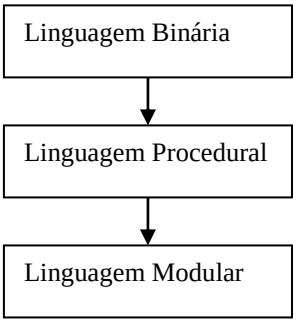
Precursores da POO



Programação Modular divide os programas em vários componentes ou módulos constituintes.
Programação Procedural separa os dados e os procedimentos, os módulos combinam os dois. Um módulo consiste consiste em dados e procedimentos, para manipular esses dados.

OO é a abreviatura de orientado a objetos. OO é um termo geral que inclui qualquer estilo de desenvolvimento que seja baseado no conceito de *objeto* – Uma entidade que exibe características e comportamentos.

A Programação orientada a objetos define seis objetivos sobrepostos para desenvolvimento de Software. A POO se esmera em produzir software que tenha as seguintes características:

Natural – Problemas mais inteligíveis . Em vez de programar em termos de regiões de memória, você pode programar usando terminologia de seu problema em particular.

Confiável – A natureza modular dos objetos permite que você faça alterações em uma parte de seu programa, sem afetar outras partes. Os objetos isolam o conhecimento e a responsabilidade de onde pertencem.

Reutilizável – Pode-se reutilizar prontamente classes orientadas a objetos bem feitas, reutilizando objetos em muitos programas diferentes.

Manutenível – Pode-se corrigir um problema em um lugar. Como uma mudança na implementação é transparente, todos os outros objetos se beneficiarão automaticamente do aprimoramento.

Extensível – O software não é estático, ele deve crescer e mudar com o passar do tempo, para permanecer útil.

Oportuno – *A POO* diminui o tempo do ciclo de desenvolvimento, fornecendo software confiável, reutilizável e facilmente extensível.

O estado de um Objeto é o significado combinado das variáveis internas do objeto.

Uma *variável Interna* é um valor mantido dentro de um objeto.

A *Implementação* define como algo é feito. Em termos de programação, Implementação é código.

Um *Objeto* é uma construção de software que encapsula estado e comportamento. Os objetos permitem que você modele seu software em termos reais de abstração.

Uma *Classe* define os atributos e comportamentos comuns compartilhados por um tipo de objeto. Os objetos de certo tipo ou classificação compartilham os mesmos comportamentos e atributos. As classes atuam de forma muito parecida com um cortador de molde ou biscoito, no sentido de que você use uma classe para criar ou *instanciar um objeto*.

Atributos são características de uma classe visíveis externamente. A cor dos olhos e a cor dos cabelos são exemplos de atributos.

OBS: Um objeto pode expor um atributo fornecendo um link direto a alguma variável interna ou retornando o valor através de um método.

Comportamento é uma ação executada por um objeto quando passada uma mensagem ou em resposta a uma mudança de estado: é algo que o objeto faz. Um objeto pode pode exercer o *comportamento* do outro, executando uma

operação sobre esse objeto. Poderá ser visto os termos *Chamada de Método, Chamada de Função, ou Passar uma Mensagem*, usados em vez de executar uma operação.

Construtores são métodos usados para inicializar objetos durante sua instanciação. Chama-se criação de objetos de *Instanciação* porque ela cria uma instância do objeto na classe.

Acessores dão acesso aos dados internos de um objeto. Entretanto, os acessores ocultam o fato de os dados estarem em uma variável, em uma combinação de variáveis ou serem calculados. Os acessores permitem que você mude ou recupere o valor e têm 'efeitos colaterias' sobre o estado interno.

Os *Mutantes* permitem que você altere o estado interno de um objeto.

Os *Objetos* se comunicam uns com os outros através de *mensagens*. As mensagens fazem com que um objeto realize algo. *Passar uma Mensagem* é o mesmo que chamar um método para mudar o estado do objeto ou exercer um comportamento.

Os três pilares da Programação Orientada a Objetos são: Encapsulamento, Herança e polimorfismo.

ENCAPSULAMENTO

Encapsulamento é a característica da OO de ocultar partes independentes da implementação. Permite que você construa partes ocultas da implementação do software, que atinjam uma funcionalidade e ocultam os detalhes de implementação do mundo exterior; O encapsulamento forma a base da Herança e do Polimorfismo.

A *Interface* lista os serviços fornecidos por um componente. A interface é um contrato com o mundo exterior, que define exatamente o que uma entidade externa pode fazer com o objeto. Uma interface é o painel de controle do objeto.

A *Implementação* define como um comportamento realmente fornece um serviço. A implementação define os detalhes internos dos componentes.

A maioria das linguagens OO suporta três níveis de acesso:

Público – Garante o acesso a todos os objetos.

Protegido – Garante o acesso à instância, ou seja, para aquele objeto, e para todas as subclasses.

Privado – Garante o acesso apenas para a instância, ou seja, para aquele objeto.

As três características do *Encapsulamento Eficaz* são:

Abstração – processo de simplificar um problema difícil.

Ocultação da Implementação – tem a vantagem de proteger seu objeto de seus usuários e protege os usuários de seu objeto do próprio objeto.

Divisão de Responsabilidade – significa que cada objeto deve executar uma função – sua responsabilidade – e executa-la bem.

TAD (Abstract Data Type – Tipo Abstrato de Dados) é um conjunto de operações sobre esses dados. Os TAD's permitem que você defina novos tipos na linguagem, ocultando dados internos e o estado, atrás de uma interface bem definida. Essa interface apresenta o TAD como uma única unidade atômica.

Tipo define espécies de valores que você pode usar em seus programas. Um tipo define o domínio a partir do qual seus valores válidos podem ser extraídos. Para inteiros positivos, são os números sem partes fracionárias e que são maiores ou iguais a zero. Para tipos estruturados, a definição é mais complexa. Além do domínio, a definição de tipo inclui quais operações são válidas no tipo e quais são os seus resultados.

OBS: Lembre-se que adicionar uma nova classe em seu sistema é o mesmo que criar um novo Tipo.

Um *objeto de primeira classe* é aquele que pode ser usado exatamente da mesma maneira que um tipo interno.

Um *objeto de segunda classe* è um tipo de objeto que você pode definir, mas não necessariamente usar, como faria com um tipo interno.

O *código fracamente acoplado* é independente da implementação de outros componentes.

O *código fortemente acoplado* é fortemente vinculado à implementação de outros componentes.

Código dependente é dependente da existência de determinado tipo. O código dependente é inevitável. Entretanto, existem graus para dependência aceitável e para a superdependência.

Encapsulamento Efetivo é a abstração mais ocultação da implementação mais responsabilidade.

Construtores Noargs são construtores que não recebem nenhum argumento.

Objeto Imutável é aquele cujo objeto não muda, uma vez construído (Constante).

OBS: Uma linguagem orientada a objetos PURA suporta a noção de que tudo é um objeto, já uma OO na considera tudo um objeto.

Um pacote é um objeto cujo único propósito é conter outro objeto ou primitiva. Um pacote fornecerá qualquer número de métodos para obter e manipular o valor possuído.

Variáveis de Classe - são variáveis que pertencem à classe e não a uma instância específica, são compartilhadas entre todoas as instâncias da classe.

Métodos de Classe são métodos que pertencem à classe e não a uma instância específica. A operação executada pelo método não é dependente do estado de qualquer instância.

HERANÇA

Herança - é um mecanismo que permite a você basear uma nova classe na definição de uma classe previamente existente. Sua nova classe herda todos os atributos e comportamentos presentes na classe previamente existente.

Herança é um mecanismo que permite estabelecer relacionamentos "é um" entre classes. Esse relacionamento também permite que uma subclasse herde os atributos e comportamentos de uma Superclasse.

Delegação é o processo de um objeto passar uma mensagem para outro objeto, para atender algum pedido.

"É um" descreve o relacionamento em que uma classe é considerada do mesmo tipo de outra.

"Tem um" descreve o relacionamento em que uma classe contém uma instância de outra classe.

Composição significa que uma classe é implementada usando-se variáveis internas(chamadas de variáveis membro), que contêm instâncias de outras classes.

Hierarquia de Herança - é um mapeamento do tipo árvore de relacionamentos que se formam entre classes como resultado de herança.

Classe Filha - é a classe que está herdando, também conhecida como Subclasse.

Classe Progenitora ou Mãe – é a classe da qual a filha herda diretamente, também conhecida como Superclasse.

Uma Classe construída através da herança pode ter três tipos importantes de métodos e atributos:

Sobreposto – a nova classe herda o método ou atributo da progenitora, mas fornece uma nova definição.

Novo – a nova classe adiciona um método ou atributo completamente novo.

Recursivo – a nova classe simplesmente herda um método ou atributo da progenitora.

Sobrepor é o processo de uma filha pegar um método que aparece na progenitora e reescrevê-lo para mudar o comportamento do método. A sopreposição de um método é conhecida como *Redefinição* de um método.

Ao considerarmos a sobreposição de um método ou atributo, é importante perceber que nem todos os métodos e atributos estão disponíveis para sua filha sobrepor. A maioria das linguagens orientadas a objetos tem alguma noção de controle de acesso. Esses níveis de acesso caem em três categorias:

Privado – um nível de acesso que restringe o acesso apenas à classe.

Protegido: um nível de acesso que restringe o acesso à classe e às filhas.

Público: um nível de acesso que permite o acesso a todos e a qualquer um.

Os métodos e atributos protegidos são aqueles aos quais você deseja que apenas as subclasses tenham acesso. O nível privado impede que qualquer outro objeto chame o método, exceto quanto ao próprio objeto.

Ao todo, existem três maneiras principais de usar herança:

Para reutilização de implementação – através da composição, a herança torna o código automaticamente disponível, como parte da nova classe. Fazendo a classe nascer com funcionalidade.

Para diferença – permite que voc~e programe especificando apenas como uma classe filha difere de sua classe progenitora.

Para substituição de tipo – Permite que você descreva relacionamentos com capacidade de substituição.

Programação por diferença significa herdar uma classe e adicionar apenas o código que torne a nova classe diferente da classe herdada.

Especialização é o processo de uma classe filha ser projetada em termos de como ela é diferente de sua progenitora. Quando tudo estiver dito e feito, a definição de classe filha incluirá apenas os elementos que a tornem diferente de sua progenitora.

Ancestral é uma classe que aparece na hierarquia de classe antes da progenitora.

Descendente é toda classe que aparece depois dela na hierarquia de classes.

*Classe Raiz (*Também conhecida como Classe Base) é a classe superior da hierarquia de herança.

Classe Folha é uma classe sem filhas.

OBS: é importante notar que as descendentes refletirão as alterações feitas nas ancestrais.

Herança Múltipla – permite que um objeto herde diretamente de mais de uma classe.

Um Subtipo é um tipo que estende outro tipo através da herança.

Método declarado, mas não implementado, é chamado de *Método Abstrato.* Somente classes abstratas podem ter métodos abstratos.

POLIMORFISMO

Ter muitas formas. Em termos de programação, muitas formas significa que um único nome pode representar um código diferente, selecionado por algum mecanismo automático, assim o polimorfismo permite que um único nome expresse muitos comportamentos diferentes.

As linguagens podem ser *Polimórficas (*com polimorfismo) *ou Monomórficas (*sem polimorfismo).

Variável Polimórfica é uma variável que pode conter muitos tipos diferentes.

Polimorfismo de Inclusão ou Puro permite que você trate objetos relacionados genericamente. Torna mais fácil adicionar novos subtipos em um programa, pois não será preciso adicionar um método especificamente para esse novo tipo.

Polimorfismo Paramétrico permite que você crie métodos e tipos genéricos. Assim como o Polimorfismo de inclusão, os métodos e tipos genéricos permitem que você codifique algo uma vez e faça isso trabalhar com muitos tipos diferentes de argumentos.

Permite que você programe métodos genéricos retirando a referência de declarações de tipo de parâmetro até o momento da execução.

A *Sobreposição* é um tipo importante de polimorfismo.

Os métodos abstratos são freqüentemente referidos como *métodos Adiados*, pois você retarda a definição para classes descendentes.

A *Sobrecarga* também conhecida como *polimorfismo Ad-Hoc*, permite que você use o mesmo nome de método para muitos métodos diferentes. Cada método diferem apenas no número e no tipo de seus parâmetros.

Conversão e *Sobrecarga* freqüentemente andam lado a lado. A conversão também pode fazer com que um método pareça como se fosse polimórfico. A coversão ocorre quando um argumento de um tipo é convertido para o tipo esperado.

Para se ter um Polimorfismo eficaz é preciso ter encapsulamento e herança eficientes.

UML (Unified Modeling Language)

Linguagem de modelagem padrão, que consiste em várias notações gráficas que você pode usar para descrever a arquitetura inteira de seu software.

Notação gráfica utilizada para descrever a arquitetura inteira de seu software, incluindo várias regras para distinguir entre desenhos corretos e incorretos, são regras que determinam a UML como uma linguagem de modelagem;

Fornece um vocabulário comum que os desenvolvedores podem usar para transmitir seus projetos.

OBS: Linguagem de modelagem é diferente de processo de metodologia.

Uma *metodologia ou processo* define um procedimento para projetar o software. As linguagens de modelagem capturam esse projeto graficamente, ou seja, freqüentemente contém uma linguagem de modelagem.

OBS: UML não é única, mas é a padrão. Documentação também é útil para alguém que não conheça a linguagem de implementação.

Notação básica de Classe

Uma caixa representa classe. A caixa do centro contém todos os atributos e a inferior contém todas as operações.

A UML faz diferença entre operação e método, onde uma operação e um serviço que você solicita de qualquer objeto de uma classe, enquanto um

método é uma implementação específica da operação.

Os caracteres -, #, + transmitem a visibilidade de um atributo ou de uma operação.

(-) significa privado, (#) significa protegido, (+) significa público.

Notação Avançada de Classe

Tem a função de criar modelos mais descritivos.

Estereótipo é um elemento da UML que permite que se amplie o vocabulário da própria linguagem UML. Consiste em uma palavra ou frase incluídas entre sinais de menor e maior duplos (<<>>), colocando um estereótipo ao lado de um elemento existente.

A UML fornece uma notação para transmitir que uma classe é abstrata, escrita em itálico.

Modelando um Relacionamento de Classe

Um *relacionamento* descreve como as classes interagem entre si. Na UML, um relacionamento é uma conexão entre dois ou mais elementos da notação.
Três tipos de alto nível de relacionamentos de objetos: Dependência, Associação, Generalização.

Dependência – é o relacionamento mais simples entre objetos. A dependência indica que um objeto depende da especificação de outro objeto. Se a especificação mudar, você precisará atualizar o objeto dependente.
Especificação é uma maneira diferente de dizer interface ou comportamento.

OBS: Em POO tenta-se minimizar ao máximo possível as dependências.

Associação – Indica que um objeto contém , ou que está conectado a outro objeto.

O nome da Associação é um nome que descreve o relacionamento.
O *papel da associação* é a parte que um objeto desempenha em um relacionamento.

A *Multiplicidade* indica quantos objetos podem tomar parte na instância de uma associação.

Só se deve modelar associações quando um objeto contiver outro objeto, ou quando um objeto usa outro.
Existem dois tipos de associação, são os de *Agregação e Composição*.

Agregação é um tipo especial de associação que modela o relacionamento 'tem um' de relacionamentos Todo/Parte entre pares.

Relacionamento Todo/Parte descreve o relacionamento entre objetos onde um objeto contém outro.
OBS: Pares significa que um objeto não é mais importante do que o outro.

Você usa agregação quando o objetivo de seu modelo for descrever a estrutura de um relacionamento de pares.

Composição não é um relacionamento entre pares. Os objetos não são independentes uns dos outros.
Ao contrário da agregação, uma composição não modela relacionamentos Todo/Parte de pares.

Generelização – Indica um relacionamento entre o geral e o específico, ou seja, a herança. Se existe esse relacionamento, se sabe que pode substituir uma classe filha pela progenitora.

Introdução à AOO

É uma estratégia orientada a objetos para entender um problema. Se usa AOO para ajudar a entender o núcleo do problema que deseja resolver.

O Processo de desenvolvimento de software

As metodologias de software definem uma maneira comum de encarar o desenvolvimento de software. Uma metodologia freqüentemente conterá uma linguagem de modelagem (como UML) e um processo.

Um Processo de Software mostra os vários estágios do desenvolvimento de software.

Processo de Cascata

Etapas: Análise de Requisitos, Projeto, Implementação, teste.
Neste processo existe a realização etapa por etapa, mas sem volta para uma etapa anterior para especializar.

Processo Iterativo

Este processo permite alterações em qualquer ponto do processo de desenvolvimento. Permite alteração adotando uma estratégia iterativa e incremental, para desenvolvimento do software.

Estratégia iterativa – permite que você continuamente volte e refine cada estágio do desenvolvimento.

Estratégia Incremental – Divide o trabalho de desenvolvimento em várias iterações pequenas.

Sistemas

È o termo da AOO para um conjunto de objetos que interagem.
Requisitos

São os recursos ou características que o sistema deve ter para resolver determinado problema.

Um modo de descobrir esses usos é através da Análise de casos de uso. Através da análise será definido vários casos de uso. Um caso de uso descreverá como um usuário vai interagir com o sistema.

Análise de Casos de Uso

É o processo de descoberta de casos de uso através da criação de cenários e histórias com usuários em potencial ou existentes de um sistema.

Um caso de uso descreve a interação entre o usuário do sistema e o sistema – como o usuário utilizará o sistema do seu próprio ponto de vista.

Ator

É tudo que interage com o sistema. Pode ser um usuário humano ou outro sistema de computador.

Variante

Uma variante de caso de uso é uma versão especializada de outro caso de uso mais geral.

Cenário

É uma sequência ou fluxo de eventos entre o usuário e o sistema.

Condições Prévias

São aquelas condições que devem ser satisfeitas para que um caso de uso comece. Condições posteriores são os resultados de um caso de uso.b

Diagramas de Caso de Uso

Diagrama de Interação – Ajudam a modelar os relacionamentos entra casos de uso. Ajudam também a capturar as interações entre os vários atores participantes do sistema, e divide-se em:

Diagrama de Seqüência – Demonstra arelação entre usuários do sistema.
Diagrama de Colaboração – Chama a atenção para a seqüência de eventos com o passar do tempo.

Diagramas de Atividades

Ajudam a modelar processos que podem ser executados em paralelo.

Modelos de Domínio

Lista os objetos que você precisa para modelar corretamente o sistema.

POO (Projeto Orientado a objetos)

POO é o processo de construir o modelo de objeto de uma solução. Dito de outra maneira, POO é o processo de dividir uma solução nem vários objetos constituintes.

O modelo de objeto é o projeto dos objetos que aparecem na solução de um problema. O modelo final de objeto pode conter muitos objetos não encontrados no domínio. O modelo de objeto descreverá as várias responsabilidades, relacionamentos e estrutura do objeto.

O processo de POO o ajuda a descobrir como você vai implementar a análise que completou durante a AOO. Principalmente, o modelo de objeto que conterá as classes principais do objeto, suas responsabilidades e uma definição de como elas vão interagir e obter suas informações.

Usando os termos do modo de construção, você usa o POO para criar as cópias heliográficas de seu programa.

Os objetos freqüentemente delegarão trabalho para outros objetos. Através de seu projeto, você precisa identificar essas colaborações.

Um objeto delega trabalho para colabores.

Colaboração é o relacionamento onde os objetos interagem para realizar o mesmo propósito.

Cartão CRC é uma maneira de distribuir responsabilidades e colaborações é através do uso de cartões CRC (Classe responsabilidade Colaboração)

Esses cartões ajudam a definir o propósito de um objeto, chamando a atenção para as responsabilidades do objeto.

Ao desenvolver os cartões CRC, é preciso garantir que cada classe tenha apenas duas ou três responsabilidades principais. Se você tiver mais responsabilidades, deverão dividir a classe em duas ou mais classes separadas.

Ponto de Interação é qualquer lugar onde um objeto use outro.

Um Agente faz a mediação entre dois ou mais objetos para atingir algum objetivo.

Reutilizações de Projeto utilizando padrões de projeto

Um padrão de Projeto é um conceito de projeto reutilizável. Este consiste em quatro elementos:

O nome do Padrão – Identifica exclusivamente cada padrão do projeto.
O Problema
A Solução
As Conseqüências

Padrão Adapter

Um adaptador é um objeto que transforma a interface de outro objeto.

Padrão Proxy

Um proxy é um substituto ou lugar reservado que intermédia o acesso ao objeto de interesse real. Para todos os efeitos, O substituto é indistinguível do objeto real que intermédia.

Padrão Iterator

Uma interface Iterator fornece uma interface genérica para iteragir sobre uma coleção.

Construindo Software Confiável através de testes

Um Caso de Teste é o bloco de construção básico do processo de teste. O processo de teste executa vários casos de teste para poder validar completamente um sistema. Cada caso de teste consiste em um conjunto de entradas e saídas esperadas. O teste executará um caminho específico através do sistema (caixa branca) ou testará algum comportamento definido (caixa preta).

O teste de Caixa Preta testa se o sistema funciona conforme o esperado. Dada uma entrada específica, o teste de Caixa Preta testa se a saída ou comportamento correto, visível externamente, resulta conforme definido pela especificação da classe ou do sistema.

No teste de caixa Branca, os testes são baseados unicamente na implementação de um método. Os testes de caixa branca tentam atingir 100% de cobertura de código.

Um teste de Unidade é o dispositivo de teste de nível mais baixo. Um teste de unidade envia uma mensagem para um objeto e depois verifica se ele recebe o resultado esperado do objeto. Um teste de unidade verifica apenas um recurso por vez.

Um Teste de Integração verifica se dois ou mais objetos funcionam em conjunto corretamente.

Um teste de sistema examina o sistema inteiro. Um teste de sistema verifica se um sistema funciona conforme mencionado nos casos de uso e se ele pode manipular normalmente situações incomuns e inesperadas.

Os Testes de Regressão – examinam as alterações nas partes do sistema que já foram validadas. Quando uma alteração é feita, a parte que foi alterada, assim como todas as partes dependentes, deve se novamente testada.

Uma Estrutura è um modelo de domínio reutilizável. A estrutura contém todas as classes comuns a um domínio inteiro de problemas e serve como a base para um aplicativo específico no domínio.

Uma Classe Anônima é uma classe que não tem nome. As classes anônimas não têm nome porque elas são simplesmente definidas ao serem instanciadas. Elas não são declaradas em um arquivo separado ou como uma classe interna.

O Acessório de teste prepara o conjunto de objetos sobre os quais um caso de teste atuará. Os acessórios também são convenientes, pois eles permitem que você compartilhe o mesmo acessório dentre um conjunto inteiro de casos de teste, sem ter de duplicar o código.

Um Objeto Falsificado é um substituto simplista de um objeto real. Ele é chamado de objeto falsificado, porque o objeto foi falsificado para propósitos de teste. Embora o objeto falsificado possa ter uma implementação simplista, ele pode conter funcionalidade extra para ajudar nos testes. São também conhecidos como Simuladores.
Esses objetos são intimamente relacionados aos objetos stubs. Objetos stubs realizam apenas trocas de mensagens enquanto que os objetos falsificados se diferem no sentido de que eles realmente executam alguma função, em vez de simplesmente aceitarem uma chamada e retornarem algum valor prévio.

Análise da Prova anterior

69 – Em Programação Orientada a Objetos (POO), podemos definir um **Objeto** formalmente como sendo:

a) Uma construção de software que encapsula estado e comportamento
b) Uma estrutura de dados que combina vetores e matrizes
c) Um código estruturado contendo condicionais e iterações
d) Uma função que retorna como resultado um inteiro

77 – A Programação Orientada a Objetos (POO) define seis objetivos sobrepostos para desenvolvimento de software. Tais objetivos representam as características que um software produzido por ela deve ter. Qual das alternativas abaixo **não** é um desses objetivos?

a) Produz software reutilizável.
b) Produz software confiável.
c) Produz software estático.
d) Produz software manutenível.

78 – Em Programação Orientada a Objetos (POO), as classes possuem alguns métodos que recebem nomes especiais. Destes podemos citar os construtores. Tais métodos são responsáveis por

a) permitir que se altere o estado interno de um objeto.
b) inicializar objeto durante sua instanciação.
c) calcular a quantidade de memória alocada para um objeto.
d) dar acesso aos dados internos de um objeto.

79 – Em Programação Orientada a Objetos (POO), as classes possuem alguns métodos que recebem nomes especiais. Destes podemos citar os acessores. Tais métodos são responsáveis por

a) permitir que se altere o estado interno de um objeto.
b) inicializar as variáveis internas de um objeto durante a instanciação.
c) calcular a quantidade de memória alocada para um objeto.
d) dar acesso aos dados internos de um objeto.

80 – **Não** é pilar da Programação Orientada a Objetos (POO):

a) encapsulamento
b) herança
c) polimorfismo
d) abstração

75 – Em Programação Orientada a Objetos (POO), sobre uma **classe** é **incorreto** afirmar que

a) define atributos e comportamentos comuns compartilhados por um tipo de objeto.
b) é usada para instanciar objetos.
c) defina quais mensagens seus objetos respondem.
d) é um valor mantido dentro de um objeto.

81 – Das respostas abaixo, qual lista os três pilares da Programação Orientada a Objetos (POO)?

a) herança, encapsulamento e construtores
b) abstração, polimorfismo e encapsulamento
c) encapsulamento, herança e polimorfismo
d) acessores, Construtores e Encapsulamento

82 – Em Programação Orientada a Objetos (POO), o encapsulamento transforma seus objetos em componentes plugáveis. Para que outro objeto use seu componente, ele só precisa saber como usar a interface pública do componente. Tal independência tem três vantagens importantes. Das respostas abaixo, qual delas **não** é uma dessas vantagens?

a) Poder reutilizar o objeto em qualquer parte.
b) Vincular seus objetos a um programa em particular.
c) Tornar transparentes as alterações no objeto.
d) Não causar efeitos inesperados entre o objeto e o restante do programa.

83 – Em Programação Orientada a Objetos (POO), na parte de herança, falamos também de especialização. Sobre a especialização é incorreto afirmar que

a) uma classe filha se especializa em relação a sua progenitora adicionando novos atributos e métodos a sua interface
b) pode redefinir atributos e métodos previamente existentes.
c) permite que se remova da classe filha atributos e métodos herdados da classe mãe.
d) é o processo de uma classe filha ser projetada em termos de ser diferente de sua progenitora.

85 – Polimorfismo significa muitas formas. Em Programação Orientada a Objetos (POO), polimorfismo significa que

a) precisamos necessariamente de um nome diferente por método, para termos um código diferente.
b) um único nome pode receber apenas um parâmetro na sua declaração.
c) podemos escrever o método em diferentes linguagens.
d) um único nome pode representar um código diferente, selecionado por algum mecanismo automático.

86 – Qual das alternativas abaixo é um exemplo de relação superclasse – subclasse, conforme definições em Programação Orientada a Objetos (POO)?

a) lei – Lei Pelé
b) torta de maçã - maçã
c) colméia – abelha
d) alcatéia – lobo

87 – Qual das alternativas abaixo **não** é um exemplo de relação superclasse – subclasse, conforme definições em Programação Orientada a Objetos (POO)?

a) lei – Lei Pelé
b) animal – gato
c) colméia – abelha
d) abelha – abelha rainha

90 - Em Programação Orientada a Objetos (POO), definimos o termo construtores como sendo

a) métodos usados para inicializar objetos durante sua instanciação.
b) aqueles que dão acesso aos dados internos de um objeto.
c) aqueles que permitem que se altere o estado interno de um objeto.
d) uma construção de software que encapsula estado e comportamento.

RESOLUÇÃO

Resposta: A

Construtores são métodos utilizados para inicializar objetos durante sua instanciação. A opção "B" está se referindo ao termo ACESSORES. A opção "C" está se referindo ao termo MUTANTES, e a opção "D" está se referindo à definição de Objetos. A Resolução encontra-se nas páginas 7, 10 e 11 de SINTES, Anthony. Aprenda programação orientada a objeto em 21 dias. São Paulo: Makron Books, 2002.

92 - Indique a opção que completa corretamente a lacuna da assertiva a seguir

O Polimorfismo _____ permite que se tratem objetos relacionados genericamente.

a) paramétrico
b) de inclusão
c) sobreposição
d) sobrecarga

91 - Em Programação Orientada a Objetos (POO), quando definimos um método e atributo recursivo, significa que

a) a nova classe não herda um método ou atributo da progenitora, mas fornece uma nova definição.
b) a nova classe adiciona um método ou atributo completamente novo.
c) a nova classe simplesmente herda um método ou atributo da progenitora.
d) a nova classe herda o método ou atributo da progenitora, mas não fornece uma nova definição.

RESOLUÇÃO

RESPOSTA: C

Um método ou atributo recursivo é definido na progenitora ou em alguma outra ancestral, mas não na filha. A Alternativa "B" refere-se ao método ou atributo NOVO. A alternativa "A" e "D" poderiam estar se referindo ao método ou atributo SOBREPOSTO se a alternativa A não estivesse negando a herança de um método ou atributo e se a alternativa "D" não estivesse negando o fornecimento de uma nova definição. (SINTES, Anthony. Aprenda programação orientada a objeto em 21 dias. São Paulo: Makron Books, 2002)

94 - Em Programação Orientada a Objetos (POO), mais especificamente em UML (Unified Modeling Language), um processo de software mostra os vários estágios do desenvolvimento de software, e um exemplo familiar de processo de software é o processo de Cascata. Qual dos itens abaixo **não** corresponde a um dos estágios do processo de Cascata?

a) Manutenção
b) Análise de requisitos
c) Projeto
d) Teste

RESOLUÇÃO

Resposta: A

Os estágios que constituem o processo de Cascata são: Análise de Requisitos, Projeto, Implementação e Teste. Portanto a Manutenção não faz parte dos estágios do processo de Cascata. (SINTES, Anthony. Aprenda programação orientada a objeto em 21 dias. São Paulo: Makron Books, 2002, pág. 194)

Resposta: B

O Polimorfismo de inclusão, às vezes chamado de polimorfismo puro, permite que se tratem os **objetos** relacionados genericamente. Já o Polimorfismo paramétrico afeta os **métodos,** e não o objeto, como o Polimorfismo de Inclusão o faz. O Polimorfismo de sobrecarga permite que se use o mesmo nome de método para muitos métodos diferentes. SINTES, Anthony. Aprenda programação orientada a objeto em 21 dias. São Paulo: Makron Books, 2002)

93 - Em Programação Orientada a Objetos (POO), quais são os tipos de relacionamento de objetos de alto nível reconhecidos pela UML (Unified Modeling Language)?

a) Esteriótipo, Abstração e Generalização
b) Propriedades, Métodos e Eventos
c) Herança, Polimorfismo e Abstração
d) Dependência, Associação e Generalização

RESOLUÇÃO

Resposta: D

A UML (Unified Modeling Language) reconhece três tipos de alto nível de relacionamento de objetos: Dependência, Associação e Generalização. A Dependência é o relacionamento mais simples entre os objetos. A Dependência indica que um objeto depende da especificação de outro objeto. A Associação indica que um objeto contém, ou que está conectado a um outro objeto. Um relacionamento de generalização é o relacionamento entre o geral e o específico. A resolução encontra-se nas páginas 183, 184 188 da bibliografia sugerida. (SINTES, Anthony. Aprenda programação orientada a objeto em 21 dias. São Paulo: Makron Books, 2002)