1 - Em um sistema de gerenciamento de contas bancárias, temos uma classe Conta, que representa uma conta bancária genérica, e subclasses ContaCorrente e ContaPoupanca, que representam tipos específicos de contas.

O método sacar() é sobrescrito nas subclasses para fornecer diferentes implementações, dependendo do tipo de conta.

O código a seguir implementa a estrutura das classes mencionadas:

```
class Conta {
    public void sacar(double valor) {
        System.out.println("Saque realizado na conta geral.");
    }
}

class ContaCorrente extends Conta {
    @Override
    public void sacar(double valor) {
        System.out.println("Saque realizado na conta corrente: R$" + valor);
    }
}

class ContaPoupanca extends Conta {
    @Override
    public void sacar(double valor) {
        System.out.println("Saque realizado na conta poupança: R$" + valor);
    }
}

public class Banco {
    public static void main(String[] args) {
        Conta conta1 = new ContaCorrente();
        Conta conta2 = new ContaPoupanca();

        try {
            conta1.sacar(500);
            conta2.sacar(1000);
        } catch (Exception e) {
            System.out.println("Erro ao realizar o saque: " + e.getMessage());
        }
    }
}
```

O código apresentado acima demonstra o uso de ligação dinâmica ao invocar o método sacar() nas instâncias das classes ContaCorrente e ContaPoupanca. Assinale a alternativa correta a respeito da execução do código e do comportamento do tratamento de exceções.

Alternativas:

a) A invocação do método sacar() no objeto conta1 resulta em "Saque realizado na conta geral." e no objeto conta2 resulta em "Saque realizado na conta poupança: R$1000", devido à ligação estática.

b) A invocação do método sacar() no objeto conta1 resulta em "Saque realizado na conta corrente: R$500." e no objeto conta2 resulta em "Saque realizado na conta poupança: R$1000", devido à ligação dinâmica. Não ocorre exceção durante a execução.

c) O código geraria uma exceção durante a execução, pois a ligação estática não permite que os métodos sobrescritos nas subclasses sejam executados corretamente.

d) O código geraria uma exceção ao tentar realizar o saque, pois o tratamento de exceção está configurado para capturar todas as exceções, mas o tipo de exceção não é especificado corretamente, resultando em um erro de compilação.

2 – Em um sistema bancário, as classes Conta, ContaCorrente e ContaPoupanca têm métodos relacionados a operações bancárias, como depositar(), sacar(), transferir(), etc. As subclasses ContaCorrente e ContaPoupanca sobrescrevem esses métodos de maneira diferente para adaptar o comportamento às características específicas de cada tipo de conta.
Considerando que o sistema utiliza ligação dinâmica para determinar o método que será invocado em tempo de execução, o código abaixo foi implementado para realizar operações bancárias:

```
class Conta {
    public void depositar(double valor) {
        System.out.println("Depositando na conta comum: R$" + valor);
    }

    public void sacar(double valor) {
        System.out.println("Saque realizado na conta comum: R$" + valor);
    }
}

class ContaCorrente extends Conta {
    @Override
    public void depositar(double valor) {
        System.out.println("Depositando na conta corrente: R$" + valor);
    }

    @Override
    public void sacar(double valor) {
        if (valor > 1000) {
            System.out.println("Erro: Limite de saque excedido na conta corrente.");
```

```
        } else {
            System.out.println("Saque realizado na conta corrente: R$" + valor);
        }
    }
}

class ContaPoupanca extends Conta {
    @Override
    public void depositar(double valor) {
        System.out.println("Depositando na conta poupança: R$" + valor);
    }

    @Override
    public void sacar(double valor) {
        if (valor > 500) {
            System.out.println("Erro: Limite de saque excedido na conta poupança.");
        } else {
            System.out.println("Saque realizado na conta poupança: R$" + valor);
        }
    }
}

public class Banco {
    public static void main(String[] args) {
        Conta conta1 = new ContaCorrente();
        Conta conta2 = new ContaPoupanca();

        conta1.depositar(1500);
        conta1.sacar(2000);  // Exceção de saque
        conta2.depositar(500);
        conta2.sacar(1000);  // Exceção de saque
    }
}
```

A partir do código apresentado, explique o que ocorre durante a execução do programa. Qual é o comportamento da ligação dinâmica nesse cenário e como ela afeta a invocação dos métodos sobrescritos nas subclasses ContaCorrente e ContaPoupanca?
Além disso, como o tratamento de exceções é aplicado nas operações de saque?
Qual seria a saída esperada do programa? Justifique sua resposta.

_____

_____

_____

3 - Em um sistema de gerenciamento de pedidos em um e-commerce, existe uma classe Pedido com métodos que processam a compra de produtos. O sistema utiliza exceções personalizadas para garantir que as operações de processamento de pedidos sejam realizadas de maneira segura.
O código abaixo mostra uma implementação do sistema com tratamento de exceções:

```
class Pedido {
   private double total;

   public Pedido(double total) {
      if (total <= 0) {
         throw new IllegalArgumentException("O valor do pedido deve ser positivo.");
      }
      this.total = total;
   }

   public void processarPagamento() {
      if (total < 50) {
         throw new PagamentoInsuficienteException("Valor mínimo de pagamento não atingido.");
```

```
        }
        System.out.println("Pagamento processado com sucesso. Total: R$"
+ total);
    }

    public void enviarPedido() {
        System.out.println("Pedido enviado com sucesso!");
    }
}

class PagamentoInsuficienteException extends RuntimeException {
    public PagamentoInsuficienteException(String mensagem) {
        super(mensagem);
    }
}

public class SistemaEcommerce {
    public static void main(String[] args) {
        try {
            Pedido pedido1 = new Pedido(30);
            pedido1.processarPagamento();
            pedido1.enviarPedido();
        } catch (IllegalArgumentException | PagamentoInsuficienteException
e) {
            System.out.println("Erro ao processar o pedido: " +
e.getMessage());
        }
    }
}
```

No contexto do sistema de gerenciamento de pedidos apresentado, explique como o **tratamento de exceções** é utilizado para garantir que erros relacionados ao valor do pedido e ao pagamento sejam adequadamente tratados.
O que acontece durante a execução do código acima e qual será a saída esperada? Justifique a sua resposta.

_____

_____

_____

_____

_____

_____

_____

_____

4 – Imagine que você está desenvolvendo um sistema de gestão para uma livraria. O sistema precisa organizar os livros disponíveis para venda, gerenciar os detalhes dos clientes e processar pedidos de forma eficiente. O sistema será baseado em programação orientada a objetos, utilizando os conceitos de classes, objetos, atributos, métodos, construtores, sobrecarga, instância e referência de objetos, envio de mensagens, ciclo de vida de um objeto, abstração e encapsulamento.

A classe principal do sistema é a Livro, que possui os seguintes atributos: título, autor, preço e quantidade em estoque. A classe Livro também possui métodos para adicionar um novo livro, consultar os detalhes de um livro e atualizar a quantidade em estoque. Para garantir a modularidade e reutilização do código, foi criada uma classe Cliente, que contém atributos como nome, e-mail e um método para realizar um pedido.

Além disso, o sistema precisa permitir que um cliente realize um pedido com um ou mais livros, incluindo a possibilidade de calcular o valor total do pedido, levando em consideração as quantidades e os preços dos livros.
Para implementar esse sistema, o desenvolvimento exige o uso de abstração e encapsulamento nas classes, sobrecarga de métodos para o cálculo de valores diferentes de pedidos, instanciação e referência de objetos nas interações entre as classes, e envio de mensagens para a comunicação entre objetos.

Com base nesse cenário, marque a alternativa correta sobre o modelo de desenvolvimento utilizando os conceitos abordados.

A) O encapsulamento é fundamental para garantir que os atributos de um livro, como o preço e a quantidade em estoque, sejam acessados apenas através de métodos específicos, evitando modificações diretas. Por exemplo, o preço pode

ser alterado apenas por um método setPreco(), que realiza uma validação para garantir que o preço não seja negativo. Este processo representa um exemplo de abstração, onde os detalhes internos do objeto Livro são ocultados para o cliente do sistema.

B) A classe Pedido pode ser responsável pela instanciação dos objetos Livro dentro do método realizarPedido(). No entanto, a referência aos objetos Livro não pode ser feita diretamente, pois a classe Pedido deve possuir métodos auxiliares para o gerenciamento dos livros do pedido. Além disso, o ciclo de vida dos objetos Livro é encerrado após o pedido ser concluído, liberando os recursos do sistema.

C) Quando um cliente deseja consultar o estoque de um livro, o método consultarDetalhes() da classe Livro pode ser sobrecarregado para lidar com diferentes tipos de consultas, como a verificação do preço ou a quantidade disponível. O envio de mensagens entre objetos é realizado quando o objeto Cliente envia uma mensagem ao objeto Livro para acessar essas informações, resultando na execução do método consultarDetalhes().

D) O ciclo de vida de um objeto Livro é iniciado quando ele é instanciado pela classe Cliente durante o processo de realização de um pedido. A sobrecarga do método calcularTotal() é útil, pois permite calcular o valor do pedido dependendo de diferentes condições, como o número de livros solicitados ou o tipo de cliente (ex: estudante ou não). No entanto, é importante que todos os objetos sejam criados dentro do escopo do cliente para garantir a integridade dos dados.

E) Em um sistema orientado a objetos, o conceito de abstração implica que o comportamento do pedido, como o cálculo do valor total, deve ser implementado diretamente dentro da classe Livro. A classe Pedido deve apenas armazenar informações sobre o cliente, enquanto a instância do pedido cria e manipula os objetos Livro. Dessa forma, a classe Livro não deve conter métodos relacionados a pedidos, como o cálculo do valor total.

5 – Em uma lista simplesmente encadeada, cada nó contém um ponteiro para o próximo nó, formando uma sequência linear. A lista duplamente encadeada é mais flexível, pois cada nó contém dois ponteiros: um para o próximo nó e outro para o nó anterior. Isso permite a navegação em ambas as direções, mas resulta em um maior custo de memória, pois cada nó precisa armazenar dois ponteiros. A lista simplesmente encadeada é mais simples e consome menos memória, mas requer a travessia completa da lista para acessar elementos anteriores.

I. Em uma lista duplamente encadeada, a navegação para o nó anterior é mais eficiente do que em uma lista simplesmente encadeada.

II. A lista duplamente encadeada consome mais memória devido à necessidade de armazenar dois ponteiros por nó, o que pode tornar operações mais lentas.

III. Uma lista simplesmente encadeada oferece maior flexibilidade para manipulação de dados em comparação à lista duplamente encadeada.

IV. A operação de remoção de um nó é mais eficiente em uma lista duplamente encadeada, pois a referência ao nó anterior já está disponível.

V. As listas simplesmente encadeadas são mais eficientes em termos de uso de memória, uma vez que cada nó contém apenas um ponteiro.

Assinale a alternativa correta:

a) I, III, V.

b) II, IV, V.

c) I, II, IV.

d) III, IV, V.

e) II, III, IV.

6 – Em uma lista circular, o último nó da lista aponta para o primeiro nó, formando um ciclo contínuo. Isso permite navegar continuamente pela lista sem verificar se o final foi alcançado. Listas circulares são úteis em algoritmos que requerem repetição contínua de operações, como o algoritmo round-robin em sistemas de escalonamento de processos. Uma lista circular simples pode ser implementada de maneira semelhante a uma lista simplesmente encadeada, mas com a diferença de que o último nó aponta de volta ao primeiro, mantendo a circularidade. As operações de inserção e remoção em listas circulares podem ser tão eficientes quanto em listas duplamente encadeadas, mas a vantagem principal é o acesso contínuo aos dados.
I. O último nó de uma lista circular aponta de volta para o primeiro nó, formando um ciclo contínuo.

II. A principal vantagem das listas circulares é o acesso contínuo aos dados, sem a necessidade de verificar se o final da lista foi alcançado.

III. Listas circulares são mais eficientes em termos de complexidade de tempo para operações de inserção e remoção do que as listas duplamente encadeadas.

IV. Uma lista circular simples pode ser implementada de maneira semelhante a uma lista simplesmente encadeada, mas o último nó aponta para o primeiro, criando o ciclo.

V. As listas circulares são mais vantajosas quando usadas em sistemas que exigem repetição contínua de operações, como algoritmos round-robin.

Assinale a alternativa correta:

a) I, III, V.

b) II, IV, V.

c) I, II, IV.

d) III, IV, V.

e) II, III, IV.

7 – Uma **multilista** é uma estrutura de dados composta por várias listas encadeadas, onde cada lista pode representar uma categoria ou dimensão distinta de dados. Cada nó de uma **multilista** pode conter um ponteiro para outra lista, permitindo a organização hierárquica dos dados. Por exemplo, em um sistema educacional, uma **multilista** poderia ser usada para representar as informações de alunos, onde cada lista associada a um aluno conteria suas disciplinas. Embora as **multilistas** possam ser implementadas usando listas encadeadas simples ou duplamente encadeadas, elas não requerem listas circulares.

I. Uma multilista é composta por várias listas encadeadas, onde cada lista representa uma categoria ou dimensão de dados.

II. Cada nó em uma multilista pode conter um ponteiro para outra lista, o que permite a organização hierárquica dos dados.

III. A implementação de uma multilista é feita exclusivamente usando listas duplamente encadeadas.

IV. As multilistas podem ser usadas para representar sistemas complexos, como a organização de informações de alunos, com cada lista representando um conjunto de dados relacionado.

V. As multilistas são frequentemente implementadas utilizando listas circulares, embora possam também ser implementadas com listas encadeadas simples ou duplamente encadeadas.

Assinale a alternativa correta:

a) I, II, III

b) II, III, IV

c) I, II, V

d) III, IV, V

e) I, II, III, V

8 – Uma **árvore binária** é uma estrutura de dados em que cada nó tem no máximo dois filhos: o filho à esquerda e o filho à direita. Esse tipo de estrutura é amplamente utilizada em diversos algoritmos e aplicações, como na implementação de **árvores binárias de busca (ABB)**. Em uma **árvore binária de busca**, para cada nó, todos os elementos à esquerda são menores que o nó, e todos os elementos à direita são maiores, o que permite buscas eficientes em tempo logaritmo (O(log n)).
A **busca binária** é uma técnica eficiente de pesquisa em listas ordenadas, onde a busca se divide progressivamente pela metade, eliminando metade da busca a cada iteração. Quando implementada em **árvores binárias de busca**, ela pode realizar buscas, inserções e remoções de maneira eficiente.

I. Em uma árvore binária de busca, os elementos à esquerda de cada nó são menores, e à direita são maiores que o nó.

II. As operações de inserção e remoção em uma árvore binária de busca podem ter complexidade O(n) no pior caso.

III. Em uma árvore binária de busca balanceada, o tempo de busca, inserção e remoção é O(log n).

IV. As árvores binárias de busca permitem uma busca eficiente em tempo linear, O(n), em todos os casos.

V. A busca binária em uma árvore binária de busca é uma técnica ineficiente, pois não há divisão sucessiva do espaço de pesquisa.

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

9 – Dado a situação problema de um vetor de inteiros V com n elementos, qual é a complexidade de tempo para encontrar o valor máximo e o valor mínimo do vetor utilizando um único laço de repetição nessa execução.

Analise as informações:

Considere que o vetor V é indexado a partir de 0, ou seja, o primeiro elemento está na posição V[0] e o último elemento está na posição V[n-1].
A complexidade de tempo, representado pela letra grega "teta" ou O, se refere ao tempo de execução do algoritmo em relação ao tamanho do vetor n.
A opção correta deve indicar corretamente a complexidade de tempo do algoritmo mais eficiente para encontrar o valor máximo e o valor mínimo do vetor utilizando um único laço de repetição.

Assinale a alternativa correta:

a) O(1)

b) O(n)

c) O(n^2)

d) O(log n)

e) O(n log n)

10 – Dentre as seguintes afirmações qual é a verdadeira em relação aos atributos e propriedades em programação orientada a objetos, tendo como base as seguintes informações:

Propriedades podem ser chamados de Getters e Setters em algumas linguagens de programação.

A opção correta deve indicar corretamente uma característica verdadeira sobre atributos e propriedades em programação orientada a objetos, considerando sua definição e uso convencional

a) Atributos e propriedades são sinônimos e são usados indistintamente para se referir a características de um objeto em uma classe.

b) Atributos são variáveis privadas de uma classe, enquanto propriedades são métodos públicos de acesso aos atributos.

c) Atributos são métodos públicos de acesso aos dados em uma classe, enquanto propriedades são variáveis privadas.

d) Atributos e propriedades são mecanismos semelhantes para representar dados em uma classe, com diferentes níveis de encapsulamento e acesso.

e) Atributos e propriedades são conceitos que não estão presentes em linguagens de programação orientadas a objetos.

11 – (ENADE - TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS – 2021 - Adaptada). Uma fundação municipal de arte e cultura oferece cursos para crianças e adolescentes de até 16 anos. Os registros de matrículas são realizados em uma planilha eletrônica ilustrada a seguir, o que dificulta bastante o controle dos dados frente ao número expressivo de

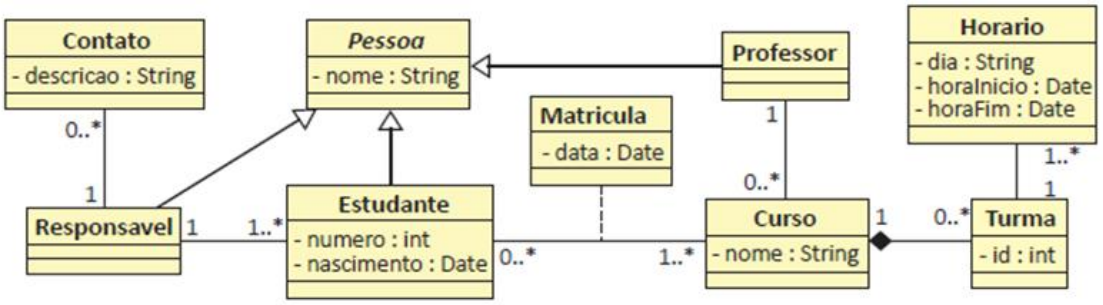
estudantes, muitos dos quais solicitam matrícula em mais de um curso. A administração da fundação, então, solicitou o desenvolvimento de um programa para facilitar o registro e a consulta desses dados, incluindo-se a data de efetivação de cada matrícula, para fins de controles específicos da secretaria.

NÚMERO	NOME ESTUDANTE	DATA NASC.	NOME RESPONSÁVEL	CONTATOS	CURSO	HORÁRIO	PROFESSOR(A)
10	Ada Lovelace	10/12/2009	Charles Babbage	(99)99999-0101 charles@babbage.com	Violão Básico Desenho	9h - 10h (ter - qui) 10h - 11h (seg - qua)	Dennis Ritchie John Backus
11	Ole-Johan Dahl	12/10/2008	Kristen Nygaard	-	Desenho	9h - 10h (seg - qua)	Mary Keller
12	Grace Hopper	09/12/2010	Howard Aiken	(99)99999-0091 (99)99999-0095	Balé	8h - 10h (sex)	Hedy Lamarr
13	Alan Turing	23/06/2009	Joan Clarke	(99)99999-0231	Desenho Violão Básico	9h - 10h (seg - qua) 9h - 10h (ter - qui)	Mary Keller Dennis Ritchie
14	Dorothy Vaughan	20/09/2009	Katherine Johnson	(99)99999-0201	Balé	8h - 10h (sex)	Hedy Lamarr
...	...	...	...	...	...	...	...

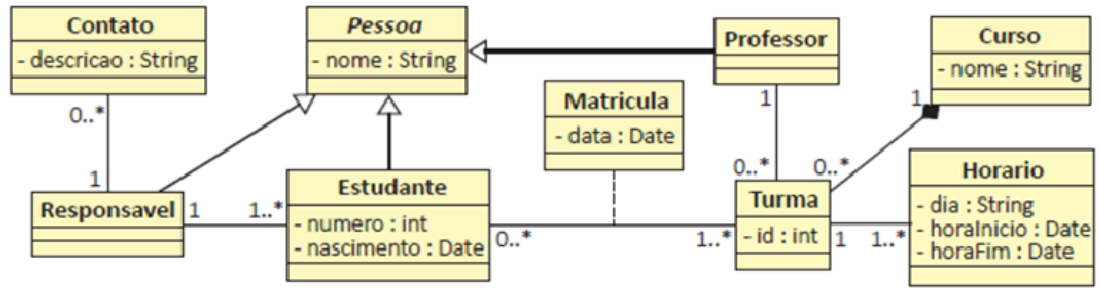
A equipe de desenvolvimento de softwares da prefeitura, após uma breve entrevista com a administração da fundação e de posse da planilha eletrônica, modelou um Diagrama de Classes como parte da especificação dos requisitos do sistema.

Considerando o cenário descrito, assinale a opção a seguir que exibe o Diagrama de Classes modelado corretamente pela equipe de desenvolvimento.

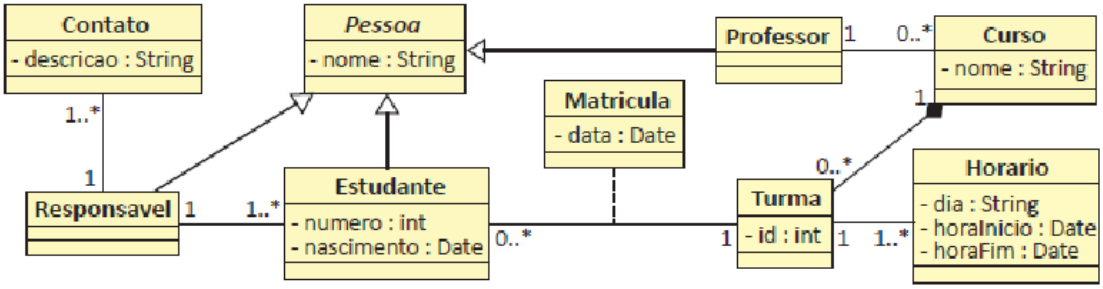
a)



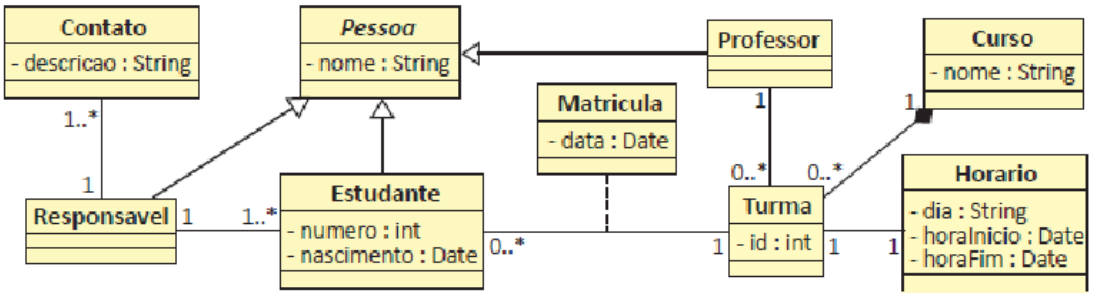
b)


Contato
- descricao : String
Pessoa
- nome : String
Professor
Curso
- nome : String
Matricula
- data : Date
Responsavel
Estudante
- numero : int
- nascimento : Date
Turma
- id : int
Horario
- dia : String
- horaInicio : Date
- horaFim : Date

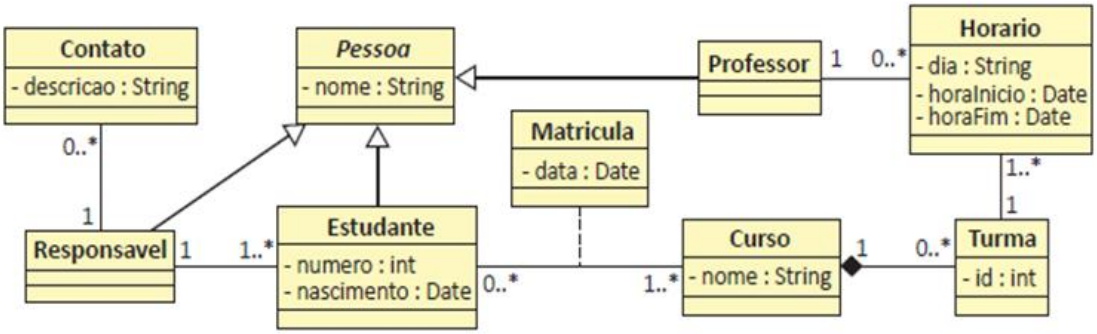
c)


Contato
- descricao : String
Pessoa
- nome : String
Professor
Curso
- nome : String
Matricula
- data : Date
Responsavel
Estudante
- numero : int
- nascimento : Date
Turma
- id : int
Horario
- dia : String
- horaInicio : Date
- horaFim : Date

d)


Contato
- descricao : String
Pessoa
- nome : String
Professor
Curso
- nome : String
Matricula
- data : Date
Responsavel
Estudante
- numero : int
- nascimento : Date
Turma
- id : int
Horario
- dia : String
- horaInicio : Date
- horaFim : Date

e)


Contato
- descricao : String
Pessoa
- nome : String
Professor
Horario
- dia : String
- horaInicio : Date
- horaFim : Date
Matricula
- data : Date
Responsavel
Estudante
- numero : int
- nascimento : Date
Curso
- nome : String
Turma
- id : int

12 – (ENADE - TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS – 2021 - Adaptada). Uma fábrica de software está realizando entrevistas para contração de um profissional que esteja alinhado às exigências do atual mundo corporativo. Sabe-se que processos ágeis de desenvolvimento têm se tornado essenciais para empresas que desejam realizar entregas rápidas e frequentes de produtos e/ou serviços de software. Essa empresa possui uma equipe de desenvolvimento que faz uso de processos ágeis como o Scrum e eXtreme Programming (XP) e o acompanhamento por meio do quadro Kanban. Sendo assim, um conjunto de características deve ser verificado durante a entrevista para garantir que o candidato a ser contratado possua conhecimentos necessários para atuar juntamente a esta equipe.

Com base no texto e nos processos ágeis de desenvolvimento de software, avalie as afirmações a seguir.

I. Métodos Ágeis são baseados em ciclos iterativo e incremental que se concentram no desenvolvimento rápido e na flexibilidade às mudanças, com a participação do cliente no processo de software.

II. Uma forte característica da XP é a garantia da qualidade do código produzido e, para isso, os desenvolvedores produzem testes automatizados antes mesmo de codificar

uma funcionalidade.

III. O planejamento no Scrum é baseado na elaboração dos itens do product backlog, que é uma lista de funcionalidades desejadas pelo cliente, sendo o Scrum Master o responsável por gerenciá-lo.

IV. O quadro Kanban permite monitorar a evolução das tarefas necessárias durante o processo ágil de desenvolvimento de software, possibilitando um acompanhamento de forma visual das atividades em construção.

É correto apenas o que se afirma em:

a) II.

b) I e III.

c) I, II e IV.

d) I, III e IV.

e) II, III e IV.

13 – A orientação a objetos (OO) é um paradigma de programação que utiliza “objetos” – instâncias de classes – como unidades fundamentais de construção de software. Este paradigma foi popularizado por autores como Grady Booch, que em sua obra “Object-Oriented Analysis and Design with Applications” (BOOCH, 2007), descreve a OO como uma abordagem que facilita a modelagem de sistemas complexos através da abstração, encapsulamento, herança e polimorfismo. Segundo Booch, a OO permite que desenvolvedores criem sistemas mais robustos e reutilizáveis, promovendo uma melhor organização do código e facilitando a manutenção e evolução do software.

BOOCH, Grady. Object-Oriented Analysis and Design with Applications. 3rd ed. Addison-Wesley, 2007. Adaptado.

Com base na leitura do texto acima e conhecimentos sobre Orientação a Objetos, assinale a alternativa correta:

a) a herança permite que os membros de uma classe, chamada de classe-mãe, possam ser reaproveitados na definição de outra classe, chamada de classe-filha. Esta classe-filha tem acesso aos membros públicos e protegidos da classe-mãe. 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.

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.

c) 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-mãe. O acesso aos atributos da classe-mãe independe do modificador utilizado.

d) o conceito de herança estabelece que uma classe possa aproveitar a implementação, definições dos atributos e métodos de uma classe-base. A classe-filha pode ter acesso aos métodos e atributos públicos e protegidos da classe-base. O polimorfismo é aplicado ao caso em que existe a necessidade de implementar métodos sobrecarregados, nos quais a classe-filha necessita implementar dois métodos com o mesmo nome e parâmetros diferentes.

e) o polimorfismo é uma técnica que permite um objeto nascer a partir do uso de sobrecarga de construtores de uma classe, ou seja, o polimorfismo permite que um objeto possa ser instanciado de diferentes maneiras. A herança permite que uma classe sirva de base para que outras classes sejam implementadas. Entretanto, os membros com modificadores públicos da classe-base podem ser acessados pela classe-filha.

14 – Considere o código Java apresentado:

```
public class Disciplina {

private String nome;

private int codigo;

public Disciplina(String nome, int codigo) {

this.nome = nome;

this.codigo = codigo;

}

public String toString() {

return "Disciplina: " + nome + ", codigo: " + codigo;

}

}

----------------------------------------------------------------------

public class Professor {

private Disciplina disciplina;

private String data;

public Professor(Disciplina disciplina, String data) {

this.disciplina = disciplina;

this.data = data;

}

public Professor() {}
```

```
public String toString() {

return "Professor\nDisciplina: " + disciplina + ", data: " + data + '}';

}

}
```

```
public class Programa {

public static void main(String[] args) {

Disciplina disc1 = new Disciplina("Português",012);

Disciplina disc2 = new Disciplina("Matemática",016);

Professor professor1 = new Professor(disc1,"01/12/2022");

System.out.println(professor1);

Professor professor2 = new Professor(disc1,"04/01/2023");

System.out.println(professor2);

}

}
```

Com base no código apresentado e nos estudos realizados, analise as assertivas e assinale a alternativa correta.

I. O código da classe Programa implementa de forma correta o uso dos atributos e métodos das classes.

II. Se os objetos professor1 e professor2 forem excluídos, o objeto disc1 continua ativo no sistema;

III. O objeto professor2 foi construído utilizando o construtor com parâmetros da classe, mas poderia ser construído sem parâmetros;

IV. O uso de polimorfismo aparece no código, ao associar Professor a Disciplina;

Assinale a alternativa correta:

a) I e II.

b) II e III.

c) I, III e IV.

d) I, II e III.

e) III e IV.

15 – O paradigma orientado a objetos é um dos mais populares na programação moderna, sendo utilizado por várias linguagens. Linguagens como Python, C++, Ruby e C# implementam o paradigma de maneira diferente, mas todas compartilham os principais conceitos: classes, objetos, herança, polimorfismo e encapsulamento. De acordo com Sweigart (2019), a escolha de uma linguagem orientada a objetos depende das necessidades do projeto, como a simplicidade da sintaxe, a flexibilidade no uso de objetos e a capacidade de manipulação de dados. A linguagem que melhor se adapta ao paradigma orientado a objetos deve permitir a criação eficiente de classes e a manipulação de objetos de forma intuitiva e eficaz.
Considere os seguintes exemplos de código em diferentes linguagens de programação que implementam o paradigma orientado a objetos. Qual das alternativas abaixo apresenta a melhor linguagem para o paradigma orientado a objetos, excluindo o Java, com base no código e explicação de seu funcionamento:

a) Python:
class Animal: def __init__(self, name): self.name = name def speak(self):
return "Some generic sound"
class Dog(Animal):
def speak(self):
return "Woof"
dog = Dog("Buddy")
print(dog.speak())

Descrição:
Python utiliza uma sintaxe simples e clara para definir classes e herança. O exemplo demonstra como a herança funciona, onde a classe Dog herda da classe Animal, e o método speak() é sobreposto para gerar um comportamento específico para o objeto Dog.

b) C++:
#include <iostream>
using namespace std;

```
class Animal {public:
string name;
Animal(string n) : name(n) {}
virtual void speak() {
cout << "Some generic sound" << endl;
}
};
class Dog : public Animal {
public:
Dog(string n) : Animal(n) {}
void speak() override {
cout << "Woof" << endl;
}
};
int main() {
Dog dog("Buddy");
dog.speak();
return 0;
}
```

Descrição:
Em C++, o exemplo usa a palavra-chave virtual para permitir o polimorfismo. A classe Dog sobrescreve o método speak() da classe base Animal. A herança e o polimorfismo são bem definidos, mas a sintaxe de C++ é mais verbosa comparada a Python.

c) Ruby:

```
class Animal
def initialize(name)
@name = name
end
def speak
"Some generic sound"
end
end
class Dog < Animal
def speak
"Woof"
end
end
dog = Dog.new("Buddy")
puts dog.speak
```

Descrição:

Ruby possui uma sintaxe limpa e legível, similar à de Python. A classe Dog herda da classe Animal, e o método speak é sobrescrito. O Ruby também permite que objetos sejam criados de maneira simples com o uso do new.

d) C#:

```
using System;
public class Animal{ public string Name { get; set; }
public virtual void Speak()
{
Console.WriteLine("Some generic sound");
}
}
public class Dog : Animal
{
public override void Speak()
{
Console.WriteLine("Woof");
}
}
class Program
{
static void Main()
{
Dog dog = new Dog();
dog.Speak();
}
}
```

Descrição:
C# é similar ao C++ em termos de orientação a objetos, mas com uma sintaxe mais moderna e recursos como virtual e override para facilitar o polimorfismo. A linguagem fornece um bom suporte para propriedades e encapsulamento, tornando-a robusta para a programação orientada a objetos.

e) Swift:

```
class Animal { var name: String init(name: String) {
self.name = name
}
func speak() -> String {
return "Some generic sound"
}
}
class Dog: Animal {
override func speak() -> String {
return "Woof"
```

```
}
}
let dog = Dog(name: "Buddy")
print(dog.speak())
```

Descrição:
Swift, similar a outras linguagens orientadas a objetos, utiliza classes e herança de forma clara e objetiva. A classe Dog herda de Animal e sobrescreve o método speak(). Swift se destaca pela sua sintaxe concisa e moderna, especialmente em ambientes de desenvolvimento para iOS e macOS, tornando a implementação do paradigma orientado a objetos simples e direta.

16 – Em um sistema de e-commerce, os produtos são armazenados e classificados em categorias. A busca por produtos no sistema precisa ser rápida e eficiente, especialmente quando há centenas de milhares de itens disponíveis. O sistema utiliza uma árvore binária de busca para organizar os produtos e aplicar técnicas de ordenação para otimizar a pesquisa e exibição dos produtos. O código a seguir apresenta uma implementação simples de uma árvore binária de busca e uma técnica de ordenação de quicksort para organizar os produtos.
Árvore Binária de Busca:

```
class ArvoreBinaria {
private class Nodo {
String produto;
Nodo esquerda;
Nodo direita;
Nodo(String produto) {
this.produto = produto;
esquerda = direita = null;
}
}
private Nodo raiz;
public ArvoreBinaria() {
raiz = null;
}
public void inserir(String produto) {
raiz = inserirRecursivo(raiz, produto);
}
private Nodo inserirRecursivo(Nodo raiz, String produto) {
if (raiz == null) {
raiz = new Nodo(produto);
return raiz;
}
if (produto.compareTo(raiz.produto) < 0) {
raiz.esquerda = inserirRecursivo(raiz.esquerda, produto);
} else if (produto.compareTo(raiz.produto) > 0) {
raiz.direita = inserirRecursivo(raiz.direita, produto);
```

```
}
return raiz;
}
public void buscar(String produto) {
Nodo resultado = buscarRecursivo(raiz, produto);
if (resultado != null) {
System.out.println("Produto encontrado: " + resultado.produto);
} else {
System.out.println("Produto não encontrado.");
}
}
private Nodo buscarRecursivo(Nodo raiz, String produto) {
if (raiz == null || raiz.produto.equals(produto)) {
return raiz;
}
if (produto.compareTo(raiz.produto) < 0) {
return buscarRecursivo(raiz.esquerda, produto);
}
return buscarRecursivo(raiz.direita, produto);
}
}
```

Técnica de Ordenação - Quicksort:

```
class Ordenacao {
public static void quicksort(String[] produtos, int inicio, int fim) {
if (inicio < fim) {
int p = particionar(produtos, inicio, fim);
quicksort(produtos, inicio, p - 1);
quicksort(produtos, p + 1, fim);
}
}
private static int particionar(String[] produtos, int inicio, int fim) {
String pivot = produtos[fim];
int i = (inicio - 1);
for (int j = inicio; j < fim; j++) {
if (produtos[j].compareTo(pivot) < 0) {
i++;
String temp = produtos[i];
produtos[i] = produtos[j];
produtos[j] = temp;
}
}
String temp = produtos[i + 1];
produtos[i + 1] = produtos[fim];
produtos[fim] = temp;
return i + 1;
}
```

}

O sistema de e-commerce utiliza uma árvore binária de busca para armazenar produtos e uma técnica de ordenação quicksort para organizar a lista de produtos. Explique como a árvore binária de busca facilita a pesquisa de produtos, destacando a complexidade das operações de inserção e busca. Em seguida, discuta os benefícios do uso do quicksort para ordenar os produtos, explicando sua complexidade e quando seria mais vantajoso usar outras técnicas de ordenação.

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____