Optei por utilizar a playlist do Gustavo Guanabara, para disponibilizar para os alunos como um material extra. Inclusive gosto muita das aulas dele e quando fiz a matéria de Programação Orientada a Objetos, estudava a partir dos vídeos dele, além das aulas presenciais.

Link para a Playlist:

[**PlayList POO**](https://www.youtube.com/playlist?list=PLHz_AreHm4dkqe2aR0tQK74m8SFe-aGsY)

Disponibilizarei os vídeos 1 ao 31, onde serão separados os conteúdos práticos e teóricos.

Lista de Boas e Más práticas na Programação Orientada a Objetos:

**Más práticas na Programação Orientada a Objetos:**

1 – A declaração de mais de uma classe public em um mesmo arquivo resulta em um erro de compilação;

2 – Caso o número de argumentos em uma chamada de método não corresponder ao número de parâmetros em sua declaração de método ocorrerá um erro de compilação;

3 - Ocorrerá um erro de compilação se o tipo de qualquer argumento em uma chamada de método não for consistente com o tipo do parâmetro correspondente na declaração do método;

4 – A tentativa de utilizar uma variável local não inicializada resultará em um erro de compilação.

5 – Um método que realiza uma única tarefa é mais fácil de testar e depurar do que aquele que realiza muitas tarefas;

6 – Declarar parâmetros de método do mesmo tipo como float a, b ao invés de float a, float b é um erro de sintaxe;

7 - É um erro de sintaxe dividir uma String em linhas, caso seja necessário, faça a divisão de uma String em várias unidades menores e utilize a concatenação de String para obter a String desejada;

8 – Confundir o operador + utilizado na concatenação de string com o operador + utilizado para a adição de resultados pode resultar em resultados estranhos. O Java avalia operadores da esquerda para a direita. Fique atento quanto a esta questão;

9 – Converter um valor do tipo primitivo em um outro tipo primitivo pode alterar o valor se o novo tipo não for válido;

10 – Um erro de compilação ocorre quando uma variável local é declarada mais de uma vez em um método;

11- Declarar métodos sobrecarregados com listas de parâmetros idênticas é um erro de compilação independentemente de os tipos de retorno serem diferentes;

12 – Um índice precisa ser um valor int ou um valor de um tipo que possa ser promovido a int, caso contrário ocorrerá um erro de compilação;

13 – Em uma declaração de array, especificar o número de elementos entre os colchetes da declaração, por exemplo (int[5], c ) é um erro de sintaxe;

14 – Declarar múltiplas variáveis de array em uma única declaração pode levar a erros sutis;

15 – Atribuir um valor a uma variável constante depois de ela ter sido inicializada é um erro de compilação;

16 – Tentar utilizar uma constante antes de ela ser inicializada é um erro de compilação;

17 – Uma tentativa por parte de um método que não é membro de uma classe de acessar um membro private dessa classe é um erro de compilação;

18 – Frequentemente é um erro de lógica quando um método contém um parâmetro ou variável local com o mesmo nome de um campo da classe, neste caso, utilize a referência this caso deseje acessar o campo da classe;

19 - É um erro de sintaxe se this for utilizado no corpo de um construtor para chamar um outro construtor da mesma classe se essa chamada não for a primeira instrução do construtor;

20 – Um erro de compilação ocorre se um programa tentar inicializar um objeto de uma classe passando o número incorreto ou tipos de argumentos para o construtor da classe;

21 – Um erro de compilação ocorre se um método static chamar um método de instância não static na mesma classe utilizando somente o nome do método;

22 – Referenciar this em um método static é considerado um erro de sintaxe;

23 – Um erro de compilação ocorre se um programa tentar importar métodos static que têm a mesma assinatura ou campos static que têm o mesmo nome proveniente de duas ou mais classes;

24 – Tentar modificar uma variável de instância final depois que ela é inicializada é um erro de compilação;

25 – Tentativas de modificar uma variável de instância final são capturadas em tempo de compilação em vez de causarem erros em tempo de execução;

26 - Não inicializar uma variável de instância final na sua declaração ou em cada construtor da classe produz um erro de compilação indicando que a variável talvez não tenha sido inicializada;

27 – Utilizar uma assinatura de método incorreta ao tentar sobrescrever um método de superclasse resulta em sobrecarga de método não intencional que pode levar a erros de lógica;

28 - É um erro de sintaxe sobrescrever um método com um modificador de acesso mais restrito;

29 – Um erro de compilação ocorre se um construtor de subclasse chamar um construtor de superclasse com argumentos que não coincidem com o número e os tipos de parâmetro em um dos construtores da superclasse;

30 – Tentar instanciar um objeto de uma classe abstrata é um erro de compilação;

31 – A falha em implementar os métodos abstratos de uma superclasse em uma subclasse é um erro de compilação a menos que a subclasse seja declarada com abstract;

32 – Atribuir uma variável de superclasse a uma variável de subclasse é um erro de compilação;

33 – Tentar declarar uma subclasse de uma classe final é um erro de compilação;

34 – Falhar em implementar qualquer método de uma interface em uma classe concreta que implementa a interface resulta em um erro de compilação indicando que a classe deve ser declarada abstract;

35 - É um erro de sintaxe colocar código entre um bloco try e seus blocos catch correspondentes;

36 – Cada bloco catch pode ter apenas um único parâmetro --- especificar uma lista de parâmetros de exceção separados por vírgulas é um erro de sintaxe;

37 – Um erro de compilação ocorre se um método tentar explicitamente lançar uma exceção verificada e essa exceção não estiver listada na cláusula throws do método;

38 – Se um método de subclasse sobrescrever um método de superclasse, é um erro o método de subclasse listar mais exceções em sua cláusula throws do que o método sobrescrito da superclasse;

39 – Colocar um bloco catch para um tipo de exceção da superclasse antes de outro bloco catch que capturam tipos de exceção de subclasse impediria que esses blocos executem, então ocorre um erro de compilação;

40 – Se uma exceção não tiver sido capturada quando o controle entrar em um bloco finally e esse bloco lançar uma exceção que não será capturada por ele, a primeira exceção será perdida e a exceção do bloco será retornada ao método chamador;

41 – Evite colocar código que possa lançar throw uma exceção em um bloco finally;

42 – Assumir que uma exceção lançada de um bloco catch será processada por esse bloco catch ou qualquer outro bloco catch associado com a mesma instrução try pode resultar em erros de lógica;

43 – Acessar um caractere fora dos limites de uma String resulta em StringIndexOutOfBoundsException;

44 – Tentar acessar um caractere que está além dos limites de uma StringBuilder resulta em StringIndexOutOfBoundsException;

**Boas práticas na Programação Orientada a Objetos:**

1 – Prefira listar os campos de uma classe primeiro, pois à medida que você ler o código, veja os nomes e os tipos das variáveis antes de utilizar os métodos em classe. Facilitando a leitura do seu código.

2 – Coloque uma linha em branco entre as declarações de método para separar os métodos e aprimorar a legibilidade do programa.

3 – O tipo declarado de uma variável indica se a variável é de um tipo primitivo ou tipo de referência.

4 - Forneça um construtor para assegurar que as variáveis de instância da sua classe sejam adequadamente inicializadas com valores significativos quando cada novo objeto de sua classe for criado;

5 - Métodos podem retornar no máximo um valor, mas o valor retornado poderia ser uma referência a um objeto que contêm muitos valores;

6 – Usar constantes de enumeração em vez de valores literais torna os programas mais fáceis de entender;

7 - Variáveis constantes também são chamadas de constantes nomeadas. Frequentemente elas tornam os programas mais legíveis do que programas que utilizam valores literais;

8 – Evite nomes de parâmetros ou variáveis locais em métodos que conflitem com nomes de campos. Isso ajuda a evitar bugs;

9 - Um construtor pode chamar métodos da classe;

10 – Quando um objeto de uma classe contém uma referência a um outro objeto da mesma classe, o primeiro objeto pode acessar todos os dados e métodos do segundo objeto, inclusive aqueles que são private;

11 – Ao implementar um método de uma classe, utilize os métodos set e get da classe para acessar os dados private da classe;

12 – Se apropriado, forneça métodos public para alterar e recuperar os valores de variáveis de instância private;

13 – Os métodos set e get ajudam a criar classes que são mais fáceis de depurar e se manter;

14 – Use uma variável static quando todos os objetos de uma classe precisarem utilizar a mesma cópia da variável;

15 - Variáveis e métodos de classe static existem e podem ser utilizados, mesmo se nenhum objeto dessa classe tiver sido instanciado;

16 – Invoque cada método static usando o nome de classe e um ponto(.) para a enfatização de que o método sendo chamado é um método static;

17 – Declarar uma variável de instância como final ajuda a impor o princípio do menor privilégio. Se uma variável de instância não deve ser modificada, declare- a como sendo final para evitar modificação;

18 – Um campo final também deverá ser declarado como static se ele for inicializado em sua declaração com um valor que é o mesmo de todos os objetos da classe. Depois dessa inicialização seu valor nunca poderá mudar;

19 – Os métodos de uma subclasse não acessam membros private diretamente da super classe. Uma subclasse pode alterar o estado de variáveis de instância private da superclasse somente por meio de métodos não private fornecidos na superclasse e herdados pela subclasse;

20 – Declarar variáveis de instância private ajuda-lhe a testar, depurar e modificar sistemas corretamente;

21 – Declare métodos sobrescritos com a notação @Override para assegurar em tempo de compilação que você definiu as assinaturas corretamente;

22 – Sempre é melhor localizar erros em tempo de compilação em vez de tempo de execução;

23 – Com a herança, as variáveis de instância comuns e os métodos de todas as classes na hierarquia são declarados em uma superclasse. Quando são feitas modificações nessas características comuns na superclasse, as subclasses herdam, portanto, as modificações;

24 – Utilize o modificador de acesso protected quando uma superclasse precisa fornecer um método somente para suas subclasses e outras classes no mesmo pacote;

25 – Declarar as variáveis de instância da superclasse private, permite a implementação de superclasse dessas variáveis de instância para alterar sem afetar as implementações de subclasse;

26 – Quando possível, não inclua variáveis de instâncias protected uma superclasse. Em vez disso, inclua métodos não private que acessam as variáveis de instância private;

27 – Se as subclasses são maiores do que precisam ser, recursos de memória e de processamento podem ser desperdiçados. Estenda a superclasse que contém a funcionalidade mais próxima daquilo de que você precisa;

28 – O polimorfismo permite tratar as generalidades e deixar que o ambiente de tempo de execução trate as especificidades. Você pode instruir objetos a se comportarem de maneiras apropriadas para esses objetos, sem nem conhecer seus tipos;

29 – O polimorfismo promove extensibilidade;

30 – Uma classe abstrata declara atributos e comportamentos comuns das várias classes em uma hierarquia de classes. Em geral, uma classe abstrata contém um ou mais métodos abstratos que as subclasses devem sobrescrever se elas precisarem de classes concretas;

31 - Ao declarar um método em uma interface, escolha um nome de método que descreva o propósito do método de uma maneira geral, pois o método pode ser implementado por muitas classes não implementadas;

32 – Todos os objetos de uma classe que implementam múltiplas interfaces têm o relacionamento do tipo “é um” com cada tipo de interface implementado;

33 – Quando um parâmetro de método é declarado com uma superclasse ou tipo de interface, o método processa o objeto recebido polimorficamente como um argumento

34 – Utilizando uma referência de superclasse, podemos invocar qualquer método declarado na superclasse e suas superclasses;

35 – O tratamento de exceções ajuda a aprimorar a tolerância a falhas de um programa;

36 – Se os problemas potenciais ocorrem raramente, mesclar o programa e a lógica do tratamento de erro pode degradar o desempenho do programa, pois o programa deve potencialmente realizar testes frequentes para determinar se a tarefa foi executada corretamente e se a próxima tarefa pode ser realizada;

37 – As exceções emergem pelo código explicitamente mencionado em um bloco try, por chamadas para outros métodos, por chamadas de métodos profundamente aninhadas iniciadas pelo código em um bloco try;

38 – Utilizar um nome de parâmetro de exceção que reflita o tipo do parâmetro promove a clareza lembrando-lhe do tipo de exceção em um tratamento;

39 – Incorpore sua estratégia de tratamento de exceções ao sistema desde o princípio do processo de design. Pode ser difícil incluir um tratamento de exceções depois que um sistema é implementado;

40 – O tratamento de exceções fornece uma técnica única e uniforme para processamento de problemas. Isso ajuda a programadores que trabalham em grandes projetos a entender melhor o código;

41 – Trabalhar com exceções verificadas, possibilita em um código mais robusto do que aquele que seria criado se fosse capaz de ignorar exceções;

42 – Se o método chamar outros métodos que lançam explicitamente exceções verificadas, essas exceções devem ser capturadas ou declaradas no método;

43 – Embora o compilador não imponha o requisito capture ou declare para as exceções não verificadas, ele fornece o código de tratamento de exceções adequado quando se sabe que tais exceções são possíveis;

44 – O bloco finally é um lugar ideal para liberar recursos adquiridos em um bloco try, o que ajuda a eliminar vazamento de recurso;

45 – Sempre libere um recurso explicitamente e logo que ele não for necessário. Isso tornará recursos disponíveis para a reutilização aprimorando a sua utilização;

46 – As exceções podem ser lançadas a partir de construtores, evitando assim que um objeto seja formado inadequadamente;

47 – Se possível, indique as exceções provenientes de seus métodos utilizando classes de exceção existentes, em vez de criar novas;

48 – Associar cada tipo de mal funcionamento sério em tempo de execução com uma classe ‘Exception’ propriamente identificada aprimora a clareza do programa;

49 – Por convenção, todos os nomes de classe de exceções devem terminar com a palavra ‘Exception’;

50 – O Java trata todos os literais string com o mesmo conteúdo de um único objeto String que tem muitas referências a ele, economizando memória;

51 - Não é necessário copiar um objeto String existente. Os objetos String são imutáveis, ou seja, seu conteúdo de caractere não pode ser alterado depois que são criados, porque a classe String não oferece nenhum método que permita que o conteúdo de um objeto String seja modificado;

52 – O Java pode realizar certas otimizações que envolvem objetos String pois ele sabe que esses objetos não se alterarão. Strings devem ser utilizadas se os dados não se alterarem;

53 – Em programas que frequentemente utilizam a concatenação de strings, ou outras modificações de strings, em geral, é mais eficiente implementar as modificações com a classe StringBuilder;

54 – Aumentar a capacidade de uma StringBuilder dinamicamente pode exigir um tempo relativamente longo, pois ao executar um longo número dessas operações podem degradar o desempenho de um aplicativo;

* Essas foram as boas e más práticas que encontrei no livro que você me indicou para ler, e achei todas elas muito úteis e correlacionadas com o que eu aprendi ao cursar a matéria de Programação Orientada a Objetos 1.
* Se tiver alguma que achar que não tem uma relevância pode ficar à vontade para fazer modificações no documento.
* A última parte do documento terá as perguntas que colocarei no site, e as suas respectivas perguntas e respostas grifadas com a cor verde.

**Classe e Objetos**

1 – O que são os métodos na Orientação a Objetos?

1. Os métodos são mecanismos que realiza tarefas a partir da definição de um objeto. Ocultando de seus usuários as tarefas que são feitas a partir dele.
2. Os métodos descrevem os mecanismos que realmente realizam as suas tarefas. Ocultando de seus usuários as tarefas que são feitas a partir dele.
3. Os métodos são as representações de um conjunto de objetos que possuem características e comportamentos comuns.
4. Os métodos são representações de um determinado elemento do mundo real. Somente são analisados os objetos que têm relevância para a solução de um problema proposto.

2 – Qual a razão de o Java ser conhecido como uma linguagem de Programação Orientada a Objetos?

1. A razão é que ela é a única linguagem a utilizar encapsulamento, abstração e herança. Facilitando o entendimento da linguagem
2. A razão é que ela facilita o reuso e manutenção do código através do encapsulamento, abstração e herança.
3. A razão é que primeiro deve se construir uma classe descrevendo qual é a funcionalidade do programa e como ele deverá se comportar perante seus objetos.
4. A razão é que primeiro deve se construir um objeto de uma classe antes de fazer um programa realizar as tarefas que a classe descreve como fazer.

3 – O que significa a sigla UML? E qual seu significado?

1. Diagrama de classes UML. É uma linguagem gráfica utilizada pelos programadores para representar sistemas orientados a objetos de uma maneira padronizada.
2. Diagrama de entidade UML. É uma linguagem gráfica utilizada pelos programadores para representar sistemas orientados a objetos de uma maneira padronizada.
3. Diagrama de classes UML. É uma linguagem estruturada utilizada pelos programadores para representar sistemas orientados a objetos de uma maneira padronizada.
4. Diagrama de entidade UML. É uma linguagem estruturada utilizada pelos programadores para representar sistemas orientados a objetos de uma maneira padronizada.

4 – Quantos e quais são os compartimentos de um diagrama de classe UML?

1. Contém apenas um compartimento, onde contém as informações do programa como classe, objetos e atributos.
2. Contém dois compartimentos, onde o superior contém o nome da classe centralizada e em negrito, o segundo contém os atributos da classe e operações da classe.
3. Contém três compartimentos, onde o superior contém o nome da classe centralizada e em negrito, o segundo contém os atributos da classe e o terceiro compartimento terá as operações da classe.
4. Não contém nenhum compartimento. A UML serve apenas para ver o status do programa.

5- Como são definidos os parâmetros de uma classe?

1. Os parâmetros são definidos em uma lista de parâmetros separador por vírgulas, que está localizado nos parênteses depois do nome do método.
2. Os parâmetros são definidos em uma lista de parâmetros separador por ponto e vírgula, que está localizado nos colchetes depois da classe.
3. Os parâmetros são definidos em uma lista de parâmetros separador por ponto e vírgula, que está localizado nos colchetes depois do objeto.
4. Os parâmetros são definidos em uma lista de parâmetros separador por vírgulas, que está localizado nos parênteses depois do nome do objeto.

6 – O que são variáveis locais?

1. São variáveis declaradas no corpo de um método particular e podem ser utilizadas por todos os métodos existentes.
2. São variáveis declaradas no corpo de um método particular e só podem ser utilizadas nesse método.
3. São variáveis declaradas no corpo de uma classe e só podem ser utilizadas nesta classe.
4. São variáveis declaradas no corpo de um objeto e só podem ser utilizadas em métodos.

7 – O que são os tipos primitivos do Java?

1. São boolean, byte, char, short, int, long, float,double e String.
2. São boolean, byte, char, short, int, long, float.
3. São boolean, byte, char e short.
4. São boolean, byte, char, short, int, long, float e double.

8 - Variáveis do tipo primitivo podem invocar métodos de instância?

1. Para invocar métodos de instância de um objeto é necessária uma referência a um objeto. Uma variável do tipo primitivo referência um objeto, com isso, ela pode ser utilizada.
2. Para invocar métodos de instância de um objeto é necessária uma referência a um objeto. Uma variável do tipo primitivo não referência um objeto, com isso, não pode ser utilizado para invocar um método.
3. Para invocar métodos de instância de um objeto é necessária uma referência a um objeto. Uma variável do tipo primitivo não referência um objeto, porém classes fazem referências com isso, elas serão utilizadas.
4. Para invocar métodos de instância de um objeto é necessária uma referência a um objeto. Uma variável do tipo primitivo referência uma classe e objeto, com isso, ela pode ser utilizada.

9 - O que é classe em Java?

1. É a representação de um conjunto de objetos que possuem características e comportamentos comuns.
2. É a representação de um conjunto de métodos que possuem funcionalidades em comum.
3. Classes são formadas por objetos e métodos.
4. Ela é formada por dados e métodos.

10 – O que são objetos em Java?

1. Termo utilizado para representar um método do mundo real. Somente são analisados os métodos que têm relevância para a funcionalidade de um problema proposto.
2. Termo utilizado para representar um método.
3. Termo utilizado para representar um determinado elemento do mundo real. Somente são analisados os objetos que têm relevância para a solução de um problema proposto.
4. Termo utilizado para representar um determinado elemento. Objetos são compostos de classes.

1.p - Porque o um método static é especial?

1. Ele é considerado especial pois ele pode ser chamado sem primeiro criar um objeto da classe em que o método é declarado.
2. Ele é considerado especial pois ele pode ser chamado sem primeiro criar uma classe onde método é declarado.
3. Ele é considerado especial pois ele é criado a partir da criação da classe.
4. Ele é considerado especial pois ele é criado a partir da criação o objeto e seus métodos são definidos a partir de suas classes.

2.p - O que o método main faz em um programa?

1. O método main define como será a execução de cada aplicativo.
2. O método main recomeça a execução de cada aplicativo.
3. O método main começa a execução de cada aplicativo.
4. O método main termina a execução de cada aplicativo.

3p – Qual a finalidade de um construtor em uma aplicação Java?

1. Ele é semelhante ao método, ele é utilizado na hora em que um objeto é criado para a inicialização dos dados no objeto.
2. Ele é semelhante a classe, ele é utilizado na hora em que um objeto é criado para a inicialização dos dados no objeto.
3. Ele é semelhante ao objeto, ele é utilizado na hora em que um objeto é criado para a inicialização dos dados no objeto.
4. Ele é semelhante ao método, ele é utilizado na hora em que uma classe é criado para a inicialização dos dados no objeto.

4p – Qual a finalidade do objeto System.out?

1. Serve para chamar os métodos print, printf, println e text.
2. Serve para chamar os métodos print e println.
3. Serve para chamar os métodos print, printf e println.
4. Serve para chamar os métodos printf e println.

5p – Qual a expressão de criação utilizada para a instanciação de um novo objeto?

1. extends.
2. new.
3. implements
4. import

6p – Em quais classes pode conter o método main?

1. Somente em classes que utilizam classes que instanciam objetos.
2. Somente em classes que utilizam classes que instanciam objetos e tem funcionalidades em seus métodos.
3. Somente em classes que utilizam construtores em suas classes.
4. Qualquer classe pode conter o método main. Pois o JVM invoca o método main somente na classe utilizada para executar o aplicativo.

7p – Qual a função do método nextLine em um programa?

1. O método nextline lê os caracteres digitados pelo usuário até o caractere de nova linha ser encontrado.
2. O método nextline lê os caracteres digitados pelo usuário.
3. O método nextline lê os caracteres digitados pelo usuário até o caractere atingir um determinado tamanho.
4. O método nextline lê os caracteres digitados pelo usuário, porém o usuário tem que utilizar String para utilizar.

8p – Qual a quantidade de argumentos em uma chamada de método?

1. O número de argumentos em uma chamada de método deve corresponder ao número de criação de objetos criados dentro da classe.
2. O número de argumentos em uma chamada de método deve corresponder ao número de parâmetros na lista de objetos da declaração no método.
3. O número de argumentos em uma chamada de método deve corresponder ao número de parâmetros na lista de parâmetros da declaração no método.
4. O número de argumentos em uma chamada de método deve corresponder ao número de classes.

9p – Qual a funcionalidade da palavra new no Java?

1. A palavra-chave new solicita memória do sistema para armazenar um objeto e então chama o construtor da classe correspondente para inicializar o objeto.
2. A palavra-chave new solicita memória do sistema para armazenar uma classe e então chama o construtor da classe correspondente para inicializar o objeto.
3. A palavra-chave new solicita memória do sistema para armazenar um método e então chama o construtor da classe correspondente para inicializar o objeto.
4. A palavra-chave new solicita memória do sistema para realizar uma nova classe e então chama o construtor da classe correspondente para inicializar o objeto.

10p – Qual a diferença entre double e float?

1. As variáveis do tipo float representam números de pontos flutuantes de precisão dupla e podem significar até dois dígitos significativos, já as variáveis double representam números de ponto simples.
2. As variáveis do tipo float representam números de ponto flutuante de precisão dupla, já as variáveis double representam números de pontos flutuantes de precisão simples e podem significar até seis dígitos significativos.
3. As variáveis do tipo float representam números de pontos flutuantes de precisão simples e podem significar até seis dígitos significativos, já as variáveis double representam números de ponto flutuante de precisão dupla.
4. Ambas as duas possuem a mesma finalidade.

**Métodos**

1- No que consiste a técnica dividir para conquistar?

1. É uma técnica que consiste em dividir dados em duas ou mais partes independentes, processar estas partes em paralelo, e ao final unificar os resultados de cada parte deixando o código pequeno e simples de entender.
2. É uma técnica que consiste em dividir dados em duas ou mais partes dependentes, processar estas partes em paralelo, e ao final unificar os resultados de cada parte deixando o código pequeno e simples de entender.
3. É uma técnica que consiste em dividir dados em duas ou mais partes independentes, processar estas partes em paralelo, e ao final mostrar aos seus usuários.
4. É uma técnica que consiste em dividir dados em duas ou mais partes dependentes, processar estas partes em paralelo, e ao final mostrar aos seus usuários.

2 – Qual a funcionalidade do método?

1. Os métodos permitem modularizar um programa separando suas tarefas em unidades contidas. As instruções em um método são escritas várias vezes.
2. Os métodos separam suas tarefas em unidades. As instruções em um método são escritas várias vezes.
3. Os métodos permitem modularizar um programa separando suas tarefas em unidades contidas. As instruções em um método são escritas somente uma vez e permanecem ocultas de outros métodos.
4. Os métodos permitem modularizar um programa separando suas tarefas em unidades contidas. As instruções em um método são escritas várias vezes e permanecem ocultas de outros métodos.

3 - O que consiste na técnica sobrecarga de método?

1. Está técnica permite que vários métodos com o mesmo nome sejam declarados em uma classe, contanto que os métodos tenham conjuntos diferentes de parâmetros.
2. Está técnica permite que um método seja declarado em uma classe.
3. Está técnica permite que vários métodos com o mesmo nome sejam declarados em um objeto, contanto que os métodos tenham conjuntos diferentes de parâmetros.
4. Está técnica permite que vários métodos com o mesmo nome sejam declarados em uma classe, contanto que os métodos tenham os mesmos conjuntos de parâmetros.

4 - Há três maneiras de fazer a chamada de um método, dentre essas três maneiras marque a alternativa correta:

1. Usar o próprio nome de um método para chamar um outro método de outra classe.
2. Usar o próprio nome de um método para chamar um outro método da mesma classe.
3. Usar o objeto instanciado para chamar outros objetos de outras classes e seus métodos.
4. Usar a classe para chamar métodos comuns entre elas.

5 – Dentre as alternativas sobre métodos, marque a alternativa incorreta.

1. Uma chamada de método especifica o nome do método a ser chamado e fornece os argumentos que o método chamado requer para realizar a sua tarefa.
2. Pode se utilizar um método main em cada classe declarada.
3. Um método pode retornar no máximo um valor, mas pode ser referenciado a um objeto que contém muitos valores.
4. Uma classe não pode conter métodos static para realizar tarefas comuns que não exigem um objeto da classe.

**Arrays e Array Lists**

1 – O que são Arrays?

1. São estruturas de dados consistindo em itens de dados do mesmo tipo relacionados.
2. São estruturas de dados consistindo em itens de dados de vários tipos relacionados.
3. São Strings que são utilizadas para escrever uma mensagem.
4. São chars que precisam guardar campos em sua memória para cada caractere utilizado.

2 – Como referenciamos um elemento particular em um array?

1. Deve se especificar o nome da referência para o array e o número de deslocamento do elemento array.
2. Deve se especificar o nome da referência para o array.
3. Deve se especificar o nome da referência para o array e o número de posição do elemento do array, o número de posição é chamado de índice ou subscrito do elemento.
4. Deve se especificar o nome da referência para o array, o número de posição do elemento do array e o seu deslocamento.

3 – O que são classes Arrays?

1. Classes Arrays fornecem métodos main que desempenham manipulações de arrays comuns, incluindo sort para classificar um array, equals para comparar arrays e entre outros.
2. Classes Arrays fornecem métodos static que desempenham manipulações de arrays comuns, incluindo sort para classificar um array, equals para comparar arrays e entre outros.
3. Classes Arrays fornecem métodos import que desempenham manipulações de arrays comuns, incluindo sort para classificar um array, equals para comparar arrays e entre outros.
4. Classes Arrays fornecem métodos static que desempenham manipulações de arrays comuns, incluindo sort para sortear um array, equals para igualar arrays e entre outros.

4 – O que são Arrays multidimensionais?

1. São Arrays que requerem três índices para identificar um elemento particular.
2. São Arrays que requerem dois índices para identificar vários elementos.
3. São Arrays que requerem quatro índices para identificar um elemento particular.
4. São Arrays que requerem dois índices para identificar um elemento particular.

5 – Como é feita a criação de um objeto Array?

1. Para a criação de um objeto array, deve se especificar o tipo de elemento e o número de elementos do array como parte de uma expressão de criação de arrays que utiliza a palavra-chave import.
2. Para a criação de um objeto array, deve se especificar o tipo de elemento e o número de elementos do array como parte de uma expressão de criação de arrays que utiliza a palavra-chave new.
3. Para a criação de um objeto array, deve se especificar o tipo de elemento e o número de elementos do array como parte de uma expressão de criação de arrays que utiliza a palavra-chave extends.
4. Para a criação de um objeto array, deve se especificar o tipo de elemento e o número de elementos do array como parte de uma expressão de criação de arrays que utiliza a palavra-chave public.

6 – Entre as afirmações a seguir identifique qual está correta:

1. Um array pode armazenar muitos tipos de valores diferentes.
2. Um índice de array deve ser um inteiro ou uma expressão do tipo inteiro.
3. Um elemento individual de um array que é passado para um método e modificado nesse método conterá o valor modificado quando o método chamado completar sua execução.
4. Argumentos de linha de comando são separados por vírgulas.

**Classes e objetos mais aprofundados**

1 – O que significa a palavra-chave this?

1. A palavra this significa que todo objeto pode acessar uma referência a si próprio utilizando a referência new.
2. A palavra this significa que todo objeto pode acessar uma referência a si próprio utilizando a referência import.
3. A palavra this significa que todo objeto pode acessar uma referência a si próprio utilizando a referência this.
4. A palavra this significa que todo objeto pode acessar uma referência a si próprio utilizando a referência extends.

2 – O que é composição em Java?

1. Consiste em uma classe pode ter referências a objetos de outras classes como membros. E as vezes a composição é referida como um relacionamento tem um.
2. Consiste em uma classe pode ter referências a classes de outros objetos como membros. E as vezes a composição é referida como um relacionamento um para um.
3. Consiste em uma classe pode ter referências a objetos de outras classes como membros. E as vezes a composição é referida como um relacionamento um para um.
4. Consiste em uma classe pode ter referências a classes de outros objetos como membros. E as vezes a composição é referida como um relacionamento tem um.

3 – O que é uma variável static em Java?

1. Uma variável static representa as informações de escopo do objeto. Onde todos os objetos da classe compartilham os mesmos dados.
2. Uma variável static representa as informações de escopo de classe. Onde todos os objetos da classe compartilham os mesmos dados.
3. Uma variável static representa as informações de escopo do método. Onde todos os métodos da classe compartilham os mesmos dados.
4. Uma variável static executa uma função com a dependência do conteúdo de um objeto ou a execução da instância de uma classe.

4 – Quais destas opções abaixo são objetos imutáveis (Não podem ser modificados depois de criados)?

1. Long
2. Integer
3. Boolean
4. String

5 – Quais dessas sentenças é falsa?

1. Os métodos set são comumente chamados de métodos modificadores porque geralmente alteram um valor.
2. Se uma classe declarar construtores, o compilador criará um construtor padrão.
3. Os métodos get são comumente chamados de métodos de acesso ou métodos de consulta.
4. Se nenhum construtor for fornecido em uma classe, o compilador criará um construtor padrão.

6 – Sobre membros da classe static indique a sentença verdadeira.

1. A referência this pode ser utilizada em um método static.
2. Um método declarado não pode acessar membros de classe não static, pois um método static pode ser chamado mesmo quando nenhum objeto da classe foi instanciado.
3. Os membros de classe static não são carregados na memória.
4. Uma variável static representa as informações de um objeto que será compartilhada entre as classes.

7 – O que a palavra chave final indica?

1. Teste
2. Teste
3. Teste
4. A palavra-chave final especifica que uma variável não é modificável. Essas variáveis podem ser inicializadas quando são declaradas ou pelos construtores da classe.

8 – Qual a diferença entre declaração import simples e import por demanda?

1. Uma declaração import do tipo simples especifica uma classe a importar, já a import por demanda importa somente as classes que o programa utiliza a partir de um pacote particular.
2. Uma declaração import do tipo simples importa somente a classe que o programa utiliza a partir de um pacote particular, já a import por demanda especifica uma classe a importar.
3. Uma declaração import do tipo simples especifica um objeto a ser instanciado, já a import por demanda importa somente objetos que o programa utilizará.
4. Ambas possuem a mesma funcionalidade.

**Herança**

1 – O que é herança?

1. É uma forma de reutilização de software em que uma nova classe é criada absorvendo membros de um objeto instanciado.
2. É uma forma de reutilização de software em que um novo método é implementado a partir de sua classe.
3. É uma forma de reutilização de software em que uma nova classe é criada absorvendo membros de uma classe de uma classe existente e aprimorada com capacidades novas ou modificadas.
4. A herança permite escrever programas que processam objetos que compartilham a mesma classe superclasse em uma hierarquia de classes como se todos os objetos fossem objetos da superclasse.

2 – Dentre as sentenças identifique a incorreta:

1. Ao criar uma nova classe, em vez de declarar membros completamente novos, você pode designar que a nova classe herde membros de uma classe existente.
2. A classe existente é chamada de superclasse e a nova de subclasse, onde cada subclasse pode tornar uma superclasse para futuras subclasses.
3. O Java só suporta herança única onde cada classe é derivada exatamente de uma superclasse direta.
4. Assim como o C++ o Java suporta heranças múltiplas, facilitando os conceitos de herança.

3 – Quais as diferenças entre modificadores de acesso public e protected?

1. Os modificadores public oferece um nível intermediário entre protected e private, já os protected são acessíveis em todo o programa.
2. Os modificadores public são acessíveis em todo o programa, já os protected oferece um nível intermediário entre public e private.
3. Os modificadores public é acessível somente em sua classe, os protected são acessíveis em todo o programa.
4. Os modificadores public oferece um nível intermediário entre protected e private, já os protected não são acessíveis em todo o programa.

4 – No modificador de acesso private ele só é acessível dentro de sua própria classe, porém como podemos manipular os dados dentro destes modificadores?

1. Declarar variáveis de instâncias como private e fornecer os métodos get e set para validar e sobrescrever as variáveis.
2. Declarar variáveis de instâncias como private e fornecer os métodos import e extends para manipular e validar as variáveis.
3. Declarar variáveis de instâncias como private e fornecer os métodos get e set para manipular e validar as variáveis.
4. Sua manipulação é feita como os modificadores public e protected.

5 - Porque é considerado uma prática de engenharia pobre declarar variáveis de instância public?

1. Pois é considerado uma técnica não segura pois permite acesso restrito as variáveis de instância, aumentando a probabilidades de chances a erros.
2. Pois é considerado uma técnica não segura pois permite acesso irrestrito as variáveis de instância, aumentando a probabilidades de chances a erros.
3. Pois as variáveis public estão sendo trocadas por protected por serem mais seguras a seus usuários.
4. Pois as variáveis public estão deixando de ser utilizadas por serem bastante complexas.

6 – Qual a diferença entre herança simples e a herança múltipla?

1. Em herança única, uma classe é derivada de uma subclasse direta. Já na herança múltipla, uma classe é derivada de mais de uma subclasse direta. Java não suporta herança múltipla.
2. Em herança única, uma classe é derivada de uma superclasse direta. Já na herança múltipla, uma classe é derivada de mais de uma superclasse direta. Java suporta herança múltipla.
3. Em herança única, uma classe é derivada de uma superclasse indireta. Já na herança múltipla, uma classe é derivada de mais de uma superclasse indireta.
4. Em herança única, uma classe é derivada de uma superclasse direta. Já na herança múltipla, uma classe é derivada de mais de uma superclasse direta. Java não suporta herança múltipla.

7 – Marque a alternativa incorreta:

1. Uma subclasse é mais específica que sua superclasse e representa um grupo menor de objetos.
2. Um relacionamento é um representa herança. Ele é um objeto da subclasse e pode ser tratado como um objeto de superclasse.
3. Os relacionamentos de herança simples não formam estruturas hierárquicas do tipo árvore e sim em formatos de lista encadeada.
4. Um relacionamento tem um representa a composição. Onde um objeto de classe contém referências a objetos de outras classes.

1 p – O que é o método ToString?

1. Método ToString retorna um Char representando um objeto, ele é chamado implicitamente sempre que um objeto precisa ser convertido em String.
2. Método ToString retorna uma String representando um método, ele é chamado implicitamente sempre que um método precisa ser convertido em String.
3. Método ToString retorna uma String representando uma classe, ela é chamada implicitamente sempre que um objeto precisa ser convertido em String.
4. Método ToString retorna uma String representando um objeto, ele é chamado implicitamente sempre que um objeto precisa ser convertido em String.

2 p – Qual a palavra-chave utilizada para a representação de herança em um código?

1. Extends
2. Heritage
3. New
4. Import

3 p – O que significa a notação @Override em um código em Java?

1. Para evitar a sobrecarga intencional indicando que um método sobrecarrega um método de superclasse indireta.
2. Para evitar a sobrecarga não intencional indicando que um método sobrecarrega um método de superclasse direta.
3. Para evitar a sobrecarga não intencional indicando que um método sobrecarrega um método de superclasse.
4. Para evitar a sobrecarga não intencional indicando que um método sobrecarrega um método de subclasse.

4p – O que significa utilizar a palavra-chave super?

1. Utilizar a palavra super significa acessar os membros da superclasse sobrescritos.
2. Utilizar a palavra super significa acessar os membros da superclasse direta.
3. Utilizar a palavra super significa acessar os membros da superclasse indireta.
4. Utilizar a palavra super significa acessar os membros da superclasse.

**Polimorfismo**

1 – O que é Polimorfismo?

1. O polimorfismo permite escrever programas que processam objetos que compartilham a mesma classe superclasse em uma hierarquia de classes como se todos os objetos fossem objetos da superclasse.
2. É uma forma de reutilização de software em que uma nova classe é criada absorvendo membros de uma classe de uma classe existente e aprimorada com capacidades novas ou modificadas.
3. O polimorfismo é muitas formas de escrever o mesmo método.
4. O polimorfismo permite sobrescrever programas que processam objetos que compartilham a mesma classe subclasse em uma hierarquia de classes como se todos os objetos fossem objetos da subclasse.

2 – Qual a funcionalidade do Polimorfismo?

1. Podemos projetar e implementar sistemas que são facilmente extensíveis. Novos objetos podem ser adicionados com pouca ou nenhuma modificação a partes gerais do programa.
2. Podemos projetar e implementar sistemas que são facilmente extensíveis. Novos métodos podem ser adicionados com pouca ou nenhuma modificação a partes gerais do programa.
3. Podemos projetar e implementar sistemas que são facilmente extensíveis. Novas classes podem ser adicionadas com pouca ou nenhuma modificação a partes gerais do programa.
4. A funcionalidade do polimorfismo é a mesma da herança em Java.

3 – Indique a sentença incorreta em relação a classes e métodos abstratos:

1. As classes abstratas não podem ser utilizadas para instanciar objetos, pois são incompletas.
2. Uma classe que contém qualquer método abstrato deve ser declarada como uma classe abstrata mesmo se contiver alguns métodos concretos.
3. As classes que podem ser utilizadas para instanciar objetos são chamadas de concretas. Essas classes fornecem implementações de cada método que declara.
4. O polimorfismo não é eficaz para implementar sistemas em camadas.

4 – Qual o propósito da classe abstrata?

1. O propósito principal de uma classe abstrata é fornecer uma subclasse apropriada a partir da qual outras classes podem instanciar, e, compartilhar um design comum.
2. O propósito principal de uma classe abstrata é fornecer uma superclasse apropriada a partir da qual outras classes podem herdar, e, compartilhar um design comum.
3. O propósito principal de uma classe abstrata é abstrair objetos em comum.
4. O propósito principal de uma classe abstrata é fornecer uma superclasse para compartilhar métodos comuns entre si.

5 – Sobre interface indique a alternativa incorreta:

1. Uma interface especifica quais operações são permitidas, mas não como as operações são realizadas.
2. Uma interface Java descreve o conjunto de métodos que pode ser chamado em um objeto.
3. Uma interface não pode declarar constantes. Pois elas são implicitamente public, static e final.
4. Para utilizar a interface, uma classe deve especificar que ela implementa a interface e declarar cada método de interface com a assinatura especificada.

6 – Qual das seguintes alternativas é verdadeira?

1. Um objeto de uma classe que implementa uma interface pode ser pensado como um objeto desse tipo de interface.
2. Todos os métodos em uma classe abstract deve ser declarados como método abstract.
3. Não é permitido invocar um método como abstract, uma subclasse deverá implementar esse método.
4. Se uma superclasse declarar um método como abstract, uma subclasse implementará esse método.

1 p – Qual a palavra – chave na declaração de uma interface?

1. Interface
2. Extends
3. Super
4. New

**Tratamento de Exceções**

1 – Qual a finalidade do tratamento de exceções?

1. O tratamento de exceções permite remover da “linha principal” de execução do programa o código de tratamento de erro, aprimorando a clareza do programa e destacando sua capacidade de modificação.
2. O tratamento de exceções permite excluir erros de compilação do programa.
3. O tratamento de exceções permite excluir erros de sintaxe do programa.
4. O tratamento de exceções permite remediar erros que podem ocorrer no programa.

2 – O que é a clausula throws?

1. Uma cláusula throws especifica as exceções que a classe lança.
2. Uma cláusula throws especifica as exceções que o objeto lança.
3. Uma cláusula throws especifica as exceções que o método lança.
4. Uma cláusula throws especifica as exceções que podem ocorrer a partir da execução de um código.

3 – Quando se deve utilizar o tratamento de exceções?

1. O tratamento de exceções é projetado para processar erros síncronos, que ocorrem quando uma instrução é executada.
2. O tratamento de exceções é projetado para processar erros assíncronos, que ocorrem quando uma instrução é executada.
3. O tratamento de exceções é projetado para processar erros síncronos, que ocorrem quando uma classe é executada.
4. O tratamento de exceções é projetado para processar erros assíncronos, que ocorrem quando uma instrução é executada.

4 – Qual a diferença entre exceções verificadas e não verificadas?

1. Exceções verificadas costumam ser causadas por deficiências no código do programa, já as não verificadas, causadas por má utilização do programa.
2. Exceções verificadas são tipicamente causadas por condições que não estão no controle do programa, já as não verificadas, costumam ser causadas por deficiências no código do programa.
3. Exceções verificadas costumam ser causadas por deficiências no código do programa, já as não verificadas, tipicamente causadas por condições que não estão no controle do programa.
4. Nenhuma ambas possuem a mesma funcionalidade.

5 – O que indica uma exceção em Java?

1. Uma exceção é uma indicação de um problema que ocorre durante a finalização de um programa.
2. Uma exceção é uma indicação de um problema que ocorre durante interrupção de um programa.
3. Uma exceção é uma indicação de um problema no decorrer da interpretação do código.
4. Uma exceção é uma indicação de um problema que ocorre durante execução de um programa.