encapsulamento.

113. (IBFC / EBSERH – 2017) Consiste no princípio pelo qual 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:

- a) abstração
- b) herança
- c) interface
- d) mensagem
- e) polimorfismo

114. (CESPE - 2010 - TRT - 21ª Região (RN) - Técnico Judiciário - Tecnologia da Informação)
Programa que utilize uma linguagem orientada a objetos (OO), ao incorporar corretamente ao seu funcionamento conceitos como os de encapsulamento, herança e polimorfismo, beneficia-se das características da OO.

115. (CESPE - 2010 - TRT - 21ª Região (RN) - Técnico Judiciário - Tecnologia da Informação)
Além dos conceitos de objeto e classe, o paradigma da orientação a objetos envolve os princípios de:

- a) abstração, encapsulamento, herança e polimorfismo.



- b) abstração, métodos, instâncias e herança.
- c) abstração, encapsulamento, generalização e especialização.
- d) generalização, especialização, herança e polimorfismo.
- e) atributos, métodos, instâncias e mensagens.

- 116. (CESPE - 2010 - Banco da Amazônia - Técnico Científico - Tecnologia da Informação - Arquitetura de Tecnologia)** Objetos têm identidade própria. Isso garante que, mesmo tendo os mesmos valores de variáveis e pertencendo à mesma classe, dois objetos sejam considerados diferentes.
- 117. (CESPE - 2010 - ABIN - Oficial Técnico de Inteligência - Área de Desenvolvimento e Manutenção de Sistemas)** Um objeto apresenta três características básicas, o estado, a identidade e o comportamento. A parte de dados de um objeto é definida por um conjunto de mensagens, e a porção funcional, por um conjunto de atributos.
- 118. (CESPE - 2010 - ABIN - Oficial Técnico de Inteligência - Área de Desenvolvimento e Manutenção de Sistemas)** Objeto é o agrupamento de classes similares que apresentam os mesmos atributos e operações. Na definição de uma classe, é necessário estabelecer a que objeto ela ocorre como instância.
- 119. (CESPE - 2010 - TCU - Auditor Federal de Controle Externo - Tecnologia da Informação - Parte II)** Uma classe pode ser vista como uma descrição generalizada de uma coleção de objetos semelhantes.
- 120. (CESPE - 2011 - FUB - Analista de Tecnologia da Informação - Específicos)** Um objeto possui dados internos e métodos que definem, respectivamente, seu estado atual e seu comportamento. Um objeto pode se comunicar com outros objetos passando mensagens.
- 121. (CESPE - 2010 - MPU - Técnico de Informática)** Uma mensagem enviada a um objeto pode levar à execução de um método que não esteja implementado na classe à qual o objeto pertence.
- 122. (CESPE - 2010 - DETRAN-ES - Analista de Sistemas)** Um dos conceitos em programação orientada a objetos é o de abstração, por meio da qual as características do mundo real podem ser modeladas, por exemplo, mediante o agrupamento de objetos e classes.
- 123. (CESPE - 2013 - INPI - Analista de Planejamento - Desenvolvimento e Manutenção de Sistemas)** Ao se utilizar o encapsulamento, não é necessário saber como ele funciona internamente, apenas como transmite os seus atributos.
- 124. (CESPE - 2011 - TJ-ES - Técnico de Informática - Específicos)** Na programação orientada a objetos, o encapsulamento representa a ação em que diversas implementações de uma operação utilizam vários tipos de parâmetros diferentes.



- 125. (CESPE - 2012 - TRE-RJ - Técnico Judiciário - Programação de Sistemas)** Cada classe pode ter implementações de operação — ou métodos — com denominações únicas. Classes diferentes podem ter métodos com denominações iguais, porém, uma classe não pode ter métodos com denominações iguais e parâmetros diferentes.
- 126. (CESPE - 2012 - MPE-PI - Analista Ministerial - Informática - Cargo 6)** É possível que um mesmo objeto tenha mais de um método com o mesmo nome.
- 127. (CESPE - 2010 - MPU - Técnico de Informática)** Considerando as características do relacionamento entre uma classe e suas subclasses, é correto afirmar que toda implementação de subclasse é polimórfica.
- 128. (CESPE - 2013 - TRE-MS - Programador de computador)** Em programação orientada a objetos, a possibilidade de haver funções de mesmo nome, com funcionalidades similares em classes sem nenhuma relação entre elas, denomina-se:
- a) encapsulamento.
 - b) objeto.
 - c) classe.
 - d) polimorfismo.
 - e) relacionamento hierárquico.
- 129. (CESPE - 2012 - TRE-RJ - Técnico Judiciário - Programação de Sistemas)** As heranças, que são princípios de orientação a objetos, permitem o compartilhamento de atributos e métodos pelas classes e são usadas com o intuito de se reaproveitar código ou comportamento generalizado ou especializar operações ou atributos.
- 130. (CESPE - 2011 - EBC - Analista - Engenharia de Software)** A herança representa uma generalização, dessa forma, por meio de herança é feito o compartilhamento de atributos e operações entre classes, com base em um relacionamento hierárquico.
- 131. (CESPE - 2011 - TRE-ES - Analista - Análise de Sistemas - Específicos)** Em programação orientada a objetos, a herança serve para criar classes que incorporem propriedades e métodos de outras classes. Assim, é possível construir uma classe a partir de outra sem ter de reescrevê-la.
- 132. (CESPE - 2010 - Banco da Amazônia - Técnico Científico - Tecnologia da Informação - Arquitetura de Tecnologia)** A herança é um conceito implementado por todas as linguagens de programação orientadas a objeto. No entanto, algumas delas somente permitem o uso de herança simples, não sendo possível a criação de classes por meio de herança múltipla.
- 133. (CESPE - 2010 - TRT - 21ª Região (RN) - Técnico Judiciário - Tecnologia da Informação)** Uma subclasse, por ser derivada de uma superclasse e possuir todos os atributos da



superclasse, além de atributos específicos, é mais especializada que a superclasse da qual foi derivada.

- 134. (CESPE - 2015 – TRE/GO - Técnico Judiciário - Tecnologia da Informação)** Uma classe abstrata possui instâncias diretas, bem como suas classes descendentes.
- 135. (CESPE - 2013 – MPU – Analista Judiciário)** Se uma subclasse herdar características de duas ou mais superclasses, ocorrerá uma herança múltipla.
- 136. (CESPE - 2013 – INPI – Analista de Planejamento)** Em uma operação de sobrecarga, uma classe derivada pode redefinir operações de sua classe base.
- 137. (CESPE - 2010 – MPU – Analista de Planejamento)** Em uma hierarquia de classes na qual exista herança múltipla, não é possível fazer uso do polimorfismo na implementação das classes.
- 138. (CESPE - 2010 – BASA – Analista de Planejamento)** Na modelagem de classes, a hierarquia entre elas é representada por meio de um relacionamento chamado generalização.
- 139. (CESPE - 2010 – TRE/BA – Analista de Planejamento)** O estado de um objeto é definido pelo conjunto de valores de suas propriedades.
- 140. (CESPE - 2009 – ANAC – Analista de Planejamento)** O uso de mais de uma super classe imediata é usualmente denominado herança múltipla; ter somente uma super classe direta é denominado herança simples.
- 141. (CESPE - 2009 – DETRAN/DF – Analista de Planejamento)** Uma classe é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica.
- 142. (CESPE - 2008 – TRT/BA – Analista Judiciário – Tecnologia da Informação)** Os objetos permitem encapsular dados e funções, que modelam comportamentos e atributos, respectivamente.
- 143. (CESPE - 2008 – MPE/RR – Analista Judiciário – Tecnologia da Informação)** Na orientação a objetos, um objeto tipicamente possui estado e operações definidas. O estado é representado por atributos, e as operações associadas ao objeto podem fornecer serviços a outros objetos. Os objetos são criados de acordo com classes.
- 144. (CESPE - 2004 – TRE/AL – Analista Judiciário – Tecnologia da Informação)** Uma hierarquia de classes é um mecanismo por meio do qual as modificações nos níveis inferiores da hierarquia se propagam de imediato para os níveis superiores.



- 145. (CESPE - 2004 – TRE/AL – Analista Judiciário – Tecnologia da Informação)** O polimorfismo ocorre quando uma subclasse herda atributos e operações de classes diferentes.
- 146. (CESPE - 2012 – TCE/ES – Auditor de Controle Externo)** A implementação de polimorfismo envolve o mecanismo de redefinição de métodos, assim como o conceito de ligação tardia.
- 147. (CESPE - 2010 – TRT/21 – Técnico Judiciário)** Denomina-se herança o princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem utilizar métodos que possuam a mesma assinatura, mas comportamentos diferentes, especializados para cada classe derivada.
- 148. (CESPE - 2016 – TCE/PR – Analista de Sistemas)** Em se tratando de orientação a objetos, o polimorfismo refere-se:
- a) ao reconhecimento do comportamento variado de um método, o que melhora o aproveitamento do código.
 - b) à transmissão dos métodos e atributos de uma classe para suas subclasses, quando ad hoc.
 - c) à variação das saídas de um método com relação às entradas recebidas, caso em que é considerado universal.
 - d) ao uso que um objeto faz dos recursos de outro objeto.
 - e) à utilização de métodos específicos para manipular dados com proteção por meio de encapsulamento.
- 149. (CESPE – 2015 – TJDF – Analista de Sistemas)** Recurso de classes é a propriedade dos objetos que viabiliza a implementação de hierarquia entre objetos.
- 150. (CESPE – 2015 – TJDF – Analista de Sistemas)** A combinação de dados com o código que os manipula em um único objeto é denominada encapsulamento.
- 151. (CESPE – 2016 – FUB – Analista de Sistemas)** Uma das vantagens de se utilizar encapsulamento em orientação a objetos é impedir o acesso direto aos atributos de um objeto.
- 152. (CESPE – 2015 – TCE/RN – Analista de Sistemas)** O processo de herança permite a reutilização de código, como também o reaproveitamento de atributos e métodos. Assim, em aplicações que utilizam herança, a obtenção de polimorfismo é uma possibilidade.



- 153. (CESPE – 2015 – MEC – Analista de Sistemas)** Objetos são definidos como entidades da modelagem de sistemas que armazenam estados com a utilização de atributos dos próprios objetos, sem interação com outros objetos por meio de mensagens.
- 154. (CESPE – 2011 – BRB – Analista de Sistemas)** Para que a interface pública de uma classe seja considerada coesa, é necessário que todos os recursos dessa interface estejam relacionados ao conceito que a classe representa.
- 155. (CESPE – 2011 – TRE/ES – Analista de Sistemas)** Objetos de software interagem e comunicam-se com os outros por meio de mensagens. Por exemplo, quando o objeto A deseja que o objeto B execute um de seus métodos, envia a este uma mensagem. Algumas vezes, o objeto receptor precisa de mais informação para que saiba exatamente o que deve fazer, de modo que essa informação seja transmitida juntamente com a mensagem por meio de parâmetros.
- 156. (CESPE – 2010 – Banco da Amazônia – Analista de Sistemas)** O comportamento de um objeto é definido em sua respectiva classe, por meio da implementação de métodos que são executados quando tal objeto recebe uma mensagem.
- 157. (CESPE – 2010 – Banco da Amazônia – Analista de Sistemas)** A abstração permite, entre outras funcionalidades, identificar e compor objetos complexos e construir estruturas, na forma de classes de objetos, para organizar objetos de diferentes tipos. Porém, conceitos implementados por classes que são construídas com base na abstração não podem ser generalizados nem especializados.
- 158. (CESPE – 2010 – TRE/BA – Analista de Sistemas)** Em programação orientada a objetos, as propriedades que definem a estrutura e o comportamento de um objeto são especificadas para a classe da qual o objeto é instância e são válidas para todos os objetos dessa classe.
- 159. (CESPE – 2017 – SEDF – Analista de Sistemas)** Um objeto define atributos, comportamentos e abstrações comuns compartilhados por um tipo de classe.
- 160. (CESPE – 2016 – FUB – Analista de Sistemas)** O conjunto de valores das características de determinado objeto é denominado estado.
- 161. (CESPE – 2012 – TRE/RJ – Analista de Sistemas)** Polimorfismo consiste em focalizar nos aspectos essenciais inerentes a uma entidade e ignorar propriedades acidentais ou menos importantes. No desenvolvimento de sistemas, isso significa concentrar-se no que um objeto é e no que ele faz antes de se decidir como esse objeto será implementado.
- 162. (CESPE – 2012 – TRE/RJ – Analista de Sistemas)** O polimorfismo de sobrecarga é realizado por meio da aplicação de parâmetros diferentes em operadores distintos com o mesmo nome e semânticas distintas.



- 163. (CESPE – 2016 – FUB – Analista de Sistemas)** Quando uma classe é subclasse de várias superclasses, mas somente herda características de uma classe, a herança é do tipo simples.
- 164. (CESPE – 2017 – TRE/BA - Analista de Sistemas)** Na orientação a objetos, o conceito de polimorfismo é implementado, em algumas linguagens, por meio da técnica de sobrecarga de métodos. Sobre a aplicação desse conceito, é possível afirmar que:
- a) um mesmo método pode estar em classes diferentes, com a mesma assinatura, sem que isso prejudique a sua identificação pelo compilador.
 - b) a identificação da assinatura do método corresponde ao seu nome.
 - c) uma classe derivada da classe-mãe tem vários métodos com a mesma assinatura, e o compilador realiza o primeiro deles.
 - d) a classe derivada da classe-mãe herda os métodos da sua classe-mãe na forma como são implementados.
 - e) cada método tem um nome único na classe derivada da classe-mãe a ser identificado pelo compilador.
- 165. (CESPE – 2017 – TRE/BA - Analista de Sistemas)** A partir de uma classe derivada de uma superclasse, podem-se invocar métodos que tenham a mesma assinatura, mas comportamentos distintos, ou seja, em que haja alteração do funcionamento interno de um método herdado de um objeto pai. Na orientação a objetos, isso é possível por meio de:
- a) polimorfismo.
 - b) abstração.
 - c) encapsulamento.
 - d) namespaces.
 - e) atributos.

ANÁLISE E PROJETO

- 166. (CESPE – 2011 – MPE/TO – Analista de Sistemas)** Entre os diversos diagramas utilizados em análise e projeto orientados a objetos, o diagrama de casos de uso, por procurar representar todas as possíveis situações de utilização do sistema, é considerado o diagrama responsável por mostrar a estrutura estática do sistema.
- 167. (CESPE – 2007 – PETROBRÁS – Analista de Sistemas)** Em um modelo de análise, as classes de fronteira modelam interações entre o sistema e os atores. Cada classe de fronteira deve estar relacionada a um ou mais atores. Pode-se também ter classes de entidade, as quais tipicamente modelam dados persistentes.



- 168. (CESPE – 2007 – PETROBRÁS – Analista de Sistemas)** Em um modelo de análise, as classes de controle podem encapsular controles relacionados a casos de uso e representar lógicas de negócio que não se relacionem a uma classe de entidade específica.
- 169. (CESPE – 2007 – PETROBRÁS – Analista de Sistemas)** Em um modelo de projeto, para que um subsistema seja coeso, seus conteúdos devem ser fortemente relacionados e, para que ele seja fracamente acoplado, é necessário que se minimizem as dependências entre subsistemas.
- 170. (CESPE – 2006 – ANATEL – Analista de Sistemas)** Uma classe na análise orientada a objeto representa uma abstração que pode ser mapeada para mais de uma classe no projeto. As classes na análise podem ser fronteiras, controladoras ou entidades. Uma fronteira modela interações entre o sistema e atores, uma entidade modela apenas objetos persistentes e uma controladora só pode controlar interações entre instâncias de uma mesma classe.
- 171. (CESPE – 2006 – TSE – Analista de Sistemas - A)** Um modelo de análise é menos abstrato que um de projeto e as classes em um modelo de análise não podem ser conceituais. As classes na análise podem modelar objetos persistentes, mas não transientes.
- 172. (CESPE – 2006 – TSE – Analista de Sistemas - B)** Uma importante responsabilidade da análise é definir a arquitetura do sistema, dividindo-o em subsistemas. Um subsistema expõe serviços via interfaces, que devem ser especificadas na análise.
- 173. (CESPE – 2006 – TSE – Analista de Sistemas - C)** Uma classe descreve objetos com as mesmas responsabilidades, relacionamentos, operações, atributos e semântica. As instâncias de uma classe têm, portanto, os mesmos valores para os seus atributos.
- 174. (CESPE – 2006 – TSE – Analista de Sistemas - D)** Um modelo de análise pode realizar casos de uso. A realização de um caso de uso descreve interações entre objetos. Na UML, essas realizações podem ser documentadas via diagramas de colaboração.
- 175. (CESPE – 2012 – IPEA – Analista de Sistemas)** A análise orientada a objetos, o projeto orientado a objetos e a programação orientada a objetos compreendem atividades de engenharia de software voltadas à construção de sistemas orientados a objetos. Nesses sistemas, objetos interagem para prover serviços. No nível de programação, as interações ocorrem via interfaces das classes das quais os objetos são instâncias. Essas interfaces contêm membros públicos das classes.
- 176. (CESPE – 2006 – SERPRO – Analista de Sistemas)** Uma das vantagens dos métodos de análise e projeto orientado a objetos é o aumento do gap conceitual entre os artefatos produzidos nas fases de análise, projeto e implementação.



- 177. (CESPE – 2004 – STJ – Analista de Sistemas)** Com a análise orientada a objetos, busca-se identificar entidades do domínio do problema e caracterizá-las de acordo com sua importância para o problema. Essa atividade tem consequências nas etapas de projeto de software, uma vez que as entidades identificadas darão sustentação para a definição das classes de objetos a serem implementadas.
- 178. (CESPE – 2004 – STJ – Analista de Sistemas)** A definição da linguagem de programação a ser usada na implementação tem igual importância e impacto no projeto e na análise orientados a objetos.



GABARITO

- | | | |
|-------------|-------------|--------------|
| 1. LETRA A | 42. LETRA D | 83. CORRETO |
| 2. LETRA C | 43. CORRETO | 84. ERRADO |
| 3. LETRA A | 44. LETRA B | 85. LETRA B |
| 4. LETRA D | 45. LETRA C | 86. ERRADO |
| 5. LETRA A | 46. CORRETO | 87. CORRETO |
| 6. ERRADO | 47. CORRETO | 88. LETRA A |
| 7. LETRA B | 48. ERRADO | 89. CORRETO |
| 8. LETRA C | 49. CORRETO | 90. ERRADO |
| 9. LETRA C | 50. CORRETO | 91. ERRADO |
| 10. LETRA C | 51. LETRA E | 92. ERRADO |
| 11. LETRA B | 52. LETRA C | 93. ERRADO |
| 12. LETRA B | 53. CORRETO | 94. CORRETO |
| 13. ERRADO | 54. LETRA A | 95. ERRADO |
| 14. LETRA E | 55. CORRETO | 96. LETRA A |
| 15. LETRA D | 56. LETRA D | 97. ERRADO |
| 16. ERRADO | 57. LETRA B | 98. LETRA A |
| 17. LETRA A | 58. ERRADO | 99. LETRA A |
| 18. LETRA D | 59. LETRA B | 100. LETRA B |
| 19. LETRA B | 60. LETRA B | 101. LETRA C |
| 20. LETRA C | 61. ERRADO | 102. LETRA C |
| 21. LETRA C | 62. LETRA B | 103. LETRA D |
| 22. LETRA B | 63. LETRA C | 104. LETRA A |
| 23. ERRADO | 64. LETRA A | 105. LETRA A |
| 24. CORRETO | 65. ERRADO | 106. LETRA E |
| 25. CORRETO | 66. CORRETO | 107. LETRA B |
| 26. LETRA E | 67. ERRADO | 108. LETRA C |
| 27. LETRA C | 68. CORRETO | 109. LETRA A |
| 28. ERRADO | 69. LETRA D | 110. LETRA B |
| 29. CORRETO | 70. CORRETO | 111. LETRA D |
| 30. CORRETO | 71. LETRA A | 112. LETRA D |
| 31. LETRA C | 72. LETRA B | 113. LETRA E |
| 32. ERRADO | 73. LETRA B | 114. CORRETO |
| 33. LETRA C | 74. ERRADO | 115. LETRA A |
| 34. CORRETO | 75. CORRETO | 116. CORRETO |
| 35. LETRA E | 76. LETRA C | 117. ERRADO |
| 36. LETRA B | 77. CORRETO | 118. ERRADO |
| 37. LETRA E | 78. CORRETO | 119. CORRETO |
| 38. LETRA C | 79. CORRETO | 120. CORRETO |
| 39. LETRA D | 80. CORRETO | 121. CORRETO |
| 40. CORRETO | 81. LETRA B | 122. CORRETO |
| 41. LETRA C | 82. ERRADO | 123. CORRETO |



- | | | |
|--------------|--------------|--------------|
| 124. ERRADO | 143. CORRETO | 162. CORRETO |
| 125. ERRADO | 144. ERRADO | 163. ERRADO |
| 126. CORRETO | 145. ERRADO | 164. LETRA A |
| 127. ERRADO | 146. CORRETO | 165. LETRA A |
| 128. LETRA D | 147. ERRADO | 166. ERRADO |
| 129. CORRETO | 148. LETRA A | 167. CORRETO |
| 130. CORRETO | 149. ERRADO | 168. CORRETO |
| 131. CORRETO | 150. CORRETO | 169. CORRETO |
| 132. CORRETO | 151. CORRETO | 170. ERRADO |
| 133. CORRETO | 152. CORRETO | 171. ERRADO |
| 134. ERRADO | 153. ERRADO | 172. ERRADO |
| 135. ERRADO | 154. CORRETO | 173. ERRADO |
| 136. ERRADO | 155. CORRETO | 174. CORRETO |
| 137. ERRADO | 156. CORRETO | 175. CORRETO |
| 138. CORRETO | 157. ERRADO | 176. ERRADO |
| 139. CORRETO | 158. CORRETO | 177. CORRETO |
| 140. CORRETO | 159. ERRADO | 178. ERRADO |
| 141. CORRETO | 160. CORRETO | |
| 142. ERRADO | 161. ERRADO | |



ESSA LEI TODO MUNDO CONHECE: PIRATARIA É CRIME.

Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.



1 Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.



2 Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).



3 Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).



4 Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).



5 Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).



6 Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).



7 Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.



8 O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.



Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.