Nome: ANDERSON RICARDO XAVIER **Data: 24/05/2025**

Responda em português ou inglês.

1. **Write a Java program to solve the following problem:**

You are tasked with creating a utility function for a text-processing application. The function must generate all possible anagrams from a given group of distinct letters. For example, the input {a, b, c} should return the output: abc, acb, bac, bca, cab, cba.

Additional Requirements:

1. **The program should accept any group of distinct letters as input and produce the correct result.**
2. **Optimize for readability and clarity.**
3. **Include basic validation (e.g., ensure the input is non-empty and contains only letters).**
4. **Write unit tests to validate your function with at least three different test cases, including edge cases (e.g., input with a single letter or empty input).**
5. **Document your code clearly, explaining the logic for generating anagrams.**

1. **Provide an example scenario where overriding the equals() method is necessary in Java. Explain the key considerations when implementing this method, such as ensuring it aligns with the hashCode() method. Include code examples if possible.**

   R: A sobrescrita do método equals é necessária quando desejamos compar dois objetos por conteúdo, pois o equals padrão compara a referência, não atributos.
   Exemplo: Objeto Pessoa com dois atributos, nome e cpf.

   ```
   Pessoa p1  = new Pessoa(“Anderson”, “123456789);
   Pessoa p2  = new Pessoa(“Anderson”, “123456789);
   p1.equals(p2)
   ```
   Isso Imprime false, porque o equals compara referência, não atributos.

   Então se sobrescrever o método.
   ```
   @Override
   Public bolean equal(Object obj){
       If(this == obj) return true;
       If(obj == null || getClass() != obj.getClass()) return false;
       Pessoa pessoa = (Pessoa) obj;
       Return Objects.equals (cpf, pessoa.cpf); // cpf único;
   }

   @Override
   Public int hashCode(){
       Return Objetcts.has(cpf);
   }
   ```

   Se usarmos os objetos em estrutruras basedas em hash, como HashSet, HashMap o contrato do java exige que:
   Se a.equals(b) for verdadeiro, então a.hashCode() == b.hashCode() deve ser verdadeiro.
   Não sobrescrever o método hashCode() pode causar comportamentos incorretos em coleções como HashSet()

   Então assim sobrescrito, temos;
   ```
   Set<Pessoa> pessoas = new HashSet<>();
   pessoas.add(p1);
   pessoas.add(p2);
   ```

   Como o cpf é único, ao incluir dois objetos com o mesmo cpf a rotia só permite um na lista;

2. **Explain how you would use a design pattern to decouple your code from a third-party library that might be replaced in the future. Describe the advantages and limitations of your chosen approach, and provide a small code snippet illustrating its application.**

R: Para desacoplar meu Código de uma bilbioteca de terceiros que pode ser subistituida futuramente, a melhor abordagem é usar o padrão de design de Adapter ou Wapper junto com o princípio da inversão de dependêcia (DIP)

Vamos supor que estamos usando uma biblioteca de envio de e-mails como JavaMail, mas que futuramente vamos subistiuir por (Amazon SES ou SendGrid).

1 Definimos uma Interface (contrato)

```
public interface EmailService {

   void enviarEmail(String para, String assunto, String corpo);

}
```

2 Criamos um adapter para bilioteca atual

```
public class JavaMailEmailService implements EmailService {
@Override
 public void enviarEmail(String para, String assunto, String corpo) {
      System.out.println("Enviando com JavaMail para " + para);
   }
}
```

Utilizamos a interface no código (desacoplado)

```
public class Notificador {

   private final EmailService emailService;

   public Notificador(EmailService emailService) {

            this.emailService = emailService;

   }

   public void notificar(String usuario) {

      emailService.enviarEmail(usuario, "Bem-vindo", "Obrigado por se cadastrar.");

   }

}
```

Vantagens:

Desacoplamento total.
Facilidade para testar, podendo usar mocks.
Flexibilidade para torocar a implementação sem alterar o Código principal.
Manutenção facilitada, já que o impacto é isolado.

Desvantagens:

Requer implementação extra.
Pode esconder funcionalidades específicas da bilblioteca.
Se a nova biblioteca tiver comportamentos muito diferentes, o adpter pode crescer e virar um anti-pattern.

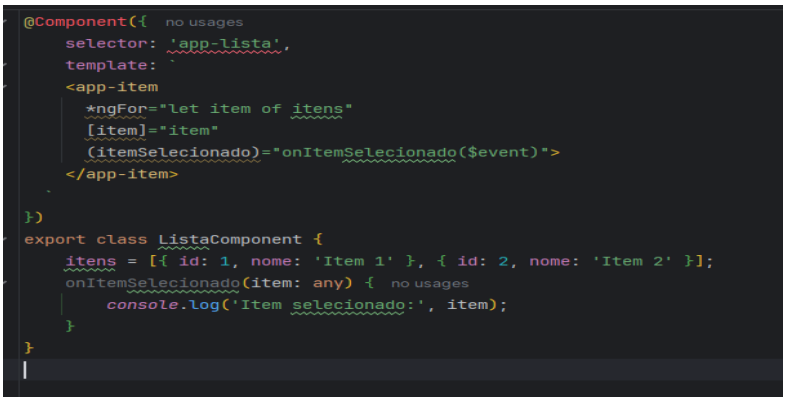
3. **Describe your experience with Angular, including its core features and use cases. Provide an example of a practical application where you used Angular and include a code snippet demonstrating a key feature, such as component communication, data binding, or service integration.**

R: Na empresa Pacto, tive a oportunidade de atuar na migração de uma aplicação monolítica para uma arquitetura baseada em microserviços.
Com essa mudança, realizamos também a transição do frontend, que anteriormente era desenvolvido em JSF, para o Angular.
Essa mudança nos permitiu modernizar a interface, melhorar a experiência do usuário e separar responsabilidades entre frontend e backend.

Durante esse projeto, trabalhei com Angular em funcionalidades como:
Criação de componentes reutilizáveis;
Comunicação entre componentes (via @Input/@Output e Services com Subject);
Integração com APIs REST via HttpClient;
Implementação de formulários reativos (ReactiveForms);
Uso de Guards e Roteamento para controle de acesso;
Boas práticas com serviços e modularização do código.

Exemplo prático: Comunicação entre componentes:

Na aplicação, tínhamos um componente pai que exibia uma lista de itens e um componente filho que permitia selecionar um item da lista. Utilizamos @Input e @Output para a comunicação entre eles.
Algo assim: Componete Pai (lista.components.ts)



```
@Component({  no usages
    selector: 'app-lista',
    template: `
    <app-item
      *ngFor="let item of itens"
      [item]="item"
      (itemSelecionado)="onItemSelecionado($event)">
    </app-item>
  `
})
export class ListaComponent {
    itens = [{ id: 1, nome: 'Item 1' }, { id: 2, nome: 'Item 2' }];
    onItemSelecionado(item: any) {  no usages
        console.log('Item selecionado:', item);
    }
}
```

Componente Filho (Item.components.ts)

```
@Component({
  selector: 'app-item',
  template: `
    <div (click)="selecionarItem()">
      {{ item.nome }}
    </div>
  `
})
export class ItemComponent {
  @Input() item: any;
  @Output() itemSelecionado = new EventEmitter<any>();

  selecionarItem() {
    this.itemSelecionado.emit(this.item);
  }
}
```

Esse padrão nos ajudou a manter o código desacoplado e facilitar reutilização dos componentes em outras telas.

4. **Discuss the techniques you use to prevent SQL injection attacks in web applications. Provide examples of code showing secure implementations, such as using parameterized queries or ORMs. Mention any additional measures you take to secure the database layer.**

R: Para previnir ataques de injeção de SQL, sigo uma série de boas práticas em todas as aplicações que desenvolvo, principalmente utilizando frameworks modernos e ORMs que já oferecem segurança nativa contra esse tipo de ameaça.
Em java sempre que possivel utilizando o Spring Data JPA.

@Query(“Select u from Usuario u Where u.email =:email)
Usuario findByEmail(@Parm(“email) String email);
Aqui o parâmetro :email é automaticamente tratado pelo framework, evitando SQL injection.

Com JDBC – sem ORM
Sting sql = “Select u from Usuario u Where u.email = ?”;
PreparedStatement stm = connection.preperStamtement(sql);
Stm.setString(1, email);
ResultSet rs = stm.executeQuery();

Além disso, valido e restinjo os dados de entrada tanto no frontend quanto no backend, quando possível implemento WAF- Web Aplication Firewall essa camada de segurança monitora, filtra e bloqueia tráfegos HTTP maliciosos antes que ele chegue até a aplicação.

Além do Famosa Injeção de SQL ela tabém protege cross-site, execução remota de commandos, ataque de força bruta entre outros.

5. **Describe the steps you would take to diagnose and improve the performance of a batch process that interacts with a database and an FTP server. Explain how you would identify bottlenecks, optimize database queries, improve logic execution, and enhance file transfer efficiency. Provide examples of tools or techniques you would use during the analysis.**

   R: Para diagnosticar e melhorar o desemplenho de um porcesso em lote sigo a seguinte abordagem.

   Medição de tempo gasto em cada etapa do processo (leitura, gravação, envio)
   Inserção de logs como timestamps para detectar qual parte do processo está consumindo mais tempo.
   Utilizo ferramentas quando necessário como:
   Spring Actuator + Micrometer, Prometheus + Grafana para métrica em tempo real.

   Se o problema for em uma consulta ao banco de dados.
   Evitar consultas dentro de loops.
   Usar consultas paginadas em lotes
   Criar indices adequados nas colunas utilizadas para filtros e joins.
   verificar o plano de execução (EXPLAIN, EXPLAIN AWALYZE) das queries para identificar gargalos.

   Para ter mais eficiência na transferência de arquivos via FTP.

   Compactar os arquivos antes de transferir (zip ou gzip)
   Utilizar FTP passivo para resduzir falhas de conexão em firewalls
   Avaliar a troca por protocolos mais modernos como SFTP ou FTPS.
   Fazer transferência paralela ou incremental.

Salesperson

ID	Name	Age	Salary
1	Abe	61	140000
2	Bob	34	44000
5	Chris	34	40000
7	Dan	41	52000
8	Ken	57	115000
11	Joe	38	38000

Customer

ID	Name	City	Industry Type
4	Samsonic	Pleasant	J
6	Panasung	Oaktown	J
7	Samony	Jackson	B
9	Orange	Jackson	B

Orders

ID	order_date	customer_id	salesperson_id	Amount
10	8/2/96	4	2	540
20	1/30/99	4	8	1800
30	7/14/95	9	1	460
40	1/29/98	7	2	2400
50	2/3/98	6	7	600
60	3/2/98	6	7	720
70	5/6/98	9	7	150

6. **Given the tables above, write the SQL query that:**

a. Returns the names of all Salesperson that don't have any order with Samsonic.

R:

```
SELECT s.name
FROM salesperson s
WHERE s.id NOT IN (
   SELECT o.salesperson_id
   FROM orders o
   JOIN customer c ON o.customer_id = c.id
   WHERE c.name = 'Samsonic'
);
```

b. Updates the names of Salesperson that have 2 or more orders. It's necessary to add an '*' in the end of the name.

R:

```
UPDATE salesperson
SET name = name || '*'
WHERE id IN (
   SELECT salesperson_id
```

```
    FROM orders
    GROUP BY salesperson_id
    HAVING COUNT(*) >= 2
);
```

c. Deletes all Ssalesperson that placed orders to the city of Jackson.
R:

```
DELETE FROM salesperson
WHERE id IN (
    SELECT DISTINCT o.salesperson_id
    FROM orders o
    JOIN customer c ON o.customer_id = c.id
    WHERE c.city = 'Jackson'
);
```

d. The total sales amount for each Salesperson. If the salesperson hasn't sold anything, show zero.
R:

```
SELECT s.name, COALESCE(SUM(o.amount), 0) AS total_vendas
FROM salesperson s
LEFT JOIN orders o ON s.id = o.salesperson_id
GROUP BY s.id, s.name;
```

7. **The customer has a system called XYZ and intends to start updates split into 3 phases. The requirements for the first phase are as follows:**

    1. **Enable new data entries in the system, which will serve as input for the second phase.**
    2. **Implement functionality to create, update, delete, and search plants.**
        - **Plants should have the following attributes:**
            - **Code: Numeric only, mandatory, and unique.**
            - **Description: Alphanumeric, up to 10 characters, optional.**
        - **Only admin users can delete plants.**
    3. **Ensure that the system prevents duplication of plant codes.**

Task:
Based on the above information:

1. **Write a use case or user story for this scenario, ensuring that it clearly addresses the requirements.**

2. **Highlight any business rules or assumptions relevant to the solution.**
3. **Describe any validations or security measures you would implement in the system.**
4. **Suggest how you would test this functionality, including examples of edge cases.**

R:

Caso de Uso / História de Usuário

Como administrador do sistema XYZ,
quero criar, atualizar, excluir e pesquisar plantas com código numérico único e descrição opcional,
para que os dados fiquem prontos para serem utilizados na segunda fase da atualização do sistema.

Regras de Negócio e Premissas

Regras de Negócio:

- O código da planta:
    - É obrigatório.
    - Deve conter somente números.
    - Deve ser único no sistema.
- A descrição da planta:
    - É opcional.
    - Deve conter no máximo 10 caracteres alfanuméricos.
- Somente usuários com perfil de administrador podem excluir plantas.
- É permitido criar, editar e pesquisar plantas para qualquer usuário com acesso ao sistema.
- Os dados de plantas inseridos servirão de entrada para processos na segunda fase. Premissas:
- O sistema possui controle de autenticação e identificação do perfil do usuário.
- O banco de dados tem capacidade de garantir unicidade de campos (via chave única).
- As ações CRUD serão expostas em uma interface segura (ex: API REST + frontend ou interface administrativa).

Validações e Medidas de Segurança

Validações no Backend e/ou Frontend:

- Validar que o código:
    - Está presente;
    - Contém apenas números (^\d+$);
    - Não se repete (verificação de duplicidade).
- Validar que a descrição:
    - Tem no máximo 10 caracteres;
    - Contém apenas caracteres alfanuméricos (se aplicável).
- Validar que o usuário tem permissão para exclusão (checar o perfil).

Segurança:

- Autenticação e controle de acesso (ex: JWT ou OAuth2).
- Autorização baseada em perfis (ex: ROLE_ADMIN para exclusão).
- Proteção contra injeção SQL (uso de ORM e queries parametrizadas).
- Log de operações críticas (exclusão de plantas, com auditoria de quem executou).

Estratégia de Testes (com casos extremos)

Testes funcionais:

- Criar planta com código válido e descrição válida.
- Criar planta **sem descrição** (opcional).
- Tentar criar planta com **código repetido** → deve falhar.
- Criar planta com **código alfanumérico** → deve falhar.
- Criar planta com **descrição maior que 10 caracteres** → deve falhar.
- Atualizar planta existente.
- Excluir planta como usuário **não administrador** → deve falhar.
- Excluir planta como **administrador** → deve passar.
- Pesquisar plantas por código ou descrição (verificar filtros e paginação).

Testes de segurança:

- Tentar acessar endpoints de exclusão **sem autenticação** → deve negar.
- Tentar acessar com token inválido ou sem perfil autorizado → deve negar.
- Verificar se logs são gerados corretamente nas ações críticas.

Casos extremos:

- Criar planta com código de **máximo valor permitido** (ex: 9999999999).
- Criar planta com código 0 (se permitido).
- Criar várias plantas rapidamente (teste de concorrência/duplicidade).
- Tentativa de **injeção de código** nos campos (XSS, SQLi etc.).

8. **Consider the following description of a system functionality:**
   User Registration
   - **A screen allows users to insert, delete, or update user information.**
   - **Each user has properties: name, email, address, and phone, where name and email are mandatory fields.**
   - **Emails must be unique across all users.**
   - **Only admin users can delete other users.**

   Task:
   1. **Describe the types of tests you would implement (e.g., unit, integration, or end-to-end tests) and explain the scenarios you would test to ensure the functionality works as expected.**
   2. **Provide examples of edge cases and how you would handle them.**
   3. **Include an example of a test case in code or pseudocode for one or more scenarios.**

Tipos de testes e cenários cobertos

Testes Unitários

- **Testam funções isoladas, como validações e regras de negócio.**

Exemplos de cenários:

- **Validar que nome e e-mail são obrigatórios.**
- **Validar formato de e-mail.**
- **Verificar se e-mail é único antes do cadastro.**
- **Garantir que a lógica de permissão para exclusão só permite admins.**

Testes de Integração

- **Testam a integração entre componentes como controlador, serviço, repositório e banco.**

Exemplos de cenários:

- **Cadastrar um usuário com dados válidos.**
- **Tentar cadastrar dois usuários com o mesmo e-mail (deve falhar).**
- **Atualizar dados do usuário.**
- **Excluir usuário com permissão de administrador.**
- **Bloquear exclusão se o usuário não for administrador.**

Testes de Ponta a Ponta (E2E)

- **Simulam o uso real do sistema pela interface.**

Exemplos de cenários:

- **Preencher formulário de cadastro com dados válidos → verificar redirecionamento/mensagem.**
- **Deixar campos obrigatórios vazios → deve exibir erro.**
- **Tentar deletar outro usuário logado como usuário comum → deve bloquear a ação.**
- **Testar login, navegação até a tela de cadastro, preenchimento e submissão.**

Caso Extremo	Como tratar
E-mail com formato inválido	Validação com regex. Retornar erro amigável ao usuário.
E-mail já existente no banco	Verificar unicidade e retornar mensagem clara.
Nome com muitos caracteres	Definir limite (ex: 100 caracteres) e validar.
Campo endereço com caracteres especiais	Sanitizar entrada para evitar XSS.
Exclusão de usuário por não-admin	Verificar permissão no backend e negar com erro 403.
E-mail com letras maiúsculas e minúsculas misturadas	Normalizar (ex: .toLowerCase()) antes da verificação.
Tentativa de injeção de código nos campos	Escapar caracteres e/ou usar frameworks seguros.

Exemplo com Spring Boot e JUnit

```
@Test
void shouldNotAllowNonAdminToDeleteUser() {
   User commonUser = new User("User", "user@example.com", Role.USER);
   User anotherUser = new User("Other", "other@example.com", Role.USER);
   userRepository.saveAll(List.of(commonUser, anotherUser));
   // Simula login como usuário comum
   mockMvc.perform(delete("/users/" + anotherUser.getId())
         .header("Authorization", generateTokenFor(commonUser)))
         .andExpect(status().isForbidden());
}
```

Exemplo de teste usando Cypress.

```
describe('Cadastro de Usuário', () => {
 beforeEach(() => {
   // Visita a página de cadastro antes de cada teste
   cy.visit('/cadastro-usuario');
 });
 it('deve exibir mensagens de erro se nome e e-mail estiverem vazios', () => {
   // Tenta submeter o formulário sem preencher
   cy.get('form').within(() => {
     cy.get('button[type="submit"]').click();
   });
   // Verifica mensagens de validação
   cy.contains('Nome é obrigatório').should('be.visible');
   cy.contains('E-mail é obrigatório').should('be.visible');
```

```
  });
  it('deve cadastrar usuário com dados válidos', () => {
    cy.get('input[name="nome"]').type('Maria Teste');
    cy.get('input[name="email"]').type('maria@teste.com');
    cy.get('input[name="endereco"]').type('Rua Exemplo, 123');
    cy.get('input[name="telefone"]').type('11999999999');
    cy.get('form').within(() => {
      cy.get('button[type="submit"]').click();
    });
    // Verifica que o cadastro foi bem-sucedido (exemplo: redireciona ou exibe mensagem)
    cy.contains('Usuário cadastrado com sucesso').should('be.visible');
  });
});
```